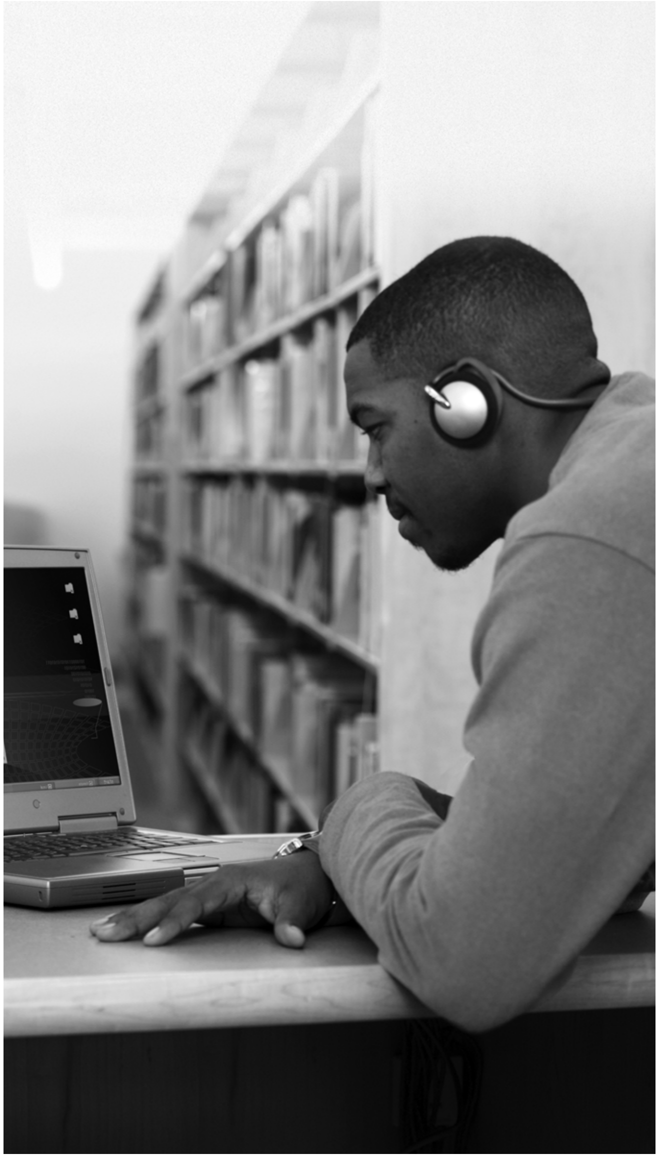


Chapter Goals

In this chapter, you will cover the following topics to learn how to present and transform content:

- **Introducing Markup Languages**—You can use markup languages to format and describe your content.
- **Transforming and Formatting Content**—Powerful tools are available to you for transforming and formatting content.

CHAPTER 7

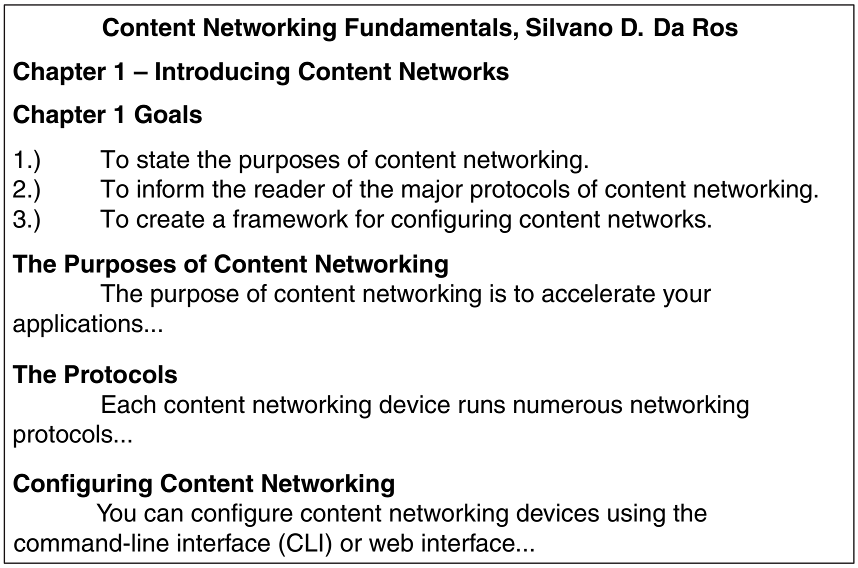
Presenting and Transforming Content

Introducing Markup Languages

You can use markup languages to manage structured documents in a standard format. To "mark up" a document means to imbed text within the content of the document in order to perform procedures on the data and convey information about it. Today's markup languages form their bases on **procedural markup** languages of the 1960s, when typesetters used proprietary coding for fine control over the font, size, and spacing of printed copy, with the intention of formatting documents destined for paper. For example, some book publishers required authors to submit their manuscript using proprietary procedural markup tags embedded in the text. Procedural coding leaves the task of formatting to the publisher who uses proprietary software to process the tags embedded in the document. This enables the author to concentrate on the content of the document.

Figure 7-1 gives a sample excerpt from the beginning of this book.

Figure 7-1 *A Sample Book Excerpt*



Content Networking Fundamentals, Silvano D. Da Ros

Chapter 1 – Introducing Content Networks

Chapter 1 Goals

1.) To state the purposes of content networking.
2.) To inform the reader of the major protocols of content networking.
3.) To create a framework for configuring content networks.

The Purposes of Content Networking

The purpose of content networking is to accelerate your applications...

The Protocols

Each content networking device runs numerous networking protocols...

Configuring Content Networking

You can configure content networking devices using the command-line interface (CLI) or web interface...

NOTE Common word processing packages, such as Microsoft Word or Corel WordPerfect, use procedural markup internally to format documents. Each package has its own set of proprietary procedural markup tags for formatting, which is why you normally require a conversion tool to read a document created by one package in the other.

To produce the formatting in Figure 7-1 using procedural markup, you can insert the imaginary procedural marks *!SkipLine=n!*, *!Center!*, *!Bold!*, and *!Indent=n!* in to the text to provide the correct formatting instructions to a word processor, browser, or printer. As you can see from Figure 7-2, the procedural markup tags specify a particular procedure that is to be applied to the text that it references.

Figure 7-2 *A Basic Procedural Markup Example*

```
!Center!!Bold!Content Networking Fundamentals, Silvano D. Da Ros!/Bold!!/Center!
!SkipLine=2!
!Bold!Chapter 1 – Introducing to Content Networks!/Bold!
!SkipLine=1!
!Bold!Chapter 1 Goals!/Bold!
!SkipLine=1!
1.)!Indent=1!To state the purposes of content networking.
2.)!Indent=1!To inform the reader of the major protocols of content networking.
3.)!Indent=1!To create a framework for configuring content networks.
!SkipLine=1!
!Bold!The Purposes of Content Networking!/Bold!
!Indent=1!The purpose of content networking is to accelerate your applications...
!SkipLine=1!
!Bold!The Protocols!/Bold!
!Indent=1!Each content networking device runs numerous networking protocols...
!SkipLine=1!
!Bold!Configuring Content Networking!/Bold!
!Indent=1!You can configure content networking devices using the command-line
interface (CLI) or web interface...
```

Descriptive markup languages, or generic coding, differs from procedural markup languages by describing the structure of the document, leaving the parsers (programs used to display, print, or store the information) to perform the desired procedure. For example, you can use exactly the same text document for printing, monitor display, or even Braille. In contrast, with procedural markup, you would require a separate document for each.

Procedural markup languages have the following disadvantages over descriptive languages for publishing documents on the web:

- **Inflexible**—The number of commands indicating how the text should be formatted (that is, skip line, indent, and so on) is often cumbersome for effective usage. For example, internally, most word processors contain thousands of tags, which are transparent to you. This number of tags is unmanageable for use in web publishing.

- **Requires Multiple Document Formats**—Procedural markups are inflexible because, if you require different styles of documents, the text needs to be marked up again for each style. For example, if a document destined for a printer needs to be displayed to a monitor, a new marked-up document is necessary, because even the slightest offset of margins requires reformatting.

Stanley Rice and William Tunnicliffe first conceived of the separation of content from its formatting in 1967. The first formal descriptive markup language was GenCode and was the first general markup specification for the typesetting industry. Gencode recognized the need of different codes for different types of documents. Together, Rice and Tunnicliffe formed the Graphic Communications Association (GCA) Gencode Committee to further develop their ideas into a nonproprietary generic coding markup standard.

IBM then took the ideas behind GenCode to produce Generalized Markup Language (GML—also the initials of its creators, Charles Goldfarb, Edward Mosher, and Raymond Lorie) in 1969, to organize IBM's legal documents into a searchable form. GML automated functions performed specifically on IBM's legal documents. The ANSI Computer Languages group expanded GML in 1980 into the Standard Generalized Markup Language (SGML) for the Processing of Text Committee. In 1984, the International Standards Organization (ISO) joined the ANSI committee to publish the SGML standard based on its sixth working draft in 1986 as an international norm (ISO 8879). The first important users of the standard were the US Internal Revenue Service (IRS) and Department of Defense (DoD).

Figure 7-3 illustrates how to mark up the example described previously using SGML.

Figure 7-3 *A Basic Structural Markup Example Using SGML*

```
<Book title="Content Networking Fundamentals" author="Silvano D. Da Ros">
<Chapter>
<ChapterTitle>Chapter 1 – Introducing Content Networks</ChapterTitle>
<ChapterGoals title="Chapter 1 Goals" >
<Goal>To state the purposes of content networking.</Goal>
<Goal>To inform the reader of the major protocols of content networking.</Goal>
<Goal>To create a framework for configuring content networks.</Goal>
</ChapterGoals>
</Chapter>
<Section title= "The Purposes of Content Networking " >
The purpose of content networking is to accelerate your applications...
</Section>
<Section title= "The Protocols" >
Each content networking device runs numerous networking protocols...
</Section>
<Section title= "Configuring Content Networking" >
You can configure content networking devices using the command-line interface (CLI)
or web interface...
</Section>
</Book>
```

The tags shown in Figure 7-3 group the document into elements. For example, you can use the tags <Book>, <Chapter>, and <Goal> to group the document into elements. Each piece of content has the general form of a start-tag (for example, "<Book>") followed by the content, and ending with the end-tag (for example, </Book>). The title of the <Book> element is an attribute of the element as opposed to being an element itself.

Because the elements, attributes, and overall structure of Figure 7-3 are specific to the application, you should formally define them for the application. This need for formally declaring elements, attributes, and structure gave birth to the Document Type Definition (DTD) file, against which you can validate markup language files. DTD files define the syntax of the markup language. That is, it declares all tags within the markup file and specifies the order with which they should appear, which ones are optional or repeatable, and that they are properly nested. You can use DTD files to establish portability and interoperability and exchange data between organizations with different file formats.

The DTD file for the SGML sample in Figure 7-3 is in Table 7-1. The <!ELEMENT> tag defines each element, preceded by the <ATTLIST> tag, if the element contains any attributes (for example, title and author).

Table 7-1 *A Sample Book.dtd DTD File*

DTD Information	Description
<!ELEMENT book (chapter+)> <!ATTLIST book title CDATA #REQUIRED author CDATA #IMPLIED>	The Book element contains one or more chapter elements. The + indicates that one or more elements may exist. The Book element contains attributes for the required book's title and optional (IMPLIED) author name.
<!ELEMENT chapter (chaptertitle,chaptergoals?,section+)>	Each chapter contains one chapter title, an optional "Chapter Goals" area (as indicated by a question mark), and one or more Sections elements.
<!ELEMENT chaptertitle (#PCDATA)>	The Chapter Title contains parsable character data (meaning it can contain tags within the data), which require special parsing to be rendered properly.
<!ELEMENT chaptergoals (goal+)> <!ATTLIST chaptergoals title CDATA #REQUIRED>	The Chapter Goals area contains one or more Goals Elements.
<!ELEMENT goal (#PCDATA) >	Each goal is parsable data.

Table 7-1 *A Sample Book.dtd DTD File (Continued)*

DTD Information	Description
<!ELEMENT section (subsection*)> <!ATTLIST section title CDATA #REQUIRED>	A section contains a title attribute, optional parsable character data, and zero or more sub-sections.
<!ELEMENT subsection (subsubsection*)> <!ATTLIST subsection title CDATA #REQUIRED>	A subsection contains a title attribute and zero or more subsubsections.
<!ELEMENT subsubsection (#PCDATA)> <!ATTLIST subsubsection title CDATA #REQUIRED>	A subsubsection contains a title attribute and parsable text.

Hypertext Markup Language

HTML was developed as a simple means to publish hyperlinked documents in a standard fashion. HTML enables you to avoid proprietary formats, and thus promote interoperability between the various devices expected to connect to the web. At the time, SGML was considered too bulky and complicated for such a "simple" environment. In general, HTML is an application of SGML, and includes a minor subset of simple tags for organizing content on the web.

NOTE You can use DTD files to define not only applications of certain markup languages but entire markup languages themselves. For example, just as in the book example above, HTML has its own SGML-compliant structure and subset of tags, which are used to mark up hypertext on the WWW.

In 1990, Tim Berners-Lee, then working at the Organsation Européenne pour la recherche nucléaire (CERN), published the first version of the HTML DTD (that is, HTML 1.0). Tim developed the first prototype browser, supporting HTML transported in HTTP over a TCP/IP network, resulting in the birth of the World Wide Web. The computer community received HTML 1.0 with welcoming arms, and many text-based browsers, such as Viola, Cello, and Lynx, became available shortly after its release.

The IETF published HTML version 2 as RFC 1866 in 1994. Version 2.0 included many new features and fixes to version 1.0, such as support for images and forms. The National Center for Supercomputing Applications (NCSA) developed the first graphical browser for HTML 2.0, then called Mosaic, in late 1993. The developers of Mosaic soon decided that leaving NCSA to form Netscape would be a profitable endeavor.

NOTE Numerous parties with vested interests in web protocols formed the W3C consortium in 1993 to take web standardization into a nonprofit and unincorporated setting. Soon after work began on HTML version 3.0 at W3C.

In 1993, Netscape developed HTML+ for its Mosaic browser, based on HTML version 2.0, but it included many additional practical features over HTML version 2.0. Numerous competitive companies followed suite with various browsers with support for HTML version 2.0, the largest being Microsoft with Internet Explorer. Although the two largest browser companies developed browsers with close interpretation of the HTML spec, they each developed new and incompatible tags. The browser manufacturers quickly diverged from one another, creating a highly competitive browser market. As a result, the differences between HTTP 2.0 and 3.0 and Netscape's HTML+ were so vast that W3C decided to avoid standardizing 3.0 and instead to include the version 3.0 and HTML+ updates in version HTML 3.2, among various fixes and other new features. Thus, HTTP 3.2 was released in 1997 and included the generally accepted practices at the time, or as general as possible given the major explosion of web applications developed with HTML. HTML 4.0 and 4.0.1 are the most recent versions of documents. The HTML 4.0.1 specification comprises three separate DTDs maintained and published by W3C.

HTML is an excellent markup language for displaying content for humans to read on a screen and for navigation between documents. However, even though the use of HTML is widespread, it soon proved to be insufficient in abstraction and structure for today's increasingly complex content-based applications. Like its procedural markup predecessor, presentation and formatting were given higher priority during the drafting of the HTML DTD than structure and organization, especially since the popularization of style sheets within HTML.

NOTE You can use style sheets to further separate content from the presentation of content of web documents written in HTML or Extensible HTML (XHTML). You will learn about Cascading Style Sheets (CSS) and XHTML later in this chapter.

Although HTML is a structural markup language based on SGML, the HTML tags do not sufficiently describe the content, and the specification is very loose in terms of syntax and structure as compared to SGML. Due to the explosion of the web, content providers quickly thirsted for control over formatting that was similar to that used with printed copy. Browser developers in conjunction with W3C responded with numerous HTML presentational controls. As HTML matured, its procedural markup features were replaced by mechanisms, such as converting text to images, using proprietary HTML extensions, and style sheets, as simple ways to separate presentation from content without severely changing the markup language specification.

Example 7-1 shows how an HTML file is structured.

Example 7-1 *A Sample HTML Document to Print "Hello World" to a Web Browser*

```
<HTML>
<HEAD>
<TITLE>Hello World Page/TITLE>
</HEAD>
<BODY>
Hello World!
</BODY>
</HTML>
```

To fully overcome the limitations of HTML, people recently favor more robust descriptive markup languages coupled with separate presentational markup languages as successors to HTML. In order to bridge the gap between the structured, self-descriptive nature of SGML and the usability of HTML for visual, interactive web applications, the W3C created the XML family of markup languages to simplify HTML.

Extensible Markup Language

Like HTML, Extensible Markup Language (XML) is an application of the SGML protocol, but includes more of the semantic aspects of SGML. XML is a true structural markup language in that it does not do anything to the data but just describe it. In contrast to HTML, which achieved a certain level of structure with abstractions such as headings, paragraphs, emphasis, and numbered lists, in XML you can create custom XML tags to describe content (for example, Book, Section, and Goals are custom XML tags). HTML has only a specific set of tags available to describe content (e.g., Header, Body, and so on), and vast numbers of tags to perform actions on the data.

NOTE In contrast to the way you use HTML, you use XML to carry data, not both data and presentation information. In order to present the data, you must use a style sheet or transform the XML document into HTML or XHTML. You will learn how to present XML later in this chapter.

XML was published as a W3C Recommendation in early 1998. Much of the out-of-date features are excluded from XML. For example, SGML typewriter directives are no longer pertinent today. XML also extends SGML with its internationalization features and typing of elements using XML schemas.

With HTML, user agents can accept any syntax and try to make sense out of it, without giving errors. User agents are therefore difficult to write because an enormous number of erroneous pages exist on the web. The validation of XML is much more deliberate, easing the pressure on user agent developers to perform the complex error correction required on poorly written HTML. The downfall is that users must conform to the strict rules imposed by XML to avoid errors in their documents.

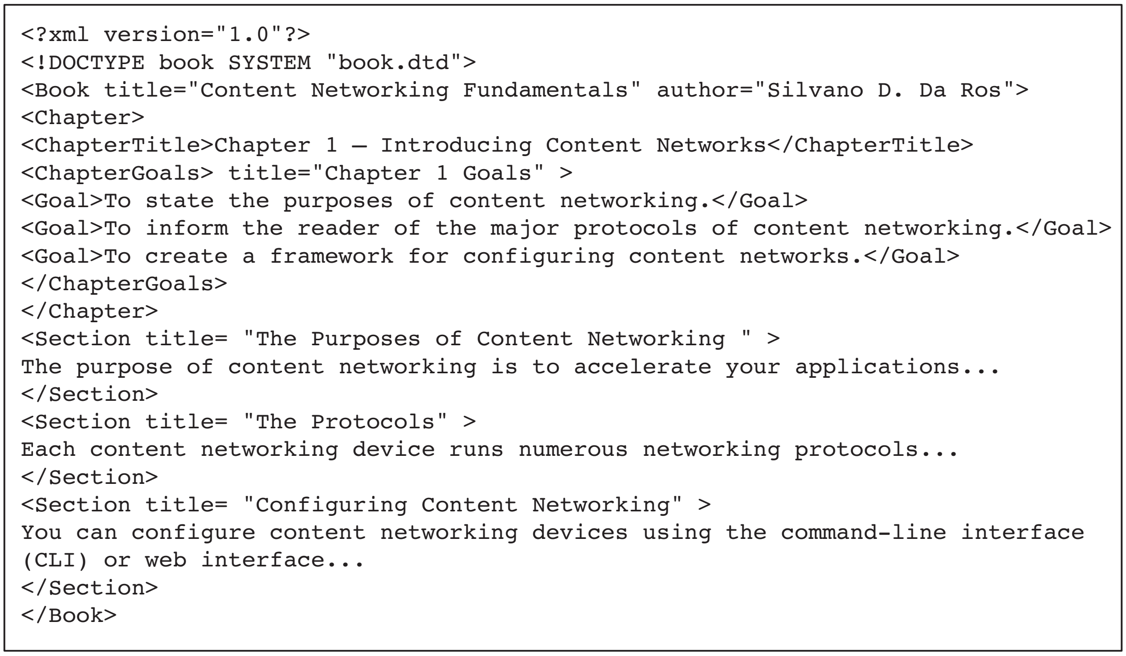
> **NOTE** The term *user agent* refers to any program that fetches, parses, and optionally displays web pages. Search engine robots are user agents, which is why you will not often see the term *web browser* used in most web texts, journals, and standardization documents to refer to all such agents.

Drawing on the ability to create custom elements in XML, numerous associated XML-based languages are available to you for extending the basic functionality of XML. Each requires special applications to recognize and perform actions on their respective custom-defined elements.

- **Extensible StyleSheet Language (XSL) and Extensible Stylesheet Transformation (XSLT)**—You can use the XSL and XSLT languages to display and transform XML documents, for Cisco IP phones, WAP cell phones, and PDAs.
- **Extensible StyleSheet Language-Format Object (XSL-FO)**—You can use XSL-FO to format documents for print, such as Adobe PDF files and barcodes.
- **XPath**—Use XPath to specify locations within XML documents, similar to the way files are organized on a standard computer file system. XPath is not an application of XML, but it is a major component in XSLT. You will see how XPath works with XSLT later in this chapter.
- **XLink**—Use XLink for hyperlinking between XML documents. XLink is similar to HTML links but includes many extensions, such as bidirectional, typed, one-to-many and many-to-many links. You can also use XLink to download links automatically or on user request.
- **XQuery**—You can perform queries on XML files using XQuery, similar to the way in which you use Structure Query Language (SQL) queries in database systems.
- **Synchronized Multimedia Integration Language SMIL**—Use SMIL for the multimedia structured markup. You will learn about SMIL in Chapter 9, "Introducing Streaming Media."
- **Scalable Vector Graphics (SVG)**—Use SVG for structuring graphics.
- **Resource Description Framework (RDF)**—Use RDF for structured metadata markup.
- **MathML**—You can use MathML for mathematical equation structured markup.

Figure 7-4 shows the sample document described previously in Figure 7-1, structured in XML. Notice that the simple SGML example discussed previously in Figure 7-3 is identical when written XML, except for the required "?xml version" header.

Figure 7-4 *Sample XML File*



```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "book.dtd">
<Book title="Content Networking Fundamentals" author="Silvano D. Da Ros">
<Chapter>
<ChapterTitle>Chapter 1 – Introducing Content Networks</ChapterTitle>
<ChapterGoals> title="Chapter 1 Goals" >
<Goal>To state the purposes of content networking.</Goal>
<Goal>To inform the reader of the major protocols of content networking.</Goal>
<Goal>To create a framework for configuring content networks.</Goal>
</ChapterGoals>
</Chapter>
<Section title= "The Purposes of Content Networking " >
The purpose of content networking is to accelerate your applications...
</Section>
<Section title= "The Protocols" >
Each content networking device runs numerous networking protocols...
</Section>
<Section title= "Configuring Content Networking" >
You can configure content networking devices using the command-line interface
(CLI) or web interface...
</Section>
</Book>
```

NOTE XML with correct syntax is *well-formed XML*. XML validated against a DTD is *valid XML*. You can optionally specify the DTD to validate an XML file against in the header of the XML document, as show in Figure 7-4. You can also use XML Schemas as an XML-based alternative to the standard DTDs. Relax NG is the schema language by OSI.

Extensible Hypertext Markup Language

Extensible HTML (XHTML) is the next step in the evolution of web documents, and its creation was motivated by the need to deliver content to many different types of devices, such as mobile phones, PDAs, and web kiosks. As the name suggests, it is a combination of XML and HTML. More specifically, the XHTML DTDs are a reformulation of the three HTML 4.0 DTDs, as an application of XML (recall that HTML 4.0 is conversely an application of SGML). In other words, the HTML DTDs where rewritten within the XML DTD, creating the new XHTML DTD. With the new definitions, the old HTML syntax must follow the same strict rules as XML. This leads the way to a more standardized language, as user agents gradually transition to XHTML.

NOTE Because documents in XHTML conform to both XML and HTML 4, you can view them in user agents supporting either type.

Although HTML may never totally retire as a web markup language, it will become much more extensible and standardized under the guise of XML. It will be extensible in that you can create your own tags and standardized in that user agents concern themselves with the standards of the XML specification, not the complex HTML error-correction methods stemming from a lack of standard syntax. Important differences between HTML and XHTML are

- You must nest XHTML elements properly.
- XHTML documents must be well-formed.
- Tag names must be in lowercase.
- You must close all XHTML elements.

Wireless Application Protocol Markup Languages

New business potential in mobile browsing has fostered the development in Wireless Application Protocol (WAP). You can use WAP to supply web content to mobile devices, such as cell phones, pagers, and PDAs. Just as the W3C is responsible for web protocols, the WAP Forum is responsible for its wireless protocol counterparts. The WAP 1.0 protocol is composed of the following specifications:

- **Wireless Markup Language (WML) 1.0 language**—Use WML structural markup language for WAP content rendering. WML is an application of XML and as such strictly adheres to the XML specification.
- **WMLScript language**—A scaled-down scripting language for wireless devices, similar to JavaScript or VBscript for HTML client or server scripting or both.
- **Wireless Telephony Application Interface (WTAI)**—API for making phone calls from data connections.

WAP is an application of XML. Using the analogy of playing cards, WML pages are called decks, and contain one or more cards. WAP devices download all the cards at once but are displayed one at a time to the user. Figure 7-5 illustrates how to publish an online book in WML.

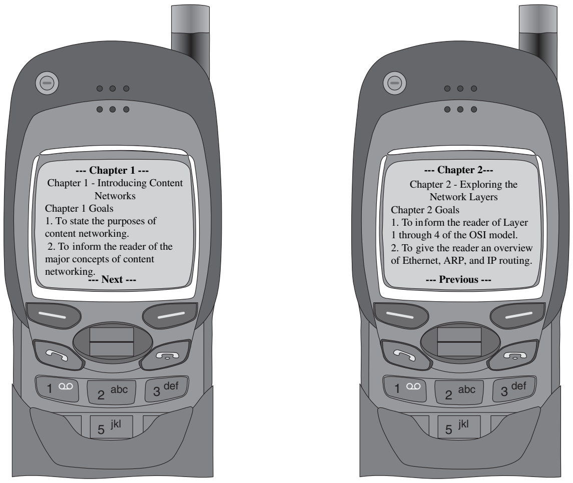
Figure 7-5 *A Sample WML File*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN" "http://www.wapforum.org/DTD/wml13.dtd">
<wml>
  <card id="ch1" title="Chapter 1 - Introducing Content Networks">
    <p>Chapter 1 Goals<br/>
      1. To state the purposes of content networking.<br/>
      2. To inform the reader of the major concepts of content networking.<br/>
      3. To detail the underlying protocols of content networking.<br/>
    </p>
  </card>

  <card id="ch2" title="Chapter 2 - Exploring the Network Layers">
    <p>Chapter 2 Goals<br/>
      1. To inform the reader of Layers 1 through 4 of the OSI model.<br/>
      2. To give the reader an overview of Ethernet, ARP, and IP routing.<br/>
      3. To illustrate basic TCP operation.<br/>
    </p>
  </card>
</wml>
```

Figure 7-6 shows how you can navigate between individual cards, or chapters, using the WAP device controls.

Figure 7-6 *Navigating a WML Document on a WAP Device*



--- Chapter 1 ---
Chapter 1 - Introducing Content Networks
Chapter 1 Goals
1. To state the purposes of content networking.
2. To inform the reader of the major concepts of content networking.
--- Next ---

--- Chapter 2---
Chapter 2 - Exploring the Network Layers
Chapter 2 Goals
1. To inform the reader of Layer 1 through 4 of the OSI model.
2. To give the reader an overview of Ethernet, ARP, and IP routing.
--- Previous ---

The W3C specifies a subset of XHTML 1.1 for small devices, called XHTML Basic. However, the WAP Forum created WAP 2.0 to include the XHTML Basic features plus some of the features from the full XHTML 1.1 specification, called the XHTML Mobile Profile (XHTMLMP), or Wireless Markup Language 2.0 (WML 2.0). WAP 2.0 was motivated by advancements in wireless transmission technologies, such as GSM, GPRS, G2.5, and G3.

WAP 2.0 also introduced support for special WAP versions of TCP/IP protocols in order to leverage the same languages and tools for mobile and standard web content (alternatively, WAP 1.0 uses the WAP protocol stack and does not support connectivity to TCP/IP networks). The wTCP/IP protocol supports TCP/IP, HTTP for content transport, and PKI for content security. Additionally, the power of CSS is available to you in WAP 2.0-enabled devices for the possibility to control a document's layout, including the text fonts, text attributes, borders, margins, padding, text alignment, text colors, and background colors to name a few. WAP 2.0 also supports XSLT transformation to transform between WML 1.0 and WML 2.0 documents.

Transforming and Formatting Content

You can use XSL to transform, filter, sort, and format your content. The XSL family consists of XSLT, XPath, and XSL-FO. Use XSLT for transforming XML documents, XPath for defining parts of an XML document, and XSL-FO for formatting XML documents.

You have two options to transform your XML documents for the purpose of publishing them to the web:

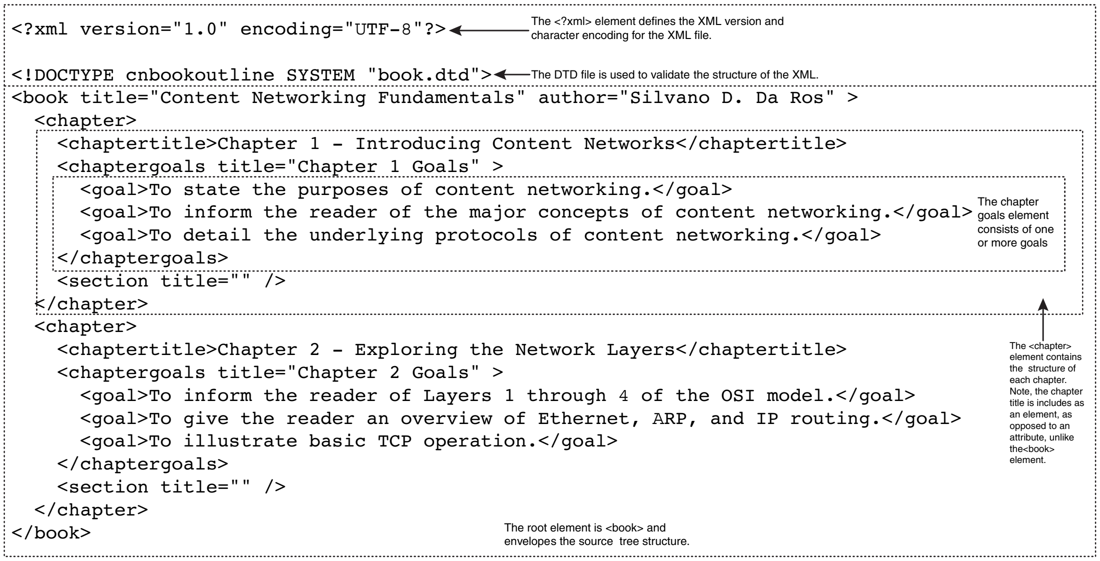
- **Transforming XML to XHMTL/HMTL**—You can translate your XML documents into HTML or XHTML and apply CSS's to the documents for display to a web browser.
- **Transforming XML to XSL-FO**—You can translate your XML documents directly into XSL-FO. You can then use a third-party program to convert the standard XSL-FO into HTML/XMTL/CSS (as mentioned previously), Braille, bar-codes, Adobe PDF, PostScript, SVG, Abstract Windowing Toolkit (AWT), or Maker Interchange Format (MIF).

NOTE You can apply XSL stylesheets to your content within your network using the Cisco Application Oriented Network (AON) network modules for the Catalyst 6500 series switches and Cisco 2600/2800/3700/3800 Series routers. For more information on the AON, refer to its product documentation on Cisco.com.

Transforming XML to XHMTL/HMTL

Consider an application where you would like to publish an outline of this book on the web. The outline will contain the book title, author name, and chapter titles followed by the goals within each chapter. The content of each chapter's sections and subsections will not be included in the outline but assume that the entire book is available and marked up with the elements discussed previously in the simple DTD in Table 7-1. Figure 7-7 gives the sample XML file containing the outline of the first two chapters.

Figure 7-7 *A Sample Book Outline XML File*



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

<!DOCTYPE cnbookoutline SYSTEM "book.dtd">
<book title="Content Networking Fundamentals" author="Silvano D. Da Ros" >
  <chapter>
    <chaptertitle>Chapter 1 - Introducing Content Networks</chaptertitle>
    <chaptergoals title="Chapter 1 Goals" >
      <goal>To state the purposes of content networking.</goal>
      <goal>To inform the reader of the major concepts of content networking.</goal>
      <goal>To detail the underlying protocols of content networking.</goal>
    </chaptergoals>
    <section title="" />
  </chapter>
  <chapter>
    <chaptertitle>Chapter 2 - Exploring the Network Layers</chaptertitle>
    <chaptergoals title="Chapter 2 Goals" >
      <goal>To inform the reader of Layers 1 through 4 of the OSI model.</goal>
      <goal>To give the reader an overview of Ethernet, ARP, and IP routing.</goal>
      <goal>To illustrate basic TCP operation.</goal>
    </chaptergoals>
    <section title="" />
  </chapter>
</book>
```

The <?xml> element defines the XML version and character encoding for the XML file.

The DTD file is used to validate the structure of the XML.

The chapter goals element consists of one or more goals

The <chapter> element contains the structure of each chapter. Note, the chapter title is includes as an element, as opposed to an attribute, unlike the<book> element.

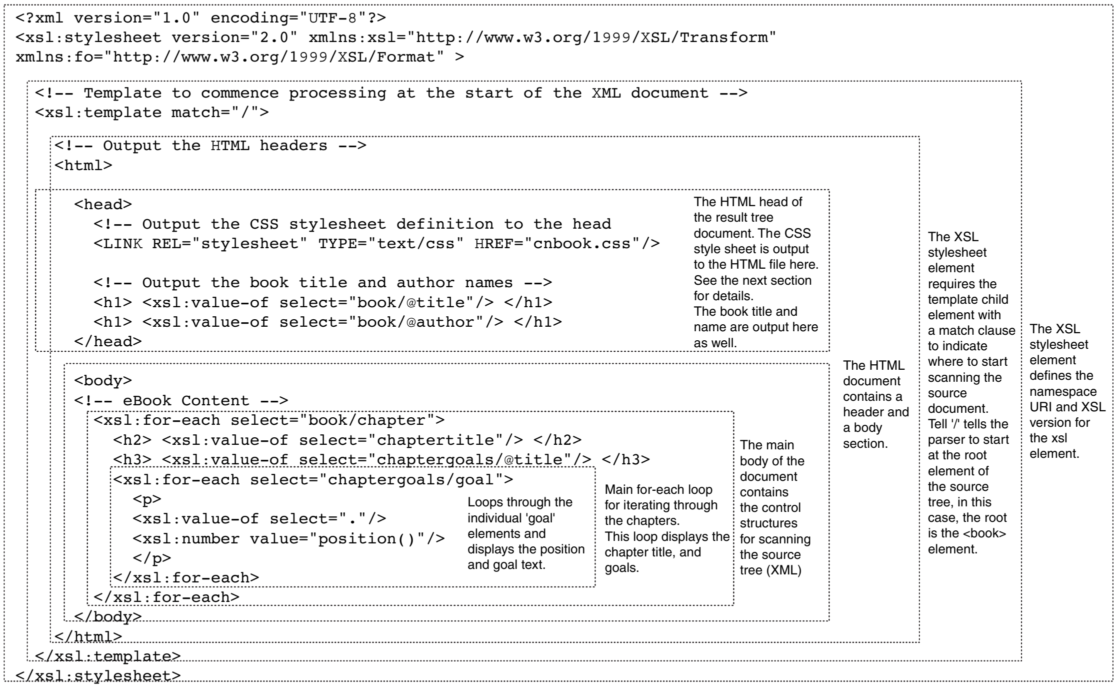
The root element is <book> and envelopes the source tree structure.

Because you define custom element names in XML, name conflicts may occur when the same name from different DTDs is used to describe two different types of elements. You can use XML namespaces to provide unique element names within an XML document. In Figure 7-8, the namespace is the string ”xsl:“ that prefixes all of the XSL elements. You are required to use a namespace to differentiate elements among languages. The particular application that parses the document will know what to do with the specific elements based on the prefix. For example, an XSLT parser will look for the xsl: namespace URI and perform the intended actions based on the elements in the document. Alternatively, a XSL-FO parser will see the “fo:” namespace and perform the appropriate actions using the respective elements.

Parsers do not use the URL of the namespace to retrieve a DTD or schema for the namespace—the URL is simply a unique identifier within the document. According to W3C, the definition of a namespace simply defines a two-part naming system and nothing else. However, you must define namespaces of individual markup languages with a specific URL for the parsing application to take action on tags within the context of the language. For example, you must define the XSLT namespace with the URL “http://www.w3.org/1999/XSL/Transform” in your documents. Additionally, you must define the XSL-FO namespace with “http://www.w3.org/1999/XSL/Format.” That said, many simple XML parsing applications do not require namespaces in order to differentiate elements; they simply treat all elements as within the same namespace. However, this chapter uses namespaces strictly for illustration purposes.

In Figure 7-8, the namespace for XSL is defined for the XSLT parser to recognize the XSL specific elements **value-of**, **for-each**, and **number**. An XSLT parser inputs the XSLT file and the XML source file. It processes these two files and outputs an HTML file as a well-formed XML document. There are many other XSLT elements available to you, but you should know at least these three to understand the content transformations in this section.

Figure 7-8 *Sample XSLT File Transforming XML to HTML*



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format" >

  <!-- Template to commence processing at the start of the XML document -->
  <xsl:template match="/">

    <!-- Output the HTML headers -->
    <html>

      <head>
        <!-- Output the CSS stylesheet definition to the head
        <LINK REL="stylesheet" TYPE="text/css" HREF="cnbook.css"/>

        <!-- Output the book title and author names -->
        <h1> <xsl:value-of select="book/@title"/> </h1>
        <h1> <xsl:value-of select="book/@author"/> </h1>
      </head>

      <body>
      <!-- eBook Content -->
        <xsl:for-each select="book/chapter">
          <h2> <xsl:value-of select="chaptertitle"/> </h2>
          <h3> <xsl:value-of select="chaptergoals/@title"/> </h3>
          <xsl:for-each select="chaptergoals/goal">
            <p>
            <xsl:value-of select="."/>
            <xsl:number value="position()"/>
            </p>
          </xsl:for-each>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

The HTML head of the result tree document. The CSS style sheet is output to the HTML file here. See the next section for details. The book title and name are output here as well.

Loops through the individual 'goal' elements and displays the position and goal text.

Main for-each loop for iterating through the chapters. This loop displays the chapter title, and goals.

The main body of the document contains the control structures for scanning the source tree (XML)

The HTML document contains a header and a body section.

The XSL stylesheet element requires the template child element with a match clause to indicate where to start scanning the source document. Tell '/' tells the parser to start at the root element of the source tree, in this case, the root is the <book> element.

The XSL stylesheet element defines the namespace URI and XSL version for the xsl element.

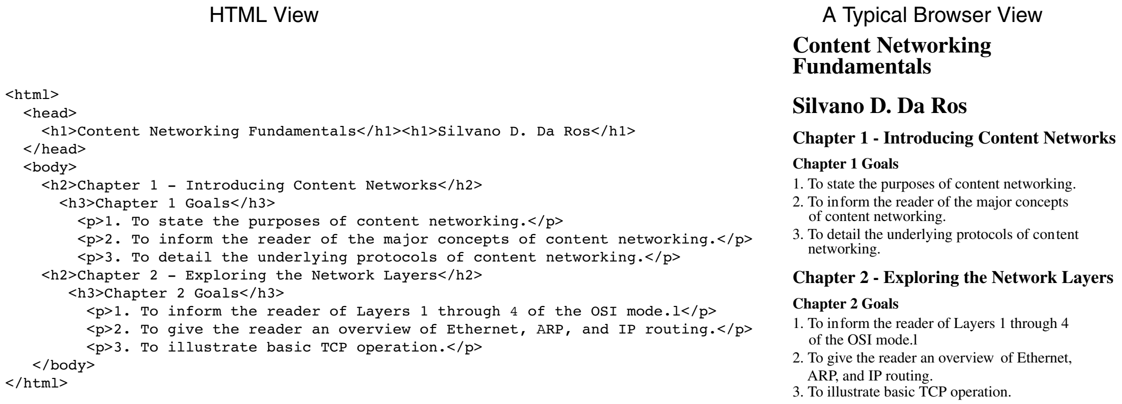
The first line in Figure 7-8 defines the XML version and encoding scheme. The second line defines the namespace for the XSL elements within the document. The XSL element "template" imposes a logical template for the whole document. The parser outputs the <head> and <html> tags without modification. The two <h1> tags within the <head> section are output containing the title and author name attributes from the source file.

> **NOTE** The elements <h1>, <h2>, and <h3> have specific implied formats when read by browsers in HTML. However, you can adjust the implied formats of these tags using CSS, as discussed in the next section.

The XSLT language organizes the XML elements into a tree structure, using XPath, similar to the way in which a standard computer file system organizes files. You reference the node elements or attributes by specifying the entire path, starting at the current location in the tree. In this case, the root element contains the desired attribute, so you should use path "book/@title." The "@" character indicates to the parser to select an attribute as opposed to an element.

The parser then reads the content of the book from the XML source file. The **for-each** element iterates through each of the elements given within the **select** attribute. Within the outer **for-each** element, the parser first outputs the chapter goals title and then begins another **for-each** loop to iterate through the list of chapter goals. Figure 7-9 gives the output from the XSLT translation file in Figure 7-8. The text view is the exact text output by the XSLT file, and the browser view is how the HTML looks from an HTML 4.0-based web browser's interpretation of the tags.

Figure 7-9 *Output HTML from XSLT Transformation*



HTML View

```
<html>
  <head>
    <h1>Content Networking Fundamentals</h1><h1>Silvano D. Da Ros</h1>
  </head>
  <body>
    <h2>Chapter 1 - Introducing Content Networks</h2>
      <h3>Chapter 1 Goals</h3>
        <p>1. To state the purposes of content networking.</p>
        <p>2. To inform the reader of the major concepts of content networking.</p>
        <p>3. To detail the underlying protocols of content networking.</p>
    <h2>Chapter 2 - Exploring the Network Layers</h2>
       <h3>Chapter 2 Goals</h3>
         <p>1. To inform the reader of Layers 1 through 4 of the OSI mode.l</p>
         <p>2. To give the reader an overview of Ethernet, ARP, and IP routing.</p>
         <p>3. To illustrate basic TCP operation.</p>
   </body>
</html>
```

A Typical Browser View

Content Networking Fundamentals

Silvano D. Da Ros

Chapter 1 - Introducing Content Networks

Chapter 1 Goals

1. To state the purposes of content networking.
2. To inform the reader of the major concepts of content networking.
3. To detail the underlying protocols of content networking.

Chapter 2 - Exploring the Network Layers

Chapter 2 Goals

1. To inform the reader of Layers 1 through 4 of the OSI mode.l
2. To give the reader an overview of Ethernet, ARP, and IP routing.
3. To illustrate basic TCP operation.

NOTE To transform the XML source file into a WML file instead, you require a new XSL transformation file. Instead of outputting HTML tags, you output WML tags, leaving the overall flow of the XSLT file the same.

Using Cascading Style Sheets

You can use CSSs to separate the formatting of a web document from the content in the document. Style sheets are useful because you can locate them in files that are separate from the content, allowing for multiple formats for the same content. For example, you can create two versions of your website: a standard style and a style for the visually impaired containing clearer images and larger font. Another example is the format specific to the different series of Cisco IP phones. Each series of IP phone has a different size display and requires special consideration with respect to content placement.

The concept of CSSs gives your authors the ability to blend different style sheets into the same document, as opposed to using completely separate styles for different groups of end users or different displays. For example, the author of a Cisco.com page can apply three different style sheets for a page within the Cisco TAC website. The first style sheet may impose the Cisco corporate look and feel. The second style sheet may apply to the standard TAC presentation, and the third may apply a format for the series of TAC documents that the author is writing for, such as network troubleshooting topics.

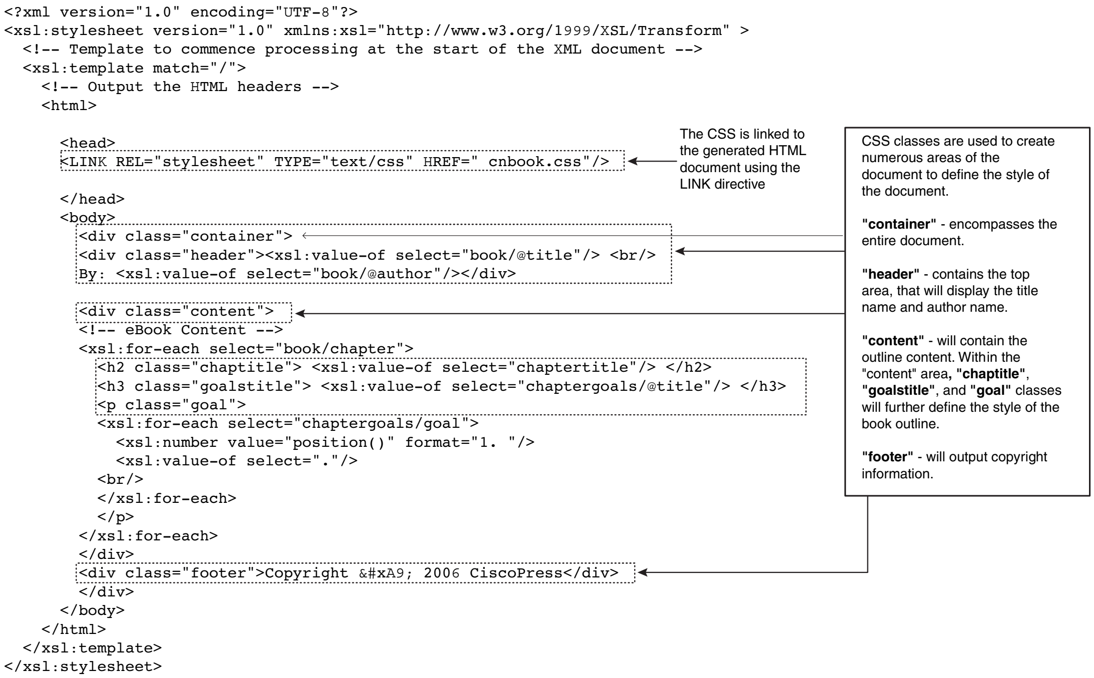
You can use the CSS file in Example 7-2 to format the HTML generated in Figure 7-9.

Example 7-2 *Sample CSS File for Formatting a Standard HTML Document*

```
body { background-color: #FFFFFF; }
h1 { font-family: Arial, sans-serif; font-size: 20px; color: #660000; text-align: center}
h2 { font-family: Arial, sans-serif; font-size: 16px; color: #660000 }
h3 { font-family: Arial, sans-serif; font-size: 14px; color: #003333; }
p { font-family: Arial, sans-serif; font-size: 12px; color: #003333;}
```

Alternatively, you can generate HTML using XSLT to include CSS classes. Figure 7-10 illustrates how you can use XSLT to generate HTML with CSS classes, to provide a robust formatting solution to your XML documents.

Figure 7-10 *XSLT for Generating HTML with Embedded CSS Classes*



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <!-- Template to commence processing at the start of the XML document -->
  <xsl:template match="/">
    <!-- Output the HTML headers -->
    <html>

      <head>
      <LINK REL="stylesheet" TYPE="text/css" HREF=" cnbook.css"/>

      </head>
      <body>
        <div class="container">
        <div class="header"><xsl:value-of select="book/@title"/> <br/>
        By: <xsl:value-of select="book/@author"/></div>

        <div class="content">
        <!-- eBook Content -->
        <xsl:for-each select="book/chapter">
          <h2 class="chaptitle"> <xsl:value-of select="chaptertitle"/> </h2>
          <h3 class="goalstitle"> <xsl:value-of select="chaptergoals/@title"/> </h3>
          <p class="goal">
          <xsl:for-each select="chaptergoals/goal">
            <xsl:number value="position()" format="1. "/>
            <xsl:value-of select="."/>
          <br/>
          </xsl:for-each>
          </p>
        </xsl:for-each>
        </div>
        <div class="footer">Copyright &#xA9; 2006 CiscoPress</div>
        </div>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

The CSS is linked to the generated HTML document using the LINK directive

CSS classes are used to create numerous areas of the document to define the style of the document.

"container" - encompasses the entire document.

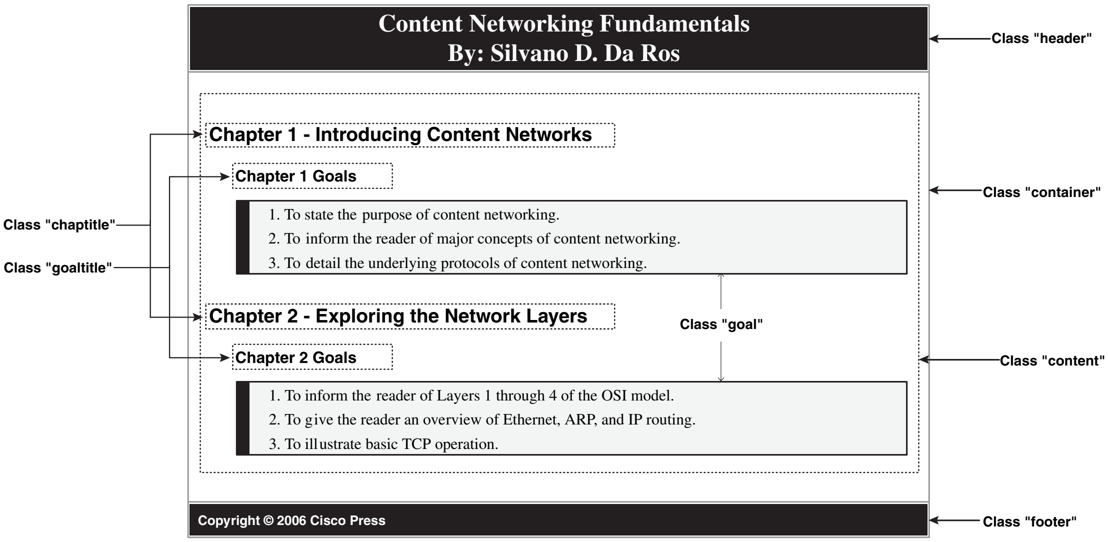
"header" - contains the top area, that will display the title name and author name.

"content" - will contain the outline content. Within the "content" area**, "chaptitle"**, **"goalstitle"**, and **"goal"** classes will further define the style of the book outline.

"footer" - will output copyright information.

The XSLT file in Figure 7-10 will generate the formatted document in Figure 7-11.

Figure 7-11 *Sample HTML Document Formatted with CSS*



Content Networking Fundamentals
By: Silvano D. Da Ros

Chapter 1 - Introducing Content Networks

Chapter 1 Goals

1. To state the purpose of content networking.
2. To inform the reader of major concepts of content networking.
3. To detail the underlying protocols of content networking.

Chapter 2 - Exploring the Network Layers

Chapter 2 Goals

1. To inform the reader of Layers 1 through 4 of the OSI model.
2. To give the reader an overview of Ethernet, ARP, and IP routing.
3. To illustrate basic TCP operation.

Copyright © 2006 Cisco Press

You can use the CSS file in Figure 7-12 to provide the format attributes for the classes described previously.

Style sheets are beneficial when you require rendering a large number of documents into the same style. With a standard XML format, your authors can create content and use a given set of markup tags to describe the content. If the documents require different versions, such as XHTML and HTML for online viewing or Braille and PDF for printing, you will require a separate XSL transformation file for each. At any time, you can create new versions of the content by writing a new transformation file without changing the XML source files. Moreover, you can further separate the style and formatting using CSS. In the future, if a style or layout change to the documents is required, only the style sheets require modification, not the source XML file or XSL transformation file.

Figure 7-12 *Sample CSS File Using Classes*

```
body { background-color: #FFFFFF; }

div.container
{width:100%; margin:0px; border:1px solid gray; line-height:150%;
}

div.header
{
font-family: Times;
font-size: 19px;
text-align: center;
color:white;
background-color:660000;
border:10px solid 0x660000;
line-height: 25px
}

div.footer
{
font-family: Arial;
font-size: 10px;
color:white;
background-color:003333;
border:1px solid white;
margin:0cm 0cm 0cm 0cm
}

div.content
{
border-bottom:1px solid gray;
padding:1em;
}

h1.header
{
font-family: Times, sans-serif;
font-size: 14px;
color: #FFFFFF;
text-align: left;
padding:0; margin:0;
}

h1.title, h1.author
{
font-family: Times, sans-serif;
font-size: 19px;
color: #FFFFFF;
text-align: center;
line-height: 5px}

p.divhead
{
font-family: Times, sans-serif;
font-size: 20px;
color: #FFFFFF;
text-align: left;
}

h2.chaptitle
{
font-family: Arial, sans-serif;
font-size: 14px;
color: #000000 ;
}

h3.goalstitle
{
font-family: Arial, sans-serif;
font-size: 12px;
color: #000000;
line-height: 1px;
margin-left: 1cm
}

p.goal
{
font-family: Times, sans-serif;
font-size: 11px;
border-color: #003333;
background:EEEFEE;
border-style: solid;
border-left-width: 10px;
border-top-width: 1px;
border-bottom-width: 1px;
border-right-width: 1px;
margin-left: 1cm;
padding-left: 15px;
line-height: 18px
}
```

Transforming XML to XSL-FO

Now that you have a solid understanding of XML, XSL, and CSSs, you can tackle the more complex and highly powerful style sheet formatting language called XSL Format Objects (XSL-FO). Like CSS, you can use XSL-FO to format XML data for output. However, unlike CSS, XSL-FO is XML-based, and you can use it to further mark up XML by including descriptive formatting elements. Once marked up with XSL-FO, the formatted XML files can be output into various formats using third-party XSL-FO processors. The output formats can include any of the online display markup languages discussed in this chapter, such as HTML, XHTML, and WML. However, the most common use of XSL-FO is to produce typeset documents for print in Adobe PDF format.

The XSL-FO in Example 7-3 is the general format for an XSL-FO formatted file. Notice that the “fo:” namespace precedes all the XSL-FO elements in the document.

Example 7-3 *The General Format for an XSL-FO Document.*

```
<?xml version="1.0" encoding="ISO-8857-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

<fo:layout-master-set>
  <fo:simple-page-master master-name="A4">
    <!-- Page template goes here -->
  </fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence master-reference="A4">
  <!-- Page content goes here -->
</fo:page-sequence>
</fo:root>
```

The <fo:layout-master-set> element declares the page layout for the document. For your book outline project, you need only a single-page layout. All XSL-FO documents are broken into three areas, **region-before**, **region-body**, and **region-after**, but you can rename them to HEADER, CONTENT, and FOOTER in this example for clarity. Within our page outline, called BOOK-OUTLINE, we define the characteristics of each region. When supplying the content of the page, you reference the outline BOOK-OUTLINE, and the particular regions in which the content will reside.

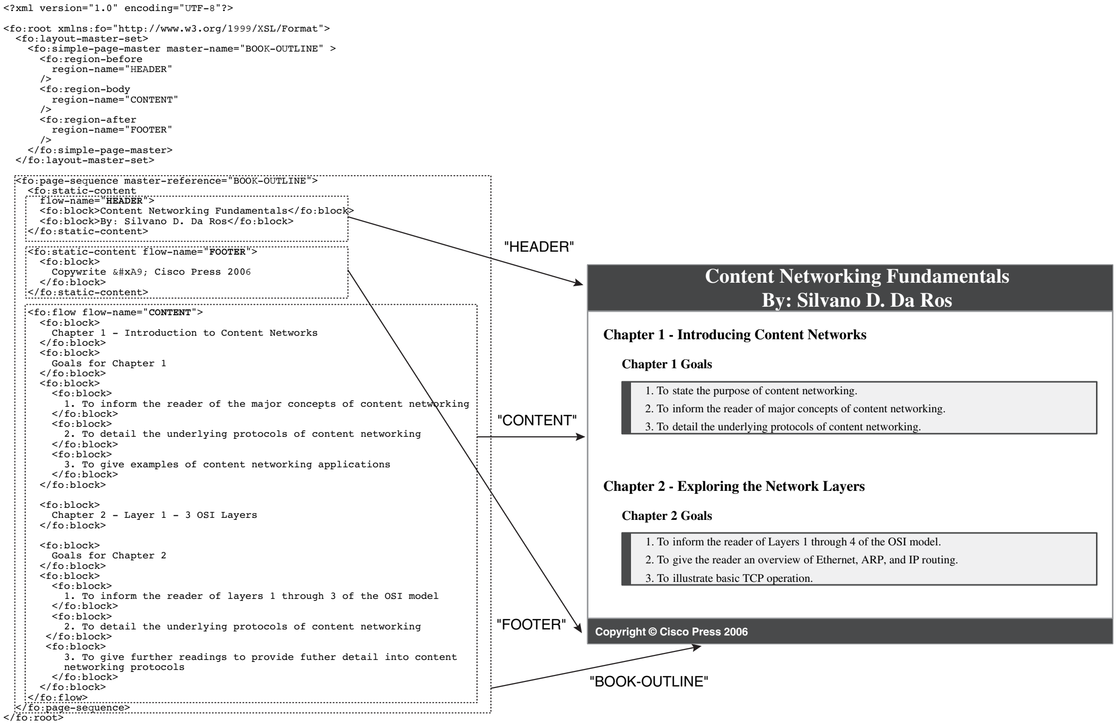
The last part of the document specifies the actual content for output. The **page-sequence** element specifies the format for each page in your output document and references BOOK-OUTLINE for the placement and structure of content on the page. In the book outline example, the HEADER region contains the book title and author name, the CONTENT region contains the book outline, and the FOOTER region contains the copyright information. The HEADER and FOOTER content do not change. As such, you should define the HEADER and FOOTER content with static-content elements. If the data in this example happened to span multiple printed pages, the header and footer data would not change. Conversely, if content is not destined for print as in previous examples in this chapter, the header and footer remain at the top and bottom of the page. For content that changes from page-to-page, use the <fo:flow> element to output the content. The <block> element specifies each area within a flow. Attributes of the <block> element give the specific formatting for each block. For example, the block containing the content for the chapter goals would contain the formatting attributes in Example 7-4.

Example 7-4 *Sample "Chapter Goal" XSL-FO Block*

```
<fo:block
 background-color="#EEEFEE"
 margin="30px"
 border="1px solid #003333"
 border-left="15px solid #003333"
 text-indent="40px"
 line-height="25px"
 font-family="Times"
 font-size="11pt"
 color="black">
</fo:block>
```

The attributes in Example 7-4 produce the indentation, fonts, and colors that you see in the final output in Figure 7-13. To simplify Figure 7-13, none of the format attributes are included in the XSL-FO output. As an exercise, you can add format attributes based on those provided in Example 7-3 and the CSS example in Figure 7-13 to produce the same results.

Figure 7-13 *Sample XML Document Formatted with XSL-FO and Generated into an Adobe PDF Document Using a XSL-FO Processor*



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

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="BOOK-OUTLINE" >
      <fo:region-before
        region-name="HEADER"
      />
      <fo:region-body
        region-name="CONTENT"
      />
      <fo:region-after
        region-name="FOOTER"
      />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="BOOK-OUTLINE">
    <fo:static-content
      flow-name="HEADER">
      <fo:block>Content Networking Fundamentals</fo:block>
      <fo:block>By: Silvano D. Da Ros</fo:block>
    </fo:static-content>

    <fo:static-content flow-name="FOOTER">
      <fo:block>
        Copywrite &#xA9; Cisco Press 2006
      </fo:block>
    </fo:static-content>

    <fo:flow flow-name="CONTENT">
      <fo:block>
        Chapter 1 - Introduction to Content Networks
      </fo:block>
      <fo:block>
        Goals for Chapter 1
      </fo:block>
      <fo:block>
        <fo:block>
          1. To inform the reader of the major concepts of content networking
        </fo:block>
        <fo:block>
          2. To detail the underlying protocols of content networking
        </fo:block>
        <fo:block>
          3. To give examples of content networking applications
        </fo:block>
      </fo:block>

      <fo:block>
        Chapter 2 - Layer 1 - 3 OSI Layers
      </fo:block>

      <fo:block>
        Goals for Chapter 2
      </fo:block>
      <fo:block>
        <fo:block>
          1. To inform the reader of layers 1 through 3 of the OSI model
        </fo:block>
        <fo:block>
          2. To detail the underlying protocols of content networking
       </fo:block>
       <fo:block>
          3. To give further readings to provide futher detail into content
          networking protocols
        </fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Content Networking Fundamentals
By: Silvano D. Da Ros

Chapter 1 - Introducing Content Networks

Chapter 1 Goals

1. To state the purpose of content networking.
2. To inform the reader of major concepts of content networking.
3. To detail the underlying protocols of content networking.

Chapter 2 - Exploring the Network Layers

Chapter 2 Goals

1. To inform the reader of Layers 1 through 4 of the OSI model.
2. To give the reader an overview of Ethernet, ARP, and IP routing.
3. To illustrate basic TCP operation.

Copyright © Cisco Press 2006

In Figure 7-13, the XSL-FO document includes the content from the source XML file. In order to automatically populate the content from a source XML file, you can use the XSLT document in Example 7-5. The XSLT elements from the previous examples are in bold—the parser generates the non-bolded text as the XSL-FO output in Figure 7-13.

Example 7-5 *XSL File That Generates XSL-FO with Embedded XML Content*

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

  <xsl:template match="/">

    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <fo:layout-master-set>
        <fo:simple-page-master master-name="BOOK-OUTLINE" >
          <fo:region-before
            region-name="HEADER" />
          <fo:region-body margin-top="60px"
            region-name="CONTENT"/>
          <fo:region-after extent="30px"
            region-name="FOOTER"/>
        </fo:simple-page-master>
      </fo:layout-master-set>

      <fo:page-sequence master-reference="BOOK-OUTLINE">
        <fo:static-content
          flow-name="HEADER">
          <fo:block><xsl:value-of select="book/@title"/></fo:block>
          <fo:block>By: <xsl:value-of select="book/@author"/></fo:block>
        </fo:static-content>
        <fo:static-content flow-name="FOOTER">
          <fo:block>
            Copyright &#xA9; Cisco Press 2005
        </fo:block>
        </fo:static-content>
       </fo:page-sequence>
        <fo:flow flow-name="CONTENT">
          <xsl:for-each select="book/chapter">
            <fo:block
              <xsl:value-of select="chaptertitle"/>
            </fo:block>
            <fo:block>
              <xsl:value-of select="chaptergoals/@title"/>
            </fo:block>

            <fo:block>
              <xsl:for-each select="chaptergoals/goal">
```

continues

Example 7-5 *XSL File That Generates XSL-FO with Embedded XML Content (Continued)*

```
        <fo:block>
         <xsl:number value="position()" format="1. "/>
         <xsl:value-of select="."/>
        </fo:block>
       </xsl:for-each>
      </fo:block>
     </xsl:for-each>
    </fo:flow>
  </fo:root>
 </xsl:template>
</xsl:stylesheet>
```

Summary

In this chapter, you learned how to present and transform printed and online content using markup languages. Procedural markup languages are inflexible, information retrieval is difficult, and they require multiple documents for files that require different formats. Descriptive markup languages separate document formatting from document content. This chapter discussed the following descriptive markup languages:

- HTML
- Extensible Markup Language (XML)
- Extensible HTML (XHTML)
- Wireless Application Protocol (WAP) Markup Languages including Wireless Markup Language 1.0 (WML) and XHTML Mobile Profile (XHTMLMP).
- You also learned how to transform XML content into XHTML and HTML using XSLT. Once you transform XML into either of these two markup languages, you can format the content for display using CSSs. As an alternative, you learned how to transform XML into XSL-FO. XSL-FO is a standard formatting output with which you can use XSL-FO processors to parse and output to online or printable form.

Review Questions

1. What are the disadvantages of using procedural markup languages?
2. What is PDATA?
3. What is the purpose of the Document Type Definition (DTD) file and XML schemas?
4. What are the benefits of XHTML over HTML?

5. What are your two options for transforming XML content into a displayable or printable form?
6. What is an XML namespace?
7. What is the benefit of CSS?
8. What is the purpose of the **position()** function in the XSLT examples in this chapter?

Recommended Reading

Erik T. Ray, *Learning XML*, O'Reilly, 2003

Thomas Powell*, HTML & XHTML: The Complete Reference*, McGraw-Hill Osborne Media, 2003

Michael Kay, *XSLT 2.0 Programmer's Reference*, Wrox, 2004

http://www.w3c.org

http://www.w3schools.com