WEB TECHNOLOGIES

UNIT - III

XML

XML

1. Introduction

- The **Extensible Markup Language (XML)** was developed in 1996 by the **World Wide Web Consortium's (W3C's)** XML Working Group.
- XML is a widely supported **open technology** (i.e., nonproprietary technology) for describing data that has become the standard format for data exchanged between applications over the Internet.
- Web applications use XML extensively, and web browsers provide many XML-related capabilities.

XML

2. XML Basics:

- XML permits document authors to create **markup** (i.e., a text-based notation for describing data) for virtually any type of information, enabling them to create entirely new markup languages for describing any type of data, such as mathematical formulas, software configuration instructions, chemical molecular structures, music, news, recipes and financial reports.
- XML describes data in a way that human beings can understand and computers can process.
- A simple XML document that describes information for a baseball player is given here:

XML

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.1: player.xml -->
4  <!-- Baseball player structured with XML -->
5  <player>
6     <firstName>John</firstName>
7     <lastName>Doe</lastName>
8     <battingAverage>0.375</battingAverage>
9  </player>
```

Fig. 15.1 | XML that describes a baseball player's information.

XML

XML Elements:

- XML documents contain text that represents content (i.e., data), such as John (line 6 of Fig. 15.1), and **elements** that specify the document's structure, such as firstName (line 6 of Fig. 15.1).
- XML documents delimit elements with **start tags** and **end tags**. A start tag consists of the element name in **angle brackets** (e.g., <player> and <firstName> in lines 5 and 6, respectively). An end tag consists of the element name preceded by a **forward slash (/)** in angle brackets (e.g., </firstName> and </player> in lines 6 and 9, respectively).
- An element's start and end tags enclose text that represents a piece of data (e.g., the player's firstName—John—in line 6, which is enclosed by the <firstName> start tag and </firstName> end tag).
- Every XML document must have exactly one **root element** that contains all the other elements. In Fig. 15.1, the root element is player (lines 5–9).

XML Vocabularies:

- XML-based markup languages—called XML **vocabularies**—provide a means for describing particular types of data in standardized, structured ways. Some XML vocabularies include XHTML (Extensible HyperText Markup Language), MathML™(for mathematics), VoiceXML™ (for speech), CML (Chemical Markup Language—for chemistry), XBRL (Extensible Business Reporting Language—for financial data exchange) and others.
- Massive amounts of data are currently stored on the Internet in many formats (e.g., databases, web pages, text files). Much of this data, especially that which is passed between systems, will soon take the form of XML.
- The next generation of the web is being built on an XML foundation, enabling you to develop more sophisticated web-based applications.
- XML allows to assign meaning to what would otherwise be random pieces of data. As a result, programs can "understand" the data they manipulate.
- For example, a web browser might view a street address in a simple web page as a string of characters without any real meaning. In an XML document, however, this data can be clearly identified (i.e., marked up) as an address.
- A program that uses the document can recognize this data as an address and provide links to a map of that location, driving directions from that location or other location-specific information.
- Likewise, an application can recognize names of people, dates, ISBN numbers and any other type of XML-encoded data. The application can then present users with other related information, providing a richer, more meaningful user experience.

XML

Viewing and Modifying XML Documents:

- XML documents are highly portable. Viewing or modifying an XML document—which is a text file that usually ends with the **.xml** filename extension—does not require special software, although many software tools exist, and new ones are frequently released that make it more convenient to develop XML-based applications.
- Any text editor that supports ASCII/Unicode characters can open XML documents for viewing and editing.
- Also, most web browsers can display XML documents in a formatted manner that shows the XML's structure.
- An important characteristic of XML is that it's both human and machine readable.

XML

Processing XML Documents:

- Processing an XML document requires software called an **XML parser** (or **XML processor**). A parser makes the document's data available to applications.
- While reading an XML document's contents, a parser checks that the document follows the syntax rules specified by the W3C's XML Recommendation (www.w3.org/XML).
- XML syntax requires a single root element, a start tag and end tag for each element, and properly nested tags (i.e., the end tag for a nested element must appear before the end tag of the enclosing element). XML is case sensitive, so the proper capitalization must be used in elements.
- A document that conforms to this syntax is a **well-formed XML document** and is syntactically correct. If an XML parser can process an XML document successfully, that XML document is well-formed.
- Parsers can provide access to XML-encoded data in well-formed documents only. XML parsers are often built into browsers and other software.

Validating XML Documents:

- An XML document can reference a **Document Type Definition (DTD)** or a **schema** that defines the document's proper structure.
- When an XML document references a DTD or a schema, some parsers (called **validating parsers**) can read it and check that the XML document follows the structure it defines.
- If the XML document conforms to the DTD/schema (i.e., has the appropriate structure), the document is **valid**. For example, if in Fig. 15.1 we were referencing a DTD that specified that a player element must have firstName, lastName and battingAverage elements, then omitting the lastName element (line 7 in Fig. 15.1) would invalidate the XML document player.xml.
- However, it would still be well-formed, because it follows proper XML syntax (i.e., it has one root element, each element has a start tag and an end tag, and the elements are nested properly).
- By definition, a valid XML document is well-formed. Parsers that cannot check for document conformity against DTDs/schemas are **non-validating parsers**—they determine only whether an XML document is well-formed, not whether it's valid.

XML

Formatting and Manipulating XML Documents:

- Most XML documents contain only data, so applications that process XML documents must decide how to manipulate or display the data.
- For example, a PDA (personal digital assistant) may render an XML document differently than a wireless phone or a desktop computer.
- Use **Extensible Stylesheet Language (XSL)** to specify rendering instructions for different platforms.
- XML-processing programs can also search, sort and manipulate XML data using XSL.
- Some other XML-related technologies are XPath (XML Path Language—a language for accessing parts of an XML document), XSL-FO (XSL Formatting Objects—an XML vocabulary used to describe document formatting) and XSLT (XSL Transformations—a language for transforming XML documents into other documents).

XML

3.Structuring Data:

- XML allows to describe data precisely in a well-structured format.

XML Markup for an Article

- In Fig. 15.2, an XML document that marks up a simple article using XML is given. The line numbers shown are for reference only and are not part of the XML document.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.2: article.xml -->
4  <!-- Article structured with XML -->
5  <article>
6     <title>Simple XML</title>
7     <date>July 4, 2007</date>
8     <author>
9        <firstName>John</firstName>
10       <lastName>Doe</lastName>
11    </author>
12    <summary>XML is pretty easy.</summary>
13    <content>This chapter presents examples that use XML.</content>
14 </article>
```

Fig. 15.2 | XML used to mark up an article.

XML

XML Declaration:

- This document begins with an **XML declaration** (line 1), which identifies the document as an XML document. The **version attribute** specifies the XML version to which the document conforms. The W3C may continue to release new versions as XML evolves to meet the requirements of different fields.

Blank Space and Comments:

- Blank lines are normally ignored by XML parsers. XML comments (lines 3–4), which begin with <!-- and end with -->, can be placed almost anywhere in an XML document and can span multiple lines.
- There must be one end marker (-->) for each begin marker (<!--).
- In Fig. 15.2, article (lines 5–14) is the root element. The lines that precede the root element (lines 1–4) are the XML **prolog**. In an XML prolog, the XML declaration must appear before the comments and any other markup.

XML Element Names:

- The elements used in the example do not come from any specific markup language. Invent elements to mark up your data. For example, element title (line 6) contains text that describes the article's title (e.g., Simple XML).
- Similarly, date (line 7), author (lines 8–11), firstName (line 9), lastName (line 10), summary (line 12) and content (line 13) contain text that describes the date, author, the author's first name, the author's last name, a summary and the content of the document, respectively.
- XML element names can be of any length and may contain letters, digits, underscores, hyphens and periods. However, they must begin with either a letter or an underscore, and they should not begin with "xml" in any combination of uppercase and lowercase letters (e.g., XML, Xml, xMl), as this is reserved for use in the XML standards.

XML

Nesting XML Elements:

- XML elements are **nested** to form hierarchies—with the root element at the top of the hierarchy. This allows document authors to create parent/child relationships between data items.
- For example, elements title, date, author, summary and content are children of article. Elements firstName and lastName are children of author. Any element that contains other elements (e.g., article or author) is a **container element**.
- Container elements also are called **parent elements**. Elements nested inside a container element are **child elements** (or children) of that container element. If those child elements are at the same nesting level, they're **siblings** of one another.

Viewing an XML Document in a Web Browser:

- The XML document in Fig. 15.2 is simply a text file named article.xml. It does not contain formatting information for the article. This is because XML is a technology for describing the structure of data.
- The formatting and displaying of data from an XML document are application-specific issues.
- For example, when the user loads article.xml in a web browser, the browser parses and displays the document's data. Each browser has a built-in **style sheet** to format the data. The resulting format of the data is similar to the format of the listing in Fig. 15.2.
- The down arrow ( ) and right arrow ( ) in the screen shots are not part of the XML document.
- Google Chrome places them next to every container element. A down arrow indicates that the browser is displaying the container element's child elements.
- Clicking the down arrow next to an element collapses that element (i.e., causes the browser to hide the container element's children and replace the down arrow with a right arrow).
- Conversely, clicking the right arrow next to an element expands that element (i.e., causes the browser to display the container element's children and replace the right arrow with a down arrow). Parsers often store XML data as tree structures to facilitate efficient manipulation.

XML

XML Markup for a Business Letter:

- Now examine a more complex one that marks up a business letter (Fig. 15.4). Begin the document with the XML declaration (line 1) that states the XML version to which the document conforms.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.4: letter.xml -->
4  <!-- Business letter marked up with XML -->
5  <!DOCTYPE letter SYSTEM "letter.dtd">
6
7  <letter>
8     <contact type = "sender">
9        <name>Jane Doe</name>
10       <address1>Box 12345</address1>
11       <address2>15 Any Ave.</address2>
12       <city>Othertown</city>
13       <state>Otherstate</state>
14       <zip>67890</zip>
15       <phone>555-4321</phone>
16       <flag gender = "F" />
17    </contact>
```

Fig. 15.4 | Business letter marked up with XML. (Part 1 of 2.)

XML

```
18
19      <contact type = "receiver">
20         <name>John Doe</name>
21         <address1>123 Main St.</address1>
22         <address2></address2>
23         <city>Anytown</city>
24         <state>Anystate</state>
25         <zip>12345</zip>
26         <phone>555-1234</phone>
27         <flag gender = "M" />
28      </contact>
29
30      <salutation>Dear Sir:</salutation>
31
32      <paragraph>It is our privilege to inform you about our new database
33         managed with XML. This new system allows you to reduce the
34         load on your inventory list server by having the client machine
35         perform the work of sorting and filtering the data.
36      </paragraph>
37
38      <paragraph>Please visit our website for availability and pricing.
39      </paragraph>
40
41      <closing>Sincerely,</closing>
42      <signature>Ms. Jane Doe</signature>
43   </letter>
```

Fig. 15.4 | Business letter marked up with XML. (Part 2 of 2.)

XML

- Line 5 specifies that this XML document references a DTD. DTDs define the structure of the data for an XML document.
- For example, a DTD specifies the elements and parent/child relationships between elements permitted in an XML document.

DOCTYPE:

- The DOCTYPE reference (line 5) contains three items: the name of the root element that the DTD specifies (letter); the keyword **SYSTEM** (which denotes an **external DTD**—a DTD declared in a separate file, as opposed to a DTD declared locally in the same file); and the DTD's name and location (i.e., letter.dtd in the current directory; this could also be a fully qualified URL). DTD document filenames end with the **.dtd** extension.

Validating an XML Document Against a DTD

- Many online tools can validate your XML documents against DTDs or schemas. The validator at http://www.xmlvalidation.com/ can validate XML documents against either DTDs or schemas.
- To use it, either paste your XML document's code into a text area on the page or upload the file. If the XML document references a DTD, the site asks you to paste in the DTD or upload the DTD file. Also select a checkbox for validating against a schema instead. Then click a button to validate your XML document.

XML

The XML Document's Contents:

- Root element letter (lines 7–43 of Fig. 15.4) contains the child elements contact, contact, salutation, paragraph, paragraph, closing and signature.
- Data can be placed between an element's tags or as **attributes**—name/value pairs that appear within the angle brackets of an element's start tag. Elements can have any number of attributes (separated by spaces) in their start tags.
- The first contact element (lines 8–17) has an attribute named type with **attribute value** "sender", which indicates that this contact element identifies the letter's sender.
- The second contact element (lines 19–28) has attribute type with value "receiver", which indicates that this contact element identifies the recipient of the letter. Like element names, attribute names are case sensitive, can be any length, may contain letters, digits, underscores, hyphens and periods, and must begin with either a letter or an underscore character.
- A contact element stores various items of information about a contact, such as the contact's name (represented by element name), address (represented by elements address1, address2, city, state and zip), phone number (represented by element phone) and gender (represented by attribute gender of element flag).
- Element salutation (line 30) marks up the letter's salutation. Lines 32–39 mark up the letter's body using two paragraph elements. Elements closing (line 41) and signature (line 42) mark up the closing sentence and the author's "signature," respectively.
- Line 16 introduces the **empty element** flag. An empty element is one that does not have any content. Instead, it sometimes places data in attributes.
- Empty element flag has one attribute that indicates the gender of the contact (represented by the parent contact element). Document authors can close an empty element either by placing a slash immediately preceding the right angle bracket, as shown in line 16, or by explicitly writing an end tag, as in line 22:**<address2></address2>**
- The address2 element in line 22 is empty because there's no second part to this contact's address. However, we must include this element to conform to the structural rules specified in the XML document's DTD—letter.dtd. This DTD specifies that each contact element must have an address2 child element.

XML

4. XML Namespaces:

- XML allows document authors to create custom elements. This extensibility can result in **naming collisions** among elements in an XML document that have the same name.
- For example, we may use the element book to mark up data about a Deitel publication. A stamp collector may use the element book to mark up data about a book of stamps. Using both of these elements in the same document could create a naming collision, making it difficult to determine which kind of data each element contains.
- An XML **namespace** is a collection of element and attribute names. XML namespaces provide a means for document authors to unambiguously refer to elements with the same name (i.e., prevent collisions). For example, **<subject>**Geometry**</subject>** and **<subject>**Cardiology**</subject>** use element subject to mark up data. In the first case, the subject is something one studies in school, whereas in the second case, the subject is a field of medicine.
- Namespaces can differentiate these two subject elements—for example: **<highschool:subject>**Geometry**</highschool:subject>** and **<medicalschool:subject>**Cardiology**</medicalschool:subject>**
- Both highschool and medicalschool are **namespace prefixes**. A document author places a namespace prefix and colon (:) before an element name to specify the namespace to which that element belongs. Document authors can create their own namespace prefixes using virtually any name except the reserved namespace prefix xml.

Differentiating Elements with Namespaces

- Figure 15.5 demonstrates namespaces. In this document, namespaces differentiate two distinct elements—the file element related to a text file and the file document related to an image file.

XML

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.5: namespace.xml -->
4  <!-- Demonstrating namespaces -->
5  <text:directory
6     xmlns:text = "urn:deitel:textInfo"
7     xmlns:image = "urn:deitel:imageInfo">
```

Fig. 15.5 | XML namespaces demonstration. (Part 1 of 2.)

```
8
9     <text:file filename = "book.xml">
10       <text:description>A book list</text:description>
11    </text:file>
12
13    <image:file filename = "funny.jpg">
14       <image:description>A funny picture</image:description>
15       <image:size width = "200" height = "100" />
16    </image:file>
17 </text:directory>
```

Fig. 15.5 | XML namespaces demonstration. (Part 2 of 2.)

XML

The xmlns Attribute:

- Lines 6–7 use the XML-namespace reserved attribute **xmlns** to create two namespace prefixes—text and image.
- Each namespace prefix is bound to a series of characters called a **Uniform Resource Identifier (URI)** that uniquely identifies the namespace.
- Document authors create their own namespace prefixes and URIs. A URI is a way to identifying a resource, typically on the Internet.
- Two popular types of URI are **Uniform Resource Name (URN)** and **Uniform Resource Locator (URL)**.

Unique URIs:

- To ensure that namespaces are unique, document authors must provide unique URIs.
- In this example, urn:deitel:textInfo and urn:deitel:imageInfo as URIs are used. These URIs employ the URN scheme that is often used to identify namespaces.
- Under this naming scheme, a URI begins with "urn:", followed by a unique series of additional names separated by colons.
- Another common practice is to use URLs, which specify the location of a file or a resource on the Internet.
- For example, www.deitel.com is the URL that identifies the home page of the Deitel & Associates website. Using URLs guarantees that the namespaces are unique because the domain names (e.g., www.deitel.com) are guaranteed to be unique. For example, lines 5–7 could be rewritten as

XML

<text:directory

xmlns:text = "http://www.deitel.com/xmlns-text"

xmlns:image = "http://www.deitel.com/xmlns-image">

where URLs related to the deitel.com domain name serve as URIs to identify the text and image namespaces.

- The parser does not visit these URLs, nor do these URLs need to refer to actual web pages.
- They each simply represent a unique series of characters used to differentiate URI names.
- In fact, any string can represent a namespace.
- For example, our image namespace URI could be hgjfkdlsa4556, in which case our prefix assignment would be **xmlns:image = "hgjfkdlsa4556"**

XML

Namespace Prefix:

- Lines 9–11 use the text namespace prefix for elements file and description. The end tags must also specify the namespace prefix text.
- Lines 13–16 apply namespace prefix image to the elements file, description and size.
- Attributes do not require namespace prefixes (although they can have them), because each attribute is already part of an element that specifies the namespace prefix.
- For example, attribute filename (line 9) is implicitly part of namespace text because its element (i.e., file) specifies the text namespace prefix.

Specifying a Default Namespace:

- To eliminate the need to place namespace prefixes in each element, document authors may specify a **default namespace** for an element and its children.
- Figure 15.6 demonstrates using a default namespace (urn:deitel:textInfo) for element directory.

XML

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.6: defaultnamespace.xml -->
4  <!-- Using default namespaces -->
5  <directory xmlns = "urn:deitel:textInfo"
6     xmlns:image = "urn:deitel:imageInfo">
7
8     <file filename = "book.xml">
9        <description>A book list</description>
10    </file>
11
12    <image:file filename = "funny.jpg">
13       <image:description>A funny picture</image:description>
14       <image:size width = "200" height = "100" />
15    </image:file>
16 </directory>
```

Fig. 15.6 | Default namespace demonstration.

XML

- Line 5 defines a default namespace using attribute xmlns with no prefix specified but with a URI as its value.
- Child elements belonging to the namespace need not be qualified by a namespace prefix.
- Thus, element file (lines 8–10) is in the default namespace urn:deitel:textInfo.
- Compare this to lines 9–10 of Fig. 15.5, where we had to prefix the file and description element names with the namespace prefix text.
- The default namespace applies to the directory element and all elements that are not qualified with a namespace prefix.
- Use a namespace prefix to specify a different namespace for a particular element.
- For example, the file element in lines 12– 15 of Fig. 15.16 includes the image namespace prefix, indicating that this element is in the urn:deitel:imageInfo namespace, not the default namespace.

Namespaces in XML Vocabularies:

- XML-based languages, such as XML Schema and Extensible Stylesheet Language (XSL), often use namespaces to identify their elements.
- Each vocabulary defines special-purpose elements that are grouped in namespaces. These namespaces help prevent naming collisions between predefined elements and user-defined elements.

XML

5. Document Type Definitions (DTDs):

- Document Type Definitions (DTDs) are one of two main types of documents you can use to specify XML document structure.

Creating a Document Type Definition:

- Figure 15.4 presented a simple business letter marked up with XML. Recall that line 5 of letter.xml references a DTD—letter.dtd (Fig. 15.7).
- This DTD specifies the business letter's element types and attributes and their relationships to one another.
- A DTD describes the structure of an XML document and enables an XML parser to verify whether an XML document is valid (i.e., whether its elements contain the proper attributes and appear in the proper sequence).
- DTDs allow users to check document structure and to exchange data in a standardized format.
- A DTD expresses the set of rules for document structure using an EBNF (Extended Backus-Naur Form) grammar.
- DTDs are not themselves XML documents.

XML

```
1  <!-- Fig. 15.7: letter.dtd        -->
2  <!-- DTD document for letter.xml -->
3
4  <!ELEMENT letter ( contact+, salutation, paragraph+,
5     closing, signature )>
6
7  <!ELEMENT contact ( name, address1, address2, city, state,
8     zip, phone, flag )>
9  <!ATTLIST contact type CDATA #IMPLIED>
10
11 <!ELEMENT name ( #PCDATA )>
12 <!ELEMENT address1 ( #PCDATA )>
13 <!ELEMENT address2 ( #PCDATA )>
14 <!ELEMENT city ( #PCDATA )>
15 <!ELEMENT state ( #PCDATA )>
16 <!ELEMENT zip ( #PCDATA )>
17 <!ELEMENT phone ( #PCDATA )>
18 <!ELEMENT flag EMPTY>
19 <!ATTLIST flag gender (M | F) "M">
20
21 <!ELEMENT salutation ( #PCDATA )>
22 <!ELEMENT closing ( #PCDATA )>
23 <!ELEMENT paragraph ( #PCDATA )>
24 <!ELEMENT signature ( #PCDATA )>
```

Fig. 15.7 | Document Type Definition (DTD) for a business letter.

XML

Defining Elements in a DTD:

- The **ELEMENT element type declaration** in lines 4–5 defines the rules for element letter. In this case, letter contains one or more contact elements, one salutation element, one or more paragraph elements, one closing element and one signature element, in that sequence.
- The **plus sign (+) occurrence indicator** specifies that the DTD requires one or more occurrences of an element.
- Other occurrence indicators include the **asterisk** (*), which indicates an optional element that can occur zero or more times, and the **question mark** (?), which indicates an optional element that can occur at most once (i.e., zero or one occurrence).
- If an element does not have an occurrence indicator, the DTD requires exactly one occurrence.
- The contact element type declaration (lines 7–8) specifies that a contact element contains child elements name, address1, address2, city, state, zip, phone and flag— in that order.
- The DTD requires exactly one occurrence of each of these elements.

Defining Attributes in a DTD:

- Line 9 uses the **ATTLIST attribute-list declaration** to define an attribute named type for the contact element. Keyword **#IMPLIED** specifies that if the parser finds a contact element without a type attribute, the parser can choose an arbitrary value for the attribute or can ignore the attribute.
- Either way the document will still be valid (if the rest of the document is valid)—a missing type attribute will not invalidate the document. Other keywords that can be used in place of #IMPLIED in an ATTLIST declaration include **#REQUIRED** and **#FIXED**.
- Keyword #REQUIRED specifies that the attribute must be present in the element and keyword #FIXED specifies that the attribute (if present) must have the given fixed value.
- For example, **<!ATTLIST** address zip **CDATA #FIXED "01757">** indicates that attribute zip (if present in element address) must have the value 01757 for the document to be valid.
- If the attribute is not present, then the parser, by default, uses the fixed value that the ATTLIST declaration specifies.

XML

Character Data vs. Parsed Character Data:

- Keyword **CDATA** (line 9) specifies that attribute type contains **character data** (i.e., a string). A parser will pass such data to an application without modification.
- Keyword **#PCDATA** (line 11) specifies that an element (e.g., name) may contain **parsed character data** (i.e., data that's processed by an XML parser).
- Elements with parsed character data cannot contain markup characters, such as less than (<), greater than (>) or ampersand (&).
- The document author should replace any markup character in a #PCDATA element with the character's corresponding **character entity reference**.
- For example, the character entity reference < should be used in place of the less-than symbol (<), and the character entity reference > should be used in place of the greater-than symbol (>).
- A document author who wishes to use a literal ampersand should use the entity reference & instead—parsed character data can contain ampersands (&) only for inserting entities.

XML

Defining Empty Elements in a DTD:

- Line 18 defines an empty element named flag. Keyword **EMPTY** specifies that the element does not contain any data between its start and end tags. Empty elements commonly describe data via attributes. For example, flag's data appears in its gender attribute (line 19).
- Line 19 specifies that the gender attribute's value must be one of the enumerated values (M or F) enclosed in parentheses and delimited by a vertical bar (|) meaning "or." Note that line 19 also indicates that gender has a default value of M.

Well-Formed Documents vs. Valid Documents:

- The validation revealed that the XML document letter.xml (Fig. 15.4) is well-formed and valid—it conforms to letter.dtd (Fig. 15.7).
- Recall that a well-formed document is syntactically correct (i.e., each start tag has a corresponding end tag, the document contains only one root element, etc.), and a valid document contains the proper elements with the proper attributes in the proper sequence.
- An XML document cannot be valid unless it's well-formed. When a document fails to conform to a DTD or a schema, an XML validator displays an error message.
- For example, the DTD in Fig. 15.7 indicates that a contact element must contain the child element name. A document that omits this child element is still well-formed but is not valid.

XML

6. W3C XML Schema Documents:

- Schemas are introduced for specifying XML document structure and validating XML documents. Many developers in the XML community believe that DTDs are not flexible enough to meet today's programming needs.
- For example, DTDs lack a way of indicating what specific type of data (e.g., numeric, text) an element can contain, and DTDs are not themselves XML documents, forcing developers to learn multiple grammars and developers to create multiple types of parsers. These and other limitations have led to the development of schemas.
- Unlike DTDs, schemas do not use EBNF grammar. Instead, they 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 (note the capital "S" in "Schema").
- Recall that a DTD describes an XML document's structure, not the content of its elements. For example, **<quantity>**5**</quantity>** contains character data.
- If the document that contains element quantity references a DTD, an XML parser can validate the document to confirm that this element indeed does contain PCDATA content.
- However, the parser cannot validate that the content is numeric; DTDs do not provide this capability. So, unfortunately, the parser also considers **<quantity>**hello**</quantity>** to be valid.
- An application that uses the XML document containing this markup should test that the data in element quantity is numeric and take appropriate action if it's not.
- XML Schema enables schema authors to specify that element quantity's data must be numeric or, even more specifically, an integer.
- A parser validating the XML document against this schema can determine that 5 conforms and hello does not. 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.

XML

Validating Against an XML Schema Document:

- Figure 15.9 shows a schema-valid XML document named book.xml, and Fig. 15.10 shows the pertinent XML Schema document (book.xsd) that defines the structure for book.xml. By convention, schemas use the **.xsd** extension.
- An online XSD schema validator provided at www.xmlforasp.net/SchemaValidator.aspx is used to ensure that the XML document in Fig. 15.9 conforms to the schema in Fig. 15.10.
- To validate the schema document itself (i.e., book.xsd) and produce the output shown in Fig. 15.10, an online XSV (XML Schema Validator) provided by the W3C at www.w3.org/2001/03/webdata/xsv is used.
- These tools are free and enforce the W3C's specifications regarding XML Schemas and schema validation.

XML

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.9: book.xml -->
4  <!-- Book list marked up as XML -->
5  <deitel:books xmlns:deitel = "http://www.deitel.com/booklist">
6     <book>
7        <title>Visual Basic 2010 How to Program</title>
8     </book>
9     <book>
10       <title>Visual C# 2010 How to Program, 4/e</title>
11    </book>
12    <book>
13       <title>Java How to Program, 9/e</title>
14    </book>
15    <book>
16       <title>C++ How to Program, 8/e</title>
17    </book>
18    <book>
19       <title>Internet and World Wide Web How to Program, 5/e</title>
20    </book>
21 </deitel:books>
```

Fig. 15.9 | Schema-valid XML document describing a list of books.

XML

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.10: book.xsd           -->
4  <!-- Simple W3C XML Schema document -->
```

Fig. 15.10 | XML Schema document for `book.xml`. (Part 1 of 2.)

```
5  <schema xmlns = "http://www.w3.org/2001/XMLSchema"
6     xmlns:deitel = "http://www.deitel.com/booklist"
7     targetNamespace = "http://www.deitel.com/booklist">
8
9     <element name = "books" type = "deitel:BooksType"/>
10
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>
```

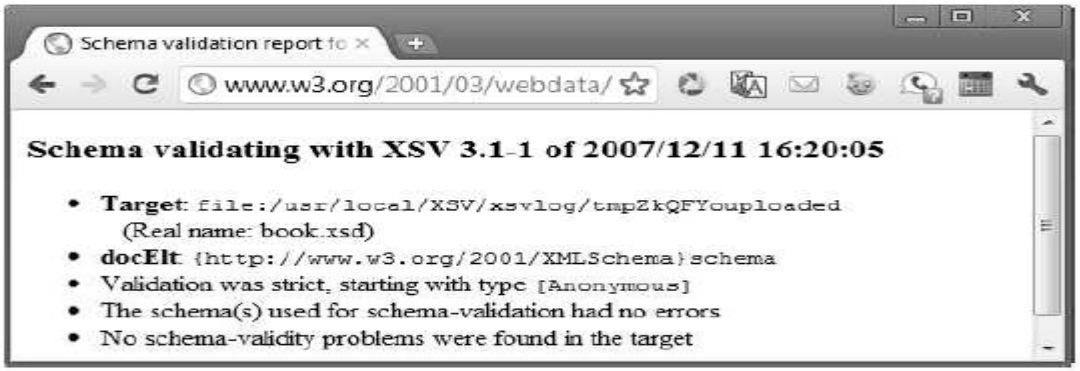

Fig. 15.10 | XML Schema document for `book.xml`. (Part 2 of 2.)

XML

- Figure 15.9 contains markup describing several Deitel books. The books element (line 5) has the namespace prefix deitel, indicating that the books element is a part of the http://www.deitel.com/booklist namespace.

Creating an XML Schema Document:

- Figure 15.10 presents the XML Schema document that specifies the structure of book.xml (Fig. 15.9). This document 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** (Fig. 15.10, lines 5–23) contains elements that define the structure of an XML document such as book.xml. Line 5 specifies as the default namespace the standard W3C XML Schema namespace URI—**http://www.w3.org/2001/XMLSchema**.
- This namespace contains predefined elements (e.g., root-element schema) that comprise the XML Schema vocabulary—the language used to write an XML Schema document.
- Line 6 binds the URI http://www.deitel.com/booklist to namespace prefix deitel. Line 7 also specifies http://www.deitel.com/booklist as the **targetNamespace** of the schema.
- This attribute identifies the namespace of the XML vocabulary that this schema defines. Note that the targetNamespace of book.xsd is the same as the namespace referenced in line 5 of book.xml (Fig. 15.9).
- This is what “connects” the XML document with the schema that defines its structure. When a schema validator examines book.xml and book.xsd, it will recognize that book.xml uses elements and attributes from the http://www.deitel.com/ booklist namespace.
- The validator also will recognize that this namespace is the namespace defined in book.xsd (i.e., the schema’s targetNamespace). Thus the validator knows where to look for the structural rules for the elements and attributes used in book.xml.

XML

Defining an Element in XML Schema:

- In XML Schema, the **element** tag (line 9 of Fig. 15.10) defines an element to be included in an XML document that conforms to the schema. In other words, element specifies the actual *elements* that can be used to mark up data.
- Line 9 defines the books element, which we use as the root element in book.xml (Fig. 15.9). Attributes **name** and **type** specify the element's name and type, respectively. An element's type indicates the data that the element may contain.
- Possible types include XML Schema-defined types (e.g., string, double) and user-defined types (e.g., BooksType, which is defined in lines 11–16 of Fig. 15.10). Figure 15.11 lists several of XML Schema's many built-in types.

Type	Description	Range or structure	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's 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, negative infinity and not-a-number (`NaN`).	`0`, `12`, `-109.375`, `NaN`

Fig. 15.11 | Some XML Schema types. (Part 1 of 2.)

XML

Type	Description	Range or structure	Examples
`double`	A floating-point number	`m * (2e)`, 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`
`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.	`2005-05-10`
`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. 15.11 | Some XML Schema types. (Part 2 of 2.)

XML

- In this example, books is defined as an element of type deitel:BooksType (line 9). BooksType is a user-defined type (lines 11–16 of Fig. 15.10) in the namespace http://www.deitel.com/booklist and therefore must have the namespace prefix deitel. It's not an existing XML Schema type.
- Two categories of type 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. Lines 11–16 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. The element (lines 13–14) nested within the complexType element indicates that a BooksType element (e.g., books) can contain child elements named book of type deitel:SingleBookType (defined in lines 18–22).
- Attribute **minOccurs** (line 14), with value 1, specifies that elements of type BooksType must contain a minimum of one book element. Attribute **maxOccurs** (line 14), with value **unbounded**, specifies that elements of type BooksType may have any number of book child elements.
- Lines 18–22 define the complex type SingleBookType. An element of this type contains a child element named title. Line 20 defines element title to be of simple type string.
- Recall that elements of a simple type cannot contain attributes or child elements. The schema end tag (</schema>, line 23) declares the end of the XML Schema document.

XML

A Closer Look at Types in XML Schema:

- Every element in XML Schema has a type. Types include the built-in types provided by XML Schema (Fig. 15.11) or user-defined types (e.g., SingleBookType in Fig. 15.10).
- Every simple type defines a **restriction** on an XML Schema-defined type or a restriction on a user-defined type. Restrictions limit the possible values that an element can hold.
- Complex types are divided into two groups—those with **simple content** and those with **complex content**. Both can contain attributes, but 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. We demonstrate complex types with each kind of content in the next example.
- The schema document in Fig. 15.12 creates both simple types and complex types. The XML document in Fig. 15.13 (laptop.xml) follows the structure defined in Fig. 15.12 to describe parts of a laptop computer.
- A document such as laptop.xml that conforms to a schema is known as an **XML instance document**—the document is an instance (i.e., example) of the schema.

XML

```
1  <?xml version = "1.0"?>
2  <!-- Fig. 15.12: computer.xsd -->
3  <!-- W3C XML Schema document  -->
4
5  <schema xmlns = "http://www.w3.org/2001/XMLSchema"
6     xmlns:computer = "http://www.deitel.com/computer"
7     targetNamespace = "http://www.deitel.com/computer">
8
9     <simpleType name = "gigahertz">
10       <restriction base = "decimal">
11          <minInclusive value = "2.1"/>
12       </restriction>
13    </simpleType>
14
15    <complexType name = "CPU">
16       <simpleContent>
17          <extension base = "string">
18             <attribute name = "model" type = "string"/>
19          </extension>
20       </simpleContent>
21    </complexType>
22
23    <complexType name = "portable">
24       <all>
25          <element name = "processor" type = "computer:CPU"/>
26          <element name = "monitor" type = "int"/>
27          <element name = "CPUSpeed" type = "computer:gigahertz"/>
28          <element name = "RAM" type = "int"/>
29       </all>
30       <attribute name = "manufacturer" type = "string"/>
31    </complexType>
32
33    <element name = "laptop" type = "computer:portable"/>
34 </schema>
```

Fig. 15.12 | XML Schema document defining simple and complex types.

XML

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 15.13: laptop.xml                    -->
4  <!-- Laptop components marked up as XML -->
5  <computer:laptop xmlns:computer = "http://www.deitel.com/computer"
6     manufacturer = "IBM">
7
8     <processor model = "Centrino">Intel</processor>
9     <monitor>17</monitor>
10    <CPUSpeed>2.4</CPUSpeed>
11    <RAM>256</RAM>
12 </computer:laptop>
```

Fig. 15.13 | XML document using the `laptop` element defined in `computer.xsd`.

XML

- Line 5 of Fig. 15.12 declares the default namespace to be the standard XML Schema namespace—any elements without a prefix are assumed to be in that namespace. Line 6 binds the namespace prefix computer to the namespace http://www.deitel.com/ computer.
- Line 7 identifies this namespace as the targetNamespace—the namespace being defined by the current XML Schema document. To design the XML elements for describing laptop computers, we first create a simple type in lines 9–13 using the **simpleType** element. We name this simpleType gigahertz because it will be used to describe the clock speed of the processor in gigahertz.
- Simple types are restrictions of a type typically called a **base type**. For this simpleType, line 10 declares the base type as decimal, and we restrict the value to be at least 2.1 by using the **minInclusive** element in line 11.
- Next, we declare a complexType named CPU that has **simpleContent** (lines 16–20). Remember that a complex type with simple content can have attributes but not child elements. Also recall that complex types with simple content must extend or restrict some XML Schema type or user-defined type.
- The **extension** element with attribute **base** (line 17) sets the base type to string. In this complexType, we extend the base type string with an attribute. The **attribute** element (line 18) gives the complexType an attribute of type string named model.
- Thus an element of type CPU must contain string text (because the base type is string) and may contain a model attribute that's also of type string.
- Last, we define type portable, which is a complexType with complex content (lines 23–31). Such types are allowed to have child elements and attributes. The element **all** (lines 24–29) encloses elements that must each be included once in the corresponding XML instance document. These elements can be included in any order.
- This complex type holds four elements—processor, monitor, CPU Speed and RAM. They're given types CPU, int, gigahertz and int, respectively. When using types CPU and gigahertz, we must include the namespace prefix computer, because these user-defined types are part of the computer namespace (http://www.deitel.com/computer)—the namespace defined in the current document (line 7). Also, portable contains an attribute defined in line 30.
- The attribute element indicates that elements of type portable contain an attribute of type string named manufacturer. Line 33 declares the actual element that uses the three types defined in the schema. The element is called laptop and is of type portable. We must use the namespace prefix computer in front of portable.

XML

- We've now created an element named laptop that contains child elements processor, monitor, CPU Speed and RAM, and an attribute manufacturer.
- Figure 15.13 uses the laptop element defined in the computer.xsd schema. Once again, we used an online XSD schema validator (www.xmlforasp.net/SchemaValidator.aspx) to ensure that this
- XML instance document adheres to the schema's structural rules. Line 5 declares namespace prefix computer.
- The laptop element requires this prefix because it's part of the http://www.deitel.com/computer namespace. Line 6 sets the laptop's manufacturer attribute, and lines 8–11 use the elements defined in the schema to describe the laptop's characteristics.
- This section introduced W3C XML Schema documents for defining the structure of XML documents, and we validated XML instance documents against schemas using an online XSD schema validator.