

# IBM XML certification success, Part 2: Prepare for IBM XML certification with more technologies

Review XPath, XSLT, XLink, XPointer, CSS, XSL-FO, SAX, and DOM

Skill Level: Introductory

[Pradeep Chopra \(authors@whizlabs.com\)](mailto:authors@whizlabs.com)

Cofounder  
WHIZlabs Software

[Hari Vignesh Padmanaban \(authors@whizlabs.com\)](mailto:authors@whizlabs.com)

Software engineer  
Invensys Foxboro

31 May 2005

This is the second part of a three-part tutorial series designed specifically for those interested in taking the IBM Certified Solution Developer Exam for XML and Related Technologies. Here, authors Pradeep Chopra and Hari Vignesh Padmanaban follow up on the lessons in Part 1 by introducing the reader to several more critical XML technologies like XPath, XSLT, XLink, XPointer, CSS, XSL-FO, SAX, and DOM. Furthermore, the authors reinforce the reader's understanding through examples and exercises.

## Section 1. Before you start

### About this tutorial

This tutorial is the second part of a three-part tutorial series designed specifically for those interested in taking the [IBM Certified Solution Developer exam for XML and](#)

## Related Technologies.

The first two parts cover XML technologies that are fundamental to the exam. Part 1 covered XML basics, Document Type Definitions (DTDs), W3C XML Schema, Web services, and security. Here in Part 2, the authors focus on:

- XML Path Language (XPath)
- Extensible Stylesheet Language Transformations (XSLT)
- XML Linking Language (XLink)
- XML Pointer Language (XPointer)
- Cascading Style Sheets (CSS)
- Extensible Stylesheet Language -- Formatting Objects (XSL-FO)
- Simple API for XML (SAX)
- The Document Object Model (DOM)

The third part will cover the objectives of the actual exam in detail.

In all three tutorials, each section of the main lesson includes:

- An example that illustrates the theory covered in the section
- An exercise for testing your skills
- An exam tip that relates to the topics covered in the section

**Note:** The XML topics covered in each section of the tutorial pertain specifically to the XML certification exam and are not intended to be a comprehensive overview of XML technology.

**Prerequisite:** It is not required, but highly recommended that you read [Part 1](#) before starting with this tutorial.

## About the XML certification exam

The XML certification exam offered by IBM tests the user's knowledge of XML and various XML-related technologies.

The exam covers the following topics:

- XML

- Web services
- XML Path Language (XPath)
- Extensible Stylesheet Language Transformations (XSLT)
- Simple API for XML (SAX)
- The Document Object Model (DOM)
- Extensible Stylesheet Language -- Formatting Objects (XSL-FO)
- Document Type Definitions (DTDs)
- W3C XML Schema
- XML security

It is one of only a few certifications available for XML. Unlike other certifications, the XML exam requires you to not only understand these technologies, but also be able to apply them to solve real-world situations. As a result, most of the questions in the XML exam are scenario-based questions in which you are expected to choose the most appropriate solution for a given scenario. This makes it more challenging than other exams.

In particular, preparation for this exam should emphasize the importance of **practice**; anyone seriously considering XML certification should have broad experience with XML-related products *and* technologies.

For more information on the exam, visit the [information page for the IBM XML certification exam](#) where you'll find:

- Details on the **job role** and **target audience** for whom the certification was built
- **Recommended prerequisites** for the knowledge and skills one should possess before considering this certification
- The **test objectives** and the **skills** measured by the exam
- Recommended **educational resources** to prepare you for the test, based on the test objectives
- A **pre-assessment/sample test** to gauge your readiness for the actual exam

**Note:** *This tutorial was produced by an independent (non-IBM) organization, and is not necessarily endorsed by the IBM Certification Team.*

For general information about the IBM Professional Certification Program, visit

<http://www-03.ibm.com/certify/>.

---

## Section 2. XPath

### XPath overview

**XPath basics:** XPath is a declarative language that is used for referring to parts of XML documents. XPath expressions are used for locating a set of nodes in a given XML document. Many XML technologies, like XSLT and XQuery, use XPath extensively. To use these technologies, you'll need to understand the basics of XPath.

**XPath syntax:** The syntax of XPath is composed of two parts, location paths and expressions. To use XPath effectively, it is important to know the syntax and usage of these two parts.

### Expressions and location paths

**Expressions** allow you to select specific nodes in an XML document. With XML documents, the most used expression type is the **location path**. When used, the location path results in a node set that contains selected nodes from a given XML document.

A location path has the following syntax:

```
AXIS :: NODE_TEST [ PREDICATE ]
```

Where:

- **AXIS** is the relationship between the current node and the selected nodes that are returned by the location path
- **NODE\_TEST** indicates the node type that is selected by the location path
- **PREDICATE** is the value (zero or more) that allows you to filter selected nodes from the resulting node set

### Example

The following example XML document (product.xml) illustrates XPath:

```

<!-- product.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<product id="200">
  <name>UML Exam Simulator</name>
  <price>100</price>

  <topics name="UML">
    <topic id="UML_SM">
      <name>Static Modeling</name>
      <questions>100</questions>
    </topic>

    <topic id="UML_AE">
      <name>Architecture</name>
      <questions>80</questions>
    </topic>

    <topic id="UML_DM">
      <name>Dynamic Modeling</name>
      <questions>67</questions>
    </topic>
  </topics>
</product>

```

## Expressions and location paths: Axis, node test, and predicate

The **axis** defines the relationship between the current node and the selected nodes that are returned by the location path. The **node test** indicates the node type that is selected by the location path. A **predicate** allows you to filter selected nodes from the resulting node set.

In XPath, you can specify 13 types of axes, described in the following table. A current node is chosen from the document in [Expressions and location paths](#), then the respective axis is applied on that node and the result is explained.

Axis and Description	Current node	Expression and explanation
ancestor  Selects all the ancestors of the current node.	<topic id="UML_SM">	<b>ancestor::topics</b> selects <topics name="UML">.  <b>ancestor::*</b> selects <topics name="UML"> and <product id="200">.
ancestor-or-self  Selects the current node and its ancestors.	<topic id="UML_SM">	<b>ancestor-or-self::topics</b> selects <topics name="UML">.  <b>ancestor-or-self::*</b> selects <topics name="UML"> and <topic id="UML_SM">.

<p>attribute</p> <p>Selects the attributes of the current node.</p>	<p><code>&lt;topic id="UML_SM" &gt;</code></p>	<p><b>attribute::id</b> selects the id attribute.</p>
<p>child</p> <p>Selects all the child nodes of the current node.</p>	<p><code>&lt;topics name="UML" &gt;</code></p>	<p><b>child::topic</b> selects the following nodes: <code>&lt;topic id="UML_SM"&gt;</code>, <code>&lt;topic id="UML_AE"&gt;</code>, and <code>&lt;topic id="UML_DM"&gt;</code>.</p>
<p>descendant</p> <p>Selects all the descendants of the current node.</p>	<p><code>&lt;topics name="UML" &gt;</code></p>	<p><b>descendant::questions</b> selects the following nodes: <code>&lt;questions&gt;100&lt;/questions&gt;</code>, <code>&lt;questions&gt;80&lt;/questions&gt;</code>, and <code>&lt;questions&gt;67&lt;/questions&gt;</code>.</p>
<p>descendant-or-self</p> <p>Selects the current node and its descendants.</p>	<p><code>&lt;topics section="1" &gt;</code></p>	<p><b>descendant-or-self::questions</b> selects the following nodes: <code>&lt;questions&gt;100&lt;/questions&gt;</code>, <code>&lt;questions&gt;80&lt;/questions&gt;</code>, and <code>&lt;questions&gt;67&lt;/questions&gt;</code>.</p> <p><b>descendant-or-self::*</b> selects the following nodes: <code>&lt;questions&gt;100&lt;/questions&gt;</code>, <code>&lt;questions&gt;80&lt;/questions&gt;</code>, and <code>&lt;questions&gt;67&lt;/questions&gt;</code>, as well as the current node, <code>&lt;topics section="1" &gt;</code> .</p>
<p>following</p> <p>Selects all the nodes after the current node.</p>	<p><code>&lt;topic id="UML_AE" &gt;</code></p>	<p><b>following::questions</b> selects the node <code>&lt;questions&gt;67&lt;/questions&gt;</code>.</p>
<p>following-sibling</p> <p>Selects all the nodes after the current node that have the same parent as the current node (sibling nodes).</p>	<p><code>&lt;topic id="UML_SM" &gt;</code></p>	<p><b>following-sibling::topic</b> selects the following nodes: <code>&lt;topic id="UML_AE"&gt;</code> and <code>&lt;topic id="UML_DM"&gt;</code>.</p>
<p>namespace</p> <p>Selects all the nodes that belong to the same namespace</p>	<p><code>&lt;topic id="UML_DM" &gt;</code></p>	<p><b>namespace::*</b> returns the default XML namespace <code>http://www.w3.org/XML/1998/namespace</code>, as no namespace was defined</p>

as the current node.		for the element.
parent  Selects the parent node of the current node.	<code>&lt;topic id="UML_SM"&gt;</code>	<b>parent::node()</b> selects <code>&lt;topics name="UML"&gt;</code> .
preceding  Selects all the nodes before the current node.	<code>&lt;topic id="UML_DM"&gt;</code>	<b>preceding: questions</b> selects <code>&lt;questions&gt;80&lt;/questions&gt;</code> and <code>&lt;questions&gt;100&lt;/questions&gt;</code> .
preceding-sibling  Selects all the nodes before the current node that have the same parent as the current node (sibling nodes).	<code>&lt;topic id="UML_DM"&gt;</code>	<b>preceding-sibling::topic</b> selects <code>&lt;topic id="UML_AE"&gt;</code> and <code>&lt;topic id="UML_SM"&gt;</code> .
self  Selects the current node.	<code>&lt;topic id="UML_AE"&gt;</code>	<b>self::node()</b> selects <code>&lt;topic id="UML_AE"&gt;</code> .

The following table demonstrates the use of predicates. Assume that the current selected node for all the XPath Expressions is the `<product id="200">` node.

Expression with predicate	Output and Explanation
<code>child::topics/child::topic [attribute::id="UML_SM"]</code>	Selects <code>&lt;topic id="UML_SM"&gt;</code> .  The predicate tells the XPath processor to return the <code>topic</code> child element of <code>topics</code> that has its attribute <code>id</code> with the value "UML_SM". In this case, it is the first topic element in the XML document.
<code>child::topics/child::topic [child::name="Dynamic Modeling"]</code>	Selects <code>&lt;topic id="UML_DM"&gt;</code> .  The predicate tells the XPath processor to return the <code>topic</code> child element of <code>topics</code> that has its <code>child</code> <code>name</code> element with the value "Dynamic Modeling". In this case, it is the third topic element in the XML document.
<code>child::topics/child::topic [position() = 2 ]</code>	Selects <code>&lt;topic id="UML_AE"&gt;</code> .  The predicate here uses a XPath function

`position()`. XPath functions are described later in this section. The predicate here tells the XPath processor to get the second `topic` element in the XML document.

## Expressions and location paths: Abbreviations

You can use abbreviated forms for certain axes. Those abbreviations are:

Abbreviation	Description and use
@	Replaces <code>attribute::</code> .
//	Replaces <code>descendant-or-self::node()</code> .
.	Replaces <code>self::node()</code> .
..	Replaces <code>parent::node()</code> .
None	Replaces <code>child::</code> (this is the default axis used if the XPath expression doesn't mention a node).

If you use the asterisk (\*), you can select all the children of a particular node. For example, `topics/*` selects all the child nodes under `topics`.

## XPath functions

Using XPath, you can perform operations on numbers, compare values using relational operators, and test for equality. XPath supports the `+`, `-`, `*`, `div`, and `mod` operators. Note that XPath uses `div` instead of `/`. For comparing values, XPath uses the following:

- The `=` and `!=` equality operators
- The `>`, `<`, `>=`, and `<=` relational operators
- The `and` and `or` boolean operators

Apart from these operators, XPath defines standard functions that you can use to perform operations on nodes, strings, numerical values and boolean values. The functions are divided into the following four categories:

- Node set functions
- Number functions
- String functions

- Boolean functions

## XPath functions: Node set functions

To perform operations on nodes, use node set functions. XPath provides seven node set functions, which are described in the following table:

Name and syntax	Description
<code>count (node-set )</code>	Returns the number of nodes that are in the node set.
<code>id (value)</code>	Returns a set of nodes that match a given ID attribute value.
<code>last ( )</code>	Returns the position of the last node in the node set.
<code>local-name (node)</code>	Returns the local part of a node's name.
<code>Name (node)</code>	Returns the fully qualified name of the node (including the prefix).
<code>namespace-uri (node)</code>	Returns the namespace URI of the node.
<code>position ( )</code>	Returns the position of the current node in the node set.

## XPath functions: Number functions

To perform numeric calculations, use number functions. XPath provides five number functions, which are described in the following table:

Name and syntax	Description
<code>ceiling (number)</code>	Returns an integer value that is equal to or greater than the specified number.
<code>floor (number)</code>	Returns an integer value that is equal to or less than the specified number.
<code>number (argument)</code>	Converts the given argument to a number.
<code>round (number)</code>	Returns the closest integer to the given number.
<code>sum (node-set)</code>	Returns the sum of the numerical values in the specified node set.

## XPath functions: String functions

To perform operations on strings, use string functions. XPath provides 10 string

functions, which are described in the following table:

Name and syntax	Description
<code>concat(string1, string2, stringN)</code>	Returns a string containing the concatenation of the specified string arguments.
<code>contains(string1, string2)</code>	Checks to see if string1 contains string2 as a substring, and returns true or false.
<code>normalize-space(string)</code>	Returns the given string with no leading or trailing whitespaces, and removes sequences of whitespaces by replacing them with a single whitespace.
<code>starts-with(string1, string2)</code>	Checks to see if string1 starts with string2.
<code>string(argument)</code>	Converts the given argument to a string.
<code>string-length(string)</code>	Returns the length of the given string.
<code>substring(string, startPosition, endPosition)</code>	Returns a part of the string argument based on the <code>startPosition</code> and <code>endPosition</code> specified in the function.
<code>substring-after(string1, string2)</code>	Returns the portion of string1 that comes after the occurrence of string2 (which is a subset of string1).
<code>substring-before(string1, string2)</code>	Returns the portion of string1 that comes before string2 (which is a subset of string1).
<code>translate(string, characterSequence1, characterSequence2 )</code>	Returns a string by translating the characters in <code>characterSequence1</code> with corresponding positional characters in <code>characterSequence2</code> .

## XPath functions: Boolean functions

Boolean functions in XPath are used to get true or false values directly, or to convert an argument to either true or false. To perform Boolean operations in XPath, use the five Boolean functions described in the following table:

Name and syntax	Description
<code>boolean(argument)</code>	Converts the argument to Boolean.
<code>false()</code>	Returns false.
<code>lang(language)</code>	Checks to see if the given language matches the language specified by the <code>xsl:lang</code> element.
<code>not(condition)</code>	Returns the opposite value of the condition argument (that is, returns true for false condition, and vice versa).
<code>true()</code>	Returns true.

## Summary

XPath is the language that is used to select nodes in an XML document. A location path is used to select the nodes. It has three parts -- axis, node test, and predicate. XPath has 13 types of axes, and provides support for arithmetic, relational, and equality operators. The built-in functions of XPath enable you to translate and perform calculations on selected nodes. The four categories of functions provided by XPath are node set functions (7), numeric functions (5), string functions (10), and boolean functions (5). You will get a better understanding of the application of XPath when you see it being used with XSLT in [XSLT](#), [XLink](#), and [XPath](#) .

## Example

For this example, consider the same product.xml file presented earlier in this section:

```

<!-- product.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<product id="200">
  <name>UML Exam Simulator</name>
  <price>100</price>

  <topics name="UML">
    <topic id="UML_SM">
      <name>Static Modeling</name>
      <questions>100</questions>
    </topic>

    <topic id="UML_AE">
      <name>Architecture</name>
      <questions>80</questions>
    </topic>

    <topic id="UML_DM">
      <name>Dynamic Modeling</name>
      <questions>67</questions>
    </topic>
  </topics>
</product>

```

Here are some possible XPath expressions for selecting desired nodes in the above document and the values returned by them.

**Note:** Try to guess what the XPath expression might return and then view the result to see if you get it right. This will help you get a good grasp of XPath.

### XPath expression and Result

```
/child::product/child::topics/child::topic
```

Selects all three of the <topic> nodes under <topics>.

<code>/product/topics/topic</code>	Also selects all three of the <code>&lt;topic&gt;</code> nodes under <code>&lt;topics&gt;</code> . Remember, if you don't specify an axis the default substitution is <code>child::</code> .
<code>/product/child::topics/topic[position() = 2]</code>	Selects the second <code>&lt;topic&gt;</code> node under <code>&lt;topics&gt;</code> -- which in this case would be <code>&lt;topic id = "UML_AE"&gt;</code> .
<code>/product/topics/topic[position() = last()]</code>	Selects the last <code>&lt;topic&gt;</code> node under <code>&lt;topics&gt;</code> , which in this case would be <code>&lt;topic id = "UML_DM"&gt;</code> .
<code>//topic[position() = 1]/following::questions</code>	Returns the <code>&lt;questions&gt;</code> nodes after the first <code>&lt;topic&gt;</code> node element -- <code>&lt;questions&gt;80&lt;/questions&gt;</code> and <code>&lt;questions&gt;67&lt;/questions&gt;</code> .
<code>/topics/topic[position() = 2]/*</code>	Selects all child nodes of the second <code>&lt;topic&gt;</code> node -- in this case, the <code>&lt;name&gt;Architecture&lt;/name&gt;</code> and <code>&lt;questions&gt;80&lt;/questions&gt;</code> nodes.
<code>/topics/topic[position() = last()]</code>	No result. The expression has to be <code>/product/topics/topic[position() = last()]</code> ; by using <code>/</code> , you tell the expression to start from the root element of the document. In the document above, the root element is <code>product</code> , so the expression should start with <code>/product</code> .

The following table demonstrates some XPath functions and their results. Again, try to work out the output of the functions first, and then refer to the results.

XPath function	Result
<code>concat("xml", "is", "great")</code>	"xmlisgreat"
<code>normalize-space(" xml is great ")</code>	"xml is great"
<code>round(6.28)</code>	6
<code>sum(//topic[position() = 1]/following::questions)</code>	147 (80 + 67)
<code>floor(6.28)</code>	6
<code>count(//topic[position() = 1]/following::questions)</code>	2
<code>string-length("xml")</code>	3

<code>ceiling(6.28)</code>	7
<code>substring("xml is great", 1, 3)</code>	"xml"
<code>translate("xml is great man", 'xml', "XML is great Man" 'XML')</code>	

## Exercises

The questions for this exercise are based on the following books.xml file:

```
<!-- books.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book id="BK1">
    <name>Understanding XML</name>
    <Author>Whizlabs</Author>
    <price>50</price>
  </book>
  <book id="BK2">
    <name>Understanding Java</name>
    <Author>Whizlabs</Author>
    <price>40</price>
  </book>
  <book id="BK3">
    <name>XML Certification Made Easy</name>
    <Author>Whizlabs</Author>
    <price>75</price>
  </book>
</books>
```

- Which of the following XPath expressions will select the first book element?
  - `//book[position() = first()]`
  - `/book[position() = 1]`
  - `//book[position() = 1]`
  - `/books/book[position() = 1]`
  - `/books/book [position() = first()]`
  - `//books/book[element() = 1]`

**Correct choices: C and D**

### Explanation:

The `//` expression selects all of the `book` elements in the document, and

`position() = 1` returns the first `book` element. Choice C is correct.

The `/books/book` expression returns the `book` element located under the `books` root element; `position() = 1` then returns the first `book` element. Choice D is correct.

When you start an expression with `/`, the expression always denotes the root element of the document. In this case, `books` is the root element and is specified whenever you use `/`.

XPath does not have a function called `first()` -- it only has a `last()` function. Choices A and E are incorrect.

Choice B is incorrect as the `books.xml` file has no root element called `book`.

XPath does not have a function called `element()`. Choice F is incorrect.

2. Which of the following XPath expressions will select the `price` element in the third `book` element?

- A. `//book[position() = last()]/price`
- B. `//book[@id = 'BK3']/price`
- C. `//book[position() = 3]/price`
- D. `/books/book[position() = 3]/price`

**Correct choices: A, B, C, and D**

### Explanation:

This is a tricky question -- all the choices are correct.

The `//` expression selects all of the specified descendant elements in the document, and `position()` returns the specified position of that element in the resulting node set. In this case, `position() = 3` and `position = last()` both return the third `book` element when the node set contains `book` elements, and return the third `price` element when the node set contains `price` elements.

The `//book[position() = last()]/price` expression returns the `price` element in the last `book` element (the third `book` element). Choice A is correct.

The `//book[@id = 'BK3']/price` expression returns the price element in the book element, which has an id attribute with the value "BK3". Choice B is correct.

The `//book[position () = 3]/price` expression returns the price element in the third book element. Choice C is correct.

The `/books/book[position() = 3]/price` expression returns the price element in the third book element under the books root element. Choice D is correct.

## Exam tips

Your understanding and mastery of XPath is crucial to success in the XML certification exam. If you don't know XPath, you won't be able to write or understand XSLT stylesheets. Although few questions directly pertain to XPath, many of the XSLT-related questions require that you understand XPath. Pay close attention to *all* of the 27 functions provided by XPath, particularly their functionality and syntax.

---

## Section 3. XSLT, XLink, and XPointer

### XSLT basics

**Extensible Stylesheet Language Transformations**, or **XSLT**, is a language that specifies how to transform XML documents into other XML documents. It also allows you to transform XML documents into certain non-XML formats (such as text or HTML). To achieve a transform, you apply an XSLT stylesheet to an XML document using an XSL processor.

**Definition and linking:** An XSLT stylesheet file is an XML file. Once you write an XSLT stylesheet using elements defined in the XSLT namespace, you can refer to it from an XML document that you want to transform. To do this, use a processing instruction with the target `xml-stylesheet`. For example, to refer to a stylesheet named `product.xslt` from the XML document, use the following processing instruction:

```
<?xml-stylesheet type="text/xsl" href="product.xslt"?>
```

## XSLT elements: xsl:stylesheet, xsl:template, and xsl:apply-templates

An XSLT stylesheet contains elements that are defined in the XSLT namespace (<http://www.w3.org/1999/XSL/Transform>). In this section, we describe some of the most important elements in the XSLT namespace that are also important for the exam:

### 1. **xsl:stylesheet**

This is the root element of an XSLT file. An `xsl:stylesheet` definition has the following two mandatory attributes:

- a. The namespace declaration for the XSLT namespace (in this case, `xsl`)
- b. The version attribute (the version required for the exam is 1.0)

And the code is:

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

### 2. **xsl:template**

The most important element in XSLT is the `xsl:template` element. XSLT stylesheets use templates to process XML documents. The `xsl:template` element has four attributes: `match`, `name`, `priority`, and `mode`.

`match` is the most important attribute and is almost always specified in the `xsl:template` element. The `match` attribute denotes a pattern, which is an XPath expression that denotes the nodes in the XML document. If a match is found, then the template is executed.

The `name` attribute is optional, and is generally used by the `xsl:call-template` element (see [XSLT elements: xsl:variable, xsl:call-template, xsl:with-param, xsl:param, and others](#)).

The `priority` attribute is optional, and specifies the execution order for templates that match for the same XPath expression. The `priority` attribute takes a numeric value. A template with a high priority is chosen over a template with a lower priority.

The `mode` attribute is also optional. With it, you can match templates based on the value specified for the `mode` attribute and the expression specified in the match expression.

In the following expression, the template is executed for every `<book>` node in the XML document:

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

When you specify a forward slash (/) as the path expression, it denotes the root of the document. Similarly, you can specify an asterisk (\*) to match all the nodes in a document.

### 3. **xsl:apply-templates**

As mentioned before, XSLT uses templates to process XML documents. Once the XSL processor finds a corresponding match in an `xsl:template` element, the code in the template is executed. Inside the template definition, you can call other templates with the `xsl:apply-templates` element. For example:

```
1. <xsl:template match="product">
2.   <xsl:apply-templates select="name" />
3. </xsl:template>
4. <xsl:template match="name">
5.   <xsl:value-of select="." />
6. </xsl:template>
```

In the above expressions, the `template` element in line 1 matches for a `product` element in the document. Then, for that `product` element, a template with a match expression of the `name` node is called in line 2. This matches the `template` element in line 4. Line 5 simply outputs the value of the `name` node using the `xsl:value-of` element, which we describe next.

## XSLT elements: `xsl:value-of`, `xsl:output`, `xsl:if`, `xsl:choose`, `xsl:when`, and `xsl:otherwise`

### 4. **xsl:value-of**

Use this element to output the value of a node. The value is expressed as a string. `xsl:value-of` is an empty element and does not have any children. The value to be output is expressed in this element's `select` attribute as an XPath expression, as shown in the

`xsl:apply-templates` example in [XSLT elements: xsl:stylesheet, xsl:template, and xsl:apply-templates](#).

## 5. **xsl:output**

You can control the output of an XSLT stylesheet with the `xsl:output` element. This element is generally declared immediately after the `xsl:stylesheet` element. It has several attributes, of which `method` is the most important. `method` specifies the type of output for the XSLT stylesheet, and takes one of three values: `xml`, `html`, or `text`.

```
<xsl:output method="xml" version="1.0"
  encoding="UTF-8" indent="yes"/>
```

In the above expression, the specified output of the XSLT stylesheet is "xml".

## 6. **xsl:if, xsl:choose, xsl:when, and xsl:otherwise**

You can specify conditional statements and thus control the transformation of an XML document with the following XSLT elements:

- a. `xsl:if` -- This element checks conditions and is similar to the "if" statement found in other programming languages. Specify the condition that you want to test in the `test` attribute. If the test evaluates to true, then the statements within the element are executed.

```
<xsl:template match="name">
  <xsl:if test="@position='developer'">
    <xsl:value-of select="." /> <!-- is a developer -->
  </xsl:if>
</xsl:template>
```

In the above expression, for the `name` node, the `xsl:if` element checks whether "developer" equals the value of the attribute `position` in the `name` element. If this is true, then the value of the `name` node is output.

- b. `xsl:choose`, `xsl:when`, and `xsl:otherwise` -- Unlike other programming languages, XSLT does not have an "if ..else if" construct to test for multiple conditions. Instead, it provides the `xsl:choose` element. With this element, you can test for multiple conditions. Each condition is tested with the `xsl:when` element, which is equivalent to the "if" statement. To test multiple conditions,

include several `xsl:when` child elements. An `xsl:choose` element can also have an `xsl:otherwise`. This is equivalent to the "else if" statement. If present, an `xsl:choose` element can contain only *one* `xsl:otherwise` element. When none of the expressions in `xsl:when` evaluate to true, then the statements inside `xsl:otherwise` are executed.

```
<xsl:template match="name">
  <xsl:choose>
    <xsl:when test="@position='developer'">
      <xsl:valueof select="." /> <!-- is a developer -->
    </xsl:when>

    <xsl:when test="@position='manager'">
      <xsl:valueof select="." /> <!-- is a manager -->
    </xsl:when>

    <xsl:otherwise>
      <xsl:valueof select="." /> <!-- should be an engineer -->
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

## XSLT elements: `xsl:for-each`, `xsl:element`, `xsl:attribute`, `xsl:import`, and `xsl:include`

### 7. `xsl:for-each`

With this element, you can iterate over a set of selected nodes. It has a `select` attribute that selects the nodes through an XPath expression. Once the nodes are selected, the statements inside the `xsl:for-each` element are executed for each node in the resulting node set.

```
<xsl:template match="names">
  <xsl:for-each select="name">
    <xsl:value-of select="." />
  </xsl:for-each>
</xsl:template>
```

The above `xsl:for-each` element selects the `name` nodes that are children of the `names` node, and for each `name` element the value of the node is selected.

### 8. `xsl:element` and `xsl:attribute`

Use these elements to create element and attribute nodes in the resulting XSLT stylesheet. In addition, XSLT also provides the following elements for creating XML elements in the resulting output:

`xsl:processing-instruction`, `xsl:comment`, and `xsl:text`.

```
<xsl:element name="position">
  <xsl:value-of select="@post" />
</xsl:element>
```

The above expression creates a `position` element and assigns the value of the `post` attribute as the `position` element's value. For example, if the `post` attribute has a value of "manager", then the following XML output results:

```
<position>manager</position>
```

#### 9. **xsl:import and xsl:include**

You can reuse an XSLT stylesheet using `xsl:import` and `xsl:include`. These elements make it possible to reuse templates defined in other stylesheets. Both are very similar and have the same syntax.

The main difference becomes clear when you use `xsl:import` to reuse any content that's defined in an external stylesheet. For example, if there is a template in the current stylesheet that has the same name as a template in an external stylesheet, then the internal one is given higher priority than the external one. However, in the case of `xsl:include`, the included template is given the same priority as the existing template. In both elements, the `href` attribute gives the link to the external stylesheet, as follows:

```
<xsl:import href="products.xslt" />
<xsl:include href="products.xslt" />
```

Definitions that are included using `<xsl:import>` have lower priority than definitions in the including stylesheet.

## XSLT elements: `xsl:variable`, `xsl:call-template`, `xsl:with-param`, `xsl:param`, and others

#### 10. **xsl:variable**

Use the `xsl:variable` element to define variables. Its mandatory `name` attribute specifies a name for the variable. With the optional `select` attribute, you can select the desired node. Once defined, you can then

refer to the variable by adding the dollar sign (\$) character before the variable name. An example is:

```
<xsl:template match="name">
  <xsl:variable name="post" select="@position" />
  <xsl:value-of select="$post" />
</xsl:template>
```

In the above example, the variable `post` is assigned the value of the `position` attribute belonging to the `name` element. It is then output by referencing it using the dollar sign (\$) as `$post`.

#### 11. **xsl:call-template, xsl:with-param, and xsl:param**

You can call a template from within an XSLT stylesheet and pass parameters to the template. This is similar to making a method call in other programming languages, and is achieved by using the `xsl:call-template`, `xsl:with-param`, and `xsl:param` elements.

To call a template, you need to know the template's name and specify it in the `name` attribute of the `xsl:call-template` element. Match this to the value of the `name` attribute that you specify in the template that you call.

If the template you are calling requires any parameters, you have to provide them using the `xsl:with-param` element inside `xsl:call-template`. `xsl:with-param` has `name` and `select` attributes; use the latter to specify a node value with an XPath expression.

In the template that you call, predefine the parameters with the `xsl:param` element. The following example illustrates how to use these elements:

```
1. <xsl:template name="Employee_Position">
2.     <xsl:param name="name" />
3.     <xsl:param name="position" />
4.     <xsl:value-of select="$name" /> ,
5.     <xsl:value-of select="$position" />
6. </xsl:template>
7. <xsl:template match="name">
8.     <xsl:call-template name="Employee_Position">
9.         <xsl:with-param name="name" select="." />
10.        <xsl:with-param name="position" select="@position" />
11.     </xsl:call-template>
12. </xsl:template>
```

In the above example, lines 1-4 define a template called `Employee_Position` which takes in two parameters called `name` and

position. It then outputs the value of the parameters as:

```
                Name: <xsl:value-of select="$name" />
Position: <xsl:value-of select="$position" />
```

Line 5 defines the template that executes for every matching `name` node in the XML document.

Lines 6-9 demonstrate the use of the `xsl:call-template` element. In line 6, the template call is made by specifying the name of the template in the `name` attribute. The parameters for the template (the values of the `name` node and its `position` attribute) are provided using the `xsl:with-param` element, as shown in lines 7 and 8.

### Other XSLT elements

In addition to the elements defined above, the XSLT namespace also defines several other elements. This section covers only the most important XSLT elements. Although it is possible to answer most (or all) of the questions in the exam with the XSLT elements discussed in this section, you should at least be aware of the other elements defined in the XSLT namespace. These include:

- `xsl:apply-imports`
- `xsl:attribute-set`
- `xsl:copy`
- `xsl:copy-of`
- `xsl:decimal-format`
- `xsl:call-fallback`
- `xsl:call-key`
- `xsl:call-message`
- `xsl:namespace-alias`
- `xsl:number`
- `xsl:preserve-space`
- `xsl:sort`
- `xsl:strip-space`
- `xsl:call-text`
- `xsl:transform`

## XSLT functions

XSLT inherits all the functions provided by XPath. It also defines the following functions:

Function	Description
<code>current()</code>	Returns the current node as a node set.
<code>document()</code>	Allows you to access nodes in other XML documents.
<code>element-available()</code>	Checks to see if the specified element is supported by the XSLT processor and returns true or false.
<code>format-number()</code>	Converts a number into a string.
<code>function-available()</code>	Returns a Boolean value indicating the availability of the specified function in the function library.
<code>generate-id()</code>	Returns a string value that uniquely identifies the node in the argument node set.
<code>key()</code>	Returns a node set based on the value of the key and the specified node set.
<code>system-property()</code>	Returns the value of a system property as an object.
<code>unparsed-entity-uri()</code>	Returns the URI of the unparsed entity as a string.

## XLink

**XML Linking Language (XLink)** allows you to insert elements into XML documents that create and describe links between XML documents. The World Wide Web Consortium (W3C) defines the linking relationship as follows:

An XLink **link** is an explicit relationship between resources or portions of resources. It is made explicit by an XLink **linking element**, which is an XLink-conforming XML element that asserts the existence of a link.

**Inserting XLink:** In order for an element in an XML document to be referenced from elements in other XML documents, it has to have attributes defined from the XLink namespace (<http://www.w3.org/1999/xlink>). These are called **XLink attributes**.

**XLink attributes:** XLink defines 10 attributes in its global namespace that can be specified for elements in XML documents. They are:

Attribute name	Description
type	<p data-bbox="799 243 1382 302">Indicates the XLink element type. It can take any one of the following six values:</p> <ul data-bbox="911 359 1082 638" style="list-style-type: none"> <li data-bbox="911 359 1046 386">• simple</li> <li data-bbox="911 411 1082 438">• extended</li> <li data-bbox="911 464 1062 491">• locator</li> <li data-bbox="911 516 999 543">• arc</li> <li data-bbox="911 569 1082 596">• resource</li> <li data-bbox="911 621 1031 648">• title</li> </ul> <p data-bbox="799 663 1374 842">While the first two are used to denote the types of links offered by XLink (simple and extended), the remaining four are used to define navigation rules and participation in the link, and are specified as attributes of sub-elements of elements that are extended links.</p>
href	<p data-bbox="799 894 1350 953">Defines the destination URI of the XLink; used with <code>simple</code> and <code>locator</code> types.</p>
role	<p data-bbox="799 968 1342 1058">Provides information about the function of the content of the XLink; used with <code>simple</code>, <code>extended</code>, <code>locator</code>, and <code>resource</code> types.</p>
arcrole	<p data-bbox="799 1073 1342 1142">Provides information about the function of the XLink; used with <code>simple</code> and <code>arc</code> types.</p>
title	<p data-bbox="799 1146 1382 1215">Contains a description of the XLink; used with all types except <code>title</code>.</p>
show	<p data-bbox="799 1220 1374 1289">Specifies how to display the destination content. It can take any one of the following five values:</p> <ul data-bbox="911 1304 1382 1837" style="list-style-type: none"> <li data-bbox="911 1304 1358 1394">• <code>new</code> -- Content is not shown in the source document; it is shown separately.</li> <li data-bbox="911 1419 1382 1509">• <code>replace</code> -- Content is shown in the same place, replacing the source content.</li> <li data-bbox="911 1535 1350 1593">• <code>embed</code> -- Content is shown along with the source content.</li> <li data-bbox="911 1619 1366 1730">• <code>other</code> -- The link does not specify the behavior; document provides additional information on how to display content.</li> <li data-bbox="911 1755 1350 1837">• <code>none</code> -- The link does not specify the behavior; document does <i>not</i> provide additional information on</li> </ul>

	<p>how to display content.</p> <p>The <code>show</code> attribute is used with <code>simple</code> and <code>arc</code> types.</p>
<code>actuate</code>	<p>Describes how to trigger the XLink. It can take any one of the following four values:</p> <ul style="list-style-type: none"> <li>• <code>onRequest</code> -- Link is triggered only when the user makes a request.</li> <li>• <code>onLoad</code> -- Link is triggered when the source document is loaded.</li> <li>• <code>other</code> -- Link does not specify the behavior; document provides additional information on how to display content.</li> <li>• <code>none</code> -- Link does not specify the behavior; document does <i>not</i> provide additional information on how to display content.</li> </ul> <p>The <code>actuate</code> attribute is used with <code>simple</code> and <code>arc</code> types.</p>
<code>label</code>	Denotes which endpoint is referenced by the <code>from</code> and <code>to</code> attributes in an <code>arc</code> type definition; used with <code>locator</code> and <code>resource</code> types.
<code>from</code>	Defines the label for the resource that is the starting point for the <code>arc</code> type definition. This label should match the <code>label</code> attribute defined in the <code>locator</code> or <code>resource</code> types. Used with the <code>arc</code> type.
<code>to</code>	Defines the label for the resource that is the end point for the <code>arc</code> type definition. This label should match with the <code>label</code> attribute defined in the <code>locator</code> or <code>resource</code> types. Used with the <code>arc</code> type.

## XLink: Required and optional attributes

The following table summarizes the attributes that are required or optional for the different XLink element types:

	<code>simple</code>	<code>extended</code>	<code>locator</code>	<code>arc</code>	<code>resource</code>	<code>title</code>
<code>type</code>	Required	Required	Required	Required	Required	Required
<code>href</code>	Optional		Required			
<code>role</code>	Optional	Optional	Optional		Optional	
<code>arcrole</code>	Optional			Optional		

title	Optional	Optional	Optional	Optional	Optional
show	Optional			Optional	
actuate	Optional			Optional	
label			Optional		Optional
from				Optional	
to					Optional

## XLink: Simple and extended links

As you can see from the table in [XLink: Required and optional attributes](#), XLink offers two kinds of links -- namely, **simple links** and **extended links**.

**Simple links** are similar to HTML links. A simple link has two participating resources -- the `source` and the `target`. An example of a simple link is:

```
<books xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="simple"
  xlink:href="books.xml"
  xlink:show="replace"
  xlink:actuate="onRequest" />
```

When the user activates the link, the contents of `books.xml` are shown in place of the source document.

However, **extended links** are where the real power of XLink comes in. Unlike simple links, extended links allow multiple resources to be linked with each other. According to the W3C:

An extended link is a link that associates an arbitrary number of resources. The participating resources may be any combination of remote and local.

As mentioned earlier, an element of extended type might contain as sub-elements any of the following four types, in any order:

- `locator`: These elements address the remote resources that participate in the link.
- `arc`: These elements provide the rules that are used to traverse among the participating link resources.
- `title`: These elements provide information about the link.
- `resource`: These elements address the local resources that participate

in the link.

Here's an example of an extended link:

```
<bookInformation
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="extended"
  xlink:role="bookInformation"
  xlink:title="Understanding Java">

  <book
    xlink:type="locator"
    xlink:href="books/understandingJava.xml"
    xlink:label="bookUnderstandingJava"
    xlink:role="http://www.publisher.com/books/java/understandingJava"
    xlink:title="Understanding Java" />

  <author
    xlink:type="locator"
    xlink:href="authors/JohnPeter.xml"
    xlink:label="JohnPeter"
    xlink:role="http://www.publisher.com/books/author"
    xlink:title="John Peter" />

  <priceInformation
    xlink:type="resource"
    xlink:label="understandingJavaPrice"
    xlink:title="Understanding Java Price">
    <price> 50</price>
  </priceInformation>

  <target
    xlink:type="arc"
    xlink:from="bookUnderstandingJava"
    xlink:to="understandingJavaPrice"
    xlink:show="new"
    xlink:actuate="onRequest"
    xlink:title="Understanding Java Price" />

  <target
    xlink:type="arc"
    xlink:from="bookUnderstandingJava"
    xlink:arcrole="http://www.publisher.com/author"
    xlink:to="JohnPeter"
    xlink:show="replace"
    xlink:actuate="onRequest"
    xlink:title="Peter John, author" />

</bookInformation>
```

The above example defines an extended link through `bookInformation`. Unlike an HTML link that can define only one resource, the `bookInformation` link defines three resources: It has two `locator` type elements -- `book` and `author` -- that point to two external resources (`understandingJava.xml` and `JohnPeter.xml`); and a `resource` type element -- `price` -- which points to a local resource (the `bookInformation` element). The navigation between the resources is defined by the two `target` elements, which are `arc` type elements.

When `bookInformation` is displayed using an XLink-aware browser, it includes the links to the resources as defined by the `arc` elements. The browser decides how

the links will be rendered. The links are most likely to be displayed with the value defined for the `xlink:title` attribute in the `target` elements.

## XPointer

**XML Pointer Language (XPointer)** is the basis for specifying the fragment identifier for any **Uniform Resource Identifier (URI)** that locates an XML resource. It is very similar to the anchor tag (`<a>`) that's used in HTML pages, and is used for pointing to specific fragments of XML documents.

In general, an XPointer has the following syntax:

```
document_uri # XPointer_Form
```

Here, `XPointer_Form` is either a shorthand or a scheme-based pointer.

**Shorthand pointer:** In this type, `XPointer_Form` supplies the ID to locate the fragment. This ID can be the value of either an `id` attribute or an `id` element. Also, the ID can be determined by any one of the following:

- Schema-determined ID for attributes
- Schema-determined ID for elements
- DTD-determined ID
- Externally-determined ID

Here's an example of a schema-determined ID for elements:

```
books.xml # WLS-002
```

This matches the second `book` element with the ID `WLS-002` in the following XML document (`books.xml`):

```
<?xml version="1.0" encoding="UTF-8" ?>
<books xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:noNamespaceSchemaLocation="books.xsd">

  <book>
    <id>WLS-001</id>
    <name>Understanding XSLT</name>
  </book>

  <book>
    <id>WLS-002</id>
    <name>Understanding XLink</name>
  </book>
```

```

<book>
  <id>WLS-003</id>
  <name>Understanding XPointer</name>
</book>
</books>

```

And here's the schema definition (books.xsd):

```

<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <xs:element name="books" type="booksType" />

  <xs:complexType name="booksType">
    <xs:sequence>
      <xs:element name="book" type="bookType" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="bookType">
    <xs:all>
      <xs:element name="id" type="xs:id" />
      <xs:element name="name" type="xs:string" />
    </xs:all>
  </xs:complexType>

</xs:schema>

```

**Scheme-based pointer:** In this type, the pointer consists of one or more pointer forms. Each pointer form has a scheme name and contains, within parenthesis, the data that conforms to the specified scheme.

Here's an example of a scheme-based pointer that uses an element scheme to locate fragments by ID:

```
books.xml # element (WLS-002)
```

This will locate the same second element that was selected in the shorthand pointer form.

## Summary

XSLT stylesheets transform XML documents into other formats. This transformation is achieved through XSLT elements that are defined in the XSLT namespace. Apart from the functions inherited from XPath, XSLT also provides functions that can transform XML documents. XLink links XML documents; XLinks can be either simple or extended links. XPointer is the language that locates fragments in XML documents. XPointers come in two types: shorthand and scheme-based. The former

relies on IDs while the latter relies on schemes to locate fragments.

## Example

Consider the following XML document (employees.xml):

```
<employees>
  <employee position="manager">
    <name>
      <first>John</first>
      <last>Dent</last>
    </name>
    <age>34</age>
  </employee>
  <employee position="developer">
    <name>
      <first>Tom</first>
      <last>Hanks</last>
    </name>
    <age>25</age>
  </employee>
  <employee position="developer">
    <name>
      <first>Mathew</first>
      <last>Perry</last>
    </name>
    <age>31</age>
  </employee>
  <employee position="engineer">
    <name>
      <first>Kim</first>
      <last>Gentile</last>
    </name>
    <age>22</age>
  </employee>
  <employee position="engineer">
    <name>
      <first>Don</first>
      <last>Bradman</last>
    </name>
    <age>34</age>
  </employee>
  <employee position="developer">
    <name>
      <first>Melissa</first>
      <last>Anderson</last>
    </name>
    <age>32</age>
  </employee>
</employees>
```

This document is linked to the following XSLT stylesheet (employee.xslt):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
    <html>
      <head>
```

```

    <title>Demonstrates transformation of xml using XSLT </title>
  </head>
  <body>
    <h3>XSLT Transformation of employees.xml using employee.xslt</h3>
    <hr/>

    Employees : <xsl:value-of select="count(//employee)"/>
    <br/>
    <br/>

    : <xsl:value-of select="count(//employee[@position='Manager'])"/>,
    : <xsl:value-of select="count(//employee[@position='Engineer'])"/>,
    : <xsl:value-of select="count(//employee[@position='Developer'])"/>
    <br/>
    <br/>

    <table border="1">
      <tr>
        <th>Employee Name</th>
        <th>Position</th>
        <th>Age</th>
      </tr>
      <xsl:apply-templates select="//employee"/>
    </table>
  </body>
</html>
</xsl:template>

<xsl:template match="employee">
  <tr>
    <td>
      <xsl:value-of select="concat(name/first,' ', name/last)"/>
    </td>
    <td>
      <xsl:value-of select="@position"/>
    </td>
    <td>
      <xsl:value-of select="age"/>
    </td>
  </tr>
</xsl:template>

</xsl:stylesheet>

```

Now try to guess the output of the XML file when the XSLT stylesheet is applied to it, and see if it corresponds to the output shown below.

Here's the output in HTML:

```

<html>
  <head>
    <META http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Demonstrates transformation of xml using XSLT </title>
  </head>
  <body>
    <h3>XSLT Transformation of employees.xml using employee.xslt</h3>
    <hr>

    Total Employees : 6<br><br>

    Managers : 1,
    Engineers : 2,
    Developers : 3<br><br>
    <table border="1">

```

```
<tr>
  <th>Employee Name</th>
  <th>Position</th>
  <th>Age</th>
</tr>

<tr>
  <td>John Dent</td>
  <td>Manager</td>
  <td>34</td>
</tr>

<tr>
  <td>Tom Hanks</td>
  <td>Developer</td>
  <td>25</td>
</tr>

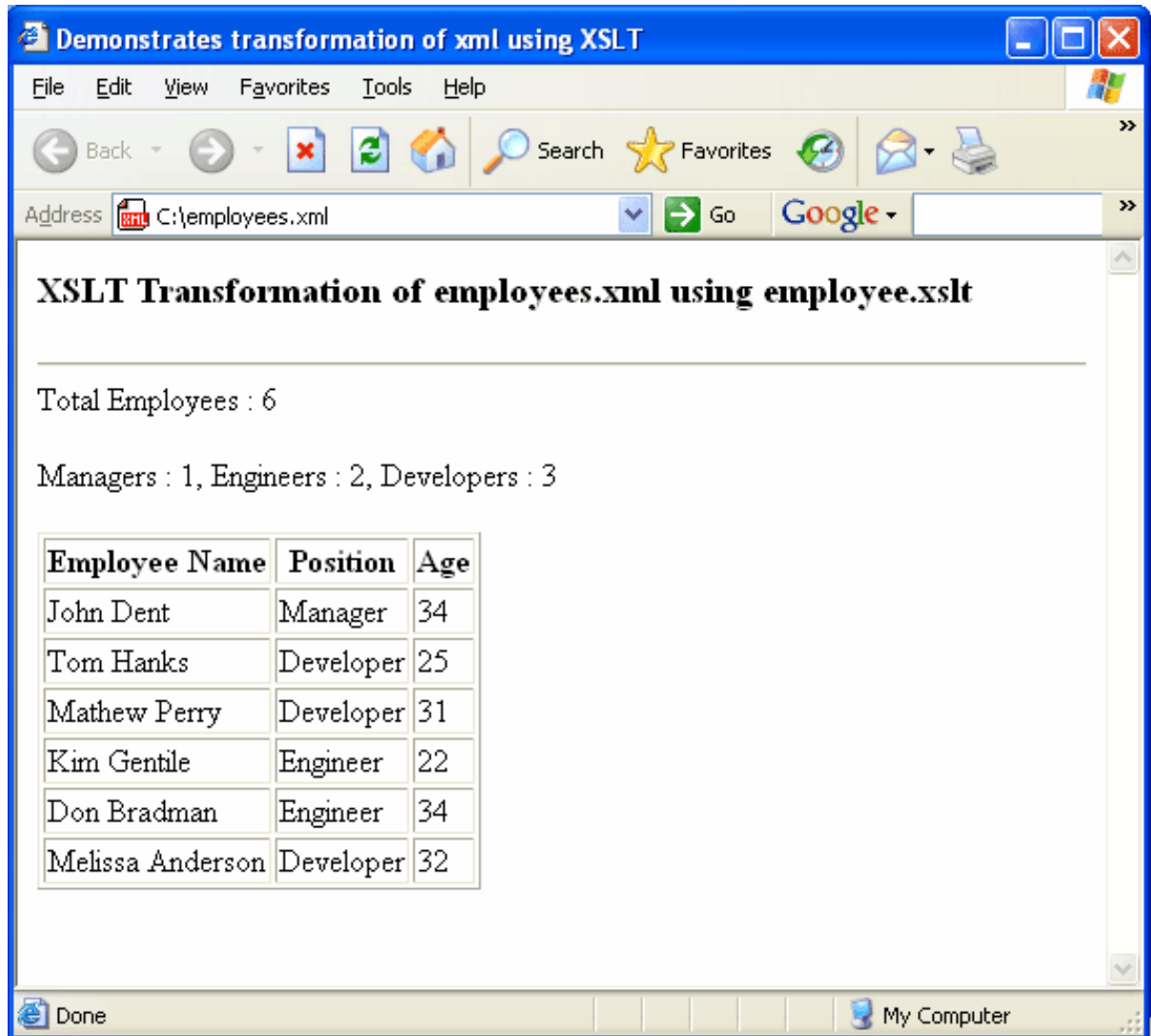
<tr>
  <td>Mathew Perry</td>
  <td>Developer</td>
  <td>31</td>
</tr>

<tr>
  <td>Kim Gentile</td>
  <td>Engineer</td>
  <td>22</td>
</tr>

<tr>
  <td>Don Bradman</td>
  <td>Engineer</td>
  <td>34</td>
</tr>

<tr>
  <td>Melissa Anderson</td>
  <td>Developer</td>
  <td>32</td>
</tr>
</table>
</body>
</html>
```

And here's the output as displayed in Internet Explorer:



## Exercises

1. After applying an XSLT stylesheet to an XML document, the output:
  - A. is always another XML document
  - B. is always an HTML document
  - C. is always a text document
  - D. can be any type of document

**Correct choice: A**

**Explanation:**

When you apply an XSL stylesheet to an XML document, the result is always an XML document. Depending upon the XSL processor that you use, the resulting XML set can then be converted into any format required by the application (such as HTML or PDF). So Choices B, C, and D are incorrect.

2. Which of the following statements with respect to XLinks and HTML links is *false*?
- A. HTML links connect only two resources.
  - B. HTML links allow navigation only in one direction.
  - C. HTML links specify only the behavior of the rendering engine and not the actual data.
  - D. Simple XLinks need not be embedded in the source document.

**Correct Choice: C**

**Explanation:**

HTML links do not specify any additional conceptual behavior for the rendering engine. No information about whether the rendering engine should automatically traverse the link or wait for user interaction can be specified.

Choice A is incorrect since a single HTML link can only connect two resources.

Choice B is incorrect because if you have a sequence of pages and you want the user to be able to navigate between them, you need to explicitly define hyperlinks between all of them.

Choice D is incorrect since it is possible to embed XLinks in a different document than the one it is linking.

## Exam tips

Many questions on the exam require you to choose the output of a given XSLT stylesheet when applied to a given XML document. In some cases, you might have to choose one of several stylesheets that will yield the output specified in the question when applied to a given XML document. In either case, it is very important to understand the XSLT elements and their functionalities; at a minimum, you should

know the important XSLT elements described in this section.

You are likely to get one or two questions on XLink and XPointer, and they require only a basic understanding of these technologies. So try to spend more time learning the XSLT elements and functions.

---

## Section 4. CSS and XSL-FO

### CSS basics

**CSS** stands for **Cascading Style Sheets**. If you've designed Web pages, you have probably used CSS to control the style and layout of your pages.

Put simply, you can use CSS to control the style and layout of elements in an XML document. This is done by defining values for their style properties. Each element in a document (like `h1` in XHTML pages) has its own set of style properties, such as color or font. You can define values for these style properties using CSS, and then apply the CSS to the document.

The exam requires that you know how to apply CSS in formatting XML and XHTML documents. We will start by showing you the basics of CSS and how it applies specifically to XHTML documents, then we will show you how to apply CSS to XML documents in general.

### CSS: Style property definition

To define a value for a style property which belongs to a particular tag in the document, you have to use the following syntax.

```
selector {property: value}
```

For example:

```
body {color: blue}
```

Where:

- `selector` is the element or tag (the `<body>` tag in the example)
- `property` is the style property you want to set (the `color` of the `<body>` tag)
- `value` is the value for the property (blue for `color`)

When you apply this CSS definition to an XHTML document, the contents of the **<body>** tag will display in blue.

It is also possible to define multiple style properties for a single element, or to define style property values for a collection of elements in the same definition. When you specify multiple property-value definitions for a single element, separate them with a semicolon (;).

### Example 1 (multiple properties for a single element):

```
body {color: blue; background-color: #000000 }
```

### Example 2 (same property values for multiple elements):

```
body, h1, p {color: blue}
```

In Example 1, the `color` and `background-color` properties of the `<body>` tag are set to the specified values.

In Example 2, the `color` property of the `<body>`, `<h1>`, and `<p>` tags is set to blue.

## CSS: Class attribute

You can also apply different styles to the same element using the `class` attribute. This allows the same elements to have different presentations, depending on the class they belong to. Consider the following definitions for the `h2` element in the CSS file:

```
h2.error {color:red}
h2.warning {color:orange}
h2.normal {color:black}
```

Where `error`, `warning`, and `normal` denote the three possible class values.

This means an element represented in the document with a `class` attribute value of `error` will display in red. Here's the code:

```
<h2 class = "error">Error! Text here will be displayed in red</h2>
```

And the output is:

## Including CSS

You can include CSS internally, externally, or inline. Internal stylesheets are defined in the document itself using the `<style>` tag. For external stylesheets, the CSS is defined in a separate document and stored with the extension `.css`. It is then included in the Web page using the `<link>` tag. To apply CSS to format XHTML documents in this tutorial, we will mostly use an external stylesheet. In most projects, external stylesheets are the preferred way to include CSS to format XHTML documents.

It is also possible to use an internal stylesheet, but this is not preferred as developers must make a change in the stylesheet for all pages that include the stylesheet internally. With an inline stylesheet, the definition for the style property is made in the definition of the element itself. Inline stylesheets are rarely implemented for the same reason as internal stylesheets.

### Internal stylesheet example:

```
<style type="text/css"> h1{color: red} </style>
```

### External stylesheet example:

```
<link rel="stylesheet" type="text/css" href = "content.css" />
```

### Inline stylesheet example

```
<h3 style="color:green; font-size:14pt">Green, 14-point text</h3>
```

## Style properties

To define CSS for XHTML pages, you need to know the different elements that make up the structure of an XHTML document and their style properties. More than 100 elements are defined for XHTML pages! Even though you are not required to know the style properties for all these elements, you should be familiar with these most frequently-used ones:

- `background-color`
- `font-size`

- background-image
- background-position
- background-repeat
- border
- border-bottom
- border-collapse
- border-color
- border-left
- border-right
- border-spacing
- border-style
- border-top
- bottom
- color
- font
- font-family
- font-style
- font-weight
- left
- list-item
- margin
- padding
- padding-bottom
- padding-left
- padding-right
- padding-top
- position
- right
- text-align
- top
- width
- word-spacing

## CSS and XML

The application of CSS to XML is very similar to how you apply CSS to XHTML documents. Instead of specifying the XHTML element name, you specify the element name in the XML document as the selector value; then you specify the style property and the required value for that property. For example:

```
name {display:list-item; list-style:circle; color:blue; margin-left:1cm;}
```

The above example displays the contents of the <name> tag as list items, with the list items denoted by a circle. The color of the content is blue and it is placed 1cm from the left margin.

As with (X)HTML documents, you can define multiple properties for a single element as shown in the above example. You can also define a common property for more than one element. In Example 1, the elements are separated by a comma (,). If you specify an asterisk (\*) instead of an element name, you can apply the style property to all the elements in the document, as in Example 2.

### Example 1 (same property values for multiple elements):

```
name, price { color:blue; background-color:#000000 }
```

### Example 2 (same property values for all elements in the document):

```
* { color:blue }
```

## XSL-FO basics

**Extensible Stylesheet Language - Formatting Objects (XSL-FO)** is another technology used to format XML documents. Similar to CSS, it has additional features that give you more control over the presentation of a document's contents. XSL-FO is based on XML so it is extensible, which means you don't need a separate parser to extract information from an XSL-FO document.

## XSL-FO structure

An XSL-FO document is basically an XML file that usually has the extension .fo. All elements defined in the document belong to the XSL-FO namespace, <http://www.w3.org/1999/XSL/Format>.

Essentially, every XSL-FO file has the following basic structure:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master>
      <fo:region-body />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence>
    <fo:flow>
      <fo:block />
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

## XSL-FO elements

The following list describes the XSL-FO elements:

- fo:root

This is the root element of an XSL-FO document and contains essentially two child elements: `fo:layout-master-set` and `fo:page-sequence`.

- `fo:layout-master-set`  
This element can contain one or more `fo:simple-page-master` elements that define the actual layout of the page in which the XML data will be rendered.
- `fo:simple-page-master`  
This element contains information about the page layout. It has a mandatory `name` attribute that is used by the `fo:page-sequence` element. The page and margin specifications are specified with the following attributes:
  - **Page:** `page-height`, `page-width`
  - **Margin:** `margin-top`, `margin-bottom`, `margin-left`, `margin-right``fo:simple-page-master` contains the following child elements, which specify the different regions of a page:
  - `fo:region-body`
  - `fo:region-before`
  - `fo:region-after`
  - `fo:region-start`
  - `fo:region-end`
- `fo:region-body`, `fo:region-before`, `fo:region-after`, `fo:region-start`, and `fo:region-end`  
These elements control the actual region where the data is rendered:
  - `fo:region-body` -- the place where data is rendered
  - `fo:region-before` -- header
  - `fo:region-after` -- footer
  - `fo:region-start` -- left sidebar
  - `fo:region-end` -- right sidebar
- `fo:page-sequence`  
The rendering of the data actually starts with this element.

`fo:page-sequence` references one of the `fo:simple-page-master` elements defined in the `fo:layout-master-set` element using its `master-name` attribute.

```
<fo:page-sequence master-name = "Report1" >
```

The contents are displayed using the `fo:static-content` and `fo:flow` elements.

- `fo:static-content`  
This element holds content that is repeated across pages (such as the header and footer).
- `fo:flow`  
This element holds content that is distributed across a sequence of pages. Usually, the contents are distributed across a sequence of child `fo:block` elements.

A `flow-name` attribute specifies the region where the contents of this element are displayed. The default names for the regions are given as:

- `xsl-region-body`
- `xsl-region-start`
- `xsl-region-end`
- `xsl-region-before`
- `xsl-region-after`

In most cases, the specified region is `xsl-region-body`.

In addition to the `fo:block` element, `fo:flow` can also contain `fo:table`, `fo:table-and-caption`, `fo:block-container`, and `fo:list-block` elements.

- This element contains the text to be displayed. The `fo:block` element might contain `fo:inline` elements that are used to change text properties.

For example:

```
<fo:block font-size = "14pt" line-height = "17pt">  
<!-- The text that follows is -->  
<fo:inline font-style = "italic">italic</fo:inline>  
<!-- but the rest of the text is not! -->  
</fo:block>
```

## Applying XSL-FO to XML documents

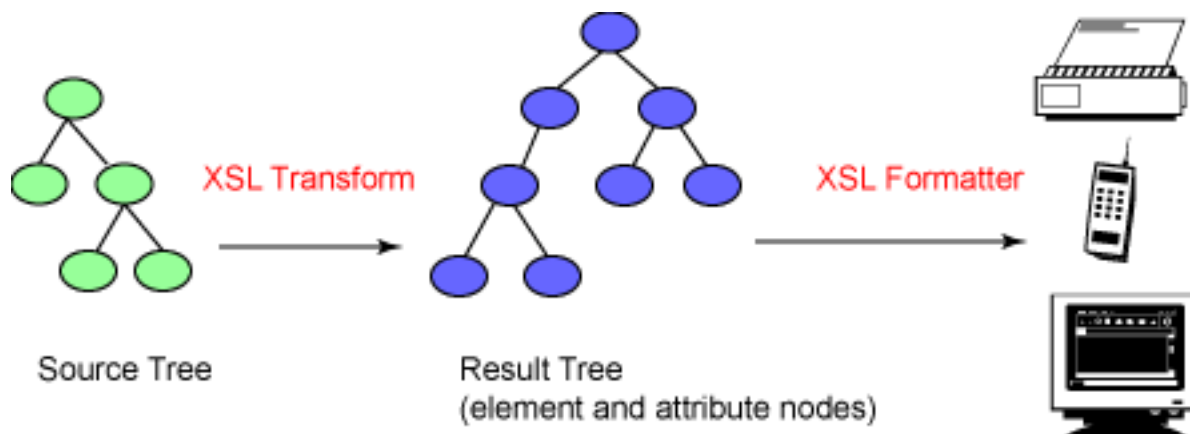
It is possible to use XSL-FO to format XML documents. To format XML documents, you first need to convert the XML file into a file that conforms to the XSL-FO structure -- that is, you need to create a file with the extension .fo.

Since XSL-FO files are XML files, you can use SAX, DOM, or XSLT to process your XML file and produce XSL-FO files. Then, you need a formatting engine to convert the generated XSL-FO files to the required format (HTML, PDF, and other file types).

In essence, the entire process consists of two steps: **transformation** and **formatting**.

The XSL-FO example illustrated later in this section will use XSLT to perform the transformation, and then Apache's open source **Formatting Objects Processor (FOP)** to format the XSL-FO file to PDF format.

The following diagram illustrates the steps involved in the process.



Result XML tree is the result of XSLT processing.

*Copyright 2001 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.*

## Summary

With CSS, you can format HTML and XML documents. These stylesheets are

generally defined in a separate file. Then, you refer to that file from the XML document. The syntax for setting style properties for selectors applies to XML elements as well. It is possible to assign the same property to multiple elements and define multiple properties for the same element.

XSL-FO is used to format XML documents. The various elements defined in the XSL-FO namespace enable you to control how the content in XML documents displays to the end user.

## CSS example

Consider the following books.css file:

```
info{
  display:block;
  color: Green;
  font: Times New Roman;
  font-size: 14pt;
  text-align:center;
  text-decoration: underline;
  padding: 3%;
}
name{
  display: list-item; list-style: circle; color:red; margin-left: 1cm;
}
publisher,price,author{
  display: none;
}
```

When this is applied to the following books.xml document, the result is a list of the book names alone (output appears below the XML listing).

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/css" href="books.css"?>
<books>
  <info>Books available for Java technology</info>
  <book>
    <name>Java Basics</name>
    <author>John Peter</author>
    <price>50</price>
    <publisher>Whizlabs</publisher>
  </book>
  <book>
    <name>Secrets of Java</name>
    <author>Rob Smith</author>
    <price>30</price>
    <publisher>Whizlabs</publisher>
  </book>
  <book>
    <name>J2EE and Web services</name>
    <author>Song Kim</author>
    <price>65</price>
    <publisher>Whizlabs</publisher>
  </book>
</books>
```

The output:

## XSL-FO example

We will use the same books.xml file that we used for the CSS example. First, it's important to transform the XML file to an XSL-FO file. This is done using the following XSLT stylesheet (booksXslFoConverter.xslt):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">

  <!--We specify the output to be an XML file -->
  <xsl:output method="xml" version="1.0" encoding="UTF-8"/>
  <xsl:template match="/">
    <fo:root>
      <!-- This defines the format for the different regions
      in the document -->
      <fo:layout-master-set>
        <fo:simple-page-master page-height="10cm"
          page-width="20cm" margin-top="5mm" left="20mm"
          margin-left="10mm" margin-right="10mm"
          bottom="5mm" master-name="Report">
          <fo:region-before border-style="none" border-width="thin"
            extent="15mm"/>
          <fo:region-body border-style="none" border-width="thin"
            margin-top="20mm"/>
          <fo:region-after border-style="none" border-width="thin"
            extent="15mm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <!-- This defines the actual layout of the page in which the
      content will be rendered -->
      <fo:page-sequence font-family="Times Roman"
        master-reference="Report">
        <!-- The Header Region-->
        <fo:static-content flow-name="xsl-region-before">
          <fo:block font-size="16pt" font-weight="bolder">
            This is the header text displayed using
            "fo:static-content" with flow-name as
            "xsl-region-before"
          </fo:block>
        </fo:static-content>

        <!-- The Footer Region-->
        <fo:static-content flow-name="xsl-region-after">
          <fo:block font-size="16pt" font-weight="bolder">
            This is the footer text displayed using
            "fo:static-content" with flow-name as
            "xsl-region-after"
          </fo:block>
        </fo:static-content>

        <!-- The region where the data is displayed starts here -->
        <fo:flow font-size="16pt" flow-name="xsl-region-body">
          <!-- Displays Information through the fo:block
          elements -->
          <fo:block color="red">The following is an example of
            a table :
          </fo:block>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
</xsl:stylesheet>
```

```

<fo:block font-size="16pt" space-before="2em">
<!-- Use fo:table element to construct tables -->
<fo:table table-layout="fixed" >
  <fo:table-column column-number="1"/>
  <fo:table-column column-number="2"/>
  <fo:table-body>
    <fo:table-row font-weight="bolder"
      text-align="center">
      <fo:table-cell border-style="solid"
        border-width="0.3mm">
        <fo:block>Book</fo:block>
      </fo:table-cell>

      <fo:table-cell border-style="solid"
        border-width="0.3mm">
        <fo:block>Price</fo:block>
      </fo:table-cell>
    </fo:table-row>

    <!-- We use the xsl:apply templates to start
      the processing of the XML document here -->
    <xsl:apply-templates/>

    </fo:table-body>
  </fo:table>
</fo:block>
</fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>

<!-- This template matches the book elements in the XML
document -->
<xsl:template match="book">
  <fo:table-row text-align="left">
    <fo:table-cell border-style="solid" border-width="0.3mm">
      <!-- Get the Book name from the XML document using
        xsl:value-of -->
      <fo:block>
        <xsl:value-of select="./name"/>
      </fo:block>
    </fo:table-cell>

    <fo:table-cell border-style="solid" text-align="center"
      border-width="0.3mm">
      <!-- Get the Price of teh book from the XML document
        using xsl:value-of -->
      <fo:block>
        <xsl:value-of select="./price"/>
      </fo:block>
    </fo:table-cell>
  </fo:table-row>
</xsl:template>
</xsl:stylesheet>

```

When the above XSLT stylesheet is applied to the books.xml document, the following books.fo file will be generated.

```

<?xml version="1.0" encoding="UTF-8"?>
<fo:root>
  <fo:layout-master-set>
    <fo:simple-page-master page-height="10cm"
      page-width="20cm" margin-top="5mm" left="20mm"
      margin-left="10mm" margin-right="10mm"
      bottom="5mm" master-name="Report">

```

```

<fo:region-before border-style="none" border-width="thin"
    extent="15mm" />
<fo:region-body border-style="none" border-width="thin"
    margin-top="20mm" />
<fo:region-after border-style="none" border-width="thin"
    extent="15mm" />
</fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence font-family="Times Roman"
    master-reference="Report">
  <fo:static-content flow-name="xsl-region-before">
    <fo:block font-size="16pt" font-weight="bolder">
      This is the header text displayed using
      "fo:static-content" with flow-name as
      "xsl-region-before"
    </fo:block>
  </fo:static-content>

  <fo:static-content flow-name="xsl-region-after">
    <fo:block font-size="16pt" font-weight="bolder">
      This is the footer text displayed using
      "fo:static-content" with flow-name as
      "xsl-region-after"
    </fo:block>
  </fo:static-content>

  <fo:flow font-size="16pt" flow-name="xsl-region-body">
    <!-- Displays Information through the fo:block
    elements -->
    <fo:block color="red">The following is an example of
      a table :
    </fo:block>
    <fo:block font-size="16pt" space-before="2em">
      <!-- Use fo:table element to construct tables -->
      <fo:table table-layout="fixed" >
        <fo:table-column column-number="1" />
        <fo:table-column column-number="2" />
        <fo:table-body>
          <fo:table-row font-weight="bolder"
            text-align="center">
            <fo:table-cell border-style="solid"
              border-width="0.3mm">
              <fo:block>Book</fo:block>
            </fo:table-cell>

            <fo:table-cell border-style="solid"
              border-width="0.3mm">
              <fo:block>Price</fo:block>
            </fo:table-cell>
          </fo:table-row>

          <fo:table-row text-align="left">
            <fo:table-cell border-style="solid"
              border-width="0.3mm">
              <fo:block>Java Basics</fo:block>
            </fo:table-cell>
            <fo:table-cell border-style="solid"
              text-align="center"
              border-width="0.3mm">
              <fo:block>50</fo:block>
            </fo:table-cell>
          </fo:table-row>

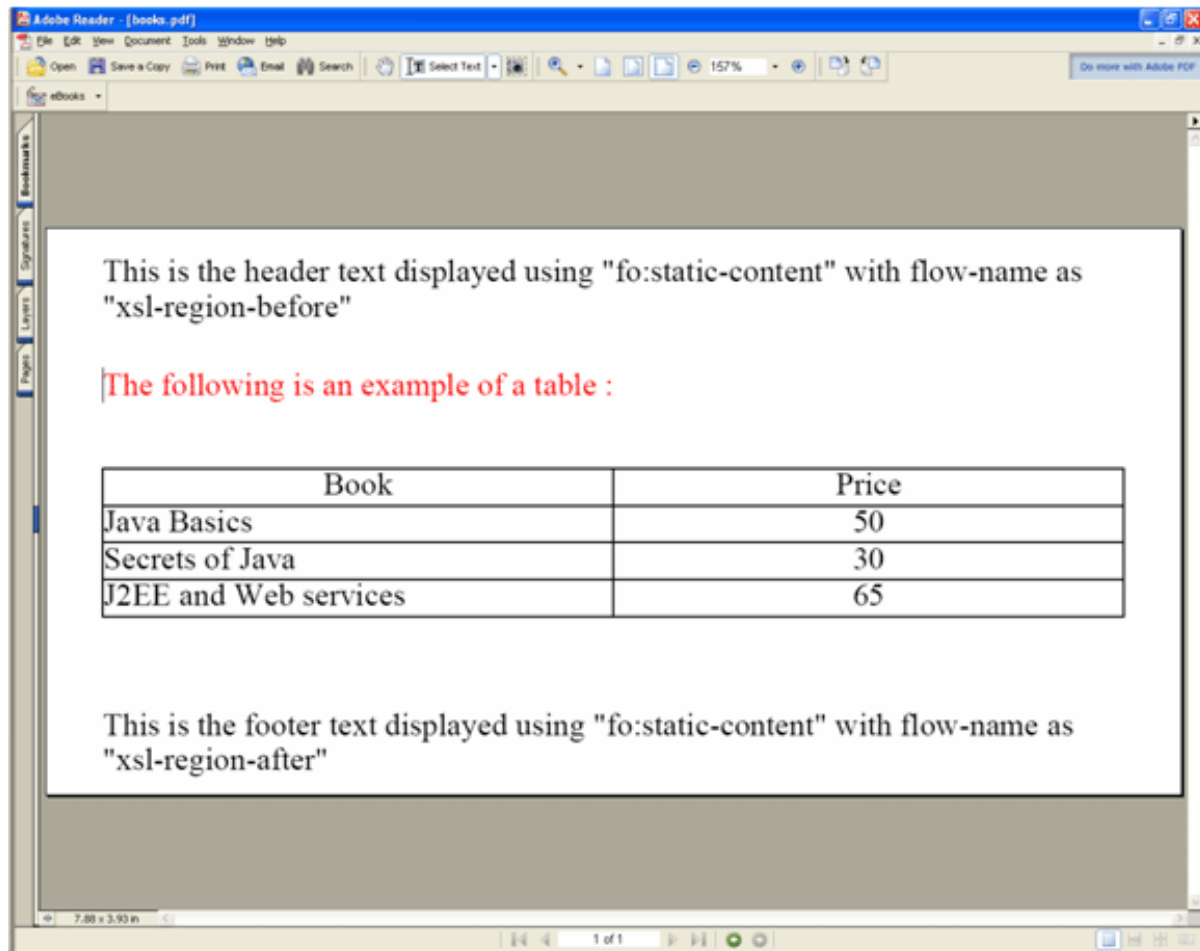
          <fo:table-row text-align="left">
            <fo:table-cell border-style="solid"
              border-width="0.3mm">
              <fo:block>Secrets of Java</fo:block>
            </fo:table-cell>

```

```
        <fo:table-cell border-style="solid"
                      text-align="center"
                      border-width="0.3mm" >
          <fo:block>30</fo:block>
        </fo:table-cell>
      </fo:table-row>

      <fo:table-row text-align="left">
        <fo:table-cell border-style="solid"
                      border-width="0.3mm">
          <fo:block>J2EE and Web
                    services</fo:block>
        </fo:table-cell>
        <fo:table-cell border-style="solid"
                      text-align="center"
                      border-width="0.3mm">
          <fo:block>65</fo:block>
        </fo:table-cell>
      </fo:table-row>
    </fo:table-body>
  </fo:table>
</fo:block>
</fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>
```

This FOP then processes the books.fo file, and the result is the following books.pdf file.



## Exercise

Which of the following is *not* possible?

### Choices

- A. Applying an XSL stylesheet to a Schema document
- B. Applying a Schema document to an XSL stylesheet
- C. Applying a CSS stylesheet to an XML document
- D. Applying an XSL stylesheet to a CSS stylesheet
- E. None of these

**Correct choice: D**

## Explanation:

Schemas and XSL stylesheets are well-formed XML documents, with all the privileges and responsibilities of other XML documents. You can validate them with DTDs or render them with stylesheets just like any other XML document. Hence choices A, B, and C are very much possible. On the other hand, CSS stylesheets are not XML documents, so you cannot apply XSL stylesheets to render them.

## Exam tips

To answer the exam questions on these technologies, it is *very* important that you know the advantages of XSL-FO over CSS. You don't actually have to remember all the style properties and their values. Most of the scenario-based questions require that you choose either one of the technologies to solve a problem.

Remember the following points:

- CSS is not based on XML and hence is not extensible.
- CSS requires a separate parser for parsing its contents.
- CSS has more browser support than XSL-FO.
- CSS is very simple.
- XSL-FO is based on XML, hence it is extensible.
- Using XSL-FO, it is possible to provide features like `right-to-left` and `footnotes`. These features are not available in CSS.

It is also important to know the syntax and functionality of the various elements in XSL-FO.

---

## Section 5. DOM

### DOM basics

The **Document Object Model (DOM)** is an application programming interface (API) that's used by various applications to modify and search for elements or content in an XML document. DOM is not tied to any particular technology (such as the Java language) and can be implemented using any programming language.

The XML exam requires you to know the **DOM 2.0** API, which consists of 14 modules. The specifications that are important for the exam are (in order of importance):

1. Core
2. Events
3. Traversal

The DOM Core is the most important of these specifications and almost all of the DOM-related questions in the XML exam are based on it. Few, if any, questions pertain to the other two specifications.

## Core Specification

The DOM Core Specification (see [Resources](#)) defines the interfaces that must be implemented by applications using DOM to access XML documents. To understand the Core specification, it is important to know the structure of the DOM with regard to XML documents -- we describe that structure in the rest of this section.

## DOM structure: Objects

The elements in an XML document are represented as a hierarchical structure in DOM. These elements are represented as **objects**. The following table describes the different objects that can be present in an XML document.

Object type	Description
Attr	Represents the <code>attribute</code> element in the XML document.
CDATASection	Represents a CDATA section, if one is present in the XML document.
Comment	Represents a comment in the XML document.
Document	Represents the entire XML document.
DocumentFragment	Can be used to represent a portion of the XML document.
DocumentType	Stores information about the XML document that is obtained from a DTD or a Schema. Even though storage of the DTD or Schema information is not required, the specification requires that you store the entities and notations declared in the DTD.

Element	Represents an element present in the XML document.
Entity	Represents the information about an entity in the XML document.
EntityReference	Represents information about a reference to an entity in the XML document.
Notation	Represents information about the notation that's declared in the DTD associated with the XML document.
ProcessingInstruction	Represents a processing instruction in the XML document.
Text	Represents the data present in an attribute or element node.

The DOM 2.0 Core Specification also defines five objects that are not part of the XML document. The following table gives a brief description of these objects.

Object name	Description
DOMImplementation	Allows the application using the DOM to view the features that are supported by the specific DOM being used.
DOMException	Defines errors that occur due to the DOM.
NamedNodeMap	A collection of a particular set of nodes. It is possible to access these nodes by name.
Node	The most important object in DOM. All the other nodes (listed in the above table) are based on this object.
NodeList	Also represents a collection of nodes. Unlike NamedNodeMap, however, nodes in NodeList are accessed by index.

## DOM structure: Attributes and methods

DOM 2.0 defines the **attributes** and **methods** that must be implemented by applications that use DOM. These are defined using the Object Management Group Interface Definition Language (OMGIDL). For the exam, it is important to know the different methods that are defined by DOM 2.0 for these objects, as well as their functionality.

The most important objects are (in order of importance for the exam):

1. Element

2. Node
3. Document
4. NamedNodeMap
5. NodeList
6. DOMException

In the exam, most of the questions center on the functionality of DOM. You need to remember all the methods for these objects. You don't really have to remember all of the attributes other than for the `DOMException` object. Next, we'll show you the Interface Definition Language (IDL) method definitions for the six important objects.

## DOM structure: Element methods

### Element (15 methods)

Function	Description
<code>DOMString getAttribute(in DOMString name)</code>	Returns the value of the specified attribute.
<code>void setAttribute(in DOMString name, in DOMString value)</code>	Sets the value of the given attribute with the given value.
<code>void removeAttribute(in DOMString name)</code>	Removes the attribute with the given name.
<code>Attr getAttributeNode(in DOMString name)</code>	Returns the <code>Attr</code> node for the attribute with the given name.
<code>Attr setAttributeNode(in Attr newAttr)</code>	Adds the given <code>Attr</code> node to the element.
<code>Attr removeAttributeNode(in Attr oldAttr)</code>	Removes the specified <code>Attr</code> node.
<code>NodeList getElementsByTagName(in DOMString name)</code>	Returns a <code>NodeList</code> that contains the elements with the specified name.
<code>DOMString getAttributeNS(in DOMString namespaceURI, in DOMString localName)</code>	Returns the value of the attribute specified in the given namespace.
<code>void setAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName, in DOMString value)</code>	Sets the value of the given attribute with the given value and the given namespace.
<code>void removeAttributeNS(in DOMString namespaceURI, in DOMString localName)</code>	Removes the attribute with the given name in the specified namespace.
<code>Attr getAttributeNodeNS(in DOMString</code>	Returns the <code>Attr</code> node for the attribute with the

<code>namespaceURI, in DOMString localName)</code>	given name in the specified namespace.
<code>Attr setAttributeNodeNS(in Attr newAttr)</code>	Adds the given <code>Attr</code> node to the element.
<code>NodeList getElementsByTagNameNS(in DOMString namespaceURI, in DOMString localName)</code>	Returns a <code>NodeList</code> that contains the elements with the name specified in the given namespace.
<code>boolean hasAttribute(in DOMString name)</code>	Returns true or false, indicating the presence or absence of the given attribute in the element.
<code>boolean hasAttributeNS(in DOMString namespaceURI, in DOMString localName)</code>	Returns true or false, indicating the presence or absence of the given attribute in the element in the given namespace.

## DOM structure: Node methods

### Node (9 methods)

Function	Description
<code>Node insertBefore(in Node newChild, in Node refChild)</code>	Inserts the <code>newChild</code> node before the <code>refChild</code> node.
<code>Node replaceChild(in Node newChild, in Node oldChild)</code>	Replaces the <code>oldChild</code> with the <code>newChild</code> node.
<code>Node removeChild(in Node oldChild)</code>	Removes the specified <code>oldChild</code> node.
<code>Node appendChild(in Node newChild)</code>	Appends the <code>newChild</code> node to the node.
<code>boolean hasChildNodes()</code>	Checks to see if the current node has child nodes.
<code>Node cloneNode(in boolean deep)</code>	Clones the given node. If <code>deep</code> is true, it also clones the child nodes associated with the node.
<code>void normalize()</code>	Combines all the text nodes beneath the node.
<code>boolean isSupported(in DOMString feature, in DOMString version)</code>	Checks to see if the given feature is supported by the node.
<code>boolean hasAttributes</code>	Checks to see if the node has attributes.

## DOM structure: Document methods

### Document (9 methods)

Name	Description
<code>ProcessingInstruction createProcessingInstruction(in DOMString target, in DOMString data)</code>	Creates a processing instruction with the given target and data values.

<code>Attr createAttribute(in DOMString name)</code>	Creates an attribute with the given name.
<code>EntityReference createEntityReference(in DOMString name)</code>	Creates an entity reference with the given name.
<code>NodeList getElementsByTagName(in DOMString tagName)</code>	Returns a node list that contains the elements with the given element name.
<code>Node importNode(in Node importedNode, in boolean deep)</code>	Allows you to import a node from another document.
<code>Element createElementNS(in DOMString namespaceURI, in DOMString qualifiedName)</code>	Creates an element with the given namespace and name.
<code>Attr createAttributeNS(in DOMString namespaceURI, in DOMString qualifiedName)</code>	Creates an attribute with the given namespace and name.
<code>NodeList getElementsByTagNameNS(in DOMString namespaceURI, in DOMString localName)</code>	Returns a node list that contains the elements with the given element name in the given namespace.
<code>Element getElementById(in DOMString elementId)</code>	Returns the element with the specified elementId.

## DOM structure: NamedNodeMap methods

### NamedNodeMap (7 methods)

Name	Description
<code>Node getNamedItem(in DOMString name)</code>	Returns the node with the given name from the NamedNodeMap.
<code>Node setNamedItem(in Node arg)</code>	Adds the node to the NamedNodeMap.
<code>Node removeNamedItem(in DOMString name)</code>	Removes the node with the given name from the NamedNodeMap.
<code>Node item(in unsigned long index)</code>	Returns the node in the given index in the NamedNodeMap.
<code>Node getNamedItemNS(in DOMString namespaceURI, in DOMString localName)</code>	Returns the node with the given name from the NamedNodeMap in the given namespace.
<code>Node setNamedItemNS(in Node arg)</code>	Adds the node to the NamedNodeMap.
<code>Node removeNamedItemNS(in DOMString namespaceURI, in DOMString localName)</code>	Removes the node with the given name from the NamedNodeMap in the given namespace.

## DOM structure: NodeList method

### NodeList (1 method)

Name	Description
Node item(in unsigned long index)	Returns the node in the given index in the NodeList.

## DOM structure: DOMException

### DOMException (15 exception codes)

The `DOMException` object has only one attribute, `code`. The exception code can be any one of the following 15 codes. (All codes are of type `const unsigned short`.)

Code name	Description
INDEX_SIZE_ERR	Indicates when the index specified is negative or greater than the allowed value.
DOMSTRING_SIZE_ERR	Indicates if the size of the specified string is too large to fit into a <code>DOMString</code> .
HIERARCHY_REQUEST_ERR	Indicates when a node is incorrectly inserted into the hierarchy.
WRONG_DOCUMENT_ERR	Indicates if a node is inserted into the wrong document.
INVALID_CHARACTER_ERR	Indicates when an invalid character is specified as a value.
NO_DATA_ALLOWED_ERR	Indicates when an attempt is made to set the value of a node that does not have any value.
NO_MODIFICATION_ALLOWED_ERR	Indicates when an attempt is made to modify a read-only node.
NOT_FOUND_ERR	Indicates when an attempt is made to access a node that does not exist in the document.
NOT_SUPPORTED_ERR	Indicates when the implementation does not support the requested object type.
INUSE_ATTRIBUTE_ERR	Indicates when an attempt is made to add an <code>Attr</code> that is being used somewhere else.
INVALID_STATE_ERR	Indicates when the requested operation could not be performed.
SYNTAX_ERR	Indicates when an attempt was made to set an illegal value to a string.

INVALID_MODIFICATION_ERR	Indicates when an operation was not able to complete in the specified way.
NAMESPACE_ERR	Indicates when the namespace prefix or URI is incorrect.
INVALID_ACCESS_ERR	Indicates when the requested operation is not supported by the implementation.

## Events Specification

The **DOM Events Specification** (see [Resources](#)) dictates what applications have to implement for event registration and event handling in DOM structures that are created from XML documents. Applications that support the Events Specification have to implement the following interfaces:

- `EventTarget`
- `EventListener`
- `Event`
- `DocumentEvent`

### EventTarget

The `EventTarget` interface allows applications to register for events by allowing registration and removal of `EventListeners`. `EventTarget` has the following three method definitions that allow an application to add and remove `EventListeners`, and dispatch events when they are raised:

Method	Description
<code>void addEventListener (in DOMString type, in EventListener listener, in boolean useCapture)</code>	Adds the given <code>EventListener</code> to the <code>EventTarget</code> .
<code>void removeEventListener (in DOMString type, in EventListener listener, in boolean useCapture)</code>	Removes the given <code>EventListener</code> from the <code>EventTarget</code> .
<code>boolean dispatchEvent (in Event evt)</code>	Dispatches the given event programmatically.

### EventListener

Applications that handle events implement this interface. The interface defines only one method for handling events:

Method	Description
<code>void handleEvent (in Event evt)</code>	Handles the given event.

**Event**

This interface provides information about the event to the event handler. It defines the following three methods for providing the event information:

Method	Description
<code>void stopPropagation()</code>	Stops the propagation of the event through the document tree.
<code>void preventDefault()</code>	Cancel the default event specific to the implementation.
<code>void initEvent (in DOMString eventTypeArg, in boolean canBubbleArg, in boolean cancelableArg)</code>	Initializes events that are created programmatically.

**DocumentEvent**

This interface allows the application to create an event that is defined in the specification. It has the following method that is used to create events:

Method	Description
<code>Event createEvent (in DOMString eventType)</code>	Creates the given event type.

**Event types:** The Events Specification specifies these event types that are applicable to XML documents.

- User interface (UI) events
- Mouse events
- Key events
- Mutation events

## Traversal Specification

The **DOM Traversal Specification** (see [Resources](#)) defines interfaces that provide fast, easy retrieval of content from XML documents. The important interfaces defined in this specification are:

- `NodeIterator`
- `TreeWalker`
- `NodeFilter`

- DocumentTraversal

### NodeIterator

When you use the `NodeIterator` interface to traverse a list of nodes in a document, the nodes are returned sequentially -- that is, in document order. To create a `NodeIterator`, call the `createNodeIterator()` method that's defined in the `DocumentTraversal` interface. The following methods to traverse the tree are defined in the `NodeIterator` interface:

Method	Description
<code>Node nextNode()</code>	Returns the node after the current node in the <code>NodeIterator</code> .
<code>Node previousNode()</code>	Returns the node before the current node in the <code>NodeIterator</code> .
<code>void detach()</code>	Releases the <code>NodeIterator</code> .

### TreeWalker

Unlike the `NodeIterator`, when you use the `TreeWalker` to traverse a list of nodes in a document, the hierarchical relationship between the nodes is maintained. Also, the `TreeWalker` defines methods that allow you to navigate to parent, sibling, and child nodes. To create a `TreeWalker`, call the `createTreeWalker()` method defined in the `DocumentTraversal` interface. The important methods defined in the `TreeWalker` interface are:

Method	Description
<code>Node parentNode()</code>	Returns the parent node of the current node in <code>TreeWalker</code> .
<code>Node firstChild()</code>	Returns the first child node of the current node in <code>TreeWalker</code> .
<code>Node lastChild()</code>	Returns the last child node of the current node in <code>TreeWalker</code> .
<code>Node previousSibling()</code>	Returns the closest previous sibling node of the current node in <code>TreeWalker</code> .
<code>Node nextSibling()</code>	Returns the closest next sibling node of the current node in <code>TreeWalker</code> .
<code>Node previousNode()</code>	Returns the node before the current node in the <code>TreeWalker</code> .
<code>Node nextNode()</code>	Returns the node after the current node in the <code>TreeWalker</code> .

### NodeFilter

Using `NodeFilter`, you can create structures that filter out particular nodes. Both `NodeIterator` and `TreeWalker` can use `NodeFilter`. For an application to use

the `NodeFilter` interface, it just needs to implement the following method. This method returns a short value that determines the appearance of the node in the result.

Method	Description
<code>short acceptNode(in Node n)</code>	<p>Indicates whether the specified node must be included in the result set by returning any one of the following three values.</p> <ul style="list-style-type: none"> <li>• <code>FILTER_ACCEPT</code> -- Node appears in the result.</li> <li>• <code>FILTER_REJECT</code> -- Node and child nodes do not appear in the result.</li> <li>• <code>FILTER_SKIP</code> -- Node does not appear in the result, but child nodes may appear.</li> </ul>

### DocumentTraversal

The `DocumentTraversal` interface contains the methods that create `NodeIterator` and `TreeWalker` interfaces. It has the following two methods.

Method	Description
<code>NodeIterator createNodeIterator(in Node root, in unsigned long whatToShow, in NodeFilter filter, in boolean entityReferenceExpansion)</code>	Creates a <code>NodeIterator</code> .
<code>TreeWalker createTreeWalker(in Node root, in unsigned long whatToShow, in NodeFilter filter, in boolean entityReferenceExpansion)</code>	Creates a <code>TreeWalker</code> .

## Summary

Applications use the Document Object Model (DOM) to access and modify XML documents. Applications that use DOM must implement the definitions of DOM 2.0, as specified by the W3C. The Core Specification contains the definitions that determine the structure of DOM and form the basis for the other specifications. The Events Specification defines events associated with a tree structure, the registration of events, and the flow of events through the structure. The Traversal specifications provide interfaces for easy traversal of the XML documents. These include `TreeWalker`, `NodeIterator`, `NodeFilter`, and `DocumentTraversal`.

## Example

Consider the following Books.xml document:

```
<Books>
  <!-- Books Information -->
  <Book id="st_001">
    <Name>Understanding XML</Name>
    <Price>50</Price>
    <![CDATA[ Some data containing a lot of & and < signs !! ]]>
  </Book>
</Books>
```

The Equivalent DOM structure for this document is represented below.

## Exercises

1. Consider the following XML document (books.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book>XML Explained</book>
  <book>Java Explained</book>
  <book>SQL Explained</book>
</books>
```

An application implementing the DOM 2.0 API stores all of the `book` elements in a `NamedNodeMap` object. Which of the following method calls in `NamedNodeMap` will get the third `book` element?

- A. `getItem ("SQL Explained")`
- B. `getNamedItem ("SQL Explained")`
- C. `getItem (3)`
- D. `getItem (2)`
- E. A and C
- F. B and D
- G. B and C

**Correct choice: F**

### Explanation:

The `NamedNodeMap` interface has both `getNamedItem` (in `DOMString`

`name`) and `getItem(in unsigned long index)` methods that return the node based on the node name or on the index.

Choice A is incorrect as `getItem()` takes a long index as its argument, and not a string.

Choice C is incorrect as the index is zero-based; in order to get the third element, the index would be 2 not 3.

Choice G is incorrect as it includes choice C which is incorrect.

2. Which of the following methods defined in the `Document` interface allows you to create an attribute?

- A. `Attribute createAttribute (String name)`
- B. `Attribute createAttribute (DOMString name)`
- C. `Attr createAttribute (String name)`
- D. `Attr createAttribute (DOMString name)`
- E. None of the above

**Correct choice: D**

### Explanation:

The correct method definition for creating an attribute is `Attr createAttribute (DOMString name)`.

The `createAttribute()` method in the `Document` interface returns an `Attr` object and not an `Attribute` object. So choices A and B are incorrect.

Also, the `createAttribute()` method takes in a `DOMString` as the argument type for the attribute name. The DOM specification describes how strings are to be manipulated using DOM by defining the `DOMString` type, which is encoded using the UTF-16 data encoding scheme. So choice C is incorrect.

## Exam tips

To answer questions based on DOM, it is crucial that you remember the IDL

definitions. The `Element` object in the Core Specification is the most important one and you should pay special attention to its IDL.

Also, you'll also want to know the advantages and disadvantages of DOM. Remember the following points:

- DOM is a good choice when you need to modify XML documents in memory.
  - DOM is efficient when you want to do a non-linear search in the XML document.
  - DOM loads the entire document into memory and is a poor choice when memory is a constraint. In such cases, you should use the **Simple API for XML (SAX)**, which is explained in [SAX](#).
- 

## Section 6. SAX

### SAX basics

**Simple API for XML (SAX)** is an application programming interface (API) that parsers implement to process XML documents. While SAX was initially developed for implementation with the Java language, most other programming languages today offer support for SAX.

SAX is highly efficient when compared with DOM. SAX is very fast and requires much less memory; it is the best option when memory is a constraint and the document to be processed is large. Unlike DOM, which is a tree-based API, SAX is event-based -- meaning parsing events are reported directly to the application by the parser. The application implements certain handlers for these events.

The SAX 2.0 specification defines four **core handler interfaces** that applications can implement for handling events. They are:

- `ContentHandler`
- `ErrorHandler`
- `DTDHandler`
- `EntityResolver`

The SAX 2.0 API defines many other interfaces, but as far as the XML certification exam is concerned, you only need to know the interfaces that are defined for applications. This section covers only those interfaces.

## ContentHandler

`ContentHandler` is the most important interface that's implemented by the application in order to process XML documents -- in fact, it's usually the *only* interface that's implemented by the application. The following table lists the methods defined in `ContentHandler`.

Method	Description
<code>setDocumentLocator(Locator locator)</code>	This method is called first. It provides a <code>Locator</code> object, which helps you get information about the location of the event within the document. In most implementations, this object is stored in an attribute of the implementing class.
<code>startDocument()</code>	Marks the beginning of parsing. It is called only once.
<code>endDocument()</code>	Marks the end of parsing. It is called only once, and is the last method that's called by the parser.
<code>processingInstruction(String target, String instruction)</code>	Called by the parser whenever it encounters a processing instruction. It provides the <code>target</code> and the <code>instruction</code> to the method.
<code>startPrefixMapping(String prefix, String uri)</code>	Called by the parser whenever it encounters a namespace declaration for an element. The namespace prefix and the URI are passed to the method. This method is called immediately before the <code>startElement()</code> method for the element that has the declaration.
<code>endPrefixMapping(string prefix)</code>	Called immediately after the <code>endElement()</code> call to an element that has a namespace declaration associated with it.
<code>startElement(String namespaceURI, String localName, String qualifiedName, Attributes attributes)</code>	Called whenever the parser encounters the opening tag of an element. It passes in the namespace URI, the local name, the qualified name (full name), and the attributes associated with the element.
<code>endElement(String namespaceURI, String localName, String qualifiedName)</code>	Called whenever the parser encounters the closing tag of an element. It passes in the namespace URI, the local name, and the qualified name (full name).
<code>Characters(char[] text, int start, int end)</code>	Called by the parser whenever text content is between element tags. It passes in the characters as a character array that specifies the

	start and end positions.
<code>ignorableWhiteSpace(char[] chars, int start, int end)</code>	Called by the parser whenever it encounters white space that the parser can ignore. It passes in the content as a character array.
<code>skippedEntity(String name)</code>	Called by the parser when it is not a validating parser and it encounters an entity in the XML document it is parsing.

## ErrorHandler

The `ErrorHandler` interface provides methods for handling errors that occur during the parsing of an XML document. It defines the following three methods for handling errors. All of these methods take in a parameter of type `SAXParseException`.

Method	Description
<code>warning(SAXParseException)</code>	Indicates that the parser encountered a problem that is not considered an error or a fatal error, as per the XML 1.0 Recommendation. The parser continues processing the XML document after reporting this event.
<code>error(SAXParseException)</code>	Indicates that the parser encountered a problem that is considered an error, as per the XML 1.0 Recommendation. The parser continues processing the XML document after reporting this event.
<code>fatalError(SAXParseException)</code>	Indicates that the parser encountered a problem that is considered a fatal error, as per the XML 1.0 Recommendation. The parser stops processing the XML document if it encounters a fatal error.

## DTDHandler and EntityResolver

The `DTDHandler` interface processes unparsed entities and notation declarations, while the `EntityResolver` interface handles the parser requests for external parsed entities.

Validating parsers that use SAX can implement `DTDHandler`, which has the following two methods:

Method	Description
<code>notationDecl(String name, String publicID, String systemID)</code>	Indicates that the parser encountered a notation declaration. It passes in the name, the

	, and the <code>systemID</code> .
<code>unparsedEntityDecl(String name, String publicID, String systemID, String notationName)</code>	Indicates that the parser encountered an unparsed entity declaration. It passes in the <code>name</code> , the <code>publicID</code> , the <code>systemID</code> , and the <code>notationName</code> that is used to handle the entity.

`EntityResolver` has the following method that must be implemented by the application to resolve entities:

Method	Description
<code>resolveEntity(String publicID, String systemID)</code>	Called whenever the parser encounters an entity. The <code>publicID</code> and <code>systemID</code> defined in the entity are given as parameters.

## Summary

SAX is an event-based API that is used by applications to process XML documents. Applications that use SAX can implement four optional interfaces:

- `ContentHandler` contains most of the methods required for processing XML documents.
- `ErrorHandler` defines three methods for handling errors: `warning()`, `error()`, and `fatalError()`.
- `DTDHandler` processes DTDs.
- `EntityResolver` processes entities.

## Example

Consider the following `books.xml` document:

```
<?processingInstruction name="PI"?>
<books>
  <pro:book xmlns:pro="http://www.whizlabs.com">
    <name>Understanding XML</name>
    <Author>Whizlabs</Author>
    <price>50</price>
  </pro:book>
</books>
```

When a parser using SAX processes this document, and the application that uses the SAX parser has implemented the `ContentHandler` interface, then the methods defined in `ContentHandler` are called by the parser (with the arguments as

shown) in the sequence shown below.

### Notes:

- Text in **bold** represents methods.
- Text in *italics* represents method arguments.
- Normal text represents argument values.
- The *qualifiedName* argument value is printed in quotes as some names contain a colon (:) character.

1. **setDocumentLocator**
2. **startDocument**
3. **processingInstruction** -- *target: xml-stylesheet, data: type = "text/css" href="books.css"*
4. **startElement** -- *namespaceURI: , localName: books, qualifiedName: "books"*
5. **characters** -- *text:*
6. **startPrefixMapping** -- *prefix: pro, uri: http://www.whizlabs.com, startElement namespaceURI : http://www.whizlabs.com, localName : book, qualifiedName : "pro:book"*
7. **characters** -- *text:*
8. **startElement** -- *namespaceURI: , localName: name, qualifiedName: "name"*
9. **characters:** -- *text: Understanding XML*
10. **endElement** -- *namespaceURI: , localName: name, qualifiedName: "name"*
11. **characters** -- *text:*
12. **characters** -- *text:*
13. **startElement** -- *namespaceURI: , localName: Author, qualifiedName: "Author"*

14. **characters** -- *text*: Whizlabs
15. **endElement** -- *namespaceURI*: , *localName*: , *Author*: ,  
*qualifiedName*: "Author"
16. **characters** -- *text*:
17. **characters** -- *text*:
18. **startElement** -- *namespaceURI*: , *localName*: price,  
*qualifiedName*: "price"
19. **characters** -- *text*: 50
20. **endElement** -- *namespaceURI*: , *localName*: price,  
*qualifiedName*: "price"
21. **characters** -- *text*:
22. **characters** -- *text*:
23. **endElement** -- *namespaceURI*: http://www.whizlabs.com,  
*localName*: book, *qualifiedName*: "pro:book"
24. **endPrefixMapping** -- *Prefix*: pro
25. **characters** -- *text*:
26. **endElement** -- *namespaceURI*: , *localName*: books,  
*qualifiedName* : "books"
27. **endDocument**

## Exercises

1. What is the exception type that is passed on to the `warning()`, `error()`, and `fatalError()` methods defined in the `ErrorHandler` interface?
  - A. `SAXException`
  - B. `Exception`
  - C. `SAXParserException`

D. `SAXParseException`

E. `ParseException`

**Correct choice: D**

**Explanation:**

The exception passed to the `warning()`, `error()`, and `fatalError()` methods is `SaxParseException`. The SAX API includes no definitions for `SAXException`, `Exception`, `SAXParserException`, or `ParseException`, so choices A, B, C, and E are incorrect.

2. Which of the following is the correct definition for the `endPrefixMapping()` method defined in the `ContentHandler` interface?

A. `endPrefixMapping()`

B. `endPrefixMapping(String uri)`

C. `endPrefixMapping(String prefix, String uri)`

D. `endPrefixMapping(String prefix)`

E. A and D

F. A and B

**Correct choice: D**

**Explanation:**

The correct definition is `endPrefixMapping (String prefix)`.

Unlike `startPrefixMapping()`, `endPrefixMapping()` does not include the URI as its argument, as this method is called after `endElement()`. The namespace mapping goes out of scope when `endPrefixMapping()` is called. No URI associates after this call, so choices B and C are incorrect. The `endPrefixMapping()` method has a single argument -- the prefix, so choices A, E, and F are incorrect.

## Exam tips

As you have seen in this section, SAX is simpler and easier to implement than DOM. It is important to remember the event methods defined in the `ContentHandler` interface, and even more important to know the differences between SAX and DOM. Most of the scenario-based questions require you to choose either SAX or DOM as a solution for processing XML documents. Remember the following points:

- If the XML document is very large and memory is a constraint, use SAX.
- If you just want to do a *linear* search and modify the document frequently, use SAX.
- If you want to do a *random* search on the document, use DOM.
- If you want to make frequent modifications to the XML document, use DOM.

---

## Section 7. Wrap up

### Summary

Here in Part 2 of this tutorial series on preparing for the IBM XML certification exam, we have covered the following technologies:

- XPath
- XSLT
- XLink
- XPointer
- CSS
- XSL-FO
- DOM
- SAX

XPath and XSLT go hand-in-hand. Most of the exam questions related to these technologies will ask you to evaluate the output of specific XSLT stylesheets, or test you on the functions available in XPath and XSLT. Few if any questions will focus on XLink, XPointer, CSS, or XSL-FO, and those that do will be very basic. Questions relating to DOM and SAX should be relatively easy compared with questions on the

other technologies. A basic understanding of these APIs should be sufficient to help you answer the exam questions.

As stated in [Part 1](#), it is important to understand how all these technologies work apart from memorizing their syntax. In order to answer scenario-based questions, you should know the advantages and disadvantages of these various technologies. In the exam, you have to choose one of several possible solutions for a given question, so it's important to know how the technologies compare to one another. For example, SAX relative to DOM and CSS relative to XSLT.

Now that we've covered the various technologies on the exam, the third part of this series will show you how these technologies map to the exam objectives.

## Resources

- Read [Part 1](#) (March 2005) and [Part 3](#) (August 2005) of this three-part developerWorks tutorial series on preparing for the IBM XML certification exam.
- Find out more about the [IBM Certified Developer in XML and Related Technologies exam](#). This page includes:
  - Details on the job role and target audience for whom the certification was built
  - Recommended prerequisites for the knowledge and skills that one should possess before considering this certification
  - The test objectives and the skills measured by the exam
  - Recommended educational resources to prepare you for the test, based on the test objectives
  - A pre-assessment/sample test to gauge your readiness for the actual exam
- Check out this [introductory article on XML certification](#) by Pradeep Chopra, co-author of this tutorial (developerWorks, March 2003).
- Practice and assess your knowledge level using the [Whizlabs XML Exam Simulator](#).
- Refer to the relevant specifications for more information about the various technologies covered in this tutorial:
  - [XML Path Language \(XPath\), Version 1.0](#)
  - [Extensible Stylesheet Language Family \(XSL\)](#), including XSL-FO
  - [XML Linking Language \(XLink\), Version 1.0](#)
  - XPointer -- [Framework](#), [element \( \) Scheme](#), and [xmlns \( \) Scheme](#)
  - [Cascading Style Sheets \(CSS\), Level 2](#)
  - [Document Object Model \(DOM\)](#)
  - [Simple API for XML \(SAX\)](#)
- Confused by all the XML standards out there? Uche Ogbuji's developerWorks article series on XML standards can help you sort through it all:
  - [Part 1](#) -- The core standards (January 2004)
  - [Part 2](#) -- XML processing standards (February 2004)

- [Part 3](#) -- The most important vocabularies (February 2004)
- [Part 4](#) -- Detailed cross-reference of the most important XML standards (March 2004)
- Visit [the Worldwide Web Consortium \(W3C\) site](#), which contains the specifications for XML and many of its related technologies. This is probably the best place to get in-depth information on XML.
- Check out the [developerWorks XML zone](#) for a wide range of articles, tutorials, columns, forums, and other resources that cover extensible technologies in detail. While you're there, take a look at the developerWorks [SOA and Web services zone](#) for more on those topics.
- Start with the basics. Read Doug Tidwell's popular "[Introduction to XML](#)" tutorial here on developerWorks (August 2002).
- Read Bertrand Portier's developerWorks tutorial "[Get started with XPath](#)" (May 2004).
- Read Michael Kay's updated developerWorks article "[What kind of language is XSLT?](#)" (April 2005).
- Two developerWorks tutorials take an introductory look at XSLT, Nicholas Chase's "[Create multi-purpose Web content with XSLT](#)" (March 2003) and Leigh Dodds' "[Code generation using XSLT](#)" (April 2003).
- Check out Brett McLaughlin's developerWorks tips "[How to use XLink with XML](#)" (February 2002) and "[Using XML and XPointer](#)" (March 2002).
- For more on CSS, read Uche Ogbuji's developerWorks tutorials:
  - "[Use Cascading Stylesheets to display XML](#)" (November 2004)
  - "[Use Cascading Stylesheets to display XML, Part 2](#)" (February 2005)
  - "[Use Cascading Stylesheets to display XML, Part 3](#)" (June 2005)
- Let IBM's Doug Tidwell show you the ropes of XSL-FO:
  - [XSL-FO basics tutorial](#) (February 2003)
  - [XSL-FO advanced techniques tutorial](#) (February 2003)
  - [HTML-to-FO conversion guide](#) (February 2003)
- Learn more about DOM and SAX with Nicholas Chase's popular tutorials "[Understanding DOM](#)" (July 2003) and "[Understanding SAX](#)" (July 2003).
- Visit [www.w3schools.org](http://www.w3schools.org) for good tutorials about various XML technologies.

- Try [www.zvon.org](http://www.zvon.org), another good site that offers tutorials on these technologies.
- Visit [PerfectXML](#) for some good tips on XSLT.
- Check out the [XML Certification forum](#) at JavaRanch.
- You will need formatter software for displaying XSL-FO stylesheets. Get [Formatting Objects Processor \(FOP\)](#), an open source tool from Apache that you can use to display the output of the XSL-FO as PDF documents.
- Read [Professional XML, 2nd Edition](#), the prescribed book for the XML Certification Exam that covers these technologies to the extent required by the exam.
- Also pick up the [XML Bible](#) by Elliotte Rusty Harold. This is a good book to get you started on XML technologies.
- [Essential XML Quick Reference](#) covers all the relevant topics except security and XSL-FO.
- Get [Processing XML with Java](#), a very good book for understanding SAX and DOM. A free online edition of this book is available at <http://www.cafeconleche.org/books/xmljava/>.

## About the authors

### Pradeep Chopra

Pradeep Chopra is the cofounder of [Whizlabs Software](#), a global leader in IT skill assessment and certification exam preparation. A graduate of the Indian Institute of Technology (Delhi), Pradeep has consulted individuals and organizations across the globe on the values and benefits of IT certifications.

---

### Hari Vignesh Padmanaban

Hari Vignesh Padmanaban is the author of the [Whizlabs Exam Simulator for XML](#). He has MCP, SCJP, SCWCD, SCBCD, Object Oriented Analysis and Design using UML, and XML and Related Technologies certifications to his credit. He is currently working as a software engineer for [Invensys Foxboro](#).