CHAPTER 2

XML GRAMMAR

OBJECTIVES

After completing "XML Grammar," you will be able to:

- **Write and read well-formed XML documents.**
- **Use a command-line XML parser to check documents for well-formedness.**
- **Use XML elements and child elements to articulate various kinds of information structures.**
- **Use XML attributes to qualify entities and to articulate more complex data structures precisely.**
- **Use XML entities to get exactly the desired character content for elements and attribute values.**

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Syntax *vs.* Model

- **As discussed in the first chapter, XML as a language allows us to describe information in a precise hierarchical structure.**
- **It is tempting to equate the XML document and its elements with the information they express, but this is not exactly valid.**
- **It's best to think of the XML document as one possible expression of the information – as the most common, readable, and useful expression perhaps, but still not as the information itself.**
- **That said, this chapter is about the XML document: its structure, its syntax, and generally how to use it effectively to describe some body of useful data.**
- **Remember that the content can always be abstracted from the document, and then re-expressed and used in a number of ways:**
  - DOM parsing
  - XSLT transformation
  - Conversion to other persistent forms such as relational databases or files that don't conform to the XML grammar
- **The precise nature of XML content, minus grammar, is treated in the proposed W3C XML Information Set recommendation.**

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Structure of an XML Document

- **A well-formed XML document consists of a series of possible sections:**
  - The **prolog**, which in turn includes the XML declaration (identifying the file as an XML document), a document type declaration, and processing instructions and comments
  - The **body**, which holds all the document content, along with optional comments
  - The **epilog**, which can contain processing instructions and comments, but which is much less frequently included than the other two sections

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Dealership SYSTEM "Cars.dtd">
<?xml-stylesheet type="text/xsl"
  href="CarsOnTheLot.xsl"?>
<!-- End prolog -->

<MyRootElement>
  <FirstChild>
    ...
  </FirstChild>
  ...
</MyRootElement>

<!-- Begin epilog -->
```

- **We'll consider each piece in more detail on the following pages.**

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

XML Declaration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
```

- **An XML document will typically begin with a standard declaration of itself, identifying:**
  - The language in use, which is of course XML, and not for instance HTML
  - `version` of XML language in use
  - An optional character `encoding`, which explains how the document's Unicode character content can be understood from the raw bits and bytes of the document.
  - An optional `standalone` attribute saying that no external document type information is needed to process this document – we'll come back to DTDs and type information in the next chapter
- **The XML declaration is optional, but if included it must occupy the very first line of the document.**
- **Default values for optional attributes:**
  - `encoding` is deduced as either "UTF-8" or "UTF-16" based on the apparent encoding of the first few characters of the document.
  - `standalone` is assumed to be "no".

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Character Content

- **All XML documents can be considered to consist of Unicode characters.**
  - ASCII control characters (numerically encoded in the range 0-31 decimal, 00 to 1F hexadecimal) are mostly not allowed.
  - The exceptions are tab (09 hex), line-feed (0A) and carriage-return (0D).
- **The finer question is that of how these 32-bit characters are encoded for storage or transmission.**
  - Storing all 32 bits of every character can be awfully inefficient if, as is often the case, only the printable ASCII characters (a 7-bit range of values) are in use.
  - Various encoding schemes have been devised, standardized, and used around the world.
- **The XML declaration explains what encoding is in use for this document. The most common values are:**
  - "UTF-8", an 8-bit encoding that uses the high bit to escape to multi-byte representations of non-printable-ASCII characters
  - "UTF-16", a 16-bit encoding that handles the vast majority of characters efficiently but is weak for some Asian languages
  - "ISO-8849-1", known also as "Latin-1".
- **An XML processor or parser is required to support the two UTF encodings mentioned above, and may support a much wider range than that listed here.**

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Case Sensitivity

- **As a general rule, all XML characters are processed and evaluated in a case-sensitive manner.**
  - This takes a lot of getting-used-to for those with a deep background in HTML.
  - Declarations, processing instructions, element start and end tags, attribute names and values are all case-sensitive, so for instance the following is not an XML declaration:

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

- **One exception is that the character encoding value is evaluated without respect for case, so "utf-8" is legal and equivalent to "UTF-8".**
  - This addresses a legacy in the naming of character encodings as standardized by other bodies than the W3C.
  - Note that in processing or parsing document, the value here must be preserved, including case; the exception here, if it is one, concerns the processor's interpretation of the value.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Whitespace

- **Whitespace** characters in XML are these:
  - **space** (20 hex)
  - **tab** (09)
  - **line-feed** (0A)
  - **carriage-return** (0D)
- As in HTML, whitespace characters are generally observed as present or absent between non-whitespace characters; several whitespace characters in a row are treated as just one whitespace character.

Character stream	Equivalent?	Character stream
`ABC`	**No**	`A BC`
`A BC`	**Yes**	`A                BC`
`A BC`	**Yes**	`A` `BC`
`A BC`	**No**	`A B C`

- Note that XML processors are required to **preserve** all whitespace in translating or parsing a document.
- However for most purposes the meaning of a stream of characters is unaffected by the amount or nature of whitespace between tokens.
- The main exceptions are in the character data of elements and attributes, which are assumed to be spelled out literally, including all desired whitespace.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Names and Name Tokens

- The XML recommendation defines the set of legal **name characters** as including the following:
  - Letters
  - Digits
  - Period
  - Hyphen
  - Underscore
  - Colon
- In addition, Unicode **combining characters** and **extenders** are recognized as name characters. (See the XML or Unicode specs for more on this.)
- A **name** then is a string of name characters, with the additional restriction that the first character must be:
  - A letter,
  - An underscore, or
  - A colon.
- Legal names: `Me`, `me`, `my-car`, `me2`, `this.that`
- Illegal names: `me&my-car`, `2pac`, `-other-thing`
- Names are used to identify many higher-level constructs in a document: primarily this means elements and attributes, both of which we'll cover in a moment.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Reserved Names

- **The XML recommendation calls out one reserved form for attribute names.**
- **To wit, only the W3C can define names beginning with "xml:".**
  - Two names are already defined – **xml:lang** and **xml:space**.
  - The whole pattern is off limits, however, so that the W3C has room to add new reserved names in the future.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Character and Entity References

- **XML allows characters that might not be easy to encode directly in the document to be referenced using the character escape sequence `&#`.**
  - What follows the escape sequence must be a decimal number, or a hexadecimal number prefixed with lowercase **`x`**.
  - The reference is closed with a trailing semicolon.

```
A long dash: &#8212;
```

- **Also, a finite set of characters which are easy enough to type, but which are intended to be translated literally instead of being interpreted as XML markup, can be referenced as follows:**

Entity reference	Translated character
`<`	`<`
`>`	`>`
`"`	`"`
`'`	`'`
`&`	`&`

- **Using a document type declaration it is also possible to define custom entities for strings that are used repeatedly in a document.**
  - These are used in the same way as the entities listed above; they just require a specific declaration, whereas those above enjoy special status and have well-known translations.
  - We'll cover this in the next chapter.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Well-Formed XML

- For most of the rest of this chapter we will focus on **well-formed** XML documents.
- A well-formed document conforms to all the basic rules of XML grammar, so for instance:
  - Legal character content
  - Legal names throughout
- Also, the well-formed document will express its content, according to a precise grammar, using constructs such as:
  - **Elements**
  - **Attributes**
  - **Entities**
  - **Processing instructions**
  - **Comments**
- Well-formedness has to do with grammar, not with meaning. It is also an entirely generic rule set, having nothing to do with any particular application.
- In the next chapter we'll consider **valid** XML, a standard which addresses the structured content of an XML document and considers whether it adheres to a specific application's **XML vocabulary**.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Elements

- The body of an XML document consists of a single **root element**.
- An element consists of a start tag, element content, and an end tag.
  - The element **name** is used in the start and end tags.
  - Content can be either character data or a series of child elements.
  - "Mixed content" is also allowed, mixing child elements in with raw characters.

```
<MyElement>content</MyElement>

<Parent>
  <Child>one</Child>
  <Child>two</Child>
</Parent>

<MixedUp>I live in <City>Boston</City></MixedUp>

<NothingToDeclare></NothingToDeclare>
```

- When there is no element content, an alternate syntax is allowed as a shorthand:

```
<NothingToDeclare/>
```

- Elements can also have attributes, which we'll consider more thoroughly a bit later:

```
<SourceFile language="Java">Point.java</SourceFile>
```

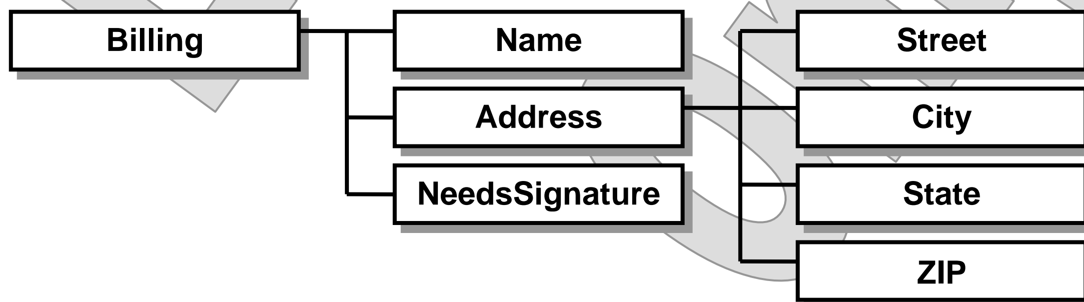
Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Hierarchical Structure

- Elements can contain other elements, ad infinitum.
- Working from the **root element** or **document element**, this establishes the XML document's structure as strictly hierarchical.
  - Each element is a node in a tree.
  - Each node can have zero, one or many children.
  - If a node is a leaf, either it is empty or it has character content only.
  - Mixed content, again, is possible, so a node can have character content as well as child nodes.

```
<Billing>
  <Name>James MacAffrey</Name>
  <Address>
    <Street>45 Vernon Place</Street>
    <City>Austin</City>
    <State>TX</State>
    <ZIP>56789</ZIP>
  </Address>
  <NeedsSignature />
</Billing>
```



© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Evaluation Only

Well-Formed Elements

- **Elements must be balanced with a start and an end tag, which must have the same name.**
  - A parser will not assume one element is closed because it encounters another start tag, even one of the same name.
  - The shorthand for an empty element is considered equivalent to a start and an end tag in sequence.
  - Not well-formed:

```
<Quantity>5689

<Quantity>5689</ >

<Quantity>5689</QUANTITY>

<p>First paragraph
<p>Second paragraph
```

- **Where elements have children, the child start and end tags must be nested inside the parent's content, and cannot overlap sibling tags.**
  - Not well-formed:

```
<P><C1>one<C2>two</C1></C2></P>

<b><i>bold and italicized</b></i>
```

  - Well-formed but probably not the intended meaning (validation would address this):

```
<P><C1>one<C2>two</C2></C1></P>
```

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Expressing Yourself in XML

- **The XML element structure lends itself to expressions of data in many forms:**
- **Some guidelines for articulating information in XML:**
  - Prefer decomposition of complex values where it will not create a great burden to writing or reading, or extraordinarily increase document size:

```
<Name>
  <First>William</First>
  <MiddleInit>W</MiddleInit>
  <Last>Provost</Last>
</Name>
```

  - Express collections of values as a series of child elements:

```
<Member>
  <Nickname>Fritz</Nickname>
  <Skills>
    <Skill>Basketweaving</Skill>
    <Skill>Juggling</Skill>
    <Skill>Eye-rolling</Skill>
  </Skills>
</Member>
```

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

String Literals

- **Another common construct in XML is the string literal, which is a quoted value expected at certain points in the XML grammar.**
- **String literals can contain any legal character content.**
- **They are identified as literals by their enclosure in either single or double quotation marks.**
  - The choice of outer quotation style allows the opposite style to be used as part of the literal string content, so single quotes can be used inside a double-quoted string, and vice-versa.
  - If the outer quotation style must also be employed in the string literal itself, use the appropriate entity reference.
  - Well-formed literals:

```
"OK"
'OK'
"He said, 'OK!'"
'I said, "Yep."'
"He said,
    'Did you say, &quot;Yep.&quot;?'
"
```

  - Ill-formed literals:

```
OK
'OK'?
"He said, "OK!""
```

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Attributes

- In addition to character content and child elements, an element can be qualified using **attributes**.
- An attribute is a named value, expressed in the element start tag as a sequence of tokens, with whitespace allowable between any or all of them:
  - Attribute name
  - Equals sign
  - Attribute value as a string literal
- An element may have zero, one or many attributes, but no two attributes on an element may have the same name.

```
<File language="Java">MyFile.java</File>

<File
  language="C++"
  role = "Header"
>
  YourFile.h
</File>

<File language="Perl" name="HisFile.perl" />
```

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Reserved Attributes

- **There are two reserved attributes which can be used for certain applications and for certain elements.**
- **The `xml:space` attribute can be used to pass a suggestion along to any XML applications that use the document, saying whether that element's whitespace should be preserved or collapsed.**
  - Recall that an XML processor must pass all whitespace to the application untouched.
  - The application, however, might be designed to keep or to discard the exact whitespace characters.
  - This attribute can request one behavior or the other, through one of two legal values: "preserve" or "default".
  - Unlike attributes in general, this attribute, where found, is understood to apply automatically to the qualified element and to all its children, grandchildren, etc.
- **The `xml:lang` attribute identifies the written language used in the qualified element's content.**
  - XML employs Unicode and thus can hold words, characters and phrases in almost any known language.
  - This attribute then is used to allow an XML application to make higher-level decisions about presentation, or to choose between multiple child elements with the same content in different languages.
  - Other application uses of the attribute are possible.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Expressing Yourself, Again

- **Attributes enrich XML significantly.**
- **They compliment element content nicely, too, but there is some confusion over whether to use attributes or child elements to express certain information.**
  - Note the last of the examples on page 18.
  - It expresses the same information model as the first example does, but it uses no element content.
- **There are two schools of thought on which is the more natural and useful way to model data.**
- **In the next chapter we'll delve into the various classes of attributes, each of which allows the attribute value to be interpreted in a different way.**
- **This will indicate that attributes are better for some applications, and likewise we'll see some uses for elements that can't be duplicated using attributes.**
- **Generally, though, there is an overlap in capabilities, and which way a document designer or author goes will be dictated at least to some extent by pure style.**
- **See Anderson, Birbeck, et. al., Professional XML, Chapter 4, "Data Modeling and XML," for a good discussion of this issue.**
  - Note however that this material is predicated on use of DTDs – XML Schema erases many of the issues raised here.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Processing Instructions

- To pass directives or suggestions along to an XML processor or application that are outside the document content, and go beyond the XML document structure and grammar, documents can include **processing instructions**.
- The syntax is

```
<?target [instruction] ?>
```

  - **target** is an XML name and is mandatory. It may not begin with a sequence of characters that is equivalent (ignoring case) to “xml”.
  - **instruction** is any string literal, and is optional.
- Note that the XML declaration shapes up as a valid processing instruction.
- The recommendation says very little of a normative nature about processing instructions and what they might mean.
  - The PI is excluded from the document’s character data in an XML processor’s output.
- Common uses are to reference stylesheets for presentation or transformation.

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

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Comments

- **XML allows comments of the following syntax:**

```
<!--Any comment or whitespace here-->
```

- **Comments can be included anywhere in the document, but cannot interrupt other logical structures:**
  - Document type declaration
  - Processing instructions
  - Elements

```
<!-- Here are some XML comments: -->
<Parent>
  <Child>one</Child><!-- legal -->
  <Child
    attr="value"
    <!-- illegal, interrupts element -->
  >two</Child>
</Parent>
```

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

CDATA Sections

- It is possible to include entirely literal sections of character content in XML documents, using a construct known as the **CDATA section**.

```
<![CDATA[<MyTag>Example of usage</MyTag>]]>
```

- Everything inside the inner pair of square brackets is recognized as literal character data.
  - Markup characters such as < and > will be passed directly by the processor.
  - Entity references are not expanded in CDATA sections.
- CDATA sections are useful for illustrating XML examples inside XML documents.
- CDATA sections are not good choices for embedding binary data in an XML file.
  - The main problem is that somewhere in the binary data, however it is encoded, the character sequence]]> might occur.
  - This would be interpreted as ending the CDATA section, often prematurely, and the rest of the data (and the real end sequence) would not parse.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

XML Parsers

- **An XML Parser is a tool that can check an XML document for well-formedness.**
  - Most parsers available today can also check for **validity**, which we'll consider in the next chapter.
  - Most parsers also provide programmatic interfaces to their parsing behavior.
  - That is, beyond functioning as simple command-line tools, they allow programs in some compatible language to monitor the parsing process, or to act on the document once parsed.
  - These interfaces typically include one or both of the **Simple API for XML** (**SAX**) and the **Domain Object Model** (**DOM**).
- **For exercises in this course, you'll need two tools:**
  - A text editor
  - A validating XML parser

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

XML Tools and Setup

- **The standard XML toolkit provided for this course uses the Xerces parser from the Apache Software Foundation.**
- **Two simple scripts have been prepared to facilitate the use of Xerces from a command console.**
  - To parse a document – checking for well-formedness but not validity – call **parse**:

```
parse MyDocument.xml
```

  - To validate an XML document against a DTD or schema, or to check a schema document itself for validity, call **validate**:

```
validate MyDocument.xml
validate MySchema.xsd
```

- **These are both no-news-is-good-news tools.**
  - Any well-formedness or validity errors will be reported to the console.
- **A simple "XML IDE" can be created by adding these two scripts as custom commands in a text editor.**
  - The standard course setup uses the **Crimson** text editor; the two scripts have been pre-configured on the Tools menu and assigned hotkeys **F9 = parse** and **F10 = validate**.
  - Your instructor can guide you as to configuring other text editors or IDEs that may have been chosen for your classroom.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

XML Repairman

LAB 2

In this lab you will familiarize yourself with legal XML syntax by fixing a number of ill-formed XML documents. Each document has a number of known errors. For each, you will read the document and try to catch all the errors by eye. Enumerate them, and then use an XML parser to check the document. Make changes to the document as indicated by parsing errors until the document checks out as well-formed. Then compare your list of errors to what you discovered in checking the document using the parser.

Detailed instructions are contained in the Lab 2 write-up at the end of the chapter.

Suggested time: 45 minutes.

Evaluation Only

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

SUMMARY

- **The XML grammar can seem a bit odd at first, and terse, especially.**
- **At bottom though it is fairly simple, and for most purposes it is easy to use.**
- **What we've seen so far is a language by which information – data – can be expressed concisely and precisely.**
  - The data being expressed can be of almost any size or shape, although non-textual information generally has to be indexed or referenced from an XML document and to reside elsewhere.
  - Most importantly, with this flexibility in format comes the ability (and, partly, obligation) to **describe** the data as it is being written and read, by way of element tags and attribute names.
  - In the previous chapter we considered a great range of applications for such well-formed, self-describing data.
- **In the next chapter we'll go beyond the grammar and core structure of the well-formed XML document, to consider formal definitions of what a document's data mean, and how a document can be validated as expressing something of the defined meaning.**

© 2002-2005 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.