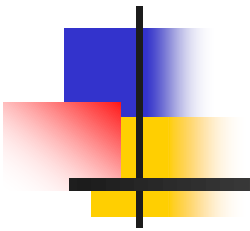


Module 7

More on XSLT

Module Overview

In this module, you will learn about:

- XPath
- XPath Expressions and Functions
- Working with different Styles

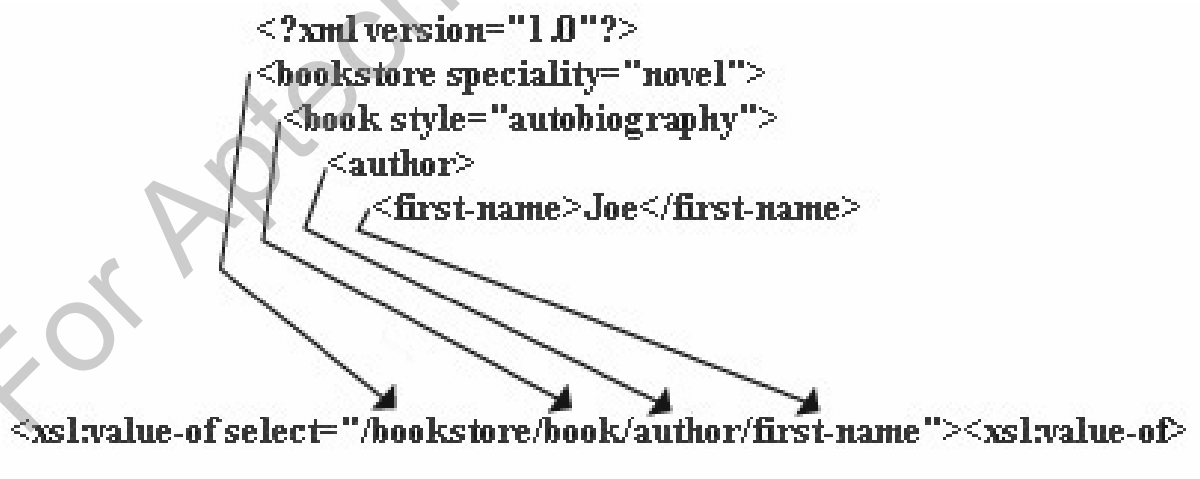
Lesson 1 - XPath

In this first lesson, **XPath**, you will learn to:

- Define and describe XPath.
- Identify nodes according to XPath.
- List operators used with XPath.
- Describe the types of matching

XPath 1-2

- XPath can be considered as a query language like SQL.
- Extracts information from an XML document.
- Language for retrieving information from a XML document.
- Used to navigate through elements and attributes in an XML document.
- Allows identifying parts of an XML document.
- Provides a common syntax for features shared by Extensible Stylesheet Language Transformations (XSLT) and XQuery.



XPath 2-2

- XSLT
  - Language for transforming XML documents into XML, HTML or text.
- XQuery
  - Builds on XPath.
  - Language for extracting information from XML documents.

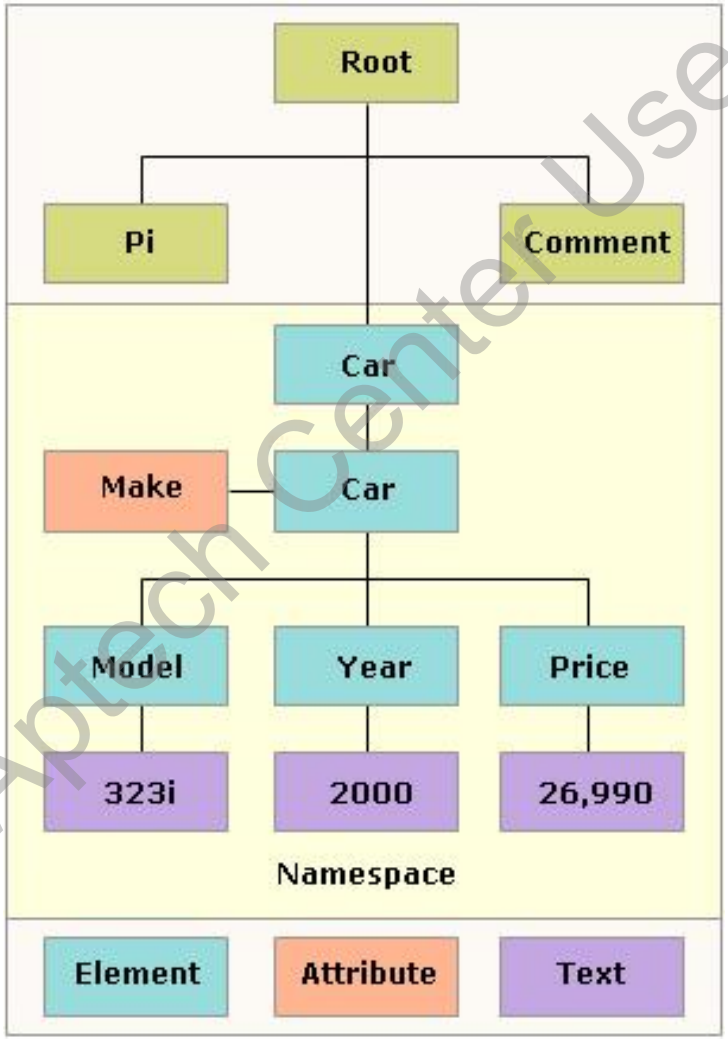
Benefits of XPath

- Provides single syntax that you can use for:
  - Queries
  - Addressing
  - Patterns
- Concise, simple, and powerful.
- Benefits:
  - Syntax is simple for the simple and common cases.
  - Any path that can occur in an XML document.
  - Any set of conditions for the nodes in the path can be specified.
  - Any node in an XML document can be uniquely identified.

XML Document in XPath 1-4

- An XML document is viewed as a tree.
- Each part of the document is represented as a node.
- Nodes
  - Root
    - A single `root` node that contains all other nodes in the tree.
  - Element
    - Every element in a document has a corresponding `element node` that appears in the tree under the `root` node.
    - Within an `element` node appear all of the other types of nodes that correspond to the element's content.
    - Element nodes may have a unique identifier associated with them that is used to reference the node with XPath.

XML Document in XPath 2-4



XML Document in XPath 3-4

- Attribute
  - Each `element` node has an associated set of attribute nodes.
  - Element is the parent of each of these attribute nodes.
  - An attribute node is not a child of its parent element.
- Text
  - Character data is grouped into text nodes.
  - Characters inside comments, processing instructions and attribute values do not produce text nodes.
  - The `text` node has a parent node and it may be the `child` node.
- Comment
  - There is a `comment` node for every comment, except within the document type declaration (DTD).
  - The `comment` node has a parent node and it may be the child node too.

XML Document in XPath 4-4

- Processing instruction
  - `Processing instruction` node exists for every processing instruction except for any `processing instruction` within the document type declaration.
  - The processing instruction node has a `parent` node and it may be the `child` node too.
- Namespace
  - Each element has an associated set of `namespace` nodes.
  - Provides descriptive information about their `parent` node.

XPath Representation

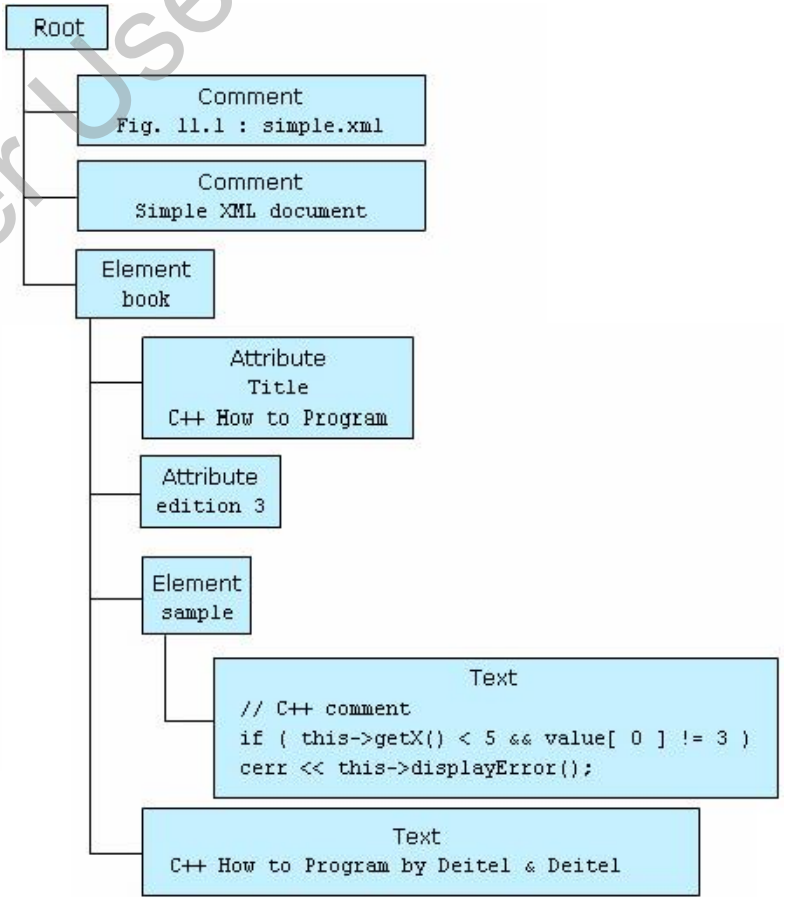
- An XPath query operates on a well-formed XML document after it has been parsed into a tree structure.

```
<?xml version = "1.0"?>

<!-- Fig. 11.1 : simple.xml -->
<!-- Simple XML document -->

<book title = "C++ How to Program" edition = "3">
<sample>
<![CDATA[
// C++ comment
 if ( this->getX() < 5 && value[ 0 ] != 3 )
cerr << this->displayError();
 ]]>
 </sample>

C++ How to Program by Deitel &amp;amp; Deitel
</book>
```



Original XML Document

XPath Representation

Operators in XPath

- An XPath expression returns a node set, a boolean, a string, or a number.
- XPath provides basic floating point arithmetic operators and some comparison and boolean operators.

Operator	Description
/	Child operator; selects immediate children of the left-side collection.
//	Recursive descent; searches for the specified element at any depth
.	Indicates the current context
..	The parent of the current context node
*	Wildcard; selects all elements regardless of the element name
@	Attribute; prefix for an attribute name
:	Namespace separator; separates the namespace prefix from the element or attribute name

Examples of XPath Operators

Expression	Refers to
`Author/FirstName`	All `<FirstName>` elements within an `<Author>` element of the current context node.
`BookStore//Title`	All `<Title>` elements one or more levels deep in the `<BookStore>` element.
`BookStore/*/Title`	All `<Title>` elements that are grandchildren of <BookStore> elements.
`BookStore//Book/Excerpt//Work`	All `<Work>` elements anywhere inside `<Excerpt>` children of `<Book>` elements, anywhere inside the `<BookStore>` element.
`.//Title`	All `<Title>` elements one or more levels deep in the current context.

Types of Matching 1-3

- XPath can create of the patterns.
- The match attribute of the `xsl:template` element supports a complex syntax.
- Allows to express exactly which nodes to be matched.
- The select attribute of `xsl:apply-templates`, `xsl:value-of`, `xsl:for-each`, `xsl:copy-of`, and `xsl:sort` supports a superset of the syntax.
- Allows to express exactly which nodes to selected and which nodes not to be selected.

Types of Matching 2-3

- Matching by name
  - The source element is identified by its name, using the `match` attribute.
  - The value given to the `match` attribute is called the pattern.

Code Snippet

```
<xsl:template match = "Greeting">
```

Matches all greeting elements in the source document.

- Matching by ancestry
  - As in CSS, a match can be made using the element's ancestry.

Code Snippet

```
<xsl:template match = "P//EM">
```

Matches any 'EM' element that has 'P' as an ancestor.

- Matching the element names
  - The most basic pattern contains a single element name which matches all elements with that name.

Code Snippet

```
<xsl:template match="Product">
 <xsl:value-of select="Product_ID"/>
</xsl:template>
```

Matches `Product` elements and marks their `Product_ID` children bold.

Types of Matching 3-3

- Matching the root
  - Enables all descendant nodes to inherit the properties on the root of document.
  - Uses a single forward slash to represent the root.

Code Snippet

```
<xsl:template match = "/">
```

Selects the root pattern.

- Matching by attribute
  - As in CSS, a match can be made using the element's ancestry.

Syntax

```
<xsl:template match = " element name['attribute'
(attribute-name)=attribute-value]">
```

Uses square brackets to hold the attribute name and value.

Code Snippet

```
<xsl:template match="Product">
  <xsl:apply-templates select="@Units"/>
</xsl:template>
```

Applies the templates to the non-existent Units attributes of Product elements.

Lesson 2 – XPath Expressions and Functions

In this second lesson, **XPath Expressions and Functions**, you will learn to:

- State and explain the various XPath expressions and functions.
- List the node set functions.
- List the boolean functions.
- State the numeric functions.
- Describe the string functions.
- Explain what result tree fragments are.

XPath Expressions 1-2

- Statements that can extract useful information from the XPath tree.
- Instead of just finding nodes, one can count them, add up numeric values, compare strings, and more.
- Are like statements in a functional programming language.
- Evaluates to a single value.

XPath Expressions 2-2

- Node-set
  - An unordered group of nodes from the input document that match an expression's criteria.
- Boolean
  - XSLT allows any kind of data to be transformed into a `boolean`.
  - Often done implicitly when a `string` or a `number` or a node-set is used where a `boolean` is expected.
- Number
  - Numeric values useful for counting nodes and performing simple arithmetic.
  - Numbers like 43 or –7000 that look like integers are stored as doubles.
  - Non-number values, such as strings and `booleans`, are converted to numbers automatically as necessary.
- String
  - A sequence of zero or more Unicode characters.
  - Other data types can be converted to strings using the `string()` function.

XPath Functions

- XPath defines various functions required for XPath 2.0, XQuery 1.0 and XSLT 2.0.
- The different functions are `Accessor`, `AnyURI`, `Node`, `Error` and `Trace`, `Sequence`, `Context`, `Boolean`, `Duration/Date/Time`, `String`, `QName` and `Numeric`.
- Can be used to refine XPath queries and enhance the programming power and flexibility of XPath.
- Each function in the function library is specified using a function prototype that provides the return type, function name, and argument type.
- If an argument type is followed by a question mark, the argument is optional; otherwise, the argument is required.
- Function names are case-sensitive.
- The default prefix for the function namespace is `fn`.

Node-Set Functions 1-4

- Take a `node-set` argument.
- Return a `node-set` or information about a particular node within a node-set.
- Are `name()`, `local-name()`, `namespace-uri()` and `root()`.

Syntax

```
name()
fn:name()
fn:name(nodeset)

local-name()
fn:local-name()
fn:local-name(nodeset)

namespace-uri()
fn:namespace-uri()
fn:namespace-uri(nodeset)

root()
fn:root()
fn:root(node)
```

where,

`name()`: Returns name of current node or the first node in specified node set.

`local-name()`: Returns name of current node or the first node in specified node set without namespace prefix.

`namespace-uri()`: Returns namespace URI of current node or first node in specified node set.

`root()`: Returns root of tree to which the current node or specified node belongs. This will usually be a document node.

Node-Set Functions 2-4

Code and Schema

<!--Book.xml -->

```
<?xml-stylesheet type="text/xsl" href="Sample.xsl"?>
<Catalog xmlns="http://www.BookCatalog.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.BookCatalog.com BookSchema.xsd">
        <Book>
                <Author>Gambardella, Matthew</Author>
                <Title>XML Developer's Guide</Title>
                <Genre>Computer</Genre>
                <Price>44.95</Price>
                <Publish>2000-10-01</Publish>
                <Description>An in-depth look at creating applications
           with XML.</Description>
        </Book>
</Catalog>
```

<!—BookSchema.xsd -->

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.BookCatalog.com"
  xmlns="http://www.BookCatalog.com"
  elementFormDefault="qualified">

  <!--definition for simple elements-->

  <xs:element name="Author" type="xs:string"/>
  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Genre" type="xs:string"/>
  <xs:element name="Price" type="xs:float"/>
  <xs:element name="Publish" type="xs:string"/>
  <xs:element name="Description" type="xs:string"/>

  <!--definition for complex elements-->
  <xs:element name="Book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Author"/>
        <xs:element ref="Title"/>
        <xs:element ref="Genre"/>
        <xs:element ref="Price"/>
        <xs:element ref="Publish"/>
        <xs:element ref="Description"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="Catalog">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Node-Set Functions 3-4

Stylesheet

```
<!—Sample.xsl -->
1  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
2     <xsl:output method="html"/>
3    <xsl:template match="/">
4        <html>
5           <body>
6              <h3>Node-set Function</h3>
7
8              <table width="100%" border="1">
9                 <tr>
10                   <td width="25%"><b>namespace-uri()</b></td>
11                   <td width="25%"><b>name()</b></td>
12                   <td width="25%"><b>local-name</b></td>
13                   <td width="25%"><b>text()</b></td>
14                </tr>
15                <xsl:apply-templates />
16             </table>
17          </body>
18       </html>
19   </xsl:template>
20
21   <xsl:template match="*">
22       <tr>
23          <td>
24             <xsl:value-of select="namespace-uri()"/>
25          </td>
26          <td>
27             <xsl:value-of select="name()"/>
28          </td>
29          <td>
30             <xsl:value-of select="local-name()"/>
31          </td>
32          <td>
33             <xsl:value-of select="text()"/>
34          </td>
35       </tr>
36       <xsl:apply-templates select="*"/>
37   </xsl:template>
38   </xsl:stylesheet>
```

Node-Set Functions 4-4

Formatted Output

Node-set Function

namespace-uri()	name()	local-name	text()
http://www.BookCatalog.com	Catalog	Catalog	
http://www.BookCatalog.com	Book	Book	
http://www.BookCatalog.com	Author	Author	Gambardella, Matthew
http://www.BookCatalog.com	Title	Title	XML Developer's Guide
http://www.BookCatalog.com	Genre	Genre	Computer
http://www.BookCatalog.com	Price	Price	44.95
http://www.BookCatalog.com	Publish	Publish	2000-10-01
http://www.BookCatalog.com	Description	Description	An in-depth look at creating applications with XML.

Boolean Functions 1-5

- boolean(arg)
  - Returns a `boolean` value for a number, string, or node-set.

Syntax

```
fn:boolean(arg)
```

Code Snippet

```
<ul>
    <li><b>boolean(0)</b> = <xsl:value-of select="boolean(0)"/></li>
    <li><b>boolean(1)</b> = <xsl:value-of select="boolean(1)"/></li>
    <li><b>boolean(-100)</b> = <xsl:value-of select="boolean(-100)"/></li>
    <li><b>boolean('hello')</b> = <xsl:value-of select="boolean('hello')"/></li>
    <li><b>boolean('')</b> = <xsl:value-of select="boolean('')"/></li>
    <li><b>boolean(//book)</b> = <xsl:value-of select="boolean(//book)"/></li>
</ul>
```

```
boolean(0) = false
boolean(1) = true
boolean(-100) = true
boolean('hello') = true
boolean('') = false
boolean(//book) = false
```

Boolean Functions 2-5

- not(arg)
  - The sense of an operation can be reversed by using the `not()` function.

Syntax

```
fn:not(arg)
```

Boolean Functions 3-5

- not(arg)
  - The sense of an operation can be reversed by using the `not()` function.

Code Snippet

```
<xsl:template match="PRODUCT[not(position()=1)]">
  <xsl:value-of select="."/>
</xsl:template>
```

The template rule selects all product elements that are not the first child of their parents.

Code Snippet

```
<xsl:template match="PRODUCT[position()!=1]">
  <xsl:value-of select="."/>
</xsl:template>
```

The same template rule could be written using the not equal operator != instead.

Boolean Functions 4-5

- true()
  - Returns the boolean value `true`.

Syntax

```
fn:true()
```

Code Snippet

```
<xsl:value-of select="true()"/>
```

```
true
```

Boolean Functions 5-5

- false()
  - Returns the boolean value `true`.

Syntax

```
fn:false()
```

Code Snippet

```
<xsl:value-of select="false() or false()"/>
<xsl:value-of select="true() and false()"/>
<xsl:value-of select="false() and false()"/>
```

```
true
false
false
```

Numeric Functions 1-5

- Return strings or numbers.
- Can be used with comparison operators in filter patterns such as:
  - `number(arg)`
  - `ceiling(num)`
  - `floor(num)`
  - `round(num)`

Syntax

```
number(arg)
fn:number(arg)

ceiling(num)
fn:ceiling(num)

floor(num)
fn:floor(num)

round(num)
fn:round(num)
```

where,

`number(arg)` : Returns numeric value of argument. Argument could be a boolean, a string, or a node-set.

`ceiling(num)` : Returns smallest integer greater than the number argument.

`floor(num)` : Returns largest integer that is not greater than the number argument.

`round(num)` : Rounds the number argument to the nearest integer.

Numeric Functions 2-5

Stylesheet(1-3)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Number.xsl"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <h3>Numeric Functions</h3>
        <ul>
          <li>
            <b>number('1548')</b> =
            <xsl:value-of select="number('1548')"/>
          </li>
          <li>
            <b>number('-1548')</b> = <xsl:value-of select="number('-1548')"/>
          </li>
          <li>
            <b>number('text')</b> = <xsl:value-of select="number('text')"/>
          </li>
          <li>
            <b>number('226.38' div '1')</b> = <xsl:value-of select="number('226.38'
            div '1')"/>
          </li>
        </ul>
```

Numeric Functions 3-5

Stylesheet(2-3)

```
<ul>
  <li>
    <b>ceiling(2.5)</b> = <xsl:value-of select="ceiling(2.5)"/>
  </li>
  <li>
    <b>ceiling(-2.3)</b> = <xsl:value-of select="ceiling(-2.3)"/>
  </li>
  <li>
    <b>ceiling(4)</b> = <xsl:value-of select="ceiling(4)"/>
  </li>
</ul>
<ul>
  <li>
    <b>floor(2.5)</b> = <xsl:value-of select="floor(2.5)"/>
  </li>
  <li>
    <b>floor(-2.3)</b> = <xsl:value-of select="floor(-2.3)"/>
  </li>
  <li>
    <b>floor(4)</b> = <xsl:value-of select="floor(4)"/>
  </li>
</ul>
```

Numeric Functions 4-5

Stylesheet(3-3)

```
        <ul>
          <li>
            <b>round(3.6)</b> = <xsl:value-of select="round(3.6)"/>
          </li>
          <li>
            <b>round(3.4)</b> = <xsl:value-of select="round(3.4)"/>
          </li>
          <li>
            <b>round(3.5)</b> = <xsl:value-of select="round(3.5)"/>
          </li>
          <li>
            <b>round(-0.6)</b> = <xsl:value-of select="round(-0.6)"/>
          </li>
          <li>
            <b>round(-2.5)</b> = <xsl:value-of select="round(-2.5)"/>
          </li>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Numeric Functions 5-5

Output

Numeric Functions

- **number('1548')** = 1548
- **number('-1548')** = -1548
- **number('text')** = NaN
- **number('226.38' div '1')** = 226.38

- **ceiling(2.5)** = 3
- **ceiling(-2.3)** = -2
- **ceiling(4)** = 4

- **floor(2.5)** = 2
- **floor(-2.3)** = -3
- **floor(4)** = 4

- **round(3.6)** = 4
- **round(3.4)** = 3
- **round(3.5)** = 4
- **round(-0.6)** = -1

String Functions 1-5

- String functions are used to:
  - Evaluate
  - Format and manipulate string arguments
  - Convert an object to a string
- Different String functions are:
  - `string(arg)`
  - `compare()`
  - `concat()`
  - `substring()`

String Functions 2-5

Syntax

string(arg)

```
fn:string(arg)
```

translate()

```
fn:translate(string, string, string)
```

concat()

```
fn:concat(string,string,...)
```

substring()

```
fn:substring(string,start,len)
fn:substring(string,start)
```

where,

`string(arg)`: Returns string value of the argument. The argument could be a number, boolean, or node- set.

`translate()`: Returns first argument string with occurrences of characters in second argument string replaced by character at the corresponding position in third argument string.

`concat()`: Returns concatenation of the strings.

`substring()`: Returns substring from the start position to specified length.

String Functions 3-5

Code Snippet

```
1  <?xml version="1.0"?>
2  <?xml-stylesheet type="text/xsl" href="BookDetail.xsl"?>
3  <BookStore>
4    <Book>
5      <Title>The Weather Pattern</Title>
6      <Author>Weather Man</Author>
7      <Price>100.00</Price>
8    </Book>
9    <Book>
10     <Title>Weaving Patterns</Title>
11     <Author>Weaver</Author>
12     <Price>150.00</Price>
13   </Book>
14   <Book>
15     <Title>Speech Pattern</Title>
16     <Author>Speaker</Author>
17     <Price>15.00</Price>
18   </Book>
19   <Book>
20     <Title>Writing Style</Title>
21     <Author>Writer</Author>
22     <Price>1500.00</Price>
23   </Book>
24 </BookStore>
```

String Functions 4-5

Stylesheet

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="BookDetail.xsl"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>example</title>
      </head>
      <body>
        <xsl:value-of select='translate("----------------Bboks-------
        ---------","bok","ook")'/>
        <br/>
        <xsl:value-of select="concat('AWARD WINNING', 'BOOK DETAILS')"/>
        <xsl:apply-templates select="//Book"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="Book">
    <xsl:if test="contains(Title, 'Pattern')">
      <DIV>
        <B>
          <xsl:value-of select="Title"/>
        </B>
        by
        <I>
          <xsl:value-of select="Author"/>
        </I>
        costs
        <xsl:value-of select="Price"/>
        .
      </DIV>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

String Functions 5-5

Output

----------Books------- ---------
AWARD WINNINGBOOK DETAILS
The Weather Pattern by *Weather Man* costs 100.00 .
Weaving Patterns by *Weaver* costs 150.00 .
Speech Pattern by *Speaker* costs 15.00 .

Result Tree Fragments

- A portion of an XML document that is not a complete node or set of nodes.
- The only allowed operation in a result tree fragment is on a string.
- The operation on the string may involve first converting the string to a number or a boolean.
- Result tree fragment is an additional data type other than four basic XPath data types, such as, string, number, boolean, node-set.
- A result tree fragment represents a fragment of the result tree.
- It is not permitted to use the /, //, and [] XPath operators on Result tree fragments.

Lesson 3 – Working with different Styles

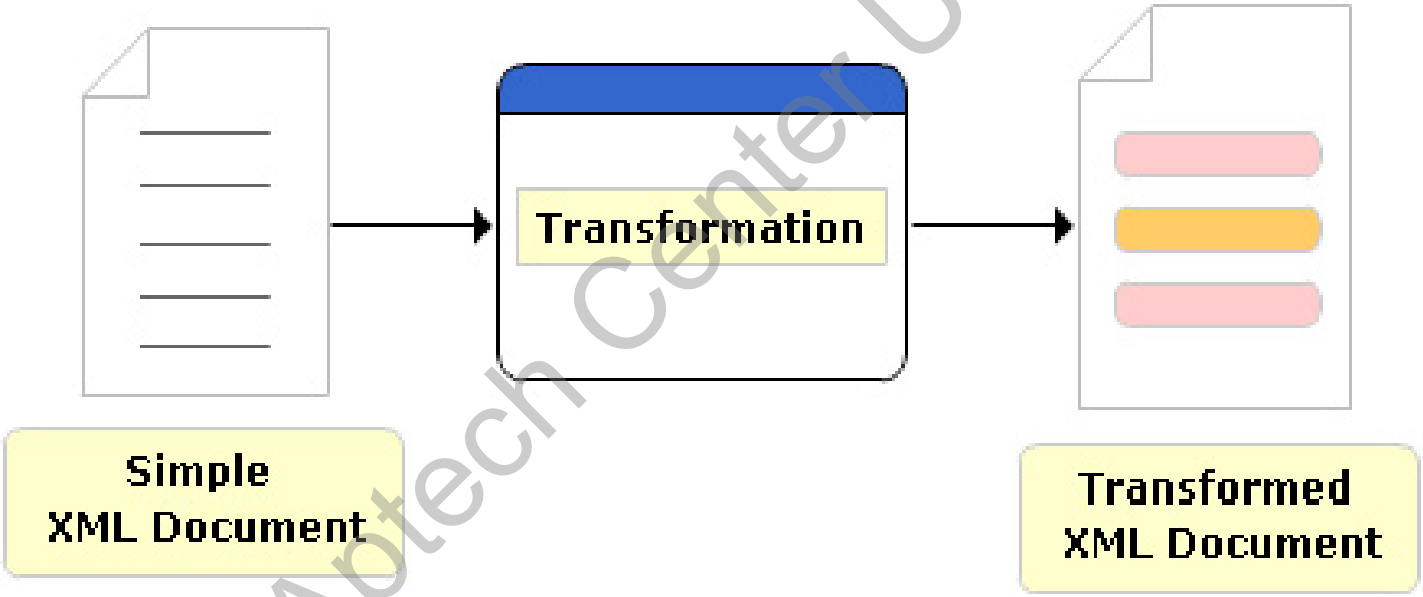
In this last lesson, **Working with different Styles**, you will learn to:

- Explain how to switch between styles.
- Describe how to transform XML documents into HTML using XSLT.

Transformation of XML Document 1-2

- Transformation is one of the most important and useful techniques for working with XML.
- XML can be transformed by changing its structure, its markup, and perhaps its content into another form.
- The most common reason to transform XML is to extend the reach of a document into new areas by converting it into a presentational format.
- Uses of transformation:
  - Formatting a document to create a high-quality presentational format.
  - Changing one XML vocabulary to another.
  - Extracting specific pieces of information and formatting them in another way.
  - Changing an instance of XML into text.
  - Reformatting or generating content.

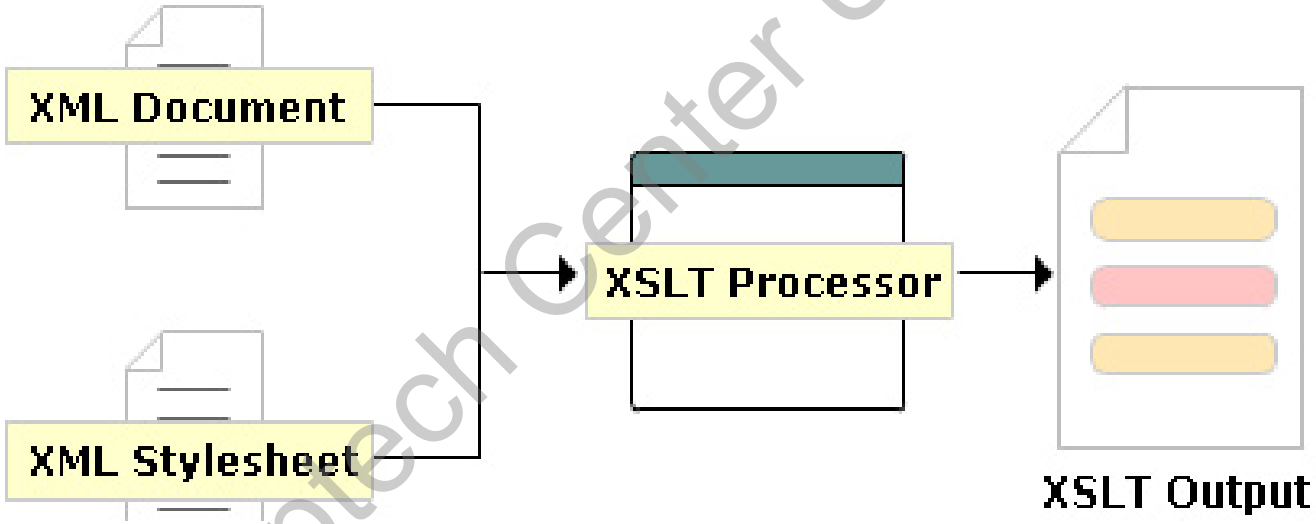
Transformation of XML Document 2-2



Transformation using XSLT Processor 1-2

- Takes two inputs:
  - An XSLT stylesheet to govern the transformation process.
  - An input document called the source tree.
- The output is called the result tree.
- The XSLT engine begins by reading in the XSLT stylesheet and caching it as a look-up table.
- XPath locates the parts of XML document such as Element nodes, Attribute nodes and Text nodes.
- For each node the XSLT processes, it will look in the table for the best matching rule to apply.
- Starting from the root node, the XSLT engine finds rules, executes them, and continues until there are no more nodes in its context node set to work with.
- At that point, processing is complete and the XSLT engine outputs the result document.

Transformation using XSLT Processor 2-2



Transforming XML using XSLT 1-2

- **Step 1**
  - Creates a normal XML document.

Code Snippet

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

- **Step 2**
  - Add the following lines.

Code Snippet

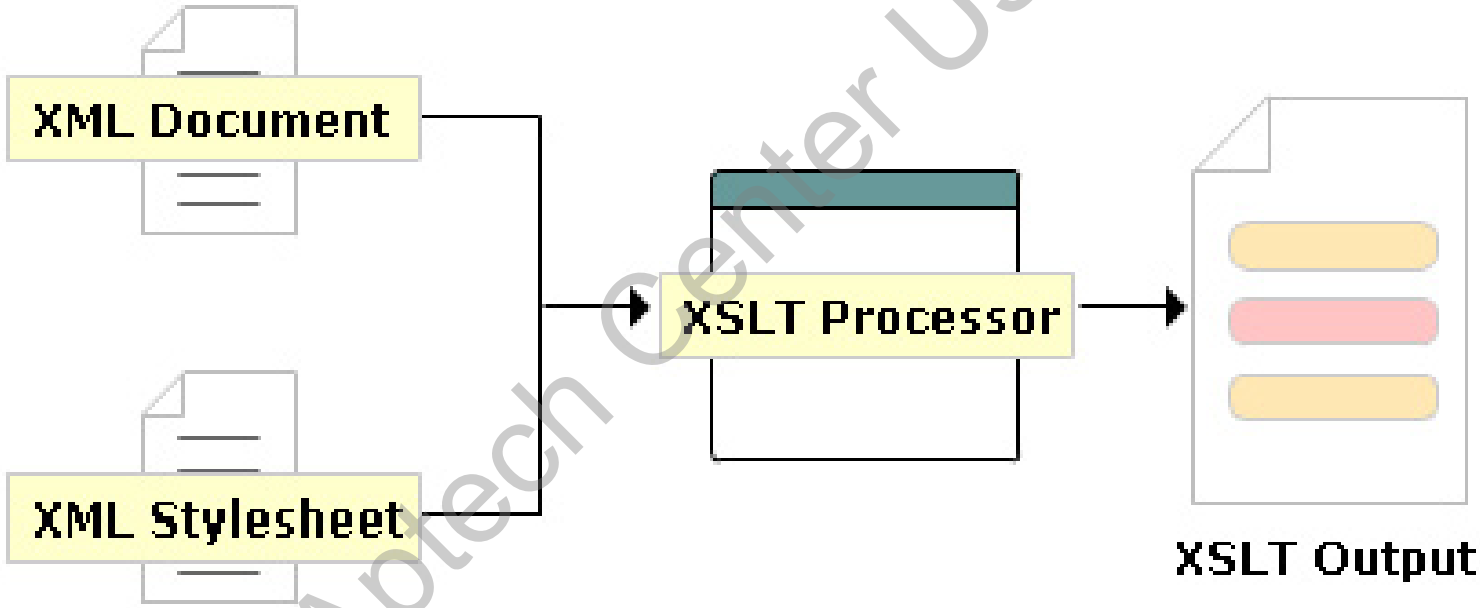
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
   ...
</xsl:stylesheet>
```

- **Step 3**
  - Set it up to produce HTML-compatible output.
  - The `<xsl:output>` tag should be used with either "text" or "html" to output anything besides well-formed XML.
  - The default value is "xml".

Code Snippet

```
<xsl:stylesheet>
   <xsl:output method="html"/>
   ...
</xsl:stylesheet>
```

Transforming XML using XSLT 2-2



Transforming XML using XSLT Example 1-2

- Example code of transforming XML documents into HTML using XSLT processor

Code Snippet

```
1  <?xml version="1.0"?>
2  <?xml-stylesheet type="text/xsl" href="ProductInfo.xsl"?>
3
4  <Company>
5
6    <Product>
7      <Product_ID>S002</Product_ID>
8      <Name>Steel</Name>
9      <Price>1000/tonne</Price>
10     <Boiling_Point Units="Kelvin">20.28</Boiling_Point>
11     <Melting_Point Units="Kelvin">13.81</Melting_Point>
12   </Product>
13
14   <Product>
15     <Product_ID>S004</Product_ID>
16     <Name>Iron</Name>
17     <Price>5000/tonne</Price>
18     <Boiling_Point Units="Kelvin">34.216</Boiling_Point>
19     <Melting_Point Units="Kelvin">23.81</Melting_Point>
20   </Product>
21
22 </Company>
```

where,

`Company`: The root `Company` element contains `Product` child elements.

`Units`: Attribute specifies the units for products melting point and boiling point.

Transforming XML using XSLT Example 2-2

Stylesheet

```
1  <?xml version="1.0"?>
2
3  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4
5    <xsl:template match="Company">
6      <html>
7        <xsl:apply-templates/>
8      </html>
9    </xsl:template>
10
11   <xsl:template match="Product">
12     <P>
13       <xsl:apply-templates/>
14     </P>
15   </xsl:template>
16
17 </xsl:stylesheet>
```

Output

S002Steel1000/tonne20.2813.81

S004Iron5000/tonne34.21623.81

Summary 1-2

- **XPath**
  - Notation for retrieving information from a document.
  - Provides a common syntax for features shared by Extensible Stylesheet Language Transformations (XSLT) and XQuery.
  - Have seven types of node as Root, Element, Attribute, Text, Comment, Processing instruction and Namespace.
  - Used in the creation of the patterns.

- **XPath Expressions and Functions**
  - The four types of expressions in XPath are
    - Node-sets.
    - Booleans.
    - Numbers.
    - Strings.

Summary 2-2

- **XPath Expressions and Functions (contd...)**
  - Functions defined for XPath are `Accessor`, `AnyURI`, `Node`, `Error` and `Trace`, `Sequence`, `Context`, `Boolean`, `Duration/Date/Time`, `String`, `QName` and `Numeric`.
  - A Result tree fragment is a portion of an XML document that is not a complete node or set of nodes.

- **Working with different styles**
  - Transformation is one of the most important and useful techniques for working with XML.
  - To transform XML is to change its structure, its markup, and perhaps its content into another form.
  - Transformation can be carried out using an XSLT processor also.