bea

BEA WebLogic
Integration™

**Guide to Data
Transformation**

Contents

Guide to Data Transformation

Transforming Data Using XQuery

Examples: Manipulating and Constraining Data Using XQuery

Transforming Non-XML Data

Transforming Data Using XSLTs

Programming Transformations

Guide to Data Transformation

In WebLogic Workshop business processes, data can be transformed using either a query or a eXtensible Stylesheet Language Transformation (XSLT). This guide describes how to use the mapper functionality of WebLogic Workshop to create a data transformation graphically. From this graphical representation of a data transformation, WebLogic Workshop generates a query. The generated query is invoked during run time by the business process to transform data. The query is written in the XQuery language—a language defined by the World Wide Web Consortium (W3C) that provides a vendor independent language for the query and retrieval of XML data.

This guide also describes how to import an existing eXtensible Stylesheet Language Transformation (XSLT) into WebLogic Workshop for data transformation. An XSLT is written in the eXtensible Stylesheet Language (XSL)—an older language defined by the W3C that supports the use of stylesheets for the conversion of XML data. In WebLogic Workshop, the preferred method for data transformations is to use queries in the XQuery language. Data transformations using XSL Transformations is supported primarily for legacy applications.

This guide also describes the design-time and run-time considerations for transforming non-XML data to other types of data.

Topics Included in This Section

Chapter 2, "Transforming Data Using XQuery"

Describes how to use the mapper functionality of WebLogic Workshop to create a query (written in the XQuery language) for transforming data between XML, non-XML, Java classes, and Java primitive data sources.

Chapter 3, "Examples: Manipulating and Constraining Data Using XQuery"

This section provides information and examples on how to use the mapper functionality of WebLogic Workshop to constraint or manipulate data in a transformation.

Chapter 4, "Transforming Non-XML Data"

Describes the design-time and run-time steps required for the transformation of data between a non-XML format and an XML format in WebLogic Integration.

Chapter 5, "Transforming Data Using XSLTs"

Describes how to import a XSLT (eXtensible Stylesheet Language Transformation) into WebLogic workshop for transforming XML data (valid to one XML Schema) to XML data (valid to a different schema.)

Chapter 6, "Programming Transformations"

Describes programming considerations for transformations outside the mapper functionality of WebLogic Workshop.

Transforming Data Using XQuery

Data transformation is the mapping and conversion of data from one format to another. This section describes how to build a data transformation using the mapper functionality of WebLogic Workshop. To transform data using an existing eXtensible Stylesheet Language Transformation (XSLT), see "Transforming Data Using XSLTs" on page 5-1.

The mapper functionality of WebLogic Workshop enables the conversion of data of different types. For example, XML data can be transformed from XML data valid to one XML Schema to another XML document valid to a different XML Schema.

The source types and target type of a data transformation can be any of the following data types:

- XML Data

- Non-XML Data

- Java Primitives

- Java Classes—To learn more, see "Using Java Classes in Transformations" on page 2-42.

A data transformation can have multiple input sources but only one output target. For example you can transform data from three input data sources to one target source, as shown in the following figure.
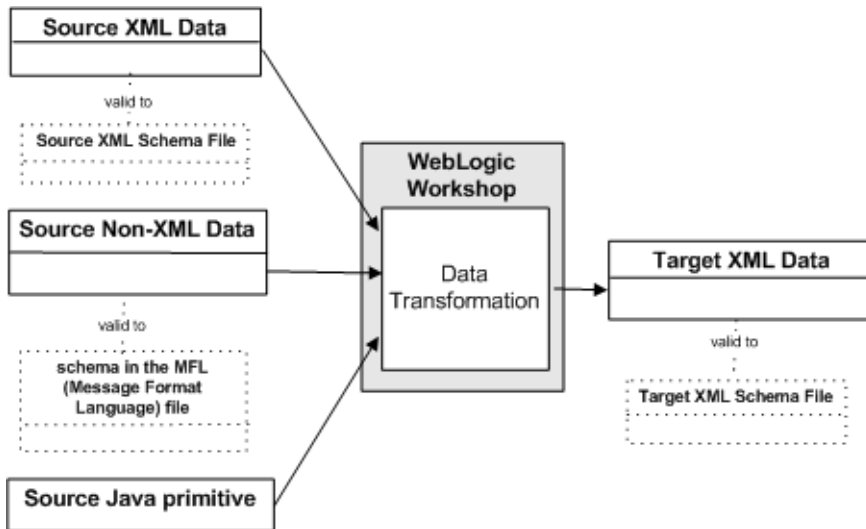
This section describes how to use the mapper functionality of WebLogic Workshop to create a data transformation graphically. From this graphical representation of a data transformation, WebLogic Workshop generates a query. The query is written in the XQuery language—a language defined by the W3C that provides a vendor independent language for the query and retrieval of XML data.

This section contains the following topics:

- Creating Schemas Projects

- Importing Schemas

- Creating a Transformation File and a Transformation Method

- Selecting Source and Target Types

- Creating Maps

- General Steps to Open or Create a Query in the Mapper

- Viewing and Editing the Generated Code of Maps

- Testing Maps in the Test View

- Link Representations

- Modifying Links Using the Target Expression Tab

- Using the Constraints Tab

- Using Java Classes in Transformations

- The Association Between XQ and DTF Files

- Using the Data Palette of the Mapper

- Validating

- Using the Property Editor of the Mapper

- XML Global Elements, Global Types, and Local Elements

Related Topics

To learn more about data transformations and for a step-by-step walk through of the mapping functionality, see *Tutorial: Building Your First Data Transformation*.

For more information and examples on how to use the mapper functionality to constraint or manipulate data see "Examples: Manipulating and Constraining Data Using XQuery" on page 3-1.

To learn about the XQuery language, see the XQuery 1.0: An XML Query Language Specification - W3C Working Draft 16 August 2002 available from the W3C web site at the following URL:

`http://www.w3.org/TR/2002/WD-xquery-20020816/`

The WebLogic XQuery engine which is invoked by the Transformation method conforms to the August 16, 2002 draft of the XQuery Specification.

To learn more about XML and XML Schemas, see Java and XML Basics.

Creating Schemas Projects

When you create a new Process application, by default a project folder named `Schemas` is created in the business process application folder of the **Application** tab of WebLogic Workshop. Additional project folders of either **Schema Project** or **WLI System Schemas** type can be created in your application folder.

In this document, the phrases *a Schemas project folder* or the *current Schemas project folder* refers to any project folder of either **Schema Project** or **WLI System Schemas** type in the current application. It can have the default name: Schemas or any other legal project name.

MFL and XSD files can be imported into project folders or subfolders of either **Schema Project** or **WLI System Schemas** type. You can have one or more project folders of either **Schema Project** or **WLI System Schemas** type in an application folder. For example, you might want to place schemas that do not change very often into the default project folder named Schemas and create another **Schema Project** folder called MySchemas which contain schemas that change more often. (If a schema file changes in the project folder all the schemas in that project folder are built again.) Partitioning your schemas in this way can reduce the schema build time. For example, if a schema keeps changing in the MySchemas project folder, the schemas in the Schemas project folder will not be built.

Project folders of either **Schema Project** or **WLI System Schemas** type are same except Project folders of the type **WLI System Schemas** also contain WebLogic Integration system XSD files. Importing MFL and XSD schema files into project folders of either type will make these schemas available in the current application. In addition, the default project folder named Schemas also contains the WebLogic Integration system XSD files. (This default project folder named Schemas is created when a new business process application is created in WebLogic Workshop.)

For example, you may want to create a Project folder of the **WLI System Schemas** type, if you deleted a WebLogic Integration system XSD file in the default project folder named Schemas, and you now want access to that system XSD file in your application.

This section contains the following tasks:

- To Create Business Process Application (Required)

- To Create Additional Schema Project or WLI System Schemas Project Folders (Optional)

To Create Business Process Application (Required)

Open or create a business process project and application. For instructions, see Creating a Business Process Application. When you create a new business process application, by default a project folder named Schemas is created. You can use the project folder named Schemas to import your MFL and XSD files into your business process application or you can create additional **Schema Project** or **WLI System Schemas** project folders and import your MFL and XSD files into those project folders as described in the following task.

To Create Additional Schema Project or WLI System Schemas Project Folders (Optional)

1. In the **Application** tab, right-click on the top-level application folder. (If the **Application** tab is not visible in WebLogic Workshop, choose **View—Application** from the menu bar.)

2. From the drop-down menu, select **New—Project...**.

   The **New Project** dialog box is displayed.

3. In the left-most pane of the **New Project** dialog box, select the **Schema** folder.

4. In the right-most pane of the **New Project** dialog box, select either **Schema Project** or **WLI System Schemas**.

5. In the **Project name** field, enter a name (for example: `MySchemas`) and click **Create**.

6. A new schema project folder is created in the **Application** tab.

Importing Schemas

In this task, you import your schemas into your application. The following schema types are supported:

- XML Schema—W3C XML Schema files describe and constrain the content of XML documents. (The XSD files that contain XML Schemas end in the `.xsd` extension.)

  Note: Multiple namespaces in XML and XSD files are supported in WebLogic Workshop. For example, you can transform two source XML files valid to different namespace to another XML file which is valid to a third namespace. To learn more, see Understanding the Transformation of the *Tutorial: Building Your First Data Transformation*.

- MFL—MFL (Message Format Language) documents describe and constrain the content of non-XML data. For example, data coming from COBOL copybooks and C structure definitions. (MFL files are created using the Format Builder and end in the `.mfl` extension. For instructions on using Format Builder, see "Using Format Builder to Create Format Schemas (MFL Files)" on page 4-7.)

  Note: The file name of the MFL document becomes the namespace of the MFL elements.

When a schema (described in an XSD or MFL file) is imported into your application, representations of these schemas are available in some of the panes of WebLogic Workshop. In addition, Java interfaces for accessing the data represented in the schemas are generated. To learn more about these Java classes see "Java Classes Created From Importing Schemas" on page 6-1.

Note: Due to a Windows operating system limitation on file path lengths (256 characters), you might have trouble compiling schemas in which the generated types are nested in a very deep package hierarchy. This may result in an error message stating that a particular class couldn't be found. To work around this limitation, try using an xsdconfig file to guide the naming of generated packages so that the hierarchy is less deep, package names are shorter, etc. For more information, see How Do I: Guide XMLBeans Type Naming During Schema Compilation?.

Importing an XML Schema (XSD file)

When an XSD file (which contains an XML Schema) is imported into a Schema project folder of a WebLogic Workshop application, a build of the application is triggered which generates XMLBeans based on the imported XML Schema. The XML Beans provides Java classes for accessing the XML data that conforms to the imported XML Schema. These Java classes are available as XML types in the type system of the application. To learn more, see Getting Started with XMLBeans.

After XML Beans are generated for an XML Schema, the following options are enabled in WebLogic Workshop:

- The ability to create an XML business process variable of the XML Schema type from the **Data Palette** of a business process. The **Data Palette** of a business process is available when a JPD file is active in WebLogic Workshop. (If the **Data Palette** is not visible with a JPD file active, choose **View→Windows→Data Palette** from the menu bar.) To learn more about business process variables, see Creating Variables.

- The ability to select the XML types as a source or target type for a transformation. To learn more, see "To Select the Source and Target Types" on page 2-12 and "To Change the Selected Source or Target Parameters" on page 2-15.

- The ability to select a displayed XML type from **Data Palette** of the mapper. (The XML Schema type must be selected as a source or target type in order for it visible in the **Data Palette** of the mapper.) The **Data Palette** of the mapper is available when an XQ file is active is WebLogic Workshop. An XQ file contains a query which is written in the XQuery language. (If the **Data Palette** is not visible, with a XQ file active, choose **View→Windows→Data Palette** from the menu bar.) To learn more about mapper variables, see "Using the Data Palette of the Mapper" on page 2-50.

  Note: The **Data Palette** of the mapper displays a different set of variables than the variables displayed in the **Data Palette** for business processes (JPDs). The variables displayed in the **Data Palette** of the mapper are used only in transformations and are distinct from the variables used in business processes.

Importing an MFL File

When an MFL file (which contains a schema describing non-XML data) is imported into a Schema project folder of a WebLogic Workshop application, a build of the application is triggered which generates XML Beans and `MflObject` class based on the schema in the imported MFL file. From the definitions in the MFL file, Non-XML types as well as XML types (from the schema in the MFL file) are available in the type system of the application.

The generated `MflObject` Java class provides methods for the conversion between non-XML and XML data, programmatically outside the mapper functionality of WebLogic Workshop. To learn more, see "Java Classes Created From Importing Schemas" on page 6-1.

The generated XML Bean classes provide methods for access the data that conforms to an MFL file. To learn more about the XML Schema representations generated from an MFL file see "Java Classes Created From Importing Schemas" on page 6-1. To learn more about XMLBeans, see Getting Started with XMLBeans.

After XML Beans are generated for the schema defined in the MFL file, the following options are enabled in WebLogic Workshop:

- The ability to create a **Non-XML** business process variable (that conforms to the schema in the MFL file) from the **Data Palette** of a business process. (If the **Data Palette** is not visible with a JPD file active, choose **View—Windows—Data Palette** from the menu bar.) For example, if a `StockQuotes.mfl` is imported into a Schemas project a `StockQuotes.mfl` node is visible from the **Create Variable** pane for the **Non-XML** variable type. To launch the **Create Variable** pane, choose **Add—Non-XML** in the top right-hand corner of the **Data Palette** pane when a JPD is active. This is an internal representation of the XML Schema not available outside of WebLogic Workshop. To learn more about business process variables, see Creating Variables.

- The ability to create an **XML** business process variable derived from the schema in the MFL file from the **Data Palette** of a business process. (If the **Data Palette** is not visible, with a JPD file active, choose **View—Windows—Data Palette** from the menu bar.) For example, if a `StockQuotes.mfl` is imported into a Schemas project a `StockQuotes.xsd.mfl` node is visible from the **Create Variable** pane for the XML variable type. To launch the **Create Variable** pane, choose **Add—XML** in the top right-hand corner of the **Data Palette** pane when a JPD is active. This is an internal representation of the XML Schema not available outside of WebLogic Workshop. To learn more about business process variables, see Creating Variables.

- The ability to select the **Non-XML** type or the **XML** type (derived from definitions in the MFL file) as a source or target type for a transformation in the **Configure XQuery**

Transformation Method pane. When you select the **Non-XML** or **XML** type as a source type for a transformation, a variable of the source type is displayed in the **Data Palette** of the mapper.

Note: The **Raw Data** type represents unstructured **Non-XML** data. This type is not available in data transformation.

- The ability to select a displayed schema type (derived from the MFL file) in the **Data Palette** of the mapper. (The XML Schema type must be selected as a source or target type in order for it visible in the **Data Palette** of the mapper.) The **Data Palette** of the mapper is available when an XQ file is active is WebLogic Workshop. An XQ file contains a query which is written in the XQuery language. (If the **Data Palette** is not visible, with a XQ file active, choose **View→Windows→Data Palette** from the menu bar.) To learn more see "Using the Data Palette of the Mapper" on page 2-50.

Note: The **Data Palette** of the mapper displays a different set of variables than the variables displayed in the **Data Palette** for business processes (JPDs). The variables displayed in the **Data Palette** of the mapper are used only in transformations and are distinct from the variables used in business processes.

To make the schemas in XSD and MFL files available in your process application, you must import them into a Schemas project folder. (To learn more, see a "Creating Schemas Projects" on page 2-3.) You can import XSD and MFL files into a Schemas project folder, by following the steps described in one of the following tasks:

- To Import an XSD or MFL file Into Your Application Using the Import Option of the Schemas Folder Drop-Down Menu
- To Import an XSD or MFL file Into Your Application Using Drag-And-Drop

To Import an XSD or MFL file Into Your Application Using the Import Option of the Schemas Folder Drop-Down Menu

In this task, you import XSD or MFL files using the **Import** option of the drop-down menu of a Schemas project folder.

1. In the **Application** tab, right-click on a Schemas project folder. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

2. From the drop-down menu, select **Import...** .

   The **Import Files** dialog box is displayed.

3. Browse the file system, and select your XSD file (ends in the `.xsd` extension) or MFL file (ends in the `.mfl` extension), and click **Import**.

The schema in the XSD or MFL file is imported. This triggers a build of the current Schemas project folder. (The build verifies that the schema file is well formed. For XSD files, it also verifies that the element and attribute names in the XML Schema do not conflict with the XSD files that have already been imported into the current Schemas project folder.)

Any errors that occur when compiling your XSD and MFL files will be displayed in the WebLogic Workshop **Build** tab. For all of the XSD and MFL files imported into a Schema project, you must fix any reported errors before you will be able to use these schemas in business processes or transformations.

To Import an XSD or MFL file Into Your Application Using Drag-And-Drop

In this task, you import XSD or MFL files by dragging-and-dropping them into a Schemas project folder.

In a Windows Explorer pane, select the XSD or MFL file and drag it into a Schemas folder. (If the **Application** tab is not visible in WebLogic Workshop, choose **View**→**Application** from the menu bar.)

The schema in the XSD or MFL file is imported. This triggers a build of the current Schemas project folder. (The build verifies that the schema file is well formed. For XSD files, it also verifies that the element and attribute names in the XML Schema do not conflict with the XSD files that have already been imported into the current Schemas project folder.)

Any errors that occur when compiling your XSD and MFL files will be displayed in the WebLogic Workshop Build pane. For all of the XSD and MFL files imported into a Schema project, you must fix any reported errors before you will be able to use these schemas in business processes or transformations.

Note: Outside of WebLogic Workshop, you can also copy XSD or MFL files directly into a Schemas project folder in the file system. For example, if your application saved in the `C:\user_projects\applications\myApp` directory contains the default Schemas project, you can save the MFL file directly into this `C:\user_projects\applications\myApp\Schemas` directory in the file system. This will trigger a build of the current Schemas project folder in WebLogic Workshop.

Creating a Transformation File and a Transformation Method

This section describes how to create a Transformation file (also know as a DTF file because it ends in the `.dtf` extension). In addition, this section describes how to add a Transformation method to the DTF file. Each Transformation method contains a single data transformation—the mapping and conversion of data from one format to another.

Data Transformations in business processes can be created in the following ways:

- While creating a **Client Request**, **Client Response**, **Control Send**, **Control Send with Return**, or **Control Receive** nodes in a specific business process—For instructions on creating a transformation from a Client or Control node, see Interacting With Clients and Interacting With Resources Using Controls, respectively.

- While building a data transformation independent of a specific business process, from the **WebLogic Workshop** menu bar—This section describes how to create a standalone data transformation from the menu bar and store it in a DTF file. Creating a transformation stored in a standalone DTF allows for the reuse of the transformation in different nodes of a business process.

This section contains the following tasks:

- To Create a Transformation File From the Menu Bar

- To Add a Transformation Method to Transformation File

To Create a Transformation File From the Menu Bar

1. Open business process project and application.

    Note: You must be active in a business process project, in order for **Transformation File** option to be available from the **New File** dialog box. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

2. In the **Application** tab, select a subfolder of a project folder. (Project folder names end with the string: Web.)

3. From the **WebLogic Workshop** menu bar, choose **File→New→Transformation File**.

    The **New File** dialog box appears.

4. In the **File name** field, enter *file*.dtf, where *file* represents the file name in which the Transformation file is stored. In this example, the file name is dataTrans is entered.

5. Click **Create**.

    In the **Design View**, a graphical representation of the **dataTrans** Transformation file appears, as shown in the following figure.
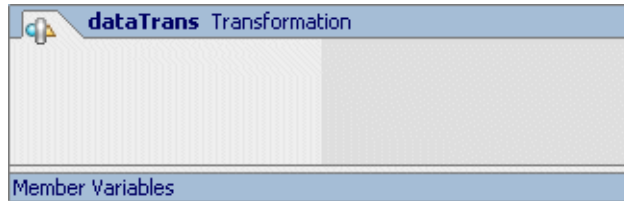
In addition, a Transformation file (ending in the .dtf extension) is created. For this example, the Transformation file called dataTrans.dtf is created and is visible in the **Application** tab.

To Add a Transformation Method to Transformation File

1. In the **Design View**, right-click in the box representing the *dataTrans* Transformation file. The box shown in the preceding figure. (Where *dataTrans* is the name of the Transformation file.)

2. From the drop-down menu, select **Add Transformation Method**.

   A transformation method is created in the Transformation file.

3. Enter *myTransMethod*, where *myTransMethod* represents the method name.

   The Transformation method in the Transformation file is created. This task does not however, create the XQ file to store the query. The XQ file is created in the following section. XQ files contain queries written in the XQuery language which end in the .xq extension.

In addition to transformation methods, user-defined Java methods can added to a Transformation file. (User methods are user-defined Java methods that can be called from XQuery code.) For instructions on adding a User method, see "To Add a User Method to a Transformation File" on page 2-39.

Note: You can also create transformation methods and user methods in the **Design View** of a DTF file, by dragging-and-dropping the **Transformation Method** and **User Method** icons, respectively from the **Palette** to the **Design View**.

Selecting Source and Target Types

In the following tasks, you select the source and target types from the **Source** or **Target** panes of the **Configure XQuery Transformation Method - *methodName*** dialog box. (Where ***methodName*** represents the current Transformation method.) Source types are the input data types for the data transformation—the data types that are transformed to the target data type.

This section contains the following tasks:

- To Select the Source and Target Types

- To Change the Selected Source or Target Parameters

- Updating the Graphical Representation Displayed in the Source View of a XQ File

To Select the Source and Target Types

1. Select or create a Transformation file.

   For instructions on creating a Transformation file see "To Create a Transformation File From the Menu Bar" on page 2-10.

   To select an existing Transformation file:

   a. In the **Application** tab, expand the folders that contain the Transformation file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

   b. In the **Application** tab, double-click the Transformation file (ends in the .dtf extension).

2. Select or create a method from a Transformation file.

   For instructions on creating a method in a Transformation file, see "To Add a Transformation Method to Transformation File" on page 2-11.

   To select an existing method, in the **Design View** of the Transformation file (ends in the .dtf extension), right-click the arrow representing the method, as shown in the following figure:
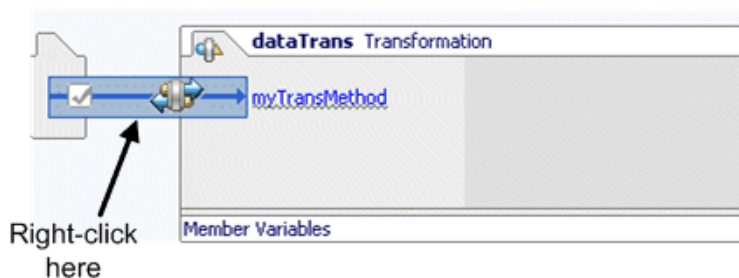
3. From the drop-down menu, select **Configure XQuery Transformation Method**.

The **Configure XQuery Transformation Method** dialog box is displayed. In the **Available Source Types** pane of **Configure XQuery Transformation Method** dialog box, the list of available source types are displayed.

Note: In order for schema representations to be available in the **Available Source Types** and **Available Target Types** pane, the XSD and MFL files which contain these schemas must be imported into a Schemas project folder and the current application must have completed building. To learn more, see "Selecting Source and Target Types" on page 2-11.

4. Select the desired source type from the **XML**, **Non-XML**, and **Java** options.

5. Specify an source type, by doing one of the following:

   To specify an XML, Non-XML, or Java primitives source type:

   a. In the **Available Source Types** pane, expand the schema and element folders, until you find the desired element.

   b. Select an source element.

   c. If desired, change the default name provided in the **Name** field. This field specifies the name the mapper uses to refer to this element.

   d. Click **Add**.

   To specify a Java class source type:

   a. The Java class for conversion must be available in the current project. To learn more about including a Java class in your project, see Using Existing Applications.

   b. In the **Type** field, of the **Available Source Types** pane, enter the full package name of the Java class. For example, for a class named `Book` in the package named `org.library`, enter: `org.library.Book` in the **Type** field.

   c. If desired, change the default name provided in the **Name** field. This field specifies the name the mapper uses to refer to this element.

   d. Click **Add**.

   To learn more, see "Using Java Classes in Transformations" on page 2-42.

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

Note: `Non-XML Types/Untyped/RawData` is not supported as a source or target type for a Transformation method. RawData has no associated structure and therefore can not

be transformed using the mapper. To learn more, see "Create a New Instance of an MflObject From Untyped Raw Data Example" on page 6-14.

6. Repeat step 5 for an additional source types.

   Note: Multiple source types can be specified.

7. Specify an target type, by doing one of the following:

   To specify an XML, Non-XML, or Java primitives target type:

   a. In the **Available Target Types** pane, expand the schema and element folders, until you find the desired element.

   b. Select an target element.

   c. Click **Add**.

   To specify a Java class target type:

   a. The Java class for conversion must be available in the current project. To learn more about including a Java class in your project, see Using Existing Applications.

   b. In the **Type** field, of the **Available Target Types** pane, enter the full package name of the Java class. For example, for a class named `Book` in the package named `org.library`, enter: `org.library.Book` in the **Type** field.

   c. Click **Add**.

      To learn more, see "Using Java Classes in Transformations" on page 2-42.

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

   Note: Only one target type can be specified.

8. If desired, select the **Schema Validate Parameters** check box.

   If this check box is selected, during run time the source parameters are validated against their schema types before the transformation is executed. To learn more, see "Schema Validating During Run Time" on page 2-53.

   Note: The **Schema Validate Parameters** check box will only be enabled if all the parameters and the return type are typed XML or typed MFL types.

9. If desired, select the **Schema Validate Return** check box.

If selected, during run time the target parameter is validated against its schema type after the transformation is executed. To learn more, see "Schema Validating During Run Time" on page 2-53.

Note: The **Schema Validate Return** check box will only be enabled if all the parameters and the return type are typed XML or typed MFL types.

10. Click **Create Transformation**.

Note: You do not have to select an source type but you must select an target type.

An XQ file (ending in the `.xq` extension) is created to store the query and displayed in the **Application** tab. The query is written in the XQuery language. XQ files are associated with a particular DTF file and appear indented under the associated DTF file in the **Application** tab. To learn more, see "The Association Between XQ and DTF Files" on page 2-47.

To Change the Selected Source or Target Parameters

If links have been created between source and target types in the **Design View** of an XQ file, then XQuery code has been generated which refers to the source and target types.

If you change the source and target types of the query, the existing XQuery code remains unchanged and may be referencing the original source or target types which may be no longer valid for this query. The XQuery code in the query may now be invalid and may require some manual XQuery code clean up as described in the last step in the following task.

Note: You may be able to minimize the amount of clean-up required by using the same name for the source variable in both the original and new query. Instead of having different names for the original and new source variables, for example: `oldOrderDoc/po-number` and `newOrderDoc/po-number`, respectively, use the same name: `orderDoc/po-number` for both.

To change or add source parameters or change the target parameter:

1. Select the Transformation file (ends in the `.dtf` extension) that contains that Transformation method:

   a. In the **Application** tab, expand the folders that contain the Transformation file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View**→**Application** from the menu bar.)

   b. In the **Application** tab, double-click the Transformation file (ends in the `.dtf` extension) that contains the Transformation method.

2. Select the method from the Transformation file. In the **Design View** of the Transformation file, right-click the arrow representing the method.

3. From the drop-down menu, select **Reconfigure XQuery Transformation Method**.

   The **Configure XQuery Transformation Method** dialog box is displayed.

4. Remove and add elements in the **Available Source Types** and **Available Target Types** pane as desired.

5. If desired, select the **Schema Validate Parameters** check box.

   If this check box is selected, during run time the source parameters are validated against their schema types before the transformation is executed. To learn more, see "Schema Validating During Run Time" on page 2-53.

   Note: The **Schema Validate Parameters** check box will only be enabled if all the parameters and the return type are typed XML or typed MFL types.

6. If desired, select the **Schema Validate Return** check box.

   If selected, during run time the target parameter is validated against its schema type after the transformation is executed. To learn more, see "Schema Validating During Run Time" on page 2-53.

   Note: The **Schema Validate Return** check box will only be enabled if all the parameters and the return type are typed XML or typed MFL types.

7. Click **Edit Transformation**.

8. Clean up the XQuery code if required. If you changed the source and target types of the query, the existing XQuery code remains unchanged—the XQuery code is not regenerated and therefore any references in the XQuery code to the original source or target types remain in the query. These references may be invalid for this query depending on what source and target types were changed as described in the following guidelines:

   - If you added an additional source type, no XQuery code clean up is required.

   - If you remove an existing source type, the XQuery code that references the removed source type will be invalid and may need to be removed.

   - If you change the existing source or target types, the XQuery code that references the changed source or target type may be invalid depending on differences between the old and new schemas and may need to be removed.

   To view the XQuery source code including any errors:

a. In the **Application** tab, expand the folders that contain the XQ file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

b. In the **Application** tab, double-click the XQ file that contains XQuery code.

c. Select the **Source View** tab.

   The XQuery code is displayed and the invalid XQuery code is underlined in red.

9. Fix the errors reported in the **Source View**. To view a detailed description of an error, move the mouse over the error in the **Source View**.

   If desired, you can delete all the XQuery code in the **Source View** of the XQ file by removing all the XQuery source code after the namespace declaration(s) and recreating your links in the **Design View**.

Updating the Graphical Representation Displayed in the Source View of a XQ File

The following procedure describes how to force the mapper to display an updated graphical representation of an XQ file in the **Design View**, if you have manually changed the source or target parameters of a Transformation method in the **Source View** of the Transformation file. To update the graphical representation of the XQ file, complete the following steps:

1. View the Transformation file that contains the query (stored in the XQ file) in the Design View:

   a. In the **Application** tab, expand the folders that contain the Transformation file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

   b. In the **Application** tab, double-click the Transformation file (ends in the `.xsd` extension).

   c. Select the **Design View** tab.

2. Select the desired Transformation method from a Transformation file:

   a. Right-click the arrow representing the method.

   b. From the drop-down menu, select **Reconfigure XQuery Transformation Method**.

   The **Configure XQuery Transformation Method** dialog box is displayed.

3. Click **Edit Transformation**.

   The graphical representation of the object is displayed in the **Design View** of the XQ file.

Creating Maps

To create maps between elements, you must have selected at least one input source schema and the output target schema. For instructions, see "To Select the Source and Target Types" on page 2-12 and "To Change the Selected Source or Target Parameters" on page 2-15.

1. Select a Transformation file.

   For instructions on creating a Transformation file see "To Create a Transformation File From the Menu Bar" on page 2-10.

   To select an existing Transformation file:

   a. In the **Application** tab, expand the folders that contain the Transformation file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

   b. In the **Application** tab, double-click the Transformation file (ends in the .dtf extension).

2. Select a Transformation method from a Transformation file.

   For instructions on creating a method in a Transformation file, see "To Add a Transformation Method to Transformation File" on page 2-11.

   To select an existing method, in the **Design View** of the Transformation file:

   a. Right-click the arrow representing the method.

   b. From the drop-down menu, select **Goto XQuery Document**.

   The mapper pane is displayed.

3. Select the **Design View** tab.

   The selected input source documents are listed in the **Source** pane and the selected output target document is listed in the **Target** pane.

   Warning: If a schema is not listed in the **Source** or **Target** panes, you will not be able to create links. For instructions to import the schema, see "To Select the Source and Target Types" on page 2-12 and "To Change the Selected Source or Target Parameters" on page 2-15.

   Note: A schema may not be listed in the **Source** or **Target** panes while an application is building. Wait until the build has completed before selecting the schema.

   The **Design View** displays the a graphical representation of the selected source schemas in the **Source** pane.

4. Drag a node from the **Source** pane to a node in the **Target** pane.

   While dragging a node from the **Source** pane over nodes in the **Target** pane, a temporary link (a dashed line) appears between the two nodes. The color of the dotted line changes depending on the compatibility between the source and target node, as shown in the following table:

The Color of the Dashed Line is . . .	Means . . .
Red	No link can be created between the source node and the target node. The data type of the target node cannot be converted to the data type of the source node. (The link represents a illegal mapping.) For example, a node of data type XML string can not be converted to an XML repeating node. A red error dialog box will be displayed when you drag the source node over the target node.
Orange	A link can be created between the source node and the target node but the data types are not completely compatible. An orange warning dialog box describing the incompatibility or any necessary conversion will be displayed when you drag the source node over the target node.
Green	A link can be created between the source node and the target node. The data type of the target node is compatible with the data type of the target node.

   Warning: Be careful when creating links between a Java Strings and a typed XML parameters. When the XQuery code, which is generated when you create a map between these two types, is run in the XQuery engine, the result is an empty typed XML target document. The XQuery engine does not parse the String into a typed XML document.

   After the target node has been dropped on the source node, a line representing a link will be displayed. Depending on the target and source nodes, a dashed line or a solid line will be displayed. To learn more, see "Link Representations" on page 2-25.

5. Repeat the preceding step until all the desired nodes are mapped.

Note: To learn more about the menu options available for links including information on how to delete links, see "Link Menu Options" on page 2-30.

Note: Instead of mapping nodes, you can create a constant for a node in **Target** pane. During run-time, the node will return the value of the constant. This functionality may be useful during the development of your application. For example, you might have the transformation return constants, so you can test the actions that occur after the

transformation, before mapping source to target nodes. To create a constant, right-click a node in the **Target** pane and from the drop-down menu, select **Create Constant**. In the **Constant Value** field, enter the value of the constant, and click **OK**.

General Steps to Open or Create a Query in the Mapper

The following procedure provides the steps for creating or opening a Business Process Application, Transformation file, and XQ file.

1. Create or open an existing business process project and application that contains the query stored as a method in the Transformation file.

   For instructions on creating a new business process project and application, see Creating a Business Process Application.

   To open an existing application:

   a. From the WebLogic Workshop menu bar, choose **File→Open→Application**.

   b. In the **Open Workshop Application** dialog box, browse for the desired application and click **Open**.

2. Import your schemas into your application. For instructions, see "Importing Schemas" on page 2-5.

3. Create or open the Transformation file.

   For instructions on creating a new Transformation file, see "Creating a Transformation File and a Transformation Method" on page 2-9.

   To open an existing Transformation file:

   a. In the **Application** tab, browse and select a Transformation file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

   b. Double-click the Transformation file (ends in the `.dtf` extension).

4. Select input source schema(s) and the output target schema. For instructions, see "To Select the Source and Target Types" on page 2-12 and "To Change the Selected Source or Target Parameters" on page 2-15

5. Create or select a Transformation method from a Transformation file.

   For instructions on creating a method in a Transformation file, see "To Add a Transformation Method to Transformation File" on page 2-11.

   To select an existing method, in the **Design View** of the Transformation file:

a. Right-click the arrow representing the method.

b. From the drop-down menu, select **Goto XQuery Document**.

The mapper pane is displayed.

Viewing and Editing the Generated Code of Maps

A query (in the XQuery language) is generated when you create mapping links from Source elements and attributes to Target elements and attributes.

1. Open a query in the mapper pane. For instructions see "General Steps to Open or Create a Query in the Mapper" on page 2-20.

2. Select the **Source View** tab.

    The generated query is displayed.

    For a description of a generated query, see Understanding the Transformation of the *Tutorial: Building Your First Data Transformation*.

3. Edit the generated query (the XQuery code). (Optional)

    The query is written in the XQuery language. To learn about the XQuery language supported with WebLogic Integration, see the XQuery 1.0: An XML Query Language Specification - W3C Working Draft 16 August 2002 available from the W3C web site at the following URL:

    `http://www.w3.org/TR/2002/WD-xquery-20020816/`

    The WebLogic XQuery engine which is invoked by the Transformation method conforms to the August 16, 2002 draft of the XQuery Specification.

    Note: You can also edit the XQuery code for a link from the **General Expression** pane of the **Target Expression** tab. To learn more see "Modifying Links Using the Target Expression Tab" on page 2-31.

Testing Maps in the Test View

The following functionality is available from the **Test View** of the mapper:

- **Test**—Clicking **Test** runs the query (transformation) to converting the data displayed in the **Source Data** pane against the mappings in the query displaying the result in the **Result Data** pane.

Note: Java interfaces and abstract Java classes are supported as source or target types for a transformation but you will not be able to execute your transformation in the **Test View**. However, a transformation with a Java interfaces and/or abstract Java classes as source or target types can be invoked from a business process (JPD) without error.

- **Select Another Source (Input) Type Using the Pull-Down Menu in the Source Data Pane**—The variables listed in drop-down menu of the **Source Data** pane are the source (input) types selected for the transformation in the **Configure XQuery Transformation Method** pane. To edit another source type in the **Source Data** pane, you can change the displayed source type by selecting another source type from the pull-down menu as shown here.
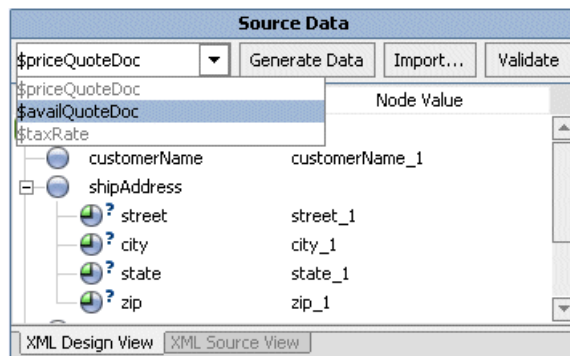
- **Import...**—Clicking **Import...** displays the **Open File To Test** pane. From the **Open File To Test** pane, you can browse for source XML and non-XML data files to load into the **Source Data** pane to use as input to the query. After importing, you can click **Test** to run the transformation using the imported source data or **Validate** to validate the displayed source data against the associated schema.

  Note: When you import source data the source data is validated against the associated schema and any warnings and errors are displayed in the **Output** pane. You can import XML data for Global Types and Local Elements but since Global Types and Local elements are not validated, no errors or warnings will be reported for invalid data. To learn more, including which types can be validated, see "Validating During Design Time" on page 2-51.

- **Generate Data**—Clicking **Generate Data** regenerates the source data used as input to the transformation. When you initially bring up the **Test View**, WebLogic Workshop generates an initial set of source sample data and displays it in the **Source Data** pane. If you make changes to the data and then decide that you want to regenerate the sample data, click **Generate Data**. For example, you might want to start over with the original sample data if

your edits result in an XML document that is no longer valid to the associated XML Schema.

● **Validate**—Clicking **Validate**, validates the displayed source data against the associated schema and any errors are reported during design time.

> **Note:** Not all types can be validated in the **Test View**. To learn more, including which types can be validated, see "Validating During Design Time" on page 2-51.

● **Editing the Data in the Source Data Pane**—When you initially bring up the **Test View**, WebLogic Workshop generates an initial set of source data and displays it in the **Source Data** pane. You can also optionally import data from source XML and non-XML data files by clicking **Import...**. You can edit and alter the data in the **Source Data** pane of the **Test View** using any of the following methods:

– You can change the generated source data by selecting the desired the **Node Value** field of the **Source Data** pane and entering your data.

– You can change the generated source data by right-clicking on a node in the **Node Name** field of the **Source Data** pane and from the drop-down menu, selecting **Edit Value**.

● **Adding, Removing, and Moving Data in the XML Design View by Right-Clicking on a Node**—You can alter the data in the **XML Design View** tab of the **Source Data** pane of the **Test View**, by right-clicking on a node (element) in the **Node Name** field of the **Source Data** pane. Only valid menu items for the selected element as defined by the XML Schema will be displayed.

In the **Node Name** field of the **Source Data** pane, right-click on a node (element) and from the drop-down menu selecting (if active) one of the following menu options:

– **Duplicate *node***—Right-clicking on an instance of a `complexType` node and selecting the **Duplicate *node*** option generates another instance of the selected node. The **Duplicate** menu option will only be available if the `maxOccurs` setting in the XML Schema element definition for the selected element is greater than 1 or unbounded.

– **Insert Child *node***—Right-clicking on a node and selecting the **Insert Child *node—child*** option, insert a child node under the selected node. This option will only be active if child nodes are allowed by the XML Schema for the selected node.

> **Note:** In the **Test View**, the option for inserting a child node to the test data for a recursive schema is disabled in the **XML Design View** tab of the **Source Data** pane. If you wish to add child nodes to the test data for a recursive schema, switch to the **XML Source View** and add the child node to the test data directly as XML.

- **Remove *node*—**Right-clicking on a node and selecting the **Remove *node*** option, deletes the node. The **Remove** menu option will only be available if the XML Schema element definition for the selected element is optional (`minOccurs=0`).

- **Move Up—**Right-clicking on an instance of a node and selecting the **Move Up** option, moves the current instance of the node up a level in the list.

- **Move Down—**Right-clicking on an instance of a node and selecting the **Move Down** option, moves the current instance of the node down a level in the list.

    Note: You can also alter the data by directly editing the XML data in the **XML Source View** tab of the **Source Data** pane.

- **Viewing and Editing XML Data—**The following two views are available for displaying XML data from the **Source Data** and **Result Data** panes:

  - **XML Design View—**Clicking the **XML Design View** tab allows you to view and edit a graphical representation of the XML data.

  - **XML Source View—**Clicking the **XML Source View** tab allows you to view and edit the data as an XML document.

- **Viewing and Editing Non-XML Data—**The following two views are available for displaying non-XML data from the **Source Data** and **Result Data** panes:

  - **Hex—**Clicking the **Hex** tab allows you to the edit the data using both hexadecimal numbers and as text (Unicode characters). The data is displayed on the left side of the pane as hexadecimal numbers and on the right side as text.

  - **Text—**Clicking the **Text** tab allows you to the edit the data as text (characters). The **Text** tab displays the printable characters (usually in the form of words and numbers) and certain control characters (carriage return, tab, and so on). For example, carriage returns are shown as line breaks. Non-printable characters, are displayed as small squares.

To View and Test an XQ file in the Test View of the Mapper

1. Open a query in the mapper pane. For instructions see "General Steps to Open or Create a Query in the Mapper" on page 2-20.

2. Select the **Test View** tab.

    A graphical display of the generated source data is displayed in the **Source Data** pane.

3. If desired, change the generated source data.

    For details see the preceding sections.

4. If desired, validate the source data against the associated schema. In the **Source Data** pane of the **Test View**, click **Validate**. To learn more, see "Validating During Design Time" on page 2-51.

5. In the **Result Data** pane, click **Test**.

   If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

   In the **Result Data** pane, after the query is run a graphical representation of the target data is displayed.

6. To view the resulting data as an XML document, in the **Result Data** pane select the **XML Source View** tab.

7. If desired, you can validate the result data against the associated schema. In the **Result Data** pane of the **Test View**, click **Validate**. To learn more, see "Validating During Design Time" on page 2-51.

Link Representations

A data link directly transforms data from a source node to a target node. For example the following figure shows a data link between the `priceQuote/customerName` element and the `quote/name` element.
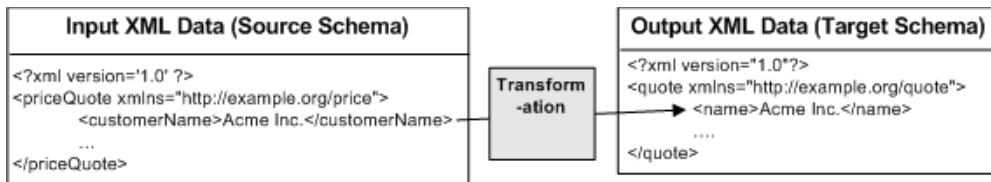
Both `priceQuote/customerName` and `quote/name` are XML String elements. During run-time, the data from the `priceQuote/customerName` element is converted to the `quote/name` element as shown in the preceding figure.

The data link between these two elements is represented by a blue line in the mapper functionality of WebLogic Workshop as shown in the following figure:
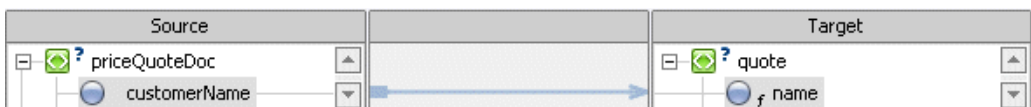
If you modify the XQuery code linking these two elements, the link between these elements changes from a data link (represented as a blue line) to an implied link (represented as a light gray line) as show in the following figure.

Note: To learn more about the menu options available for links including information on how to delete links, see "Link Menu Options" on page 2-30.

For an example of modifying the XQuery code between elements, see the task: *To Edit and Retest the Simple Query* in the Step 3: Mapping Elements and Attributes in the *Tutorial: Building Your First Data Transformation*.

The following table summaries the different link representations.

Link Type	Is the Link a Mapper Generated Link?	Description	Is the Link Currently Selected in the Mapper?	Representation of Link
Data Link	Link is generated by dragging-and-dropping.	A link that converts the value of the source node directly to the value of the target node.	Not Selected	
			Selected	
Structural Link	Link is generated by dragging-and-dropping.	A link between two parent structures that does not map data directly.	Not Selected	
			Selected	

Link Type	Is the Link a Mapper Generated Link?	Description	Is the Link Currently Selected in the Mapper?	Representation of Link
Data Structural Link	Link is generated by dragging-and-dropping.	A data structural link is the combination of the following two links: • A data link between two nodes—a link that converts the value of the source node directly to the value of the target node. • A structural link—a link between two structures. **Example:** The link between the optional child nodes of a repeating element.	Not Selected	
			Selected	
Constraint Link	Link is generated by dragging-and-dropping.	A link that constrains or limits the resulting data of a join between source parent structures. The constraint link is created with created with two source nodes. **Example:** You could add a constraint link to a join of two source repeating elements to only return the data when the values of a particular source element are equal to each other. To learn more, see "Creating a Conditional Constraint Using the Constraints Tab" on page 3-19.	Not Selected	
			Selected	

Link Type	Is the Link a Mapper Generated Link?	Description	Is the Link Currently Selected in the Mapper?	Representation of Link
Copy Link	Link is generated by dragging-and-dropping	A link between two identical schema substructures. During run time, the source data is directly copied as a block to the target data. A copy link is also generated when mapping between a untyped XML node and a typed XML complex-type node.	Not Selected	
			Selected	

Link Type	Is the Link a Mapper Generated Link?	Description	Is the Link Currently Selected in the Mapper?	Representation of Link
Implied Link	Link between the nodes is created by writing new XQuery or modifying existing XQuery code.	A link whose XQuery code can not be interpreted by the mapper to be another type of link. **Examples:** • A data link that was modified using the **General Expression** section of the **Target Expression** tab or the **Source View** of the mapper. (For example, adding the `xf:upper-case` function to a link. To learn more see "Invoking Functions or Operators in a Query" on page 2-36.) • The data links generated between a second set of child nodes when a union constraint has been applied a set of two structural links. The child nodes must be of the same subschema. To learn more see "Using the Union Option of the Constraints Tab" on page 3-9.	Not Selected Selected	

Link Menu Options

This section describes the menu options available with links in the **Design View** of the mapper, as shown in the following table.

Menu Option	Invoke by . . .	Result . . .
Delete All Links	In the **Design View** of the mapper, right-click on an empty section between the **Source** and **Target** panes and from the drop-down menu select **Delete All Links**. **Note:** Right-clicking on an empty section between the **Source** and **Target** panes causes all the nodes to be unselected.	In the **Design View**, the graphical representations of all the links are deleted and in the **Source View** the generated XQuery code linking the source nodes to the target nodes is deleted.
Induce Map	In the **Design View** of the mapper, right-click on a structural link and from the drop-down menu select **Induce Map**. **Note:** This menu option is only available when a structural link is selected in the **Design View** of the mapper.	Data links or data structural links will be created between the child nodes of the selected structural link if source and target child elements of the link are the same subschema type. The target and source child elements must have the same name, must be the data types, and must be in the same order for the **Induce Map** option to create child links.
View Code	In the **Design View** of the mapper, right-click on a link and from the drop-down menu select **View Code**.	The view changes from the **Design View** to the **Source View** and generated XQuery code between the links is displayed. The XQuery code for the selected link is highlighted in yellow.
Disable Target Node	In the **Design View** of the mapper, right-click on a link and from the drop-down menu select **Disable Target Node**.	Blocking XQuery code is added around the XQuery code of the selected link that prevents the XQuery code for the link from being executed during run time (either from the **Test View** or invoked from a JPD.)

Menu Option	Invoke by . . .	Result . . .
Enable Target Node	In the **Design View** of the mapper, right-click on a disabled link and from the drop-down menu select **Enable Target Node**.	The blocking XQuery code is removed around the selected link. During run time, the XQuery code for the link is executed.
Delete Link	In the **Design View** of the mapper, right-click on a selected link and from the drop-down menu select **Delete Link**.	In the **Design View**, the graphical representations of the selected link is deleted and in the **Source View** the generated XQuery code linking the source node to the target node is deleted.

Modifying Links Using the Target Expression Tab

You can view and modify the link between a source and target element using the **Target Expression** tab. You can use the **Target Expression** tab to build a more complex expression between a source and target element as shown in the following sections:

- Editing Links Using the Target Expression Tab
- Adding If-Then-Else Constructs to a Link
- Invoking Functions or Operators in a Query

Editing Links Using the Target Expression Tab

This section describes how to edit the XQuery code of a link using the **Target Expression** tab.

To Edit the XQuery Code of a Link

1. Open a query in the mapper pane. For instructions see "General Steps to Open or Create a Query in the Mapper" on page 2-20.

2. In the **Design View**, select or create a link between a source and target node.

   Keep the link selected for the next step.

3. Select the **Target Expression** tab. (If the **Target Expression** tab is not visible in WebLogic Workshop, choose **View→Windows→Target Expression** from the menu bar.)

   By default the **General** option is selected and the XQuery code for the link is displayed in the **General Expression** pane.

4. Select the **Target Expression** tab. (If the **Target Expression** tab is not visible in WebLogic Workshop, choose **View—Windows—Target Expression** from the menu bar.)

5. Edit the generated query (the XQuery code).

   The query is written in the XQuery language. To learn about the XQuery language supported with WebLogic Integration, see the XQuery 1.0: An XML Query Language Specification - W3C Working Draft 16 August 2002 available from the W3C web site at the following URL:

   http://www.w3.org/TR/2002/WD-xquery-20020816/

   The WebLogic XQuery engine which is invoked by the Transformation method conforms to the August 16, 2002 draft of the XQuery Specification.

Adding If-Then-Else Constructs to a Link

This section describes how to add *if-then-else* constructs to a link using the **Target Expression** tab. When a query is invoked with an i*f-then-else*, the conditions that make up the *if* expression are evaluated and depending on the result, different values are returned for a target node. For example, if the value of $quantity$ source node is greater than 500 then 4554 is returned as the value of the id target node but if the value of quantity source node is less than or equal to 500, then 5894 is returned as the value of the id target node as shown in the example in the following figure.

Figure 2-1 If-Then-Else
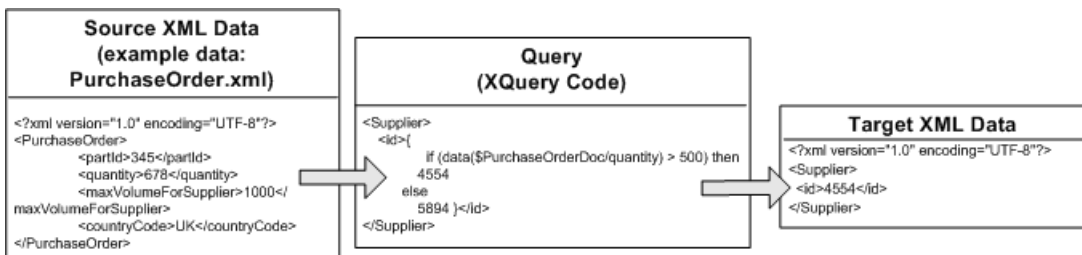
In addition to the following procedure for adding a simple *if-then-else* expression to a link, a more complex example is available. To learn more see "Adding Nested If-Then-Else Expressions with Complex Conditions to a Link" on page 3-25.

To Add an If-Then-Else Expression to a Link

1. Open a query in the mapper pane. For instructions see "General Steps to Open or Create a Query in the Mapper" on page 2-20.

   For this example, import the files: `PurchaseOrder.xsd` and `Supplier.xsd` files (step 2 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.) If you installed WebLogic Platform in the `c:\bea` directory, import these files from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\ifThenElse` directory.

   For this example, select and add the `PurchaseOrder.xsd/PurchaseOrder` node as an source type (step 4 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.)

   For this example, select and add the `Supplier.xsd/Supplier` node as an target type (step 4 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.)

2. In the **Design View**, select or create a link between a target and source element to add the *if-then-else*.

   For this example, select the `PurchaseOrderDoc/quantity` source element and drag it to the `Supplier/id` target node.

   Keep the link selected for the next step.

3. Select the **Target Expression** tab. (If the **Target Expression** tab is not visible in WebLogic Workshop, choose **View→Windows→Target Expression** from the menu bar.)

4. Select the **If Then Else** option.

   The XQuery *if-then-else* construct is added to the link. For example, the following XQuery source code segment for the link is replaced:

   ```
   data($PurchaseOrderDoc/quantity)
   ```

   By the following XQuery source code segment for the link:

   ```
   if (xf:boolean("true")) then
      data($PurchaseOrderDoc/quantity)
   else
      ()
   ```

5. Click **If Condition**.

   The **If Condition** pane appears.

6. In this step, you add condition(s) to the *if* section of the *if-then-else*.

During run time, when the condition(s) in the *if* are evaluated and if the *if* evaluates to *true*, the *then* part of the *if-then-else* is invoked but if the *if* evaluates to *false*, the *else* part of the *if-then-else* is invoked.

The *if* section of an *if-then-else* can be a simple condition that compares a source element and a simple value. The *if* section of an *if-then-else* can also be complex—involving multiple conditions.

To create a condition, first populate the left and right side expressions:

– Drag and drop elements from **Source** pane or the **Data Palette** into the **Left Hand Expression** or **Right Hand Expression** section of the **If Condition** pane.

– Enter values directly into the **Right Hand Expression** section of the **If Condition** pane.

– Drag and drop functions or operators from the **Palette** into the **Left Hand Expression** or **Right Hand Expression** section of the **If Condition** pane.

After the **Left Hand Expression** or **Right Hand Expression** sections for a simple expression are specified, select an operator from the **Operator** drop-down list and click **Add**.

For this example, complete the following steps to create a simple condition:

a. From the **Source** pane, drag the `PurchaseOrderDoc/quantity` element into the **Left Hand Expression** section of the **If Condition** pane.

b. Select the operator: **>**.

c. In the **Right Hand Expression** section of the **If Condition** pane, enter: `500` and click **Add**.

The condition is added to the *if* section of the *if-then-else*.

To create an *if* section with multiple conditions:

a. Create a simple condition as described previously.

b. Add additional conditions:

Populate the **Left Hand Expression** and the **Right Hand Expression**. Select the appropriate operator.

Select the appropriate **Join Type**.

Click **Add**.

For an example, see "Adding Nested If-Then-Else Expressions with Complex Conditions to a Link" on page 3-25.

7. Click **Then Expression**.

8. In the **Then Expression** section, enter a value or an expression. If you specify an expression, during run time if the **If Condition** evaluates to *true*, the expression in the **Then Expression** section will be evaluated and the resulting value will become the value of the target node. You can build an expression by dragging functions or operators from the **Palette** and elements from the **Source** pane or **Data Palette** into the **Edit Then Expression** section as shown in **"Adding Nested If-Then-Else Expressions with Complex Conditions to a Link" on page 3-25**.)

   For this example, replace the following text:

   ```
   data($PurchaseOrderDoc/quantity)
   ```

   With the supplier id value of `4554`.

   Note: In addition, nested *if-then-else* constructs can be added to the *else* expression. To learn more, see **"Adding Nested If-Then-Else Expressions with Complex Conditions to a Link" on page 3-25**.

9. Click **Apply**.

10. Click **Else Expression**.

11. In the **Else Expression** section, enter a value or an expression. If you specify an expression, during run-time if the **If Condition** evaluates to *false*, the expression in the **Else Expression** section will be evaluated and the resulting value will become the value of the target node. You can build an expression by dragging functions or operators from the **Palette** and elements from the **Source** pane or **Data Palette** into the **Edit Then Expression** section as shown in **"Adding Nested If-Then-Else Expressions with Complex Conditions to a Link" on page 3-25**.)

   For this example, enter `5894` and click **Apply**.

   Note: In addition, nested *if-then-else* constructs can be added to the *else* expression. To learn more, see **"Adding Nested If-Then-Else Expressions with Complex Conditions to a Link" on page 3-25**.

12. Select the **General** option, to view the generated `if-then-else` XQuery source code.

13. To run the query, select the **Test View** of the XQ file.

14. In the **Source Data** pane of the **Test View**, enter or import the desired source XML. To learn more see "Testing Maps in the Test View" on page 2-21.

For this example, in the **Source Data** pane of the **Test View**, select the `quantity` **Node Value** field and enter `678`.

15. In the **Result Data** pane, click **Test**.

If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

In the **Result Data** pane, after the query is run a graphical representation of the target data is displayed.

For this example, in the **Result Data** pane, the resulting value of `Supplier/id` element is `4554` because the source `quantity` was over `500` as shown in the preceding figure.

16. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Invoking Functions or Operators in a Query

This section describes how to insert calls to functions or operators into a query using the **Target Expression** tab.

This section contains the following topics:

- Invoking XQuery Functions or Operators in a Query
- Invoking User Defined Methods in a Query
- Invoking Control Methods in a Query

Invoking XQuery Functions or Operators in a Query

A set of standard W3C XQuery functions and operators are provided in the mapper functionality of WebLogic Integration. When you use the mapper functionality to design a transformation, a query (written in the XQuery language) is generated that does actual data conversion. In the generated query, you can add function calls to this set of standard XQuery functions. For example, as part of your transformation you might want to convert the XML String to uppercase characters.

The procedure below describes how to add a function call to a simple link between a XML String source node and an XML String target node. Adding a function to a more complicated query is described in the Step 3: Mapping Elements and Attributes in the *Tutorial: Building Your First Data Transformation*.

For listings and detailed descriptions of the XQuery functions and operators available in the mapper functionality of WebLogic Workshop, see XQuery Reference.

In addition to the XQuery functions available in the mapper functionality, a larger set functions is provided. You can manually add invocations to these functions to queries in the **Source View** of the mapper functionality. For a list of these additional functions, see the XQuery 1.0 and XPath 2.0 Functions and Operators - W3C Working Draft 16 August 2002 available from the W3C web site at the following URL:.

http://www.w3.org/TR/2002/WD-xquery-operators-20020816/

To Add a XQuery Function or Operator Call to a Query

1. Open a query in the mapper pane. For instructions see "General Steps to Open or Create a Query in the Mapper" on page 2-20.

2. In the **Design View** of the XQ file, select or create the link to add the function or operator call.

   The link between these two nodes becomes green.

   Adding a function or an operator to a link means that during run time, as part of the transformation of the data between the source node and the target node, the function will be invoked.

3. In the bottom pane of the **Design View**, choose the **Target Expression** tab. (If the **Target Expression** tab is not visible in WebLogic Workshop, choose **View—Windows—Target Expression** from the menu bar.)

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

4. Click the *Delete* key.

   The text in the **General Expression** pane is removed.

5. View function and operator folders of the **Palette**. (If the **Palette** is not visible in WebLogic Workshop, from the menu bar choose **View—Windows—Palette**.)

6. In the **Palette**, collapse and expand the folders to find the desired function or operator.

   For this example, from the **String Functions** folder select the upper-case function.

7. In the **XQuery Functions** pane, select the desired function, and drag it into the **General Expression** pane.

   For this example, the following text is displayed in the **General Expression** pane, as shown in the following figure.
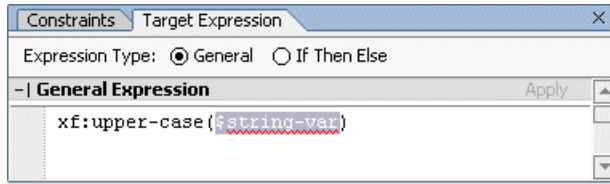
For the following step, leave the parameter selected (in this example: `$string-var`) in the **General Expression** pane as shown in the preceding figure.

8. Select a source parameter using one of the following options:

   – From the **Source** pane of the **Design View** select a source node and drag-and-drop it over the parameter in the **General Expression** pane.

   – From the **Data Palette** pane select a source variable and drag-and-drop it over the parameter in the **General Expression** pane. (If the **Data Palette** is not visible in WebLogic Workshop, from the menu bar choose **View—Windows—Palette**.)

   For this example, the following text is displayed in the **General Expression** pane, as shown in the following figure.
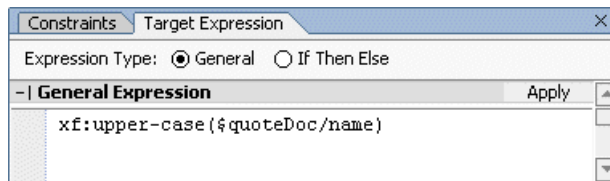
9. Repeat step 8 until all the parameters in the function or operator are assigned.

10. Click **Apply**.

   During run time for this example, the `upper-case` function will convert all the characters of the `$quoteDoc/name` element to upper case.

11. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Invoking User Defined Methods in a Query

This section describes the following tasks:

- To Add a User Method to a Transformation File
- To Add a Java User-Defined Method Call to a Query

To Add a User Method to a Transformation File

A User method is a user-defined Java method that can be called from a query (written in the XQuery language). You can add User Methods to a Transformation file and then add invocations to these User methods in queries. Adding a User method to a Transformation file is described in *Create a User Defined Java Method to Invoke From the Join Query* in Step 4: Mapping Repeating Elements—Creating a Join in the *Tutorial: Building Your First Data Transformation*.

To Add a Java User-Defined Method Call to a Query

Before you can add a Java user-defined method call to a query, the method must already have been created in the Transformation file. For instructions, see "To Add a User Method to a Transformation File" on page 2-39.

Calling a User method from a query is described in *Call the calculateTotalPrice User Method From the Query* task the in Step 4: Mapping Repeating Elements—Creating a Join in *Tutorial: Building Your First Data Transformation*.

You may want to add an exception path to the node in the business process which calls the Transformation file. To learn more, see "Getting the TransformException Fault Code Programmatically" on page 6-14.

Warning: The User method you call from the query should contain only stateless functionality.

Note: You cannot call a User method which returns a void from a query.

Invoking Control Methods in a Query

To Add a Control Function Call to a Query

1. Create or open an existing business process project and application that contains the query stored as a method in the Transformation file.

   For instructions on creating a new business process project and application, see Creating a Business Process Application.

   To open an existing application:

   a. From the WebLogic Workshop menu bar, choose **File→Open→Application**.

   b. In the **Open Workshop Application** dialog box, browse for the desired application and click **Open**.

2. Import your schemas into your application. For instructions, see "Importing Schemas" on page 2-5.

3. Create or open the Transformation file.

For instructions on creating a new Transformation file, see "Creating a Transformation File and a Transformation Method" on page 2-9.

To open an existing Transformation file:

a. In the **Application** tab, browse and select a Transformation file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.)

b. Double-click the Transformation file (ends in the .dtf extension).

4. Add an instance of a control to the Transformation file:

a. In the **Controls** section of the **Data Palette**, select **Add→Control_Name**, where **Control_Name** represents the name of the control to add.

b. Follow the steps in the **Insert Control** dialog box and click **Create**.

To learn more, see Working with Java Controls.

An instance of the control is created and a representation of the controls appears in the graphical representation of the Transformation file.

5. Select input source schema(s) and the output target schema. For instructions, see "To Select the Source and Target Types" on page 2-12 and "To Change the Selected Source or Target Parameters" on page 2-15

6. Create or select a Transformation method from a Transformation file.

For instructions on creating a method in a Transformation file, see "To Add a Transformation Method to Transformation File" on page 2-11.

To select an existing method, in the **Design View** of the Transformation file:

a. Right-click the arrow representing the method.

b. From the drop-down menu, select **Goto XQuery Document**.

The mapper pane is displayed.

7. Open the XQ file that contains the transformation in the **Design View**.

8. In the **Design View**, select the link to add the function call.

The link between these two nodes becomes green.

Adding a function to a link means that during run time, as part of the transformation of the data between the source node and the target node, the function will be invoked.

9. In the bottom pane of the **Design View**, choose the **Target Expression** tab. (If the **Target Expression** tab is not visible in WebLogic Workshop, choose **View—Windows—Target Expression** from the menu bar.)

    In the **General Expression** pane, the XQuery code linking the selected target and source node is displayed.

10. In the **Palette**, expand the **Controls Functions** folder. (If the **Palette** is not visible in WebLogic Workshop, from the menu bar choose **View—Windows—Palette**.)

11. In the **Palette**, collapse and expand the folders to find the desired function.

12. In the **Palette**, select the desired function, and drag it into the **General Expression** pane.

    Warning: When you select a Control in **Palette** all the functions in a Control are listed. You should however, only use the Control functions in queries with look-up, read-only stateless functionality. For example, a query could call a read-only function, which accepts as a parameter a record id and returns the string associated with the record id. This read-only function does not change or add any values in the database. It just reads a values from the database. Control functions that are stateful should not be called from queries. For example, a Database control function that adds a record to the database should not be called from a query. Functions that are stateful or modify the database should be called from the business process directly.

13. Select a source parameter using one of the following options:

    – From the **Source** pane of the **Design View** select a source node and drag-and-drop it over the parameter in the **General Expression** pane.

    – From the **Data Palette** pane select a source variable and drag-and-drop it over the parameter in the **General Expression** pane. (If the **Data Palette** is not visible in WebLogic Workshop, from the menu bar choose **View—Windows—Palette**.)

14. Click **Apply**.

    During run time, the query will invoke the selected function from the control.

15. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Using the Constraints Tab

The **Constraints** tab of the mapper allows you to constrain or manipulate the relationship between source and target repeating elements.

The following **Constraint Type** options are available from the **Constraints** tab:

- **Repeatability/Join**—To learn more, see "Merging the Contents of Repeating Elements" on page 3-7 and "Creating a Conditional Constraint Using the Constraints Tab" on page 3-19.

- **Union**—To learn more, see "Using the Union Option of the Constraints Tab" on page 3-9.

- **Group By Key Fields**—To learn more, see "Using the Group by Key Fields Option of the Constraints Tab" on page 3-15.

When you create structural links between repeating elements in the **Design View**, XQuery `for` loops are generated to iterate through the repeating elements. You can use the **Where Clause Expression** pane of the **Constraints** tab to limit or constrain the target repeating elements by adding `where` clauses to the XQuery `for` loops. In the **Where Clause Expression** pane of the **Constraints** tab, you can build complex conditions for the `where` clause of the XQuery `for` loop. A complex condition is made up of conditions that are joined together by OR or AND operators, for example:

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

During run time, the `for` loop will iterate only over those repeating elements that meet the complex condition. For an example see "Adding a Constraint With Multiple Conditions" on page 3-20.

Using Java Classes in Transformations

This section describes how to use Java classes as source or target types in transformations.

To Use a Java Class in Transformations

1. Create or open a business process project and application.

   For instructions on creating a new business process project and application, see Creating a Business Process Application.

   To open an existing application that contains the query:

   a. From the WebLogic Workshop menu bar, choose **File→Open→Application**.

   b. In the **Open Workshop Application** dialog box, browse for the desired application and click **Open**.

2. The Java class for conversion must be available in the current project. To learn more about including a Java class in your project, see Using Existing Applications.

Note: Java interfaces and abstract Java classes are supported as source or target types for transformations but cannot be executed in **Test View**. However, a transformation with a Java interfaces and/or abstract Java classes as source or target types can be invoked from a business process (JPD) without error.

For the example shown in this procedure, create a Java file called `Book.java` in a subfolder named `processes` in the project folder:

a. Right-click the project folder or a subfolder in the project folder.

b. From the drop-down menu, select **New→Java Class**.

   The **New File** dialog box appears.

c. In the **Field Name** field, enter `Book.java`.

d. Click **Create**.

e. Paste the following code segment in between the starting and ending curly brackets of the Book class:

```
public String title;        // Will convert to xsd type
public Author[] authors;      // Will convert to xsd type
private int copiesPrinted;  // Private member with no get/set methods;
will not convert to xsd type
private int copiesSold;     // Private member with get/set methods will
convert to xsd
public int getCopiesSold(){
    return copiesSold;
}
public void setCopiesSold(int in){
    copiesSold = in;
}
public HashMap stores;        // Will not convert to xsd type, HashMap
not supported
```

   Errors will be reported in the **Source View**. You will fix those errors in the following steps.

   To learn more about which fields are supported in Java classes, see "Java Class Conversion" on page 6-6.

   Note: If you use the Java class variable in a business process, you may enable process variable instance monitoring for the variable by implementing a `toString()` method in the custom class.

f. Add the following import definition in the third line of the `Book.java` file (after the line: `package processes;`):

```
import java.util.HashMap;
```

g. Save the `Book.java` file.

   In **Application** tab, expand the your application folder. (If the **Application** tab is not visible in WebLogic Workshop, from the menu bar choose **View→Application**.)

   Expand the ***myproject*Web** project folder, where *myproject* represents the name of your project folder.

   If required, expand the folder(s) that contain the `Book.java` file.

   Right-click the **Book.java** file and in the drop-down menu select **Save**.

You also need to create a Java file called `Author.java` in the `processes` subfolder:

a. Right-click the project folder or a subfolder in the project folder.

b. From the drop-down menu, select **New→Java Class**.

   The **New File** dialog box appears.

c. In the **Field Name** field, enter `Author.java`.

d. Click **Create**.

e. Paste the following code segment in between the starting and ending curly brackets of the Author class:

```
public String lastname;      // Will convert to xsd type
public String firstname;     // Will convert to xsd type
```

f. Save the `Author.java` file.

   In **Application** tab, expand the your application folder. (If the **Application** tab is not visible in WebLogic Workshop, from the menu bar choose **View→Application**.)

   Expand the ***myproject*Web** project folder, where *myproject* represents the name of your project folder.

   If required, expand the folder(s) that contain the `Author.java` file.

   Right-click the **Author.java** file and in the drop-down menu select **Save**.

3. Import the necessary XSD and MFL files for the other source or target parameters of the Transformation method into a Schemas project folder. To learn more, see "Selecting Source and Target Types" on page 2-11.

   For the example in this procedure, import the `Book.xsd` file. For example, if you installed WebLogic Platform in the `c:\bea` directory, import the `Book.xsd` file from the

`C:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\j avaClass` directory.

The XML Schema in the `Book.xsd` file is the target type for this example transformation.

Importing schemas files triggers a build of the current Schemas project folder. Wait until the Schemas project folder is built before proceeding to the next step. (The representations of the schemas will not be available in **Available Source Types** and **Available Target Type** panes until build is complete.)

4. Create a Transformation file and Transformation method.

   For instructions on creating a new Transformation file, see "Creating a Transformation File and a Transformation Method" on page 2-9.

5. Open the Transformation method which contains the query.

   For instructions on creating a new method in a Transformation file, see "To Add a Transformation Method to Transformation File" on page 2-11.

   To open an existing method which contains the query:

   a. Right-click the arrow representing the method that contains the query.

   b. From the drop-down menu, select **Configure Transformation Method**.

      The **Configure XQuery Transformation Method** dialog box is displayed.

6. Select the source and target parameters for the Transformation method. For detailed instructions, see "Selecting Source and Target Types" on page 2-11.

   For this example, in the **Available Source Types** pane, select the **XML** option, select `Typed/Book.xsd/Book` element as the target parameter, and click **Add**.

   For this example, in the **Available Target Types** pane, select the **Java** option, enter: `processes.Book` in the **Type** field, and click **Add**.

   Click **Create Transformation**.

   A graphical representation of the Java class and XML Schema is displayed in the **Design View**.

   Note: Not all the fields in the `Book.java` class are displayed. Only supported public members or private members with JavaBean style `get` and `set` methods are displayed. In this example, the private member: `copiesSold` is displayed because the associated JavaBeans `set` and `get` methods for this member are provided. However, the class member `stores` is not displayed because it is of type: `java.util.HashMap`

which is not a supported type. To learn more about which fields of a Java class are supported in transformations, see "Java Class Conversion" on page 6-6.

7. In the **Source** pane select a node and drag it into the **Target** pane.

   A link represented by a line between the two nodes is displayed.

   Repeat this step as necessary to create additional links.

   For this example, in the **Source** pane select the **$BookDoc/Title** node and drag it to the **Book/title** node in the **Target** pane.

   For this example, in the **Source** pane select the **$BookDoc/Author** node and drag it to the **Book/authors** node in the **Target** pane. These nodes are both repeating nodes. A repeating node means more than one instances of this node can be specified. In the **Source** pane, repeating nodes are represented with a + symbol to the right of the node. A dashed line linking the two repeating nodes is displayed.

   For this example, in the **Source** pane select the **$BookDoc/Author/LastName** node and drag it to the **Book/authors/Author/lastname** node in the **Target** pane. A solid line linking the two nodes is displayed.

   In the **Source** pane select the **$BookDoc/Author/FirstName** node and drag it to the **Book/authors/firstname** node in the **Target** pane. A solid line linking the two nodes is displayed.

   In the **Source** pane select the **$BookDoc/CopiesSold** node and drag it to the **Book/copiesSold** node in the **Target** pane. A solid line linking the two nodes is displayed.

8. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

9. Test the query:

   a. Select the **Test View** tab.

   b. In the **Result Data** pane, click **Test**.

   c. The query is run with the default test data. A graphical representation of the resulting XML data is shown in the **Result Data** pane.

   d. To view resulting data as XML, in the **Result Data** pane, select the **XML Source View** tab.

The Association Between XQ and DTF Files

Associated Transformation (DTF) and XQ files have references to each other in their source code. For example, if you create a Transformation file named `union` which contains a Transformation method called `convert` and you create maps between the source and target nodes of the Transformation method the following files are generated:

- A DTF file called `union.dtf`

- A XQ file called `convert.xq`

In the **Application** tab, the following is displayed:
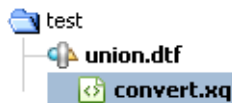
These two files are associated with each other, the `union.dtf` refers to the `convert.xq` file and the `convert.xq` refers to the `union.dtf` file. If you change the name of either of these files or the transformation method name you must update the reference to it in the other file.

For the preceding example, the following transform annotation is displayed in the **Source View** of the `union.dtf` file, as shown in the following figure.

The following comment is displayed in the **Source View** of the `convert.xq` file, as shown in the following figure:

```
          Package  Name of    Transform-
                   associated ation
          name     DTF file   method
                              name
          {-- test/union.dtf#convert --}
```

This section contains the following topics:

- Rename the DTF File and References in Associated XQ Files
- Rename the XQ File and References In the Associated DTF File

Rename the DTF File and References in Associated XQ Files

1. Save the DTF and the associated XQ file(s). From the menu bar, choose **File–Save All**.

2. In the **Application** tab, right-click the DTF file and from the drop-down menu, select **Rename**.

   For this example, select the **union.dtf** file.

3. Enter the new name and enter the return key.

   For this example, replace union with myunion.

   In the **Application** tab, the DTF file is renamed to **myunion.dtf** and the associated XQ file (convert.xq) no longer appears under the **myunion.dtf** file as shown in the following figure:

4. For each of the XQ files associated with a DTF file:

   a. In the **Application** tab, double-click a XQ file associated with the renamed DTF file.

      For this example, in the **Application** tab double-click **convert.xq**.

   b. Select the **Source View** tab of the XQ file.

      For this example, select the **Source View** tab of the convert.xq file.

   c. In the first line of the XQ file, change the listed DTF file to the new name.

      The red underline in the first line of the XQ file disappears.

For this example, change the first line from the following code:

```
{-- test/union.dtf#convert --}
```

To this code:

```
{-- test/myunion.dtf#convert --}
```

5. Save the DTF and the associated XQ file(s). From the menu bar, choose **File—Save All**.

Rename the XQ File and References In the Associated DTF File

1. Save the DTF and the associated XQ file(s). From the menu bar, choose **File—Save All**.

2. In the **Application** tab, right-click the XQ file and from the drop-down menu, select **Rename**.

    For this example, select the **convert.xq** file.

3. Enter the new name and enter the return key.

    For this example, replace `convert` with `myconvert`.

4. In the **Application** tab, double-click the DTF file associated with the renamed XQ file.

    For this example, in the **Application** tab double-click **union.dtf**.

5. Select the **Source View** tab of the DTF file.

    For this example, select the **Source View** tab of the `union.dtf` file.

6. In the DTF file, change the listed XQ file in the transform annotation to the new name.

    The red underline under the XQ name disappears.

    For this example, change the annotation from the following code:

```
/**
 * @dtf:transform xquery-ref="convert.xq"
 * @dtf:schema-validate return-value="false" parameters="false"
 */
```

    To the following annotation:

```
/**
 * @dtf:transform xquery-ref="myconvert.xq"
 * @dtf:schema-validate return-value="false" parameters="false"
 */
```

7. Save the DTF and the associated XQ file(s). From the menu bar, choose **File—Save All**.

Using the Data Palette of the Mapper

The **Data Palette** of the mapper provides a location to access the variables and their subelements. You can drag-and-drop variables or their subelements from the **Data Palette** of the mapper into the **Constraints** and **Target Expression** tabs of the mapper.

Note: You can also drag-and-drop elements from the **Source** pane of the mapper directly into the **Constraints** and **Target Expression** tabs of the mapper.

The following types of variables are displayed in the **Data Palette** of the mapper:

- **Source**—The variables listed in **Source** section of the **Data Palette** are the source (input) types selected for the transformation in the **Configure XQuery Transformation Method** pane. To learn more, see "Selecting Source and Target Types" on page 2-11.

- **Structural Link**—When a structural link is selected in the **Design View** of an XQ file, the **Structural Link** section of the **Data Palette** lists the loop iteration variable associated with the XQuery `for` loop generated by the structural link. This variable is in scope for all subelements of the node with the structural link.

  When you create link(s) between a source repeating element and a target repeating element, a structural link between the two elements is created and a XQuery *for* loop is generated. When you use the **Constraints** tab to limit or constrain repeating element(s), a `where` clause is added to the XQuery `for` loop. To learn more about which features of the mapper functionality to use to manipulate and constrain data, see Examples: Manipulating and Constraining Data Using XQuery.

Caution: The **Data Palette** of the mapper displays a different set of variables than the variables displayed in the **Data Palette** for business processes (JPDs). The variables displayed in the **Data Palette** of the mapper are used only in transformations and are distinct from the variables used in business processes.

To View the Variables Associated with Transformation in the Data Palette

1. For the **Data Palette** of the mapper to be visible, the **Design View** of an XQ file must be selected.

   To view an XQ file in the **Design View**:

   a. If the **Application** tab is not visible in WebLogic Workshop, from the menu bar choose **View**→**Application**.

   b. In the **Application** tab, double-click XQ file and select the **Design View** tab.

2. (Optional) Select a structural link to view the repeating element(s) associated with the current link that can be constrained using **Constraints** tab.

3. If the **Data Palette** is not visible in WebLogic Workshop, from the menu bar choose **View→Windows→Application**.

Related Topics

Modifying Links Using the Target Expression Tab

Examples: Manipulating and Constraining Data Using XQuery

Adding If-Then-Else Constructs to a Link

Business Process Variables and Data Types

Validating

The schema validating done during run time is different than the validate done when you click **Validate** in the **Test View** tab of the XQ file during design time. The Schema validating done on XML and non-XML typed data during run time can actually modify the resulting data while the validating during design time does *not* modify the resulting data but it does report if any required elements or attributes defined in the schema are not present.

This section provides the following topics:

- Validating During Design Time
- Schema Validating During Run Time

Validating During Design Time

During design time, **Validate** in the **Source Data** and **Result Data** panes in the **Test View** tab of an XQ file will be active if the selected source parameter or resulting data is a typed global XML element. To learn more see "XML Global Elements, Global Types, and Local Elements" on page 2-55.

For example, if a typed global XML parameter is selected in the **Source Data** pane, **Validate** will be active as shown in the following figure.
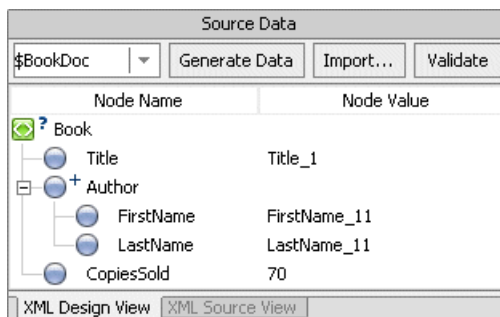
Validate will not be active if the selected source parameter or resulting data is one of the following types:

- Typed Non-XML

    Note: Untyped Non-XML (RawData) data cannot be used in transformations.

- `XmlObject` (untyped XML object)

- `XmlObjectList` (untyped XML object list)

- Java Types (both Java primitive types and Java classes)

- XML global type—To learn more see "XML Global Elements, Global Types, and Local Elements" on page 2-55.

- XML local element—To learn more see "XML Global Elements, Global Types, and Local Elements" on page 2-55.

For these types, **Validate** will not be active as shown in the following figure.
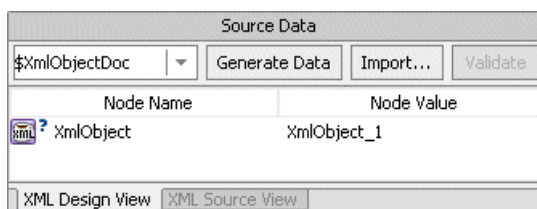
If you click **Validate** in either the **Source Data** and **Result Data** panes in the **Test View** tab of an XQ file, the displayed XML is checked against it schema and any errors are reported during design time. The validating done during design time in the **Test View** is not the same as the schema validating that occurs during run time. The validating during design time does *not* modify

the resulting XML document but it does check if any required elements or attributes defined in the schema are not present.

Schema Validating During Run Time

In the **Selected Source Types** pane of the **Configure XQuery Transformation Method** pane, if the **Schema Validate Parameters** check box is selected, during run time, the source parameters that support schema validation will be schema validated against their associated schema before the transformation is executed. The XML parameters will be schema validated against their XML Schema while the typed non-XML parameters will be validated against the schema in the MFL file. The **Schema Validate Parameters** check box will be ignored for the parameters that do not support schema validation.

Note: In the **Selected Source Types** pane of the **Configure XQuery Transformation Method** pane, the **Schema Validate Parameters** check box is not active if any of selected source parameters cannot be schema validated.

The following table shows which types will and will not be schema validated during run time.

Type	Schema Validated?	To Learn More See . . .
XML Global Element	Yes	"XML Global Elements, Global Types, and Local Elements" on page 2-55
Typed Non-XML	Yes	
XML Global Type	No	"XML Global Elements, Global Types, and Local Elements" on page 2-55
XML Local Element	No	"XML Global Elements, Global Types, and Local Elements" on page 2-55
`XmlObject` (untyped XML object)	No	

Type	Schema Validated?	To Learn More See . . .
`XmlObjectList` (untyped XML object list)	No	
Java Types (both Java primitive types and Java classes)	No	

In the **Selected Source Types** pane of the **Configure XQuery Transformation Method** pane, if you select the **Schema Validate Parameters** check box, the `parameter` attribute of the `@dtf:schema-validate` annotation will be set to `true`. To learn more see @dtf:schema-validate Annotation.

In the **Selected Source Types** pane of the **Configure XQuery Transformation** pane, if you select the **Schema Validate Return** check box and the target parameter is a type that can be schema validated, during run time the target parameter is schema validated against its schema type after the transformation is executed. To learn more see @dtf:schema-validate Annotation.

The Schema validating done on typed XML or non-XML data during run time can actually modify the resulting data. For XML data, if default attributes and elements are specified in the XML Schema and these attributes and elements do not have values in the source document, the resulting XML will have these defaults specified. To learn more about XML schema validating, see Occurrence Constraints.

If schema validating fails during run time, the `com.bea.transform.TransformException` exception is thrown. How the exception is handled depends on the node that invokes the transformation. If there is an exception path associated with node at the node level, group level or globally for the business process, the exception path is invoked. If there is no exception path associated with the node, the exception will force the business process to fail. To learn about exception paths in business processes, see Handling Exceptions.

Using the Property Editor of the Mapper

This section provides the following topics:

- Using the Property Editor When Editing an DTF File
- Using the Property Editor When Editing an XQ file

Using the Property Editor When Editing an DTF File

While editing an DTF file in the **Design View** or the **Source View**, you can use the **Property Editor** to view or edit the properties of the selected method.

For example, if you select a XQuery transformation method in the **Design View** of a DTF file, you can change the name of the method by editing the **name** property in the **Property Editor**. You can also edit the annotations and attributes of the selected method in the **Property Editor**.

Note: Context Sensitive Help is available for the dtf annotations by selecting the annotation property or attribute in the **Property Editor** and clicking F1.

To learn more see Data Transformation Annotations.

Using the Property Editor When Editing an XQ file

While editing an XQ file in the **Design View**, the **Property Editor** allows you to view schema properties about the nodes in the current map without opening the source and target XSD or MFL files. The **Property Editor** of the mapper displays a read-only view of the schema properties for the selected element(s). To change XML Schema properties you must edit the XSD file that contains the XML Schema. To change the schema properties for non-XML data, you must edit the corresponding MFL file.

Note: Selecting a link will select the target and source elements of the link. To deselect a link and the target and source nodes of the link, click an empty section of the middle pane (a section with no links) between the **Source** and **Target** panes of the mapper.

A selected target or source node is displayed with a gray or blue box around the node as shown in the following figure.
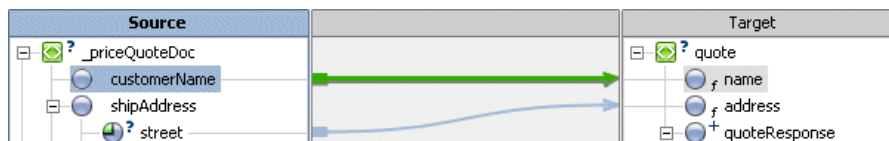
In the preceding figure, both `customerName` and `name` are selected elements, so the schema properties for both these elements will be displayed in the **Property Editor**.

XML Global Elements, Global Types, and Local Elements

An XML Schema type or element is considered *global* if it is a direct child of the `schema` element while an XML Schema type or element is considered *local* if it is not a direct child of the `schema` element (is nested to another element) as shown in the following XML Schema:

```
<?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: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 example XML Schema, the `globalElement` is *global* because it is a direct child of the `schema` element while `localElement` is *local* because it is child of `globalElement`.

You can also define a global type as shown by the `globalType` element at the bottom of the preceding XML Schema. While you can only have 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 these different XML components in the mapper.

Name	Representation in the Mapper	Name in Preceding Example XML Schema
Global Element	⊞ ◈ globalElement	`localElement` and `anotherLocalElement`

Name	Representation in the Mapper	Name in Preceding Example XML Schema
Local Element	localElement	`globalElement`
Global Type	Global Types globalType	`globalType`

Related Topics

[Global Versus Local](#)

Examples: Manipulating and Constraining Data Using XQuery

This section provides information and examples on how to use the mapper functionality of WebLogic Workshop to constraint or manipulate data in a transformation, as summarized in the following table.

If You Want to Manipulate Your Data to . . .	As Shown in . . .	To Learn More, See the Following Example . . .
Combine the contents of two different schemas. Subelements of the repeating elements are not merged.	Figure 3-1	Combining Data From Different Schemas
Merge the contents of repeating elements. Subelements of repeating elements are merged.	Figure 3-2	Merging the Contents of Repeating Elements
Combine sets of data of the same type (same schema) into larger sets of data. Subelements of the repeating elements are not merged.	Figure 3-3	Using the Union Option of the Constraints Tab
Combine data and repeating elements based on a passed in key value.	Figure 3-4	Using the Group by Key Fields Option of the Constraints Tab
Add a condition constraint which limits the repeating elements that are returned.	Figure 3-5	Creating a Conditional Constraint Using the Constraints Tab
Add a complex condition constraint which limits the repeating elements that are returned. A complex conditional constraint contains two or more conditions joined together one of the following join type: *and*, *or*.	Figure 3-6	Adding a Constraint With Multiple Conditions

If You Want to Manipulate Your Data to . . .	As Shown in . . .	To Learn More, See the Following Example . . .
Return a different value for target node based on the outcome a simple *if-then-else* containing a simple condition.	Figure 2-1	Adding If-Then-Else Constructs to a Link
Return a different value for target node based on the outcome of nested *if-then-else*(s) with complex conditions.	Figure 3-7	Adding Nested If-Then-Else Expressions with Complex Conditions to a Link
Convert data from a XML Schema simple type to a complex type.	Figure 3-8	Creating a Transformation Between a Non-Repeating Source Element and Repeating Target Group
Convert data from a XML Schema complex type to a simple type.	Figure 3-9	Creating a Transformation Between a Repeating Source Group and Non-Repeating Target Element
Create mappings with schemas that have recursive elements.	Figure 3-10	Using Recursive Schemas in Transformations

Related Topics

To learn more about how to use the mapper functionality of WebLogic Workshop to create a data transformation graphically, see "Transforming Data Using XQuery" on page 2-1.

For a step-by-step walk through of the mapping functionality, see *Tutorial: Building Your First Data Transformation*.

To learn about the XQuery language, see the XQuery 1.0: An XML Query Language Specification - W3C Working Draft 16 August 2002 available from the W3C web site at the following URL:

```
http://www.w3.org/TR/2002/WD-xquery-20020816/
```

The WebLogic XQuery engine which is invoked by the Transformation method conforms to the August 16, 2002 draft of the XQuery Specification.

To learn more about XML and XML Schemas, see Java and XML Basics.

Combining Data From Different Schemas

You can use the mapper functionality to combine the contents of two different schemas, as shown in the following figure:

Figure 3-1 Combining Data From Different Schemas
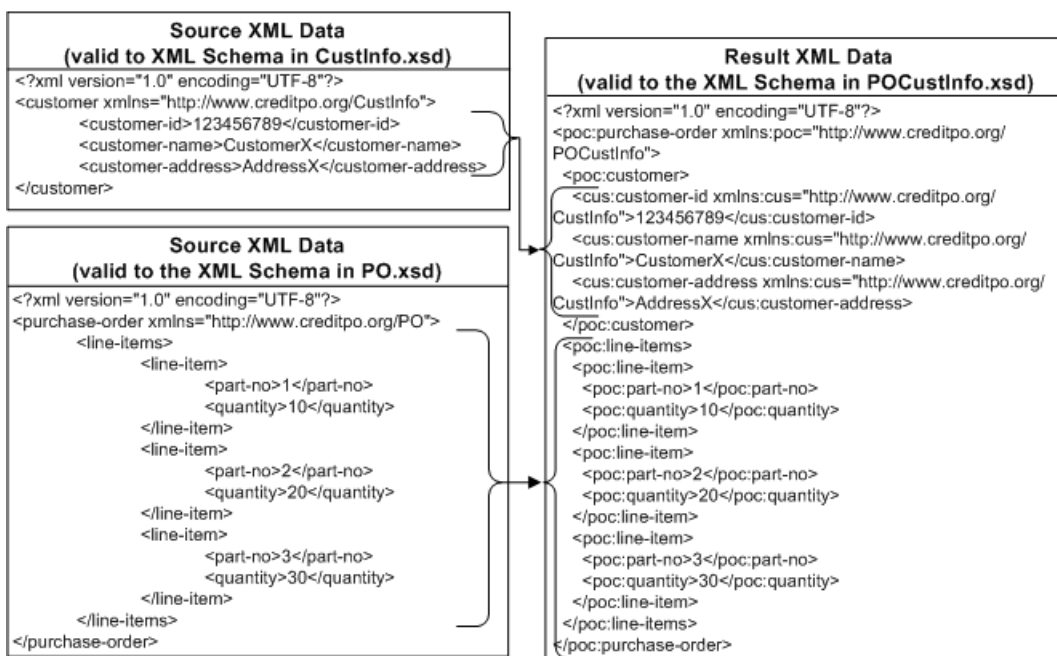
In this case, the customer information is merged with the `line-items` repeating element to form one combined XML document.

This example describes how to create a transformation which combines the data from two different XML Schemas. This example shows how to combine the XML sample data shown in the preceding figure. In this example, the source data is not constrained or limited to produce the target data—a `where` clause was not added to the *for* loop iterating over the `purchase-order/line-items/line-item`s using the **Constraints** tab.

To Combine Data From Different Schemas

1. Import the two XSD files that contain the XML Schemas for the source types of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

For this example, import the files: `CustInfo.xsd` and `PO.xsd` files. If you installed WebLogic Platform in the `c:\bea` directory, import these files from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\dataDiffSchemas` directory.

> **Note:** The `PO.xsd` file used in this example is identical to the `PO.xsd` file used in "Using the Union Option of the Constraints Tab" on page 3-9. If you have already imported the `PO.xsd` file for the Using the Union Option of the Constraints Tab example, you do not need to import it again.

2. Import the XSD file that contains the XML Schema for the target type of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

   For this example, import the file: `POCustInfo.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\dataDiffSchemas` directory.

   Importing schemas files triggers a build of the current Schemas project folder. Wait until the current Schemas project folder is built before proceeding to the next step. (The representations of the schemas will not be available in **Available Source Types** and **Available Target Type** panes until build is complete.)

3. Create a Transformation file and a method in the Transformation file. For instructions, see "Creating a Transformation File and a Transformation Method" on page 2-10.

4. Select the Transformation method from a Transformation file.

   To select an existing method, in the **Design View** of the Transformation file:

   a. Right-click the arrow representing the method.

   b. From the drop-down menu, select **Configure XQuery Transformation Method**.

5. Select the source types for the transformation:

   a. In the **Available Source Types** pane, expand the schema and element folders, until you find the desired element.

   b. In the **Available Source Types** pane, select the desired element.

   c. Click **Add**.

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

For this example, select and add the `CustInfo.xsd/customer` and `PO.xsd/purchase-order` nodes.

6. Select the target type for the transformation:

   d. In the **Available Target Types** pane, expand the schema and element folders, until you find the desired element.

   For this example, expand the `POCustInfo.xsd` schema folder.

   e. In the **Available Source Types** pane, select the desired element.

   For this example, select the `POCustInfo.xsd/purchase-order` element.

   f. Click **Add**.

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

7. Click **Create Transformation**.

   The **Design View** of the XQ file is displayed.

8. Create links between repeating element nodes:

   a. In the **Source** pane, select the repeating element and drag it to the repeating element in the **Target** pane.

   For this example, link the `$purchase_orderDoc/line-items/line-item` repeating element to the `purchase-order/line-items/line-item` repeating element.

   A dashed line linking the two repeating elements is displayed. The dashed line with short dashes represents a structural link—a link between two parent structures that does not map data directly. The dashed-line representation for a structural link is shown in the following figure:

   To learn more about links, see "Link Representations" on page 2-25.

   b. In the **Source** pane, select each of the subelements of the repeating element and drag them to the analogous subelement of the repeating element in the **Target** pane.

   For this example, link the `$purchase_orderDoc/line-items/line-item/part-no` element to the `purchase-order/line-items/line-item/part-no` element. In addition, link the `$purchase_orderDoc/line-items/line-item/quantity` element to the `purchase-order/line-items/line-item/quantity` element.

Two solid lines linking the two subelements is displayed. These solid lines represents a data link—a link that converts the value of the source node directly to the value of the target node The solid line representation for a data link is shown in the following figure.

To learn more about links, see "Link Representations" on page 2-25.

9. Create links between the second set of nodes:

   a. In the **Source** pane, select the `$customerDoc` node and drag it to the `$purchase-order/customer` node in the **Target** pane. A structural link between the two nodes is created.

   b. Right-click the structural link between the `$customerDoc` node and the `$purchase-order/customer` node and from the pull-down menu select **Induce Map**.

   A data link between the `$customerDoc/customer-id` node and the `$purchase-order/customer/customer-id` node is created.

   A data link between the `$customerDoc/customer-name` node and the `$purchase-order/customer/customer-name` node is created.

   A data link between the `$customerDoc/customer-address` node and the `$purchase-order/customer/customer-address` node is created.

   The map between the source and target elements is shown in the following figure.
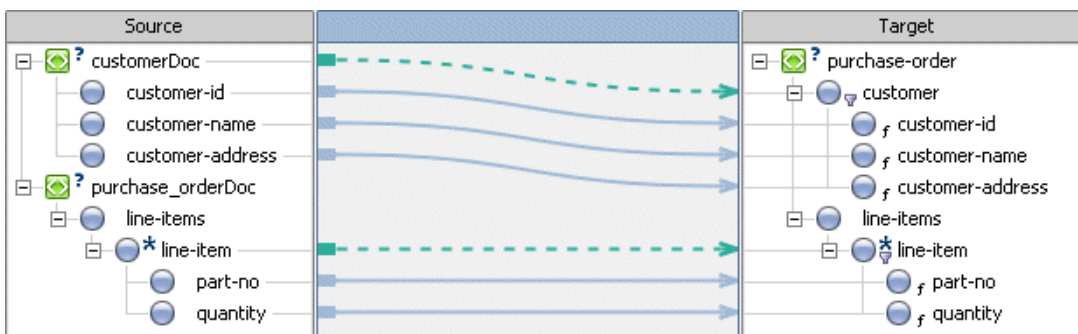
10. Select the **Test View** tab.

11. Import XML or non-XML files as source data for the transformation. For more information, see "Creating Maps" on page 2-18.

For this example, in the **Source Data** pane, select the `$purchase_orderDoc` node and import the file: InputPO.xml. In the **Source Data** pane, select the `$customerDoc` node and import the file: InputCust.xml. If you installed WebLogic Platform in the `c:\bea` directory, import these files from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\dataDiffSchemas\XML` directory.

Note: You can cut and past directory paths into the **Name** field of the **Open File to Test** pane to jump to directory locations. If you installed WebLogic Platform in the `c:\bea` directory, you can jump to the directory that contains the XML files for this example, by pasting the following directory path into the **Name** field: `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\dataDiffSchemas\XML` and then pressing enter.

12. In the **Result Data** pane, click **Test**.

    If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

    In the **Result Data** pane, after the query is run a graphical representation of the target data is displayed.

13. If the target data is XML data, in the **Result Data** pane, you can view the resulting data as an XML document by selecting the **XML Source View** tab. The XML result for this example is also displayed in Figure 3-1.

14. If desired, you can validate the result data against the associated schema. In the **Result Data** pane of the **Test View**, click **Validate**. To learn more, see "Validating During Design Time" on page 2-51.

15. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Merging the Contents of Repeating Elements

You can use mapper functionality to merge the contents of repeating elements, as shown in the following figure:

Figure 3-2 Merging the Contents of Repeating Elements


```
Source XML Data
(example data: PriceQuote.xml)

<?xml version='1.0' ?>
<priceQuote xmlns="http://example.org/price">
    ....
        <priceRequests>
            <priceRequest>
                <widgetId>12</widgetId>
                <price>1.00</price>
            </priceRequest>
            <priceRequest>
                <widgetId>134</widgetId>
                <price>34.10</price>
            </priceRequest>
        </priceRequests>
</priceQuote>
```

```
Source XML Data
(example data: AvailQuote.xml)

<?xml version="1.0"?>
<availQuote xmlns="http://www.example.org/avail">
    <availRequest>
        <widgetId>12</widgetId>
        <requestedQuanity>10</requestedQuanity>
        <quanityAvail>true</quanityAvail>
        <shipDate>2003-03-22</shipDate>
    </availRequest>
    <availRequest>
        <widgetId>134</widgetId>
        <requestedQuanity>345</requestedQuanity>
        <quanityAvail>false</quanityAvail>
        <shipDate>BackOrder</shipDate>
    </availRequest>
</availQuote>
```

```
Result XML Data
(example data: Quote.xml)

<?xml version="1.0"?>
<quote xmlns="http://example.org/quote">
    ....
        <quoteResponse>
            <widgetId>12</widgetId>
            <requestedQuanity>10</requestedQuanity>
            <fillOrder>true</fillOrder>
            <shipDate>2002-03-22</shipDate>
            <unitPrice>1.00</unitPrice>
            ...
        </quoteResponse>
        <quoteResponse>
            <widgetId>12</widgetId>
            <requestedQuanity>10</requestedQuanity>
            <fillOrder>true</fillOrder>
            <shipDate>2002-03-22</shipDate>
            <unitPrice>34.10</unitPrice>
            ...
        </quoteResponse>
        <quoteResponse>
            <widgetId>134</widgetId>
            <requestedQuanity>345</requestedQuanity>
            <fillOrder>false</fillOrder>
            <shipDate>BackOrder</shipDate>
            <unitPrice>1.00</unitPrice>
            ...
        </quoteResponse>
        <quoteResponse>
            <widgetId>134</widgetId>
            <requestedQuanity>345</requestedQuanity>
            <fillOrder>false</fillOrder>
            <shipDate>BackOrder</shipDate>
            <unitPrice>34.10</unitPrice>
            ...
        </quoteResponse>
</quote>
```
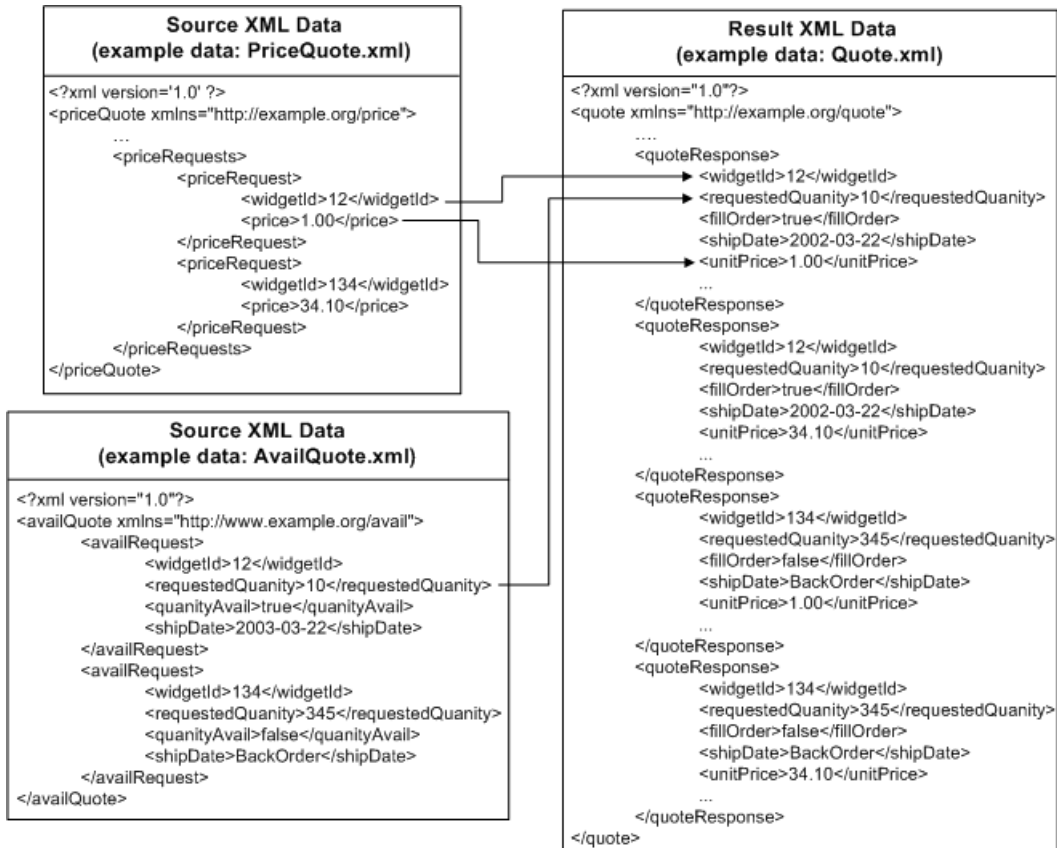
The join, shown in the preceding figure, merges the price and availability from the two source documents to one output (target) document called `Quote.xml`. Specifically, the price (element: `price`) and widget Id (element: `widgetId`) for the widgets is supplied by the `PriceQuote.xml` document and the number of widgets available (element: `requestedQuanity`) is supplied by the `AvailQuote.xml` document. The `widgetId` and `requestedQuanity` elements are part of the `availRequest` repeating element and `price` element is part of the `priceRequest` repeating element. These subelements to repeating elements are merged into subelements of the `quoteResponse` repeating element.

For this example, a complete merge of the two sets of elements resulting in four elements as shown in Figure 3-2 is not desired. Instead a conditional constraint is needed that will return the

merged element only if the condition is true. To learn more, see "Creating a Conditional Constraint Using the Constraints Tab" on page 3-19.

For a step-by-step walk through of using the mapping functionality to create a join with a conditional constraint, see *Tutorial: Building Your First Data Transformation*. Specifically, the join is created in Step 4: Mapping Repeating Elements—Creating a Join in the *Tutorial: Building Your First Data Transformation*.

Using the Union Option of the Constraints Tab

You can use the **Union** option of the **Constraints** tab to combine sets of data of the same type into larger sets of data, as shown in the following figure.

Figure 3-3 Combining Sets of the Same Data
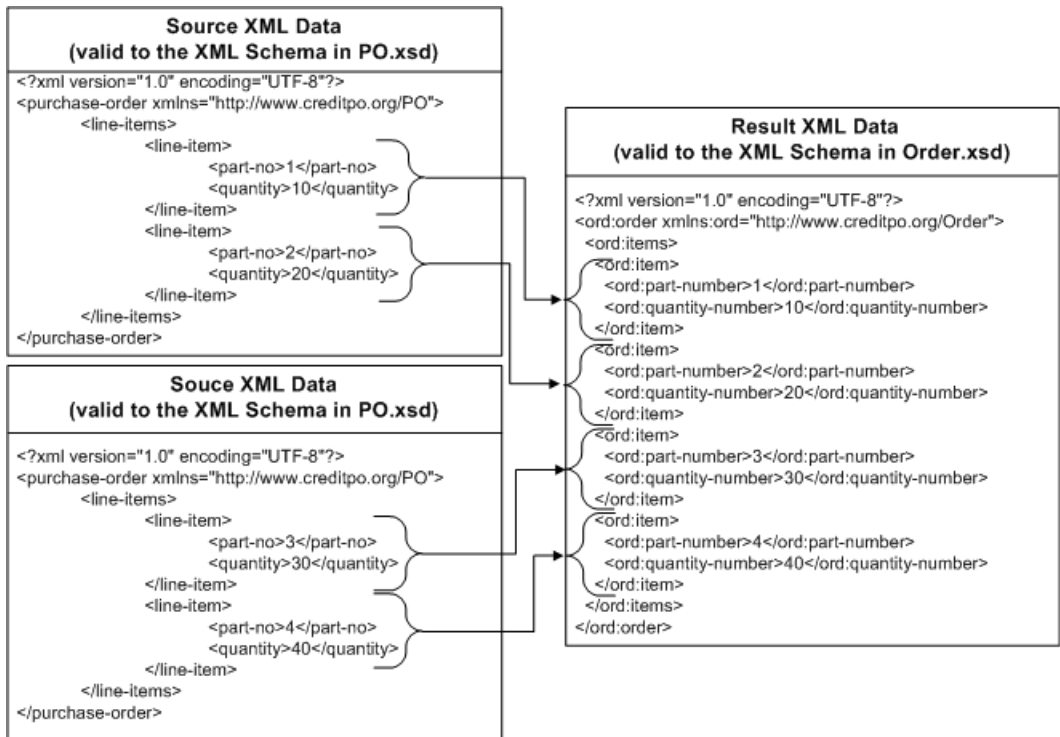
In this *union*, repeating elements of the same type are combined into a larger set but in the preceding *join* example in "Merging the Contents of Repeating Elements" on page 3-7, the contents of repeating elements are merged.

This section describes how to create a transformation which combines two sets of repeating elements using the **Union** option. This section shows how to combine the example XML data shown in the preceding figure.

To Combine Sets of Data of the Same Type

1. Create a Transformation file and a method in the Transformation file. For instructions, see "Creating a Transformation File and a Transformation Method" on page 2-10.

2. Import the XSD file that contains the XML Schema for the source type of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

   For this example, import the file: `PO.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the
   `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\union`

   Note: The `PO.xsd` file used in this example is identical to the `PO.xsd` file used in "Combining Data From Different Schemas" on page 3-3. If you have already imported the `PO.xsd` file for the Combining Data From Different Schemas example, you do not need to import it again.

3. Import the XSD file that contains the XML Schema for the target type of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

   For this example, import the file: `Order.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the
   `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\union` directory.

   Importing schemas files triggers a build of the current Schemas project folder. Wait until the current Schemas folder is built before proceeding to the next step. (The representations of the schemas will not be available in **Available Source Types** and **Available Target Type** panes until build is complete.)

4. Select the Transformation method from a Transformation file.

   To select an existing method, in the **Design View** of the Transformation file:

   a. Right-click the arrow representing the method.

   b. From the drop-down menu, select **Configure XQuery Transformation Method**.

5. Select the source types for the transformation:

   a. In the **Available Source Types** pane, expand the schema and element folders, until you find the desired element.

   b. In the **Available Source Types** pane, select the desired element.

   c. Click **Add**.

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

   For this example, add the `PO.xsd/purchase-order` element *twice*.

6. Select the target type for the transformation:

   a. In the **Available Target Type** pane, expand the `Order.xsd` schema folder.

   b. In the **Available Source Types** pane, select the `Order.xsd/order` element.

   c. Click **Add**.

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

7. Click **Create Transformation**.

   The **Design View** of the XQ file is displayed.

8. In the **Source** pane, select the first repeating element and drag it to the repeating element in the **Target** pane.

   For this example, link the `$purchase_orderDoc/line-items/line-item` repeating element in the **Source** pane to the `order/items/item` repeating element in the **Target** pane.

   A dashed line linking the two repeating elements is displayed. The dashed line with short dashes represents a structural link—a link between two parent structures that does not map data directly. The dashed-line representation for a structural link is shown in the following figure:

   To learn more about links, see "Link Representations" on page 2-25.

9. In the **Source** pane, select the second repeating element and drag it to the repeating element in the **Target** pane.

For this example, link the `$purchase_orderDoc1/line-items/line-item` repeating element in the **Source** pane to the `order/items/item` repeating element in the **Target** pane.

A dashed line linking the two repeating elements is displayed. The dashed line with short dashes represents a structural link—a link between two parent structures that does not map data directly. The dashed-line representation for a structural link is shown in the following figure:

At this point, in the **Constraint Type** pane of the **Constraints** tab, the **Repeatability/Join** option is selected.

Keep the last link selected for the next step.

10. In the **Constraint Type** pane of the **Constraints** tab, select the **Union** option.

11. Create data links from the first set of repeating element nodes.

In the **Source** pane, select each of the subelements of the repeating element and drag them to the analogous subelement of the repeating element in the **Target** pane.

For this example, link the nodes shown in the following table:

Drag This Element From the Source Pane . . .	To This Element in the Target Pane . . .
`$purchase_orderDoc/line-items/line-item/part-no`	`order/items/item/line-number`
`$purchase_orderDoc/line-items/line-item/quantity`	`order/items/item/quantity-number`

Solid blue lines linking the two subelements are displayed. These solid lines represents a data link—a link that converts the value of the source node directly to the value of the target node The solid line representation for a data link is shown in the following figure.

Since the two structural links have the union constraint applied to them, a set of implied data links between the second set of subelements are generated as shown in the following figure.
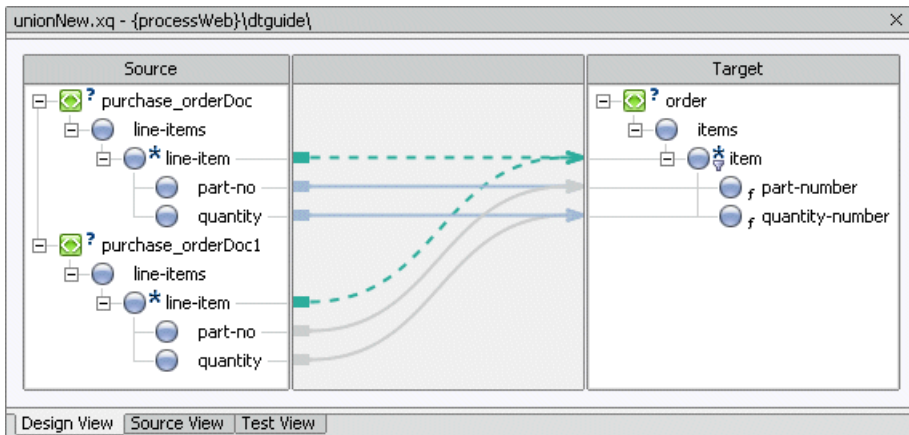
The solid gray lines represents implied links.

12. Select the **Test View** tab.

13. Import XML or non-XML files as source data for the transformation. For more information, see "Creating Maps" on page 2-18.

   For this example, in the **Source Data** pane, select the $purchase_orderDoc node and import the file: InputPO1.xml. In the **Source Data** pane, select the $purchase_orderDoc1 node and import the file: InputPO2.xml. If you installed WebLogic Platform in the c:\bea directory, import these files from the c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\union\XML directory.

   Note: You can cut and past directory paths into the **Name** field of the **Open File to Test** pane to jump to directory locations. If you installed WebLogic Platform in the c:\bea directory, you can jump to the directory that contains the XML files for this example, by pasting the following directory path into the **Name** field: c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\union\XML and then pressing enter.

14. In the **Result Data** pane, click **Test**.

   If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

   In the **Result Data** pane, a graphical representation of the target data is displayed.

15. If the resulting data is XML data, in the **Result Data** pane you can view the resulting data as an XML document by selecting the **XML Source View** tab. The XML result for this example is also displayed in .

16. If desired, you can validate the result data against the associated schema. In the the **Result Data** pane of the **Test View**, click **Validate**.

17. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Note: The nodes are being joined in the union and not the data in the nodes. For example, if you are merging repeating elements in which the value of the `part-no` element is equal to `1`, both `part-no` elements appear in the target data as shown in the following figure.
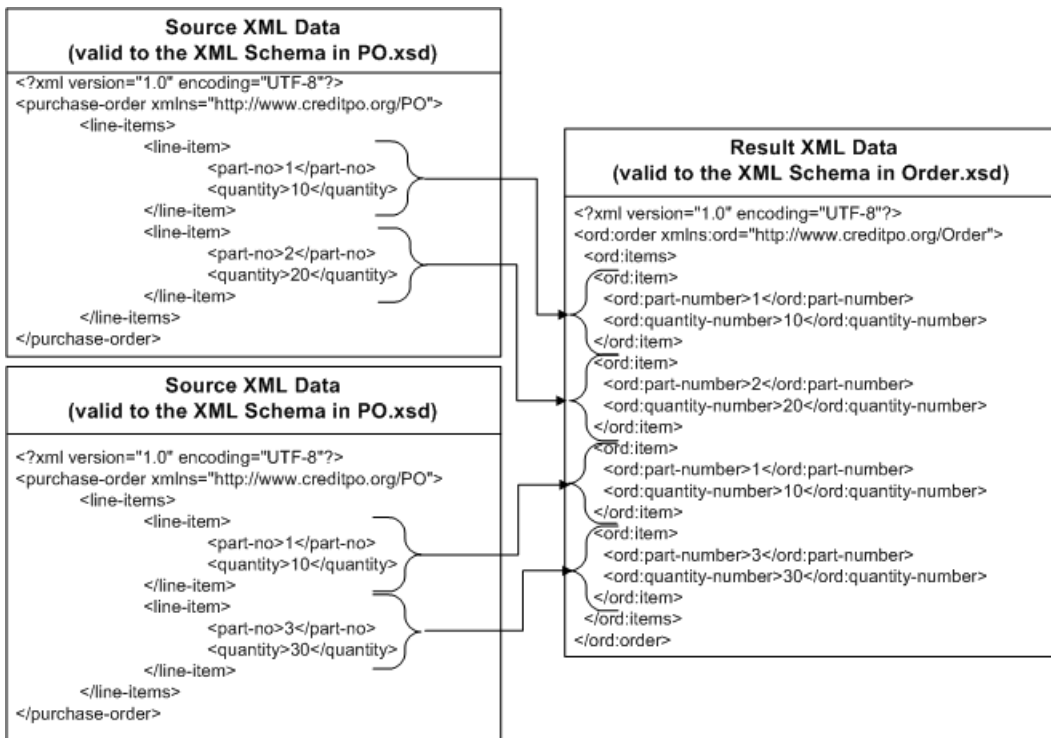

The query returns both repeating elements because it determines the repeating nodes are unique, even if the values of the `part-no` element are both equal to `1`.

Using the Group by Key Fields Option of the Constraints Tab

You can use the **Group by Key Fields** option of the mapper functionality to group data based on a key value, as shown in the following figure:

Figure 3-4 Merging Data Using a Key Value
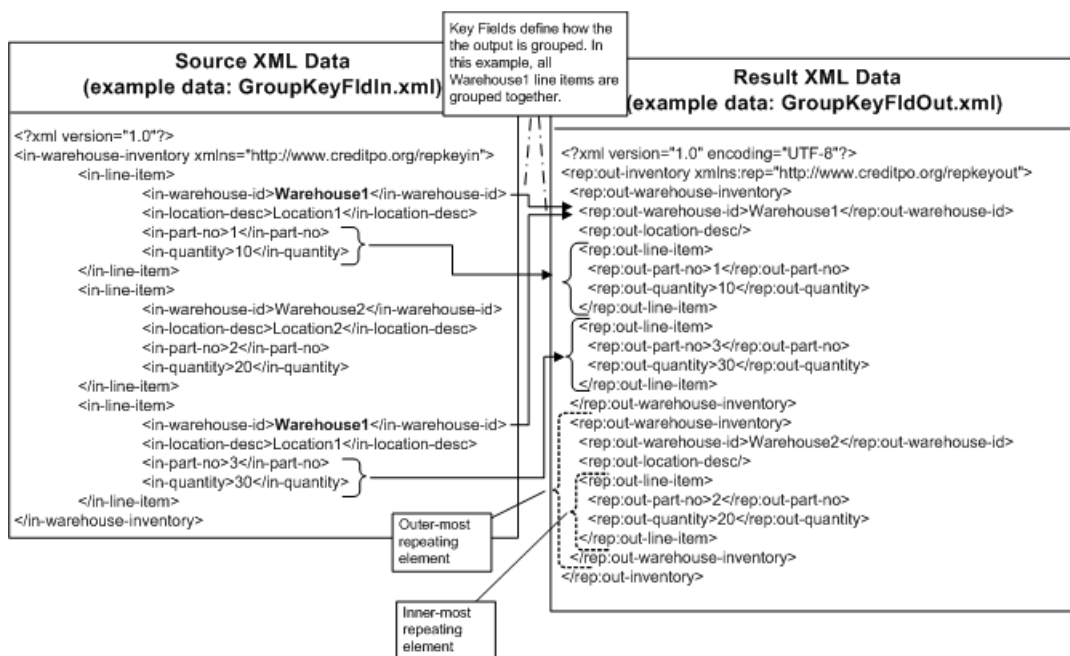
In the example shown in the preceding figure, the `in-warehouse-id` element is the key field that is used to group the output. Both the first and third instances of the `in-line-item` repeating element in the source document contain the same value of the `in-warehouse-id` element (`Warehouse1`), so these elements are grouped together in the target (output) document.

This section describes how to group data by a key field using the **Group by Key Fields** option. This section shows how to group the example XML data shown in the preceding figure.

To Group Sets of Data Based on a Key Field

1. Create a Transformation file and a method in the Transformation file. For instructions, see "Creating a Transformation File and a Transformation Method" on page 2-10.

2. Import the XSD file that contains the XML Schema for the source and target types of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

   For this example, import the files: `GroupKeyFldIn.xsd` and `GroupKeyFldOut.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import these files from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\groupKeyFields` directory.

   Importing schemas files triggers a build of the current Schemas project folder. Wait until the current Schemas folder is built before proceeding to the next step. (The representations of the schemas will not be available in **Available Source Types** and **Available Target Type** panes until build is complete.)

3. Select the source type(s) for the transformation:

   a. In the **Available Source Types** pane, expand the schema and element folders, until you find the desired element.

   b. In the **Available Source Types** pane, select the desired element.

   c. Click **Add**.

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

   For this example, complete this step for the `GroupKeyFldIn.xsd/in-warehouse-inventory` element.

4. Select the target type for the transformation:

   d. In the **Available Target Type** pane, expand the schema and element folders, until you find the desired element.

   For this example, expand the `GroupKeyFldOut.xsd` schema folder.

   e. In the **Available Source Types** pane, select the desired element.

   For this example, select the `GroupKeyFldOut.xsd/out-inventory` element.

   f. Click **Add**.

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

5. Click **Create Transformation**.

   The **Design View** of the XQ file is displayed.

6. Create all the data links.

For this example, make the following links:

– From the **Source** pane, drag the
`$in_warehouse_inventoryDoc/in-line-item/in-warehouse-id` element to the
`out-inventory/out-warehouse-inventory/out-warehouse-id` repeating element
in the **Target** pane.

– From the **Source** pane, drag the
`$in_warehouse_inventoryDoc/in-line-item/in-location-desc` element to the
`out-inventory/out-warehouse-inventory/out-location-desc` element in the
Target pane.

– From the **Source** pane, drag the
`$in_warehouse_inventoryDoc/in-line-item/in-part-no` element to the
`out-inventory/out-warehouse-inventory/out-line-item/out-part-no`
element in the **Target** pane.

– From the **Source** pane, drag the
`$in_warehouse_inventoryDoc/in-line-item/in-quality` element to the
`out-inventory/out-warehouse-inventory/out-line-item/out-quantity`
element in the **Target** pane.

7. Create a link between source repeating element and the inner-most target repeating element.
(See Figure 3-4 for an example of an inner-most and outer-most repeating elements.) In the
Source pane, drag the source repeating element to the inner-most target repeating element
in the **Target** pane.

For this example, link the `$in_warehouse_inventoryDoc/in-line-item` repeating
element to the `out-inventory/out-line-item` repeating element.

A dashed line linking the two repeating elements is displayed. The dashed line with short
dashes represents a structural link—a link between two parent structures that does not map
data directly. The dashed-line representation for a structural link is shown in the following
figure:

To learn more about links, see "Link Representations" on page 2-25.

8. In the **Source** pane, drag the source repeating element that contains the key field(s) to the
outer-most target repeating element that will contain the key field(s) in the **Target** pane.

For this example, link the `$in_warehouse_inventoryDoc/in-line-item` repeating
element to the `out-inventory/out-warehouse-inventory` repeating element.

A dashed line linking the two repeating elements is displayed. The dashed line with short dashes represents a structural link—a link between two parent structures that does not map data directly. The dashed-line representation for a structural link is shown in the following figure:

To learn more about links, see "Link Representations" on page 2-25.

At this point, in the **Constraint** tab, the **Constraint Type** is **Repeatability/Join** but in a preceding step, the **Constraint Type** will be set to **Group by Key Fields**)

Keep this link selected for the next step.

9. In the bottom half of the **Design View** for the XQ file, in the **Constraints** tab select the **Group by Key Fields** option.

10. In the **Select Group Key** pane, select the `in-warehouse-id` node and click **Add**.

The following is displayed in the **Design View** as shown in the following figure:
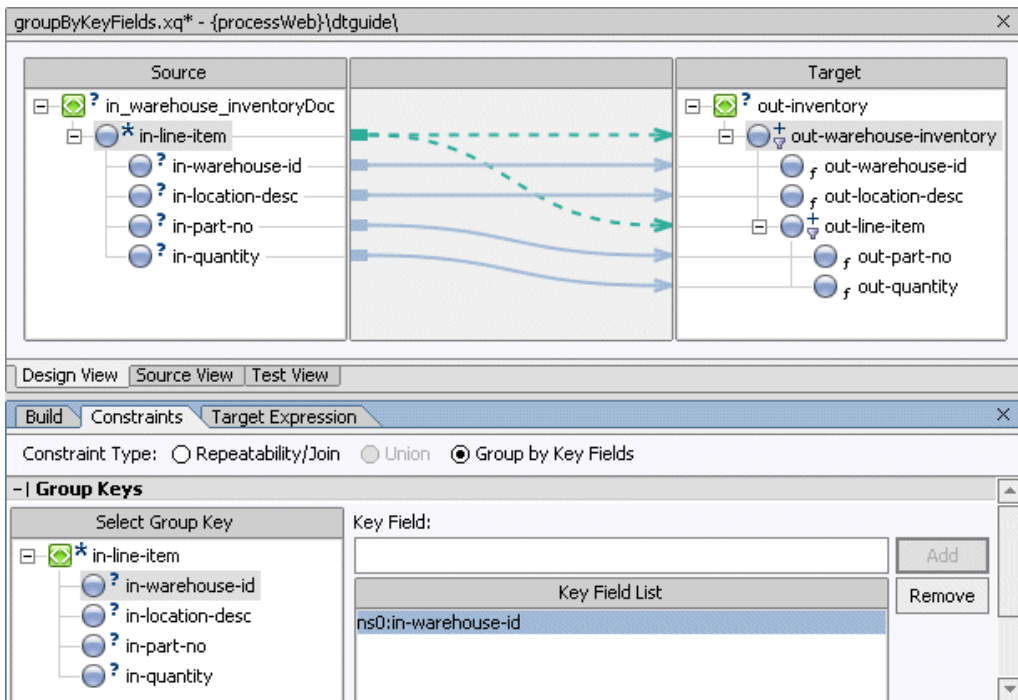
11. Select the **Test View** tab.

12. Import XML or non-XML files as source data for the transformation. For more information, see "Creating Maps" on page 2-18.

   For this example, in the **Source Data** pane, select the `$in_warehouse_inventoryDoc` node and import the file: `GroupKeyFldIn.xml`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\groupKeyFields\XML` directory.

13. In the **Result Data** pane, click **Test**.

   If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

   In the **Result Data** pane, a graphical representation of the target data is displayed. The XML result for this example is also displayed in Figure 3-4.

14. If the resulting data is XML data, in the **Result Data** pane you can view the resulting data as an XML document by selecting the **XML Source View** tab.

15. If desired, you can validate the result data against the associated schema. In the the **Result Data** pane of the **Test View**, click **Validate**.

16. Save the DTF and the XQ file. From the menu bar, choose **File→Save All**.
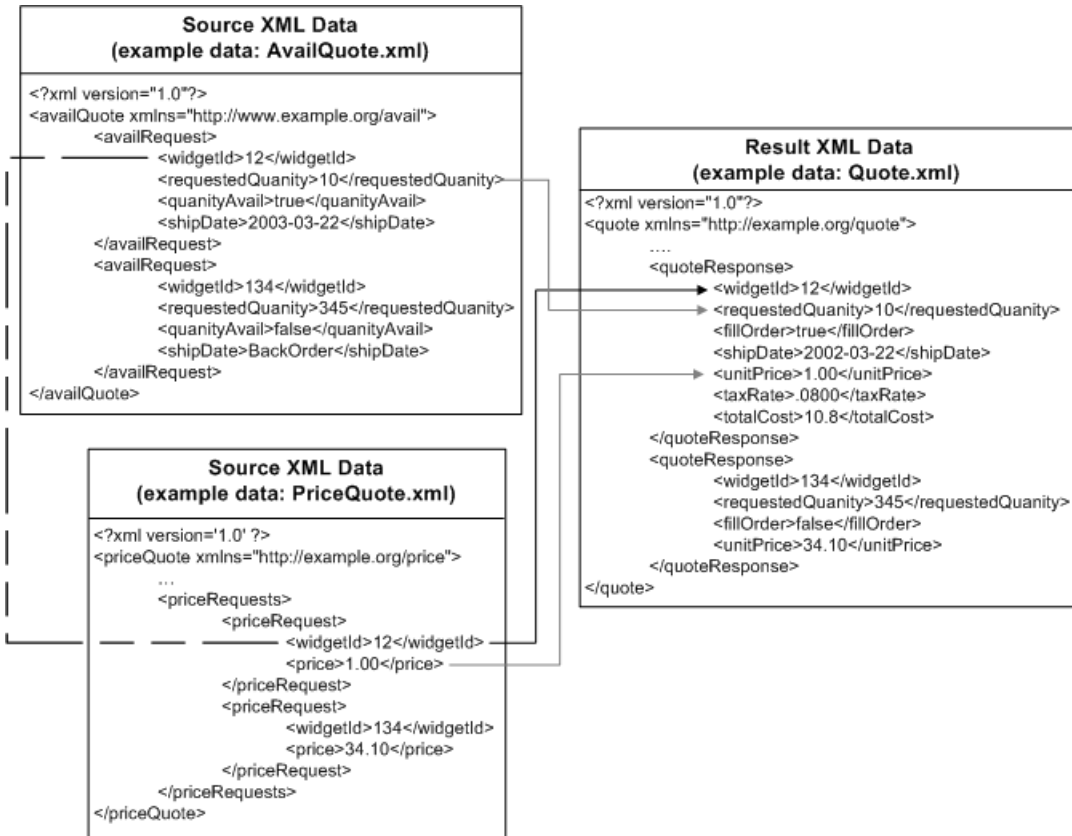
Creating a Conditional Constraint Using the Constraints Tab

In "Merging the Contents of Repeating Elements" on page 3-8, both of the source documents (`PriceQuote.xml` and `AvailQuote.xml`) share the common element `widgetId`. A constraint (as a condition) can be added to the join that specifies if the `widgetId` of the `availRequest` element is equal to the `widgetId` of the `priceRequest` element the merged repeating element `quoteResponse` be returned.

When you create a structural link between a repeating element in the **Design View**, an XQuery `for` loop is generated to iterate through the repeating elements. Creating a constraint using the **Where Clause Expression** pane of the **Constraints** tab adds a `where` clause to the XQuery `for` loops which limits the target repeating elements which are returned during run time. The condition that makes up the `where` clause are evaluated for each iteration of the `for` loop.

Adding this constraint would change the resulting data as shown in Figure 3-2 to the data shown in Figure 3-5.

Figure 3-5 Using a Conditional Constraint to Merge Data


```
Source XML Data
(example data: AvailQuote.xml)

<?xml version="1.0"?>
<availQuote xmlns="http://www.example.org/avail">
        <availRequest>
                <widgetId>12</widgetId>
                <requestedQuanity>10</requestedQuanity>
                <quanityAvail>true</quanityAvail>
                <shipDate>2003-03-22</shipDate>
        </availRequest>
        <availRequest>
                <widgetId>134</widgetId>
                <requestedQuanity>345</requestedQuanity>
                <quanityAvail>false</quanityAvail>
                <shipDate>BackOrder</shipDate>
        </availRequest>
</availQuote>
```

```
Source XML Data
(example data: PriceQuote.xml)

<?xml version='1.0' ?>
<priceQuote xmlns="http://example.org/price">
        ...
        <priceRequests>
                <priceRequest>
                        <widgetId>12</widgetId>
                        <price>1.00</price>
                </priceRequest>
                <priceRequest>
                        <widgetId>134</widgetId>
                        <price>34.10</price>
                </priceRequest>
        </priceRequests>
</priceQuote>
```

```
Result XML Data
(example data: Quote.xml)

<?xml version="1.0"?>
<quote xmlns="http://example.org/quote">
        ....
        <quoteResponse>
                <widgetId>12</widgetId>
                <requestedQuanity>10</requestedQuanity>
                <fillOrder>true</fillOrder>
                <shipDate>2002-03-22</shipDate>
                <unitPrice>1.00</unitPrice>
                <taxRate>.0800</taxRate>
                <totalCost>10.8</totalCost>
        </quoteResponse>
        <quoteResponse>
                <widgetId>134</widgetId>
                <requestedQuanity>345</requestedQuanity>
                <fillOrder>false</fillOrder>
                <unitPrice>34.10</unitPrice>
        </quoteResponse>
</quote>
```

For a step-by-step walk through of using the mapping functionality to create a join with a constraint, see *Tutorial: Building Your First Data Transformation*. Specifically, the join is created in Step 4: Mapping Repeating Elements—Creating a Join in the *Tutorial: Building Your First Data Transformation*.

In the next section an additional condition is added to the where clause. To learn more see "Adding a Constraint With Multiple Conditions" on page 3-20.
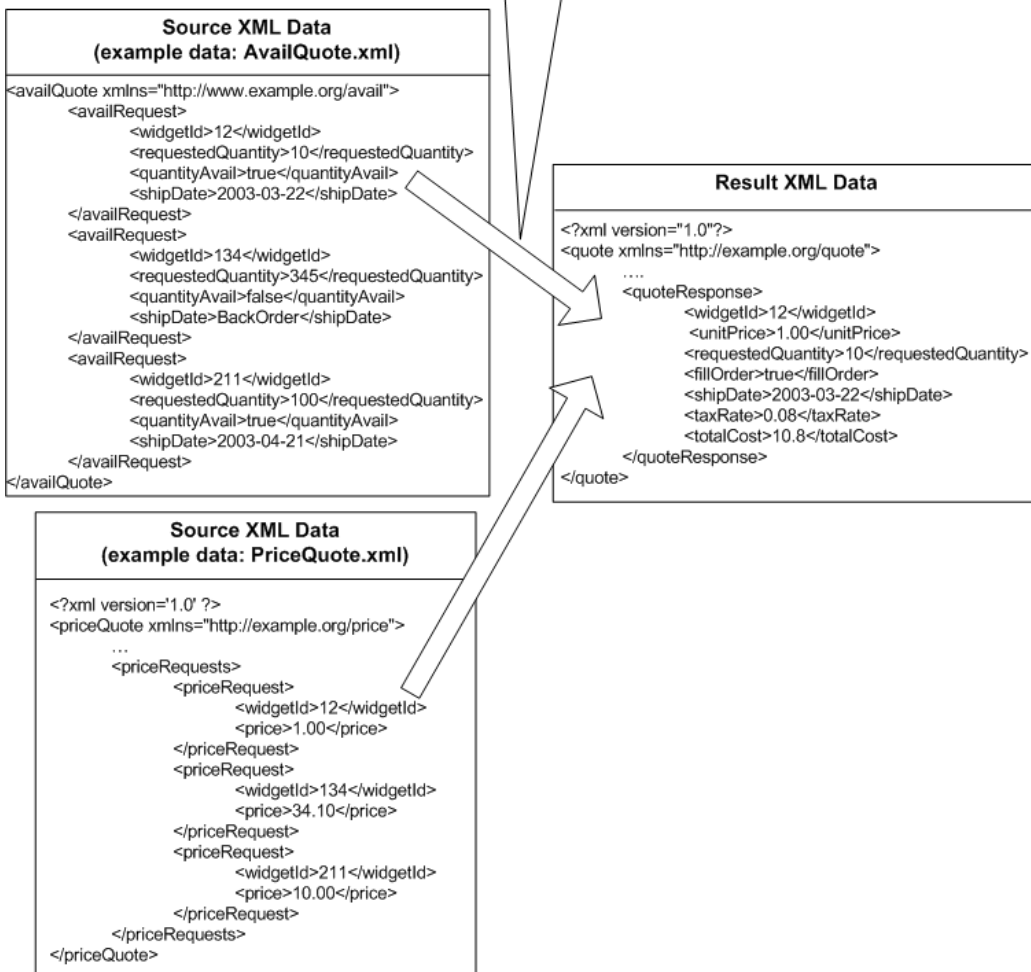
Adding a Constraint With Multiple Conditions

In "Creating a Conditional Constraint Using the Constraints Tab" on page 3-19, a constraint was added to the join that specifies if the widgetId of the availRequest element is equal to the

widgetId of the priceRequest element the merged repeating element quoteResponse be returned. Creating a constraint using the **Where Clause Expression** pane of the **Constraints** tab adds a where clause to the XQuery for loops which limits the target repeating elements which are returned during run time. During run time, the for loop will iterate only over those repeating elements that meet the complex condition.. In this section you add another condition (resulting in a complex condition) to the where of the for loop to further limit what is returned by the for loop as shown in the figure.

Figure 3-6 Complex Constraint

The constraint (the `where` clause of the `for` loop) contains the following complex condition:
```
where (data($priceRequest/ns0:widgetId) = data($availRequest/ns1:widgetId)
       and data($availRequest/ns1:requestedQuantity) < 50)
```
The `where` clause limits what is returned from the `for` loop during run time.
If the two `widgetIds` are equal and the `requestedQuanity` is less than 50, the `quoteResponse` element is returned.

Source XML Data
(example data: AvailQuote.xml)
```
<availQuote xmlns="http://www.example.org/avail">
        <availRequest>
                <widgetId>12</widgetId>
                <requestedQuantity>10</requestedQuantity>
                <quantityAvail>true</quantityAvail>
                <shipDate>2003-03-22</shipDate>
        </availRequest>
        <availRequest>
                <widgetId>134</widgetId>
                <requestedQuantity>345</requestedQuantity>
                <quantityAvail>false</quantityAvail>
                <shipDate>BackOrder</shipDate>
        </availRequest>
        <availRequest>
                <widgetId>211</widgetId>
                <requestedQuantity>100</requestedQuantity>
                <quantityAvail>true</quantityAvail>
                <shipDate>2003-04-21</shipDate>
        </availRequest>
</availQuote>
```

Result XML Data
```
<?xml version="1.0"?>
<quote xmlns="http://example.org/quote">

    ....
    <quoteResponse>
            <widgetId>12</widgetId>
             <unitPrice>1.00</unitPrice>
            <requestedQuantity>10</requestedQuantity>
            <fillOrder>true</fillOrder>
            <shipDate>2003-03-22</shipDate>
            <taxRate>0.08</taxRate>
            <totalCost>10.8</totalCost>
    </quoteResponse>
</quote>
```

Source XML Data
(example data: PriceQuote.xml)
```
<?xml version='1.0' ?>
<priceQuote xmlns="http://example.org/price">
        ...
        <priceRequests>
                <priceRequest>
                        <widgetId>12</widgetId>
                        <price>1.00</price>
                </priceRequest>
                <priceRequest>
                        <widgetId>134</widgetId>
                        <price>34.10</price>
                </priceRequest>
                <priceRequest>
                        <widgetId>211</widgetId>
                        <price>10.00</price>
                </priceRequest>
        </priceRequests>
</priceQuote>
```

To Add a Condition to the Join in the Data Transformation Tutorial

1. From the BEA WebLogic Workshop menu bar, choose **File→New→Application...**.

   The **New Application** dialog box is displayed.

2. In the left pane, select the **Tutorial** folder.

3. In the right pane, select **Tutorial: Process Application**.

4. In the **Name** field, enter `Tutorial_Add_Condition`.

5. From the **Server** drop-down menu, select the integration server. For example, if you installed WebLogic Platform in the `c:\bea` directory on Windows, the path to the integration server is:

   `c:\bea\weblogic81\samples\domains\integration`

6. Click **Create**.

   Your Tutorial Process application is created and displayed in the **Application** tab.

7. In the **Application** tab, expand the **Tutorial_Add_ConditionWeb** folder.

   The directories and files that make up the Tutorial project are displayed.

8. In the **Application** tab, expand the **Tutorial_Add_ConditionWeb/requestquote** folder.

   The DTF, XQ, and JPD files used in the tutorial are displayed. These files are part of the Tutorial project.

9. View `Join.xq` in the **Design View**. In the **Application** tab, double-click `Tutorial_Process_Application\Tutorial_Add_ConditionWeb\requestquote\TutorialJoin.dtf\Join.xq` and select the **Design View** tab.

10. Select the link between the **availQuoteDoc\availRequest** node and the **quote\quoteResponse** node in the **Target** pane.

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

11. Create the second condition in the `where` clause:

    a. In the **Source** pane select the **availQuoteDoc/availRequest/requestedQuanity** node and drag-and-drop it into the **Left Hand Expression** section of the **Where Clause Expression** pane.

    b. Select the **<** operator.

c. Remove the text in the **Right Hand Expression** section of the **Where Clause Expression** pane.

d. In the **Right Hand Expression** section of the **Where Clause Expression** pane enter 50.

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

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

f. Click **Add.**

   The second condition is added to the where clause of the for loop.

12. To run the query, select the **Test View** of the XQ file.

13. In this step, you import AvailQuote.xml for the source parameter: $availQuoteDoc:

a. From the drop-down menu in the **Source Data** pane, select **$availQuoteDoc**.

b. Click **Import...** .

   The **Open XML File to Test** dialog box is displayed.

c. Double-click the **requestquote** folder.

d. Double-click the **testxml** folder.

e. Double-click the AvailQuote.xml file.

14. In this step, you import PriceQuote.xml for the source parameter: $priceQuoteDoc:

a. From the drop-down menu in the **Source Data** pane, select **$priceQuoteDoc**.

b. Click **Import...** .

   The **Open XML File to Test** dialog box is displayed.

c. Double-click the **requestquote** folder.

d. Double-click the **testxml** folder.

e. Double-click the PriceQuote.xml file.

15. Enter the **taxRate** value:

a. From the drop-down menu in the **Source Data** pane, select **$taxRate**.

b. In the **Node Value** field of the **$taxRate** node, double-click on the existing value, and enter: `0.08` and click **Enter**.

16. In the **Result Data** pane, click **Test**.

If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

In the **Result Data** pane, a graphical representation of the target data is displayed. The XML result for this example is also displayed in Figure 3-6.

During run time, the `where` clause limits what is returned by the `for` loop. If the two `widgetId`s are equal and the `requestedQuanity` is less than `50`, the `quoteResponse` element is returned.

Adding Nested If-Then-Else Expressions with Complex Conditions to a Link

The following example shows adding nested *if-then-else* expressions to a link to supply the appropriate supplier id for an incoming purchase order. In addition this section shows how you can use functions in *if-then-else* expressions.

In this example, if `partId` is between `0` and `200`, use the `Supplier` with the id of `12453`. If the `partId` is between `201` and `400`, the `Supplier` id is dependent on the `quantity` requested. If the `quantity` requested is over the maximum volume for one `Supplier` (id: 35667) use the high volume `Supplier` (id: 20045.) In addition, this example appends the source `countryCode` to the supplier id code using the `concat` function call as shown in the following figure.

Figure 3-7 Complex If-Then-Else
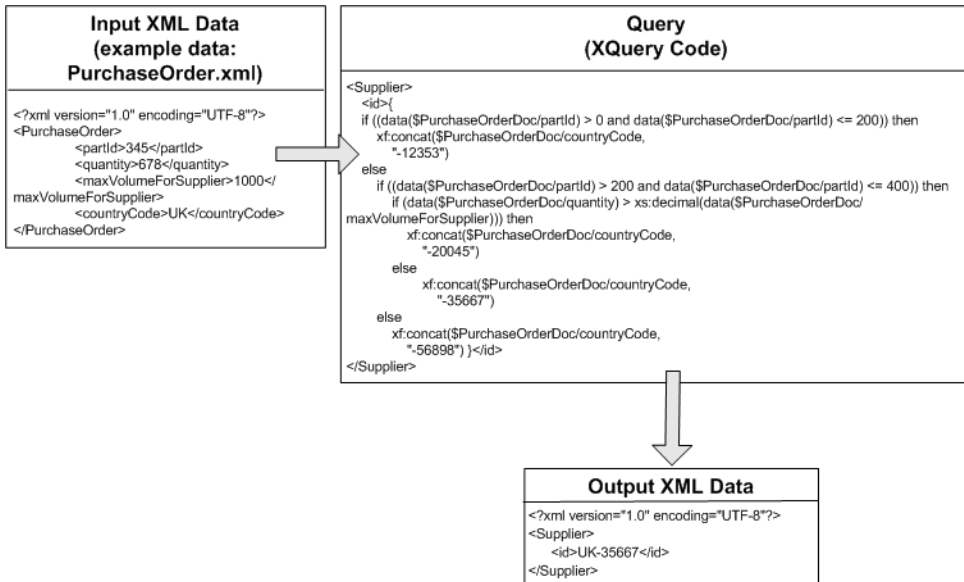
To Create the If-Then-Else Example with Complex Conditions

1. Open a query in the mapper pane. For instructions see "General Steps to Open or Create a Query in the Mapper" on page 2-20.

   For this example, import the files: `PurchaseOrder.xsd` and `Supplier.xsd` files (step 2 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.) If you installed WebLogic Platform in the `c:\bea` directory, import these files from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\ifThenElse` directory.

   For this example, select and add the `PurchaseOrder.xsd/PurchaseOrder` node as an source type (step 4 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.)

   For this example, select and add the `Supplier.xsd/Supplier` node as an target type (step 4 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.)

2. In the **Design View**, create a link by dragging-and-dropping the `PurchaseOrderDoc/countryCode` Source element over the `Supplier/id` Target node.

   Keep the link selected for the next step.

3. Select the **Target Expression** tab. (If the **Target Expression** tab is not visible in WebLogic Workshop, choose **View**→**Windows**→**Target Expression** from the menu bar.)

4. Select the **If Then Else** option.

   An XQuery *if-then-else* construct is added to the link.

5. Click **If Condition**.

   The **If Condition** pane appears.

6. Add the first condition to the *if* section of the *if-then-else*:

   a. From the **Source** pane, drag the `PurchaseOrderDoc/partId` element into the **Left Hand Expression** section of the **If Condition** pane.

   b. Select the operator: `>`.

   c. In the **Right Hand Expression** section of the **If Condition** pane, enter: `0` and click **Add**.

   The first condition is added to the *if* section of the *if-then-else*.

7. Construct the second condition to the *if* section of the *if-then-else*:

   a. From the **Source** pane, drag-and-drop the `PurchaseOrderDoc/partId` element into the **Left Hand Expression** section of the **If Condition** pane.

   b. Select the operator: `<=`.

   c. In the **Right Hand Expression** section of the **If Condition** pane enter: `200`.

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

   The **Join Type** determines how the conditions that make up *if* section of the *if-then-else* are evaluated during run time.

9. Click **Add.**

   The second condition is added to the *if* section of the *if-then-else*.

   During run time, for example if `partId` is equal to `100`, the *if* condition is evaluated as described here:

   – The first condition (`partId>0`) evaluates to *true*.

   – The second condition (`partId<=200`) evaluates to *true* resulting in the *if* condition: (*true and true*).

- The *if* condition: (*true and true*) evaluates to *true* resulting in execution of the *then* expression.

10. Click **Then Expression**.

The contents of the **Then Expression** are selected.

Keep the text selected for the next step.

11. Click the *Delete* key.

The text in the Then Expression pane is deleted.

12. Create the *then* expression:

a. Select the **Palette**. (If the **Palette** is not visible in WebLogic Workshop, choose **View→Windows→Palette** from the menu bar.)

b. In the **Palette**, expand **String Functions**.

c. Select the **concat** function and drag it into the **Then Expression** pane. Leave the first `$string-var` argument selected for the next step.

d. From the **Source** pane, drag-and-drop the `PurchaseOrderDoc/countryCode` element over the first `$string-var` argument of the `concat` function in the **Then Expression** pane.

e. Select the text: `$string-var` and click the *Delete* key. Leave the cursor in the current location.

f. Enter: `"-12453"`.

The `$string-var` argument is replaced by the string: `"-12453"`.

g. Click **Apply**.

The following XQuery code of the *then* expression is saved:

```
xf:concat($PurchaseOrderDoc/countryCode,
        "-12453")
```

During run time, when the *if* condition evaluates to *true*, the *then* expression is evaluated. When the *then* expression created in this step is evaluated, it concatenates the `countryCode` with supplier id of `12453` and returns this value as the Target `Supplier/id`.

13. In this step, you add a nested *if-then-else* to the *else*.

The nested *if-then-else* added in this step, tests if the supplied source `partId` is between `201` and `400`. For `partIds` in this range, the returned `Supplier id` is dependent on the

`quantity` requested. If the `quantity` requested is over the maximum volume for one `Supplier` (id: `20045`) the high volume `Supplier` (id: `35667`) is returned.

To add the nested *if-then-else* to the *else* expression:

a. Select the down arrow to the right of **Else Expression**, and from the drop-down menu select **Insert Nested If-Then-Else**.

    A nested *if-then-else* is added to the *else* and **Else** becomes **Else If Condition**.

b. Add the first condition to the *if* section of the second level *if-then-else*:

    From the **Source** pane, drag-and-drop the `PurchaseOrderDoc/partId` element into the **Left Hand Expression** section of the **If Condition** pane.

    Select the operator: **>**.

    In the **Right Hand Expression** section of the **Else If Condition** pane, enter: `200` and click **Add**.

    The first condition is added to the *if* section of the second level *if-then-else*.

c. Construct the second condition to the *if* section of the second level *if-then-else*:

    From the **Source** pane, drag the `PurchaseOrderDoc/partId` element into the **Left Hand Expression** section of the **Else If Condition** pane.

    Select the operator: **<=**.

    In the **Right Hand Expression** section of the **Else If Condition** pane, enter: `400`.

d. From the **Join Type** field, select the **AND** option.

    The selected **Join Type** determines how the conditions that make up *if* section of the *if-then-else* are evaluated during run time.

e. Click **Add**.

    The second condition is added to the *if* section of the second level *if-then-else*.

14. Select **Else If Condition** of the second level *if-then-else* to collapse the **Else If Condition** pane.

    In the **Target Expression** tab the following is displayed as shown in the following figure.
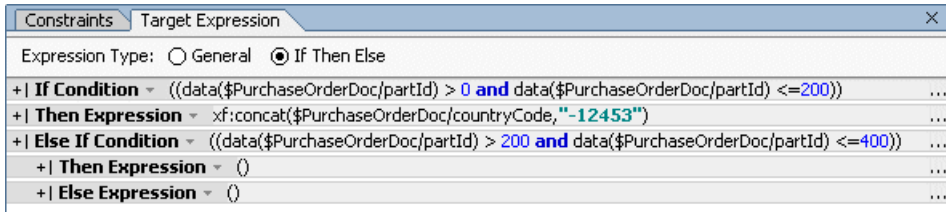
15. In this step, you add another nested *if-then-else* to the *then* of the second level *if-then-else*. This *if-then-else* determines if the quantity requested is over the maximum volume for one Supplier (id: 20045) return the high volume Supplier (id: 35667).

To add the third *if-then-else* to the *then* expression:

a. Select the down arrow to the right of the second **Then Expression** at the bottom of the pane, and from the drop-down menu select **Insert Nested If-Then-Else** as shown in the following figure.
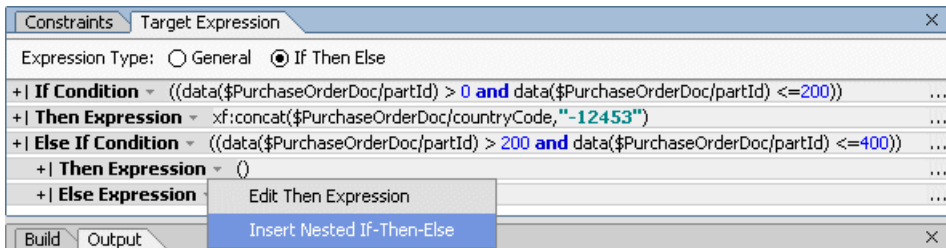
A nested third level *if-then-else* is added to the *then* and **Then Expression** becomes **Then If Condition**.

b. Add a condition to the *if* section of the third level *if-then-else*:

From the **Source** pane, drag the PurchaseOrderDoc/quantity element into the **Left Hand Expression** section of the **Then If Condition** pane as in the following figure.
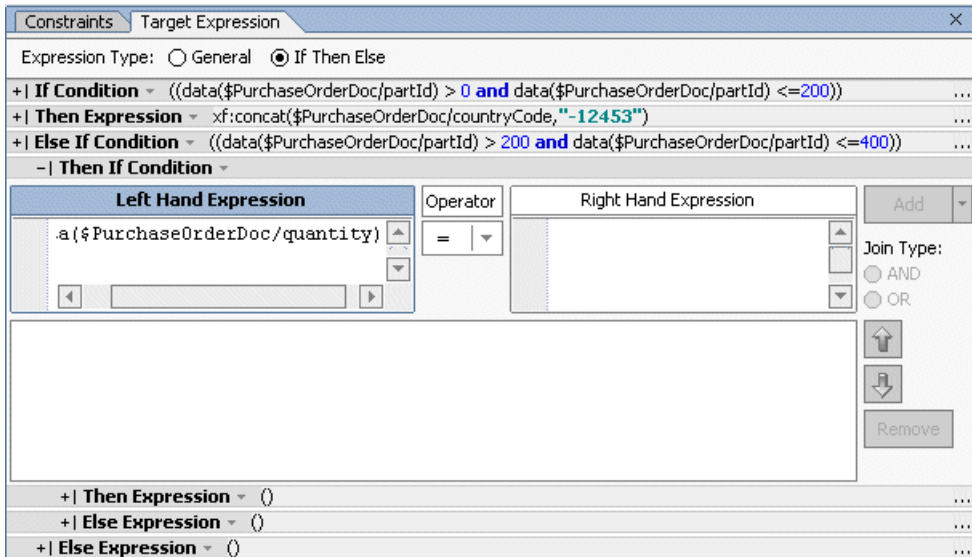
Select the operator: **>**.

Select the **Palette**. (If the **Palette** is not visible in WebLogic Workshop, choose **View**—**Windows**—**Palette** from the menu bar.)

In the **Palette**, expand **Type Functions**.

Select the **decimal** function and drag-and-drop it into the **Right Hand Expression** section of the **Then If Condition** pane. Leave the first `$string-var` argument selected for the next step.

From the **Source** pane, drag the `PurchaseOrderDoc/maxVolumeForSupplier` element over the first `$string-var` argument of the `decimal` function in the **Right Hand Expression** section of the **Then If Condition** pane and click **Add**.

A condition is added to the *if* section of the third level *if-then-else*.

Warning: You must explicitly cast the `PurchaseOrderDoc/maxVolumeForSupplier` element as a decimal to force the comparison between the **Left Hand Expression** and the **Right Hand Expression** to be a numerical comparison and not a lexicographical one.

16. Create the *then* expression of the third level *if-then-else*:

   a. Select the **Then Expression** of the third level *if-then-else*.

The following is displayed in the **Target Expression** tab.

b. Select the **Palette**. (If the **Palette** is not visible in WebLogic Workshop, choose **View**—**Windows**—**Palette** from the menu bar.)

c. In the **Palette**, expand **String Functions**.

d. Select the **concat** function and drag it into the **Then Expression** pane. Leave the first `$string-var` argument selected for the next step.

e. From the **Source** pane, drag-and-drop the `PurchaseOrderDoc/countryCode` element over the first `$string-var` argument of the `concat` function in the **Then Expression** pane.

f. Select the text: `$string-var` and click the *Delete* key. Leave the cursor in the current location.

g. Enter: `"-20045"`.

The `$string-var` argument is replaced by *"-20045"*.

h. Click **Apply**.

The XQuery code of the *then* expression of the third level *if-then-else* is saved.

During run time, when the *if* condition evaluates to *true*, the *then* expression is evaluated. If the *then* expression created in this step is evaluated, it concatenates the countryCode with supplier id of 20045 and returns this value as the Target Supplier/id.

17. Create the *else* expression of the third level *if-then-else*:

a. Select the **Else Expression** of the third level *if-then-else*.

The following is displayed in the **Target Expression** tab.

b. Select the **Palette**. (If the **Palette** is not visible in WebLogic Workshop, choose **View→Windows→Palette** from the menu bar.)

c. In the **Palette**, expand **String Functions**.

d. Select the **concat** function and drag it into the **Then Expression** pane. Leave the first $string-var argument selected for the next step.

e. From the **Source** pane, drag-and-drop the PurchaseOrderDoc/countryCode element over the first $string-var argument of the concat function in the **Then Expression** pane.

f. Select the text: $string-var and click the *Delete* key. Leave the cursor in the current location.

g. Enter: `"-35667"`.

The `$string-var` argument is replaced by `"-35667"`.

h. Click **Apply**.

The XQuery code of the *else* expression of the third level *if-then-else* is saved.

During run time, when the *if* condition evaluates to *false*, the *else* expression is evaluated. If the *else* expression created in this step is evaluated, it concatenates the `countryCode` with supplier id of `35667` and returns this value as the Target `Supplier/id`.

18. Create the *else* expression of the second level *if-then-else*:

a. Select the **Else Expression** of the second level *if-then-else*.

The following is displayed in the **Target Expression** tab.
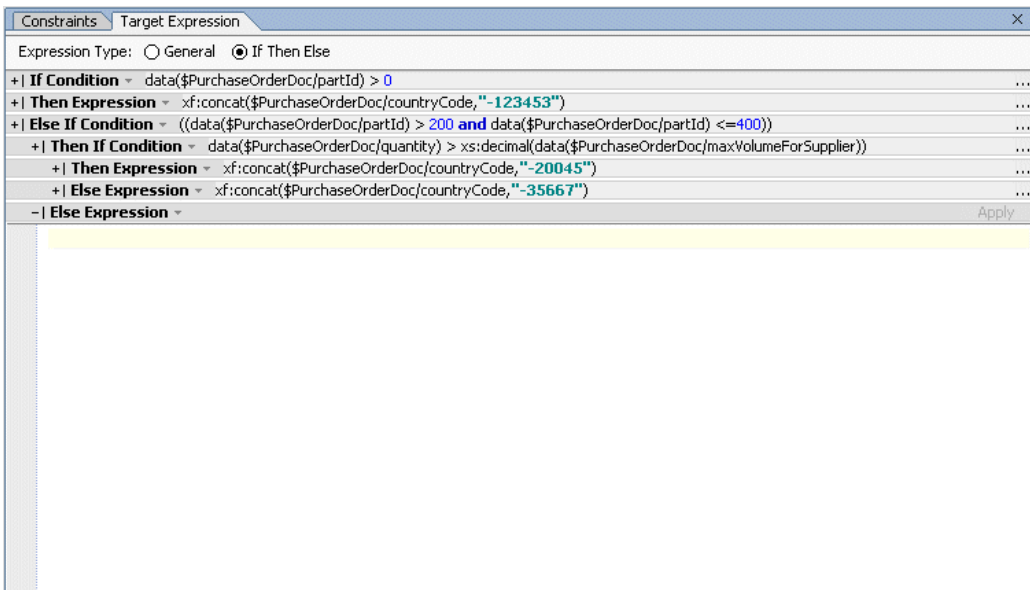
b. In the **Palette**, expand **String Functions**.

c. Select the **concat** function and drag it into the **Then Expression** pane. Leave the first `$string-var` argument selected for the next step.

d. From the **Source** pane, drag-and-drop the `Purchase Order Doc/contracted` element over the first `$string-var` argument of the `concat` function in the **Then Expression** pane.

e. Select the text: `$string-var` and click the *Delete* key. Leave the cursor in the current location.

f. Enter: "`-56898`".

The `$string-var` argument is replaced by "`-56898`".

g. Click **Apply**.

19. Select the **General** option, to view the generated XQuery source code. The following XQuery code is displayed:

```
if ((data($PurchaseOrderDoc/partId) > 0
and data($PurchaseOrderDoc/partId)<= 200)) then
    xf:concat($PurchaseOrderDoc/countryCode,
        "-12353")
else
    if ((data($PurchaseOrderDoc/partId) > 200
    and data($PurchaseOrderDoc/partId) <= 400)) then
        if (data($PurchaseOrderDoc/quantity) >
        xs:decimal(data($PurchaseOrderDoc/maxVolumeForSupplier))) then
            xf:concat($PurchaseOrderDoc/countryCode,
            "-20045")
        else
            xf:concat($PurchaseOrderDoc/countryCode,
            "-35667")
    else
        xf:concat($PurchaseOrderDoc/countryCode,
            "-56898")
```

20. To run the query, select the **Test View** of the XQ file.

21. Import the file: `PurchaseOrder.xml`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\ifThenElse\XML` directory.

Note: You can cut and past directory paths into the **Name** field of the **Open File to Test** pane to jump to directory locations. If you installed WebLogic Platform in the `c:\bea` directory, you can jump to the directory that contains the XML files for this example, by pasting the following directory path into the **Name** field: `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\ifThenElse\XML` and then pressing enter.

22. In the **Result Data** pane, click **Test**.

    If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

    In the **Result Data** pane, a graphical representation of the target data is displayed.

    In the **Result Data** pane, the resulting value of `Supplier/id` element is `UK-35667`. The specified source `partId` (`345`) is between `200` and `400` and the specified source `quantity` (`678`) is less than the specified source `maxVolumeForSupplier` (`1000`), so the *else* expression: `xf:concat($PurchaseOrderDoc/countryCode, "-",` `xf:string(35667))` is evaluated (shown in bold in the preceeding XQuery code listing), resulting in the string: `UK-35667` being returned as the value of the `Supplier/id` element as shown in the following resulting XML data:

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <Supplier>
       <id>UK-35667</id>
    </Supplier>
    ```

23. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Creating a Transformation Between a Non-Repeating Source Element and Repeating Target Group

This example shows how to map a XML Schema simple type (for example: a non-repeating single element) to a XML Schema complex type (for example: a repeating element) as shown in the following figure. In this example, you will create a transformation that during run time will take a single source element and maps it to repeating target element as shown in the following figure.

Figure 3-8 Non-Repeating Source Element to Repeating Target Group
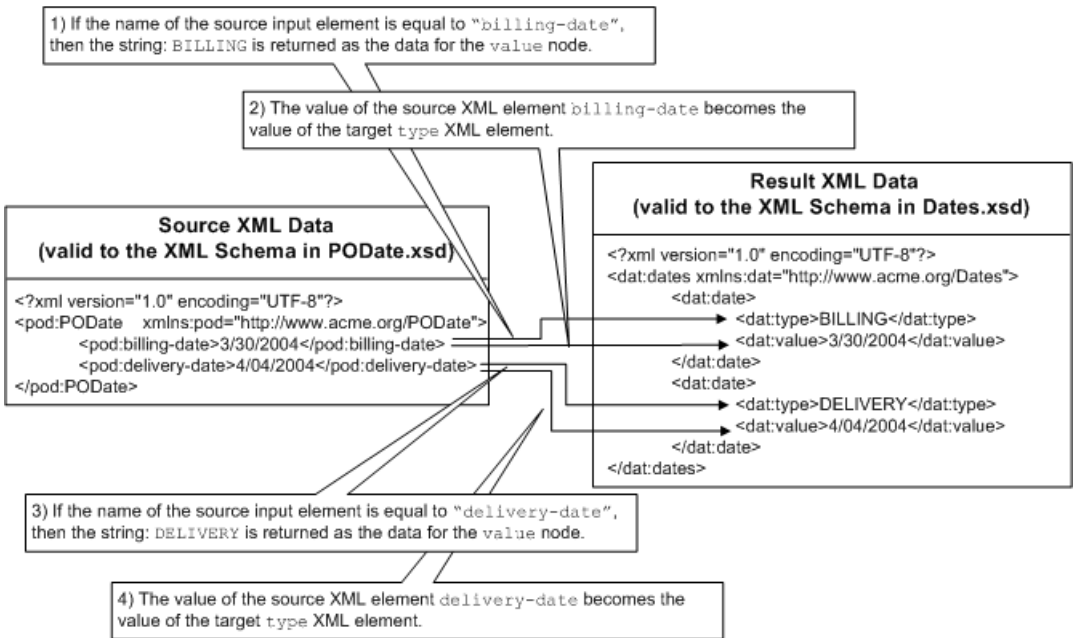
To Create and Run a Transformation Between a Non-Repeating Source Element and a Repeating Target Group

1. Create a Transformation file and a method in the Transformation file. For instructions, see "Creating a Transformation File and a Transformation Method" on page 2-10.

2. Import the XSD file that contains the XML Schema for the source type of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

   For this example, import the file: `PODate.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\singleToRepeat` directory.

   Note: The `PODate.xsd` file used in this example is identical to the `PODate.xsd` file used in "Creating a Transformation Between a Repeating Source Group and Non-Repeating Target Element" on page 3-43. If you have already imported the `PODate.xsd` file for the Creating a Transformation Between a Repeating Source

Group and Non-Repeating Target Element example, you do not need to import it again.

3. Import the XSD file that contains the XML Schema for the target type of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

For this example, import the file: `Dates.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\singleToRepeat` directory.

Note: The `Dates.xsd` file used in this example is identical to the `Dates.xsd` file used in "Creating a Transformation Between a Repeating Source Group and Non-Repeating Target Element" on page 3-43. If you have already imported the `Dates.xsd` file for the Creating a Transformation Between a Repeating Source Group and Non-Repeating Target Element example, you do not need to import it again.

Importing schemas files triggers a build of the current Schemas project folder. Wait until the current Schemas folder is built before proceeding to the next step. (The representations of the schemas will not be available in **Available Source Types** and **Available Target Type** panes until build is complete.)

4. Select the Transformation method from a Transformation file.

To select an existing method, in the **Design View** of the Transformation file:

a. Right-click the arrow representing the method.

b. From the drop-down menu, select **Configure XQuery Transformation Method**.

5. Select the source type for the transformation:

a. In the **Available Source Types** pane, expand the `PODate.xsd` folder.

b. In the **Available Source Types** pane, select the `PODate.xsd/PODate` element.

c. Click **Add**.

The elements and attributes that make up the `PODate.xsd/PODate` element are displayed in the **Selected Source Types** pane.

6. Select the target type for the transformation:

a. In the **Available Target Types** pane, expand the `Dates.xsd` folder.

b. In the **Available Target Types** pane, select the desired element.

For this example, select the `Dates.xsd/date` element.

c. Click **Add**.

The elements and attributes that make up the `Dates.xsd/date` element are displayed in the **Selected Target Types** pane.

7. Click **Create Transformation**.

The **Design View** of the XQ file is displayed.

8. In the **Source** pane, select the element (`PODateDoc/billing-date`) and drag it the repeating element (`dates/date`) in the **Target** pane.

A dashed line linking the two elements is displayed. The dashed line with short dashes represents a structural link—a link that does not map data directly. The dashed-line representation for a structural link is shown in the following figure.

9. In the **Source** pane, select the element (`PODateDoc/delivery-date`) and drag it the repeating element (`dates/date`) in the **Target** pane.

A dashed line (representing a structural link) between two elements is displayed and the two dates (`PODateDoc/delivery-date` and `PODateDoc/billing-date`) are automatically unioned together. In addition, the **Structural Link** variable `PODate` is automatically created and displayed in the **Data Palette**.

The following XQuery is code is generated:

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

`$PODateDoc` is the source or input variable to the transformation, while `$PODate` is the iteration variable for union of the elements: `$PODateDoc/ns0:billing-date` and `$PODateDoc/ns0:delivery-date`.

During run time, the `for` loop in the preceding XQuery code is executed twice. The first time the `for` loop is run, the iteration variable `$PODate` is equal to the first element in the union: `$PODateDoc/ns0:billing-date` and the second time the `for` loop is run the

iteration variable `$PODate` is equal to the second element in the union: `$PODateDoc/ns0:delivery-date`. The XML data returned by the preceding query returns two empty elements: `<ns1:date/>`. The following steps will add the XQuery code to return the billing and delivery dates to the query.

10. In the **Source** pane, select the `PODateDoc/billing-date` element and drag it the `dates/value` element in the **Target** pane.

   Two data links are created as shown in the following figure.
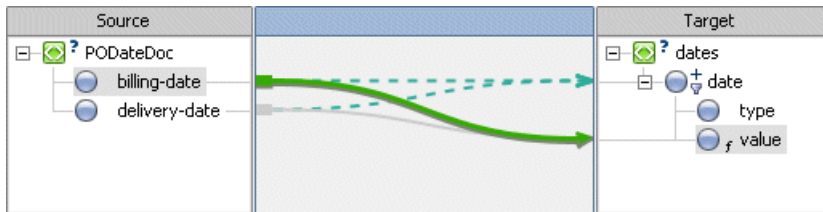
   Since the structural links (`PODateDoc/billing-date` to `dates/date`) and (`PODateDoc/delivery-date dates/date`) are unioned together, when you created the link from the `PODateDoc/billing-date` element to the `dates/value` element (`dates/value`), a second data link between the `PODateDoc/deilivery-date` element and `dates/value` element is automatically created.

11. In the **Source** pane, select the `PODateDoc/billing-date` element and drag it the `dates/type` element in the **Target** pane.

   Two data links are created.

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

12. Select the **Target Expression** tab. (If the **Target Expression** tab is not visible in WebLogic Workshop, choose **View→Windows→Target Expression** from the menu bar.)

13. Select the **If Then Else** option.

   The XQuery *if-then-else* construct is added to the link. For example, the following XQuery source code segment for the link is replaced:

```
data($PODate)
```

   By the following XQuery source code segment for the link:

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

14. Click **If Condition**.

   The **If Condition** pane appears.

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

   a. Select the **Palette**. (If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the menu bar.)

   b. In the **Palette**, expand **Node Functions**.

   c. Select the **local-name** function and drag it into the **Left Hand Expression** pane of the **If Condition** pane. Leave the `$node-var` argument selected.

   d. Select the **Data Palette**. (If the **Data Palette** is not visible in WebLogic Workshop, choose **View—Windows—Data Palette** from the menu bar.)

   e. From the **Data Palette** pane, drag-and-drop the `PODate` **Structural Link** variable over the `$node-var` argument of the `local-name` function in the **If Condition** pane.

   f. Select the operator: =.

   g. In the **Right Hand Expression** section of the **If Condition** pane, enter: "`billing-date`" and click **Add**.

   Note: You must enter double quotes around the "`billing-date`" string.

   The condition is added to the *if* section of the *if-then-else*.

16. Click **Then Expression**.

17. In the **Then Expression** section, replace the existing text with the string: "`BILLING`".

   Replace the following text:

   `data($PODate)`

   With the following string:

   "`BILLING`"

   Note: You must enter double quotes around the "`BILLING`" string.

18. Click **Apply**.

19. Click **Else Expression**.

20. In the **Else Expression** section, enter the string: "`DELIVERY`".

   Note: You must enter double quotes around the "`DELIVERY`" string.

21. Click **Apply**.

22. Select the **General** option, to view the generated `if-then-else` XQuery source code.

The following XQuery code is displayed:

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

During run time, this *if-then-else* statement is executed twice by the XQuery `for` loop. The *if* part of the i*f-then-else* executes the following condition:

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

In the left-hand expression of the *if* condition, the iteration variable `$PODate` is passed into the `local-name` function as a parameter. The `local-name` function returns the name of the XML element. For example: if the `$PODate` iteration variable contains the XML element named `billing-date`, the string: `billing-date` is returned by the `local-name` function and therefore, the left-hand expression of the *if* condition evaluates to the string: `billing-date`.

The right-hand expression of the *if* condition is equal to the string: "`billing-date`".

In the *if* condition, the left-hand expression is compared with the right-hand expression, and if the two strings are equal, the *if* condition evaluates to *true* and the string: "`BILLING`" is returned as the value of the target node: `dates/date/type`.

Note: For simplicity, in this example it is assumed that if `$PODate` does not equal "`billing-date`", it must equal "`delivery-date`".

23. Select the **Test View** tab.

24. Import XML files as source data for the transformation.

For this example, in the **Source Data** pane, select the `$datesDoc` node and import the file: PODateInput.xml. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the
`c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\s`
`ingleToRepeat\XML` directory.

Note: You can cut and past directory paths into the **Name** field of the **Open File to Test** pane to jump to directory locations. If you installed WebLogic Platform in the `c:\bea` directory, you can jump to the directory that contains the XML files for this example, by pasting the following directory path into the **Name** field: `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\trans` `form\singleToRepeat\XML` and then pressing enter.

25. In the **Result Data** pane, click **Test**.

    If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

    In the **Result Data** pane, a graphical representation of the target data is displayed.

26. If the resulting data is XML data, in the **Result Data** pane you can view the resulting data as an XML document by selecting the **XML Source View** tab. The XML result for this example is also displayed in Figure 3-8.

27. If desired, you can validate the result data against the associated schema. In the **Result Data** pane of the **Test View**, click **Validate**.

28. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Creating a Transformation Between a Repeating Source Group and Non-Repeating Target Element

This example shows how to map a XML Schema complex type (for example: a repeating element) to a XML Schema simple type (for example: a non-repeating single element). In this example, you will create a transformation that during run time will take a source repeating element and maps it to single target field using constraints as shown in the following figure.

Figure 3-9 Repeating Source Group to Non-Repeating Target Element
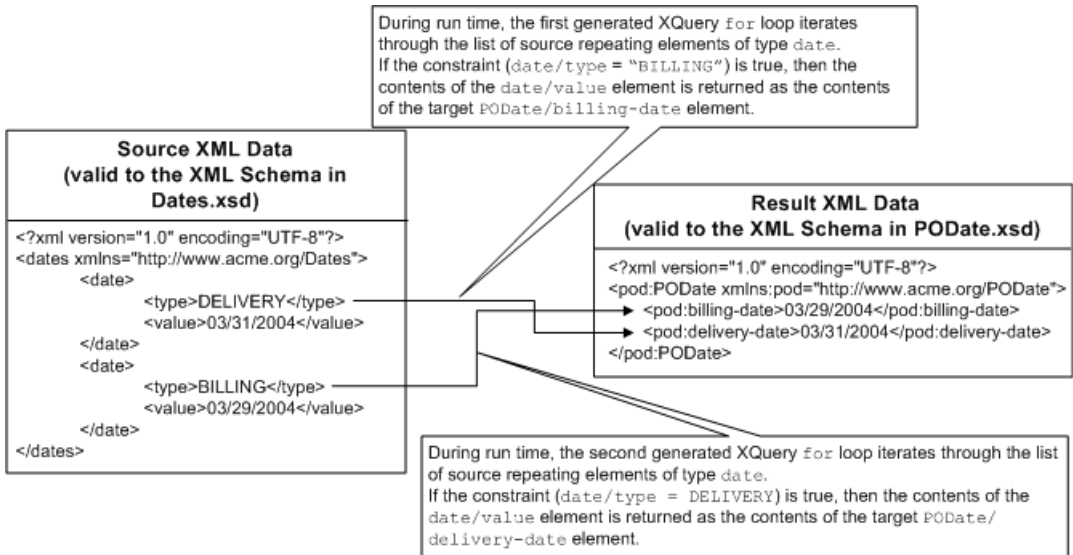
To Create and Run a Transformation Between a Repeating Source Group and a Non-Repeating Target Element

1. Create a Transformation file and a method in the Transformation file. For instructions, see "Creating a Transformation File and a Transformation Method" on page 2-10.

2. Import the XSD file that contains the XML Schema for the source type of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

   For this example, import the file: `Dates.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\repeatToSingle` directory.

   Note: The `Dates.xsd` file used in this example is identical to the `Dates.xsd` file used in "Creating a Transformation Between a Non-Repeating Source Element and Repeating Target Group" on page 3-36. If you have already imported the `Dates.xsd` file for the Creating a Transformation Between a Non-Repeating Source Element and Repeating Target Group example, you do not need to import it again.

3. Import the XSD file that contains the XML Schema for the target type of the transformation. For instructions, see "Selecting Source and Target Types" on page 2-11.

For this example, import the file: `PODate.xsd`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\repeatToSingle` directory.

Note: The `PODate.xsd` file used in this example is identical to the `PODate.xsd` file used in "Creating a Transformation Between a Non-Repeating Source Element and Repeating Target Group" on page 3-36. If you have already imported the `PODate.xsd` file for the Creating a Transformation Between a Non-Repeating Source Element and Repeating Target Group example, you do not need to import it again.

Importing schemas files triggers a build of the current Schemas project folder. Wait until the current Schemas folder is built before proceeding to the next step. (The representations of the schemas will not be available in **Available Source Types** and **Available Target Type** panes until build is complete.)

4. Select the Transformation method from a Transformation file.

   To select an existing method, in the **Design View** of the Transformation file:

   a. Right-click the arrow representing the method.

   b. From the drop-down menu, select **Configure XQuery Transformation Method**.

5. Select the source type for the transformation:

   a. In the **Available Source Types** pane, expand the `Dates.xsd` folder.

   b. In the **Available Source Types** pane, select the `Dates.xsd/dates` element.

   c. Click **Add**.

   The elements and attributes that make up the `Dates.xsd/dates` element are displayed in the **Selected Source Types** pane.

6. Select the target type for the transformation:

   a. In the **Available Target Types** pane, expand the `PODate.xsd` folder.

   b. In the **Available Target Types** pane, select the desired element.

   For this example, select the `PoDate.xsd/PODate` element.

   c. Click **Add**.

   The elements and attributes that make up the `PoDate.xsd/PODate` element are displayed in the **Selected Target Type** pane.

7. Click **Create Transformation**.

   The **Design View** of the XQ file is displayed.

8. In the **Source** pane, select the repeating element (datesDoc/date) and drag it the first single element (PODate/billing-date) in the **Target** pane.

   Keep this link selected for the next step.

   A dashed line linking the two elements is displayed. The dashed line with short dashes represents a structural link—a link that does not map data directly. The dashed-line representation for a structural link is shown in the following figure.

   To learn more about links, see "Link Representations" on page 2-25.

9. In this step, you add a constraint to the structural link created in the preceding step:

   a. From the **Source** pane select the datesDoc/date/type node and drop it into the **Left Hand Expression** pane of the **Where Clause Expression** in the **Constraints** tab.

   b. Select the operator: =.

   c. In the **Right Hand Expression** pane of the **Where Clause Expression** in the **Constraints** tab, enter the following string: "BILLING".

      Warning: You must enter the quotes around the string.

   d. Clicking **Add**.

   During run time, the constraint created in this step, just tests if constraint: datesDoc/date/type is equal to the string: "BILLING". The next step will add the XQuery code to the for loop to return data if during run time the constraint is equal to *true*.

10. In the **Source** pane, select the datesDoc/date/value element and drag it the PODate/billing-date element in the **Target** pane.

    A data link is created.

    During run time, this data link will return the value of datesDoc/date/value as the value of billing-date if the constraint: data($date/ns0:type) = "BILLING" evaluates to *true*.

11. In the **Source** pane, select the repeating element (datesDoc/date) and drag it the second single element (PODate/delivery-date) in the **Target** pane.

A dashed line linking the two elements is displayed.

Keep this link selected for the next step.

12. In this step, you add a constraint to the structural link created in the preceding step:

    a. From the **Source** pane select the `datesDoc/date/type` node and drop it into the **Left Hand Expression** pane of the **Where Clause Expression** in the **Constraints** tab.

    b. Select the operator: =.

    c. In the **Right Hand Expression** pane of the **Where Clause Expression** in the **Constraints** tab, enter the following string: `"DELIVERY"`. The next step will add the XQuery code to return data if during run time the constraint is equal to *true*.

        Warning: You must enter the quotes around the string.

    d. Clicking **Add**.

    During run time, the constraint created in step, tests if `datesDoc/date/type` is equal to the string: `"DELIVERY"`.

13. In the **Source** pane, select the `datesDoc/date/value` element and drag it the `PODate/delivery-date` element in the **Target** pane.

    A data link is created.

    During run time, this data link will return the value of `datesDoc/date/value` as the `delivery-date` if the constraint: `data($date/ns0:type) = "DELIVERY"` evaluates to *true*.

14. Select the **Test View** tab.

15. Import XML files as source data for the transformation.

    For this example, in the **Source Data** pane, select the `$datesDoc` node and import the file: `InputDates.xml`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\repeatToSingle\XML` directory.

    Note: You can cut and past directory paths into the **Name** field of the **Open File to Test** pane to jump to directory locations. If you installed WebLogic Platform in the `c:\bea` directory, you can jump to the directory that contains the XML files for this example, by pasting the following directory path into the **Name** field: `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\repeatToSingle\XML` and then pressing enter.

16. In the **Result Data** pane, click **Test**.

    If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

    In the **Result Data** pane, a graphical representation of the target data is displayed.

17. If the resulting data is XML data, in the **Result Data** pane you can view the resulting data as an XML document by selecting the **XML Source View** tab. The XML result for this example is also displayed in Figure 3-9.

18. If desired, you can validate the result data against the associated schema. In the **Result Data** pane of the **Test View**, click **Validate**.

19. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Using Recursive Schemas in Transformations

This example shows how to create mappings with schemas that have recursive elements. A recursive element contains a child element of the same type as the parent as shown in the following example XML Schema listing:

```
<?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>
```

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) as shown in bold in the preceding code listing.

Figure 3-10 Non-Recursive Source Elements to a Recursive Target Element
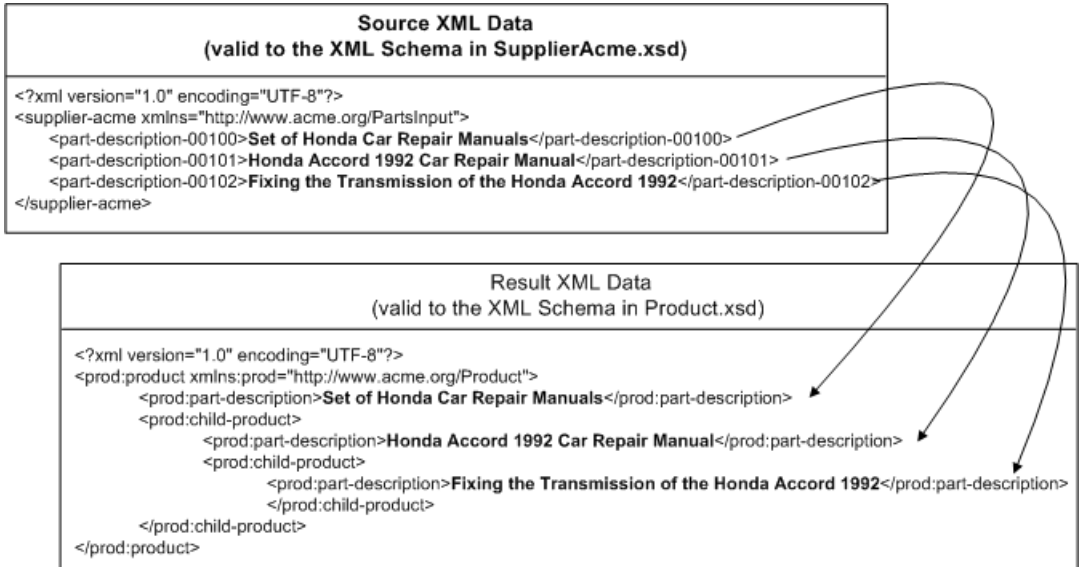
To Create and Run a Transformation Between a Non-Recursive Source Elements and a Recursive Target Element

1. Open a query in the mapper pane. For instructions see "General Steps to Open or Create a Query in the Mapper" on page 2-20.

   For this example, import the files: `SupplierAcme.xsd` and `Product.xsd` files (step 2 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.) If you installed WebLogic Platform in the `c:\bea` directory, import these files from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\recursive` directory.

   For this example, select and add the `SupplierAcme.xsd/supplier-acme` node as an source type (step 4 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.)

   For this example, select and add the `Product.xsd/product` node as an target type (step 4 in "General Steps to Open or Create a Query in the Mapper" on page 2-20.)

2. Select the **Design View**.

   The `product` node is displayed with one level of the recursive `child-product` node expanded as shown in the following figure.
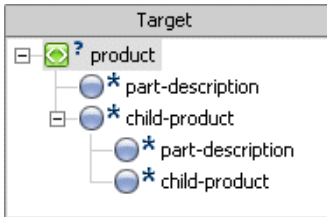
3. In the **Target** pane of the **Design View**, expand the second level child-product node by right-clicking the second-level or bottom child-product node and from the drop-down menu selecting **Expand Recursive Node**.

   Another child node appears under the second-level child-product node as shown in the following figure.
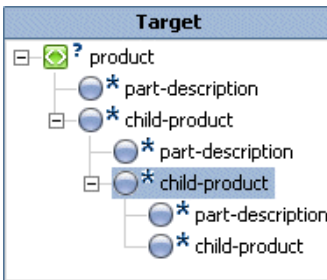
4. In the **Design View**, create a link by dragging-and-dropping the supplier_acmeDoc/part-description-00100 Source element over the top-level product/part-description Target node as shown in the following figure.

5. In the **Design View**, create a link by dragging-and-dropping the supplier_acmeDoc/part-description-00102 Source element over the second level product/part-description Target node as shown in the following figure.

6. In the **Design View**, create a link by dragging-and-dropping the supplier_acmeDoc/part-description-00103 Source element over the third level product/part-description Target node as shown in the following figure.

Source		Target
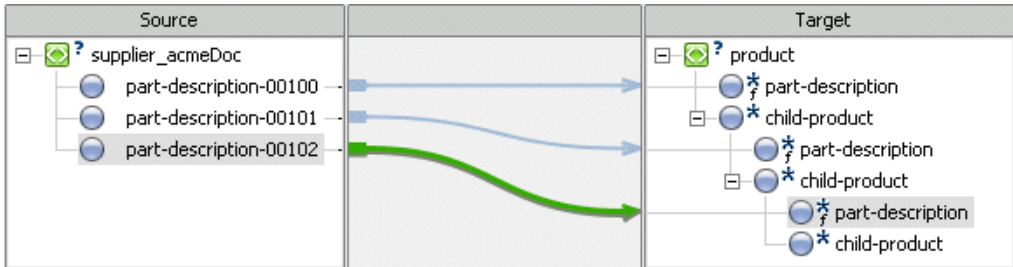
7. Select the **Test View** tab.

8. Import XML files as source data for the transformation.

   For this example, in the **Source Data** pane, select the `$supplier_acmeDoc` node and import the file: `InputRecursive.xml`. If you installed WebLogic Platform in the `c:\bea` directory, import this file from the `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\recursive\XML` directory.

   Note: You can cut and past directory paths into the **Name** field of the **Open File to Test** pane to jump to directory locations. If you installed WebLogic Platform in the `c:\bea` directory, you can jump to the directory that contains the XML files for this example, by pasting the following directory path into the **Name** field: `c:\bea\weblogic81\workshop\help\doc\en\integration\reffiles\transform\recursive\XML` and then pressing enter.

9. In the **Result Data** pane, click **Test**.

   If not currently running, the WebLogic Server for the current application will be started. In order for a query to run, the WebLogic Server for the current application must be running.

   In the **Result Data** pane, a graphical representation of the target data is displayed.

10. If the resulting data is XML data, in the **Result Data** pane you can view the resulting data as an XML document by selecting the **XML Source View** tab. The XML result for this example is also displayed in Figure 3-10.

11. Save the DTF and the XQ file. From the menu bar, choose **File—Save All**.

Transforming Non-XML Data

This section describes design-time and run-time steps required for creating and executing transformations involving non-XML data sources. It also describes how to create schemas (MFL files) for non-XML data using an included WebLogic Integration utility called Format Builder.

This section covers the following topics:

- Using Non-XML Data in Business Processes

- Using Format Builder to Create Format Schemas (MFL Files)

- Importing Existing Metadata to Create Format Schemas (MFL Files)

- Testing the Format Schemas (MFL Files)

Using Non-XML Data in Business Processes

Non-XML data that is sent to or received from legacy applications is often platform-specific information organized in a format unique to the machine on which the information originated. Non-XML data is not self-describing, so to be understood by an application, information about the format (metadata) of this data must be embedded within each application that uses non-XML data from a legacy application.

This section covers the following topics:

- Understanding Transformations That Use Non-XML Data

- Using WebLogic Integration for Transforming Non-XML Data

Understanding Transformations That Use Non-XML Data

Data transformation is the mapping and conversion of data from one format to another. For example, data in a non-XML format can be transformed to an XML format and the converse is also true, data in an XML format can be transformed to an non-XML format. In order to transform data, you must create a schema which contains a description (metadata) for each of the data fields in the non-XML data. During the transformation of data from a non-XML format to an XML format, each field of non-XML data is transformed to XML according to the metadata defined for that field. The metadata you specify must include the name of the field, the data type, the size, and an indication of whether the field is always present or optional. This description of the non-XML data is used to transform the data to XML, as shown in the following figure:
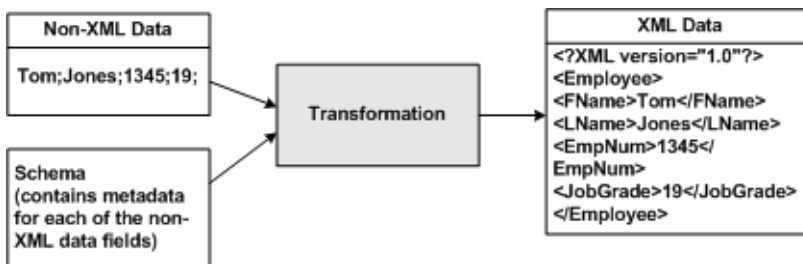
WebLogic Integration can also transform data from XML to non-XML format, as shown in the following figure:
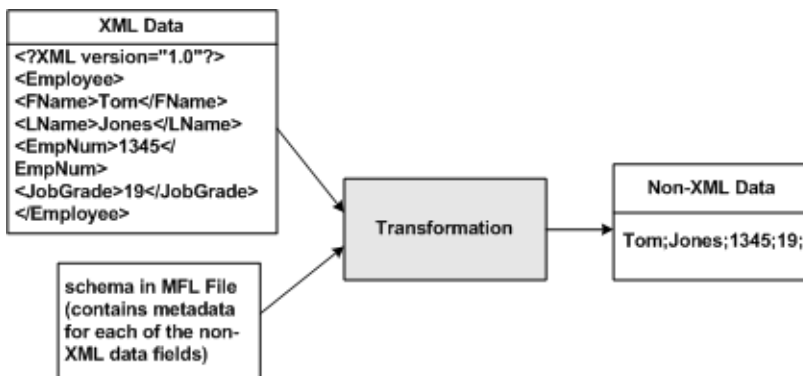
WebLogic Integration can also transform data from one non-XML format to another non-XML format, as shown in the following figure:
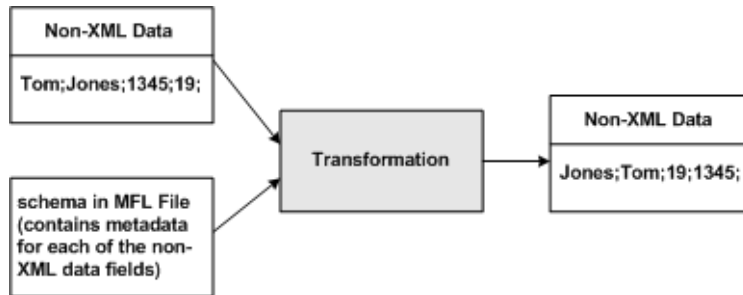
Using WebLogic Integration for Transforming Non-XML Data

WebLogic Integration facilitates the integration of data from diverse enterprise applications by supporting the transformation of non-XML legacy system data to other data types (XML and Java primitives). Once legacy data is available as XML or a Java primitive, it can be used directly by WebLogic Integration business processes. WebLogic Integration supports transformations with non-XML data using the following data integration tools:

- Design-Time Component

  - Format Builder

  - WebLogic Workshop

- Run-Time Component

Steps 1-8 occur at design-time and step 9 occurs during run time, as shown in the following two figures.

The steps for an example non-XML to XML data transformation is shown in the following figure:
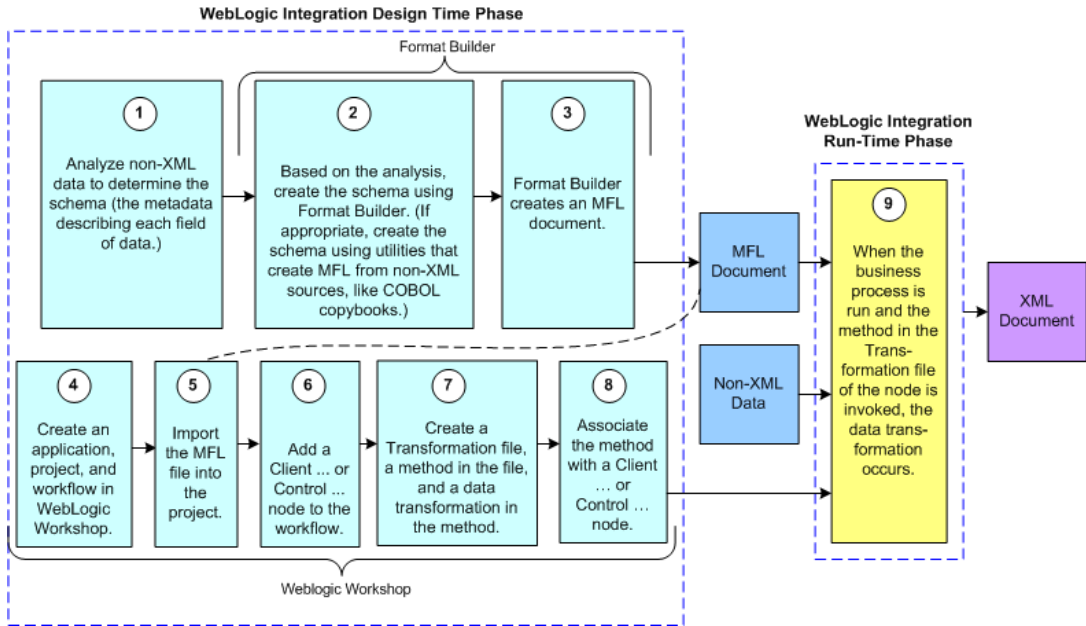
The steps for an example XML to non-XML transformation is shown in the following figure:
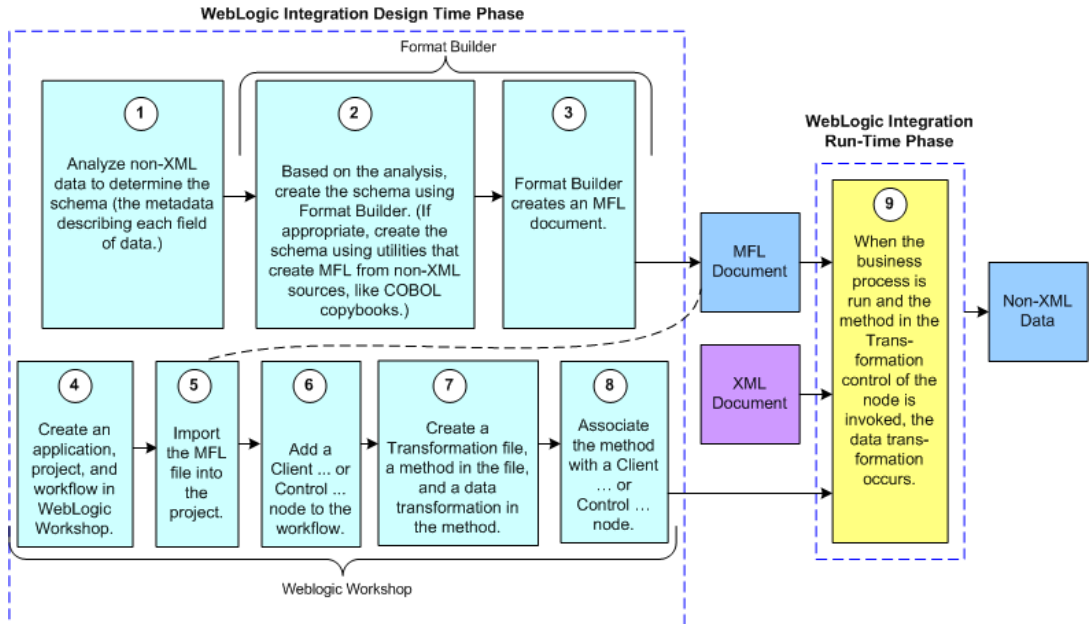
Steps 2-3 are done in the Format Builder tool and steps 4-8 are done in WebLogic Workshop.

Design-Time Component

Format Builder

A design-time component of WebLogic Integration is a Java application called Format Builder. In the first design-time phase (steps 2-3 in preceding figures), you use the Format Builder to create descriptions of non-XML data records. Specifically, you describe the layout and hierarchy of the non-XML data (the schema) in the Format Builder so it can be transformed to or from other data sources, like XML.

You can describe sequences of bytes as fields and specify, for each field, the type of data (floating point, string, and so on), the size of the data, and the name of the field. You can further define sets of fields (groups), multiple instances of fields and groups, and aggregation.

The description you create is saved in an XML grammar called Message Format Language (MFL). MFL documents contain metadata that describes the structure of the non-XML document. Once the non-XML document has been described via an MFL document, it can be

used in XQuery data transformations just like XML documents that have been described by XML Schema (XSD) files.

You can also use Format Builder to retrieve, validate, and edit stored MFL documents and to test message format definitions with your own data. MFL documents are stored in the file system.

The test feature allows you to verify the MFL documents created in Format Builder by transforming a sample XML file to non-XML format, or transforming a sample non-XML file to XML format. You can save the transformed data in a file for future testing.

Note: To learn more about using Format Builder, see "Using Format Builder to Create Format Schemas (MFL Files)" on page 4-7.

WebLogic Workshop

In this second design-time phase, you use WebLogic Workshop to create an application, project, and business process. For instructions on creating applications, projects, and business processes, see Creating a Business Process Application.

In order to map non-XML data in a transformation, you must first import an MFL file which describes the non-XML data into WebLogic Workshop. For instructions, see "Selecting Source and Target Types" on page 2-11.

You also use WebLogic Workshop to create a Transformation file, a method in the Transformation file, and a **Client** or **Control** node in a business process. The method contains a transformation, which when invoked by the business process during run time, maps data types. You design **Control** or **Client** nodes in your business process to call a method in a Transformation file, as shown in steps 6-8 in the preceding figures.

The following table lists the **Client** or **Control** nodes that can be added to a business process:

- **Client Request**

- **Client Response**

- **Control Send**

- **Control Send with Return**

- **Control Receive**

For instructions on add **Client** and **Control** nodes to a business process with Transformations, see Interacting With Clients and Interacting With Resources Using Controls, respectively.

Run-Time Component

If you design a **Client** and **Control** node to call a method in a Transformation file, during run time, the business process invokes the node and then that node invokes the method that contains the transformation. For example, a Client Receive node could receive non-XML data and pass that data to the transformation method, which transforms the non-XML data to XML data, as shown in the following figure:
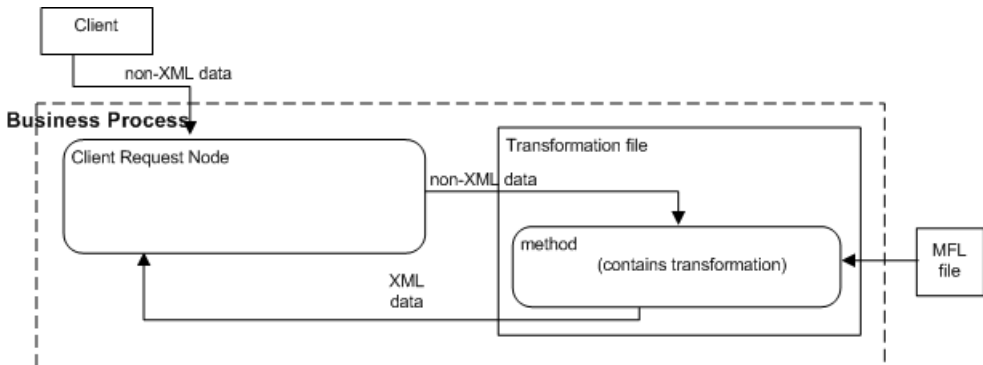
Using Format Builder to Create Format Schemas (MFL Files)

WebLogic Integration uses MFL files to represent the schemas of non-XML documents, just as XSD files are used to represent the schemas of XML documents. At run-time WebLogic Integration uses these MFL files to carry out transformation operations involving non-XML data. This section provides information about creating these MFL files using Format Builder.

It includes the following topics:

- Understanding Data Formats
- Analyzing the Data to Be Transformed
- Using Format Builder

Understanding Data Formats

To understand how to use the Format Builder, it helps to understand the following format and document types:

- Non-XML Data

- XML Documents

- MFL Documents

Non-XML Data

Because computers are based on the binary numbering system, a binary format is often used in applications to represent data. A file stored in binary format can be read by a computer, but not necessarily by a human. Binary formats are used for executable programs and numeric data; text formats are used for pure text. Many files contain a combination of binary and text formats. *Non-XML data* refers to both binary and text formatted data.

Note: The term *non-XML data* replaces the term *Binary data* that was used in previous versions of WebLogic Integration. (WebLogic Integration 7.0 or earlier.)

Unlike XML data, non-XML data is not self-describing. In other words, non-XML data does not include a description of how the data is grouped, divided into fields, or otherwise arranged. Non-XML data is a sequence of bytes that can be interpreted as an integer, a string, or a picture, depending on the intent of the application that generates that sequence.

For example, consider the following non-XML data string:

```
2231987
```

You can interpreted it in many different ways. For example:

- As a date: 2/23/1987

- As a phone number (223-1987)

Without a clear understanding of the purpose of this data string, the application cannot interpret the string appropriately.

In order for non-XML data to be understood by an application, the layout of the data must be embedded in the application itself. The character set used to encode the character data included in a non-XML file may also vary. For example, character data on an IBM mainframe is usually encoded using the EBCDIC character set, while data from a desktop computer is either ASCII or unicode.

You can use Format Builder to create a Message Format Language (MFL) file that describes the layout or schema of your non-XML data. MFL is an XML language that includes elements for describing each field of data, as well as groupings of fields (groups), repetition, and aggregation. The hierarchy of a non-XML record, the layout of fields, and the grouping of fields and groups are expressed in an MFL document. This MFL document is used at run time to transform non-XML data to and from an XML document.

Listing 4-1   Example of Non-XML Data

```
1234;88844321;SUP:21Sprockley's Sprockets01/15/2000123 Main St.;
Austin;TX;75222;555 State St.;Austin;TX;75222;PO12345678;666123;150;
Red Sprocket;
```

XML Documents

The eXtensible Markup Language (XML) is the universal format for structured documents and data. Unlike non-XML data, XML data is self-describing; it makes use of *tags* (words bracketed by '<' and '>') that signal the start and end of each block of data. These tags define the hierarchy of related data components that constitute the *elements* in a structured document.

The properties of XML make it suitable for representing and structuring data in a platform-neutral manner. By making the structure explicit, XML can simplify the task of exchanging data between applications. Because the data is presented in a standard form, applications on disparate systems can interpret it using XML parsing tools, instead of having to interpret data in proprietary binary formats.

The following listing shows an example XML document:

```
<?xml version="1.0"?>
<PurchaseRequest>
 <PR_Number>1234</PR_Number>
 <Supplier_ID>88844321</Supplier_ID>
  <Supplier_Name>Sprockley&apos;s Sprockets</Supplier_Name>
  <Requested_Delivery_Date>2000-01-15T00:00:00:000</Requested_Delivery_Date>
  <Shipping_Address>
    <Address>
      <Street>123 Main St.</Street>
      <City>Austin</City>
      <State>TX</State>
      <Zip>75222</Zip>
    </Address>
  </Shipping_Address>
  </PurchaseRequest>
```

MFL Documents

A Message Format Language (MFL) document (also known simply as a message format document) is a specialized XML document used to describe the layout of non-XML data. When you use Format Builder to define the hierarchy of a non-XML record, the layout of fields, and the grouping of fields and groups, the information is saved as an MFL document that can then be used

to perform run-time transformations. The information captured in the MFL document can also be used to generate a DTD that describes the content model for the output generated by the MFL document.

The top-level element of a message format document is the `MessageFormat` element, which defines the message format name and version. For example, the following is the root element of the sample `po.mfl` document installed with WebLogic Integration:

```
<MessageFormat name='PurchaseRequest' version='2.01'>
```

WebLogic Integration supports Message Format Language Version 2.02. This version supports new features related to padding, truncation, and trimming. Message Format Language Version 2.01 is still supported.

The name assigned to the message format document becomes the root element in the XML instances that are generated based on the MFL document. For example, The following is the root element of any XML document generated based on the sample `po.mfl` document:

```
<PurchaseRequest>
```

The other elements and attributes available in an MFL document are used to define the following:

- Fields and Field Formats – A field is a sequence of bytes that is meaningful in the context of an application and that defines the format of a field. (For example, the field `EMPNAME` contains an employee name.) You can define the following formatting parameters:

  - Tagged – Indicates that a literal precedes the data field, denoting the beginning of the field.

  - Length – Indicates that a numeric value precedes the data field, denoting the length of this field.

  - Occurrence – Indicates the number of times the field is shown in the message format. You can specify the number of times the field is to be shown, or define a delimiter that indicates the end of the repeating field.

  - Optional – Indicates that the field may or may not be included in the format of the named message.

  - Code Page – Identifies the type of character encoding used for the data in the field.

  Note: You must specify unique field names in a single MFL document. To learn more, see "A Note of Caution—Must Specify Unique Field and Group Names in the Same MFL File" on page 4-34.

- Groups and Group Formats – A group is a collection of fields, comments, and other groups or references that are related in some way (for example, the fields PAYDATE, HOURS, and RATE belong to the PAYINFO group). The parameters you can define for a group include:

  - Tagged – Means that a literal precedes the other content of the group, which may be other groups or fields.

  - Occurrence – Indicates either the number of times the group is to be repeated in the message format, or a delimiter that marks the end of the repeated group. For more information about delimiters, see "Specifying Delimiters" on page 4-26.

  - Choice of Children – Indicates that only one item in the group will appear in the message format.

  - Optional – Indicates that the data in this structure may or may not be included in the named message format.

  Note: You must specify unique group names in a single MFL document. To learn more, see "A Note of Caution—Must Specify Unique Field and Group Names in the Same MFL File" on page 4-34.

- References and Reference Formats – A reference indicates that another instance of the field or group format exists in the data. The format of a reference field or group is the same as the format original field or group, but you can change the optional setting and the occurrence setting for the reference field or group. For example, if your data includes a *bill to* address and a *ship to* address and the same format is used for both addresses, you can create the address format once, and then reference it. That is, you can create the an address definition for the *bill to* address and reference it for the *ship to* address.

- Comments – Notes containing additional information about the message format.

Analyzing the Data to Be Transformed

Before a message format can be created, the layout of the non-XML data must be understood. Sample data for a legacy purchase order, with corresponding MFL and XML documents for a purchase order record, are installed with WebLogic Integration. The sample purchase order illustrates how WebLogic Integration transforms data from one format to another.

For more information about this sample data, see Non-XML Data Mapping Sample.

The key to transforming non-XML data to and from XML is to create an accurate description of it. For non-XML data (data that is not self-describing), you must identify the following elements:

- Hierarchical groups

- Group attributes, such as name, optional, repeating, delimited

- Data fields

- Data field attributes, such as name, data type, length/termination, optional, repeating

Use Format Builder to incorporate these elements into the format definitions used for data transformations.

Using Format Builder

Format Builder helps you create format descriptions for non-XML data and store them in MFL documents. Your description should include hierarchical and structural information derived from a detailed analysis of your data. These format descriptions are stored in an MFL document. You can also use Format Builder to test your format descriptions before applying them to your data.

WebLogic Integration also provides utilities that allow you to import COBOL copybooks, import XML Schemas, and convert C structure definitions into MFL files. To learn more about these utilities, see "Importing Existing Metadata to Create Format Schemas (MFL Files)" on page 4-51.

Starting Format Builder

You can launch Format Builder using one of the following options:

- To Start Format Builder From WebLogic Workshop

- To Start Format Builder on Windows Without Launching WebLogic Workshop

- To Start Format Builder on Linux Without Launching WebLogic Workshop

To Start Format Builder From WebLogic Workshop

1. Start WebLogic Workshop: choose **Start→Programs→BEA WebLogic Platform 8.1→WebLogic Workshop 8.1**.

2. The main **WebLogic Workshop** window is displayed.

3. From the WebLogic Workshop menu bar, choose **Tools→WebLogic Integration→Format Builder**.

   The Format Builder main window is displayed.

To Start Format Builder on Windows Without Launching WebLogic Workshop

Choose **Start→Programs→BEA WebLogic Platform 8.1→Other Development Tools→Format Builder**.

The Format Builder main window is displayed.

To Start Format Builder on Linux Without Launching WebLogic Workshop

1. In command line shell, go to the WebLogic Integration bin directory. For example, if WebLogic Platform is installed in the `/usr2/bea` directory, go to the `/usr2/bea/weblogic81/integration/bin` directory as shown here:

    ```
    cd /usr2/bea/weblogic81/integration/bin
    ```

2. Run the **Format Builder** start script, as shown here:

    ```
    ./fb.sh
    ```

    The Format Builder main window is displayed.

Using the Format Builder Window

The Format Builder window is split into two vertical panes. The left pane contains the navigation tree which shows the structural relationship of the groups and fields defined in the active MFL document. The right pane displays the properties that define the item.

Information about the file you are editing is displayed in the title bar of the Format Builder window.
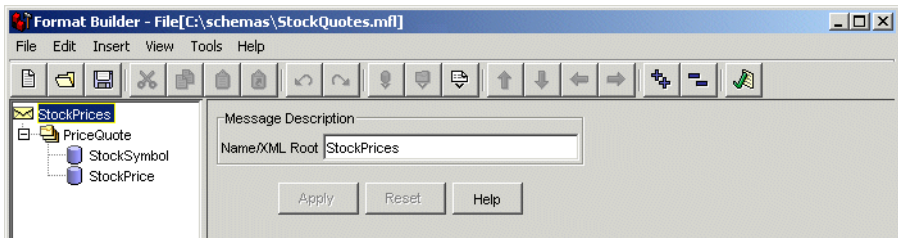
The structure of the non-XML data is defined in the navigation tree through a combination of fields and groups that match the target data.

The following topics explain how to use the various tools provided in the Format Builder window to navigate and execute commands:

- Using the Navigation Tree

- Using the Format Builder Menu Bar

- Using the Toolbar

- Using Drag and Drop

- Using the Shortcut Menus

Note: For additional information about Format Builder, see the help included with the Format Builder executable. (To access the Format Builder help, start the Format Builder as described in "Starting Format Builder" on page 4-12 and then from the **Format Builder** menu bar, choose **Help—Help Topics**.)

Using the Navigation Tree

The navigation tree represents the structure of the non-XML data in a hierarchical layout. The root node of the navigation tree, the *Message node*, corresponds to the MFL document being created or edited. Child nodes are labeled with the names of groups or fields. Fields are represented by leaf nodes in the navigation tree. Groups contain fields or other groups and are represented by non-leaf nodes in the navigation tree.

The icon for each node encapsulates the following information about the node: whether the node represents a message, a group, a field, a comment, or a reference; whether a group or field is repeating; whether a group is a *Choice of Children;* and whether a group or field is optional or mandatory.

You can add, delete, move, copy, or rename nodes in the navigation tree though menus or the toolbar. (For details, see "Using the Format Builder Menu Bar" on page 4-16 and "Using the Toolbar" on page 4-17.)

The following table describes the icons displayed in the navigation tree.

Table 4-1 Navigation Tree Icons

Tree Icon	Icon Name	Description
	Message Format	The top-level element.
	Group	Collections of fields, comments, and other groups or references that are related in some way. (For example, the fields PAYDATE, HOURS, and RATE belong to the PAYINFO group.) Defines the formatting for all items in the group.
	Optional Group	A group that may or may not be included in the message format.
	Repeating Group	A group that is included one or more times.
	Optional Repeating Group	A group that may or may not be included, but if included, may occur more than once.
	Group Reference	Indicates the existence of another instance of the group in the data. The format of a reference group is the same as that of the original group, but you can change the optional setting and the occurrence setting for the reference group.
	Group Choice	Indicates that only one of the items in the group is included in the message format.
	Field	Sequence of bytes that is meaningful in the context of the application and that defines the formatting for the field. (For example, the field EMPNAME contains an employee name.)
	Optional Field	A field that may or may not be included in the message format.
	Repeating Field	A field is included one or more times.

Table 4-1 Navigation Tree Icons (Continued)

Tree Icon	Icon Name	Description
	Optional Repeating Field	A field that may or may not be included, but, if included, may occur more than once in the message format.
	Field Reference	Indicates the existence of another instance of the field in the data. The format of a reference field is the same as that of the original field, but you can change the optional setting and the occurrence setting for the reference field.
	Comment	Contains notes about the message format or the data transformed by the message format.
	Collapse	A minus sign next to an item indicates that the specified item can be collapsed.
	Expand	A plus sign next an item indicates that the specified item can be expanded to show child items.

Using the Format Builder Menu Bar

The menu bar provides quick access to Format Builder functions.

File   Edit   Insert   View   Tools   Help

The items available in a menu depend on the actions you have taken and the node currently selected in the navigation tree. If a menu item is not available, it is shown in gray in the menu.

You can display a menu in either of two ways:

- Click the name of the menu in the menu bar.

- On your keyboard, press Alt + *key*, where *key* is the first letter in the menu name. For example, press Alt + F to select the File menu option.

To execute a command, select it from the menu. Some commands can also be executed via the keyboard shortcut indicated on the menu (For example, a Ctrl + *key* sequence.) The commands available on each menu are described in "Format Builder Menus" on page 4-47.

Using the Toolbar

The toolbar is a menu of icons that provide alternative ways to access frequently used commands.

To execute a command, click the appropriate icon in the toolbar. If a command is unavailable, the icon for it appears *grayed-out*.

The following table describes the icons in the Format Builder tool bar.

Table 4-2 Format Builder Toolbar Icons

Toolbar Icon	Name	Description
	New	Creates a new message format.
	Open	Opens an existing message format.
	Save	Saves the current message format.
	Cut	Removes the item currently selected in the left pane, and its child objects, from the navigation tree. The item can be pasted elsewhere in the navigation tree.
		Note: This action is not available if the message format (root) item is selected.
	Copy	Makes a copy of the item currently selected in the left pane for insertion elsewhere in the navigation tree.
		Note: This action is not available if the message format (root) item is selected.
	Paste as Sibling	Inserts the cut or copied item as a sibling object of the selected item.

Table 4-2 Format Builder Toolbar Icons (Continued)

Toolbar Icon	Name	Description
	Paste as Reference	Inserts a reference to the cut or copied item as a sibling object of the selected item.
	Undo	Reverses the previous action. The tool tip indicates the action that can be undone. For example, if you change the name of a field to Address and click Apply, the tool tip displays the following message: Undo Apply Field Address. Format Builder supports multiple undoing of previous actions.
	Redo	Reverses the effects of an Undo command. The tool tip indicates the action that can be redone. For example, if you change the name of a field to Address and then Undo that change, the Redo tool tip displays the following message: Redo Apply Field Address. Format Builder supports multiple redoing of previous actions.
	Insert Field	Inserts a field as a sibling of the item selected in the navigation tree.
	Insert Group	Inserts a group as a sibling of the item selected in the navigation tree.
	Insert Comment	Inserts a comment as a sibling of the item selected in the navigation tree.
	Move Up	Moves the selected item up one position under its parent.
	Move Down	Moves the selected item down one position under its parent.

Table 4-2 Format Builder Toolbar Icons (Continued)

Toolbar Icon	Name	Description
	Promote item	Assigns the selected item to the next highest level in the navigation tree. For example, suppose Field1 is a child object of Group1. If you select Field1 and click the Promote tool, you make Field1 a sibling of Group1.
	Demote item	Assigns the selected item to the next lower level in the navigation tree. For example, suppose Group1 is the sibling of Field1 and it is listed immediately after Group1 in the navigation tree. If you select Field1 and click the Demote tool, you make Field11 a child of Group1.
	Expand All	Expands all the items in the navigation tree to show child items.
	Collapse All	Collapses the navigation tree to show first-level items only.
	Format Tester	Opens the Format Tester window.

Using the Shortcut Menus

When you right-click an item in the navigation tree, a menu of the most frequently used commands for that item is displayed. The following table describes the commands that are available from the shortcut menus.

Note: The availability of a command depends on the item you select and the previous actions you have taken.

Command	Description
Cut	Removes the item currently selected in the left pane, and its child objects, from the navigation tree.
Copy	Makes a copy of the item currently selected in the left pane for insertion elsewhere in the navigation tree.

Command	Description
Paste	Inserts the cut or copied item. An additional menu is displayed when you select Paste. You can paste the item as either a child or a sibling of the selected item. In addition, you can paste a reference to the cut or copied item as a sibling of the selected item.
Insert Group	Inserts a new group as either a child or a sibling of the selected item, depending on your specification.
Insert Field	Inserts a new field as either a child or a sibling of the selected item, depending on your specification.
Insert Comment	Inserts a comment as either a child or a sibling of the selected item, depending on your specification.
Duplicate	Makes a copy of the currently selected item and pastes it as a sibling. The duplicate item contains the same values and child objects as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original item is MyGroup1, then the name of the duplicate is NewMyGroup1.
Delete	Deletes the selected item.

Using Drag and Drop

You can drag and drop to copy and paste, or move items in the navigation tree.

Note: The node being copied or moved is always inserted as a sibling of the selected node during the drag-and-drop process. If you drag and drop the node onto the Message Format node, it is inserted as the last child.

To move an item:

1. Select the item you want to move.

2. Press and hold the left mouse button while you drag the item to the desired node.

3. When the item is in the desired location, release the left mouse button. The item is moved to the new location.

To copy and paste an item:

1. Select the item you want to copy.

2. Press and hold the Ctrl key.

3. Keeping the Ctrl key depressed, press and hold the left mouse button while you drag the item to the desired node.

4. With the sibling object selected, release the left mouse button. A copy of the item is pasted the new location.

Creating Message Formats

The first step in creating a message format definition file is to create a message format (the root node of a message format file).

To create a message format:

1. Choose **File—New**. The detail window for the message format is displayed the right pane.
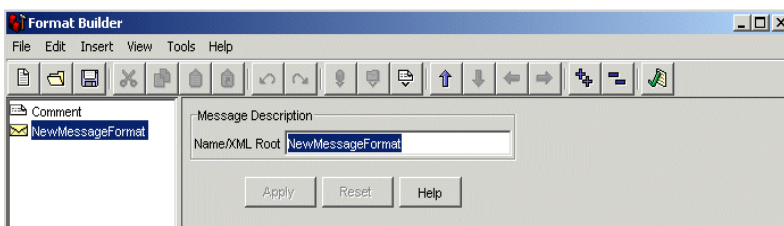
2. Enter the name of the message format in the **Name/XML root** field.

   Note: The entry in the **Name/XML Root** field becomes the name of the root element of each XML instance generated based on this message format document. Therefore, the entry must comply with the conventions described in the following section, "XML Element Naming Conventions" on page 4-21.

3. Click one of the following:

   – **Apply**—updates the message format properties.

   – **Reset**—discards your changes to the detail window and resets all fields to the values that were last applied.

   – **Help**—displays online help information for the message format detail window.

   Note: The **Apply** and **Reset** options are enabled only after changes are made in the detail window.

XML Element Naming Conventions

The names you assign to the root node, fields, groups, and references in a message format document are transformed to XML element names in the XML instances generated based on the

message format document. Therefore, the names must comply with the following XML naming rules:

- A name must start with a letter or underscore.

- A name can contain letters, digits, periods, hyphens, or underscores.

The following strings are examples of valid names:

- `MyField`

- `MyField1`

- `MyField_again`

- `MyField-again`

The following strings are examples of invalid names:

- `1MyField` (starts with a digit)

- `My>Field` (includes a greater-than sign (>), which is an illegal character)

- `My Field` (includes a space, which is not permitted)

- `My/Field` (includes the back slash (/), which is an illegal character)

- `My\Field` (includes the forward slash (\), which is an illegal character)

- `My:Field` (includes a colon (:), which is an illegal character)

- `My;Field` (includes a semi-colon (;), which is an illegal character)

Creating Groups

A group is a collection of fields, comments, references, and other groups that are related in some way. For example, the fields PAYDATE, HOURS, and RATE might all belong to the PAYINFO group. You can create a group as a child of the message format item, as a child of another group, or as a sibling of a group or field.

> **Note:** You must specify unique field names in a single MFL document. To learn more, see .

To create a group:

1. Select the an item in the navigation tree.

2. Choose one of the following:

– If the selected item is the root node, or another group, and you want to create the group as the child of the selected item, choose **Insert—Group—As Child**.

– If you want to create the group as a sibling the selected item, choose **Insert—Group—As Sibling**.

The detail window for the group is displayed the right pane.

3. Define the properties for the group as described in the following table:

Table 4-3 Group Properties

Category	Property	Description
Group Description	Name	The name of the group. The entry must comply with the conventions described in "XML Element Naming Conventions" on page 4-21.
	Optional	Select Optional if the group is optional.
	Choice of Children	Select Choice of Children if only one of the items in the group will be included in the message format.

Table 4-3 Group Properties (Continued)

Category	Property	Description
Group Occurrence (Unless defined as Optional in the Group Description, all groups occur at least once.)	Once	Select this option to indicate that the group appears only once.
	Repeat Delimiter	Select this option to indicate that the group will repeat until the specified delimiter is encountered.
	Repeat Field	Select this option to indicate that the group will repeat the number of times specified in the field selected as the repeat field.
	Repeat Number	Select this option to indicate that the group will repeat the specified number of times.
	Unlimited	Select this option to indicate that the group will repeat an unlimited number of times.

Table 4-3 Group Properties (Continued)

Category	Property	Description
Group Attributes	Group is Tagged	Select this option if the group is tagged, that is, if a literal precedes the other content of the group, which may be other groups or fields.
	Group Delimiter	The termination point of a group can be specified by a *delimiter*: a string of characters that marks the end of a group of fields. The group continues until delimiter characters are encountered.
		Note: Normally, groups are not delimited. They are usually parsed by content; the group ends when all child objects have been parsed. For more information about delimiters, see "Specifying Delimiters" on page 4-26.
		Select from among the following options to specify the group delimiter attributes:
		None — Select this option if there is no delimiter for the group.
		Delimited — Select this option if the termination point of the group is marked with a delimiter character string, then enter the delimiter characters in the Value field.
		Delimiter Field — Select this option if the termination point of the group is marked by a field that contains a delimiter character string. When you select this option, you are prompted to provide the following:
		Field—select the field that contains the delimiter character string. A list of valid fields is presented in a drop-down list.
		Default—enter the default delimiter character used if the selected field is not included in the data. This value is required.
		Delimiter is Shared — Select this option to indicate that the delimiter marks both the end of the group of data, and the end of the last field of the group. The delimiter is shared by the group, and the last field of the group, to indicate the end of the data.

4. Click one of the following:

  – **Apply**—updates the group properties.

  – **Duplicate**—makes a copy of the group currently displayed and pastes it as a sibling.

    The duplicate group contains the same values and child objects as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original group is MyGroup1, then the name of the duplicate is NewMyGroup1.

  – **Reset**—discards your changes to the detail window and resets all fields to the values that were last applied.

  – **Help**—displays online help information for the detail window.

  Note: The **Apply** and **Reset** options are enabled only after changes are made in the detail window.

Specifying Delimiters

You can specify delimiters in Format Builder by entering the correct syntax. For example, if you want to specify a tab character as a delimiter ('\u009'), you must enter the construct `\t` to match it.

The following tables maps characters you can use as delimiters to the constructs you must use to designate these characters as delimiters.

Table 4-4 Character Delimiters

Use this construct . . .	To designate the following character as a delimiter . . .
x	x
\\	\ (backlash)
\0n	Character with octal value 0n (<= n <= 7)
\0nn	Character with octal value 0nn (0 <= n <= 7)
\0mnn	Character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)
\xhh	Character with hexadecimal value 0xhh
\uhhhh	Character with hexadecimal value 0xhhhh

Table 4-4 Character Delimiters (Continued)

Use this construct . . .	To designate the following character as a delimiter . . .
\t	Tab character ('\u0009')
\n	Newline (line feed) character ('\u000A')
\r	Carriage-return character ('\u000D')
\f	Form-feed character ('\u000C')
\a	Alert (bell) character ('\u0007')
\e	Escape character ('\u001B')
\cx	Control character corresponding to x

To learn more, see the java.util.regex.Pattern class description.

Creating Fields

A field is a sequence of bytes that is meaningful to an application. (For example, the field EMPNAME contains an employee name.) You can create a field as a child of the message format node, as a child of a group, or as a sibling of a group or another field. Field names are used as element names in the XML output; they must comply with the conventions described in "XML Element Naming Conventions" on page 4-21.

To create a field:

1. Select an item in the navigation tree.

2. Choose one of the following:

   – If you want to create the field as the child of the selected item, choose **Insert→Field→As Child**.

   – If you want to create the field as the sibling of the selected item choose **Insert→Field→As Sibling**.

   The detail window for the field is displayed the right pane.

3. Define the properties for the field as described in the following table:

Table 4-5 Field Properties

Category	Property	Description
Field Description	Name	The name of the field. The entry must comply with the conventions described in "XML Element Naming Conventions" on page 4-21 and "A Note of Caution—Must Specify Unique Field and Group Names in the Same MFL File" on page 4-34.
	Optional	Select this option if this is an optional field. Optional means that the data for the field may or may not be present.
		If the Optional option is selected for a file, then you must also set the **Field is Tagged** option from the **Field Attributes** pane. In addition, in the **Field Is Tagged** text box enter a unique value for each optional field in a group. Multiple groups can use the same tag value but the tag value for each optional field in a group must be unique.
	Type	Select the data type of the field from the drop-down list. The default is String.
		Note: Which field type you select dictates which field data options are displayed.
		For a list of data types supported by WebLogic Integration, see the online help of Format Builder.
Field Occurrence (Unless defined as Optional in the Field Description, all fields occur at least once.)	Once	Select this option to indicate that the field appears only once.
	Repeat Delimiter	Select this option to indicate that the field will repeat until the specified delimiter is encountered.
	Repeat Field	Select this option to indicate that the field will repeat the number of times specified in the field selected as the repeat field.
	Repeat Number	Select this option to indicate that the field will repeat the specified number of times.
	Unlimited	Select this option to indicate that the field will repeat an unlimited number of times.

Table 4-5 Field Properties (Continued)

Category	Property	Description
Field Attributes (The Field Attributes properties that are displayed are dependent on the Type specified in the Field Description)	Field is Tagged	Select this option if the field is tagged, that is, if a literal proceeds the data, indicating that the data is present. You must also choose the data type of the tag field from the drop-down list. For example in the following: `SUP:ACME INC` `SUP:` is a tag and `ACME INC` is the field data. If you select the **Field is Tagged** option, enter the tag in the field to the right of the check box.
	Field Default Value	Select this option to specify a value for the data in field that is inserted into the non-XML data if the field is not included in the XML. If the field is not included in the non-XML data and it is not optional, then the non-XML data fails to parse, even if a default value is given.
	Data Base Type	If the field is a date or time field, the base type indicates the type of characters (ASCII, EBCDIC, or Numeric) used to represent the data.
	Year Cutoff	If the field is a date field with a 2-digit year, the year cutoff attribute allows the 2-digit year to be converted to a 4-digit year. If the 2-digit year is greater than or equal to the year cutoff value, a prefix of 19 is added to the year value. Otherwise a prefix of 20 is used.
	Code Page	The character encoding of the field data. The default code page is set by choosing Tools—Options and selecting the default encoding from the Default Field Code Page drop down list.
	Value	The value displayed in a literal field.

Table 4-5 Field Properties (Continued)

Category	Property	Description	
Field Attributes (Continued)	Termination	Select from among the following options to specify the group delimiter attributes:	
		Length	Select this option to set the length of variable-sized data types to a fixed value. When you select this option, you are prompted to provide the following:
			Length—enter the number of bytes in the field.
			Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data.
			Pad—if the XML data is shorter than the specified length, appends the specified data to correct its length. Select one of the following padding options:
			• Select the Trailing option to append padding at the end of a field.
			• Select the Leading option to append padding at the beginning of a field.
			Truncate—remove a specified number of characters from a field. Select any combination of the following truncation options:
			• Select the Truncate First option to remove the specified number of characters from the beginning of the field.
			• Select the Truncate After option to remove the specified number of characters from the end of the field.
			If you select both truncation options, the Truncate First option is implemented initially, and the Truncate After option is invoked on the remaining characters.

Table 4-5 Field Properties (Continued)

Category	Property	Description	
Field Attributes (Continued)	Termination (Continued)	Embedded Length	Select this option to indicate that the termination point of a variable-sized data type is specified by an embedded length. An embedded length precedes the data field and indicates the number of bytes in the data. When you select this option, you are prompted to provide the following: • Type—specifies the data type and, if necessary, the length or delimiter for termination. • Tag/Length Order—specifies the order of the tag and length fields when both are included. The default order is: tag, length. • Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data. • Truncate—remove a specified number of characters from a field. For more information, see the description of the Truncate option for the Length option.
		Delimiter	Select this option to indicate that the termination point of a variable-sized data type is specified by a *delimiter*: a value that marks the end of the field. The field data continues until the delimiter is encountered. When you select this option, you are prompted to provide the following: • Value—enter the delimiter that marks the end of the field data. • Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data. • Truncate—remove a specified number of characters from a field. For more information, see the description of the Truncate option for the Length option.

Table 4-5 Field Properties (Continued)

Category	Property	Description	
Field Attributes (Continued)	Termination (Continued)	Delimiter Field	Select this option if the termination point of a variable-sized data type is specified by a field that contains a delimiter value. When you select this option, you are prompted to provide the following:
			• Field—select the field that contains the delimiter.
			• Default—enter a default delimiter that can be used when the delimiter field is not present. You must enter a value in this field.
			• Trim Leading/Trailing—removes the specified data from the leading or trailing edge of the data.
			• Truncate—remove a specified number of characters from a field. For more information, see the description of the Truncate option for the Length option.
			For more information about delimiters, see "Specifying Delimiters" on page 4-26.
		Decimal Position	Specifies the number of digits (0-16) to the left of the decimal point.

4. Click one of the following:

   – **Apply**—updates the field properties.

   – **Duplicate**—makes a copy of the field currently displayed and pastes it as a sibling.

     The duplicate field contains the same values as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original field is MyField1, then the name of the duplicate is NewMyField1.

   – **Reset**—discards your changes to the detail window and resets all fields to the values that were last applied.

   – **Help**—displays online help information for the field detail window.

   Note: The **Apply** and **Reset** options are enabled only after changes are made in the detail window.

Padding Mandatory Fields

Prior to the WebLogic Integration 7.0 release, no padding was performed on mandatory fields when data for the field did not exist at run time. For the WebLogic Integration 7.0 release and all subsequent releases, during an XML to non-XML transformation, a mandatory field that does not contain data is padded with the default value, if a default value has been specified. If no default value is specified and a field does not contain data at transformation time, an error occurs.

Note: Padding of mandatory fields is not supported for non-XML to XML transformations.

This feature is useful when a group is specified multiple times, but data is provided for only one occurrence. When padding of mandatory fields is invoked, all occurrences of a group for which data are not provided are padded with default values, if specified.

A Note of Caution—Must Specify Unique Field and Group Names in the Same MFL File

You must specify unique field (`FieldFormat`) and group (`StructFormat`) format names in a single MFL file. Format Builder allows the creation of duplicate field and group format names in the same MFL file but when an MFL file with duplicate names is imported into WebLogic Workshop causing the schemas project is built, errors will be reported, as shown in the following examples.

If the following example MFL file is imported into WebLogic Workshop, the build of the schema project fails because the field name: `StockSymbol` is used in two different groups:

```
<?xml version='1.0' encoding='windows-1252'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name='StockPrices' version='2.01'>
    <StructFormat name='PriceQuoteOne' repeat='*'>
        <FieldFormat name='StockSymbol' type='String' delim=':'
codepage='windows-1252'/>
        <FieldFormat name='StockPrice' type='String' delim='|'
codepage='windows-1252'/>
    </StructFormat>
    <StructFormat name='PriceQuoteTwo' repeat='*'>
        <FieldFormat name='StockSymbol' type='String' delim=':'
codepage='windows-1252'/>
        <FieldFormat name='StockPrice' type='String' delim='|'
codepage='windows-1252'/>
    </StructFormat>
</MessageFormat>
```

The following schema build error is reported:

```
ERROR: Error compiling MFL file
-C:\bea\weblogic81\DTGuide\Schemas\StockQuotesSameStructFormat.mfl:
FieldFormat already defined: StockSymbol
```

If the following example MFL file is imported into WebLogic Workshop, the build of the schema
project fails because the group name: `PriceQuote` is used twice, even though the first
`PriceQuote` is nested in the group: `Level`:

```
<?xml version='1.0' encoding='windows-1252'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name='StockPrices' version='2.01'>
    <StructFormat name='Level' repeat='*'>
        <StructFormat name='PriceQuote' repeat='*'>
            <FieldFormat name='StockSymbol' type='String' delim=':'
codepage='windows-1252'/>
            <FieldFormat name='StockPrice' type='String' delim='|'
codepage='windows-1252'/>
        </StructFormat>
    </StructFormat>
    <StructFormat name='PriceQuote' repeat='*'>
        <FieldFormat name='StockSymbol' type='String' delim=':'
codepage='windows-1252'/>
        <FieldFormat name='StockPrice' type='String' delim='|'
codepage='windows-1252'/>
    </StructFormat>
</MessageFormat>
```

The following schema build error is reported:

```
ERROR: Error compiling MFL file -
C:\bea\weblogic81\DTGuide\Schemas\StockQuotesSameNameInDiffGroup.mfl:
 StructFormat already defined: PriceQuote
```

Note: You can, however, use the same field and group names in different MFL files because
the file name of the MFL document becomes the namespace for the field or group format names
in WebLogic Workshop, making the field and group format names unique from file to file. For
example, the `StockPrice` field in the `Stock.mfl` file is prefixed with the namespace: `Stocks`
making it unique from the `StockPrice` field in the `Price.mfl` file which is prefixed with
the namespace: `Price`.

Creating Comments

Comments are notes about the message format or the data transformed by the message format.
Comments are included in the message format definition for documentation and informational
purposes only; they are unnumbered and are not transformed to XML or non-XML data. You can
create a comment as a child or sibling of any message format, group, or field.

Note: Conventionally, a comment precedes the node it annotates.

To create a comment:

1. Select an item in the navigation tree.

2. Choose one of the following:

   – If you want to create the comment as the child of the selected item, choose **Insert—Comment—As Child**.

   – If you want to create the comment as the sibling of the selected item, choose **Insert—Comment—As Sibling**.

3. Enter the comment text in the **Comment Details** field.

```
┌─Comment Details─────────────────────────────────┐
│ ┌─────────────────────────────────────────────┐ │
│ │XYZ Corporation legacy Purchase Request Form  │ │
│ │                                             │ │
│ │                                             │ │
│ │                                             │ │
│ └─────────────────────────────────────────────┘ │
│        ┌─────────┐ ┌─────────┐ ┌─────────┐       │
│        │  Apply  │ │  Reset  │ │  Help   │       │
│        └─────────┘ └─────────┘ └─────────┘       │
└─────────────────────────────────────────────────┘
```

4. Click one of the following:

   – Apply—updates the comment text.

   – Reset—discards your changes to the detail window and resets all fields to the values that were last applied.

   – Help—displays online help information for the comment detail window.

   Note: The Apply and Reset options are enabled only after changes are made in the detail window.

Creating References

References allow you to reuse an existing field or group format in a new context. When you create a reference to an existing field or group, the same format is used, but you can modify the optional and occurrence properties for the reference field or group.

For example, if your data includes a *bill to* address and a *ship to* address and the same format is used for both addresses, you can create the address format once, and then reference it. That is,

you can create the an address definition for the *bill to* address and reference it for the *ship to* address.

Note: A reference item is given exactly the same name as the original item, therefore, you should use a generic name, such as `address`, when you create a field or group that is be referenced. For instance, in the previous example, you can create an `address` group as a child of the `bill_to` group and then reference the `address` group from within the `ship_to` group.

To create a reference:

1. Select the item to be referenced in the navigation tree.

2. Choose **Edit→Copy**.

3. Select an item at the desired location for the reference. When you paste the item as a reference in the next step, the reference is pasted as a sibling of the selected item.

4. Choose **Edit→Paste→As Reference**.

   The detail window for the reference is displayed. For example, the following figure shows the detail window for a Field Reference:

5. Define the properties for the reference as described in the following table:

Table 4-6 Reference Properties

Category	Property	Description
Field or Group Reference Description	Name	The name of the field or group for which you created this reference. This value cannot be changed.
	Optional	Select this option if the reference is optional.

Table 4-6 Reference Properties

Category	Property	Description
Field or Group Reference Occurrence (Unless defined as Optional all referenced items occur at least once.)	Once	Select this option to indicate that the referenced item appears only once.
	Repeat Delimiter	Select this option to indicate that the referenced item will repeat until the specified delimiter is encountered.
	Repeat Field	Select this option to indicate that the referenced item will repeat the number of times specified in the field selected as the repeat field.
	Repeat Number	Select this option to indicate that the referenced item will repeat the specified number of times.
	Unlimited	Select this option to indicate that the referenced item will repeat an unlimited number of times.

6. Click one of the following:

   – Apply—updates the reference properties.

   – Reset—discards your changes to the detail window and resets all fields to the values that were last applied.

   – Edit Reference—displays the detail window for the original item to allow you to edit the item.

   – Help—displays online help information for the reference detail window.

   Note: The Apply and Reset options are enabled only after changes are made in the detail window.

Working with the Palette

The Format Builder palette allows you to store commonly used message format components so they are available whenever you need to insert them into your message format definitions.

The default palette, `palette.xml`, is an MFL document which is stored in the WebLogic Integration installation directory. The default palette contains common date formats, literals, and strings. You can use these items in the message formats you create, as well as add your own items to the default palette. You can also create your own MFL documents for use in the palette, or open and use items from any existing MFL document.

The following topics provide the information you need to use the palette:

- Opening the Palette

- Using the Palette File Menu

- Using the Palette Shortcut Menu

- Copying Items From the Active Message Format to the Palette

- Deleting Items From the Palette

- Copying Palette Items from the Palette to the Active Message Format

Opening the Palette

To open the palette:

1. Start Format Builder.

2. Choose **View—Show Palette**.

   The Palette window displays the default palette.
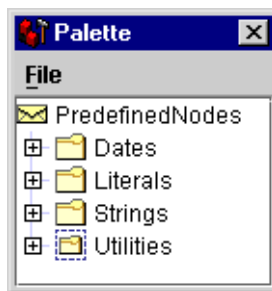
You can copy items from the navigation tree to the palette, and vice versa. You can use drag and drop, or the commands available on the shortcut menu, to organize items in the palette. The contents of the palette are automatically saved when you exit Format Builder.

Note: Only copying items, whether from the navigation tree to the palette or vice versa, is allowed. You cannot move items between the windows.

Using the Palette File Menu

The commands described in the following table are available from the Palette File menu.

Table 4-7 Palette File Menu Commands

Command	Description
Open	Displays the Open dialog box to allow you to select and open an existing MFL document in the palette.
Save	Saves any changes you have made to the MFL document currently open in the palette.
Hide Palette	Closes the Palette window.

Using the Palette Shortcut Menu

A shortcut menu is displayed when you right-click an item or folder in the palette. The following table describes the commands available from the shortcut menu.

Note: Some commands may be unavailable, depending on the item you select.

Table 4-8 Palette Shortcut Menu Commands

Command	Description
Insert	Inserts a new folder. When you select this command, you are prompted to supply the name of the folder.
Rename	Renames a folder. When you select this command, you are prompted to supply the new name.
Delete	Deletes the selected item.
Move Up	Moves the selected item up one position under its parent.
Move Down	Moves the selected item down one position under its parent.

Table 4-8 Palette Shortcut Menu Commands (Continued)

Command	Description
Promote	Assigns the selected item to the next level up in the hierarchy. For example, suppose Field1 is a child of Group1. If you select Field1 and click the Promote tool, then Field1 becomes a sibling of Group1.
Demote	Assigns the selected item to the next lower level in the hierarchy. When you demote an item, it becomes a child of the sibling that immediately precedes it. For example, suppose Field1 is a sibling of Group1, and that it and immediately follows Group1. If you select Field1 and click the Demote tool, Field1 becomes a child of Group1.

Copying Items From the Active Message Format to the Palette

To copy an item from the document currently open in Format Builder to the palette:

1. If it is not already displayed, choose **View—Show Palette**.

2. In the navigation tree, select the item you want to add to the palette.

3. Drag the item to palette window, then drop it in the desired location in the hierarchy.

   The item is copied to the selected location.

   Note: You cannot add an item that depends on the existence of another item to the palette. For example, you cannot add a field or group reference, and you cannot add an item for which a Repeat Field is specified.

   Adding comments is possible, but not recommended because comments do not have unique names and therefore are indistinguishable on the palette.

Deleting Items From the Palette

To delete an item from the palette:

1. Right-click the item to be deleted to display the shortcut menu.

2. Select **Delete**.

   You are prompted to confirm the deletion.

3. Click **OK** to delete the item.

Copying Palette Items from the Palette to the Active Message Format

To copy an item from the palette to a message format document currently open in Format Builder:

1. If it is not already displayed, choose **View—Show Palette** to display the palette.

2. In the palette window, select the item you want to add to your message format.

3. Drag the item to navigation tree, then drop it in the desired location in the hierarchy.

   The item is copied to the selected location in the message format.

Saving a Message Format

You can either save your MFL files directly into a Schemas folder in the file system or you can save the MFL file to the file system and later import the MFL file into a Schemas folder as described in the preceding bullets:

- You can save MFL files directly into a Schemas project folder in the file system. For example, if your application saved in the `c:\bea\weblogic81\apps\myApp` directory contains the default Schemas project, you can save the MFL file directly into this `c:\bea\weblogic81\apps\myApp\Schemas` directory in the file system.

- You can save a message format document to your file system as described in this section and later when creating a transformation, you import this MFL file into a Schemas folder of a business process application. To learn more, see "Importing Schemas" on page 2-5.

To save a message format file for the first time:

1. In the Format builder menu bar, choose **File—Save As**.

   The **Save As** dialog box is displayed as shown in the following figure:
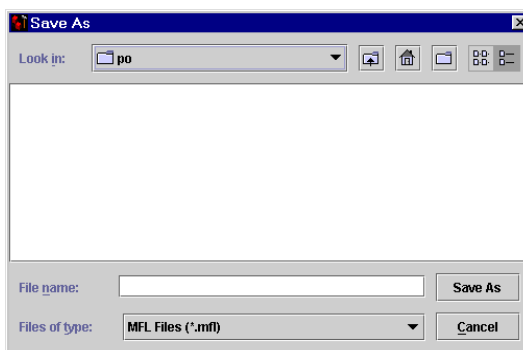
2. Navigate to the folder in which you want to save the file.

3. Enter the name you want to assign to the file in the **File Name** field.

    Note: If you do not include an extension in your filename, Format Builder automatically assigns the default extension: .mfl.

4. Click **Save As** to save the file in the specified location with the specified name and extension.

To save changes to an existing file, choose **File—Save**.

To save an existing file to a new name, choose **File—Save As** and follow steps 2 through 4 in the preceding procedure.

Opening an Existing Message Format File

You can open a message format document on your file system as described in the this section.

To open an existing message format file:

1. In the Format Builder menu bar, choose File—Open.

    The **Open** dialog box is displayed as shown in the following figure:
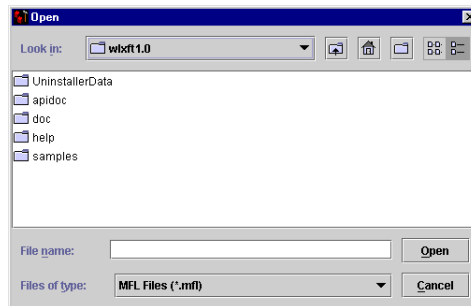
2. Locate and select the desired file.

3. Click **Open**.

Using Internationalization Features

You can use the internationalization features in Format Builder by changing the options for an individual message file or by setting the default Format Builder options to include internationalization. For details, see:

- "Changing Options for a Message Format" on page 4-45
- "Setting Format Builder Options" on page 4-45

Changing Options for a Message Format

To change options for a message format file:

1. Select the root node of the message format in the navigation tree.

2. Choose **File—Properties**.

   The File Properties dialog box displays the Message Format Version and the Default Message Format (MFL) Encoding.
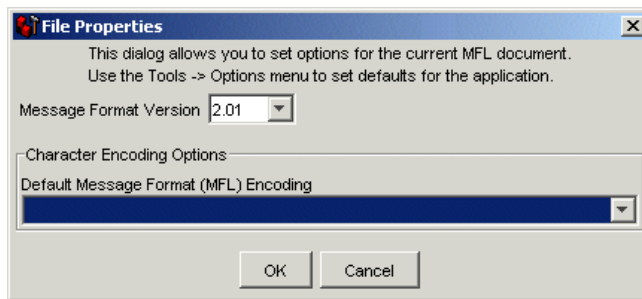
3. Select a type of character encoding for the MFL document from the list of encoding names and descriptions for this file. (To change the default settings for all new message format documents, choose **Tools—Options**.)

4. Click **OK**.

   Your changes are reflected in the MFL document when you test it using Format Tester.

Setting Format Builder Options

You can set several options to control the overall operation of Format Builder.

To set Format Builder options:

1. Choose **Tools—Options**.

   The **Options** dialog box is displayed.

2. Enter data in the fields as described in the following table:

Table 4-9 Format Builder Options

Category	Option	Description
N/A	Default Message Format Version	Select the version to be associated with new MFL documents.
		Note: Each message format document is associated with its own message format version. The version specified for a message format can be changed from the default from the File Properties dialog box described in the preceding section "Changing Options for a Message Format.".
Character Encoding Options	Default Message Format (MFL) Encoding	Select the character encoding to be associated with new MFL documents. The character encoding associated with an MFL document specifies the encoding used for the MFL document itself, and the XML output it generates.
	Default Field Code Page	Select the code page, from the list of non-XML formats, to be used as the default code page for each field created in your MFL documents. A code page specifies the character encoding of the non-XML data in the field.

Table 4-9 Format Builder Options (Continued)

Category	Option	Description
XML Formatting Options	Initial Indent	Enter the number of spaces by which to indent the root element when generating the XML output.
	New Line Indent	Enter the number of spaces by which to indent a new child element when generating the XML output.
XML Content Model Options	Auto-generate DTD	Generates a DTD document which captures the content model defined in the MFL document. If you specify Auto-generate DTD, when you save an MFL document to the file system, the DTD is saved in the same directory.

3. Click one of the following:

   – **OK**—saves your changes and dismisses the dialog box.

   – **Cancel**—discards your changes and dismisses the dialog box.

Format Builder Menus

The following menus are available in Format Builder: **File**, **Edit**, **Insert**, **View**, **Tools**, and **Help**.

The commands available on each menu are described in the following sections.

Note: Some commands may be unavailable, depending on which actions you have taken and what is selected in the navigation tree.

File Menu

The following commands are available from the **File** menu.

Table 4-10 File Menu Commands

Command	Description
New	Creates a new message format document.
Open	Opens an existing message format document.
Close	Closes the current message format document.
Save	Saves the current message format document.

Table 4-10 File Menu Commands (Continued)

Command	Description
Save As	Saves the current message format under a different name.
Properties	Opens the File Properties dialog box for the active message format document. This dialog allows you to set options for the active MFL document (see "Changing Options for a Message Format" on page 4-45). Choose **Tools—Option** to set defaults for the application (see "Setting Format Builder Options" on page 4-45).
Exit	Exits the Format Builder.

Edit Menu

The following commands are available from the **Edit** menu.

Table 4-11 Edit Menu Commands

Command	Description
Undo *action*	Reverses the previous action. The Undo command on the Edit menu is constantly refreshed to indicate the action most recently performed that can be nullified. For example, if you change the name of a field to Field1 and click Apply, the listing for the Undo command contains the following text: Undo Apply Field Field1.
	Format Builder supports multiple undoing of previous actions.
Redo *action*	Reverses the effects of the Undo command. The Redo command in the Edit menu is constantly refreshed to indicate the action that can be redone. For example, if you change the name of a field to Field1 and then click Undo, the listing for the Redo command contains the following text: Redo Apply Field Field1.
	Format Builder supports multiple redoing of actions previously undone.
Cut	Removes the selected item along with its child objects. The item is placed in the clipboard and can be pasted in a new location.
	Note: This action is not available if the Message Format (root) item is selected.

Table 4-11 Edit Menu Commands (Continued)

Command	Description
Copy	Makes a copy of the selected item along with its child objects. The copy is placed in the clipboard and can be pasted in a new location. **Note:** This action is not available if the Message Format (root) item is selected.
Paste	Inserts the current contents of the clipboard. When you select Paste, the following Paste menu options are displayed: • As Child • As Sibling • As Reference
Duplicate	Makes a copy of the currently selected item and pastes it as a sibling. The duplicate item contains the same values and child objects as the original. The name of the duplicate is the same as that of the original, with the addition of a prefix: *New*. Thus, for example, if the name of the original item is MyField1, then the name of the duplicate is NewMyField1.
Delete	Deletes the item selected in the navigation tree, as well as all child objects of that item.
Move Up	Moves the selected item up one position under its parent.
Move Down	Moves the selected item down one position under its parent.
Promote	Assigns the selected item to the next level up in the hierarchy. For example, suppose Field1 is a child of Group1. If you select Field1 and select Promote, then Field1 becomes a sibling of Group1 and is inserted immediately after Group1.
Demote	Assigns the selected item to the next lower level in the hierarchy. When you demote an item, it becomes a child of the group that immediately precedes it. For example, suppose Field1 is a sibling of Group1 and immediately follows Group1. If you select Field1 and select Demote, Field1 becomes a child of Group1.

Insert Menu

The following commands are available from the **Insert** menu.

Table 4-12 Insert Menu Commands

Command	Description
Field	Inserts a new field. You can insert the field as either a child or sibling of the item selected in the navigation tree.
Group	Inserts a new group. You can insert the group as either a child or sibling of the item selected in the navigation tree.
Comment	Inserts a comment. You can insert the comment as either a child or sibling of the item selected in the navigation tree.

View Menu

The following commands are available from the **View** menu.

Table 4-13 View menu Commands

Command	Description
Show Palette	Displays the Palette window.
Expand All	Expands the entire navigation tree to show the child objects of all items in the navigation tree.
Collapse All	Collapses the entire navigation tree to show only the root message format.

Tools Menu

The following commands are available from the **Tools** menu.

Table 4-14 Tools Menu Commands

Command	Description
Import	Displays a list of the installed importers. Choose the importer from which you want to import a message.
Test	Opens the Format Tester.
Options	Displays the Format Builder Options dialog box.

Help Menu

The following commands are available from the **Help** menu.

Table 4-15 Help Menu Commands

Command	Description
Help Topics	Displays the online help in your default browser.
How Do I	Displays a list of common Format Builder tasks. Click a task to view the step-by-step instructions.
About	Displays version and copyright information for the Format Builder and the JDK you are running.

Importing Existing Metadata to Create Format Schemas (MFL Files)

WebLogic Integration provides utilities that allow you to import COBOL copybooks, import XML Schemas, and convert C structure definitions into MFL files. The following topics explain how to perform these import operations:

- Importing a COBOL Copybook
- Importing C Structures

Importing a COBOL Copybook

WebLogic Integration includes a feature that allows you to import a COBOL copybook into Format Builder by creating a message definition to transform the COBOL data. When importing a copybook, you can use comments to document the imported copybook and the Groups and Fields it contains.

To import a COBOL copybook:

1. Choose **Tools—Import—COBOL Copybook Importer**.

   The **COBOL Copybook Importer** dialog box is displayed.
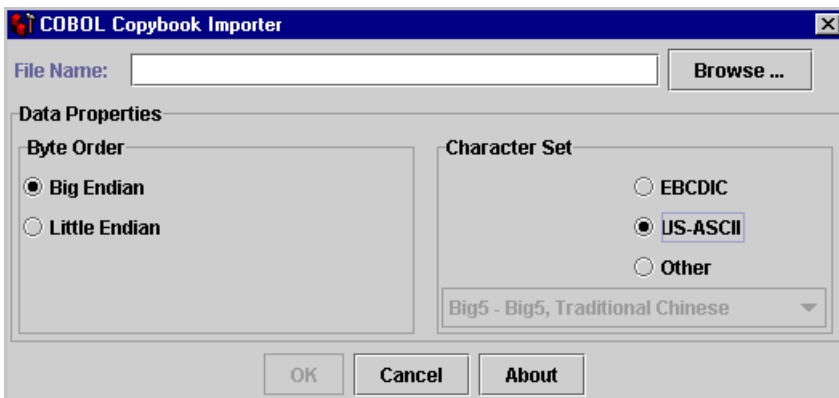
2. Designate the properties, as described in the following table:

Table 4-16 COBOL Copybook Importer Properties

Property	Value	Description
File Name	text string	Type the full pathname of the file you want to import or use the Browse button to navigate to the location of the file.
Byte Order	Big Endian	Select this option for IBM 370, Motorola, and most RISC designs (IBM mainframes and most UNIX platforms).
	Little Endian	Select this option for Intel, VAX, and Unisys processors (Windows, VMS, Digital, UNIX, and Unisys)
Character Set **Note:** The character set is an attribute of the originating host machine.	EBCDIC	Select this option to set the character set to EBCDIC.
	US-ASCII	Select this option to set the character set to US-ASCII.
	Other	Select character encoding of the field data by using a list of code pages.

3. Click one of the following:

   – **OK**—imports the COBOL Copybook using the settings you defined.

   – **Cancel**—closes the dialog box and returns you to Format Builder without importing.

   – **About**—displays information about the COBOL Copybook importer, including the version being used and copybook features that are supported.

After you import a copybook, you can work in the same way you work with any message format definition. If you find an error or unsupported data type in the copybook, a message is displayed, informing you of the error. You can choose to have the error displayed or saved in a log file for future reference.

The following table provides a listing and descriptions of the sample files installed for the COBOL copybook importer. All directory names are relative; the specified directories are under the `WL_HOME`\integration\samples\di directory where `WL_HOME` is the top-level directory of your WebLogic Platform installation. (For example, if you installed WebLogic Platform in the c:\bea directory, the di directory is located at the following location: c:\bea\weblogic81\integration\samples\di.)

Table 4-17 Sample COBOL Copybook Files

Directory	File	Description
COBOL\	emprec5.cpy	Sample copybook file
COBOL\	emprec5.data	Test data corresponding to emprec5.cpy

Importing C Structures

WebLogic Integration includes a C struct importer utility that converts a C struct definition into an MFL message definition by generating the following types of output data:

- MFL document

- C code

Whichever type of output you want, you must first specify a .c or .h input file, which must be parsed, and then select the desired structure. Then you can choose between MFL (default) or C code for your output.

All input to the parser must be valid C code. In addition, all external references, such as #include, #define, and typedef statements, must be resolved before you can use them. You can resolve them by editing them manually or by using the compiler's preprocessor.

Various platform-specific parameters may affect the description of data for C code. For example, the length of a long on a particular platform affects the non-XML data that conforms to a particular structure definition.

Two methods are available for dealing with these platform dependencies, depending on whether or not MFL is generated directly into Format Builder. If you want to generate MFL and have that

MFL displayed immediately in Format Builder, you must supply the platform-dependent parameters in a configuration file.

Alternately, if you choose to generate your source in C, you may compile the C code on the desired machine. The compiler on that machine accounts for the necessary platform-dependent information. This approach allows you to produce an executable file that, when run, produces two files: an MFL document and non-XML data that conforms to that MFL. The MFL document can be opened in Format Builder and the non-XML data file can be opened in Format Tester.

Generating MFL directly into Format Builder requires platform configuration parameters found in an existing configuration file or a new configuration file created with the hardware profile editor. The hardware profile editor allows you to specify an existing profile that can be loaded, updated, and saved.

The source code for a utility that generates hardware profiles according to your needs is provided in the `WL_HOME`\integration\samples\di\CFG directory where `WL_HOME` is the top-level directory of your WebLogic Platform installation. (For example, if you installed WebLogic Platform in the c:\bea directory, the cfg directory is located at the following location: c:\bea\weblogic81\integration\samples\di\CFG.)

Sample C Struct Importer Files

The following table provides a listing and descriptions of the sample files installed for the C struct importer. All directory names are relative; the specified directories are under `WL_HOME`\integration\samples\di directory where `WL_HOME` is the top-level directory of your WebLogic Platform installation. (For example, if you installed WebLogic Platform in the c:\bea directory, the di directory is located at the following location: c:\bea\weblogic81\integration\samples\di.)

Table 4-18 Sample C Struct Importer Files

Directory	File	Description
C	emprec5.h	C version of the emprec5.cpy sample Copybook file, with some typedefs.
C	emprec5n.h	Variant of the emprec5.h file in which a nested struct definition, but no typedef is used.
C	emprec5s.h	Simple version of the emprec5.h file.
C	ntfsez.h	Small sample, extracted from the ntfs.h file, designed to test recursive typedefs.

Table 4-18 Sample C Struct Importer Files (Continued)

Directory	File	Description
Cfg	cprofile.c	Source code for the `cprofile.c` utility; designed to generate profiles on various platforms.

The following `.cfg` files were generated by the `cprofile` program on various platforms. Each `.cfg` file contains a value for DESCRIPTION.

Directory	File	Description
Cfg	dec8cc.cfg	DEC Alpha 1091, Digital UNIX 4.0e, cc compiler
Cfg	hp5cc.cfg	HP-UX B.11.00, cc compiler
Cfg	nt4bcc5.cfg	Windows NT 4.0, Borland 5.x compiler, default switches
Cfg	nt4vc6.cfg	Windows NT 4.0, Visual C++ 6.x compiler, default switches
Cfg	sun7cc.cfg	SunOS 5.8, cc compiler
Cfg	w95bcc5.cfg	Windows 95, Borland 5.x compiler, default alignment
Cfg	w95vc5.cfg	Windows 95, Visual C++ 5.x compiler, default alignment

Starting the C Struct Importer

To start the C Struct Importer:

1. Start Format Builder. For instructions, see "Starting Format Builder" on page 4-12.

2. Choose **Tools→Import→C Struct Importer**.

The **C Struct Importer** dialog box is displayed.
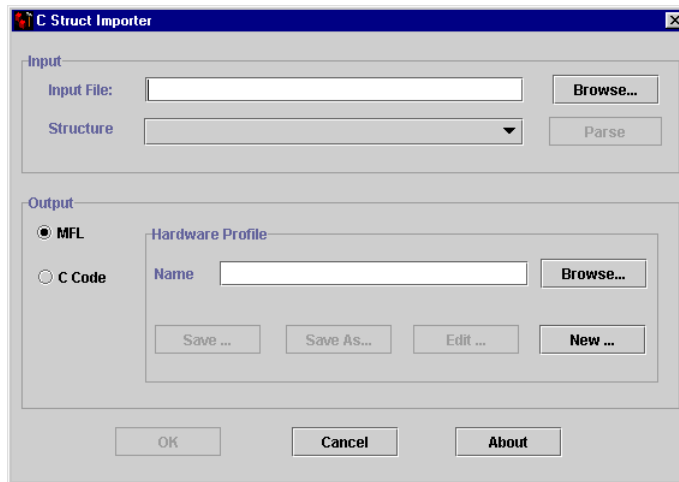
The **C Struct Importer** dialog box allows you to specify import properties, as described in the following table:

Note: Initially, MFL is specified as the default output type.

Table 4-19 C Struct Importer Properties

Category	Property	Description
Input	Input File	Type the full pathname of the file you want to import or use the Browse button to navigate to the location of the file.
	Structure	Drop-down list of structures found in the input file after parsing is successful.
	Parse	Select this option to parse the input file. If successful, the Structure list box is populated with the list of structures found in the input file.

Table 4-19 C Struct Importer Properties (Continued)

Category	Property	Description
Output	MFL	If you select this option, you can generate MFL from a structure definition and a hardware configuration file. The Hardware Profile dialog box is displayed with the following options.
		• Name—Specify an existing profile either by entering the file name or using the Browse option. The prebuilt hardware profiles may be found in the `samples\di\cfg` directory.
		• Save—saves the current hardware profile.
		• Save As—allows you to save the current hardware profile under another name.
		• Edit—allows you to edit the current hardware profile.
		• New—allows you to create a new hardware profile.
	C Code	If you select this option, you can generate C source code to compile on the target machine and execute to produce MFL. The C Code File Names dialog box is displayed with the following options.
		• MFL Gen—specifies the C source code file name that must be compiled on the target machine to generate MFL. Use the Browse option to navigate to the directory where you want the file to reside.
		• Data Gen—specifies the C source code file name that must be compiled on the target machine for generating test data. Use the Browse option to navigate to the directory where you want the file to reside.

3. Click one of the following:

   – **OK**—saves your hardware profile changes.

   – **Cancel**—dismisses your hardware profile changes.

   – **About**—displays information about the C Struct Importer, including the version number and the release date.

Understanding Hardware Profiles

The hardware profiles used by the C Struct Importer contain data size and alignment information for specific hardware and compiler combinations and are used to generate MFL for C structures. They are stored in configuration files that can be created, loaded, updated, and saved.

The `cprofile.c` source file in the `WL_HOME`\integration\samples\di\CFG directory is used to generate these profiles for any platform. This code is designed to be compiled and executed on the target platform with the compiler normally used. You should be able to compile and execute it on any platform with an ANSI standard C compiler in order to generate a profile configuration file that can be imported into the C Struct Importer. (Where `WL_HOME` is the top-level directory of your WebLogic Platform installation. For example, if you installed WebLogic Platform in the `c:\bea` directory, the CFG directory is located at the following location: `c:\bea\weblogic81\integration\samples\di\CFG`.)

Building the Hardware Profile Utility

To produce acceptable parser input, execute the appropriate commands for your platform:

- On Windows NT, use the VC++ preprocessor:

  VC++ Compiler
  ```
  cl /P cprofile.c (output in cprofile.i)
  ```

  GNU Compiler
  ```
  gcc -P -E cprofile.c>cprofile.i
  ```

- On UNIX

  ```
  cc -P cprofile.c (output in cprofile.i)
  ```

Running the Hardware Profile Utility

To execute the `cprofile` program and specify a hardware profile name, enter the following text at a command prompt:

```
cprofile configfilename [DESCRIPTION]
```

A description is optional. If you decide to provide one, put it in the configuration file as the value of DESCRIPTION. If the description contains embedded blanks, enclose it in quotes.

Generating MFL

To generate MFL:

1. Enter a filename in the **Input File** field, or click **Browse** and select a file from the list that is displayed.

2. Click **Parse** to parse the file.

   Upon completion, the **Structure** list is populated with the structures found in the input file.

   Note: If your file does not parse correctly, we recommend that you proceed in one of two ways:

   - Run your `.h` or `.c` source code through the compiler, preprocessor, and then run the processor output through the parser.

   - Comment out the character creating the parsing failure and attempt to parse again. Please note that the parser fails at the first instance of incompatible data it encounters. Therefore, repetition of this step may be required.

3. Select the desired structure from the **Structure** drop-down list.

   At this point, you must provide some profile configuration data to generate the MFL directly. You can do this by either creating a new hardware profile or specifying an existing profile.

4. Specify an existing profile or create a new one by performing one of the following procedures:

   – Specify an existing profile in one of the following ways: enter the filename in the **Hardware Profile Name** field, or click Browse to select a file from the list that is displayed.

   Click **Edit** to open the hardware profile editor if you need to view or edit the profile parameters.

   Note: Hardware profiles for common configurations are prebuilt and may be found in the `WL_HOME\integration\samples\di\CFG` directory. (Where `WL_HOME` is the top-level directory of your WebLogic Platform installation. For example, if you installed WebLogic Platform in the `c:\bea` directory, the `CFG` directory is located at the following location: `c:\bea\weblogic81\integration\samples\di\CFG`.)

   – Click **New** to create a new hardware profile. The Hardware Profile editor is displayed with the default parameters loaded. Specify a name and description for the new profile, and modify the primitive data types and byte order as required.
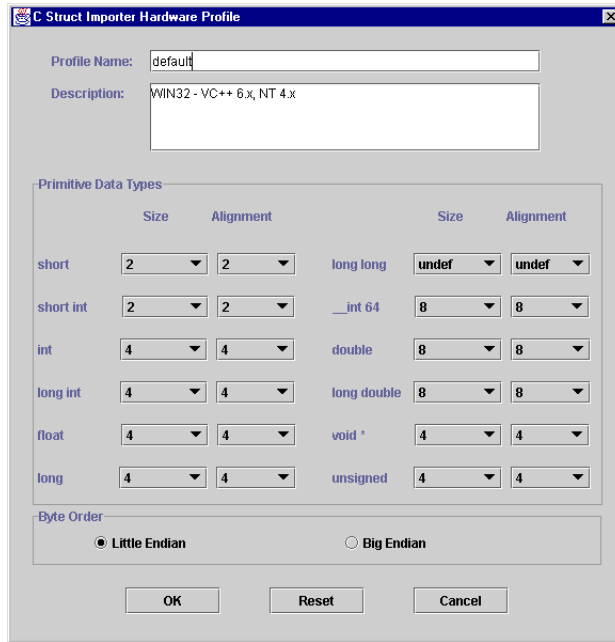
5. Click **OK** to save your hardware profile changes and return to the C Struct Importer dialog box.

6. Click **OK** to generate your MFL. If the generation is successful, you are returned to Format Builder with an MFL object listed in the navigation tree. The MFL object reflects the same name as the input file used in the parse operation.

   If errors are detected during the generation process, the MFL Generation Errors dialog box is displayed providing you with an opportunity to view or file the error log.
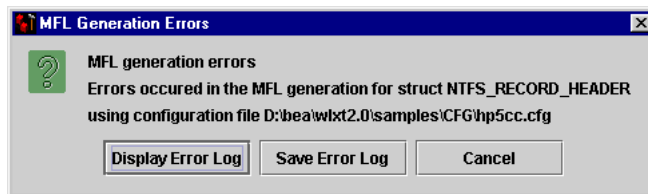
7. Click one of the following:

   – **Display Error Log**—to view any errors encountered,

- **Save Error Log**—to save the error log to the location of your choice, or

- **Cancel**—to dismiss the MFL Generation Errors dialog box.

After you determine which errors were generated, you can return to the C Struct Importer and repeat the applicable steps.

Generating C Code

To generate C code:

1. Enter a filename in the **Input File** field, or click **Browse** and select a file from the list that is displayed.

2. Click **Parse** to parse the file.

   Upon completion, the **Structure** list is populated with the structures found in the input file.

   Note: If your file does not parse correctly, we recommend that you proceed in one of two ways:

   - Run your `.h` or `.c` source code through the compiler, preprocessor, and then run the processor output through the parser.

   - Comment out the character creating the parsing failure and attempt to parse again. Please note that the parser fails at the first instance of incompatible data it encounters. Therefore, repetition of this step may be required.

3. Select the desired structure from the **Structure** drop-down list.

4. Select the **C Code** option.

5. Enter a filename in either the **MFL Gen** or **Data Gen** field, or click **Browse** and select a file from the list that is displayed.

6. Click **OK**.

   Messages are displayed if you are about to overwrite an existing file or if the code generation has succeeded or failed.

7. Copy the generated source code to the target platform, compile and execute it.

   Note: You must copy the input file containing the struct declarations, as well. When compiled, both programs accept the name of the output file as an argument.

8. Copy the generated MFL or data back to the platform on which Format Builder is running.

Importing an XML Schema

WebLogic Integration includes a feature that allows you to import an XML Schema into Format Builder. The imported XML Schema provides a starting point for creating the MFL message definitions used for transforming data between XML and non-XML formats.

To import a XML Schema from an XSD file:

1. Choose **Tools→Import→XML Schema Importer**.

   The **Select XSD File & Root Element** dialog box is displayed.

2. In the **XML Schema Definition** field, select an XSD file (ends in the `.xsd` extension.)

3. In the **Root Element** drop-down menu, select a root element.

4. Enter a value in the **MFL Field Delimiter Default** field.

   A delimiter is a character that marks the end of the data field.

5. Click **OK**.

   Note: Imported Element attributes are not converted.

Testing the Format Schemas (MFL Files)

After you build a format schema, you can test it using the Format Tester. The Format Tester parses and reformats data as a validation test and then generates sample non-XML or XML data. This sample data can be edited, searched, and debugged to produce the expected results. Format Tester uses the data transformation run-time engine to perform the test transformation.

This section discusses the following topics:

- Starting the Format Tester
- Using the Format Tester Dialog Box
- Testing Format Definitions
- Debugging Format Definitions

Starting the Format Tester

To start Format Tester:

1. Start Format Builder is not already running. For instructions, see "Starting Format Builder" on page 4-12.

2. Choose **Tools—Test**.

    The **Format Tester** dialog box is displayed as shown in the following figure:
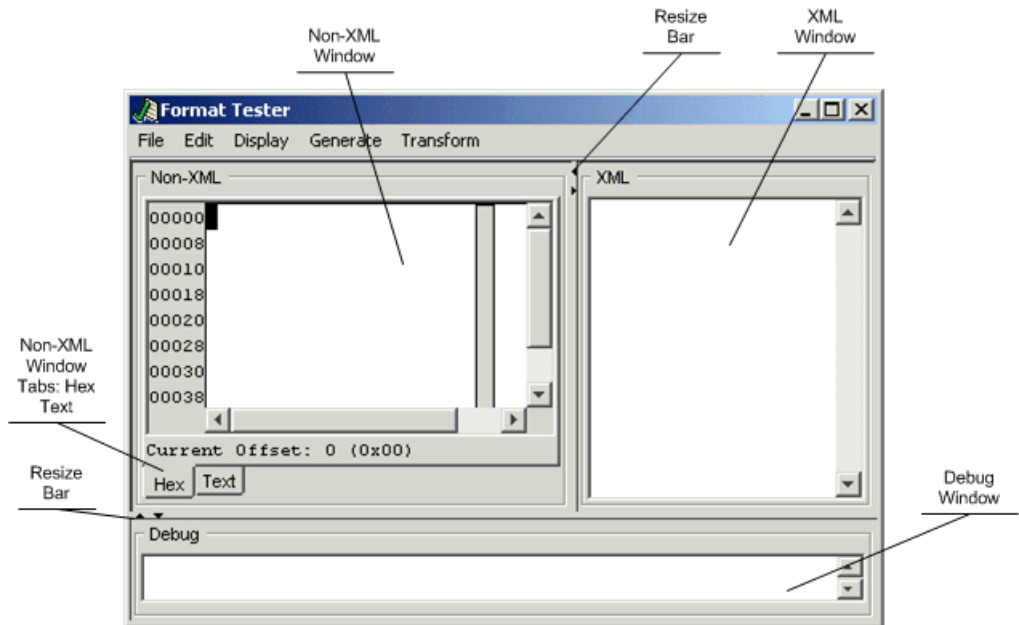
The **Format Tester** dialog box is divided into three windows: the **Non-XML** window, the **XML** window, and the **Debug** window. Resize bars divide the windows. You can drag the resize bar to adjust window size, or click an arrow on the bar to show or hide a window. For example, you can click the left arrow on the bar dividing the **Non-XML** and **XML** windows to hide the **Non-XML** window. If a window is hidden, you can drag the bar or click the appropriate arrow to restore the window.

Note: When you open the Format Tester for the first time in a session, only the **Non-XML** and **XML** windows are visible. To open the **Debug** window, use the resize bar at the bottom of the **Format Test** dialog box, or choose **Display—Debug** to toggle the **Debug** window on and off.

Using the Format Tester Dialog Box

The following topics explain how to use various tools provided in the **Format Tester** dialog box to navigate and execute commands:

- Using the Non-XML Window
- Using the XML Window
- Using the Debug Window
- Using the Resize Bars
- Using the Menu Bar
- Using the Shortcut Menus

The following topics explain how to use each of these features to help you accomplish your task.

Using the Non-XML Window

The **Non-XML** window can contain sample data that has been:

- Generated based on the active MFL document.
- Transformed from the contents of the XML window.
- Specifically designed to test the active MFL document.

You can open an existing non-XML data file, edit or save the contents of the window, or clear the window as required for your test situation. For details, see "Using the Menu Bar" on page 4-66 and "Using the Shortcut Menus" on page 4-70.

The **Non-XML** window of the **Format Tester** dialog box serves as a non-XML file editor. The window contains the following tabs:

- **Hex**—displays data offsets, the hex value of individual bytes, and the corresponding text, which can be displayed in either ASCII or EBCDIC format.
- **Text**—Text only display.

The editor allows you to edit a hex byte or a text value. If a hex data value is modified, the corresponding text value is updated, and vice versa.

Using the Data Offset Feature

The data offset feature of the **Hex** tab allows you to display the data offsets as hexadecimal or decimal addresses.

To change the format of the data offsets:

1. Choose **Display—Hex**.

   The following two data offset options are displayed:

   – Offsets as Hexadecimal

   – Offsets as Decimal

2. Select an option that best suits your needs. The data offset portion of the **Non-XML** window changes dynamically to reflect your choice.

Using the Text Feature

The **Text** tab of the **Non-XML** window displays the printable characters (usually in the form of words and numbers) and certain control characters (carriage return, tab, and so on). For example, carriage returns are shown as line breaks. Non-printable characters, are displayed as small squares.

Using the XML Window

The XML window can contain sample XML that has been:

- Generated based on the active MFL document.

- Transformed from the contents of the **Non-XML** window.

- Specifically designed to test the active MFL document.

You can open an existing XML file, edit or save the contents of the window, or clear the window as required for your test situation. For details, see "Using the Menu Bar" on page 4-66 and "Using the Shortcut Menus" on page 4-70.

When XML is generated, the XML Formatting Options specified in the Format Builder **options** dialog box are used. For additional information, see "Setting Format Builder Options" on page 4-45.

Using the Debug Window

The **Debug** window displays the actions that occur during a transformation, any errors that are encountered, and field and group values, along with delimiters. To determine the cause of an error, identify the last field that parsed successfully and examine the properties of the field listed after it in the navigation tree.

When you open the Format Tester for the first time in a session, only the **Non-XML** and **XML** windows are visible. To open the **Debug** window, choose **Display—Debug** to toggle the **Debug** window on and off. The **Debug** window opens below the **Non-XML** and **XML** windows.

Debug output is restricted to the most recent 64 KB of messages. This restriction prevents large debug output from causing a JVM out of memory event.

The debug log feature allows you to save all debugging information in a file. For details, see "Using the Debug Log" on page 4-74.

Note: Use of the Debug window or log file increases the time required to transform from XML to non-XML.

Using the Resize Bars

You can change the dimensions of any window in the Format Tester by using the resize bars located between the **Non-XML**, **XML**, and **Debug** windows. To change the size of a window, select a resize bar and drag in the appropriate direction (up or down or to the left or right) to enlarge one of the windows and reduce the other.

Each resize bar also contains two directional buttons. Click the appropriate button to show or hide any of the three windows.

Using the Menu Bar

Format Tester functions can be accessed from the five menus listed in the menu bar at the top of the main window.


```
File   Edit   Display   Generate   Transform
```

You can expand a **Format Tester** menu in either of two ways:

- Click the name of the menu in the menu bar.

- On your keyboard, press Alt + *key*, where *key* is the underlined letter in the menu name.

To execute a command, select it from the menu. Some commands can also be executed via the keyboard shortcut indicated on the menu (For example, a Ctrl + *key* sequence.) The commands available on each menu are described in the following sections.

File Menu

The following commands are available from the **File** menu.

Table 4-20 File Menu Commands

Command	Description
Open Non-XML	Displays the **Open** dialog box to allow you to select and open a file in the **Non-XML** window. **Note:** The default file extension for non-XML files is `.DATA`.
Open XML	Displays the Open dialog box to allow you to select and open a file in the XML window. **Note:** The default file extension for XML files is `.XML`.
Save Non-XML	Displays the **Save** dialog box to allow you to save the contents of the **Non-XML** window.
Save XML	Displays the **Save** dialog box to allow you to save the contents of the **XML** window.
Debug Log	Displays the **Save** dialog box to allow you to save the debugging information in a text file.
Close	Closes the **Format Tester** window.

Edit Menu

The following commands are available from the **Edit** menu.

Table 4-21 Edit Menu Commands

Command	Description
Cut	Removes the currently selected text and places it on the clipboard for pasting in another location.
Copy	Copies the currently selected text and places it on the clipboard for pasting in another location.
Paste	Inserts the cut or copied text at the cursor location.
Find	Allows you to search for a hex or text value. This command applies to the content of the **Non-XML** window only. **Note:** The text search is case sensitive.
Find Next	Repeats the last Find from the current cursor position. This command applies to the content of the **Non-XML** window only.
Go To	Allows you to move the cursor to a specified byte offset in the **Non-XML** window.

Display Menu

The following commands are available from the **Display** menu.

Table 4-22 Display Menu Commands

Command	Description
XML check box	Check to display the **XML** window, uncheck to hide the window. When unchecked, the **Non-XML** window expands to fill the **Format Tester** dialog box.
Debug check box	Check to display the **Debug** window, uncheck to hide the window.
Clear→Non-XML	Clears the contents of the **Non-XML** window.
Clear→XML	Clears the contents of the **XML** window.
Clear→Debug	Clears the contents of the **Debug** window.

Table 4-22 Display Menu Commands (Continued)

Command	Description
Hex—Offsets as Hexadecimal option button	Displays the offset values as hexadecimal. Mutually exclusive with the Hex—Offsets as Decimal option.
Hex—Offsets as Decimal option button	Displays the offset values as decimal. Mutually exclusive with the Hex—Offsets as Hexadecimal option.

Generate Menu

The following commands are available from the **Generate** menu.

Table 4-23 Generate Menu Commands

Command	Description
Non-XML	Generates non-XML data to match the MFL document specification.
XML	Generates XML data to match the MFL document specification.
Prompt while generating data check box	If checked, you are prompted to specify the following during the generation process: • Whether or not to include optional fields or groups in the output. • Which choice of children to include in the output. • The number of times to include repeating fields or groups in the output.

Transform Menu

The following commands are available from the **Transform** menu.

Table 4-24 Transform Menu Commands

Command	Description
Non-XML to XML	Based on the MFL document specification, converts the contents of the **Non-XML** window to XML. The XML output is displayed in the **XML** window.
XML to Non-XML	Based on the MFL document specification, converts the contents of the XML window to non-XML data. The non-XML output is displayed in the **Non-XML** window.

Using the Shortcut Menus

When you right-click in the **Non-XML**, **XML**, or **Debug** window, a menu of the most frequently used commands for that window is displayed. The following table describes the commands that are available from the shortcut menus.

Table 4-25 Non-XML, XML, and Debug Shortcut Menu Commands

Command	Description
Cut	Removes the currently selected text and places it on the clipboard for pasting in another location.
Copy	Copies the currently selected text and places it in the clipboard for pasting in another location.
Paste	Inserts the cut or copied text at the cursor location.
Clear	Clears the contents of the Non-XML, XML, or Debug window.
Generate	Generates non-XML or XML data to match the MFL document specification. This command is only available on the **Non-XML** and **XML** shortcut menus.
To XML	Converts the contents of the **Non-XML** window to XML. This command is only available on the **Non-XML** shortcut menu.
To Non-XML	Converts the contents of the XML window to non-XML. This command is only available on the XML shortcut menu.

**Table 4-25 Non-XML, XML, and Debug Shortcut Menu Commands (Continued)**

Command	Description
Text in ASCII	Changes the character set used for the text displayed in the text portion of the Hex tab to ASCII.
Text in EBCDIC	Changes the character set used for the text displayed in the text portion of the Hex tab to EBCDIC.

Testing Format Definitions

To test a message format definition:

1. Start Format Builder.

2. Open a Message Format file.

3. Start Format Tester.

4. Choose **File—Open Non-XML**, or **File—Open XML** to load the file you want to transform and view, or enter your own data in one of the two data windows.

5. Choose **Display—Debug** if you want to view the actions that take place during the transformation operation. This step is optional. If you want to be able to view debugging information later, you must open the **Debug** window before starting the transformation operation.

6. Choose **Transform—Non-XML to XML**, or **Transform—XML to Non-XML** to transform your data to the appropriate format.

   The transformed data is displayed in the **Non-XML** or **XML** window as shown in the following figure:
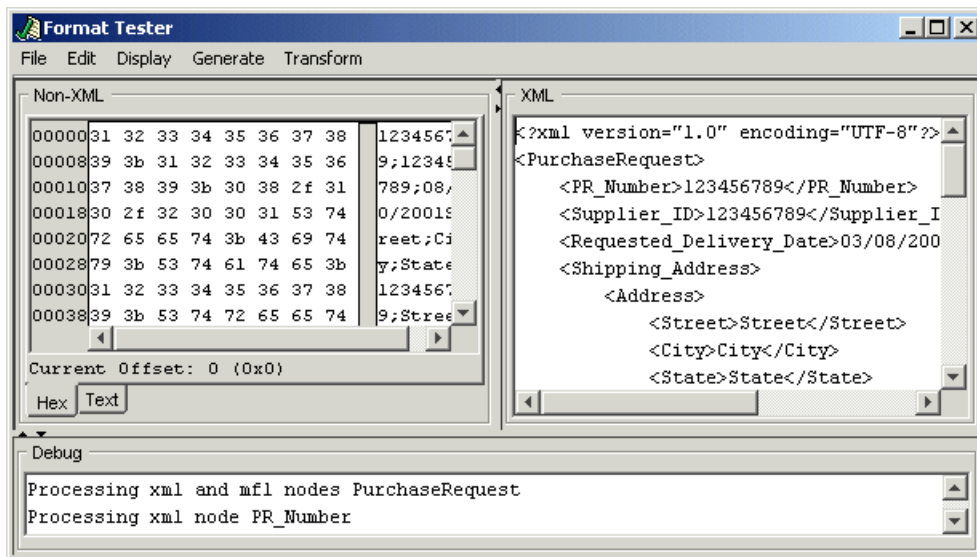
7. Correct any errors and test the transformation again.

8. Repeat steps 6 and 7 until the transformation is successful.

Note: You can leave Format Tester open while you modify the message format document in Format Builder. Changes to the document are detected automatically by Format Tester.

Debugging Format Definitions

The following topics explain how to use three Format Tester features to debug and correct your data:
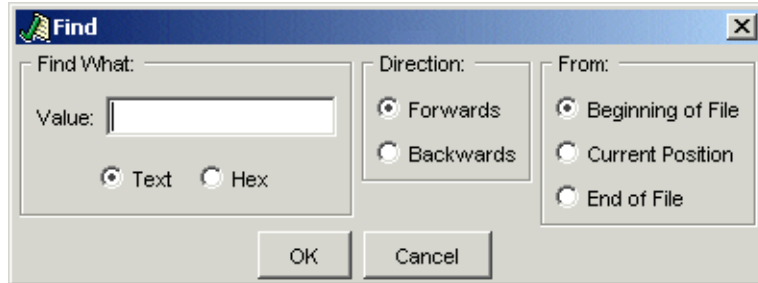
- Searching for Values
- Positioning to an Offset
- Using the Debug Log

Searching for Values

The Find command allows you to search for hex or text values in the non-XML data.

To search for a hex or text value:

1. In the **Format Tester** dialog box, choose **File—Non-XML** to open the non-XML data file you want to search.

2. Choose **Edit—Find** to open the **Find** dialog box.

3. Enter the target of the search in the **Value** field.

4. Select the **Text** or **Hex** option button to specify the value type.

5. Select the **Forwards** or **Backwards** option button to specify the search direction.

6. Select the **Beginning of File**, **Current Position**, or **End of File** option button to specify the search starting position.

7. Click **OK** to dismiss the **Find** dialog box and execute the specified search.

   If the value is found, the cursor moves the location of the value. If the value is not found, the following message is displayed: The specified search string was not found.

8. To repeat the search from the current cursor position, choose **Edit—Find Next**.

Positioning to an Offset

The Go To command allows you to move the cursor to a specified hexidecimal or decimal address (offset).

To move to a specified offset:

1. In the **Format Tester** dialog box, choose **Edit—Go To** to display the **Go To** dialog box.
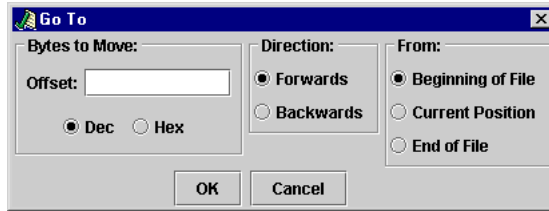
2. Enter the target offset in the **Offset** field.

3. Select the **Dec** or **Hex** option button to specify the type of offset.

4. Select the **Forwards** or **Backwards** option button to specify the direction.

5. Select the **Beginning of File**, **Current Position**, or **End of File** option button to specify the starting position.

6. Click **OK** to dismiss the dialog box and move the cursor to the specified offset.

Using the Debug Log

Although debugging information is not saved by default, the **Format Tester** dialog box allows you to specify a debug log file. When you specify a debug log file, all debugging information generated during your testing session is appended to the specified file.

To specify a debug log file:

1. In the **Format Tester** dialog box, choose the **File–Debug Log** to display the `Save` dialog box.

    Note: The `Debug Log` check box on the **File** menu is toggled upon selection. If the check box is checked, choosing **File–Debug Log** turns off logging to the file.

2. Select the desired directory, and then do one of the following:

    – To create a new log file, enter the name in the **File name** field and then click **Save**.

    – To use an existing log file, select the file and then click **Save**.

    If you select an existing file, the new debug information is appended to the end of the existing file.

Transforming Data Using XSLTs

In WebLogic Workshop business processes, XML data can be transformed using either XQuery expressions or eXtensible Stylesheet Language Transformations (XSLTs). An XQuery expression or query, is written in the XQuery language—a language defined by the World Wide Web Consortium (W3C) that provides a vendor independent language for the query and retrieval of XML data. An XSLT is written in the eXtensible Stylesheet Language (XSL)—an older language defined by the W3C that supports the use of stylesheets for the conversion of XML data.

To learn about XSLT, see the XSL Transformations (XLST) Version 1.0-W3C Recommendation 16 November 1999 at the web site of the W3C at the following URL:

http://www.w3.org/TR/xslt

The XSLT processor which is invoked by the transformation conforms to the November 16, 2002 Recommendation of the XSLT Specification.

WebLogic Workshop provides functionality for executing existing XSLTs in business processes. However, in WebLogic Workshop, the preferred method for data transformations is to use queries in the XQuery language. To learn more about adding queries to your business process, see "Transforming Data Using XQuery" on page 2-1. Data transformation using XSLT is supported primarily for customers who have upgraded from prior versions of WebLogic Integration and wish to continue using their XSLT-based maps without modification.

This section contains the following tasks:

- To Import an Existing XSLT file

- To Add a Data Transformation to a Business Process Using an XSLT

To Import an Existing XSLT file

This task describes how to import an XSLT file into your project.

1. In the **Application** tab, right-click any project folder (project folder names end with the string: Web) or product subfolder. (If the **Application** tab is not visible in WebLogic Workshop, choose **View**→**Application** from the menu bar.)

   Warning: Do not import the XSLT file into a Schemas project folder. (To learn more, see a "Creating Schemas Projects" on page 2-3.)

2. From the drop-down menu, select **Import...** .

   The **Import Files** dialog box is displayed.

3. Browse the file system, select your XLST file (files that end with the .xsl extension), and click **Import**.

   The XSLT file is imported into the project.

To Add a Data Transformation to a Business Process Using an XSLT

1. Select or create a Transformation file (also know as a DTF file because it ends in the .dtf extension). (For instructions on creating a Transformation file see "Creating a Transformation File and a Transformation Method" on page 2-10.

   In the **Application** tab, expand the folders that contain the Transformation file. (If the **Application** tab is not visible in WebLogic Workshop, choose **View**→**Application** from the menu bar.)

2. Select or create a Transformation method in a Transformation file. (For instructions on creating a method in a Transformation file, see "Creating a Transformation File and a Transformation Method" on page 2-10.)

   In the **Design View**, right-click the arrow representing the method.

3. From the drop-down menu, select **Configure XSLT Transformation Method**.

   The **Configure XSLT Transformation Method** dialog box is displayed.

4. From the **XSLT Transformation File** drop-down menu, select the appropriate XSLT file.

5. If your XSLT accepts parameters, add parameters to the Transformation method.

   Repeat the following steps for each parameter of the XSLT:

   a. In the **Parameter Name** field, enter the name of the XSLT parameter as it appears in the XSLT file.

b. From the **Parameter Type** drop-down menu, select the appropriate **Parameter Type**.

c. Click Add.

Note: The parameter name entered in the **Parameter Name** field, must match the parameter name specified in the XSLT file. For example, if `taxrate` is specified as a parameter in the XSLT source file, the same name (`taxrate`) must be specified in the **Parameter Name** field. The following segment of an example XSLT file shows the declaration of the variable `taxrate`:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.acme.com/trans" version="1.0">
<xsl:output method="update" indent="yes" />
<xsl:param name="taxrate"/>
...
```

Note: In the **Configure XSLT Transformation Method** dialog box, the order of the parameters specified is not significant. The parameters of the XSLT are matched to the parameters of the Transformation method by name.

6. Click **OK**.

This links the XSLT file with the selected method in the Transformation file. During run time, if the business process invokes this method, this XSLT is invoked.

Programming Transformations

This section describes programming considerations for transformations outside the mapper functionality of WebLogic Workshop.

This section contains the following topics:

- Java Classes Created From Importing Schemas

- Java Class Conversion

- Using the MflObject Interface to Transform Non-XML Data Programmatically

- Getting the TransformException Fault Code Programmatically

- Using the com.bea.WLXT Package (Deprecated)

Java Classes Created From Importing Schemas

When a schema is imported into your application, representations of these schemas are available in some of the panes of WebLogic Workshop. To learn more, see "Selecting Source and Target Types" on page 2-11.

In addition, Java classes for accessing the data represented in the schemas are generated, as shown in the following figure.
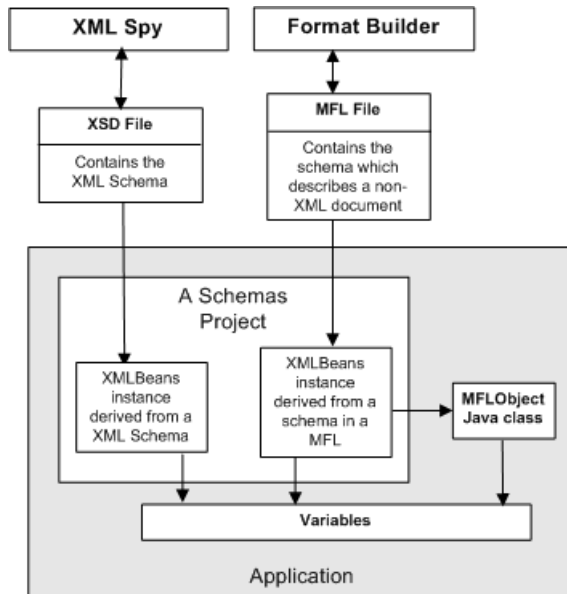
The generated Java classes are described in the following table:

| Importing . . .	Generates . . .	Example
An XSD file (contains an XML Schema)	XML Bean classes are generated for the XML Schema when the XML Schema is imported and built. These XML Bean classes provide methods for accessing the XML data that conforms to the imported XML Schema. To learn more, see Getting Started with XMLBeans.	If an XML Schema file with the document or root level elements: `price` and `widgitId` is imported into a Schemas project folder with a namespace of `http://www.example.org/quote`, the classes: `PriceDocument` and `WidgetIdDocument` are generated into the `org/example/quote` folder.

Importing . . .	Generates . . .	Example
An MFL file (contains a schema which describes non-XML data)	MflObject container classes are generated for the MFL file. These Java classes provide methods for the conversion between non-XML and XML data, programmatically outside the mapper functionality of WebLogic Workshop.	If the MFL file: `StockQuotes.mfl`, that specifies the `MessageFormat` name of `StockPrices`, is imported into a Schemas project folder, the `StockPricesMflObject` Java class is generated in the `Schemas/MFL Classes/stockquotes` folder. To learn more, see "Using the MflObject Interface to Transform Non-XML Data Programmatically" on page 6-9.
	A top-level XML Bean class based on the main MessageFormat name is generated from the MFL file when the MFL is imported. Other XML Bean classes may also be generated depending on the schema. The XML Bean classes contain `get` and `set` methods for accessing the data, similar to the XML Bean class that is generated when a XML Schema is imported and built. To learn more about XML Beans, see Getting Started with XMLBeans.

Before using the `get` and `set` methods of the XML Bean class, the non-XML data must first be converted to XML data. To learn more, see "Transforming Non-XML Data to Typed XML" on page 6-11.

The file name of the MFL document becomes the namespace of the MFL elements in the XML Bean class. | If the MFL file: `StockQuotes.mfl` which specifies a MessageFormat name of `StockPrices` is imported into a Schemas project folder, the class `StockPricesDocument` is generated under the folder named `stockquotes`. |

Importing . . .	Generates . . .	Example
	One or more XML Bean classes that correspond to the `StructFormat` element(s) that are children of the main `MessageFormat` element in the MFL file.	If the MFL file: `StockQuotes.mfl` contains a single `StructFormat` element named `PriceQuote`, which is a child of the `MessageFormat` element named `StockPrices` the following XML Beans classes are generated in the `Schemas/XML Bean Classes/stockquotes` folder:
	`StructFormat` elements are equivalent to root or document level elements in XML Schemas.	`PriceQuoteDocument`
	The file name of the MFL document becomes the namespace of the MFL elements in the XML Bean class. In addition, if the MFL is stored in a subfolder of the Schemas folder, the subfolders pathname becomes the package name of the namespace. For example, if the `StockQuotes.mfl` file is stored in the `Schemas/trading` folder, the full namespace for the generated XML Beans class is `trading/stockquotes`.	`PriceQuoteDocument.PriceQuote` `StockPricesDocument` `StockPricesDocument.StockPrices`

For example if the following `StockQuotes.mfl` file is imported into a Schemas folder:

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name='StockPrices' version='2.01'>
    <StructFormat name='PriceQuote' repeat='*'>
        <FieldFormat name='StockSymbol' type='String' delim=':'
codepage='windows-1252'/>
        <FieldFormat name='StockPrice' type='String' delim='|'
codepage='windows-1252'/>
    </StructFormat>
</MessageFormat>
```

The following is displayed in the current Schemas folder as shown in the following figure:
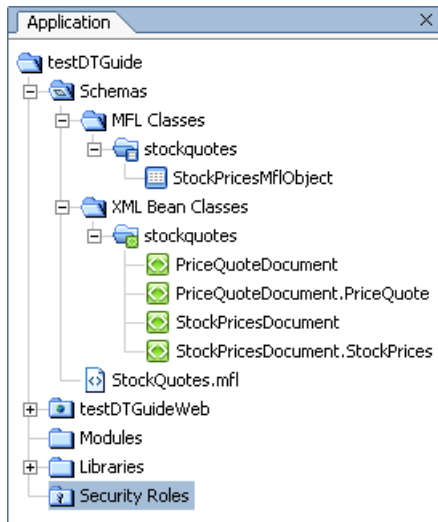
When schemas are imported into your application, representations of these schemas are available in some panes of the WebLogic Workshop. To learn more, see "Selecting Source and Target Types" on page 2-11.

Java Class Conversion

Java classes can be used as source and target types for transformations. Also, business process variables can be created from Java classes.

However, not all the fields that can make up a Java class are supported. This section describes which fields of a Java class are supported.

When you use a Java class or primitive in a transformation, WebLogic Integration converts it into an internal representation of XML Schema data types. Java fields that are not supported are ignored. WebLogic Integration then uses this internal representation to transform your Java classes to and from XML and non-XML (MFL) documents

Note: This internal representation of the XML Schema is not exposed in WebLogic Workshop.

Note: The mapper functionality of WebLogic Workshop will only display the fields of a Java class that are supported.

For a Java class to be used in WebLogic Integration, it must first be available in the WebLogic Workshop project. To learn more about including Java class in your project, see Using Existing Applications.

Note: Java interfaces and abstract Java classes are supported as source or target types for a transformation but you will not be able to execute your transfomation in the **Test View**. However, a transformation with a Java interfaces and/or abstract Java classes as source or target types can be invoked from a business process (JPD) without error.

Java classes are converted using the following procedure:

1. If the Java class is selected as an input type for a DTF, by default the name of the class becomes the name of the variable. (For example, if the class `Book` is selected as an input type, by default, the `BookDoc` variable is created by the mapper.) If the Java class is selected as the output type of a DTF, the name of the class becomes the name of the XML document's root element.

    All the Java fields that can be mapped are converted to the appropriate XML Schema elements in the new XML Schema. If a field cannot be mapped it is ignored. To learn more, see the following table.

2. For each field of the Java class, this conversion procedure is applied recursively to determine the type of each subelement. Fields that are themselves Java classes will be subelements in the internal XML Schema. These subelements will have the same name in the XML schema as their name in the Java class. For an example, see "Using Java Classes in Transformations" on page 2-42.

3. Java fields of the class are converted according to the following rules:

    -   Public instance member variables of supported Java types are converted.

        The following table describes the supported fields of Java classes and how these fields are converted to the internal XML Schema representation of the Java class. Also, the XML Schema data type can be converted back to the Java field unless otherwise noted.

Java Field or Class	Converts to this XML Schema Data Type . . .
Java primitive types	equivalent XML data types For example, Java primitive float type is converted to a XML Schema float type.
java.lang.String	xsd:string
java.util.Date	xsd:dateTime

Java Field or Class	Converts to this XML Schema Data Type . . .
`java.util.Calendar`	`xsd:dateTime`
`java.util.Date`	`xsd:dateTime`
`java.sql.Date`	`xsd:date`
`java.sql.Time`	`xsd:time`
`org.w3c.dom.Document`	`xsd:any`
`org.w3c.dom.Document Fragment`	`xsd:any`
`org.w3c.dom.Element`	`xsd:any`
`org.w3c.dom.Node`	`xsd:any`
XML Bean classes	To learn more see XMLBeans Support for Built-In Schema Types.
MflObject classes	The type of the corresponding XML Schema.
Java arrays	`xsd:sequence` of specified data type For example, an array of Java integers is converted to a XML sequence of integers.
`javax.sql.RowSet`	`xsd:anyType` **Note:** You cannot convert from XML back to a RowSet.
`java.util.Collection` and classes that implement the interface	`xsd:anySimpleType*` **Note:** You cannot convert from XML back to a Collection nor any class that implements the Collection interface because the data type of objects inside the Collection implementing class is not known. Use an array of a specific Java type as a target instead.

Note: Static member variables are not converted.

Note: Final instance member variables (constants) are not converted.

- Private fields are converted if JavaBean style `get` and `set` property methods for that field are available in the class. For example, the following private member field: `testBool` is converted to an XML schema type because the class that contains `testBool` has `set` and `get` methods for this private field as show in the following code segment:

```
public class TestSimpleTypes
{
    private boolean testBool;
    public boolean getTestBool(){
        return testBool;
    }
    public void setTestBool(boolean b){
        testBool = b;
    }
}
```

- Interface type fields will be converted if JavaBean style get and set property methods for the interface are available in the class. You cannot convert from XML back to an interface. You can, however, convert back to a class that implements a interface.

Related Topics

To learn more about creating Java class variables in business processes, see Creating Variables.

To lean more about using Java classes as input or output parameters in transformations, see "Using Java Classes in Transformations" on page 2-42.

Using the MflObject Interface to Transform Non-XML Data Programmatically

When an MFL is imported into a Schema project folder, a corresponding MflObject Java class is generated. The name of the generated Java class file is derived from the `MessageFormat` name specified in the MFL file as shown in the following example `StockQuotes.mfl` file:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name='StockPrices' version='2.01'>
    <StructFormat name='PriceQuote' repeat='*'>
        <FieldFormat name='StockSymbol' type='String' delim=':'
codepage='windows-1252'/>
        <FieldFormat name='StockPrice' type='String' delim='|'
codepage='windows-1252'/>
    </StructFormat>
</MessageFormat>
```

For example, if the preceding `StockQuotes.mfl` file, which specifies the `MessageFormat` name of `StockPrices` is imported into the Schemas project folder, a `StockPricesMflObject.class` file is generated with following methods:

```
public final class StockPricesMflObject extends MflObject {
public StockPricesDocument convertToXml() {}
public static StockPricesMflObject newInstance(StockPricesDocument xml) {}
public static StockPricesMflObject newInstance(byte[] bytes) {}
public static StockPricesMflObject newInstance(InputStream in) {}
public static StockPricesMflObject newInstance(String in) {}
public static StockPricesMflObject newInstance(RawData in) {}
}
```

You can use these methods to programmatically convert non-XML data to and from XML data outside the mapper functionality of WebLogic Workshop. The following table lists the available MflObject methods and describes functionality provided with each method. In the following table, *MFName* represents the specified `MessageFormat` name. (In the preceding example, the specified MessageFormat name was `StockPrices`.)

Method Name and Signature	Functionality
public *MFName*Document convertToXml()	Transforms the non-XML data, valid to the schema in the MFL file, to an instance of the associated XMLBeans Java interface. The XMLBeans Java interface can then be used to access this data in the XML document. To learn more about XMLBeans, see Getting Started with XMLBeans.
	For an example of using this interface see "Transforming Non-XML Data to Typed XML" on page 6-11.
public static *MFName*MflObject newInstance(*MFName*Docume nt xml)	Transforms an instance of the associated XMLBeans (XML data) to non-XML data that is valid to the schema in the MFL file.
public static *MFName*MflObject newInstance(byte[] bytes)	Transforms byte non-XML data to an instance of the MflObject

Method Name and Signature	Functionality

`public static`
`MFNameMflObject`
`newInstance(InputStream`
`in)`
	Transforms the non-XML input stream to an instance of the MflObject.

`public static`
`MFNameMflObject`
`newInstance(String in)`
	Transforms non-XML data as a string to an instance of the MflObject.

`public static`
`MFNameMflObject`
`newInstance(RawData in)`
	Transforms non-XML data as raw data to an instance of the MflObject.

The following section shows three different examples of using the `StockPricesMflObject` class:

- Transforming Non-XML Data to Typed XML

- Create a New Instance of an MflObject From Typed XML Example

- Create a New Instance of an MflObject From Untyped Raw Data Example

Transforming Non-XML Data to Typed XML

The example in this section shows a business process programmatically converting an incoming typed non-XML message to an typed XML representation of that data.

This example assumes the `StockPrice.mfl` file is imported into a Schemas project folder and the following XML Bean classes were generated:

- `PriceQuoteDocument`

- `PriceQuoteDocument.PriceQuote`

- `StockPricesDocument`

- `StockPricesDocument.StockPrices`

To learn more about the Java classes that are generated when MFL files are imported, see "Java Classes Created From Importing Schemas" on page 6-1. For a listing of the `StockPrice.mfl`, see "Java Classes Created From Importing Schemas" on page 6-1.

The business process executes the following steps:

1. A **Client Request** start node receives an non-XML message of type: `stockquotes.StockPricesMflObject` and stores in a typed non-XML variable called `stockPriceIn`.

2. In a **Perform** node of the business process following the **Client Request** start node, the typed non-XML data is transformed to the XML Beans representation of the data by calling the `convertToXml` function as shown in the first line of the `perform` method of the following Process Definition for Java (JPD) code listing:

```
package processes;
public class convertToXMLExample implements com.bea.jpd.ProcessDefinition
{
    public stockquotes.StockPricesDocument stockPriceXML;
    public stockquotes.PriceQuoteDocument.PriceQuote priceQuoteXML;
    public stockquotes.StockPricesMflObject stockPriceIn;
...
    public void perform() throws Exception
    {
        stockPriceXML = stockPriceIn.convertToXml();
        priceQuoteXML = stockPriceXML.getStockPrices().addNewPriceQuote();
        priceQuoteXML.setStockPrice("10.99");
    }
```

3. The second line in the `perform` function executes the following steps:

   a. From the XML representation of the non-XML data, the `stockquotes.StockPricesDocument` XML Beans class gets the current instance, as shown in the following section of code:

      ```
      stockPricesXML.getStockPrices()
      ```

   b. Adds a new instance of `PriceQuote` using the `addNewPriceQuote` method of the `stockquotes.StockPricesDocument` XML Beans class, as shown in the following section of code:

      ```
      stockPricesXML.getStockPrices().addNewPriceQuote()
      ```

4. The last line in the `perform` function sets the StockPrice element to the string: `10.99` using the `setStockPrice` method of the `stockquotes.StockPricesDocument` XML Beans class, as shown in the following section of code:

   ```
   priceQuoteXML.setStockPrice("10.99");
   ```

Create a New Instance of an MflObject From Typed XML Example

The example in this section shows a business process creating a new instance of an MflObject from incoming typed XML data.

This example assumes the `StockPrice.mfl` file is imported into a Schemas project folder and the following XML Bean classes were generated:

- `PriceQuoteDocument`

- `PriceQuoteDocument.PriceQuote`

- `StockPricesDocument`

- `StockPricesDocument.StockPrices`

To learn more about the Java classes that are generated when MFL files are imported, see "Java Classes Created From Importing Schemas" on page 6-1. For a listing of the `StockPrice.mfl`, see "Java Classes Created From Importing Schemas" on page 6-1.

The business process executes the following steps:

1. The **Client Request** start node receives an XML message of type: `StockPricesDocument` and stores in a typed XML variable called `stockPriceXML`.

2. In the perform node, an instance of the `stockquotes.StockPricesMflObject` is created from the typed XML data by calling the static `newInstance` function which accepts a parameter of type: `StockPricesDocument` as shown in the following Process Definition for Java (JPD) code listing:

```
package processes;
public class newInstanceFromXMLExample implements com.bea.jpd.ProcessDefinition
{
    public stockquotes.StockPricesDocument stockPriceXML;
...
    public void perform() throws Exception
    {
        stockquotes.StockPricesMflObject stockPriceMFLObj =
stockquotes.StockPricesMflObject.newInstance(stockPriceXML);
    }
}
```

Create a New Instance of an MflObject From Untyped Raw Data Example

The example in this section shows a business process creating a new instance of an MflObject from incoming raw data (a stream of non-XML data that is untyped and has no known structure).

This example assumes the `StockPrice.mfl` file is imported into a Schemas project folder and the following XML Bean classes were generated:

- `PriceQuoteDocument`
- `PriceQuoteDocument.PriceQuote`
- `StockPricesDocument`
- `StockPricesDocument.StockPrices`

To learn more about the Java classes that are generated when MFL files are imported, see "Java Classes Created From Importing Schemas" on page 6-1. For a listing of the `StockPrice.mfl`, see "Java Classes Created From Importing Schemas" on page 6-1.

The business process executes the following steps:

1. The **Client Request** start node receives an untyped non-XML data message and stores in a **RawData** variable called `stockPriceRaw`.

2. In the perform node, an instance of the `stockquotes.StockPricesMflObject` is created from the untyped non-XML data by calling the static `newInstance` function which accepts a parameter of type: `RawData` as shown in the following Process Definition for Java (JPD) code listing:

```
package processes;
public class newInstanceFromXMLExample implements com.bea.jpd.ProcessDefinition
{
    public com.bea.data.RawData stockPriceRaw;
...
    public void perform() throws Exception
    {
        stockquotes.StockPricesMflObject stockPriceMFLObj =
stockquotes.StockPricesMflObject.newInstance(stockPriceRaw);
    }
}
```

# Getting the TransformException Fault Code Programmatically

In the mapper functionality of WebLogic Workshop, you create transformations that transform data from one format to another. From these transformations, queries (written in the XQuery)

language are generated. You can use the mapper functionality to edit these queries to add invocations to standard XQuery functions and operators, User functions, or Controls functions. To learn more, see Invoking Functions or Operators in a Query.

During run time, these queries may throw a `TransformException` exception with an associated fault code. (For example, your query might call one of the provided standard XQuery functions, which may throw an exception.) The fault code provides information about why the `TransformException` was thrown. You may want to add Java code to business process to get the fault code of the `TranformException` and do some action in the code based on the fault code. You may also display the description string associated with the fault code in the console using a `System.out.println` function. You add this code to the Process Definition for Java (JPD) file which contains the Java code for the business process.

Note: If you are only interested in the fault code string, use the **Exception** pane of the **Workshop Test Browser** or the **Test View** pane of the mapper functionality of WebLogic Workshop to display the error string associated with the fault code.

For example, a business process whose **Start** node is a **Client Receive** node invokes a Transformation method of a Transformation file. The **Client Receive** node has an exception path with a **Perform** node to catch exception, as shown in the following figure:
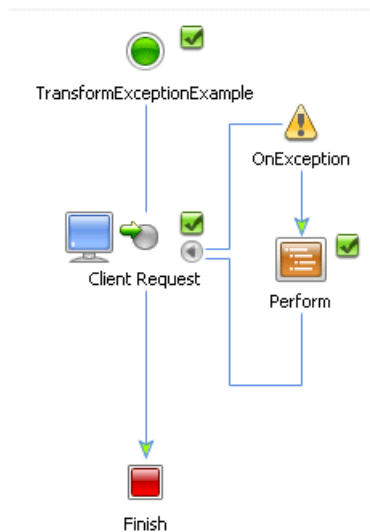
The Transformation method invokes a query, which invokes the standard XQuery function `xf:date`. During run time when the query is executed, the `date` function is invoked with the string: 2003-8-16. This string is not a legal date format because the month is not specified with

two digits. This causes the `date` function to thrown a `TransformException` exception with the `TransformFaultCodes.RT_ILLEGAL_CAST` fault code. The **Perform** node retrieves the information about the exception that invoked it, checks that the exception is a TransformException, and then prints out fault code number and sting that describes the fault code, as shown in the following JPD code segment:

```
import com.bea.jpd.JpdContext;
import com.bea.data.RawData;
import com.bea.xml.XmlObject;
import com.bea.transform.TransformException;
import com.bea.transform.TransformFaultCodes;
...
public class TransformExceptionExample implements com.bea.jpd.ProcessDefinition
{
    ...
    public void perform() throws Exception
    {
        Exception e;
        com.bea.transform.TransformException te;
        e = this.context.getExceptionInfo().getException();
        System.out.println("Caught exception in transformation: " +
e.toString());
        if (e instanceof TransformException)
        {
            te = (TransformException) e;
            System.out.println("Fault Code Number=" + te.getFaultCode());
            System.out.println("Fault Code String=" + te.getCause());
             switch (te.getFaultCode()) {
                    case TransformFaultCodes.RT_ILLEGAL_CAST:
                    // Add code here to do some action based on the error code.
                    break;
                default:
                    break;
            }
        }
    }
}
```

For example, if some corrective action should occur if the returned fault code is equal to the `TransformFaultCodes.RT_ILLEGAL_CAST` fault code, replace the comment that starts with the string: `//Add code here` with the corrective Java code.

For a list of the supported fault codes, see the JavaDoc for the `TransformFaultCodes` class.

Note: The following two import lines must be added manually in the **Source View** of the JPD file:

```
import com.bea.transform.TransformException;
import com.bea.transform.TransformFaultCodes;
```

Using the com.bea.WLXT Package (Deprecated)

The public methods provided in the `com.bea.WLXT` package are deprecated. This deprecated package will be removed from the product in a future release. You should migrate your application away from using this package to using the functionality provided in WebLogic Integration 8.1 for transforming data between non-XML and XML formats. To learn more, see "Transforming Non-XML Data" on page 4-1.