14

XML

OBJECTIVES

In this chapter you will learn:

- To mark up data using XML.
- How XML namespaces help provide unique XML element and attribute names.
- To create DTDs and schemas for specifying and validating the structure of an XML document.
- To create and use simple XSL style sheets to render XML document data.
- To retrieve and manipulate XML data programmatically using JavaScript..

Outline

14.1 Introduction

14.2 XML Basics

14.3 Structuring Data

14.4 XML Namespaces

14.5 Document Type Definitions (DTDs)

14.6 W3C XML Schema Documents

14.8 Extensible Stylesheet Language and XSL Transformations

14.9 Document Object Model (DOM)

14.1 Introduction

- XML is a portable, widely supported, open (i.e., nonproprietary) technology for data storage and exchange

14.2 XML Basics

- XML documents are readable by both humans and machines
- XML permits document authors to create custom markup for any type of information
  - Can create entirely new markup languages that describe specific types of data, including mathematical formulas, chemical molecular structures, music and recipes
- An XML parser is responsible for identifying components of XML documents (typically files with the `.xml` extension) and then storing those components in a data structure for manipulation
- An XML document can optionally reference a Document Type Definition (DTD) or schema that defines the XML document's structure
- If an XML parser can process an XML document successfully, that XML document is well-formed

14.2 XML Basics

- XML stands for EXtensible Markup Language
- XML was designed to carry and store data, not to display data
- XML is a markup language much like HTML
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML Simplifies Data Sharing
- XML is a W3C Recommendation

The difference Between XML and HTML

•XML Separates Data from HTML

•XML is not a replacement for HTML.

•HTML is about displaying information, while XML is about carrying information.

14.2 XML Basics

XML Makes Your Data More Available

Since XML is independent of hardware, software and application, XML can make your data more available and useful.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

Future applications will exchange their data in XML.

The future might give us word processors, spreadsheet applications and databases that can read each other's data in a pure text format, without any conversion utilities in between.

XML is Used to Create New Internet Languages

A lot of new Internet languages are created with XML.

Here are some examples:

XHTML the latest version of HTML

WSDL for describing available web services

WAP and WML as markup languages for handheld devices

RDF and OWL for describing resources and ontology

SMIL for describing multimedia for the web

14.3 Structuring Data

- An XML document begins with an optional XML declaration, which identifies the document as an XML document. The version attribute specifies the version of XML syntax used in the document.
- XML comments begin with <!-- and end with -->

- An XML document contains text that represents its content (i.e., data) and elements that specify its structure. XML documents delimit an element with start and end tags

- The root element of an XML document encompasses all its other elements
- An XML element is everything from (including) the element's start tag to (including) the element's end tag.

- XML element names can be of any length and can contain letters, digits, underscores, hyphens and periods
  - Must begin with either a letter or an underscore, and they should not begin with “xml” in any combination of uppercase and lowercase letters, as this is reserved for use in the XML standards

14.3 Structuring Data

XML Attributes

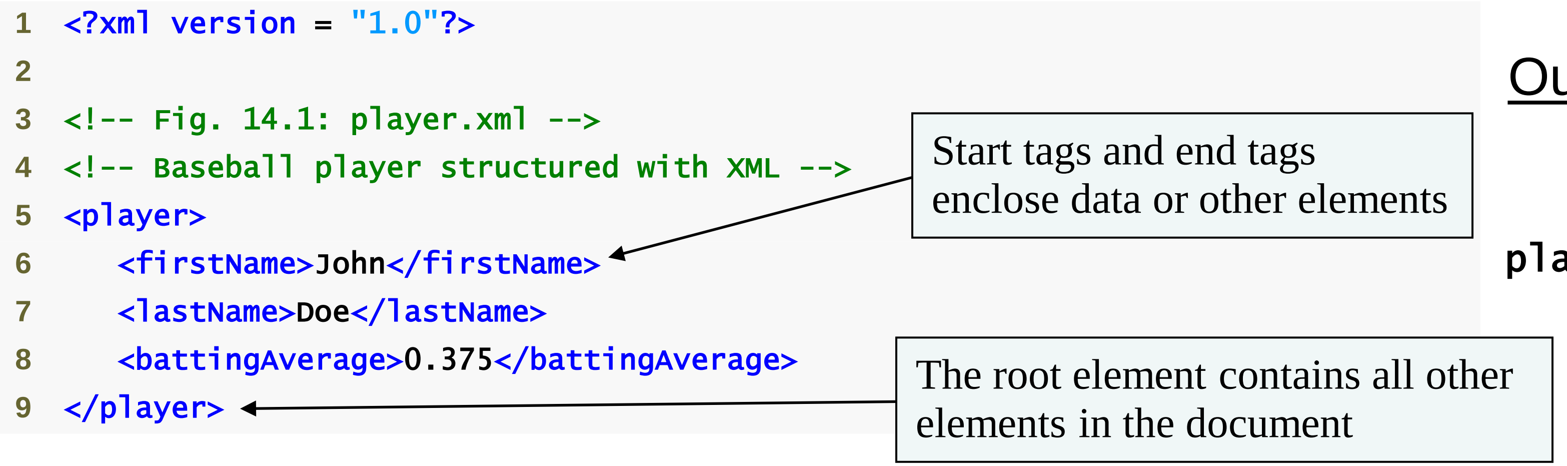
- In HTML (and in XML) attributes provide additional information about elements:

- Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but important to the software that wants to manipulate the element:

  ```
  <file type="gif">computer.gif</file>
  ```

- Attribute values must always be enclosed in quotes, but either single or double quotes can be used.

Outline



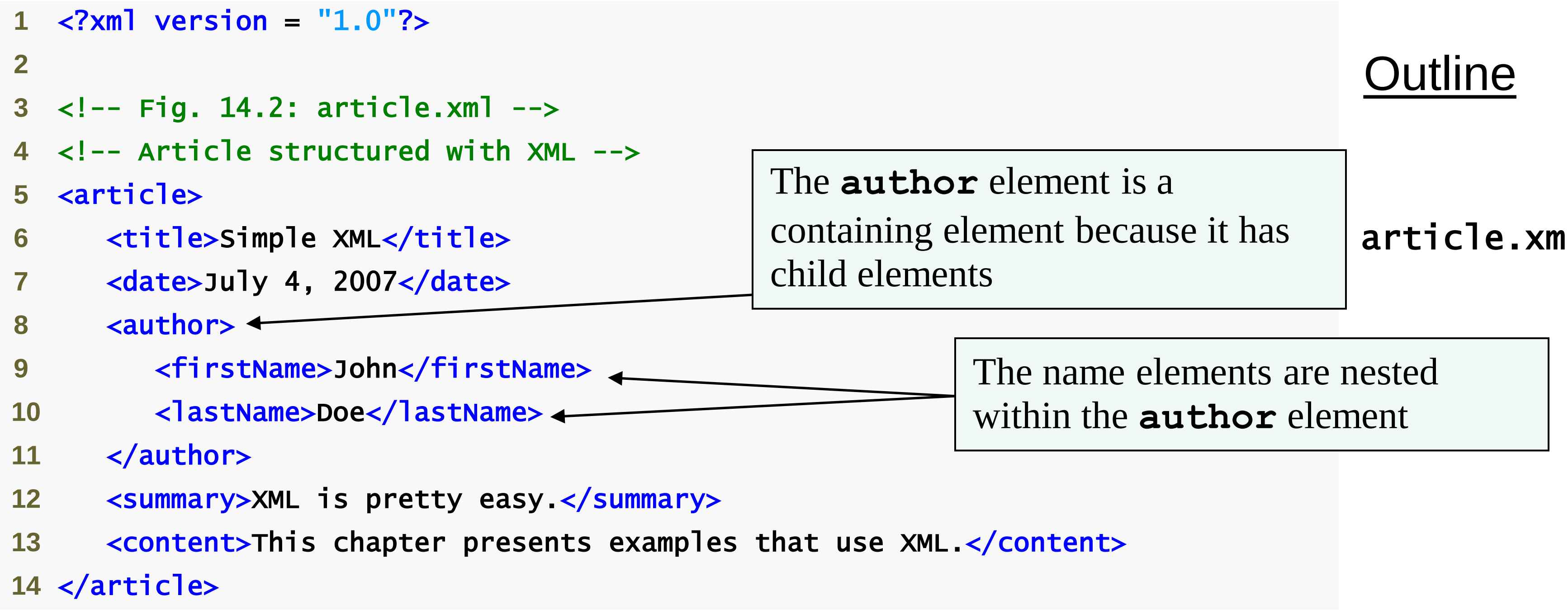
```
<?xml version = "1.0"?>

<!-- Fig. 14.1: player.xml -->
<!-- Baseball player structured with XML -->
<player>
   <firstName>John</firstName>
   <lastName>Doe</lastName>
   <battingAverage>0.375</battingAverage>
</player>
```

Start tags and end tags enclose data or other elements

The root element contains all other elements in the document

`player.xml`

Outline



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

<!-- Fig. 14.2: article.xml -->
<!-- Article structured with XML -->
<article>
   <title>Simple XML</title>
   <date>July 4, 2007</date>
   <author>
      <firstName>John</firstName>
      <lastName>Doe</lastName>
   </author>
   <summary>XML is pretty easy.</summary>
   <content>This chapter presents examples that use XML.</content>
</article>
```

The **`author`** element is a containing element because it has child elements

The name elements are nested within the **`author`** element

`article.xml`

Outline

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

<!-- Fig. 14.4: letter.xml -->
<!-- Business letter marked up as XML -->
<!DOCTYPE letter SYSTEM "letter.dtd">

<letter>
   <contact type = "sender">
      <name>Jane Doe</name>
      <address1>Box 12345</address1>
      <address2>15 Any Ave.</address2>
      <city>Othertown</city>
      <state>Otherstate</state>
      <zip>67890</zip>
      <phone>555-4321</phone>
      <flag gender = "F" />
   </contact>

   <contact type = "receiver">
      <name>John Doe</name>
      <address1>123 Main St.</address1>
      <address2></address2>
      <city>Anytown</city>
      <state>Anystate</state>
      <zip>12345</zip>
      <phone>555-1234</phone>
      <flag gender = "M" />
   </contact>

   <salutation>Dear Sir:</salutation>
```

The **DOCTYPE** specifies an external DTD in the file **letter.dtd**

Data can be stored as attributes, which appear in an element’s start tag

flag is an empty element because it contains no child elements or content

letter.xml

(1 of 2)

Outline

```

   <paragraph>It is our privilege to inform you about our new database
      managed with XML. This new system allows you to reduce the
      load on your inventory list server by having the client machine
      perform the work of sorting and filtering the data.
   </paragraph>

   <paragraph>Please visit our website for availability and pricing.
   </paragraph>

   <closing>Sincerely,</closing>
   <signature>Ms. Jane Doe</signature>
</letter>
```

letter.xml

(2 of 2)

XML Elements vs. Attributes

- Both examples provide the same information

```
 <person>
   <citizenship>american</citizenship>
</person>

<person citizenship=“american">
</person>
```

- In XML it is recommended to use elements instead.
    - attributes cannot contain multiple values (child elements can)
    - attributes are not easily expandable (for future changes)
    - attributes cannot describe structures (child elements can)
    - attributes are more difficult to manipulate by program code
    - attribute values are not easy to test against a Document Type Definition (DTD) - which is used to define the legal elements of an XML document

14.4 Namespaces

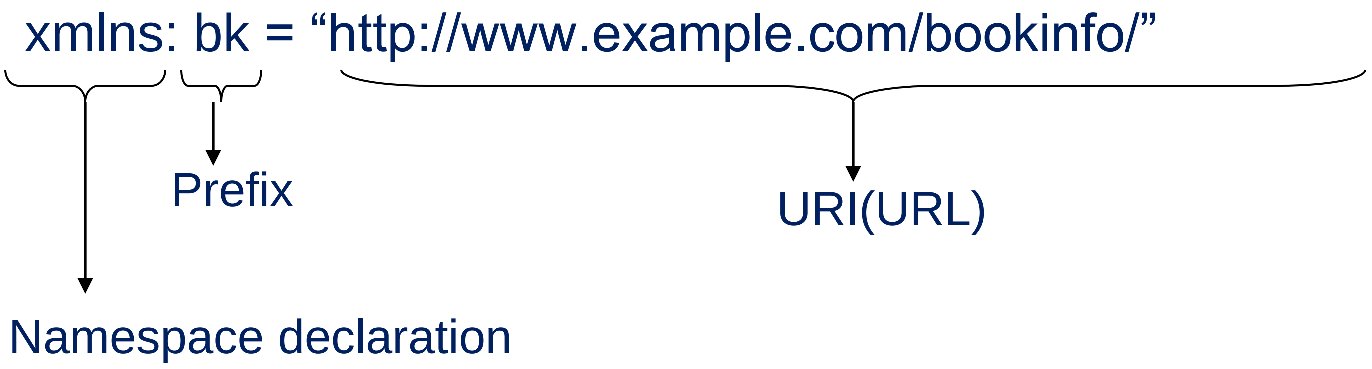
- XML namespaces provide a means for document authors to prevent naming collisions

- Each namespace prefix is bound to a uniform resource identifier (URI) that uniquely identifies the namespace
  - A URI is a series of characters that differentiate names
  - Document authors create their own namespace prefixes

- To eliminate the need to place a namespace prefix in each element, authors can specify a default namespace for an element and its children
  - We declare a default namespace using keyword `xmlns` with a URI (Uniform Resource Identifier) as its value

- Document authors commonly use URLs (Uniform Resource Locators) for URIs, because domain names (e.g., `deitel.com`) in URLs must be unique

14.4 Namespaces



xmlns: bk = “http://www.example.com/bookinfo/”

Prefix

URI(URL)

Namespace declaration

Example:

```
<BOOK xmlns:bk=“http://www.bookstuff.org/bookinfo”>
            <bk:TITLE>  All About XML    </bk:TITLE>
            <bk:AUTHOR> Joe Developer    </bk:AUTHOR>
            <bk:PRICE currency=‘US Dollar’> 19.99  </bk:PRICE>
</BOOK>
```

14.4 Namespaces

- An XML namespace declared without a prefix becomes the default namespace for all sub-elements

- All elements without a prefix will belong to the default namespace:

```
<BOOK xmlns=“http://www.bookstuff.org/bookinfo”>
        <TITLE> All About XML </TITLE>
        <AUTHOR> Joe Developer </AUTHOR>
</BOOK>
```

14.4 Namespaces

- Unqualified elements belong to the inner-most default namespace.
  - BOOK, TITLE, and AUTHOR belong to the default book namespace
  - PUBLISHER and NAME belong to the default publisher namespace

```
<BOOK xmlns="www.bookstuff.org/bookinfo">
        <TITLE> All About XML </TITLE>
        <AUTHOR> Joe Developer </AUTHOR>
        <PUBLISHER xmlns="urn:publishers:publinfo">
                <NAME> Microsoft Press </NAME>
        </PUBLISHER>
</BOOK>
```

Outline

namespace.xml

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

<!-- Fig. 14.7: namespace.xml -->
<!-- Demonstrating namespaces -->
<text:directory
   xmlns:text = "urn:deitel:textInfo"
   xmlns:image = "urn:deitel:imageInfo">

   <text:file filename = "book.xml">
      <text:description>A book list</text:description>
   </text:file>

   <image:file filename = "funny.jpg">
      <image:description>A funny picture</image:description>
      <image:size width = "200" height = "100" />
   </image:file>
</text:directory>
```

Two namespaces are specified using URNs

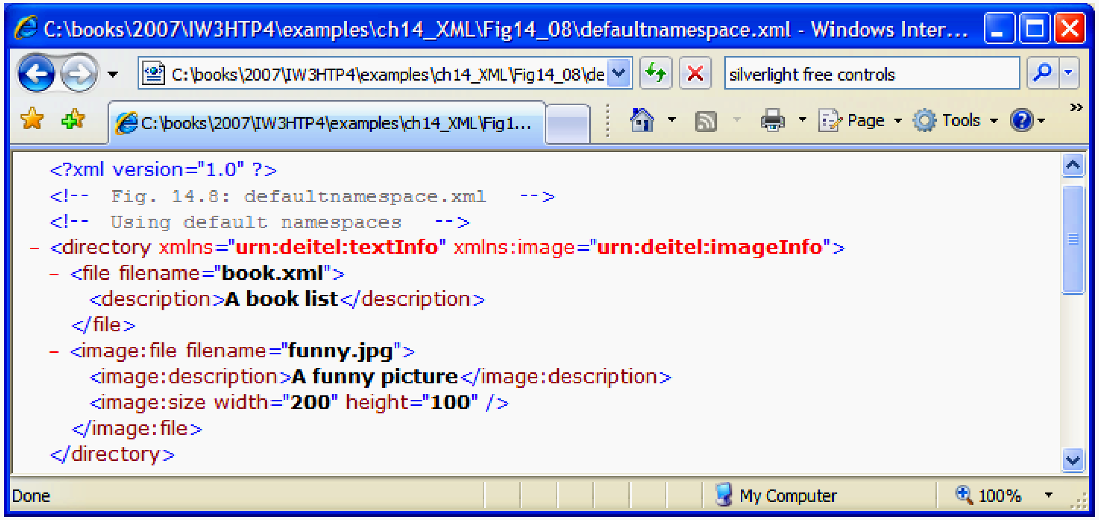
The namespace prefixes are used in element names throughout the document

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

<!-- Fig. 14.8: defaultnamespace.xml -->
<!-- Using default namespaces -->
<directory xmlns = "urn:deitel:textInfo"
   xmlns:image = "urn:deitel:imageInfo">

   <file filename = "book.xml">
      <description>A book list</description>
   </file>

   <image:file filename = "funny.jpg">
      <image:description>A funny picture</image:description>
      <image:size width = "200" height = "100" />
   </image:file>
</directory>
```

The default namespace is set in the **directory** element

Elements with no namespace prefix use the default namespace

defaultnamespace.xml


C:\books\2007\IW3HTP4\examples\ch14_XML\Fig14_08\defaultnamespace.xml - Windows Inter...
C:\books\2007\IW3HTP4\examples\ch14_XML\Fig14_08\de
silverlight free controls
C:\books\2007\IW3HTP4\examples\ch14_XML\Fig1...
Page
Tools
Done
My Computer
100%

```
  <?xml version="1.0" ?>
  <!--  Fig. 14.8: defaultnamespace.xml   -->
  <!--  Using default namespaces   -->
- <directory xmlns="urn:deitel:textInfo" xmlns:image="urn:deitel:imageInfo">
  - <file filename="book.xml">
      <description>A book list</description>
    </file>
  - <image:file filename="funny.jpg">
      <image:description>A funny picture</image:description>
      <image:size width="200" height="100" />
    </image:file>
  </directory>
```

14.5 Document Type Definitions (DTDs)

- DTDs and schemas specify documents’ element types and attributes, and their relationships to one another

- DTDs and schemas enable an XML parser to verify whether an XML document is valid (i.e., its elements contain the proper attributes and appear in the proper sequence)

- A DTD expresses the set of rules for document structure using an EBNF (Extended Backus-Naur Form) grammar

- In a DTD, an ELEMENT element type declaration defines the rules for an element. An ATTLIST attribute-list declaration defines attributes for a particular element

- You can force text to be treated as unparsed character data by enclosing it in <![CDATA[...]]>

Outline

```
<!-- Fig. 14.9: letter.dtd      -->
<!-- DTD document for letter.xml -->

<!ELEMENT letter ( contact+, salutation, paragraph+,
   closing, signature )>

<!ELEMENT contact ( name, address1, address2, city, state,
   zip, phone, flag )>
<!ATTLIST contact type CDATA #IMPLIED>

<!ELEMENT name ( #PCDATA )>
<!ELEMENT address1 ( #PCDATA )>
<!ELEMENT address2 ( #PCDATA )>
<!ELEMENT city ( #PCDATA )>
<!ELEMENT state ( #PCDATA )>
<!ELEMENT zip ( #PCDATA )>
<!ELEMENT phone ( #PCDATA )>
<!ELEMENT flag EMPTY>
<!ATTLIST flag gender (M | F) "M">

<!ELEMENT salutation ( #PCDATA )>
<!ELEMENT closing ( #PCDATA )>
<!ELEMENT paragraph ( #PCDATA )>
<!ELEMENT signature ( #PCDATA )>
```

Define the requirements for the `letter` element

`letter.dtd`

Define the requirements for the `contact` element

A `contact` element may have a `type` attribute, but it is not required

Each of these elements contains parsed character data

The `flag` element must be empty and its `gender` attribute must be set to either `M` or `F`. If there is no `gender` attribute, gender defaults to `M`

14.6 W3C XML Schema Documents

- Unlike DTDs
  - Schemas use XML syntax not EBNF grammar
  - XML Schema documents can specify what type of data (e.g., numeric, text) an element can contain
- An XML document that conforms to a schema document is schema valid
- Two categories of types exist in XML Schema: simple types and complex types
  - Simple types cannot contain attributes or child elements; complex types can
- Every simple type defines a restriction on an XML Schema-defined schema type or on a user-defined type
- Complex types can have either simple content or complex content
  - Both can contain attributes, but only complex content can contain child element
- **XML schema types:** string , boolean, decimal, float, double, int, long, short, date, time, …

Outline

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

<!-- Fig. 14.11: book.xml -->
<!-- Book list marked up as XML -->
<deitel:books xmlns:deitel = "http://www.deitel.com/booklist">
   <book>
      <title>Visual Basic 2005 How to Program, 3/e</title>
   </book>
   <book>
      <title>Visual C# 2005 How to Program, 2/e</title>
   </book>
   <book>
      <title>Java How to Program, 7/e</title>
   </book>
   <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>
```

book.xml

book.xsd

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

<!-- Fig. 14.12: 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"/>

   <complexType name = "BooksType">
      <sequence>
         <element name = "book" type = "deitel:SingleBookType"
            minOccurs = "1" maxOccurs = "unbounded"/>
      </sequence>
   </complexType>

   <complexType name = "SingleBookType">
      <sequence>
         <element name = "title" type = "string"/>
      </sequence>
   </complexType>
</schema>
```

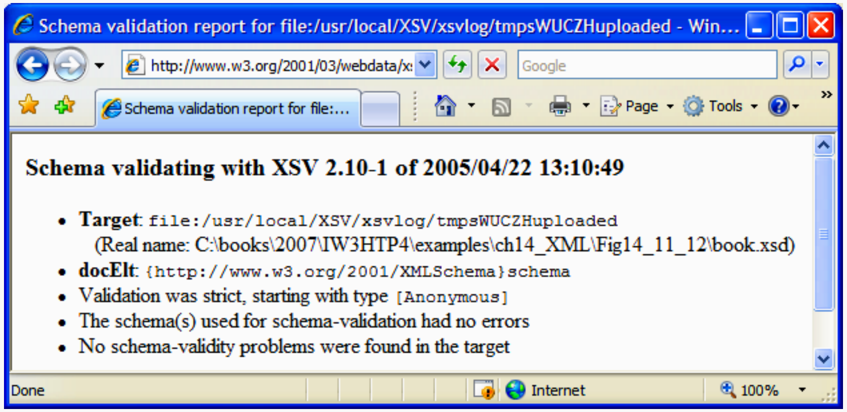
Specify the namespace of the elements that this schema defines

Define the **books** element

Define the requirements for any element of type **BooksType**

An element of type **BooksType** must contain one or more book elements, which have type **SingleBookType**

A **SingleBookType** element has a **title** element, which contains a **string**


Schema validation report for file:/usr/local/XSV/xsvlog/tmpsWUCZHuploaded - Win...
http://www.w3.org/2001/03/webdata/x:
Google
Schema validation report for file:...
Page
Tools
Done
Internet
100%

Schema validating with XSV 2.10-1 of 2005/04/22 13:10:49

- **Target**: `file:/usr/local/XSV/xsvlog/tmpsWUCZHuploaded`
  (Real name: C:\books\2007\IW3HTP4\examples\ch14_XML\Fig14_11_12\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

Outline

computer.xsd

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig. 14.14: 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">
            <attribute name = "model" type = "string"/>
         </extension>
      </simpleContent>
   </complexType>

```

Define a **`simpleType`** that contains a decimal whose value is **`2.1`** or greater

Define a **`complexType`** with simpleContent so that it can contain only attributes, not child elements

The **`CPU`** element's data must be of type **`string`**, and it must have an attribute **`model`** containing a **`string`**

Outline

```
   <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>
```

computer.xsd

(2 of 2)

The all element specifies a list of elements that must be included, in any order, in the document

The types defined in the last slide are used by these elements

Outline

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

<!-- Fig. 14.15: 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>
```

`laptop.xml`

Example Schema

```
<?xml version="1.0"encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="book“ type="BookType"/>
        <xsd:complexType name="BookType">
                <xsd:sequence>
                        <xsd:element name="title“ type="xsd:string"/>
                        <xsd:element name="author“ type="PersonType“ minOccurs="1“
maxOccurs="unbounded"/>
                        <xsd:element name="publisher“ type="xsd:anyType"/>
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="PersonType">
                <xsd:sequence>
                        <xsd:element name="first“ type="xsd:string"/>
                        <xsd:element name="last“ type="xsd:string"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

Example Schema

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

- Vocabulary of Schema defined in special Namespace. Prefix "xsd" is commonly used
- ” Schema” Element is always the Root

<xsd:element name="book" type="BookType"/>

- ”name” attribute defines the name of the element.
- ”type” defines the type of the element
- Declarations under ”schema” are global, "book" is the only global element, root element of a valid document must be a "book".
  - The type of a "book" is BookType.

<xsd:complexType name="BookType">

- User-defined type
- Defines a sequence of sub-elements
- Attribute "name" specifies name of Type
- This Type definition is **global, BookType** can be used in any other definition.

Example Schema

- **<xsd:element name="title“ type="xsd:string"/>**
- Local element declaration within a complex type (” title” cannot be root element of documents)

- ” xsd:string” is built-in type of XML Schema

- ” minOccurs", ” maxOccurs” specify cardinality of "author” Elements in BookType.
  Default: minOccurs=1, maxOccurs=1

- Every book has exactly one “publisher”
  minOccurs, maxOccurs by default 1
- ” anyType” is built-in Type , allows any content

- Attributes may only have a SimpleType
  – Default values (default= "”)
  – Required and optional attributes (use= ” required” , use= ” optional”)
  – Fixed attributes (fixed= ” EUR”)

Pre-defined SimpleTypes

- Numeric Values

  Integer, Short, Decimal, Float, Double, HexBinary, ...
- Date, Timestamps, Periods

  Duration, DateTime, Time, Date, gMonth, ...
- Strings

  String, NMTOKEN, NMTOKENS, NormalizedString
- Others

  AnyURI, ID, IDREFS, Language, Entity, ...

- Restrict domain
  - minInclusive, maxInclusive are "Facets"

14.8 Extensible Stylehsheet Language and XSL Transformations

- Convert XML into any text-based document
- XSL documents have the extension `.xsl`
- XPath
  - A string-based language of expressions used by XML and many of its related technologies for effectively and efficiently locating structures and data (such as specific elements and attributes) in XML documents
  - Used to locate parts of the source-tree document that match templates defined in an XSL style sheet. When a match occurs (i.e., a node matches a template), the matching template executes and adds its result to the result tree. When there are no more matches, XSLT has transformed the source tree into the result tree.
- XSLT does not analyze every node of the source tree
  - it selectively navigates the source tree using XPath's `select` and `match` attributes
- For XSLT to function, the source tree must be properly structured
  - Schemas, DTDs and validating parsers can validate document structure before using XPath and XSLTs
- XSL style sheets can be connected directly to an XML document by adding an `xml:stylesheet` processing instruction to the XML document

14.8 Extensible Stylehsheet Language and XSL Transformations (Cont.)

- Two tree structures are involved in transforming an XML document using XSLT
  - source tree (the document being transformed)
  - result tree (the result of the transformation)
- XPath character `/` (a forward slash)
  - Selects the document root
  - In XPath, a leading forward slash specifies that we are using absolute addressing
  - An XPath expression with no beginning forward slash uses relative addressing
- XSL element `value-of`
  - Retrieves an attribute’s value
  - The `@` symbol specifies an attribute node
- XSL node-set function `name`
  - Retrieves the current node’s element name
- XSL node-set function `text`
  - Retrieves the text between an element’s start and end tags
- The XPath expression `//*`
  - Selects all the nodes in an XML document

Outline

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

<!-- Fig. 14.20: 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>

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

```

The **`xml-stylesheet`** declaration points to an XSL style sheet for this document

`sports.xml`

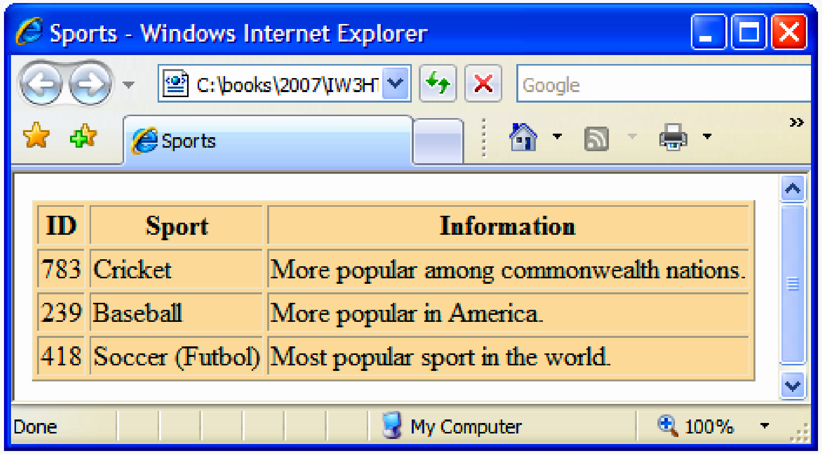
(1 of 2)

Outline

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

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

sports.xml

(2 of 2)



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.

`sports.xsl`

(1 of 2)

```
<?xml version = "1.0"?>
<!-- Fig. 14.21: 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 = "html" 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>

      <body>
         <table border = "1" bgcolor = "wheat">
            <thead>
               <tr>
                  <th>ID</th>
                  <th>Sport</th>
                  <th>Information</th>
               </tr>
            </thead>

```

Use **`xsl-output`** to write a doctype.

```
            <!-- 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>
```

Write the following HTML for each **game** element in the **sports** element that is contained in the root element

sports.xsl

(2 of 2)

Write the value of the **game**’s **id** attribute in a table cell

Write the value of the **game**’s **name** child element in a table cell

Write the value of the **game**’s **paragraph** child element in a table cell

14.9 Document Object Model

- Retrieving data from an XML document using traditional sequential file processing techniques is neither practical nor efficient

- Some XML parsers store document data as tree structures in memory
  - This hierarchical tree structure is called a Document Object Model (DOM) tree, and an XML parser that creates this type of structure is known as a DOM parser
  - Each element name is represented by a node
  - A node that contains other nodes is called a parent node
  - A parent node can have many children, but a child node can have only one parent node
  - Nodes that are peers are called sibling nodes
  - A node’s descendant nodes include its children, its children’s children and so on
  - A node’s ancestor nodes include its parent, its parent’s parent and so on

14.9 Document Object Model (Cont.)

- Many of the XML DOM capabilities are similar or identical to those of the XHTML DOM
- The DOM tree has a single root node, which contains all the other nodes in the document
- `window.ActiveXObject`
  - If this object exists, the browser is Internet Explorer
  - Loads Microsoft's MSXML parser is used to manipulate XML documents in Internet Explorer
- MSXML `load` method
  - loads an XML document
- `childNodes` property of a document
  - contains a list of the XML document's top-level nodes
- If the browser is Firefox 2, then the `document` object's `implementation` property and `createDocument` method will exist
- Firefox loads each XML document asynchronously
  - You must use the XML document's `onload` property to specify a function to call when the document finishes loading to ensure that you can access the document's contents
- `nodeType` property of a node
  - contains the type of the node

14.9 Document Object Model (Cont.)

- `nodeName` property of a node
  - Obtain the name of an element
- `childNodes` list of a node
  - Nonzero if the currrent node has children
- `nodeValue` property
  - Returns the value of an element
- `firstChild` property of a node
  - Refers to the first child of a given node
- `lastChild` property of a node
  - refers to the last child of a given node
- `nextSibling` property of a node
  - refers to the next sibling in a list of children of a particular node.
- `previousSibling` property of a node
  - refers to the current node’s previous sibling
- `parentNode` property of a node
  - refers to the current node’s parent node

14.9 Document Object Model (Cont.)

- **Use XPath expressions to specify search criteria**
  - **In IE7, the XML document object's `selectNodes` method receives an XPath expression as an argument and returns a collection of elements that match the expression**
  - **Firefox 2 searches for XPath matches using the XML document object's `evaluate` method, which receives five arguments**
    - the XPath expression
    - the document to apply the expression to
    - a namespace resolver
    - a result type
    - an `XPathResult` object into which to place the results
    - If the last argument is `null`, the function simply returns a new `XPathResult` object containing the matches
    - The namespace resolver argument can be `null` if you are not using XML namespace prefixes in the XPath processing

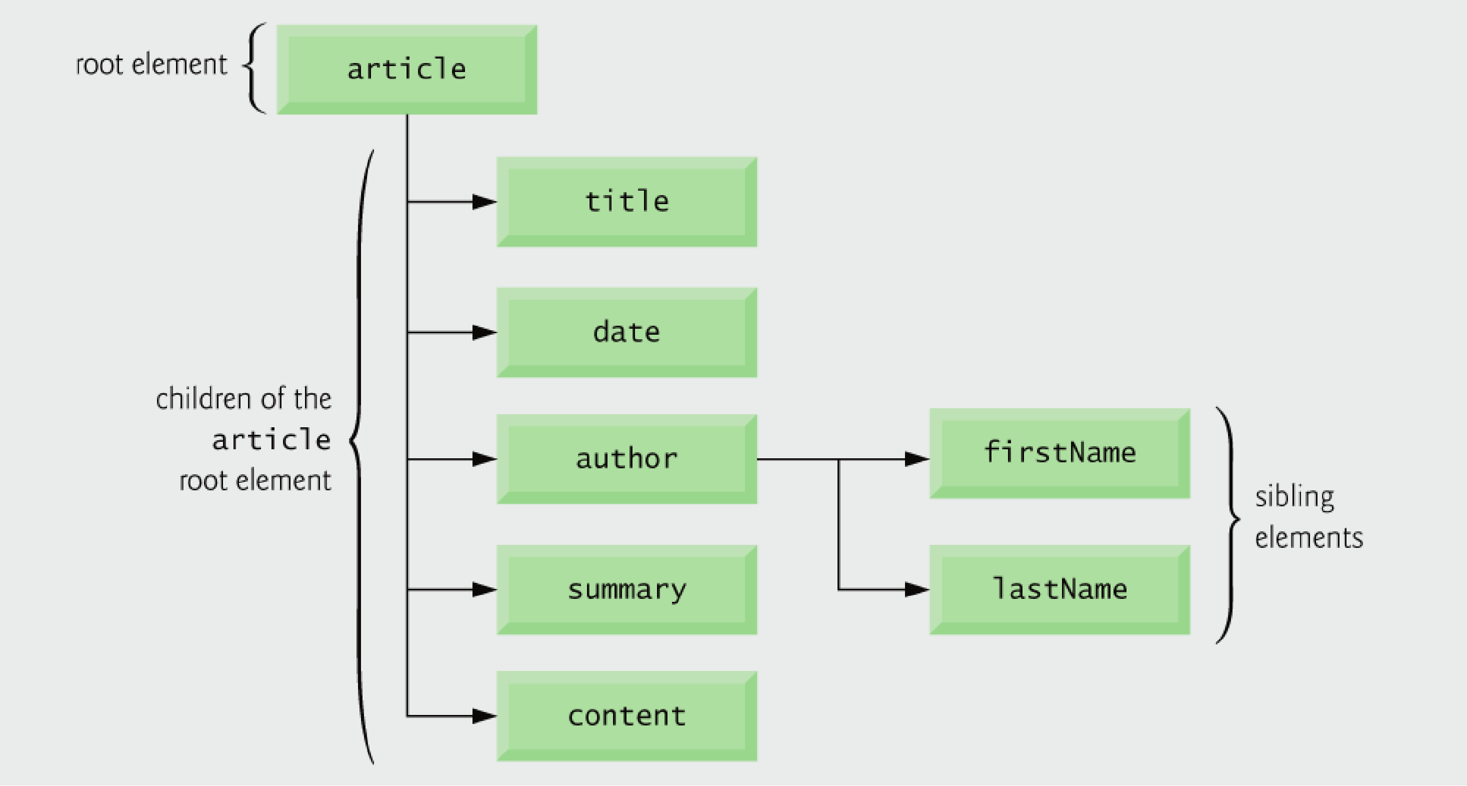


Fig. 14.25 | Tree structure for the document **`article.xml`** of Fig. 14.2.

Property/Method	Description
`nodeType`	An integer representing the node type.
`nodeName`	The name of the node.
`nodeValue`	A string or null depending on the node type.
`parentNode`	The parent node.
`childNodes`	A `NodeList` (Fig. 14.28) with all the children of the node.
`firstChild`	The first child in the `Node`’s `NodeList`.
`lastChild`	The last child in the `Node`’s `NodeList`.
`previousSibling`	The node preceding this node; `null` if there is no such node.
`nextSibling`	The node following this node; `null` if there is no such node.
`attributes`	A collection of `Attr` objects (Fig. 14.31) containing the attributes for this node.

Fig. 14.27 | Common **`Node`** properties and methods. (Part 1 of 2.)

Property/Method	Description
`insertBefore`	Inserts the node (passed as the first argument) before the existing node (passed as the second argument). If the new node is already in the tree, it is removed before insertion. The same behavior is true for other methods that add nodes.
`replaceChild`	Replaces the second argument node with the first argument node.
`removeChild`	Removes the child node passed to it.
`appendChild`	Appends the node it receives to the list of child nodes.

Fig. 14.27 | Common **`Node`** properties and methods. (Part 2 of 2.)

Property/Method	Description
`item`	Method that receives an index number and returns the element node at that index. Indices range from 0 to *length* – 1. You can also access the nodes in a `NodeList` via array indexing.
`length`	The total number of nodes in the list.

Fig. 14.28 | `NodeList` property and method.

Property/Method	Description
documentElement	The root node of the document.
createElement	Creates and returns an element node with the specified tag name.
createAttribute	Creates and returns an Attr node (Fig. 14.31) with the specified name and value.
createTextNode	Creates and returns a text node that contains the specified text.
getElementsBy TagName	Returns a NodeList of all the nodes in the subtree with the name specified as the first argument, ordered as they would be encountered in a preorder traversal. An optional second argument specifies either the direct child nodes (0) or any descendant (1).

Fig. 14.29 | **Document** properties and methods.

Property/Method	Description
`tagName`	The name of the element.
`getAttribute`	Returns the value of the specified attribute.
`setAttribute`	Changes the value of the attribute passed as the first argument to the value passed as the second argument.
`removeAttribute`	Removes the specified attribute.
`getAttributeNode`	Returns the specified attribute node.
`setAttributeNode`	Adds a new attribute node with the specified name.

Fig. 14.30 | **`Element`** property and methods.

Property	Description
`value`	The specified attribute’s value.
`name`	The name of the attribute.

Fig. 14.31 | `Attr` properties.

Property	Description
`data`	The text contained in the node.
`length`	The number of characters contained in the node.

Fig. 14.32 | `Text` methods.

Outline

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 14.33: xpath.html -->
<!-- Using XPath to locate nodes in an XML document. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <title>Using XPath to Locate Nodes in an XML Document</title>
   <style type = "text/css">
      #outputDiv { font: 10pt "Lucida Console", monospace; }
   </style>
   <script type = "text/javascript">
   <!--
   var doc; // variable to reference the XML document
   var outputHTML = ""; // stores text to output in outputDiv
   var browser = ""; // used to determine which browser is being used

   // load XML document based on whether the browser is IE7 or Firefox 2
   function loadXMLDocument( url )
   {
      if ( window.ActiveXObject ) // IE7
      {
         // create IE7-specific XML document object
         doc = new ActiveXObject( "Msxml2.DOMDocument.6.0" );
         doc.async = false; // specifies synchronous loading of XML doc
         doc.load( url ); // load the XML document specified by url
         browser = "IE7"; // set browser
      } // end if
```

xpath.html

(1 of 5)

Outline

```
      else if ( document.implementation &&
         document.implementation.createDocument ) // other browsers
      {
         // create XML document object
         doc = document.implementation.createDocument( "", "", null );
         doc.load( url ); // load the XML document specified by url
         browser = "FF2"; // set browser
      } // end else
      else // not supported
         alert( 'This script is not supported by your browser' );
   } // end function loadXMLDocument

   // display the XML document
   function displayDoc()
   {
      document.getElementById( "outputDiv" ).innerHTML = outputHTML;
   } // end function displayDoc

   // obtain and apply XPath expression
   function processXPathExpression()
   {
      var xpathExpression = document.getElementById( "inputField" ).value;
      outputHTML = "";

```

xpath.html

(2 of 5)

Outline

xpath.html

(3 of 5)

```
      if ( browser == "IE7" )
      {
         var result = doc.selectNodes( xpathExpression );

         for ( var i = 0; i < result.length; i++ )
            outputHTML += "<div style='clear: both'>" +
               result.item( i ).text + "</div>";
      } // end if
      else // browser == "FF2"
      {
         var result = document.evaluate( xpathExpression, doc, null,
            XPathResult.ANY_TYPE, null );
         var current = result.iterateNext();

         while ( current )
         {
            outputHTML += "<div style='clear: both'>" +
               current.textContent + "</div>";
            current = result.iterateNext();
         } // end while
      } // end else

      displayDoc();
   } // end function processXPathExpression
   // -->
   </script>
</head>
```

IE7 uses the **`document`** object's **`selectNodes`** method to select nodes using an XPath.

Other browsers use the **`document`** object's **`evaluate`** method.

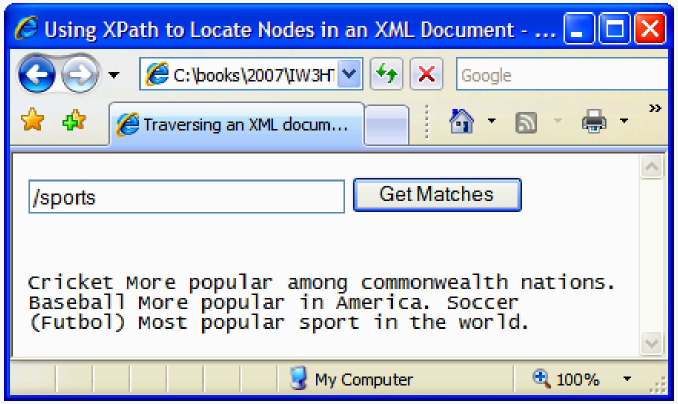
Outline

```
<body id = "body" onload = "loadXMLDocument( 'sports.xml' );">
   <form action = "" onsubmit = "return false;">
      <input id = "inputField" type = "text" style = "width: 200px"/>
      <input type = "submit" value = "Get Matches"
         onclick = "processXPathExpression()"/>
   </form><br/>
   <div id = "outputDiv"></div>
</body>
</html>
```

xpath.html

(4 of 5)

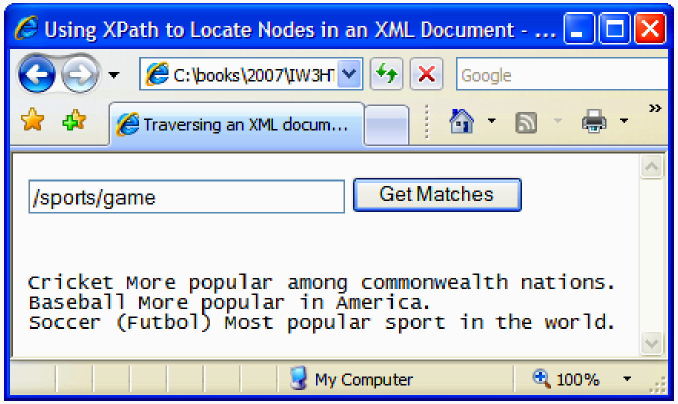
a)


Using XPath to Locate Nodes in an XML Document - ...
C:\books\2007\IW3H
Google
Traversing an XML docum...
Get Matches
My Computer
100%

/sports

```
Cricket More popular among commonwealth nations.
Baseball More popular in America. Soccer
(Futbol) Most popular sport in the world.
```

b)


Using XPath to Locate Nodes in an XML Document - ...
C:\books\2007\IW3H
Google
Traversing an XML docum...
Get Matches
My Computer
100%

/sports/game

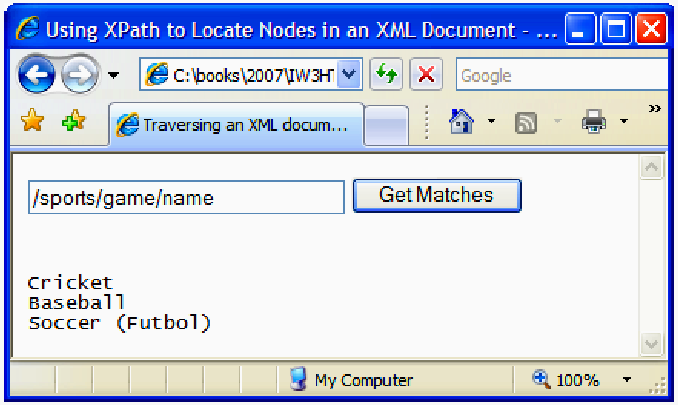
```
Cricket More popular among commonwealth nations.
Baseball More popular in America.
Soccer (Futbol) Most popular sport in the world.
```

Outline

`xpath.html`

(5 of 5)

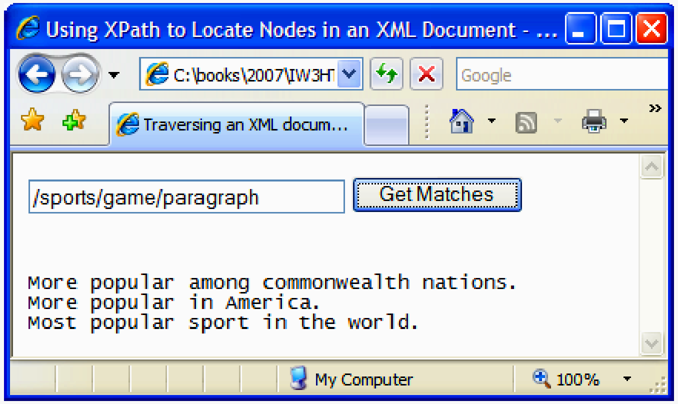
c)


Using XPath to Locate Nodes in an XML Document - ...
C:\books\2007\IW3HT
Google
Traversing an XML docum...
Get Matches
My Computer
100%

/sports/game/name

```
Cricket
Baseball
Soccer (Futbol)
```

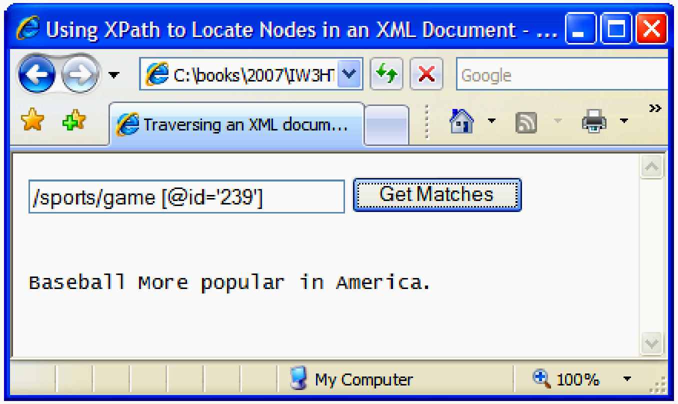
d)


Using XPath to Locate Nodes in an XML Document - ...
C:\books\2007\IW3HT
Google
Traversing an XML docum...
Get Matches
My Computer
100%

/sports/game/paragraph

```
More popular among commonwealth nations.
More popular in America.
Most popular sport in the world.
```

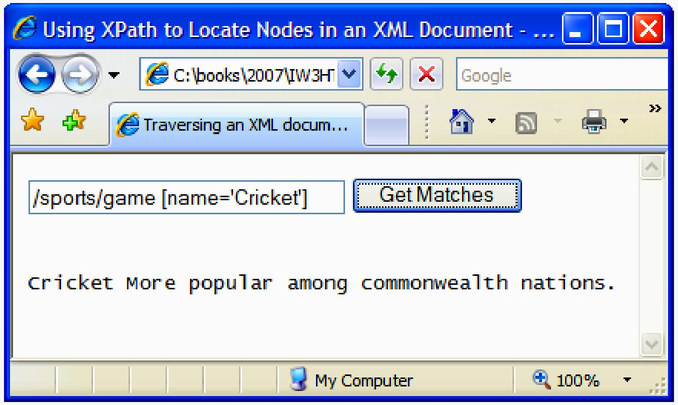
e)


Using XPath to Locate Nodes in an XML Document - ...
C:\books\2007\IW3HT
Google
Traversing an XML docum...
Get Matches
My Computer
100%

/sports/game [@id='239']

```
Baseball More popular in America.
```

f)


Using XPath to Locate Nodes in an XML Document - ...
C:\books\2007\IW3HT
Google
Traversing an XML docum...
Get Matches
My Computer
100%

/sports/game [name='Cricket']

```
Cricket More popular among commonwealth nations.
```

Outline

sports.xml

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

<!-- Fig. 14.34: 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>
      <paragraph>
         More popular in America.
      </paragraph>
   </game>
   <game id = "418">
      <name>Soccer (Futbol)</name>
      <paragraph>
         Most popular sport in the world.
      </paragraph>
   </game>
</sports>
```

Expression	Description
`/sports`	Matches all `sports` nodes that are child nodes of the document root node.
`/sports/game`	Matches all `game` nodes that are child nodes of `sports`, which is a child of the document root.
`/sports/game/name`	Matches all `name` nodes that are child nodes of `game`. The `game` is a child of `sports`, which is a child of the document root.
`/sports/game/paragraph`	Matches all `paragraph` nodes that are child nodes of `game`. The `game` is a child of `sports`, which is a child of the document root.
`/sports/game [@id='239']`	Matches the `game` node with the `id` number 239. The `game` is a child of `sports`, which is a child of the document root.
`/sports/game [name='Cricket']`	Matches all `game` nodes that contain a child element whose name is `Cricket`. The `game` is a child of `sports`, which is a child of the document root.

Fig. 14.35 | XPath expressions and descriptions.