20

XML and LINQ to XML

20.3 W3C XML Schema Documents

- Unlike DTDs, schemas use XML syntax and are actually XML documents that programs can manipulate.
- Like DTDs, schemas are used by validating parsers to validate documents.
- In this section, we focus on the W3C's **XML Schema** vocabulary.
- XML Schema enables schema authors to specify that element `quantity`'s data must be numeric or, even more specifically, an integer.
- An XML document that conforms to a schema document is **schema valid**, and one that does not conform is **schema invalid**.
- Schemas are XML documents and therefore must themselves be valid.

Validating Against an XML Schema Document

- By convention, schemas use the `.xsd` extension.
- Figure 20.3 contains markup describing several books.

`book.xml`

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig. 20.3: book.xml -->
<!-- Book list marked up as XML -->

<deitel:books xmlns:deitel = "http://www.deitel.com/booklist">
   <book>
      <title>Visual Basic 2008 How to Program</title>
   </book>

   <book>
      <title>Visual C# 2008 How to Program, 3/e</title>
   </book>

   <book>
      <title>Java How to Program, 7/e</title>
   </book>
```

The `books` element has the namespace prefix `deitel`, indicating that the `books` element is a part of the namespace `http://www.deitel.com/booklist`.

Fig. 20.3 | Schema-valid XML document describing a list of books. (Part 1 of 2.)

Outline

book.xml

(2 of 2)

```

   <book>
      <title>C++ How to Program, 6/e</title>
   </book>

   <book>
      <title>Internet and World Wide Web How to Program, 4/e</title>
   </book>
</deitel:books>
```

Fig. 20.3 | Schema-valid XML document describing a list of books. (Part 2 of 2.)

Outline

Creating an XML Schema Document

- Figure 20.4 presents the XML Schema document that specifies the structure of `book.xml`.

`book.xsd`

(1 of 3)

```
<?xml version = "1.0"?>
<!-- Fig. 20.4: book.xsd           -->
<!-- Simple W3C XML Schema document -->

<schema xmlns = "http://www.w3.org/2001/XMLSchema"
   xmlns:deitel = "http://www.deitel.com/booklist"
   targetNamespace = "http://www.deitel.com/booklist">

   <element name = "books" type = "deitel:BooksType"/>

```

Specifies the standard W3C XML Schema namespace as the default namespace.

Root element `schema` contains elements that define the structure of an XML document.

Bind the URI http://`www.deitel.com/booklist` to namespace prefix deitel.

Also specifiy `http://www.deitel.com/booklist` as the `targetNamespace` of the schema. This attribute identifies the namespace of the XML vocabulary that this schema defines.

In XML Schema, the `element` tag defines an element to be included in an XML document that conforms to the schema.

Fig. 20.4 | XML Schema document for `book.xml`. (Part 1 of 3.)

Outline

book.xsd

(2 of 3)

```
11     <complexType name = "BooksType">
12        <sequence>
13           <element name = "book" type = "deitel:SingleBookType"
14              minOccurs = "1" maxOccurs = "unbounded"/>
15        </sequence>
16     </complexType>
17
18     <complexType name = "SingleBookType">
19        <sequence>
20           <element name = "title" type = "string"/>
21        </sequence>
22     </complexType>
23  </schema>
```

Use element `complexType` to define `BooksType` as a complex type that has a child element named `book`.

The `sequence` element (lines 12–15) allows you to specify the sequential order in which child elements must appear.

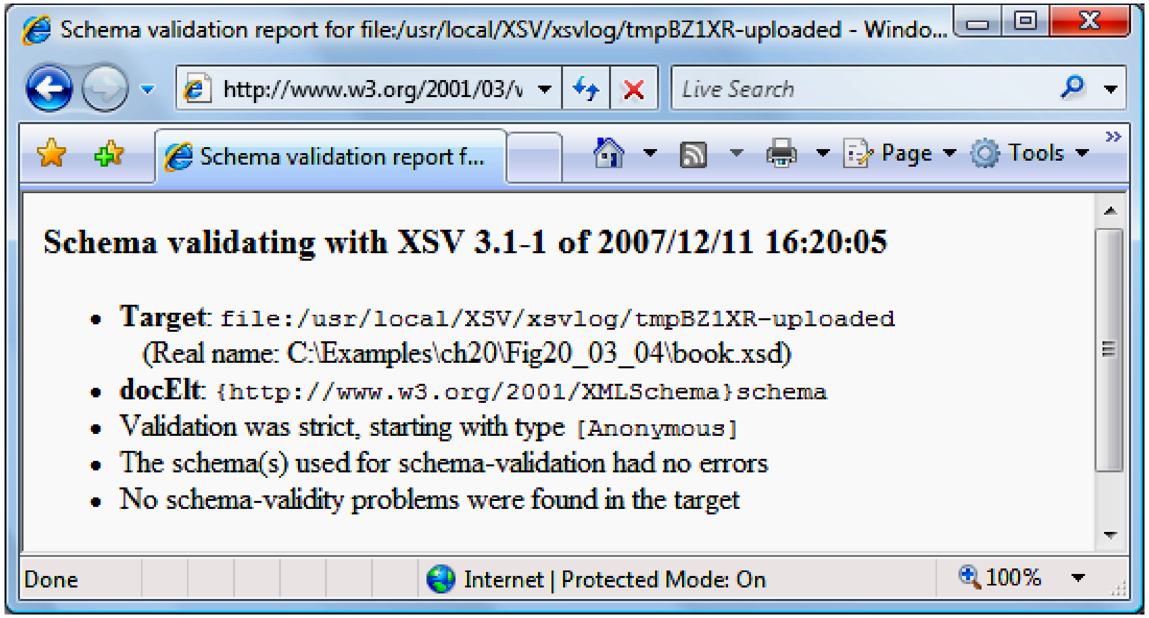
Attribute `minOccurs` specifies the minimum occurrences of an element. `maxOccurs` specifies the maximum; it can be set to `unbounded`.

Fig. 20.4 | XML Schema document for `book.xml`. (Part 2 of 3.)

Outline

`book.xsd`

(3 of 3)


Schema validation report for file:/usr/local/XSV/xsvlog/tmpBZ1XR-uploaded - Windo...
http://www.w3.org/2001/03/v
Live Search
Schema validation report f...
Page
Tools
Done
Internet | Protected Mode: On
100%

Schema validating with XSV 3.1-1 of 2007/12/11 16:20:05

- **Target**: `file:/usr/local/XSV/xsvlog/tmpBZ1XR-uploaded`
  (Real name: C:\Examples\ch20\Fig20_03_04\book.xsd)
- **docElt**: `{http://www.w3.org/2001/XMLSchema}schema`
- Validation was strict, starting with type `[Anonymous]`
- The schema(s) used for schema-validation had no errors
- No schema-validity problems were found in the target

Fig. 20.4 | XML Schema document for `book.xml`. (Part 3 of 3.)

20.3 W3C XML Schema Documents (Cont.)

- This schema defines an XML-based language (i.e., a vocabulary) for writing XML documents about collections of books.
- The schema defines the elements, attributes and parent-child relationships that such a document can (or must) include.
- The schema also specifies the type of data that these elements and attributes may contain.
- Root element `schema` contains elements that define the structure of an XML document.

20.3 W3C XML Schema Documents (Cont.)

Defining an Element in XML Schema

- In XML Schema, the `element` tag defines an element to be included in an XML document that conforms to the schema.
- An element's `type` attribute indicates the data type that the element may contain.
- Possible types include XML Schema–defined types and user-defined types.

20.3 W3C XML Schema Documents (Cont.)

- Figure 20.5 lists several of XML Schema's many built-in types.

Type	Description	Ranges or structures	Examples
`string`	A character string.		`"hello"`
`boolean`	True or false.	`true`, `false`	`true`
`decimal`	A decimal numeral.	`i * (10`^{n}`)`, where `i` is an integer and `n` is an integer that is less than or equal to zero.	`5`, `-12`, `-45.78`
`float`	A floating-point number.	`m * (2`^{e}`)`, where `m` is an integer whose absolute value is less than `2`^{24} and `e` is an integer in the range `-149` to `104`. Plus three additional numbers: positive infinity (`INF`), negative infinity (`-INF`) and not-a-number (`NaN`).	`0`, `12`, `-109.375`, `NaN`

Fig. 20.5 | Some XML Schema types. (Part 1 of 3.)

20.3 W3C XML Schema Documents (Cont.)

Type	Description	Ranges or structures	Examples
`double`	A floating-point number.	`m * (2`^{e}`)`, where `m` is an integer whose absolute value is less than `2`^{53} and `e` is an integer in the range `-1075` to `970`. Plus three additional numbers: positive infinity, negative infinity and not-a-number (`NaN`).	`0`, `12`, `-109.375`, `NaN`
`long`	A whole number.	`-9223372036854775808` to `9223372036854775807`, inclusive.	`1234567890`, `-1234567890`
`int`	A whole number.	`-2147483648` to `2147483647`, inclusive.	`1234567890`, `-1234567890`

Fig. 20.5 | Some XML Schema types. (Part 2 of 3.)

20.3 W3C XML Schema Documents (Cont.)

Type	Description	Ranges or structures	Examples
`short`	A whole number.	`-32768` to `32767`, inclusive.	`12`, `-345`
`date`	A date consisting of a year, month and day.	`yyyy-mm` with an optional `dd` and an optional time zone, where `yyyy` is four digits long and `mm` and `dd` are two digits long. The time zone is specified as `+hh:mm` or `-hh:mm`, giving an offset in hours and minutes.	`2008-07-25+01:00`
`time`	A time consisting of hours, minutes and seconds.	`hh:mm:ss` with an optional time zone, where hh, mm and ss are two digits long.	`16:30:25-05:00`

Fig. 20.5 | Some XML Schema types. (Part 3 of 3.)

20.3 W3C XML Schema Documents (Cont.)

- Two categories of types exist in XML Schema—**simple types** and **complex types**.
- They differ only in that simple types cannot contain attributes or child elements and complex types can.
- A user-defined type that contains attributes or child elements must be defined as a complex type.
- The `sequence` allows you to specify the sequential order in which child elements must appear.
- Attribute **`minOccurs`** specifies the minimum occurrences of an element. **`maxOccurs`** specifies the maximum; it can be set to **`unbounded`**.
- Both of these attributes have default values of `1`.

20.3 W3C XML Schema Documents (Cont.)

A Closer Look at Types in XML Schema

- Every element in XML Schema has a type that is either built-in or user-defined.
- Every simple type defines a **restriction** on a previously defined type.
- Complex types are divided into two groups—those with **simple content** and those with **complex content**.
  - Both can contain attributes.
  - Only complex content can contain child elements.
- Complex types with simple content must extend or restrict some other existing type.
- Complex types with complex content do not have this limitation.

- The schema document in Fig. 20.6 creates both simple types and complex types.

computer.xsd

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig. 20.6: computer.xsd -->
<!-- W3C XML Schema document -->

<schema xmlns = "http://www.w3.org/2001/XMLSchema"
   xmlns:computer = "http://www.deitel.com/computer"
   targetNamespace = "http://www.deitel.com/computer">

   <simpleType name = "gigahertz">
      <restriction base = "decimal">
         <minInclusive value = "2.1"/>
      </restriction>
   </simpleType>

   <complexType name = "CPU">
      <simpleContent>
         <extension base = "string">
```

Specify the base type as `decimal`.

Restrict the value to be at least `2.1`.

Create a simple type using the `simpleType` element.

Declare a `complexType` named `CPU` that has `simpleContent` and a base type of `string`.

The `extension` element with attribute `base` sets the base type.

Fig. 20.6 | XML Schema document defining simple and complex types. (Part 1 of 2.)

Outline

computer.xsd

(2 of 2)

```
            <attribute name = "model" type = "string"/>
         </extension>
      </simpleContent>
   </complexType>

   <complexType name = "portable">
      <all>
         <element name = "processor" type = "computer:CPU"/>
         <element name = "monitor" type = "int"/>
         <element name = "CPUSpeed" type = "computer:gigahertz"/>
         <element name = "RAM" type = "int"/>
      </all>
      <attribute name = "manufacturer" type = "string"/>
   </complexType>

   <element name = "laptop" type = "computer:portable"/>
</schema>
```

The `attribute` element specifies that an element of type `CPU` and may contain a `model` attribute that is of type `string`.

The element `all` encloses elements that must each be included once in the document. These elements can be included in any order.

`portable` contains an attribute of type `string` named `manufacturer`.

Declare the actual element, called `laptop`, that uses the three types defined in the schema.

Fig. 20.6 | XML Schema document defining simple and complex types. (Part 2 of 2.)

20.3 W3C XML Schema Documents (Cont.)

- A document that conforms to a schema is known as an **XML instance document**.
- Create a simple type using the **`simpleType`** element.
- Simple types are restrictions of a type typically called a **base type**.
- The **`extension`** element with attribute **`base`** sets the base type.
- The element **`all`** encloses elements that must each be included once in the document. These elements can be included in any order.

- Figure 20.7 uses the `laptop` element defined in the `computer.xsd` schema.

`laptop.xml`

```xml
<?xml version = "1.0"?>
<!-- Fig. 20.7: laptop.xml              -->
<!-- Laptop components marked up as XML -->

<computer:laptop xmlns:computer = "http://www.deitel.com/computer"
   manufacturer = "IBM">

   <processor model = "Centrino">Intel</processor>
   <monitor>17</monitor>
   <CPUSpeed>2.4</CPUSpeed>
   <RAM>256</RAM>
</computer:laptop>
```

Fig. 20.7 | XML document using the laptop element defined in `computer.xsd`.

20.3 W3C XML Schema Documents (Cont.)

Automatically Creating Schemas using Visual Studio

- Visual Studio includes a tool that allows you to create a schema from an existing XML document, using the document as a template.
- With an XML document open, select **XML > Create Schema** to use this feature.

Good Programming Practice 20.1

The schema generated by Visual Studio is a good starting point, but you should refine the restrictions and types it specifies so they are appropriate for your XML documents.

20.4 Extensible Stylesheet Language and XSL Transformations

- **Extensible Stylesheet Language** (**XSL**) documents specify how programs are to render XML document data.
- XSL is a group of three technologies—**XSL-FO** (**XSL Formatting Objects**), **XPath** (**XML Path Language**) and **XSLT** (**XSL Transformations**).
  - XSL-FO is a vocabulary for specifying formatting.
  - XPath is a string-based language used by XML to locate structures and data in XML documents.
  - XSL Transformations (XSLT) is a technology for transforming XML documents into other documents.
- XSLT allows you to convert an XML document to an **XHTML** (**Extensible HyperText Markup Language**) document for display in a web browser.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- Transforming an XML document using XSLT involves two tree structures—the **source tree** and the **result tree**.
- XPath is used to locate parts of the source-tree document that match **templates** defined in an **XSL style sheet**.
- When a match occurs, the matching template executes and adds its result to the result tree.
- The XSLT does not analyze every node of the source tree; it selectively navigates the source tree using XSLT’s `select` and `match` attributes.
- For XSLT to function, the source tree must be properly structured.

Outline

A Simple XSL Example

- Figure 20.8 lists an XML document that describes various sports.

sports.xsl

(1 of 2)

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "sports.xsl"?>

<!-- Fig. 20.8: sports.xml -->
<!-- Sports Database -->

<sports>
   <game id = "783">
      <name>Cricket</name>

      <paragraph>
         More popular among Commonwealth nations.
      </paragraph>
   </game>

   <game id = "239">
      <name>Baseball</name>

```

This **processing instruction** (**PI**) specifies the location of the XSL style sheet `sports.xsl`, which will be used to transform the XML document.

Fig. 20.8 | XML document that describes various sports. (Part 1 of 2.)

Outline

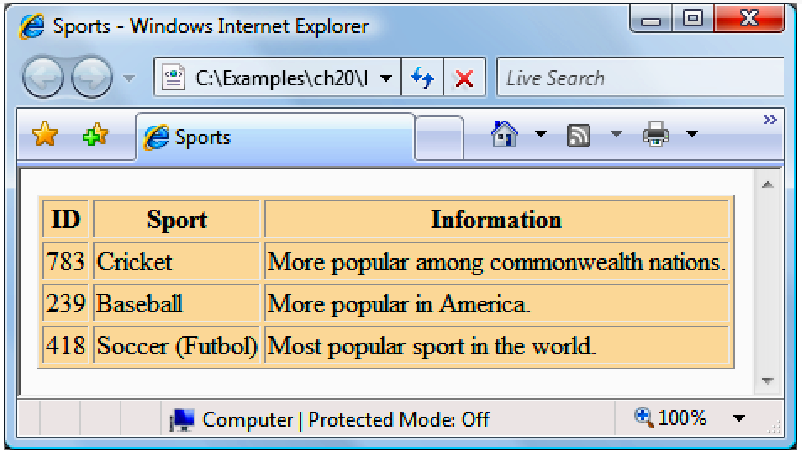
```
      <paragraph>
         More popular in America.
      </paragraph>
   </game>

   <game id = "418">
      <name>Soccer (Futbol)</name>

      <paragraph>
         Most popular sport in the world.
      </paragraph>
   </game>
</sports>
```

sports.xsl

(2 of 2)



Sports - Windows Internet Explorer

C:\Examples\ch20\

ID	Sport	Information
783	Cricket	More popular among commonwealth nations.
239	Baseball	More popular in America.
418	Soccer (Futbol)	Most popular sport in the world.

Computer | Protected Mode: Off

Fig. 20.8 | XML document that describes various sports. (Part 2 of 2.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The output shows the result of the transformation (specified in the XSLT template of Fig. 20.9) rendered by Internet Explorer 7.
- To perform transformations, an XSLT processor is required.
- The XML document shown in Fig. 20.8 is transformed into an XHTML document by MSXML when the document is loaded in Internet Explorer.
- MSXML is both an XML parser and an XSLT processor.
- The characters **<?** and **?>** delimit a processing instruction, which consists of a **PI target** and a **PI value**.

Software Engineering Observation 20.4

XSL enables document authors to separate data presentation (specified in XSL documents) from data description (specified in XML documents).

- Figure 20.9 shows the XSL document for transforming the structured data of the XML document of Fig. 20.8 into an XHTML document for presentation.

sports.xsl

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig. 20.9: sports.xsl -->
<!-- A simple XSLT transformation -->

<!-- reference XSL style sheet URI -->
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

   <xsl:output method = "xml" omit-xml-declaration = "no"
      doctype-system =
         "http://www.w3c.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
      doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>

   <xsl:template match = "/"> <!-- match root element -->

   <html xmlns = "http://www.w3.org/1999/xhtml">
      <head>
         <title>Sports</title>
      </head>

```

The style sheet begins with the `stylesheet` start tag. Attribute `version` specifies the XSLT version.

Attribute `method` is assigned `"xml"`, which indicates that XML is being output to the result tree. Attribute `omit-xml-declaration` specifies that we should not want to omit the XML declaration

Use element `xsl:output` to write an XHTML document type declaration (`DOCTYPE`) to the result tree.

Attributes `doctype-system` and `doctype-public` write the `DOCTYPE` DTD information to the result tree.

Use the `match` attribute to select the **document root** of the XML source document.

The XSLT processor writes this XHTML to the result tree exactly as it appears in the XSL document.

Fig. 20.9 | XSLT that creates elements and attributes in an XHTML document. (Part 1 of 2.)

Outline

sports.xsl

(2 of 2)

```
      <body>
         <table border = "1" style = "background-color: wheat">
            <thead>
               <tr>
                  <th>ID</th>
                  <th>Sport</th>
                  <th>Information</th>
               </tr>
            </thead>

            <!-- insert each name and paragraph element value -->
            <!-- into a table row. -->
            <xsl:for-each select = "/sports/game">
               <tr>
                  <td><xsl:value-of select = "@id"/></td>
                  <td><xsl:value-of select = "name"/></td>
                  <td><xsl:value-of select = "paragraph"/></td>
               </tr>
            </xsl:for-each>
         </table>
      </body>
   </html>

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

Use element `value-of` to retrieve an attribute's value using the XPath symbol `@`.

Element `xsl:for-each` iterates through the source XML document, searching for the element specified by the `select` attribute.

Fig. 20.9 | XSLT that creates elements and attributes in an XHTML document. (Part 2 of 2.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The style sheet begins with the `stylesheet` start tag. Attribute `version` specifies the XSLT version.
- The `DOCTYPE` identifies XHTML as the type of the resulting document.
- Attributes `doctype-system` and `doctype-public` write the `DOCTYPE` DTD information to the result tree.
- XSLT uses **templates** (i.e., `xsl:template` elements) to describe how to transform particular nodes from the source tree to the result tree.
- A template is applied to nodes that are specified in the `match` attribute.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The XPath character `/` (a forward slash) is used as a separator between element names.
- In XPath, a leading forward slash specifies that we are using **absolute addressing** (i.e., we are starting from the root).
- Element `xsl:for-each` iterates through the source XML document, searching for the element specified by the `select` attribute.
- Use element `value-of` to retrieve an attribute's value using the XPath symbol `@`.
- When an XPath expression has no beginning forward slash, the expression uses **relative addressing**.

Using XSLT to Sort and Format Data

- Figure 20.10 presents an XML document (`sorting.xml`) that marks up information about a book.

`sorting.xml`

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig. 20.10: sorting.xml -->
<!-- XML document containing book information -->

<?xml-stylesheet type = "text/xsl" href = "sorting.xsl"?>

<book isbn = "999-99999-9-X">
   <title>Deitel&apos;s XML Primer</title>

   <author>
      <firstName>Jane</firstName>
      <lastName>Blue</lastName>
   </author>

   <chapters>
```

Fig. 20.10 | XML document containing book information. (Part 1 of 2.)

Outline

`sorting.xml`

(2 of 2)

```
      <frontMatter>
         <preface pages = "2" />
         <contents pages = "5" />
         <illustrations pages = "4" />
      </frontMatter>

      <chapter number = "3" pages = "44">Advanced XML</chapter>
      <chapter number = "2" pages = "35">Intermediate XML</chapter>
      <appendix number = "B" pages = "26">Parsers and Tools</appendix>
      <appendix number = "A" pages = "7">Entities</appendix>
      <chapter number = "1" pages = "28">XML Fundamentals</chapter>
   </chapters>

   <media type = "CD" />
</book>
```

Fig. 20.10 | XML document containing book information. (Part 2 of 2.)

- XSL style sheet can sort an XML file’s data for presentation purposes.

Outline

- Figure 20.11 presents an XSL document (`sorting.xsl`) for transforming `sorting.xml` (Fig. 20.10) to XHTML.

sorting.xsl

(1 of 5)

```
<?xml version = "1.0"?>
<!-- Fig. 20.11: sorting.xsl -->
<!-- Transformation of book information into XHTML -->

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

   <!-- write XML declaration and DOCTYPE DTD information -->
   <xsl:output method = "xml" omit-xml-declaration = "no"
      doctype-system = "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
      doctype-public = "-//W3C//DTD XHTML 1.1//EN"/>

   <!-- match document root -->
   <xsl:template match = "/">
      <html>
         <xsl:apply-templates/>
      </html>
   </xsl:template>

```

The `<xsl:apply-templates/>` element specifies that the XSLT processor is to apply the `xsl:template`s defined in this XSL document to the current node's children.

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 1 of 5.)

Outline

sorting.xsl

(2 of 5)

```
   <!-- match book -->
   <xsl:template match = "book">
      <head>
         <title>ISBN <xsl:value-of select = "@isbn"/> -
            <xsl:value-of select = "title"/></title>
      </head>

      <body>
         <h1 style = "color: blue"><xsl:value-of select = "title"/></h1>
         <h2 style = "color: blue">by
            <xsl:value-of select = "author/firstName"/>
            <xsl:text> </xsl:text>
            <xsl:value-of select = "author/lastName"/>
         </h2>

         <table style = "border-style: groove; background-color: wheat">

            <xsl:for-each select = "chapters/frontMatter/*">
               <tr>
                  <td style = "text-align: right">
                     <xsl:value-of select = "name()"/>
                  </td>
```

Use the `book`'s ISBN (from attribute `isbn`) and the contents of element `title` to create the string that appears in the browser window's title bar.

The `xsl:text` element is used to insert literal text.

Select each element (indicated by an asterisk) that is a child of element `frontMatter`.

Node-set function `name` retrieves the current node's element name.

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 2 of 5.)

Outline

```

                  <td>
                     ( <xsl:value-of select = "@pages"/> pages )
                  </td>
               </tr>
            </xsl:for-each>

            <xsl:for-each select = "chapters/chapter">
               <xsl:sort select = "@number" data-type = "number"
                    order = "ascending"/>
               <tr>
                  <td style = "text-align: right">
                     Chapter <xsl:value-of select = "@number"/>
                  </td>

                  <td>
                     <xsl:value-of select = "text()"/>
                     ( <xsl:value-of select = "@pages"/> pages )
                  </td>
               </tr>
            </xsl:for-each>
```

sorting.xsl

(3 of 5)

Use element `xsl:sort` to sort the selected elements by the value given by its `select` attribute.

Use **node-set function** `text` to obtain the text between the `chapter` start and end tags.

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 3 of 5.)

Outline

sorting.xsl

(4 of 5)

```

               <xsl:for-each select = "chapters/appendix">
                  <xsl:sort select = "@number" data-type = "text"
                     order = "ascending"/>
                  <tr>
                     <td style = "text-align: right">
                        Appendix <xsl:value-of select = "@number"/>
                     </td>

                     <td>
                        <xsl:value-of select = "text()"/>
                        ( <xsl:value-of select = "@pages"/> pages )
                     </td>
                  </tr>
               </xsl:for-each>
            </table>

```

Fig. 20.11 | XSL document that transforms sorting.xml into XHTML. (Part 4 of 5.)

Outline

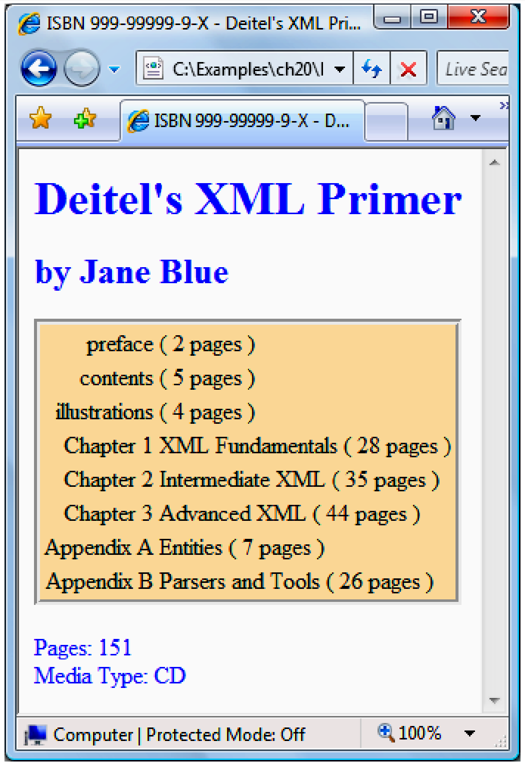
```
            <p style = "color: blue">Pages:
               <xsl:variable name = "pagecount"
                  select = "sum(chapters//*/@pages)"/>
               <xsl:value-of select = "$pagecount"/>
            <br />Media Type: <xsl:value-of select = "media/@type"/></p>
         </body>
      </xsl:template>
   </xsl:stylesheet>
```

sorting.xsl

(5 of 5)

Use an **XSL variable** to store the value of the book's total page count and output the page count to the result tree.

Function **sum** totals a set of values found using XPath.


ISBN 999-99999-9-X - Deitel's XML Pri...
C:\Examples\ch20\I
Live Sea
ISBN 999-99999-9-X - D...
Computer | Protected Mode: Off
100%

Deitel's XML Primer

by Jane Blue

preface (2 pages)
contents (5 pages)
illustrations (4 pages)
Chapter 1 XML Fundamentals (28 pages)
Chapter 2 Intermediate XML (35 pages)
Chapter 3 Advanced XML (44 pages)
Appendix A Entities (7 pages)
Appendix B Parsers and Tools (26 pages)

Pages: 151
Media Type: CD

Fig. 20.11 | XSL document that transforms `sorting.xml` into XHTML. (Part 5 of 5.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- The `<xsl:apply-templates/>` element specifies that the XSLT processor is to apply the `xsl:template`s defined in this XSL document to the current node's children.
- The **`xsl:text`** element is used to insert literal text.
- **Node-set function `name`** retrieves the current node's element name.
- Use element **`xsl:sort`** to sort the selected elements by the value given by its **`select`** attribute.
- Attribute **`data-type`**, with value `"number"`, specifies a numeric sort. This attribute also accepts the value `"text"`.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

- Attribute `order` can be set to either `"ascending"` or `"descending"`.
- XSL variables cannot be modified after they are initialized.
  - Attribute `name` specifies the variable's name
  - Attribute `select` assigns a value to the variable.
- Function `sum` totals a set of values found using XPath.
- Two slashes in an XPath expression indicate **recursive descent**.

Performance Tip 20.1

Selecting all nodes in a document when it is not necessary slows XSLT processing.

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

Summary of XSL Style-Sheet Elements

- Figure 20.12 lists commonly used XSL elements.

Element	Description
`<xsl:apply-templates>`	Applies the templates of the XSL document to the children of the current node.
`<xsl:apply-templates match = "`*expression*`">`	Applies the templates of the XSL document to the children of the nodes matching *expression*. The value of the attribute match (i.e., *expression*) must be an XPath expression that specifies elements.
`<xsl:template>`	Contains rules to apply when a specified node is matched.
`<xsl:value-of select = "`*expression*`">`	Selects the value of an XML element and adds it to the output tree of the transformation. The required `select` attribute contains an XPath expression.

Fig. 20.12 | XSL style-sheet elements. (Part 1 of 2.)

20.4 Extensible Stylesheet Language and XSL Transformations (Cont.)

Element	Description
`<xsl:for-each select = "`*expression*`">`	Applies a template to every node selected by the XPath specified by the `select` attribute.
`<xsl:sort select = "`*expression*`">`	Used as a child element of an `<xsl:apply-templates>` or `<xsl:for-each>` element. Sorts the nodes selected by the `<xsl:apply-template>` or `<xsl:for-each>` element so that the nodes are processed in sorted order.
`<xsl:output>`	Has various attributes to define the format (e.g., XML, XHTML), version (e.g., 1.0, 2.0), document type and MIME type of the output document. MIME types are discussed in Section 22.2. This tag is a top-level element—it can be used only as a child element of an `xsl:stylesheet`.
`<xsl:copy>`	Adds the current node to the output tree.

Fig. 20.12 | XSL style-sheet elements. (Part 2 of 2.)