CHAPTER

Well-Formed XML Documents

In This Chapter

What XML documents are made of

Markup and character data

Well-formed XML in stand-alone documents

Well-formed HTML

HTML 4.0 has about a hundred different tags. Most of these tags have half a dozen possible attributes for several thousand different possible variations. Because XML is more powerful than HTML, you might think you need to know even more tags, but you don't. XML gets its power through simplicity and extensibility, not through a plethora of tags.

In fact, XML predefines almost no tags at all. Instead XML allows you to define your own tags as needed. However these tags and the documents built from them are not completely arbitrary. Instead they have to follow a specific set of rules which we will elaborate upon in this chapter. A document that follows these rules is said to be *well-formed*. Well-formedness is the minimum criteria necessary for XML processors and browsers to read files. In this chapter, you'll examine the rules for well-formed XML documents and well-formed HTML. Particular attention is paid to how XML differs from HTML.

What XML Documents Are Made Of

An XML document contains text that comprises XML markup and character data. It is a sequential set of bytes of fixed length, which adheres to certain constraints. It may or may not be a file. For instance, an XML document may:

- Be stored in a database
- Be created on the fly in memory by a CGI program
- Be some combination of several different files, each of which is embedded in another
- Never exist in a file of its own

However, nothing essential is lost if you think of an XML document as a file, as long as you keep in the back of your mind that it might not really be a file on a hard drive.

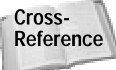
XML documents are made up of storage units called *entities*. Each entity contains either text or binary data, never both. Text data is comprised of characters. Binary data is used for images and applets and the like. To use a concrete example, a raw HTML file that includes `<IMG>` tags is an entity but not a document. An HTML file plus all the pictures embedded in it with `<IMG>` tags is a complete document.

In this chapter, and the next several chapters, I will treat only simple XML documents that are made up of a single entity, the document itself. Furthermore, these documents are only going to contain text data, not binary data like images or applets. Such documents can be understood completely on their own without reading any other files. In other words they stand alone. Such a document normally contains a `standalone` attribute in its XML declaration with the value `yes`, like the one following:

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

External entities and entity references can be used to combine multiple files and other data sources to create a single XML document. These documents cannot be parsed without reference to other files. These documents normally contain a `standalone` attribute in the XML declaration with the value `no`.

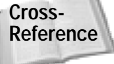
```
<?xml version="1.0" standalone="no"?>
```



External entities and entity references will be discussed in Chapter 9, *Entities and External DTD Subsets.*

Markup and Character Data

XML documents are text. Text is made up of characters. A character is a letter, a digit, a punctuation mark, a space, a tab or something similar. XML uses the Unicode character set, which not only includes the usual letters and symbols from the English and other Western European alphabets, but also the Cyrillic, Greek, Hebrew, Arabic, and Devanagari alphabets. In addition, it also includes the most common Han ideographs for the Chinese and Japanese alphabet and the Hangul syllables from the Korean alphabet. For now, in this chapter, I'll stick to English text.



International character sets are discussed in Chapter 7, *Foreign Languages and Non-Roman Text.*

The text of an XML document serves two purposes, character data and markup. Character data is the basic information of the document. Markup, on the other hand, mostly describes a document's logical structure. For example, recall Listing 3-2, greeting.xml, from Chapter 3, repeated below:

```
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
</GREETING>
```

Here `<?xml version="1.0" standalone="yes"?>`, `<GREETING>`, and `</GREETING>` are markup. `Hello XML!` is the character data. One of the big advantages of XML over other formats is that it clearly separates the actual data of a document from its markup.

To be more precise, markup includes all comments, character references, entity references, CDATA section delimiters, tags, processing instructions, and DTDs. Everything else is character data. However, this is tricky because when a document is processed some of the markup turns into character data. For example, the markup `>` is turned into the greater than sign character (`>`). The character data that's left after the document is processed and all of the markup that stands for particular character data has been replaced by the actual character data it stands for is called parsed character data.

Comments

XML comments are almost exactly like HTML comments. They begin with `<!—` and end with `—>` . All data between the `<!—` and `—>` is ignored by the XML processor. It's as if it wasn't there. Comments can be used to make notes to yourself or to temporarily comment out sections of the document that aren't ready. For example,

```
<?xml version="1.0" standalone="yes"?>
<!—This is Listing 3-2 from The XML Bible—>
<GREETING>
Hello XML!
<!—Goodbye XML—>
</GREETING>
```

There are some rules that must be followed when using comments. These rules are outlined below:

1. Comments may not come before the XML declaration, which absolutely must be the very first thing in the document. For example, the following is not acceptable:

   ```
   <!—This is Listing 3-2 from The XML Bible—>
   <?xml version="1.0" standalone="yes"?>
   <GREETING>
   Hello XML!
   <!—Goodbye XML—>
   </GREETING>
   ```

2. **Comments may not be placed inside a tag. For example, the following is illegal:**

```
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
</GREETING <!—Goodbye—> >
```

3. **Comments may be used to surround and hide tags. In the following example, the `<antigreeting>` tag and all its children are commented out; they are not shown when the document is rendered, as if they don't exist:**

```
<?xml version="1.0" standalone="yes"?>
<DOCUMENT>
<GREETING>
Hello XML!
</GREETING>
<!—
<ANTIGREETING>
Goodbye XML!
</ANTIGREETING>
—>
</DOCUMENT>
```

Since comments effectively delete sections of text, care must be taken to ensure that the remaining text is still a well-formed XML document. For instance, be careful not to comment out a start tag unless you also comment out the corresponding end tag. For example, the following is illegal:

```
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
<!—
</GREETING>
—>
```

Once the commented text is removed what remains is:

```
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
```

Since the `<GREETING>` tag is no longer matched by a closing `</GREETING>` tag, this is no longer a well-formed XML document.

4. **The two-hyphen string (—) may not occur inside a comment except as part of its opening or closing tag. For example, the following is an illegal comment:**

```
<!—The red door—that is, the second one—was left open—>
```

This means, among other things, that you cannot nest comments like this:

```
<?xml version="1.0" standalone="yes"?>
<DOCUMENT>
  <GREETING>
```

```
    Hello XML!
  </GREETING>
<!—
  <ANTIGREETING>
    <!—Goodbye XML!—>
  </ANTIGREETING>
—>
</DOCUMENT>
```

It also means that you may run into trouble if you're commenting out a lot of C, Java, or JavaScript source code that's full of expressions like i— or numberLeft—. Generally it's not too hard to work around this problem once you recognize it.

Entity References

Entity references are markup that is replaced with character data when the document is parsed. XML predefines the five entity references listed in Table 6-1. Entity references are used in XML documents in place of specific characters that would otherwise be interpreted as part of markup. For instance, the entity reference < stands for the less-than sign (<), which would otherwise be interpreted as the beginning of a tag.

Table 6-1
XML Predefined Entity References

Entity Reference	*Character*
&	&
<	<
>	>
"	"
'	'

In XML, unlike HTML, entity references must end with a semicolon. Therefore, > is a correct entity reference and > is not.

Raw less-than signs (<) and ampersands (&) in normal XML text are always interpreted as starting tags and entity references, respectively. (The abnormal text is CDATA sections, described below.) Therefore, less-than signs and ampersands must always be encoded as < and & respectively. For example, you would write the phrase "Ben & Jerry's New York Super Fudge Chunk Ice Cream" as Ben & Jerry's New York Super Fudge Chunk Ice Cream.

Greater-than signs, double quotes, and apostrophes must be encoded when they would otherwise be interpreted as part of markup. However, it's easier just to get in the habit of encoding all of them rather than trying to figure out whether a particular use would or would not be interpreted as markup.

Entity references may also be used in attribute values. For example,

```
<PARAM NAME="joke" VALUE="The diner said,
  &quote;Waiter, There&apos;s a fly in my soup!&quote;">
</PARAM>
```

CDATA

Most of the time anything inside a pair of angle brackets (<>) is markup and anything that's not is character data. However there is one exception. In CDATA sections all text is pure character data. Anything that looks like a tag or an entity reference is really just the text of the tag or the entity reference. The XML processor does not try to interpret it in any way.

CDATA sections are used when you want all text to be interpreted as pure character data rather than as markup. This is primarily useful when you have a large block of text that contains a lot of <, >, &, or " characters, but no markup. This would be true for much C and Java source code.

CDATA sections are also extremely useful if you're trying to write about XML in XML. For example, this book contains many small blocks of XML code. The word processor I'm using doesn't care about that. But if I were to convert this book to XML, I'd have to painstakingly replace all the less-than signs with < and all the ampersands with & as I did in the following:

```
&lt;?xml version="1.0" standalone="yes"?&gt;
&lt;GREETING&gt;
Hello XML!
&lt;/GREETING&gt;
```

To avoid having to do this, I can instead use a CDATA section to indicate that a block of text is to be presented as is with no translation. CDATA sections begin with <![CDATA[and end with]]>. For example:

```
<![CDATA[
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
</GREETING>
 ]]>
```

The only text that's not allowed within a CDATA section is the closing CDATA delimiter]]>. Comments may appear in CDATA sections, but do not act as comments. That is, both the comment tags and all the text they contain will be rendered.

Since `]]>` may not appear in a CDATA section, CDATA sections cannot nest. This makes it relatively difficult to write about CDATA sections in XML. If you need to do this, you just have to bite the bullet and use the `<` and `&` entity references.

CDATA sections aren't needed that often, but when they are needed, they're needed badly.

Tags

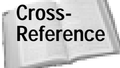
What distinguishes XML files from plain text files is markup. The largest part of the markup is the tags. While you saw how tags are used in the previous chapter, this section will define what tags are and provide a broader picture of how they're used.

In brief, a tag is anything in an XML document that begins with `<` and ends with `>` and is not inside a comment or a CDATA section. Thus, an XML tag has the same form as an HTML tag. Start or opening tags begin with a `<` which is followed by the name of the tag. End or closing tags begin with a `</` which is followed by the name of the tag. The first `>` encountered closes the tag.

Tag Names

Every tag has a name. Tag names must begin with a letter or an underscore (_). Subsequent characters in the name may include letters, digits, underscores, hyphens, and periods. They may not include white space. (The underscore often substitutes for white space.) The following are some legal XML tags:

```
<HELP>
<Book>
<volume>
<heading1>
<section.paragraph>
<Mary_Smith>
<_8ball>
```



Colons are also technically legal in tag names. However, these are reserved for use with namespaces. Namespaces enable you to mix and match tag sets that may use the same tag names. Namespaces are discussed in Chapter 18, *Namespaces*.

The following are not syntactically correct XML tags:

```
<Book%7>
<volume control>
<1heading>
```

```
<Mary Smith>
<.employee.salary>
```

The rules for tag names actually apply to names of many other things as well. The same rules are used for attribute names, ID attribute values, entity names, and a number of other constructs you'll encounter in the next several chapters.

Closing tags have the same name as their opening tag but are prefixed with a / after the initial angle bracket. For example, if the opening tag is `<FOO>`, then the closing tag is `</FOO>`. These are the end tags for the previous set of legal start tags.

```
</HELP>
</Book>
</volume>
</heading1>
</section.paragraph>
</Mary_Smith>
</_8ball>
```

XML names are case sensitive. This is different from HTML where `<P>` and `<p>` are the same tag, and a `</p>` can close a `<P>` tag. The following are *not* end tags for the set of legal start tags we've been discussing.

```
</help>
</book>
</Volume>
</HEADING1>
</Section.Paragraph>
</MARY_SMITH>
</_8BALL>
```

Although both lower- and uppercase letters may be used in XML tags, from this point forward I will mostly follow the convention of making my tags uppercase, mainly because this makes them stand out better in the pages of this book. However, on occasion when I'm using a tag set developed by someone else it will be necessary to adopt that person's case convention.

Empty Tags

Many HTML tags that do not contain data do not have closing tags. For example, there are no `</LI>`, `</IMG>`, `</HR>`, or `</BR>` tags in HTML. Some page authors do

include </LI> tags after their list items, and some HTML tools also use </LI>. However the HTML 4.0 standard specifically denies that this is required. Like all unrecognized tags in HTML, the presence of an unnecessary </LI> has no effect on the rendered output.

This is *not* the case in XML. The whole point of XML is to allow new tags to be discovered as a document is parsed. Thus unrecognized tags may not simply be ignored. Furthermore, an XML processor must be able to determine on the fly whether a tag it's never seen before does or does not have an end tag.

XML distinguishes between tags that have closing tags and tags that do not, called *empty tags*. Empty tags are closed with a slash and a closing angle bracket (/>). For example, <BR/> or <HR/>.

Current Web browsers deal inconsistently with tags like this. However, if you're trying to maintain backwards compatibility, you can use closing tags instead, and just not include any text in them. For example,

```
<BR></BR>

<HR></HR>

<IMG></IMG>
```

When you learn about DTDs and style sheets in the next few chapters, you'll see a couple more ways to maintain backward and forward compatibility with HTML in documents that must be parsed by legacy browsers.

Attributes

As discussed in the previous chapter, start tags and empty tags may optionally contain *attributes*. Attributes are name-value pairs separated by an equals sign (=). For example,

```
<GREETING LANGUAGE="English">
  Hello XML!
  <MOVIE SRC="WavingHand.mov"/>
</GREETING>
```

Here the <GREETING> tag has a LANGUAGE attribute, which has the value *English*. The <MOVIE> tag has a SRC attribute, which has the value *WavingHand.mov*.

Attribute Names

Attribute names are strings that follow the same rules as tag names. That is, attribute names must begin with a letter or an underscore (_). Subsequent letters in the name may include letters, digits, underscores, hyphens, and periods. They may not include white space. (The underscore often substitutes for whitespace.)

The same tag may not have two attributes with the same name. For example, the following is illegal:

```
<RECTANGLE SIDE="8cm" SIDE="10cm"/>
```

Attribute names are case sensitive. The `SIDE` attribute is not the same as the `side` or the `Side` attribute. Therefore the following is acceptable:

```
<BOX SIDE="8cm" side="10cm" Side="31cm"/>
```

However, this is extremely confusing, and I strongly urge you not to write markup like this.

Attribute Values

Attributes values are also strings. Even when the string shows a number, as in the `LENGTH` attribute below, that number is the two characters 7 and 2, not the binary number 72.

```
<RULE LENGTH="72"/>
```

If you're writing code to process XML, you'll need to convert the string to a number before performing arithmetic on it.

Unlike attribute names, there are few limits on the content of an attribute value. Attribute values may contain white space, begin with a number, or contain any punctuation characters (except, sometimes, single and double quotes).

XML attribute values are delimited by quote marks. Unlike HTML attributes, XML attributes *must* be enclosed in quotes. Most of the time double quotes are used. However, if the attribute value itself contains a double quote, then single quotes may be used. For example:

```
<RECTANGLE LENGTH='7"' WIDTH='8.5"'/>
```

If the attribute value contains both single and double quotes, then the one that's not used to delimit the string must be replaced with the proper entity references. I generally just go ahead and replace both, which is always okay. For example:

```
<RECTANGLE LENGTH='8&apos;7&quot;' WIDTH="10&apos;6&quot;"/>
```

Well-Formed XML in Standalone Documents

Although you can make up as many tags as you need, your XML documents do need to follow certain rules in order to be *well-formed*. If a document is not well-formed, most attempts to read or render it will fail.

In fact, the XML specification strictly prohibits XML parsers from trying to fix and understand malformed documents. The only thing a conforming parser is allowed to do is report the error. It may not fix the error. It may not make a best-faith effort to render what the author intended. It may not ignore the offending malformed markup. All it can do is report the error and exit.

Note

The objective here is to avoid the bug-for-bug compatibility wars that have hindered HTML, and made writing HTML parsers and renderers so difficult. Because Web browsers allow malformed HTML, Web page designers don't make the extra effort to ensure that their HTML is correct. In fact, they even rely on bugs in individual browsers to achieve special effects. In order to properly display the huge installed base of HTML pages, every new Web browser must support every nuance, every quirk of all the Web browsers that have come before. Customers would ignore any browser that strictly adhered to the HTML standard. It is to avoid this sorry state that XML processors are explicitly required to only accept well-formed XML.

In order for a document to be well-formed, all markup and character data in an XML document must adhere to the rules given in the previous sections. Furthermore, there are several rules regarding how the tags and character data must relate to each other. These rules are summarized below:

1. **The XML declaration must begin the document.**
2. **Elements that contain data must have both start and end tags.**
3. **Elements that do not contain data and use only a single tag must end with** `/>`**.**
4. **The document must contain exactly one element that completely contains all other elements.**
5. **Elements may nest but may not overlap.**
6. **Attribute values must be quoted.**
7. **The characters** `<` **and** `&` **may only be used to start tags and entity references respectively.**
8. **The only entity references which appear are** `&`, `<`, `>`, `'` **and** `"`.

These eight rules must be adjusted slightly for documents that do have a DTD, and there are additional rules for well-formedness that define the relationship between the document and its DTD, but we'll explore these rules in later chapters. For now let's look at each of these simple rules for documents without DTDs in more detail.

Cross-Reference

DTDs are discussed in Part II.

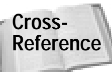
#1: The XML declaration must begin the document

This is the XML declaration for stand-alone documents in XML 1.0:

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

If the declaration is present at all, it must be absolutely the first thing in the file because XML processors read the first several bytes of the file and compare those bytes against various encodings of the string `<?xml` to determine which character set is being used (UTF-8, big-endian Unicode, or little-endian Unicode). Nothing (except perhaps for an invisible byte order mark) should come before this, including white space. For instance, this line is not an acceptable way to start an XML file because of the extra spaces at the front of the line:

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



UTF-8 and the variants of Unicode are discussed in Chapter 7, *Foreign Languages and Non-Roman Text*.

XML does allow you to omit the XML declaration completely. In general, this practice is not recommended. However, it does have occasional uses. For instance, omitting the XML declaration enables you to build one well-formed XML document by combining other well-formed XML documents, a technique we'll explore in Chapter 9. Furthermore, it makes it possible to write well-formed HTML documents, a style we'll explore later in this chapter.

#2: Use Both Start and End Tags in Non-Empty Tags

Web browsers are relatively forgiving if you forget to close an HTML tag. For instance, if a document includes a `<B>` tag but no corresponding `</B>` tag, the entire document after the `<B>` tag will be made bold. However, the document will still be displayed.

XML is not so forgiving. Every start tag must be closed with the corresponding end tag. If a document fails to close a tag, the browser or renderer simply reports an error message and does not display any of the document's content in any form.

#3: End Empty Tags with "/>"

Tags that do not contain data, such as HTML's `<BR>`, `<HR>`, and `<IMG>`, do not require closing tags. However, empty XML tags must be identified by closing with a `/>` rather than just a >. For example, the XML equivalents of `<BR>`, `<HR>`, and `<IMG>` are `<BR/>`, `<HR/>`, and `<IMG/>`.

Current Web browsers deal inconsistently with tags like this. However, if you're trying to maintain backwards compatibility, you can use closing tags instead, and just not include any text in them For example:

```
<BR></BR>
<HR></HR>
<IMG></IMG>
```

Even then, Netscape has troubles with
</BR> (It interprets both as line breaks, rather than only the first.), so unfortunately it is not always practical to include well-formed empty tags in HTML.

#4: Let One Element Completely Contain All Other Elements

An XML document has a root element that completely contains all other elements of the document. This sometimes called the document element instead. Assuming the root element is non-empty (which is almost always the case), it must be delimited by start and end tags. These tags may have, but do not have to have, the name `root` or `DOCUMENT`. For instance, in the following document the root element is `GREETING`.

```
<?xml version="1.0" standalone="yes"?>
<GREETING>
Hello XML!
</GREETING>
```

The XML declaration is not an element. Rather it's a processing instruction. Therefore it does not have to be included inside the root element. Similarly, other non-element data in an XML document like other processing instructions, DTDs, and comments does not have to be inside the root element. But all actual elements (other than the root itself) must be contained in the root element.

#5: Do Not Overlap Elements

Elements may contain (and indeed often do contain) other elements. However, elements may not overlap. Practically, this means that if an element contains a start tag for an element, it must also contain the corresponding end tag. Likewise, an element may not contain an end tag without its matching start tag. For example, the following is acceptable XML:

```
<PRE><CODE>n = n + 1;</CODE></PRE>
```

However the following is not legal XML because the closing </PRE> tag comes before the closing </CODE> tag:

```
<PRE><CODE>n = n + 1;</PRE></CODE>
```

Most HTML browsers can handle this case with ease. However XML browsers are required to report an error for this construct.

Empty tags may appear anywhere, of course. For example,

```
<PLAYWRIGHTS>Oscar Wilde<HR/>Joe Orton</PLAYWRIGHTS>
```

This rule, in combination with Rule 4, implies that for all non-root elements, there is exactly one other element that contains the non-root element, but which does not contain any other element that contains the non-root element. This immediate container is called the *parent* of the non-root element. The non-root element is referred to as the *child* of the parent element. Thus each non-root element always has exactly one parent, but a single element may have an indefinite number of children or no children at all.

Consider Listing 6-1, shown below. The root element is the `DOCUMENT` element. This contains two state children. The first `STATE` element contains four children: `NAME`, `TREE`, `FLOWER`, and `CAPITOL`. The second `STATE` element contains only three children: `NAME`, `TREE`, and `CAPITOL`. Each of these contains only character data, not more children.

Listing 6-1: Parents and Children

```
<?xml version="1.0" standalone="yes"?>
<DOCUMENT>
  <STATE>
    <NAME>Louisiana</NAME>
    <TREE>Bald Cypress</TREE>
    <FLOWER>Magnolia</FLOWER>
    <CAPITOL>Baton Rouge</CAPITOL>
  </STATE>
  <STATE>
    <NAME>Mississippi</NAME>
    <TREE>Magnolia</TREE>
    <CAPITOL>Jackson</CAPITOL>
  </STATE>
</DOCUMENT>
```

In programmer terms, this means that XML documents form a tree. Figure 6-1 shows Listing 5-1's tree structure as well as why this structure is called a tree. It starts from the root and gradually bushes out to the leaves on the end of the tree.

Trees also have a number of nice properties that make them easy for computer programs to read, though this doesn't matter to you as the author of the document.

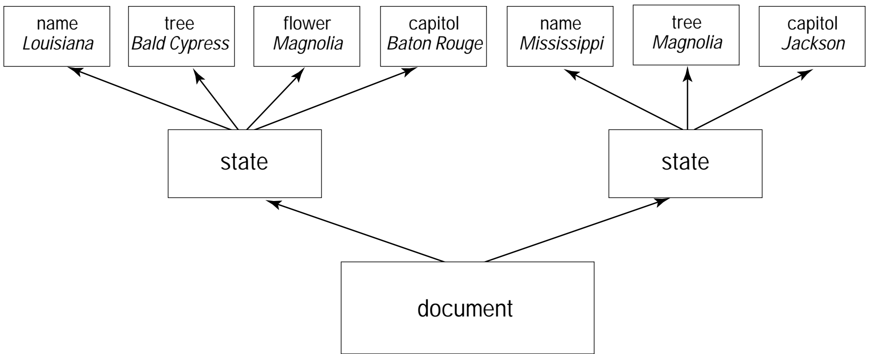


Figure 6-1: Listing 6-1's tree structure

Trees are more commonly drawn from the top down. That is, the root of the tree is shown at the top of the picture rather than the bottom. While this looks less like a real tree, it doesn't affect the topology of the data structure in the least.

#6: Enclose Attribute Values in Quotes

XML requires all attribute values to be enclosed in quote marks, whether or not the attribute value includes spaces. For example:

```
<A HREF="http://metalab.unc.edu/xml/">
```

This isn't true in HTML. For instance, HTML allows tags to contain unquoted attributes. For example, this is an acceptable HTML <A> tag:

```
<A HREF=http://metalab.unc.edu/xml/>
```

The only restriction is that the attribute value must not itself contain embedded spaces.

If an attribute value itself includes double quotes, you may use single quotes to surround the value instead. For example,

```
<IMG SRC="sistinechapel.jpg"
     ALT='And God said, "Let there be light,"
          and there was light'/>
```

If an attribute value includes both single and double quotes, you may use the entity reference ' for a single quote (an apostrophe) and " for a double quote. For example:

```
<PARAM name="joke" value="The diner said,
   &quot;Waiter, There&apos;s a fly in my soup!&quot;">
```

#7: Only Use < and & to Start Tags and Entities

XML assumes that the opening angle bracket always starts a tag, and that the ampersand always starts an entity reference. (This is often true of HTML as well, but most browsers will assume the semicolon if you leave it out.) For example, consider this line,

```
<H1>A Homage to Ben & Jerry's
    New York Super Fudge Chunk Ice Cream</H1>
```

Web browsers will probably display it correctly, but for maximum safety you should escape the ampersand with & like this:

```
<H1>A Homage to Ben &amp; Jerry's New York Super Fudge Chunk
Ice Cream</H1>
```

The open-angle bracket (<) is similar. Consider this common line of Java code:

```
<CODE>     for (int i = 0; i <= args.length; i++ ) { </CODE>
```

Both XML and HTML consider the less-than sign in <= to be the start of a tag. The tag continues until the next >. Thus this line gets rendered as:

```
for (int i = 0; i
```

rather than:

```
for (int i = 0; i <= args.length; i++ ) {
```

The = args.length; i++) { is interpreted as part of an unrecognized tag.

The less-than sign can be included in text in both XML and HTML by writing it as <. For example:

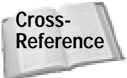
```
<CODE>     for (int i = 0; i &lt;= args.length; i++ ) { </CODE>
```

Well-formed XML requires & to be written as & and < to be written as < whenever they're used as themselves rather than as part of a tag or entity.

#8: Only Use the Five Preexisting Entity References

You're probably familiar with a number of entity references from HTML. For example `©` inserts the copyright symbol ". `®` inserts the registered trademark symbol ". However, other than the five entity references already discussed, XML can only use entity references that are defined in a DTD first.

You don't know about DTDs yet. If the ampersand character `&` appears anywhere in your document, it must be immediately followed by `amp;`, `lt;`, `gt;`, `apos;` or `quot;`. All other uses violate well-formedness.



In Chapter 9, *Entities and External DTD Subsets*, you'll learn how DTDs make it possible to define new entity references that insert particular symbols or chunks of boiler-plate text.

Well-Formed HTML

You can practice your XML skills even before most Web browsers directly support XML by writing well-formed HTML. This is HTML that adheres to XML's well-formedness constraints, but only uses standard HTML tags. Well-formed HTML is easier to read than the sloppy HTML most humans and WYSIWYG tools like FrontPage write. It's also easier for Web robots and automated search engines to understand. It's more robust, and less likely to break when you make a change. And it's less likely to be subject to annoying cross-browser and cross-platform differences in rendering. Furthermore, you can then use XML tools to work on HTML documents, while still maintaining backwards compatibility for readers whose browsers don't support XML.

Real-World Web Page Problems

Real-world Web pages are extremely sloppy. Tags aren't closed. Elements overlap. Raw less-than signs are included in pages. Semicolons are omitted from the ends of entity references. Web pages with these problems are formally invalid, but most Web browsers accept them. Nonetheless, your Web pages will be cleaner, display faster, and be easier to maintain if you fix these problems.

Some of the common problems that Web pages have include the following:

1. Start tags without matching end tags (unclosed elements)
2. End tags without start tags
3. Overlapping elements
4. Unquoted attributes

5. Unescaped <, >, &, and " signs
6. No root element
7. End tag case doesn't match start tag case

I've listed these in rough order of importance. Exact details vary from tag to tag, however. For instance, an unclosed <STRONG> tag will turn all elements following it bold. However, an unclosed <LI> or <P> tag causes no problems at all.

There are also some rules that only apply to XML documents, and that may actually cause problems if you attempt to integrate them into your existing HTML pages. These include:

1. Begin with an XML declaration
2. Empty tags must be closed with a />
3. The only entity references used are &, <, >, ' and "

Fixing these problems isn't hard, but there are a few pitfalls that can trip up the unwary. Let's explore them.

Close All Start Tags

Any element that contains content, whether text or other child elements, should have a start tag and an end tag. HTML doesn't absolutely require this. For instance, <P> , <DT>, <DD>, and <LI> are often used in isolation. However, doing this relies on the Web browser to make a good guess at where the element ends, and browsers don't always do quite what authors want or expect. Therefore it's best to explicitly close all start tags.

The biggest change this requires to how you write HTML is probably thinking of <P> as a container rather than a simple paragraph break mark. For instance, previously you would probably format the opening of the Federalist Papers like this:

```
To the People of the State of New York:
<P>

AFTER an unequivocal experience of the inefficiency of the
subsisting federal government, you are called upon to
deliberate on a new Constitution for the United States of
America. The subject speaks its own importance; comprehending
in its consequences nothing less than the existence of the
UNION, the safety and welfare of the parts of which it is
composed, the fate of an empire in many respects the most
interesting in the world. It has been frequently remarked that
it seems to have been reserved to the people of this country,
by their conduct and example, to decide the important question,
whether societies of men are really capable or not of
```

```
establishing good government from reflection and choice, or
whether they are forever destined to depend for their political
constitutions on accident and force. If there be any truth in
the remark, the crisis at which we are arrived may with
propriety be regarded as the era in which that decision is to
be made; and a wrong election of the part we shall act may, in
this view, deserve to be considered as the general misfortune
of mankind.
<P>
```

Well-formedness requires that it be formatted like this instead:

```
<P>
To the People of the State of New York:
</P>

<P>
AFTER an unequivocal experience of the inefficiency of the
subsisting federal government, you are called upon to
deliberate on a new Constitution for the United States of
America. The subject speaks its own importance; comprehending
in its consequences nothing less than the existence of the
UNION, the safety and welfare of the parts of which it is
composed, the fate of an empire in many respects the most
interesting in the world. It has been frequently remarked that
it seems to have been reserved to the people of this country,
by their conduct and example, to decide the important question,
whether societies of men are really capable or not of
establishing good government from reflection and choice, or
whether they are forever destined to depend for their political
constitutions on accident and force. If there be any truth in
the remark, the crisis at which we are arrived may with
propriety be regarded as the era in which that decision is to
be made; and a wrong election of the part we shall act may, in
this view, deserve to be considered as the general misfortune
of mankind.
</P>
```

You've probably been taught to think of `<P>` as ending a paragraph. Now you have to think of it as beginning one. This does give you some advantages though. For instance, you can easily assign a variety of formatting attributes to a paragraph. For example, here's the original HTML title of House Resolution 581 as seen on `http://thomas.loc.gov/home/hres581.html`:

```
<center>
<p><h2>House Calendar No. 272</h2>

<p><h1>105TH CONGRESS 2D SESSION H. RES. 581</h1>

<p>[Report No. 105-795]
```

```
<p><b>Authorizing and directing the Committee on the
Judiciary to investigate whether sufficient grounds
exist for the impeachment of William Jefferson Clinton,
President of the United States.</b>
</center>
```

Here's the same text, but using well-formed HTML. The `align` attribute now replaces the deprecated `center` element, and a CSS style attribute is used instead of the `<b>` tag.

```
<h2 align="center">House Calendar No. 272</h2>

<h1 align="center">105TH CONGRESS 2D SESSION H. RES. 581</h1>

<p align="center">[Report No. 105-795]</p>

<p align="center" style="font-weight:bold">
Authorizing and directing the Committee on the Judiciary to
investigate whether sufficient grounds exist for the
impeachment of William Jefferson Clinton,
President of the United States.
</p>
```

Delete Orphaned End Tags and Don't Let Elements Overlap

When editing pages, it's not uncommon to remove a start tag and forget to remove its associated end tag. In HTML an orphaned end tag like a `</STRONG>` or `</TD>` that doesn't have any matching start tag is unlikely to cause problems by itself. However, it does make the file longer than it needs to be, the download slower, and has the potential to confuse people or tools that are trying to understand and edit the HTML source. Therefore, you should make sure that each end tag is properly matched with a start tag.

However, more often an end tag that doesn't match any start tag means that elements incorrectly overlap. Most elements that overlap on Web pages are quite easy to fix. For instance, consider this common problem:

```
<B><I>This text is bold and italic</B></I>
```

Since the `I` element starts inside the `B` element, it must end inside the `B` element. All that you need to do to fix it is swap the end tags like this:

```
<B><I>This text is bold and italic</I></B>
```

Alternately, you can swap the start tags instead:

```
<I><B>This text is bold and italic</B></I>
```

On occasion you may have a tougher problem. For example, consider this fragment from the White House home page (http://www.whitehouse.gov/, November 4, 1998). I've emboldened the problem tags to make it easier to see the mistake:

```
<TD valign=TOP width=85>
<FONT size=+1>
<A HREF="/WH/New"><img border=0
src="/WH/images/pin_calendar.gif"
align=LEFT height=50 width=75 hspace=5 vspace=5></A><br> </TD>
<TD valign=TOP width=225>
<A HREF="/WH/New"><B>What's New:</B></A><br>
</FONT>
What's happening at the White <nobr>House - </nobr><br>
 <font size=2><b>
<!— New Begin —>
<a href="/WH/New/html/19981104-12244.html">Remarks Of The
President Regarding Social Security</a>
<BR>
<!— New End —>
 </font>
</b>
</TD>
```

Here the <FONT size=+1> element begins inside the first <TD valign=TOP width=85> element but continues past that element, into the <TD valign=TOP width=225> element where it finishes. The proper solution in this case is to close the FONT element immediately before the first </TD> closing tag; then add a new <FONT size=+1> start tag immediately after the start of the second TD element, as follows:

```
<TD valign=TOP width=85>
<FONT size=+1>
<A HREF="/WH/New"><img border=0
src="/WH/images/pin_calendar.gif"
align=LEFT height=50 width=75 hspace=5 vspace=5></A><br>
</FONT></TD>
<TD valign=TOP width=225>
<FONT size=+1>
<A HREF="/WH/New"><B>What's New:</B></A><br>
</FONT>
What's happening at the White <nobr>House - </nobr><br>
 <font size=2><b>
<!— New Begin —>
<a href="/WH/New/html/19981104-12244.html">Remarks Of The
President Regarding Social Security</a>
<BR>
<!— New End —>
 </font>
</b>
</TD>
```

Quote All Attributes

HTML attributes only require quote marks if they contain embedded white space. Nonetheless, it doesn't hurt to include them. Furthermore, using quote marks may help in the future if you later decide to change the attribute value to something that does include white space. It's quite easy to forget to add the quote marks later, especially if the attribute is something like an ALT in an <IMG> whose malformedness is not immediately apparent when viewing the document in a Web browser.

For instance, consider this <IMG> tag:

```
<IMG SRC=cup.gif WIDTH=89 HEIGHT=67 ALT=Cup>
```

It should be rewritten like this:

```
<IMG SRC="cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup">
```

Escape <, >, and & Signs

HTML is more forgiving of loose less-than signs and ampersands than XML. Nonetheless, even in pure HTML they do cause trouble, especially if they're followed immediately by some other character. For instance, consider this email address as it would appear if copied and pasted from the From: header in Eudora:

```
Elliotte Rusty Harold <elharo@metalab.unc.edu>
```

Were it to be rendered in HTML, this is all you would see:

```
Elliotte Rusty Harold
```

The <elharo@metalab.unc.edu> has been unintentionally hidden by the angle brackets. Anytime you want to include a raw less-than sign or ampersand in HTML, you really should use the < and & entity references. The correct HTML for such a line would be:

```
Elliotte Rusty Harold &lt;elharo@metalab.unc.edu&gt;
```

You're slightly less likely to see problems with an unescaped greater-than sign because this will only be interpreted as markup if it's preceded by an as yet unfinished tag. However, there may be such unfinished tags in a document, and a nearby greater-than sign can mask their presence. For example, consider this fragment of Java code:

```
for (int i=0;i<10;i++) {
  for (int j=20;j>10;j—) {
```

It's likely to be rendered as:

```
for (int i=0;i10;j–) {
```

If those are only two lines in a 100-line program, it's entirely possible you'll miss the omission when casually proofreading. On the other hand, if the greater-than sign is escaped, the unescaped less-than sign will hide the rest of the program, and the problem will be easier to spot.

Use a Root Element

The root element for HTML files is supposed to be `html`. Most browsers forgive your failure to include this. Nonetheless, it's definitely better to make the very first tag in your document `<html>` and the very last `</html>`. If any extra text or markup has gotten in front of `<html>` or behind `</html>`, move it between `<html>` and `</html>`.

One common manifestation of this problem is forgetting to include `</html>` at the end of the document. I always begin my documents by typing `<html>` and `</html>`, then type in between them, rather than waiting until I've finished writing the document and hoping that by that point, possibly days later, I still remember that I need to put in a closing `</html>` tag.

Use the Same Case for All Tags

HTML isn't case-sensitive but XML is. I recommend picking a single convention for tag case, either all uppercase or all lowercase, and sticking to it throughout the document. This is easier than trying to remember the details of each tag. I normally pick all lowercase, because it's much easier to type. Furthermore, the W3C's effort to reformulate HTML as an XML application also uses this convention.

Chapter 20, *Reading Document Type Definitions*, will explore the reformulation of HTML in XML in great detail. However, further exploration will have to wait because that effort uses XML techniques you won't learn for several chapters.

Close Empty Tags with a />.

Empty tags are the ***bête noir*** of converting HTML to well-formed XML. HTML does not formally recognize the XML `<elementname/>` syntax for empty tags. You can convert `<br>` to `<br/>`, `<hr>` to `<hr/>`, `<img>` to `<img/>` and so on quite easily. However, it's a crapshoot whether any given browser will render the transformed tags properly or not.

Do not confuse truly empty elements like `<br>`, `<hr>`, and `<img>` with elements that do contain content but often only have a start tag in standard HTML such as `<p>`, `<li>`, `<dt>`, and `<dd>`.

The simplest solution, and one approved by the XML specification, is to replace the empty tags with start-tag/end-tag pairs with no content. The browser should then ignore the unrecognized end tag. Take a look at the following example,

```
<br></br>
<hr></hr>
<IMG SRC="cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup"></IMG>
```

This seems to work well in practice with one notable exception. Netscape 4.5 and earlier treats </br> the same as <br>; that is, as a signal to break the line. Thus while <br> is a single line break,
</br> is a double line break, more akin to a paragraph mark in practice. Furthermore, Netscape ignores <br/> completely. Web sites that must support legacy browsers (essentially all Web sites) cannot use either
</br> or <br/>. What does seem to work in practice for XML and legacy browsers is the following:

```
<br />
```

Note the space between <br and />. I can't really explain why this works when the more natural variants don't. All I can do is offer it to you as a possible solution if you really care about well-formed HTML.

Use Only the &, <, >, ' and " Entity References

Many Web pages don't need entity references other than &, <, >, ' and ". However, the HTML 4.0 specification does define many more including:

- ™ the trademark symbol (tm)
- © the copyright symbol (c)
- ∞ the infinity symbol ∞
- π the lower case Greek letter pi, π

There are several hundred others. However, using any of these will make your document not well-formed. The real solution to this problem is to use a DTD. We'll discuss the effect DTDs have on entity references in Chapter 9. In the meantime, there are several short- term solutions.

The simplest solution is to write your document in a character set that has all of the symbols you need, then use a <META> directive to specify the character set in use. For example, to specify that your document uses UTF-8 encoding (a character set we'll discuss in Chapter 7 that contains all the characters you're likely to want) you would place this <META> directive in the head of your document:

```
<META http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
```

Alternately, you can simply tell your Web server to emit the necessary content type header. However, it's normally easier to use the <META> tag.

```
Content-Type: text/html; charset=UTF-8
```

The problem with this approach is that many browsers are unlikely to be able to display the UTF-8 character set. The same is true of most other character sets you're likely to use to provide these special characters.

HTML 4.0 supports character entity references just like XML's; that is, you can replace a character by &# and the decimal or hexadecimal value of the character in Unicode. For example:

- `™` the trademark symbol (tm)
- `©` the copyright symbol (c)
- `∞` the infinity symbol ∞
- `π` the lower case Greek letter pi, π

HTML 3.2 only officially supports the numeric character references between 0 and 255 (ISO Latin-1) but 4.0 and later versions of Navigator and Internet Explorer do recognize broader sections of the Unicode set.

If you're really desperate for well-formed XML that's backwards compatible with HTML you can include these characters as inline images. For example:

- `<img src="tm.gif" width="12" height="12" alt="TM"></img>` the trademark symbol (tm)
- `<img src="copyright.gif" width="12" height="12" alt="Copyright"></img>` the copyright symbol (c)
- `img src="infinity.gif" width="12" height="12" alt="infinity"></img>` the infinity symbol ∞
- `<img src="pi.gif" width="12" height="12" alt="pi"></img>` the lowercase Greek letter pi, π

In practice, however, I don't recommend using these. Well-formedness is not nearly so important in HTML that it justifies the added download and rendering time this imposes on your readers.

The XML Declaration

HTML documents don't need XML declarations. However, they can have them. Web browsers simply ignore tags they don't recognize. From their perspective, the following line is just another tag:

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

Since browsers that don't understand XML, don't understand the `<?xml?>` tag, they quietly ignore it. Browsers that do understand XML will recognize this as an indication that this document is composed of well-formed XML, and will be treated as such.

Unfortunately, browsers that halfway understand XML may have troubles with this syntax. In particular, Internet Explorer 4.0 for the Mac (but not Netscape Navigator or other versions of IE) uses this as a signal to download the file rather than displaying it. Consequently I've removed the XML declaration from my Web pages.

Follow the Rules

It is not particularly difficult to write well-formed XML documents that follow the rules described in this chapter. However XML browsers are less forgiving of poor syntax than HTML browsers, so you do need to be careful.

If you violate any well-formedness constraints, XML parsers and browsers will report a syntax error. Thus the process of writing XML can be a little like the process of writing code in a real programming language. You write it, then you compile it, then when the compilation fails, you note the errors reported and fix them.

Generally this is an iterative process in which you go through several edit-compile cycles before you first get to look at the finished document. Despite this, there's no question that writing XML is a lot easier than writing C or Java source code, and with a little practice, you'll get to the point at which you have relatively few errors, and you can write XML almost as quickly as you can type.

HTML Clean-Up Tools

There are several tools that will help you clean up your pages, most notably RUWF (Are You Well Formed?) from XML.COM and HTML Tidy from Dave Raggett of the W3C.

RUWF

Any tool that can check XML documents for well-formedness can test well-formed HTML documents as well. However, one of the easiest tools to use is the RUWF well-formedness checker from XML.COM. Figure 6-2 shows this tester. Simply type in the URL of the page you want to check, and RUWF returns the first several dozen errors on the page.

Here's the first batch of errors RUWF found on the White House home page. Most of these errors are malformed XML, but legal (if not necessarily good style) HTML. However, at least one error ("Line 55, column 30: Encountered </FONT> with no start-tag.") is a problem for both HTML and XML.

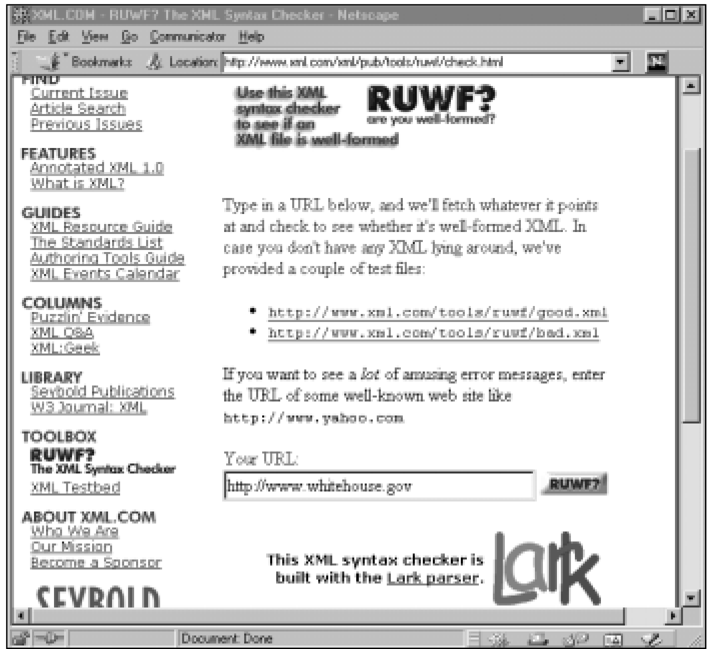


Figure 6-2: The RUWF well-formedness tester

```
Line 28, column 7: Encountered </HEAD> expected </META>
...assumed </META> ...assumed </META> ...assumed </META>
...assumed </META>
Line 36, column 12, character '0': after AttrName= in start-tag
Line 37, column 12, character '0': after AttrName= in start-tag
Line 38, column 12, character '0': after AttrName= in start-tag
Line 40, column 12, character '0': after AttrName= in start-tag
Line 41, column 10, character 'A': after AttrName= in start-tag
Line 42, column 12, character '0': after AttrName= in start-tag
Line 43, column 14: Encountered </CENTER> expected </br>
...assumed </br> ...assumed </br>
Line 51, column 11, character '+': after AttrName= in start-tag
Line 52, column 51, character '0': after AttrName= in start-tag
Line 54, column 57: after &
Line 55, column 30: Encountered </FONT> with no start-tag.
Line 57, column 10, character 'A': after AttrName= in start-tag
Line 59, column 15, character '+': after AttrName= in start-tag
```

HTML Tidy

Once you've identified the problems, you'll want to fix them. Many common problems — for instance, putting quote marks around attribute values — can be fixed automatically. The most convenient tool for doing this is Dave Raggett's

command-line program HTML Tidy. Tidy is a character-mode program written in ANSI C that can be compiled and run on most platforms including Windows, Unix, BeOS and the Mac.

Tidy is on the CD-ROM in the directory utilities/tidy. Binaries are included for Windows NT and BeOS. Portable source is included for all platforms. You can download the latest version from `http://www.w3.org/People/Raggett/tidy/`.

Tidy cleans up HTML files in several ways, not all of which are relevant to XML well-formedness. In fact, in its default mode Tidy tends to remove unnecessary (for HTML, but not for XML) end tags like `</LI>` and make other modifications that break well-formedness. However, you can use the `-asxml` switch to specify that you want well-formed XML output. For example, to convert the file index.html to well-formed XML, you would type from a DOS window or shell prompt:

```
C:\> tidy -m -asxml index.html
```

The `-m` flag tells Tidy to convert the file in place. The `-asxml` flag tells Tidy to format the output as XML.

Summary

In this chapter, you learned how to write well-formed XML. In particular, you learned:

- ✦ XML documents are sequences of characters that meet certain well-formedness criteria.
- ✦ The text of XML documents is divided into character data and markup.
- ✦ Comments can document your code with notes to yourself or to temporarily comment out sections of the document that aren't ready.
- ✦ Entity references allow you to include <, >, &, ", and ' in your document.
- ✦ CDATA sections are useful for embedding text that contains a lot of <, >, and & characters
- ✦ Tags are anything in an XML document that begins with < and ends with >, and are not inside a comment or CDATA section.
- ✦ Start tags and empty tags may contain attributes, which describe elements.
- ✦ HTML documents can also be well-formed with a little extra effort.

In the next chapter, we'll explore how to write XML in languages other than English, in particular in languages that don't look even remotely like English, such as Arabic, Chinese, and Greek.

✦ ✦ ✦