

XSLT Essentials

XSLT Essentials



Overview

Introduction

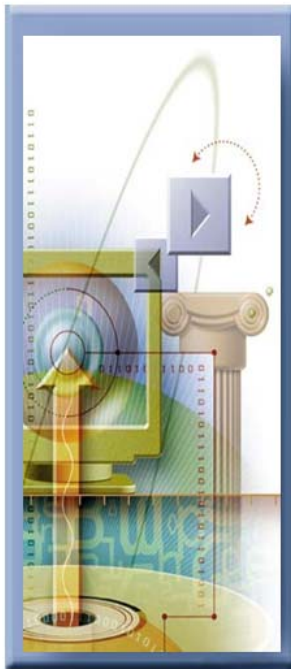
In order to use the XML Transformation Service of the ESB or any other XSLT processor in your application, you first need to learn the language used for transforming XML documents, called the eXtensible Stylesheet Language(XSL). XSLT is the transformation processing that is done using this language.

This lesson will teach you the basic syntax of XSL. You will learn how to create an XSL file called an XSLT stylesheet which describes the XML transformations from one XML document to another. You will use Stylus Studio to create and test your XSLT stylesheets. You also will learn how to use the XML-to-XML mapping capability of Stylus Studio which enables you to generate an XSLT stylesheet.

This lesson is not intended to be a complete XSLT course, but rather one that gets you started with the basics of XSLT. If you want to learn more about XSLT, you should read any of the many books that address XSLT development.

continued on next page

Overview



When you complete this lesson you should be able to:

- Describe why transformation is necessary
- Describe what XSL is
- Create XSLT stylesheets with Stylus Studio
- Describe what XSLT processing is
- Define some useful XSL elements in your XSLT stylesheets
- Test XSLT stylesheets that you have developed
- Create and test an XML to XML mapping stylesheet

Overview, continued

Learning objectives

When you complete this lesson, you should be able to:

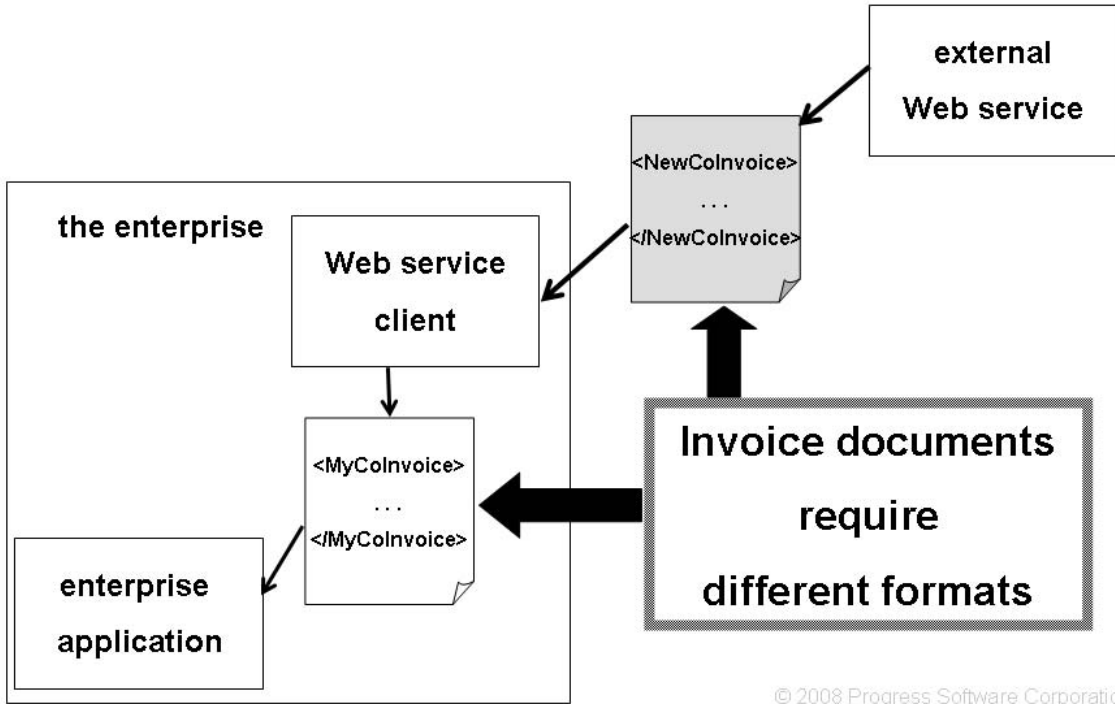
- Describe why transformation is necessary,
 - Describe what XSL is,
 - Create and test a simple XSLT stylesheet using Stylus Studio,
 - Describe what XSLT processing is,
 - Define some useful XSL elements in your XSLT stylesheet,
 - Test an XSLT stylesheet that you have developed and
 - Create and test XML to XML mapping with Stylus Studio.
-

Prerequisites

Before you begin this lesson, you should be able to:

- Understand XML document structure and
 - Create XPath expressions.
-

Need for XML transformation



3

© 2008 Progress Software Corporation

Understanding the need for XML transformation

Introduction

Your application may need to exchange information with a single application or partner. Your application also may need to participate in a business process where there may be multiple exchanges of information. For both of these scenarios, the format used for exchanging data will most likely be XML. For example, Web services require that the messages be exchanged using the SOAP messaging protocol, defined by XML Schema.

Why transform?

The information exchanged needs to be transformable. This is necessary because:

- The information your application requires may be in a different format than the format of the partner with whom you are exchanging information.
 - You may want to switch partners; the ability to “plug in” a different partner or service provider will most likely require a change in the format of the information you are exchanging.
-

Example of the need for transformation

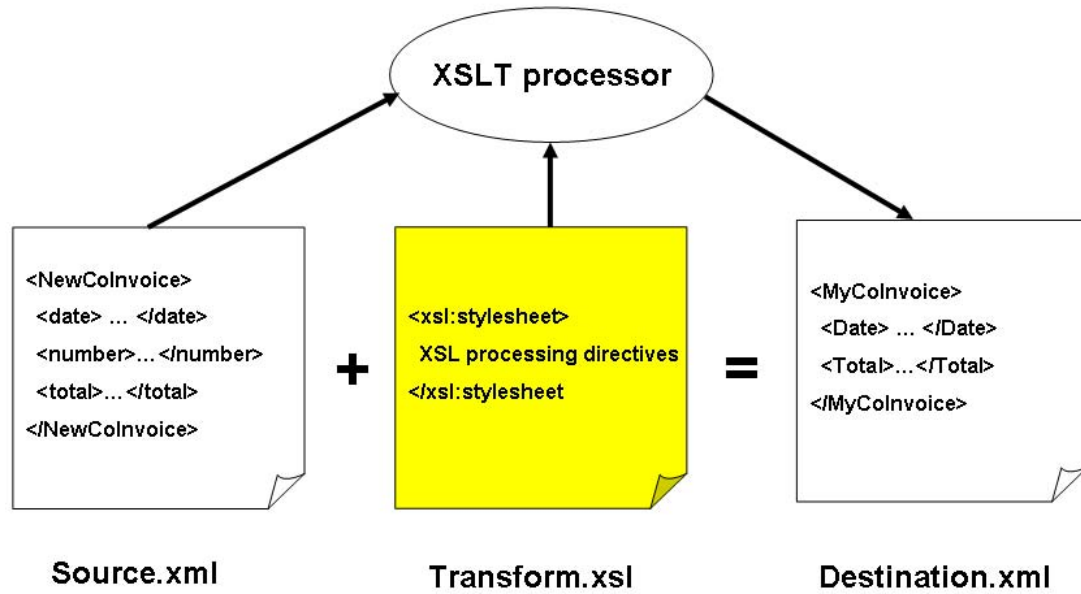
Let us assume that you have an application that utilizes a Web service which returns an Invoice XML document. The format used to describe this XML document is **Invoice-NewCo.43.2.xsd**. This is an XML Schema file the Web service provider has published that fully describes the format for the *Invoice* XML document that will be returned by the *NewCo* Web service.

Your application has its own format that it requires for an *Invoice* XML document once it has been received by your *MyCo* application. This format is described by the **MyCo.xsd** file. If the XML Schema for *NewCo* and *MyCo* for an *Invoice* XML document are not identical, then your application needs to turn the *Invoice* XML document received from *NewCo* into an *Invoice* XML document that your enterprise (*MyCo*) application will understand. This transformation is done using an XSLT processor.

XSLT processing in Sonic ESB

XSLT processing is available to services that run in a Sonic ESB container. In this lesson, you will learn some of the basics of creating the XSLT stylesheet used by an XSLT processor to transform an XML document into a different format.

What is XSL?



What is XSL?

Introduction

The eXtensible XSLT Stylesheet Language (XSL) was developed and standardized by a number of participants in the W3C (www.w3c.org). It is written in XML and adheres to the XML Schema for XSL, which has been defined by the W3C.

The XML elements defined in an XSLT stylesheet are used to tell the XSLT processor how to take a source XML document and turn it into another output format (typically XML). When an XSLT stylesheet is applied to a particular source XML document, the XSL elements are used to control the XSLT processing. The Sonic ESB container and Stylus Studio have built-in XSLT processors.

Elements of XSL

The XML elements in an XSL document are defined by the XML Schema for XSL. The XML Schema for XSL defines a namespace called “xsl”. Every element you create that is an XSL element will be preceded by the “xsl:” namespace identifier.

The types of tasks you can perform on a source XML document include:

- Search for element and attribute names and values within the source XML document and copy them from the source XML document to the destination.
- Identify values of elements and attributes that are in the source XML document for modification and write the modified elements to the destination.
- Identify elements and attributes in the source XML document that need to be rearranged and write them in the new order to the destination.
- Add new data to the destination.

In this lesson, we will learn how to perform some of the above tasks by developing and testing an XSLT stylesheet using Stylus Studio.

An XSL document

```
<xsl:stylesheet ...>  
  <xsl:template>  
    ...  
    ...  
  </xsl:template>  
</xsl:stylesheet
```

Structure of an XSL document

Root element

Every valid XSL document must have a root element that is used to tell the XSLT processor that it is an XSLT stylesheet. Since it is the root element, it will have only one *xsl:stylesheet* element.

Template elements

The child elements of the root element of an XSL document are called template elements and have the name *xsl:template*. It is within the template that other XSL elements or literals are placed. The XSLT processor uses the information in the template to create a destination document from the source document. An XSL document can have any number of templates, depending on the transformation processing required.

Syntax for the *xsl:stylesheet* element

Here is the syntax for the *xsl:stylesheet* element:

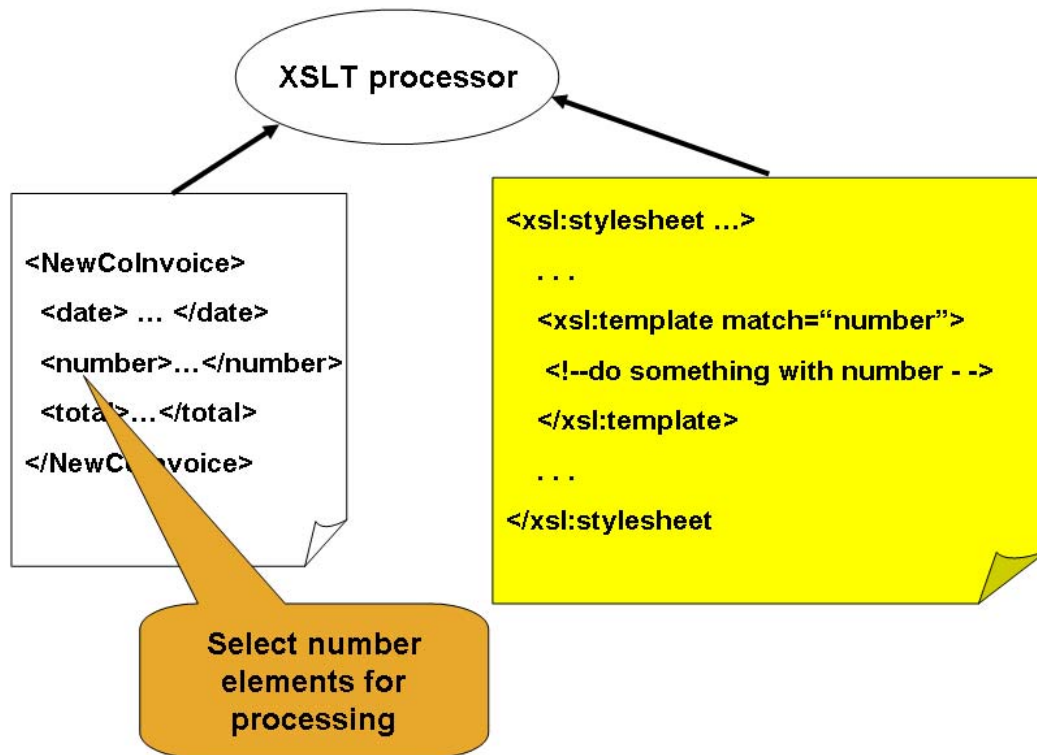
Syntax

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
<!--The rest of the XSLT stylesheet goes here -->  
</xsl:stylesheet>
```

XSLT version compatibility

You should always use the above syntax for your *xsl:stylesheet* element. The namespace and version for XSLT has not been changed by the W3C since 1999. If it changes in the future, then you must ensure that whatever XSLT processor you are using conforms to the namespace and version that you have specified in your *xsl:stylesheet* element.

XSL templates for selecting elements



6

© 2008 Progress Software Corporation

Understanding XSLT stylesheet elements

XSLT template

An XSLT template element is used to select part of a source XML document for transformation. The template typically has two parts: the match attribute, which is used to select the elements or attributes of the XML document, and the processing directive, which tells the XSLT processor what to do with the selected elements or attributes. It is the XSLT templates that comprise your XSLT stylesheet.

continued on next page

A simple XSLT stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <Hello>This is a test</Hello>
  </xsl:template>
</xsl:stylesheet>
```

Understanding XSLT stylesheet elements, *continued*

Syntax for the `xsl:template` element

An XSLT stylesheet may contain one or more templates for controlling the XSLT processing. It is within the template that you define what processing you want the XSLT processor to perform when it encounters a match in the source XML document. Here is the syntax for the `xsl:template` element:

Syntax

```
<xsl:template match="XPathExpression">
<!--The XSLT processing directives for this template goes here -->
</xsl:template>
```

where:

XPathExpression is an expression that the XSLT processor will use to match against the source XML document to see if processing for matched portion of the document needs to be performed.

The processing directives are `xsl` elements. You use these `xsl` elements to tell the XSLT processor what processing to perform with the matched content. You will learn more about the `xsl` elements available to you later in this lesson.

Example of an `xsl:template` element

Here is an example of a template that matches the root element of the source XML document denoted by the match string of `"/`. In XPath, `"/` always means the root element of the XML document. In the following example, the XSLT processor finds the root element in the source XML document then executes the directives in the body of the template:

```
<xsl:template match="/">
<!--The XSLT processing directives for this template goes here -->
</xsl:template>
```

Example XSLT stylesheet

The example on the facing page is a very simple XSLT stylesheet that directs the XSLT processor to write the complete *Hello* element to the destination. Anything in the template which is not an XSL element is considered a literal. No specific elements from the source XML document are selected so none of the source XML elements are written to the destination. We will next learn how to create an XSLT stylesheet in Stylus Studio.

Demonstration 1: Creating your first XSLT stylesheet



8

© 2008 Progress Software Corporation

Demonstration 1: Creating your first XSLT stylesheet

Introduction

In this demonstration, you will use Stylus Studio to create a very simple XSLT stylesheet that creates an XML document. It will look for the root of the **bookstore.xml** document and then commence XSLT processing. You also will create the XSLT stylesheet that writes the *Hello* element to the destination XML document.

Here are the steps to create and test the XSLT stylesheet:

Step	Description
1.	Start Stylus Studio.
2.	Create a new XSLT stylesheet.
3.	Create a scenario which is an environment for doing the transformation.
4.	Edit the XSLT stylesheet.
5.	Save your XSLT stylesheet.
6.	Test your XSLT stylesheet.
7.	View the resulting destination document.

Starting Stylus Studio

Start Stylus Studio by selecting Start→Programs→Stylus Studio 6 XML Home Edition→Stylus Studio.

Creating an XSL file (XSLT stylesheet)

Create a new XSLT stylesheet by selecting File→New→XSLT:Text Editor.

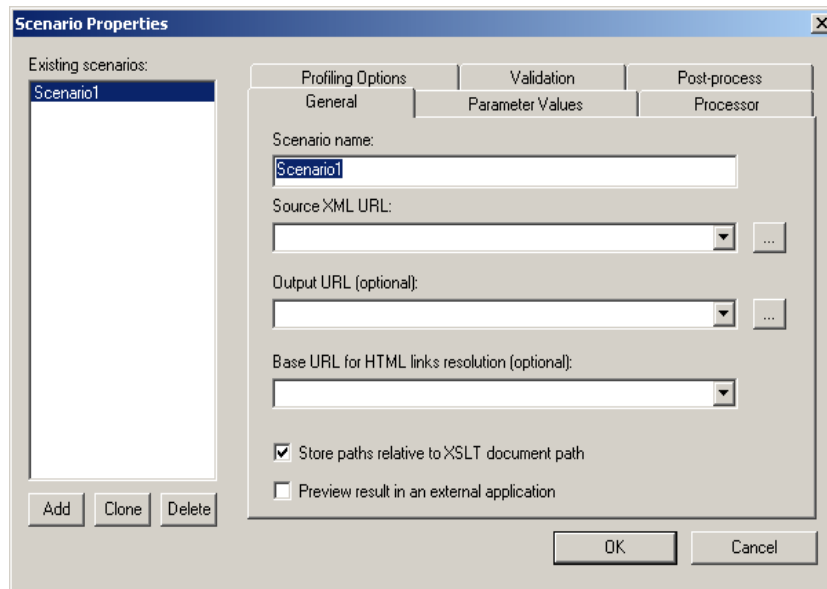
When you create a new XSLT stylesheet, you will eventually receive the edit pane as shown later in this demonstration. It is within this pane that you can create the XSLT processing directives (XSLT template content) that define your XSLT stylesheet.

continued on next page

Demonstration 1: Creating your first XSLT stylesheet, continued

Creating an XSL file (XSLT stylesheet), continued

The dialog box you should now see looks as follows:



continued on next page

Demonstration 1: Creating your first XSLT stylesheet, continued

What is a scenario?


A scenario is a name that is associated with a group of documents:

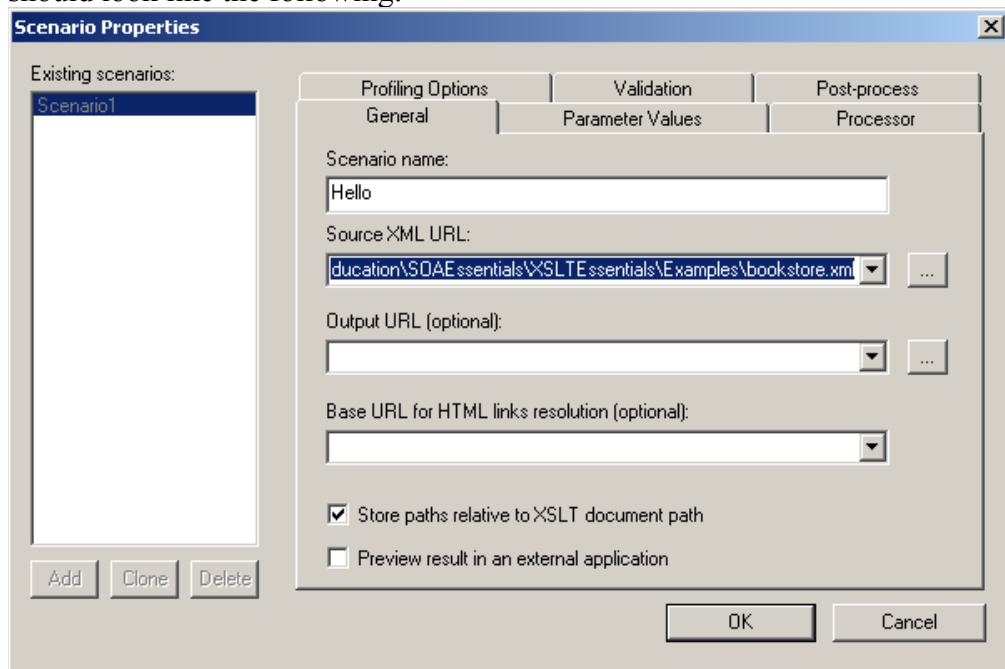
- XML source document,
- XSL document (XSLT stylesheet) and
- Output document (optional).

In order to test any XSLT processing in Stylus Studio, you must create a scenario.

Creating a scenario

You should enter the following information in the Scenario Properties window:

1. The scenario name is something that helps to identify the XSLT stylesheet. Name it Hello.
2. For your Source XML URL, select  to browse to the **c:\progress_education\SOAEssentials\XSLTEssentials\Examples\bookstore.xml** file. You can leave the Output URL information blank. Your new scenario should look like the following:

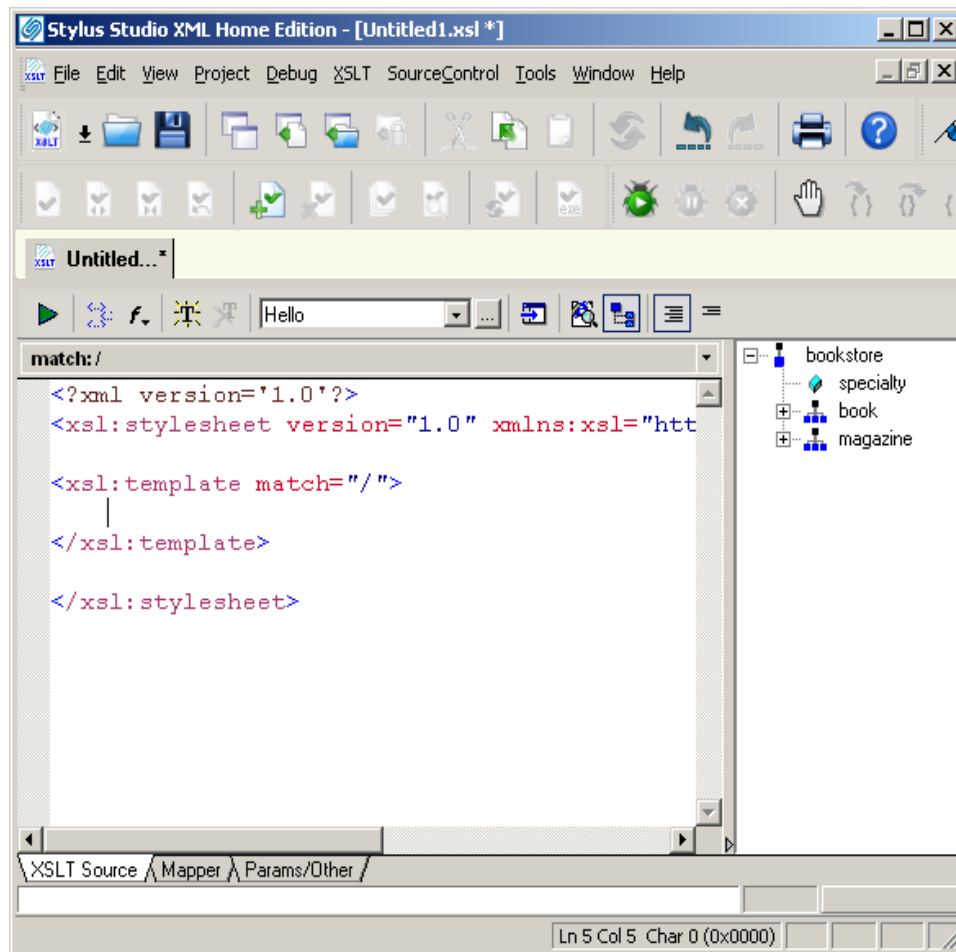


3. Select OK.

Demonstration 1: Creating your first XSLT stylesheet, continued

Creating a scenario, continued

You will then be in the XSLT Editor and see something like the following:



Notice how Stylus Studio uses color to help you identify the XML constructs in the XSL file. Notice, too, that it has created a template with no content.

continued on next page

Demonstration 1: Creating your first XSLT stylesheet, continued

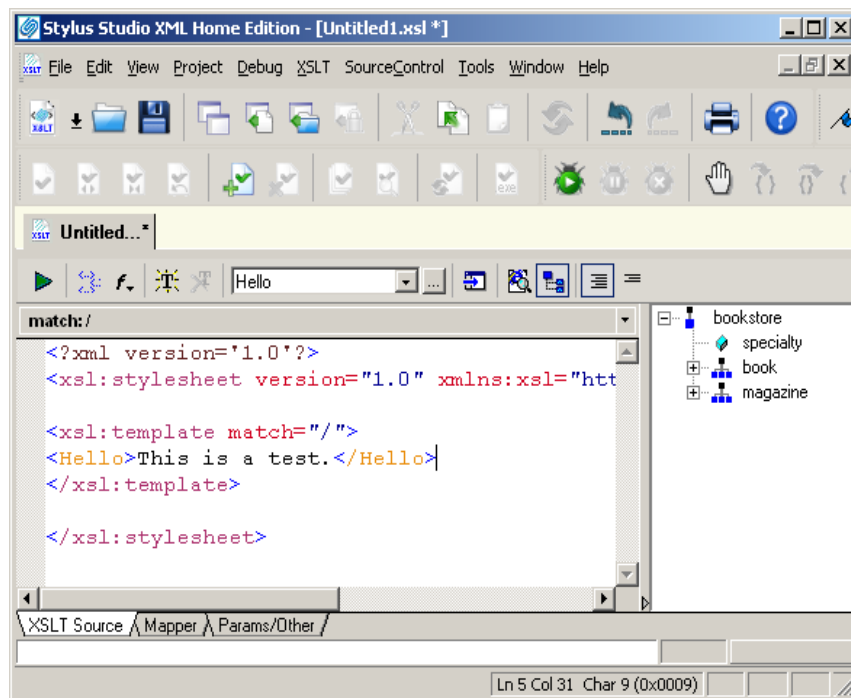
Editing the XSLT stylesheet

With the XSLT stylesheet in view, you now have XSLT stylesheet editing capabilities. Within this template that matches “/”, add the following element:

```
<Hello>This is a test</Hello>
```

Make sure that this element is within the `xsl:template` element.


Your XSLT stylesheet should look like the following:



Notice that the right-hand pane displays the XML Schema of the source XML document in tree format. This can help you when creating content for the XSLT stylesheet.

Saving your XSLT stylesheet

You need to save your XSLT stylesheet before you can test it.


1. Select the  icon or select File → Save.
2. Name your XSLT stylesheet **hello.xml**.

continued on next page

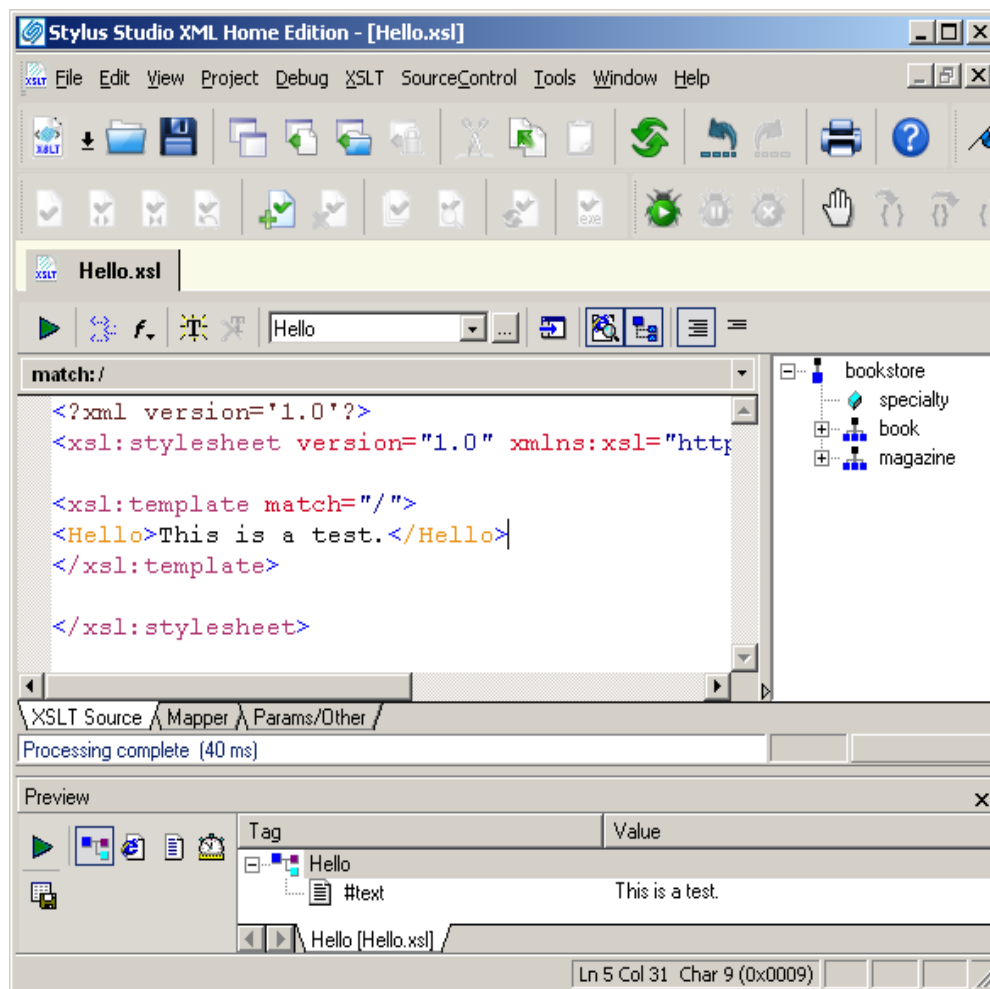
Demonstration 1: Creating your first XSLT stylesheet, continued

Testing your XSLT stylesheet

Testing your XSLT stylesheet involves invoking the XSLT processor that is built into Stylus Studio. You test your XSLT stylesheet by applying the stylesheet to the source

XML document. You do so by selecting the preview button (the green arrow  to the left of the scenario name pane).

When the XSLT processing completes, you should see the following:



The preview pane at the bottom of your window should show the result of the XSL transformation.

continued on next page




Demonstration 1: Creating your first XSLT stylesheet, continued

Viewing the destination XML document

When you have applied an XSLT stylesheet to your source XML document as specified in your scenario, a third pane is displayed (called the Preview pane) that you saw on the previous page.

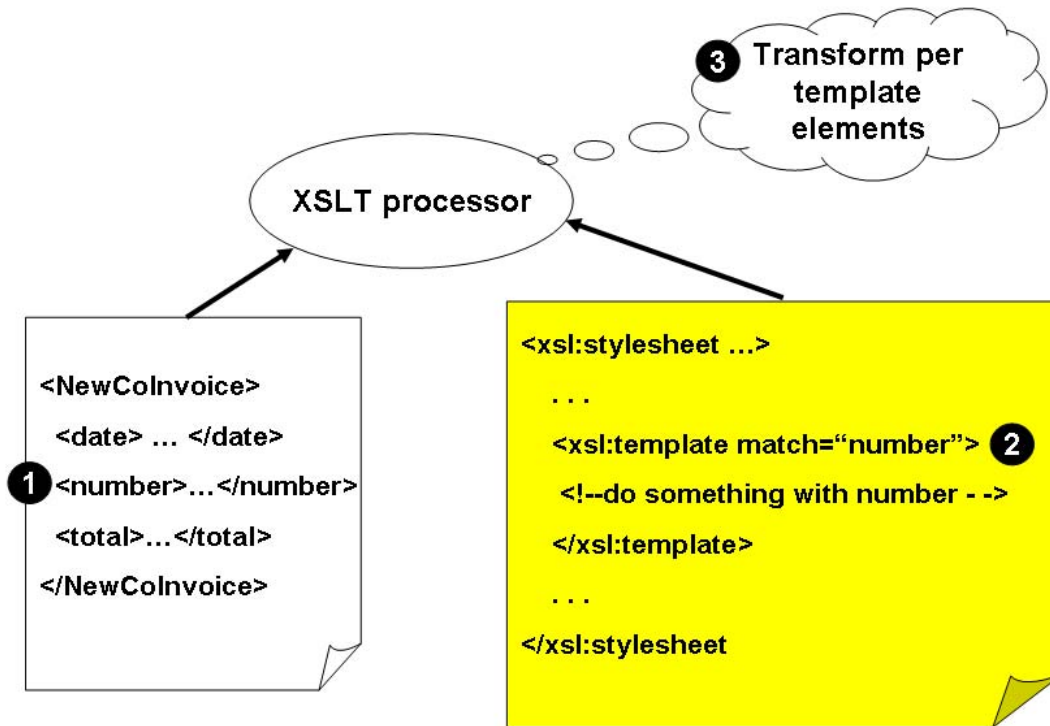
This is the result of performing the XSLT processing on your source **bookstore.xml** document using the **hello.xsl** XSLT stylesheet that you just created.

Notice that in the Preview pane, there are icons on the left to select which presentation style you would like to see:

Stylus Studio icon	Description
	Tree view
	IE view
	XML text view

Try each of these views for viewing your output.

XSLT processing



9

© 2008 Progress Software Corporation

Understanding XSLT processing

Introduction

Thus far you have seen a simple template that is processed when the XSLT processor encounters the root “/” of the document. Recall that a XSLT template contains a *match* element and the elements that direct the XSLT processor. You will learn more about how XSLT processing works and how templates are used during XSLT processing.

Matching templates

Earlier in this lesson, you saw what a simple *xsl:template* element looks like in an XSLT stylesheet. This element will always have a *match* attribute. The *match* attribute is an XPath expression that identifies a node (elements or attributes) or set of nodes that are to be processed with this template. If there are nodes in the current context which match the template, then the processing defined in the template is performed.

The template we saw earlier had the *match* attribute value of “/”. Here is an example of a template with a *match* attribute for the *book* element:

```
<xsl:template match="book">
<!--The XSLT processing directives for a book element -->
</xsl:template>
```

Template instantiation

A template in the XSLT stylesheet is instantiated in XSLT processing when a particular node (element or attribute) has been encountered in the parse of the source XML document that matches the *match* attribute of an existing template in the XSLT stylesheet. Once found, the XSLT processing directives in the matched template are executed.

continued on next page

Default XSL template elements

```
<xsl:template match="* | /">  
  <xsl:apply-templates/>  
</xsl:template>
```

```
<xsl:template match="text()">  
  <xsl:value-of select="."/>  
</xsl:template>
```

Understanding XSLT processing, continued

Default templates

Every XSLT processor has default behavior defined by the two templates shown on the facing page. What this means is if you have not defined a template for a particular element or attribute match, then the default processing will occur.

Although we have not learned about some of the XSL element syntax for these default templates (namely *xsl:apply-templates* and *xsl:value-of*), you should understand the default behavior for the XSLT processor.

The first template shown on the facing page matches any element (*) or the root (/) of the source XML document. It then continues applying these default templates (using *xsl:apply-templates*) for the child elements.

The second template matches any text nodes encountered in the source XML document. It then writes the values of the text nodes to the destination document. What is important to understand with this default template is that the element and attribute names are not written to the destination document; only the values are written.

Default behavior of the XSLT processor

The result of applying an XSLT stylesheet to a source XML document using only the default behavior will be to take all of the values of elements and attributes from the source XML document and write them to the destination document. If you have not defined any templates to override the default templates, this is the behavior of the XSLT processor you will see.

Next you will try applying an XSLT stylesheet in Stylus Studio using the default templates and see the result.

Demonstration 2: Using the default templates



11

© 2008 Progress Software Corporation

Demonstration 2: Using the default templates

Introduction

The purpose of this demonstration is for you to see the result of using default templates. In the previous demonstration, you created an XSLT stylesheet defining the template that matched “/”. In this demonstration, you will create an XSLT stylesheet that utilizes the default templates of the XSLT processor. Then you will add templates to override the default templates.

Create an XSLT stylesheet without a template defined

Follow these steps for creating the template:

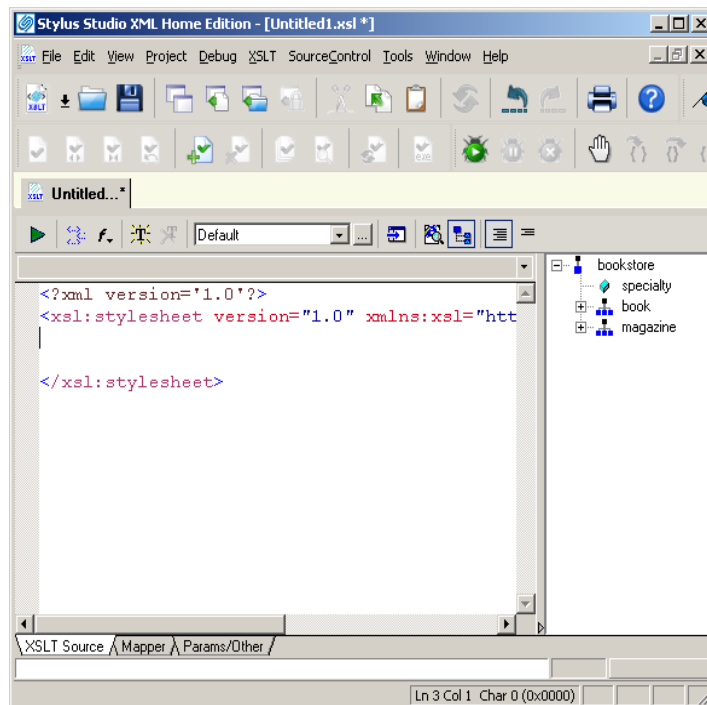
1. Open Stylus Studio.
 2. Create a new XSLT stylesheet by selecting File→New→XSLT: Text Editor.
 3. Define a new scenario with the name of **Default**. For your Source XML URL, select the **bookstore.xml** file.
 4. You can leave the Output URL information blank.
 5. Select OK.
-

continued on next page

Demonstration 2: Using the default templates, continued

Remove the template defined

In the XSLT stylesheet editing pane, select and delete the template that Stylus Studio automatically defines for you which matches “?”. Your XSLT stylesheet should now look like the following with no templates explicitly defined:




Save your XSLT stylesheet

Save your XSLT stylesheet as **default.xsl**.

continued on next page

Demonstration 2: Using the default templates, continued

Try the XSLT processor

Select the Preview icon  to perform the XSLT processing on your source XML document. You can view the results of the XSLT processing by opening the Preview window.

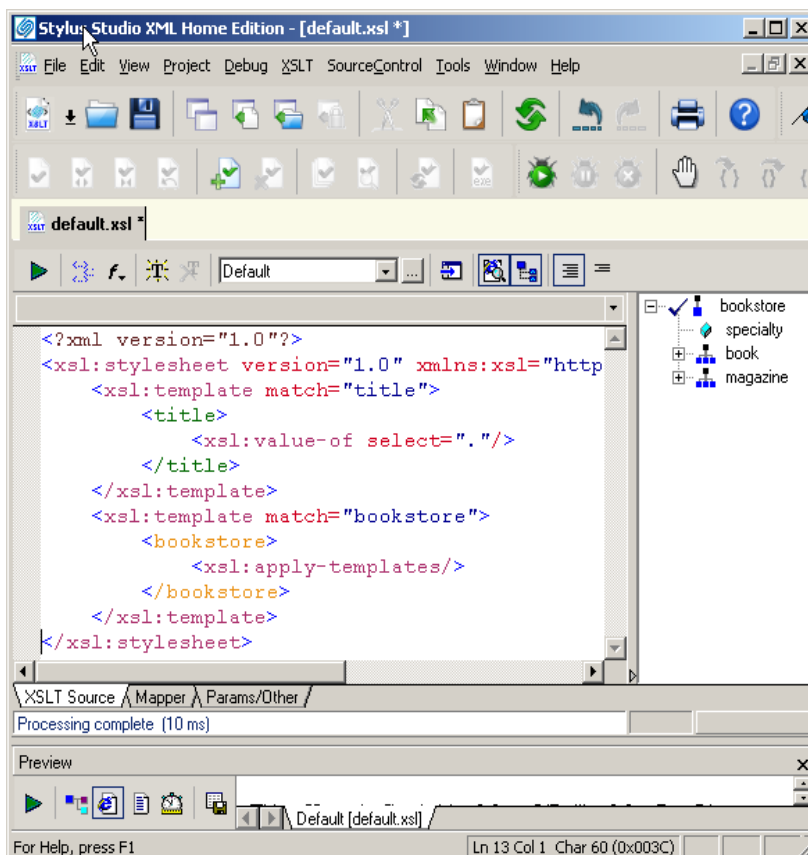
Notice that the output simply has the content of all text nodes and attributes in the source XML document, **bookstore.xml** and that the output is not XML. Using the default templates, you have applied the default XSLT processing directives to the source XML document. No XML tags have been output because the default template only specifies the values of the nodes, not their tags. If you want tags to be output, you must add the output of literal element start and end tags in the template being processed.

continued on next page

Demonstration 2: Using the default templates, continued

Add some templates to your XSLT stylesheet

Add the following templates to your XSLT stylesheet in order to override the default templates the XSLT processor uses:



The first template will output the value of the *title* element text nodes with the *title* XML tags around them. The second template matches the root element of the source XML document and places the root element tags in the output. Notice that we have specified the *xsl:apply-templates* element because we require that all child nodes of this root element be processed. If we leave this XSLT processing directive out, the child elements will not be processed.

When the XSLT Processor executes using the above stylesheet, it parses the elements in the source XML document and checks every element and attribute in the XSLT stylesheet for a template match. The processing sequence is based upon the order of the elements in the source XML document and not the order of the templates in the XSLT stylesheet.

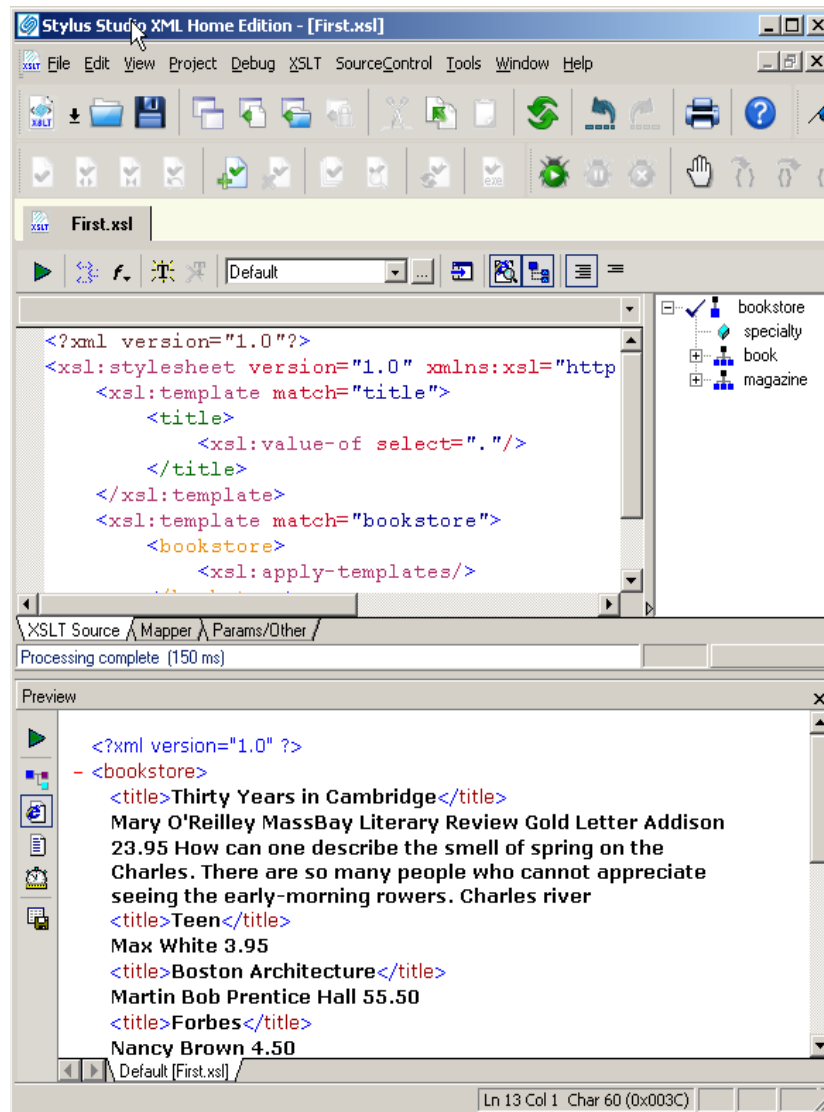
continued on next page

Demonstration 2: Using the default templates, continued

Save and preview your XSLT stylesheet

Save your stylesheet as **First.xml**.

Preview the result. You should now see the following for output:



Notice that only the *bookstore* and *title* elements are properly formatted as XML and all of the other text and attribute nodes are simply written to the output “as is.” On the left of the Preview pane, you can select whether you want to preview the output using tree view, IE view or text view. Try each of these views.

A default XSLT stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="* | /">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="text() | @*">
    <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

Using basic xsl elements

Introduction

Up to this point, you have created and tested some simple XSLT templates in Stylus Studio. In doing so, you have already used the *xsl* elements:

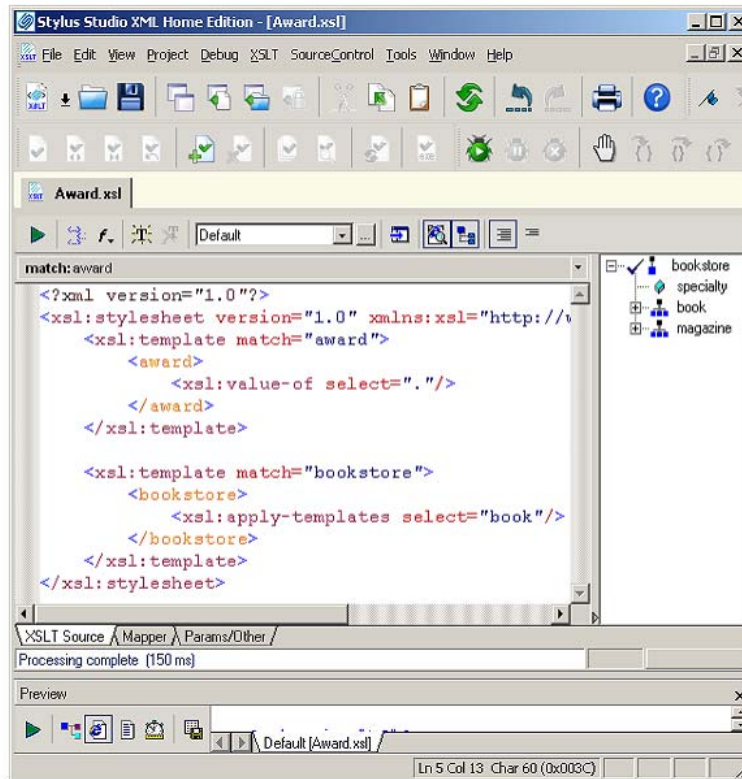
- *xsl:stylesheet*
- *xsl:template*

Within a template, you have used the XSLT processing directives of:

- *xsl:apply-templates*
- *xsl:value-of*

You will now learn more about these *xsl* elements.

Example: Using xsl:apply-templates



13

ess Software Corporation

Using *xsl:apply-templates*

xsl:apply-templates

The *xsl:apply-templates* element tells the XSLT processor to perform more processing by continuing the parse of the source XML document and matching the appropriate templates to apply to the source XML document. You must have this element within the body of at least one template so that if desired, the source XML document can continue to be parsed.

Syntax for *xsl:apply-templates*

Here is the syntax for this element:

Syntax
<code><xsl:apply-templates [select = "expression"]</code>

where:

expression is an XPath expression that identifies a node or nodeset that should be processed that is relative to the current context of the parse.

If the select attribute is not specified, then all child elements of the current context in the parse will be processed. In our previous demonstration, we specified *xsl:apply-templates* with no select attribute which meant that all descendants of the *bookstore* element would be processed.

Example of using *apply-templates*

On the facing page is an example of using *xsl:apply-templates*.

The XSLT processor will parse the bookstore XML document. The first element it encounters is the *bookstore* element. The template match occurs and the processing is done for the matching template. The processing for this template writes the *bookstore* start tag and continues processing for all *book* child elements. For each *book* child element, a match is found for the *award* element and the *award* tag and its value are written. Finally, the end tag for *bookstore* is written.

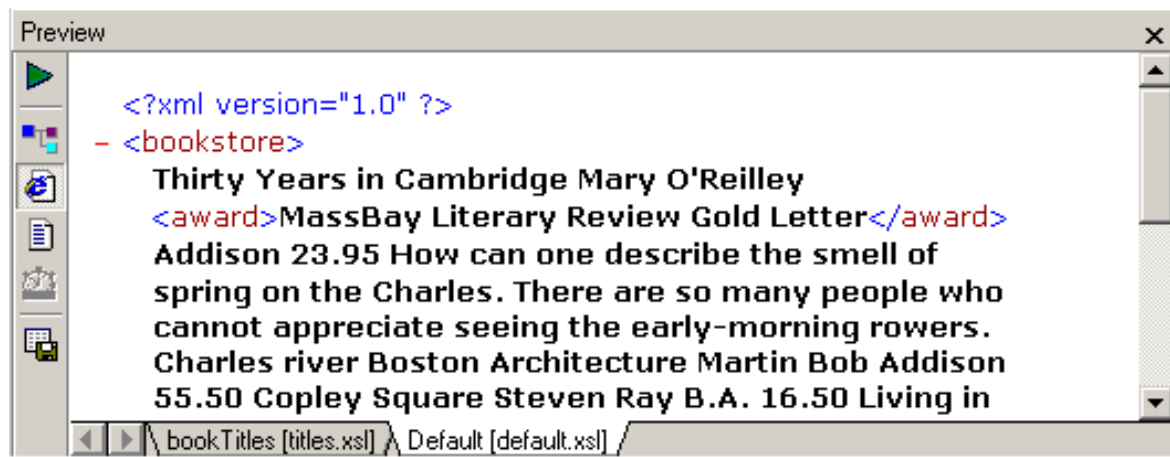
On the next page, you will see the output generated from this XSLT stylesheet.

continued on next page

Using *xsl:apply-templates*, continued

Output generated by XSLT processor

The XSLT processing output for the XSLT stylesheet on the previous page yields the following:



You will notice that in addition to the *award* element, all of the values of elements and attributes for the *book* element are output. This is because the XSLT processor is using the default behavior for everything else which is to output the values of the nodes.

continued on next page

Notes

Selection of nodes by book/author



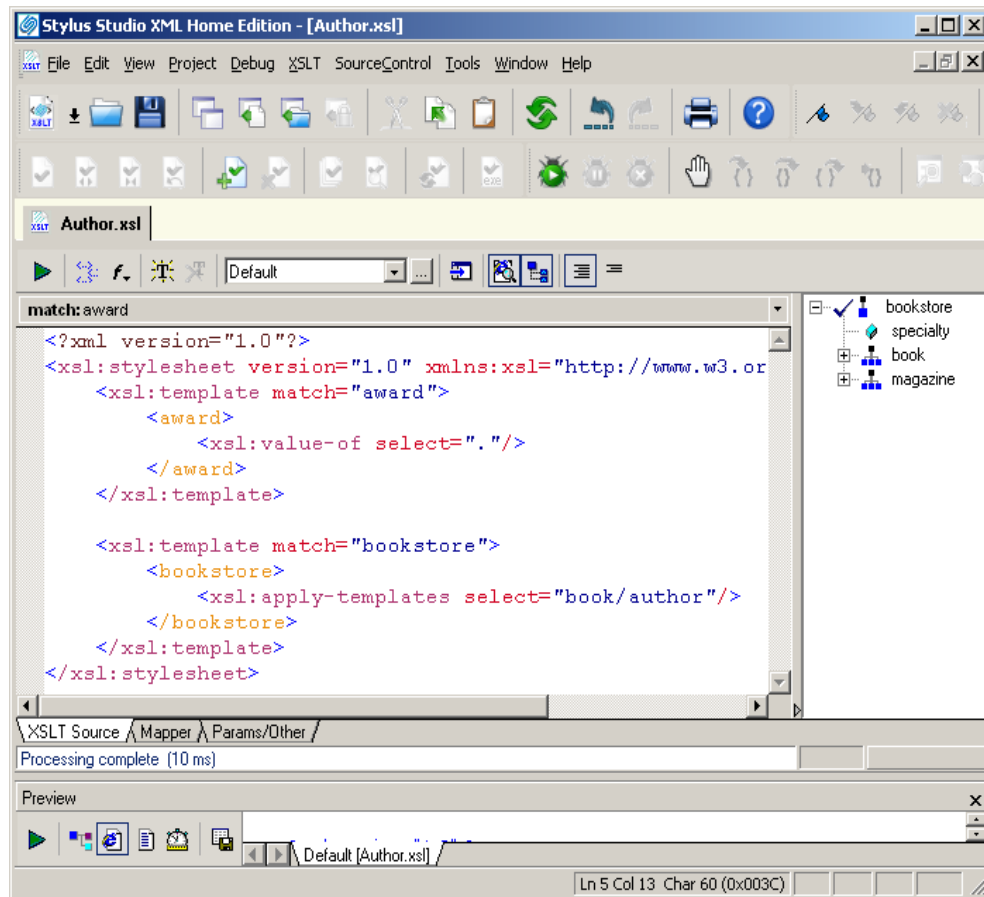
```
<?xml version="1.0" ?>
- <bookstore>
  Mary O'Reilley
  <award>MassBay Literary Review Gold Letter</award>
  Martin Bob Steven Ray B.A. Robert Read
  <award>Harvard Literary Review Honorable
  Mention</award>
  Toni Bob B.A. Ph.D.
  <award>Pulitzer</award>
  Modern Short Stories
</bookstore>
```


Using *xsl:apply-templates*, continued

Narrowing the selection for processing

In the *xsl:apply-templates* element, you can specify a selection that limits the scope of the processing. In the previous example, we specified the book XPath expression to tell the XSLT processor that from the *bookstore* element, apply the template for all *book* elements from this (*bookstore*) context.

If you specify XSLT processing for the *author* elements underneath the *bookstore* element, then the XSLT stylesheet would look as follows:



While processing the *bookstore* element, you are telling the XSLT processor you are going to narrow the processing to the *book/author* elements and their children in the source XML document. The output on the facing page represents the result of the XSLT processing. The template for *bookstore* is instantiated and executed which in turn instantiates the templates for all *book/author* nodes below *bookstore*. All child elements of *author* are processed because the default template is used which specifies that all child elements of the current context will be processed.

Match used for a selection

```
<?xml version="1.0"?>
1 <bookstore>
  <book>
    <title> ... </title>
    2 <author>
      <last-name> ...</last-name>
      3 <award> ... </award>
    </author>
    <publisher> ... </publisher>
  </book>
  <book>
    <title> ... </title>
    2 <author>
      <last-name> ...</last-name>
      3 <award> ... </award>
    </author>
    <publisher> ... </publisher>
  </book>
</bookstore>
```

```
<xsl:stylesheet ...>
  <xsl:template match="bookstore"> 1
    <Awards>
      <xsl:apply-templates select="book/author"/> 2
    </Awards>
  </xsl:template>

  <xsl:template match="award"> 3
    <Award>
      <xsl:value-of select="."/>
    </Award>
  </xsl:template>
</xsl:stylesheet>
```

NOTE: XSLT processor narrows selection of elements to look at: *author* elements

Understanding difference between match and select

Match versus select

Thus far we have seen the XSLT processing directives for determining which template to instantiate and how to process the source XML document. It is important to understand the difference between the match and select attributes in an XSLT template:

- Match tells the XSLT processor which template to instantiate (execute) for the nodeset defined by the match.
 - Select is used during the execution of the template to narrow down the nodeset being processed.
-

Order of templates does not matter

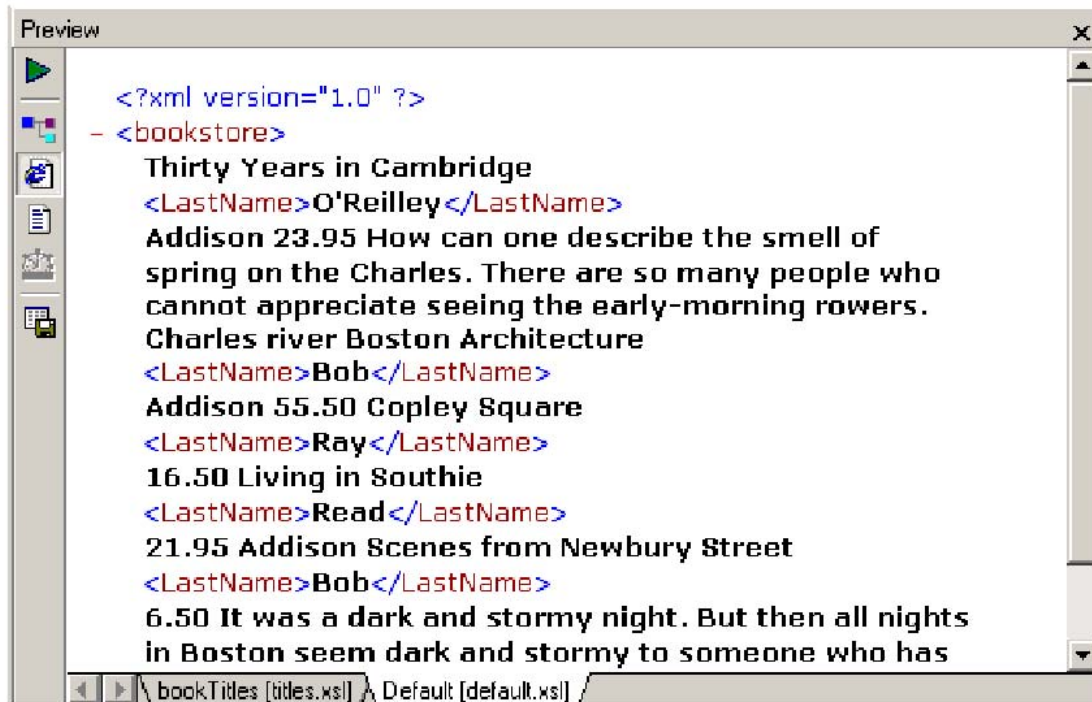
When the XSLT processor processes a source XML document, it performs a sequential parse of the XML looking for elements or attributes that match the templates in the XSLT stylesheet. The order of the elements in the source XML document combined with the processing instructions within the templates control execution of the XSLT processor. The order of the templates in the XSLT stylesheet is insignificant.

Example using match and select

On the facing page is an XSLT stylesheet that includes templates for *bookstore* and *award*. It uses a combination of select with apply-templates to narrow the processing to the *book/author* nodesets. The XSLT processing for this XSLT stylesheet is as follows:

1. Parse the source XML document and match a *bookstore* element.
 2. Within the *bookstore* element, look only at *book/author* nodes and, unless otherwise directed, output element values.
 3. When looking at the *bookstore/book/author* node, find an *award* element and output the value of the *award*.
-

XSLT processing with xsl:value-of



```
<?xml version="1.0" ?>
- <bookstore>
  Thirty Years in Cambridge
  <LastName>O'Reilley</LastName>
  Addison 23.95 How can one describe the smell of
  spring on the Charles. There are so many people who
  cannot appreciate seeing the early-morning rowers.
  Charles river Boston Architecture
  <LastName>Bob</LastName>
  Addison 55.50 Copley Square
  <LastName>Ray</LastName>
  16.50 Living in Southie
  <LastName>Read</LastName>
  21.95 Addison Scenes from Newbury Street
  <LastName>Bob</LastName>
  6.50 It was a dark and stormy night. But then all nights
  in Boston seem dark and stormy to someone who has
```

Using *xsl:value-of*

Introduction

You use the *xsl:value-of* element to move content from the source XML to the destination. You can place literals in the template which will be written to the destination and you can use *xsl:value-of* which takes a value determined at run-time to write to the destination.

Syntax for *xsl:value-of*

Here is the syntax for using *xsl:value-of*:

Syntax

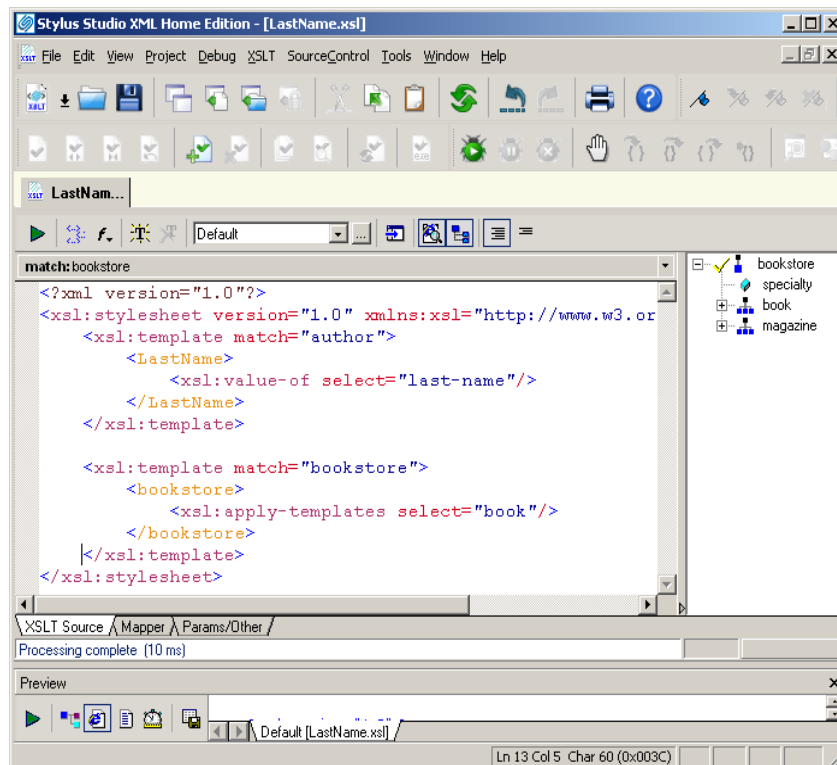
```
xsl: value-of select="expression"
```

where:

expression is an XPath expression that identifies a node or nodeset that should be selected relative to the current context; "." Means this text element.

Example of using *xsl:value-of*

In the this example of the *author* template, rather than selecting the value of the *author* element for output (select = "."), we select the *last-name* element value for output:



Lab 1: Transform the bookstore XML document



17

© 2008 Progress Software Corporation

Lab 1: Transform the bookstore XML document

Instructions

Follow the instructions in the Lab Guide for Lab 1.

Demonstration 3: Using Stylus Studio for development of templates



18

© 2008 Progress Software Corporation

Demonstration 3: Using Stylus Studio for development of templates

Introduction

Stylus Studio has some built-in features that make developing an XSLT stylesheet easier. Stylus Studio provides you help with:

- Creating a new template,
- Specifying a match pattern for the new template and
- Editing your XSL document using the full view vs. template view.

In this demonstration, you will practice creating new templates in Stylus Studio.

Creating a new XSLT stylesheet

You create a new XSLT stylesheet as follows:

1. In Stylus Studio, select File→New→XSLT: Text Editor.
2. A new scenario dialog box will appear and in it enter:

Field	Enter the following:
Scenario Name	FirstScenario
Source XML	Browse to: c:\progress_education\SOAEssentials\XSLTEssentials\Examples\bookstore.xml


3. Select OK.

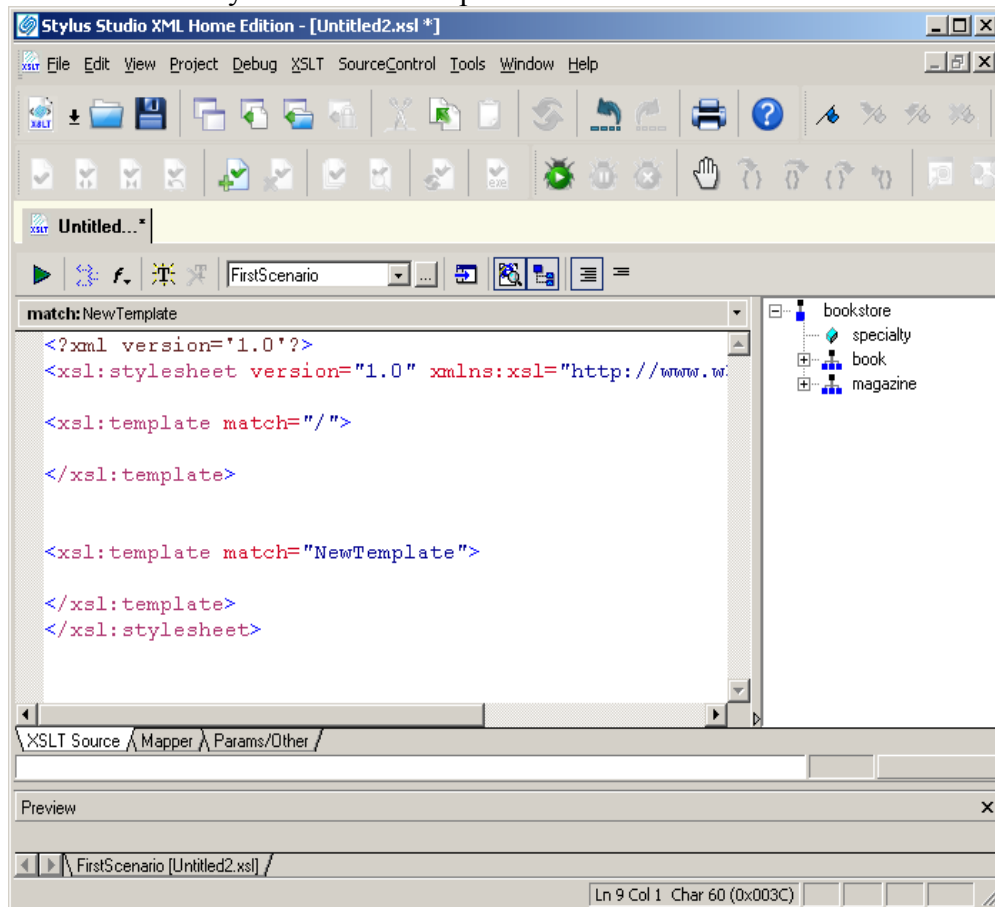
continued on next page

Demonstration 3: Using Stylus Studio for development of templates, continued

Creating a new template

Create a new template by following these steps:

1. When you first create an XSLT stylesheet, you automatically start in the XSLT editor at the template for the match to “/”. Notice that the source XML document’s schema is displayed in tree form in the right-hand pane.
2. If you select the  icon, the `xsl:template` element is added to the XSLT stylesheet with a match attribute value of “NewTemplate”. Below is what you should see when you select the template icon:



continued on next page

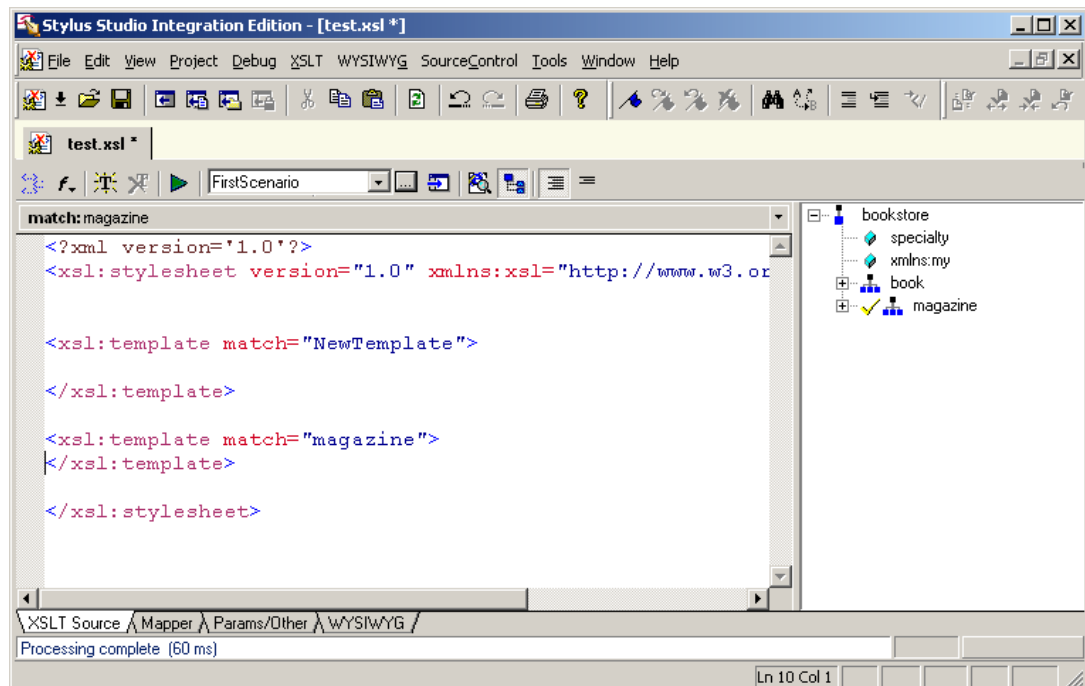
Demonstration 3: Using Stylus Studio for development of templates, continued

Modifying a template

The new template is not very interesting, so you will modify it as follows:

In the left-hand pane, replace “NewTemplate” with “magazine”.

Notice that a check mark is now placed in the right-hand pane where the tree view is displayed. This is because you have defined a template for the magazine element. You should see something like the following:



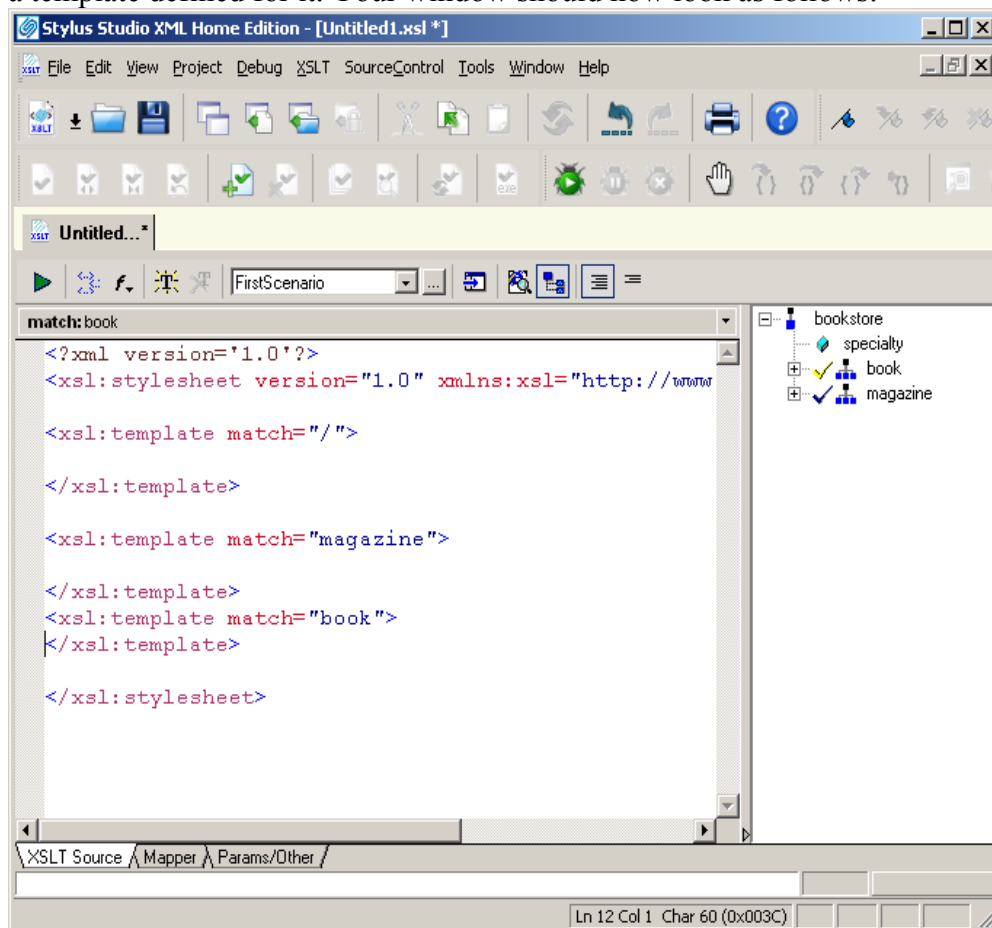
continued on next page

Demonstration 3: Using Stylus Studio for development of templates, continued

Creating a template from the XML schema tree view

If you want to create a template for a specific element, you can select the element from the XML source tree view and drag it into the XSL editing pane instead of using the template icon:



1. Select the *book* element from the right-hand pane where the tree view is displayed for the source XML Schema.
2. Drag it into the left-hand pane which contains the XSL document, making sure you drag it to a correct position within the source XSL document. Notice that this automatically creates the *xsl:template* element in the XSL editing pane and places a check mark to the left of the *book* element. This means that this element now has a template defined for it. Your window should now look as follows:

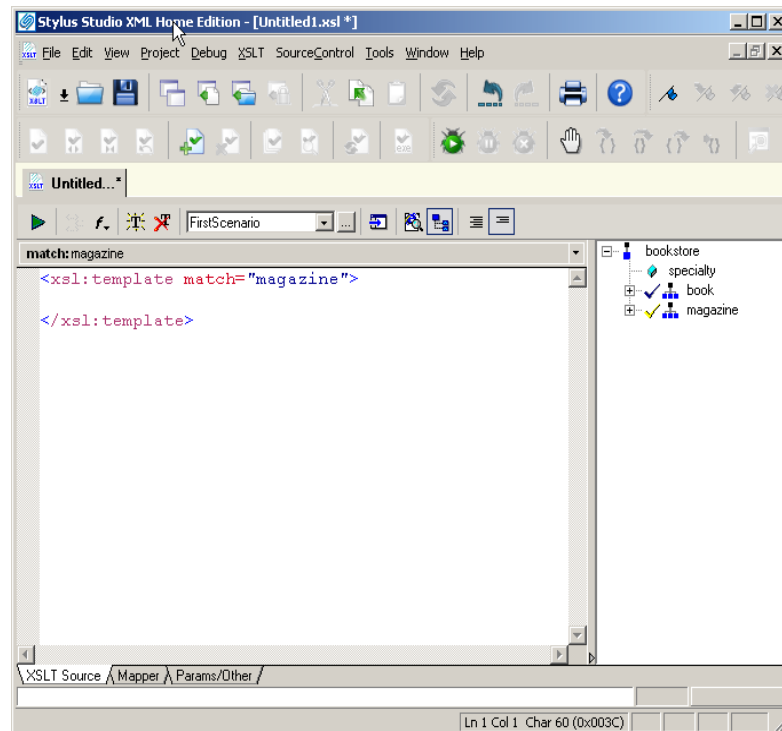


continued on next page

Demonstration 3: Using Stylus Studio for development of templates, continued




Full view vs. template view

You can toggle between full view  and template view  in the XSL editing pane. When you are in template view, the title on the XSL editing pane displays the template match you are viewing. You can select different templates to view by selecting the match in the source XML Schema tree that you want to view. Below is the template view for the *magazine* element:

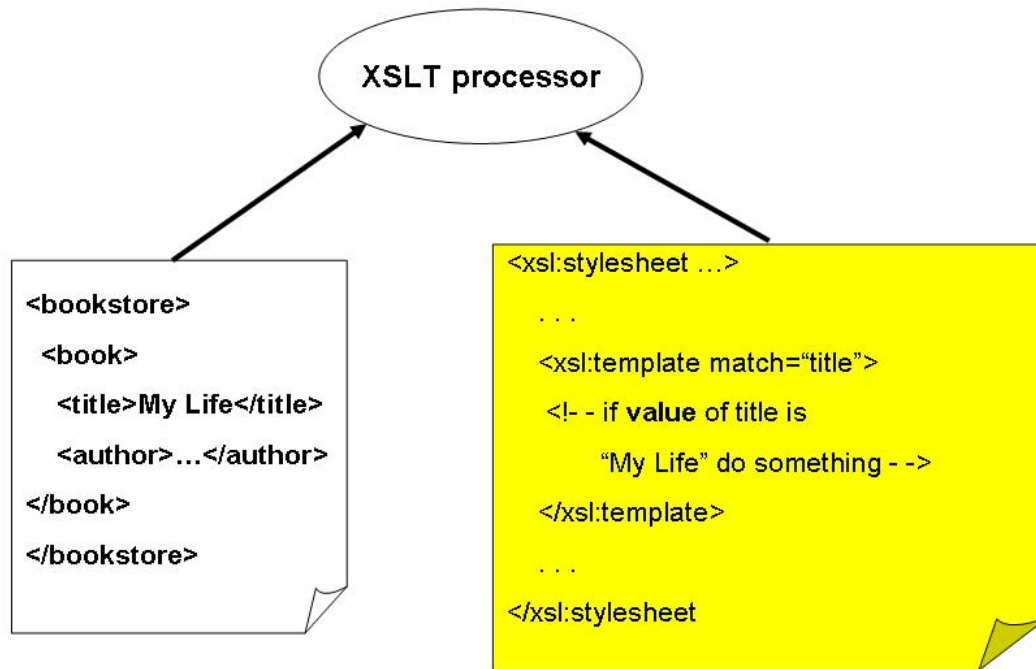


Practice toggling template views

In Stylus Studio, you can toggle between viewing all templates to viewing a particular template.

1. Select the  icon. Notice that this enables you to view or not view the source XML Schema tree in the right-hand pane.
 2. Select the  icon. Notice that you now have the view of a single template. You can use the title bar to select which template you will be viewing.
 3. Select the  icon. Notice that you now have a view of the full XSL document.
-

Controlling XSLT processing



Developing xsl template code

Introduction

Thus far within a template, you have learned how to write to the destination document by placing a literal in the template or by using the *xsl:value-of* element. You also have learned how to control which nodes to process next in the XSLT processing by using the *xsl:apply-templates* with a *select* attribute.

In some cases, you may want to have more programmatic control over what happens during the XSLT processing. It is common to have the XSLT processing depend upon element names or the values of the elements or attributes in the source XML document. You will now learn how to control the XSLT processing using the following XSL elements:

- `xsl:for-each`
 - `xsl:if`
 - `xsl:choose`
 - `xsl:copy`
 - `xsl:copy-of`
-

Using xsl:for-each

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <Books>
      <xsl:for-each select="bookstore/book">
        <Book>
          <xsl:for-each select="title">
            <Title>
              <xsl:value-of select="."/>
            </Title>
          </xsl:for-each>
        </Book>
      </xsl:for-each>
    </Books>
  </xsl:template>
</xsl:stylesheet>
```


Using *xsl:for-each*

Introduction

If the source XML document that you are transforming has an element (or set of elements) that all require the same processing, you can use the *xsl:for-each* element to tell the XSLT processor to repeat the same processing for all nodes selected. This also can be achieved by having a template for the selected nodes. As you gain more experience in XSLT development, you will learn that there are many ways to do the same processing.

Syntax for using *xsl:for-each*

Here is the syntax you can use to repeat some XSLT processing on a set of nodes:

Syntax

```
<xsl:for-each select="nodeSet">
  XSLTCode
</xsl:for-each>
```

where:

nodeSet is an XPath expression that identifies a node or set of nodes relative to the current node.

XSLTCode is whatever XSL elements or literal values you want to embed within *xsl:for-each* element.

Example using *xsl:for-each*

On the facing page is an XSLT stylesheet that contains a single template. Within this template, which is executed once, the XSLT processor looks for each *book* element and then within each *book* element looks for each *title* element. Within this repetitive processing, the appropriate element tags and values for the *title* element are generated.

Using xsl:if

```
<xsl:template match="title">
  <xsl:if test="parent::magazine">
    <Title type="magazine">
      <xsl:value-of select="."/>
    </Title>
  </xsl:if>
  <xsl:if test="parent::book">
    <Title type="book">
      <xsl:value-of select="."/>
    </Title>
  </xsl:if>
</xsl:template>
```

Using *xsl:if*

Introduction

Using the bookstore XML document as input, suppose we want to create a Titles document and each Title element will have an attribute named *type*. There are a couple of ways this XSLT processing could be done. One way would be to have templates that matches “book/title” and “magazine/title”. Within these templates, you generate the desired output. You did this in the previous lab.

Another solution would be to match “*title*” and within the template test to see if the current node is a child of a *book* or child of a *magazine*. If the current node is a child of a *book* element or a *magazine* element, then generate the output. The construct that can be used to do this test is the *xsl:if* processing directive.

Syntax for using *xsl:if*

Here is the syntax you can use to perform some conditional XSLT processing on a set of nodes:

Syntax

```
<xsl:if test="testValue">
    XSLTCode
</xsl:if>
```

where:

<i>testValue</i>	is an XPath expression that, when evaluated, will be either TRUE or FALSE.
<i>XSLTCode</i>	is whatever XSL elements or literal values you want to embed within the <i>xsl:if</i> element.

Example using *xsl:if*

On the facing page is an example of using *xsl:if*. During the processing of a *title* element, the XSLT processor generates the *Title* element. It has an attribute called *type* with a value of *book* or *magazine*. This setting of the attribute will depend on whether the *title* element has a *book* for a parent or a *magazine* for a parent.

Using xsl:choose

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates select="//publisher"/>
  </xsl:template>
  <xsl:template match="publisher">
    <xsl:choose>
      <xsl:when test=". = 'Addison'">
        <Publisher>Addison Wesley</Publisher>
      </xsl:when>
      <xsl:when test=". = 'Random House'">
        <Publisher>
          <xsl:value-of select="."/>
        </Publisher>
      </xsl:when>
      <xsl:otherwise>
        <Publisher>
          <xsl:value-of select="."/>
        </Publisher>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

22

© 2008 Progress Software Corporation

Using *xsl:choose*

Introduction

xsl:choose is useful if you need to perform some processing in the template based upon multiple values. In our previous example, we had a test for whether the parent node was a *book* element or a *magazine* element. We could use an *xsl:choose* for this rather than two *xsl:if* elements. It is more efficient to use *xsl:choose* than to have multiple *xsl:if* statements.

Syntax for *xsl:choose*

Here is the syntax for the for using the *xsl:choose* element:

```
Syntax
<xsl:choose>
  <xsl:when test="testValue1">
    XSLTCode
  </xsl:when>
  [<xsl:when test="testValueN">
    XSLTCode
  </xsl:when>]
  [<xsl:otherwise>
    XSLTCode
  </xsl:otherwise>]
</xsl:choose>
```

where:

- testValue1* is an XPath expression that, when evaluated, will be either TRUE or FALSE.
- testValueN* is a different XPath expression that, when evaluated, will be either TRUE or FALSE.
- XSLTCode* is whatever XSL elements or literal values you want to embed within the *xsl:when* element.

When the *xsl:when* element is encountered, each test is performed sequentially until a TRUE is returned. There can be any number of *xsl:when* elements. If the *xsl:otherwise* is defined, its XSLT processing directives are executed if all of the *xsl:when* tests evaluate to FALSE.

Example using *xsl:choose*

The template shown on the facing page utilizes the *xsl:choose* element to test each *publisher* element and create a *Publisher* element. If the tests enumerated do not cover all of the possible values for *publisher*, then the *xsl:otherwise* element is processed.

Getting branches of your document

```
<?xml version="1.0"?>
<bookstore>
  <book>
    <title> ... </title>
    <author>
      <last-name> ...</last-name>
      <award> ... </award>
    </author>
    <publisher> ... </publisher>
  </book>
  <book>
    <title> ... </title>
    <author>
      <last-name> ...</last-name>
      <award> ... </award>
    </author>
    <publisher> ... </publisher>
  </book>
</bookstore>
```

source.xml

```
<xsl:stylesheet ...>
...
<xsl:template match="book/author">
  <!-- generate complete author branch
       using xsl:copy-of -->
  </xsl:template>
</xsl:stylesheet>
```

bookstoreToAuthors.xsl

```
<?xml version="1.0"?>
<Authors>
  <author>
    <last-name> ...</last-name>
    <award> ... </award>
  </author>
  <author>
    <last-name> ...</last-name>
    <award> ... </award>
  </author>
</Authors>
```

destination.xml

© 2008 Progress Software Corporation

Using *xsl:copy-of*

Introduction

Many times when you want to perform an XML-to-XML transformation, there are sections of the source XML document that you want to duplicate “as is” without having to process them. An example of this for the bookstore XML document is the *excerpt* element. This element has children and grandchildren and we do not want to have to perform a lot of processing to retrieve the *excerpt* element from the source XML for writing to the destination. In order to perform this type of processing, you use the *xsl:copy-of* element.

Syntax for *xsl:copy-of*

Here is the syntax for copying entire sections of an XML document:

Syntax

```
<xsl:copy-of select="nodeSet" />
```

where:

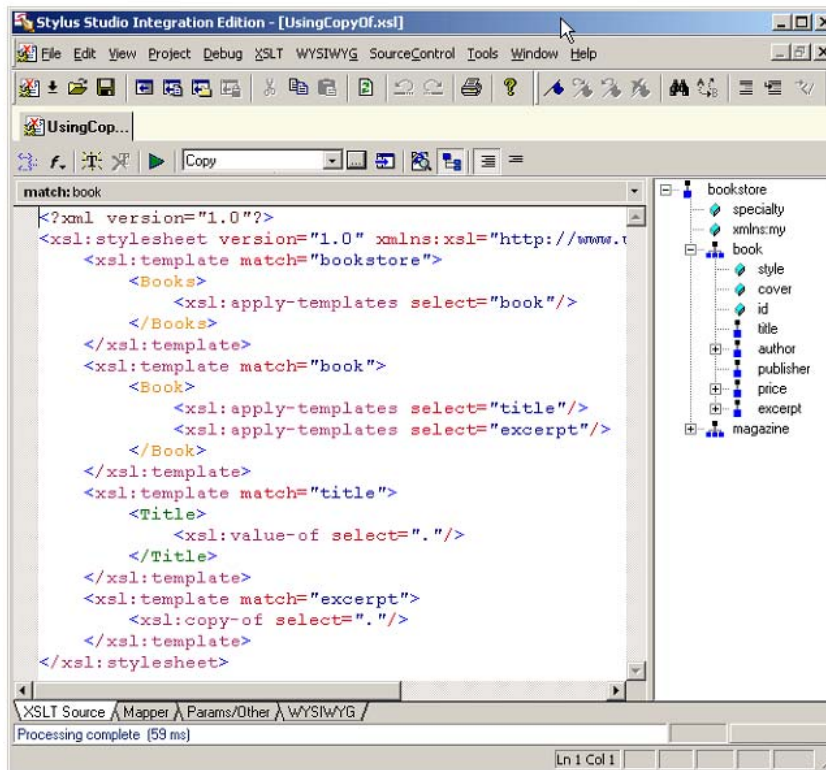
nodeSet

is the XPath expression which selects the node or nodeset specified relative to the current node.

This element directs the XSLT processor to copy all nodes, attributes and descendants of the node or nodeset to the destination.

continued on next page

Using xsl:copy-of



24

Software Corporation

Using *xsl:copy-of*, continued

Example: Using *xsl:copy-of*

The XSLT stylesheet on the facing page writes the *book* and *title* elements and copies the *excerpt* elements from the source XML document. The resulting output is as follows:

```
<Books>
  <Book>
    <Title>Thirty Years in Cambridge</Title>
    <excerpt language="English">
      <p>How can one describe the smell of spring on the Charles.</p>
      <p>There are so many people who cannot appreciate seeing the early-morning
        rowers.</p>
      <index-list>
        <term>Charles</term>
        <definition>river</definition>
      </index-list>
    </excerpt>
  </Book>
  <Book>
    <Title>Boston Architecture</Title>
  </Book>
  <Book>
    <Title>Copley Square</Title>
  </Book>
  <Book>
    <Title>Living in Southie</Title>
  </Book>
  <Book>
    <Title>Scenes from Newbury Street</Title>
    <excerpt validated="no" language="English">
      <p>It was a dark and stormy night.</p>
      <p>But then all nights in Boston seem dark and stormy to someone who has gone
        through what<emph>I</emph>have.</p>
      <index-list>
        <term>Boston</term>
        <definition>city</definition>
      </index-list>
    </excerpt>
  </Book>
</Books>
```

Lab 2: Creating XSLT templates



25

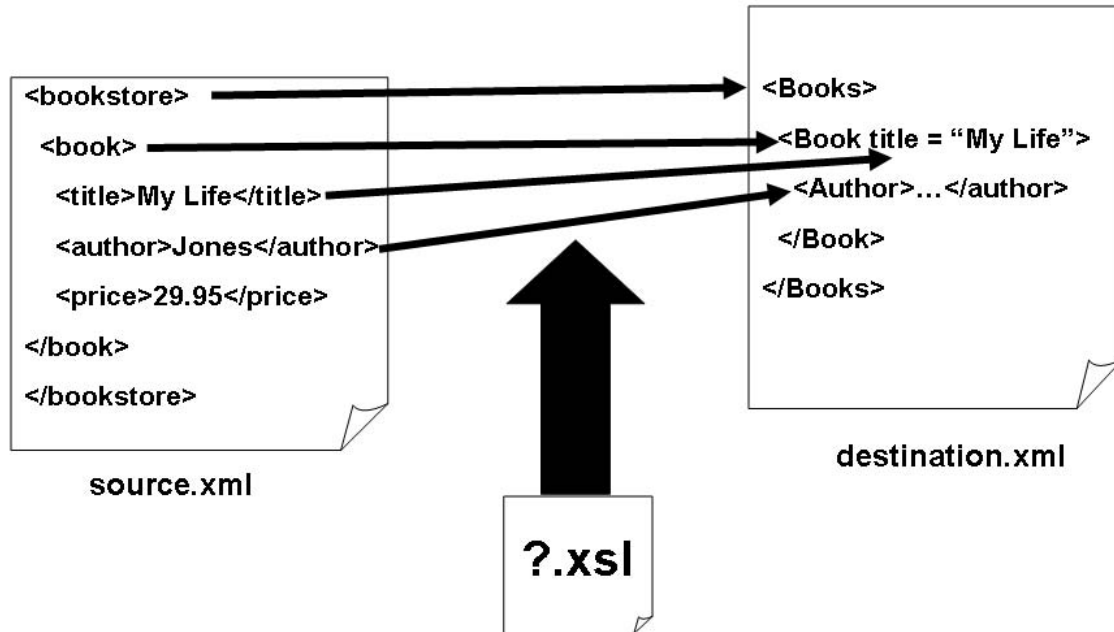
© 2008 Progress Software Corporation

Lab 2: Creating XSLT templates

Instructions

Follow the instructions in the Lab Guide for Lab 2.

Mapping XML to XML



Mapping XML to XML with Stylus Studio

Introduction

Thus far, you have learned some of the basic XSLT processing directives that can be executed when a template is instantiated. You should know enough about XSLT to be able to understand a basic XSLT stylesheet. Stylus Studio has a very useful feature that enables you to automatically create an XSLT stylesheet if you know what your source XML document looks like and what your destination XML document should look like.

In an SOA environment in which information is exchanged using XML documents from various services or partners, it is very useful to be able to transform a document from one format to another. You will now learn how to automatically create an XML to XML mapping stylesheet using Stylus Studio.

When to automatically generate an XSLT stylesheet?

In the real world, not all XML documents can be well-defined. Sometimes elements or attributes are optional or repeated making generating the XSLT stylesheet difficult. You can first automatically generate an XSLT stylesheet and then use your knowledge of XSLT to customize the XSLT stylesheet for your application needs.

Before you begin

In order to use the XML to XML mapping functionality of Stylus Studio, you must have a representative XML document for the source and destination. These documents should conform to some DTD or XML Schema that is agreed upon by the senders and receivers of the documents. The example shown on the facing page shows two XML documents. We want to create the XSLT stylesheet that will transform the source XML document to the destination XML document.

continued on next page

Mapping XML to XML with Stylus Studio, continued

Creating an XML to XML mapping stylesheet

In Stylus Studio, you can create and test an XML to XML mapping stylesheet as follows:

Step	Description
1.	Create a new XML to XML mapping stylesheet.
2.	Specify the source and destination XML documents.
3.	Examine the source and destination schemas to determine mapping.
4.	Map elements and attributes from the source to the destination.
5.	Save the XSLT stylesheet.
6.	Test the mapping by applying the XSLT stylesheet to the source XML document.
7.	Examine the generated XSLT stylesheet.

Next you will practice creating an XML to XML mapping stylesheet using Stylus Studio.

Notes

Demonstration 4: Creating an XML to XML mapping stylesheet



27

© 2008 Progress Software Corporation

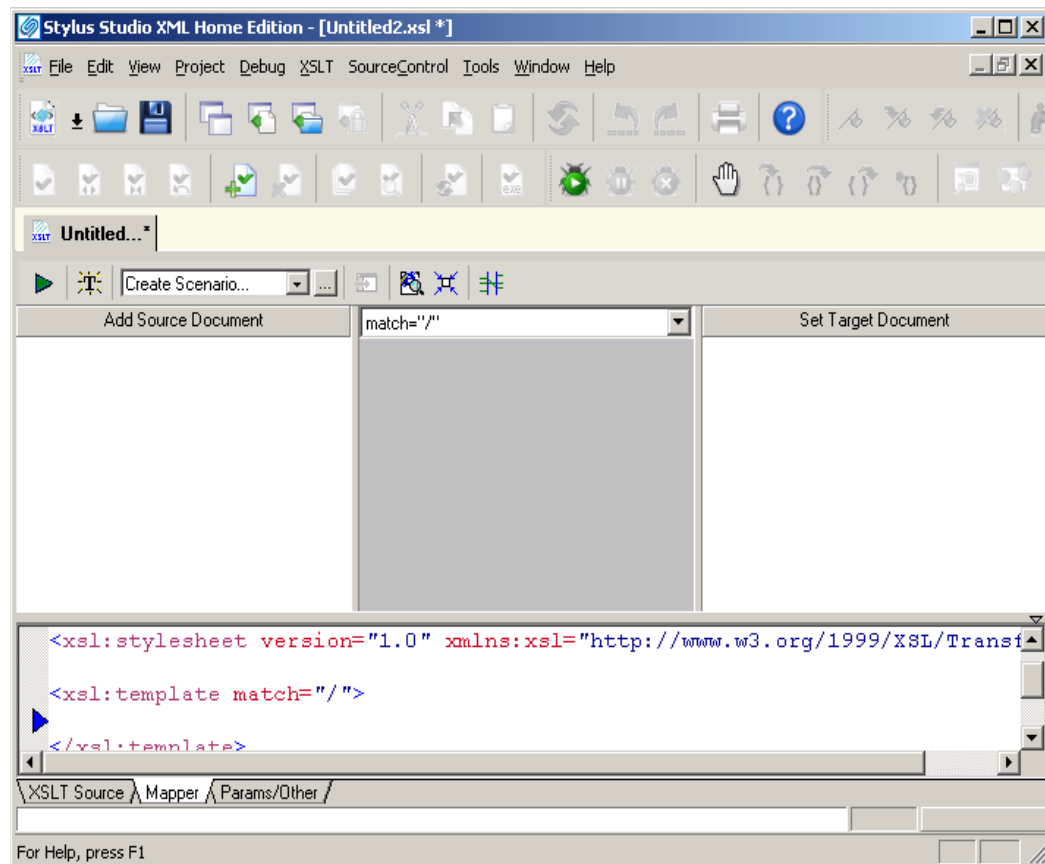
Demonstration 4: Creating an XML to XML mapping stylesheet

Introduction

In this demonstration, you will use Stylus Studio to create a very simple XSLT stylesheet that maps the bookstore XML document to the Books XML document.

Create a new XML to XML mapping stylesheet

In Stylus Studio, select File→New→XSLT: Mapper. You should see the following:



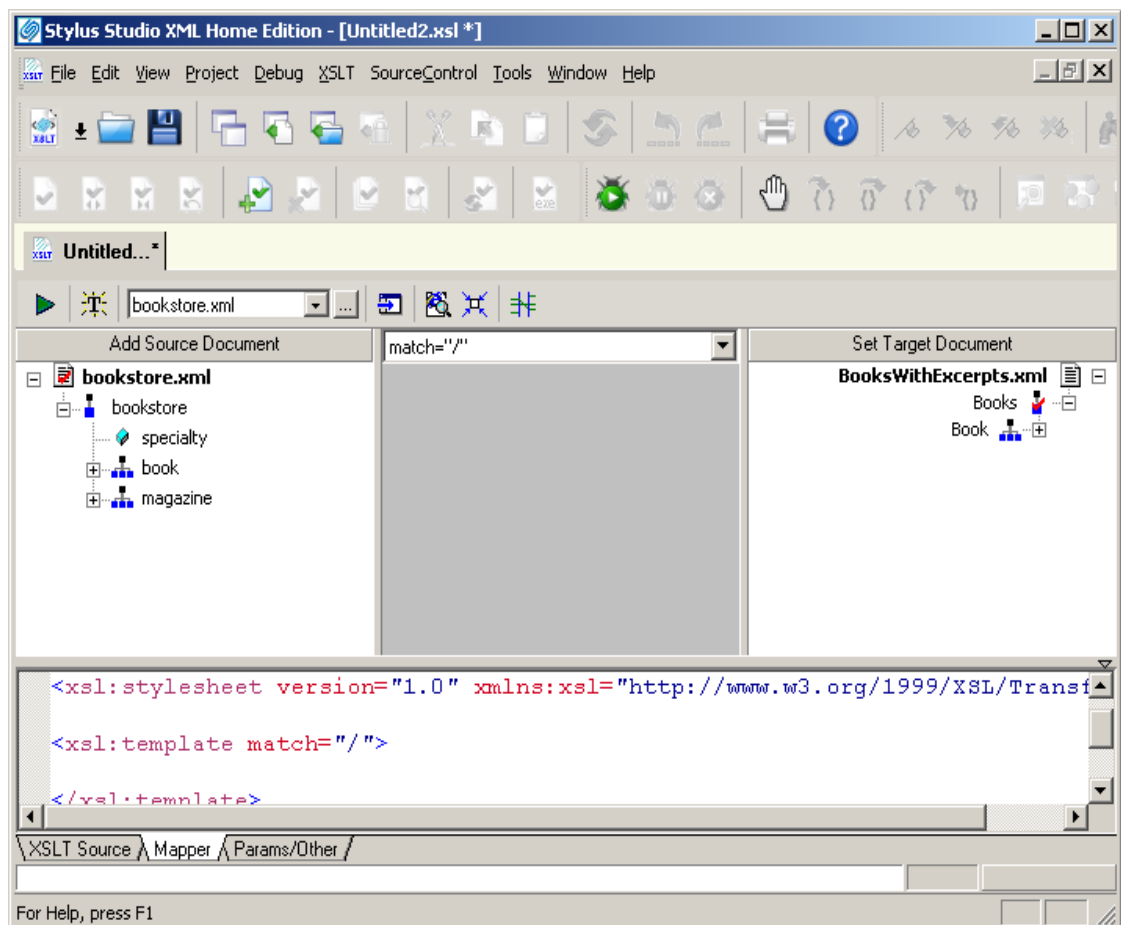
continued on next page

Demonstration 4: Creating an XML to XML mapping stylesheet, continued

Select the source and destination XML documents for the mapping

In order to perform the mapping, you need to specify what your source and destination XML documents look like.

1. Select the area of the window “Add Source Document” and select:
c:\progress_education\SOAEssentials\XSLTEssentials\Examples\bookstore.xml
2. Select the area of the window “Set Target Document” and select:
c:\progress_education\SOAEssentials\XSLTEssentials\Examples\BooksWithExcerpts.xml
3. Select OK. Your window should now look as follows:

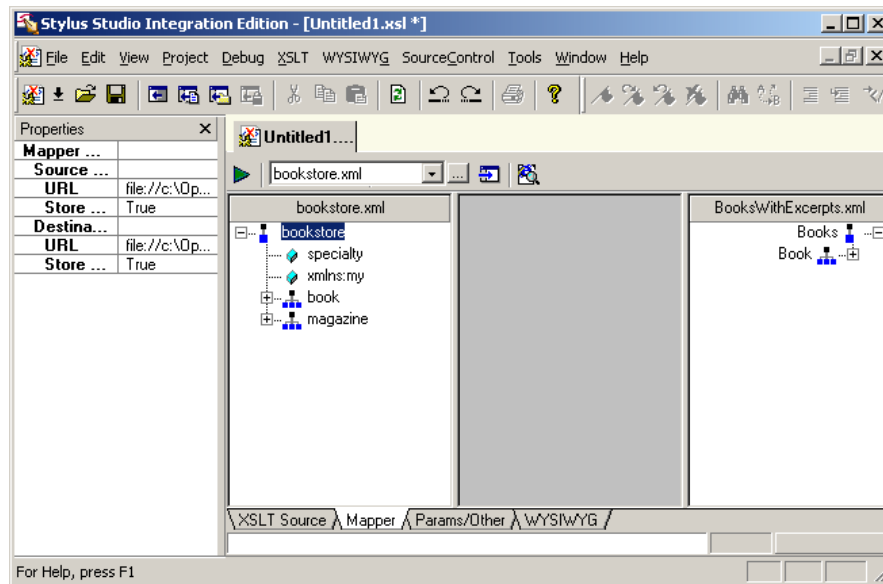


continued on next page

Demonstration 4: Creating an XML to XML mapping stylesheet, continued

Examine the source and destination schemas

Once you have selected the source and destination XML documents, you will see the Stylus Studio mapping environment. The left-hand pane contains the source schema in tree view and the right-hand pane contains the destination schema in tree view. The center pane is reserved for displaying the mapping you will select. You should see the following in Stylus Studio:



You can open any of the elements in the source or destination schema views in order to see child elements. You will map all elements or attributes in the source XML document to elements or attributes in the destination document. Notice that the names of the elements and attributes are slightly different. Also notice that elements may map to attributes and visa versa.

continued on next page

Demonstration 4: Creating an XML to XML mapping stylesheet, continued

Perform the mapping

In order to map elements and/or attributes from the source XML schema to the destination XML schema, you simply select an element or attribute from one side and drag it over to the other side. When you connect two elements or attributes, a line is shown connecting them. You can connect an element to an element, an element to an attribute or an attribute to an element. If an element has a text node, then its value is automatically mapped to the destination XML document. Here is the mapping you will perform:

From	To
book	Book
title	Title

Follow these steps in order to perform the mapping:

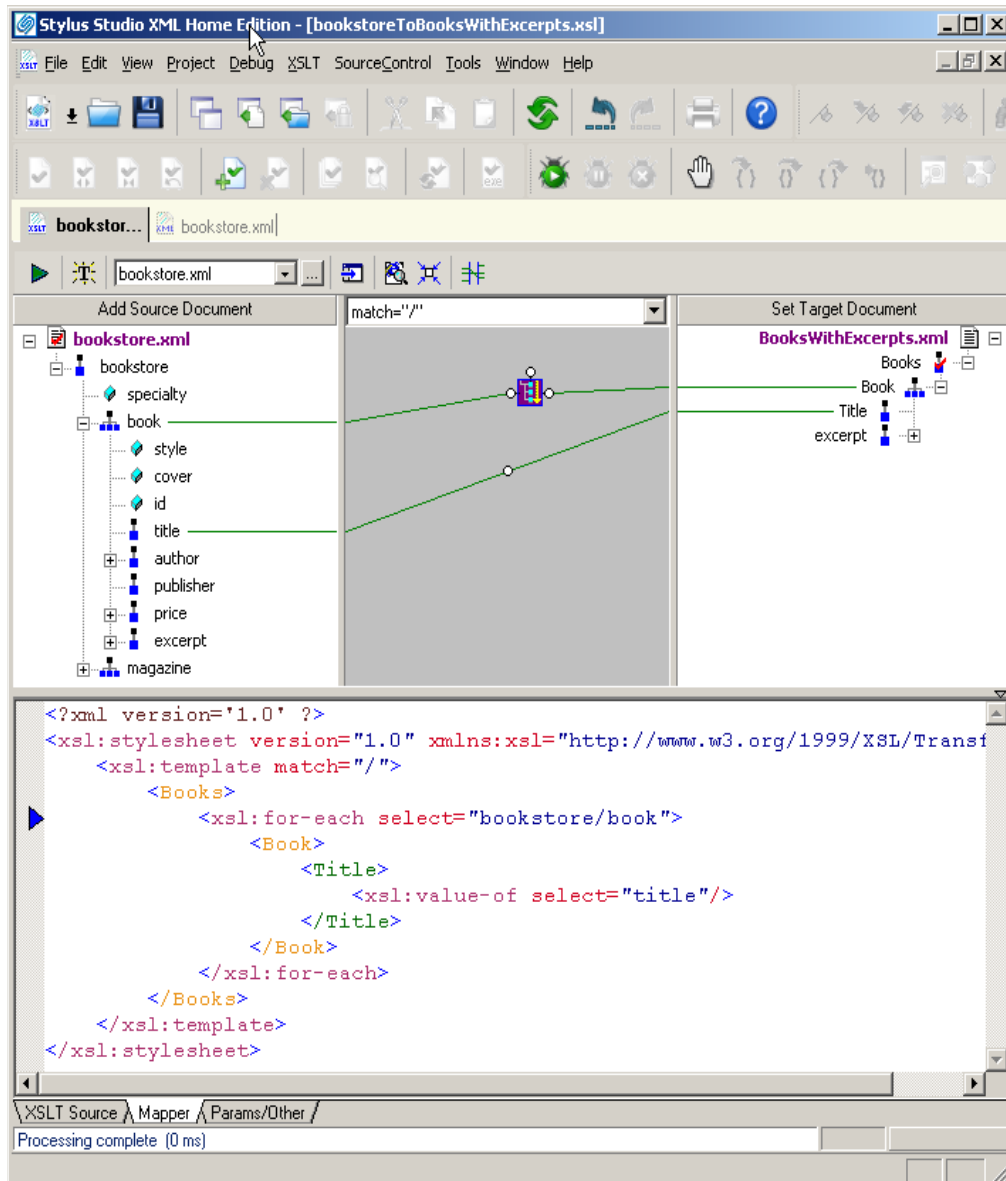
1. In the left-hand pane entitled “bookstore.xml”, select the *book* element and drag it over to the *Book* element in the right-hand pane entitled “BooksWithExcerpts.xml”.
2. Repeat the process again with the *title* element, mapping it to the *Title* element in the right-hand pane.

continued on next page

Demonstration 4: Creating an XML to XML mapping stylesheet, continued

Perform the mapping, continued

You should now have two connections between the source XML document and the destination XML document. Your mapping should look as follows:



The screenshot shows the Stylus Studio XML Home Edition interface. The main window displays the XSLT mapping between the source XML document (bookstore.xml) and the target XML document (BooksWithExcerpts.xml). The mapping is defined by two connections:

- A connection from the `bookstore/book` element to the `Books/Book` element.
- A connection from the `bookstore/book/title` element to the `Books/Book/Title` element.

The XSLT code in the main window is as follows:

```
<?xml version='1.0' ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transf
<xsl:template match="/">
  <Books>
    <xsl:for-each select="bookstore/book">
      <Book>
        <Title>
          <xsl:value-of select="title"/>
        </Title>
      </Book>
    </xsl:for-each>
  </Books>
</xsl:template>
</xsl:stylesheet>
```

continued on next page


Demonstration 4: Creating an XML to XML mapping stylesheet, continued

Save the XSLT stylesheet

Save your XSLT stylesheet as **bookstoreToBooksWithExcerpts.xsl**.

Test the mapping

Once you have connected all elements and/or attributes that you want to map from the source XML document to the destination XML document, you are ready to test the mapping.

Test the mapping by selecting the  icon.

The resulting mapping occurs because the XSLT stylesheet is mapped to the source XML document that you specified when you created the mapping XSLT stylesheet.

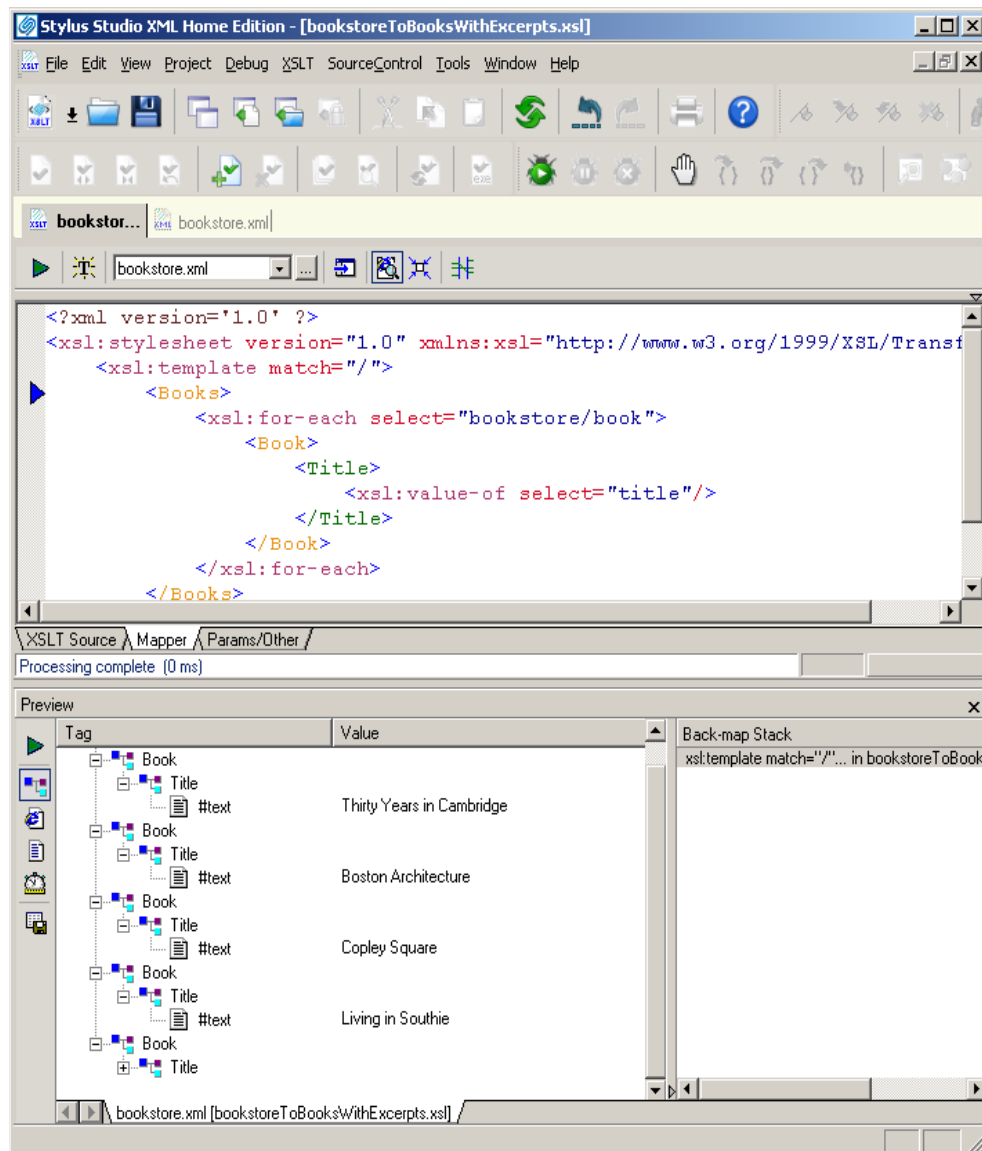
A Preview pane also should be created.

continued on next page

Demonstration 4: Creating an XML to XML mapping stylesheet, continued

Test the mapping, continued

Review the results to see if you have generated a root element of *Books* that has sub-elements of *Book* which have a sub-element of *Title*. The *Title* should have a text node with a value. It should look something like the following:



The screenshot shows the Stylus Studio XML Home Edition interface. The main window displays an XSLT stylesheet with the following code:

```
<?xml version='1.0' ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transf
  <xsl:template match="/">
    <Books>
      <xsl:for-each select="bookstore/book">
        <Book>
          <Title>
            <xsl:value-of select="title"/>
          </Title>
        </Book>
      </xsl:for-each>
    </Books>
  </xsl:template>
</xsl:stylesheet>
```

The Preview window shows the resulting XML structure:

Tag	Value
Book	
Title	
#text	Thirty Years in Cambridge
Book	
Title	
#text	Boston Architecture
Book	
Title	
#text	Copley Square
Book	
Title	
#text	Living in Southie
Book	
Title	

You should make sure that the new target document looks like what you had as your original target when you created the mapping.

Lab 3: Create an XML to XML mapping stylesheet

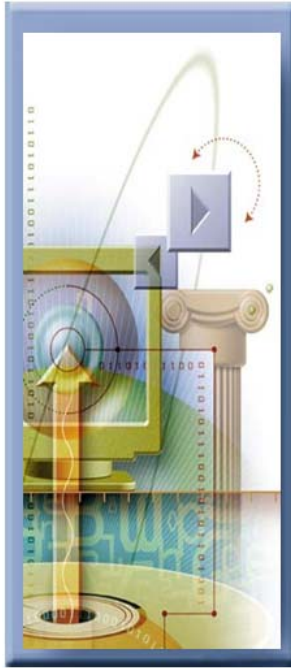


Lab 3: Create an XML to XML mapping stylesheet

Instructions

Follow the instructions in the Lab Guide for Lab 3.

Summary



You should be able to:

- Describe why transformation is necessary
- Describe what XSL is
- Create XSLT stylesheets with Stylus Studio
- Describe what XSLT processing is
- Define some useful XSL elements in your XSLT stylesheets
- Test XSLT stylesheets that you have developed
- Create and test an XML to XML mapping stylesheet

Summary

Lesson summary

You should now be able to:

- Describe why transformation is necessary,
 - Describe what XSL is,
 - Create and test a simple XSLT stylesheet using Stylus Studio,
 - Describe what XSLT processing is,
 - Define some useful XSL elements in your XSLT stylesheet,
 - Test an XSLT stylesheet that you have developed and
 - Create and test XML to XML mapping with Stylus Studio.
-

Review



Review questions

Assess what you have learned in this lesson

Answer the following questions:

1. In what language is an XSLT stylesheet written?
 2. What are the inputs and outputs to an XSLT stylesheet?
 3. What is the main *xsl* element used to define the XSLT processing?
 4. Does the XML document created by an XSLT stylesheet need to be a well-formed XML document?
 5. Which *xsl* element is used to retrieve information from the source XML document using a selected set of nodes?
 6. What must you create in Stylus Studio in order to test XSLT processing for an XSLT stylesheet that you have created?
 7. Which *xsl* element is used to retrieve text nodes or attribute values from the source XML document?
 8. Which *xsl* element is used to drive continued template processing?
-

Review answers

Assess what you have learned in this lesson

Here are the answers to the review questions:

1. In what language is an XSLT stylesheet written?
Answer: XML, which conforms to the XML Schema for XSLT defined by the W3C. This dialect of XML is called XSL.
2. What are the inputs and outputs to an XSLT stylesheet?
Answer: The input is an XML document. The output can be an XML document, text, html, and even PDFs when you use xslfo (formatting objects).
3. What is the main *xsl* element used to define the XSLT processing?
Answer: *xsl:template*.
4. Does the XML document created by an XSLT stylesheet need to be a well-formed XML document?
Answer: It doesn't need to be well-formed, but if the recipient of the XML document is expecting it to be well-formed, then it should be well-formed.
5. Which *xsl* element is used to retrieve information from the source XML document using a selected set of nodes?
Answer: *xsl:for-each*.

continued on next page

Review answers, continued

Assess what you have learned in this lesson, continued

6. What must you create in Stylus Studio in order to test XSLT processing for an XSLT stylesheet that you have created?

Answer: A scenario.

7. Which *xsl* element is used to retrieve text nodes or attribute values from the source XML document?

Answer: *xsl:value-of*.

8. Which *xsl* element is used to drive continued template processing?

Answer: *xsl:apply-templates*.

Notes