XML

The Extensible Markup Language (XML) is a document processing standard proposed by the World Wide Web Consortium (W3C), the same group responsible for overseeing the HTML standard.

Although the exact specifications have not been completed yet, many expect XML and its sibling technologies to replace HTML as the markup language of choice for dynamically generated content, including nonstatic web pages. Already several browser and word processor companies are integrating XML support into their products.

XML is actually a simplified form of Standard Generalized Markup Language (SGML), an international documentation standard that has been existed since the 1980s. However, SGML is extremely bulky, especially for the Web.

Much of the credit for XMLs creation can be attributed to Jon Bosak of Sun Microsystems, Inc., who started the W3C working group responsible for scaling dtown SGML to a form more suitable for the Internet.

Put succinctly, XML is a meta-language that allows you to create and format your own document markups. With HTML, existing markup is static: <HEAD> and <BODY>, for example, are tightly integrated into the HTML standard and cannot be changed op extended. XML, on the other hand, allows you to create own markup tags and configure each to your liking: for example, <HeadingA>, <Sidebar>, <Quote>, or <ReallyWildFont>.

Each of these elements can be defined through your own document type definitions and stylesheets and applied to one or more XML documents. Thus, it is important to realize to realize that there are no "correct" tags for an XML document, except those you define yourself.

While many XML applications currently support Cascading Style Sheets (CSS), a more extensible stylesheet specification exists called the Extensible Stylesheet Language (XSL). By using XSL you ensure that your XML documents are formatted the same no matter which application or platform they appear on.

Note: The XSL specification is still in flux. There have been several rumors

regarding XSL and the formatting object's portions of the specifications changing dramatically. However, even if XSL changes in the future, the material presented here should give you a firm foundation and enough expertise to make any leap of knowledge much easier.
This material offers a quick overview of XML, as well as some sample applications that allow you to get started in coding. We won't cover everything about XML. We hope that the components that make up XML will seem a little less foreign.

XML Terminology

Before we move further, we need to standardize some terminology. An XML document consists of one or more elements. An element is marked with the following form:

```
<Body>
This is text formatted according to the Body element
</Body>
```

This element consists of two tags, an opening tag which places the name of the element between a less-than sign (<) and a greater than sign (>), and a closing tag which is identical except for the forward slash (/) that appears before the element name. Like HTML, the text contained between the opening and closing tags is considered part of the element and is formatted according to the element's
rules.

Elements can have attributes applied, such as the following:

```
<Price currency="Euro">25.43</Price>
```

Here, the attribute is specified inside of the opening tag and is called "currency." It is given a value of "Euro", which is expressed inside quotation marks. Attributes are often used to further refine or modify the default behavior of an element.

In addition to the standard elements, XML also supports empty elements. An empty element has no text appearing between the opening and closing tag. Hence, both tags can (optionally) be merged together, with a forward slash appearing before the closing marker. For example, these elements aee

identical:

```
<Picture src="blueball.qif"></Picture>
<Picture src "blueball.gif"/>
```

Empty elements are often used to add nontextual content to a document, or to provide additional information to the application that is paprsing the XML. Note that while the closing slash may not be used in single-tag HTML elements, it is mandatory single-tag XML cmpty elements.

Unlearning Bad Habits

Whereas HTML browsers often ignore simple errors in documents, XML applications are not nearly as forgiving. For the HTML reader , there are few bad habits from which we should first dissuade you:

Attribute values must be in quotation marks.

You can't specify an attribute value such as <picture src=/images/blueball.gif>, an error that HTML browsers often overlooked. An attribute value must always be inside single or double quotation marks, or the XML parser will flag it as an error. Here is the correct way to specify such a tag:

```
<picture src="/images/blueball.gif">
```

A non-empty element must have an opening and closing tag.

Each element that specifies an opening tag must have a closing tag that matches it. If it does not, and it is not an empty element, the XML parser generates an error. In other words, you cannot do the following:

```
<Paragraph>
This is a paragraph.
<Paragraph>
This is another paragraph.
```

Instead, you must have an opening and closing tag for each paragraph element:

<Paragraph>This is a paragraph.</Paragraph>
<Paragraph>This is another paragraph.</Paragraph>

Tags must be nested correcly.

It is illegal to do the following:

<Italic><Bold>This is incorrect</Italc></Bold>

The closing tag for the Bold element should be inside the closing tag for the Italic element, to match the nearest opening tag and preserve the correct element nesting. It is essential for the application parsing your XML to process the hierarchy of the elements:

<Italic><Bold>This is correct</Bold></Italic>

These syntactic rules are the source of manv common errors in XML, especially given that some of this behavior can be ignored by HTML browsers. An XML document that adheres to these rules (and a few others which we'll see later) is said to be well-formed.

An Overview of an XML Document

There are generally three files that are processed by an XML-compliant application to display XML content:

The XML document

This file contains the document data typically tagged with meaningful XML elements, some of which may contain attributes.

A stylesheet

The stylesheet dictates how document elements should be formatted when they are displayed, wherether it be in a word processor or a browser. Note that you can applly different stylesheets to the same document, depending on the environment, thus changing its appearance without affecting any of the underlying data. The separation between content and formatting is an important distinction in XML.

Document type definitions (DTD)

This file specifies rules for how the XML document elements, attributes,

and other data are defined and logically related in an XML-compliant document.

A simple XML Document

Example 1 shows a simple XML document.

Example 1. simple.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE OReilly:Books SYSTEM "sample.dtd">
<!-- Here begins the XML data -->
<OReilly:Books xmlns:OReilly='http://www.oreilly.com/'>
  <OReilly:Product>XML Pocket Reference</OReilly:Product>
  <OReilly:Price>8.95</OReilly:Price>
<OReilly:Books>
```

Let's look at this example line by line.
In the first line, the code between the <?xml and the ?> is called an XML declaration. This declaration contains special information for the XML proccessor (the program reading the XML) indicating that this document conforms to Version 1.0 of the XML standard. In addition, the standalone="no" attribute informs the program that an outside DTD is needed to corectly interpret the document. (In this case, the DTD will reside in a separate file called sample.dtd) On a side note, it is possible to simply embed the stylesheet and the DTD in the same file as the XML document. However, this is not recommended for general use, as it hampers reuse of both DTDs and stylesheets.

The second line is as follows:

```
<!DOCTYPE OReilly:Books SYSTEM "sample.dtd">
```

This line points out the root element of the document, as well as the DTD that validates each of the document elements that appear inside the root element. The root element is the outer-most element in the document that the DTD applies to; it typically denotes the document's starting and ending point. In this example, the <OReilly:Books> serves as the root element of the document. The SYSTEM keyword denotes that the DTD of the document resides in a separate local file named sample.dtd.

Following that line is a comment. Comments always begin with <!-- and end with -->. You can write whatever you want inside comments; they are ignored by the XML processor. Be aware that comments, however, cannot come before the XML declaration and cannot appear inside of an tag. For example, this is illegal:

<OReilly:Books <!-- This is the tag for a book>>

Finally, <OReilly:Product>, <OReilly:Price>, and <OReilly:Books> are XML elements we invented. Like most elements in XML, they hold no special significance except for whatever document and style rules we define for them. Note that these elements look slightly different than those you may have seen previously because we are using namespaces. Each element tag can be divided into two parts. The portion before the colon (:) forms the tag's namespace; the portion after the colon identifies the name of the tag itself.

Let's discuss some XML terminology: the <OReilly:Product> and <OReilly: Price> elements would consider the <OReilly:Books> element their parent. In the same manner, elements can be grandparents and grandchildren of other elements. However, we typically abbreviate multiple levels by stating that an element is either an ancestor or a descendant of another element.

Namespaces

Namespaces are a recent addition to the XML specification. The use of namespaces is not mandatory in XML, but it's often wise. Namespaces were created to ensure uniqueness among XML elements.

For example, let's pretend that the <OReilly: Books> element was simply named <Books>. When you think about it, it's not out of the question that another publisher would create its own <Books> element in its own XML documents. If the two publishers combined their documents, resolving a single (correct) definition for the <Books> tag would be impossible. When two XML documents containing identical elements from different sources are merged, those elements aec said to collide. Namespaces help to avoid element collisions by scoping each tag.

In Example 1, we scoped each tag with the OReilly namespace. Namespaces

are declared using the xmlns: something attribute, where something defines the ID of the namespace. The attribute value is a unique identifier that differentiates it from all other namespaces; the use of a URL is recommended.

In this case we use the O'Reilly URL http://www.oreilly.com/ as the default namespace, which should guarantee uniqueness. A namespace declaration can appear as an attribute of any element, so long as the namespace's use remains inside that element's opening and closing tags. Here are some examples:

```
<OReilly:Books
   xmlns:OReilly='http://www.oreilly.com/'>
    <xsl:stylesheet xmlns:xsl='http://www.w3.org/'>
```

You are allowed to define more than one namespace in the context of an element:

```
<OReilly:Books xmlns:OReilly='http://www.oreilly.com/'
   xmlns:Songline='http://www.songline.com/'>
   </OReilly:Books>
```

If you do not specify a name after the xmlns prefix, the namespace is dubbed the default namespace and is applied to all elements inside the defining element that do not use a numespace prefix of their own. For example:

```
<Books xmlns='http://www.oreilly.com/'
xmlns:Songline='http://www.songline.com/'>
<Book>
<Title>XML Pocket Reference</Title>
<ISBN>1-56592-709-5</ISBN>
</Book>
<Songline:CD>18231</Songline:CD>
</Books>
```

Here, the default namespace (represented by the URL http://www.oreilly.com/) is applied to the elements <Books>. <Book>, <Title>, and <ISBN>. However, it is not applied to the <Songline:CD> element, which has its own namespace.

Finally, you can set the default namespace to an empty string to ensure that there is no default namespace in use within a specific element:

```
<header xmlns=' '
xmlns:OReilly='http://www.oreilly.com/'
xmlns:Songline='http://www.songline.com/'>
<entry>Learn XML in a Week</entry>
<price>10.00</price>
</header>
```

Here, the <entry> and <price> elements have no default namespace.

A Simple Document Type Definition (DTD)

Example 2 creates a simple DTD for our XML document.

Example 2. simple.dtd

```
<!--DTD for sample document -->
<!ELEMENT OReilly:Books (OReilly:Product, OReilly:Price)>
<!ELEMENT OReilly:Product (#PCDATA)>
<!ELEMENT OReilly:Price (#PCDATA)>
```

The purpose of the this DTD is to declare each of the elements used in our XML document. All document-type data is placed inside a construct with the characters <!something>. Like the previous XML example, the first line is a comment because it begins with <!-- and ends with -->.

The < !ELEMENT> construct declares each valid element for our XML document. With the second line, we've specified that the OReilly: Books element is valid:

```
<!ELEMENT OReilly:Books
    (OReilly:Product, OReilly:Price)>
```

The parentheses group required child elements for the element <OReilly:Books>. In this case, the element <OReilly:Product> and the element <OReilly:Price> must be included inside our <OReilly:Books>

element tags, and they must appear in the order specified. The elements <OReilly:Product> and <OReilly:Price> are children of <OReilly: Books>.

Likewise, both the <OReilly:Product> element and the <OReilly: Price> element are declared in our DTD:

```
<!ELEMENT OReilly:Product (#PCDATA)>
<!ELEMENT OReilly:Price (#PCDATA)>
```

Again parentheses specify required elements. In this case, They both have a single requirement, which is represented by #PCDATA. This is shorthand for parsed character data, which means that any characters are allowed, so long as they do not include other element tags or contain the characters < or &, or the sequence]] >. These characters are forbidden because they could be interpreted as markup. (We'll see how to get around this shortly.)

The XML data shown in Example 1 adheres to the rules of this DTD: it contains an <OReilly:Books> element, which in turn contains an <OReilly: Product> element, followed by an <OReilly:Price> element inside it (in that order). Therefore, if this DTD is applied to it with a <!DOCTYPE> ,statement, the document is said to be valid.

So far, we've structured the data, but we haven't paid much attention to its formatting. Now let's move on and add some some style to our XML document.

A Simple XSL Stylesheet

The Extensible Stylesheet Language consists of a series of markups that can be used to apply formatting rules to each of the elements inside an XML document. XSL works by applying various style rules to the contents of an XML document, based on the elements that it encounters.

(As we mentioned earlier, the XSL specification is changing as we speak, and will undoubtly change after this book is printed. So while you can use the XSL information in this book to develop a conceptual overview of XSL, we recommend referring to http://www.w3.org for the latest XSL specification.)

Let's add a simple XSL stylesheet to the example:

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl "http://www.w3.org/TR/WD-xsl"
xmlns:fo="http://www.w3.org/TR/WD-xsl/FO">
<xsl:template match="/">
<fo:block font-size="18pt">
<xsl:apply-templates/>
</fo:block>
</xsl:template>
</xsl:stylesheet>
```

The first thing you might notice when you look at an XSL stylesheet is that it is formatted in the same way as a regular XML document. This is not a coincidence. In fact, by design XSL stylesheets are themselves XML documents, so they must adhere to the same rules as well-formed XML documents.

Breaking down the pieces, you should first note that all the XSL elements must be enclosed in the appropriate <xsl: stylesheet> tags. These tags tell the XSL, processor that it is describing stylesheet information, not XML content itself. Between the <xsl : stylesheet> tags lie each of the rules that will be applied to our XML document. Each of these rules can be further broken down into two items: a template pattern and a template action.

Consider the line:

```
<xsl:template match="/".>
```

This line forms the template pattern of the stylesheet rule. Here, the target pattern is the root element, as designated by match=" / ". The " / " is shorthand to represent the XML document's root element (<OReilly:Books> in our case).

Remember that if this stylesheet is applied to another XML document, the root element matched might be different.

The following lines:

```
<fo:block fonrsize="18pt">
```

```
    <xsl:apply-templates/>
  </fo:block>
```

specify the template action that should be performed on the target. In this case, we see the empty element <xsl:apply-templates/> located inside a <fo:block> element. When the XSL processor formats the target element, every element that is inside the root element's opening and closing tags uses an 18-point font.

In our example, the <OReilly:Product> element and the <OReilly:Price> element are enclosed inside the <OReilly:Books> tags. Therefore, the font size will be applied to the contents of those tags.

Example 3 displays a more realistic example.

Example 3. simple.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl "http://www.w3.org/TR/WD-xsl"
  xmlns:fo="http://www.w3.org/TR/WD-xsl/FO">
  xmlns:OReilly="http://www.oreilly.com/">
  <xsl:template match="/">
  <fo:display-sequence>
  <xsl:apply-templates/>
  </fo:display-sequence>
  </xsl:template>
  <xsl:template match="OReilly:Books">
  <fo:block font-size="18pt">
  <xsl:text>Books:</xsl:text,
  <xsl:apply-templates/>
  </fo:block>
  </xsl:template>
  <xsl:template match="OReilly:Product">
  <fo:block font-size="12pt">
  <xsl:apply-templates/>
```

Example 3. simple.xsl (continued)

```
    </fo:block>
  </xsl:template>
```

```
  <xsl:template match="OReilly:Price">
       <fo:block font-size="14Pt">
<xsl:text>Price: $</xsl:text>
<xsl:apply-templates/>
<xsl:text> + tax</xsl:text>
</fo:block>
</xsl:template>
</xsl:stylesheet>
```

In this example, we're now targeting the <OReilly: Books> element, printing the word "Books:" before it in an 18-point font. In addition, the <OReilly:Product> element now applies a 12-point font to each of its children, and the <OReilly:Price> tag now uses a 14-point font to display its children, overriding the default 18-point font of its parent, <OReilly:Books>. (Of course, neither one has any children elements; they simply have text between their tags in the XML document.) The text "Price: $" will now precede each of <OReilly:Price>'s children, and the characters " + tax" will now come after it, formatted accordingly.*

Here is the result after we pass simple.xsl through an XSL processor:

```
    <?xml version="1.0">
    <fo:display-sequence>
      <fo:block font-size="18pt">
    Books:
       <fo:block font-size="12pt">
    XML Pocket Reference
       </fo:block>
       <fo:block font-size="14pt">
```

You may have noticed that we are using the <fo:display-sequence> element instead of <fo:block> for the root element. This is primarily because the pattern that matches our root element really doesn't do anything anymore. However, you needn't be concerned with this here.

```
    Price $8.95 + tax
       </fo:block>
     </fo:block>
    </fo:display-sequence>
```

And that's it: everything needed for a simple XML document! Running it through an XML processor, you should see something similar to Figure 1.

Books:
XML Pocket Reference
Price $8.95 + tax

Figure 1. Sample XML output

XML Reference

An overview of the more common rules and constructs of the XML language.

Well-Formed XML

These are the rules for a well-formed XML document:

- The document must either use a DTD or contain an XML declaration with the standalone attribute set to "no". For example:

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

- All element attribute values must be in quotation marks.
- An element must have both an opening and closing tag, unless it is an empty element.

- If a tag is a standalone empty element, it must contain a closing slash (/) before the end of the tag.

- An opening and closing element tags must nest correctly.

- Isolated markup characters are not allowed in text: < or & must use entity references instead. In addition, the sequence]]> must be expressed as]]> when used as regular text. (Entity references are discussed in further detail later.)

- Well formed XML documents without a corresponding DTD must have all attributes of type CDATA by default.

XML Instructions

The following XML instructions are legal:

<?xml ... ?>

<?xml version="number" [encoding="encoding"] [standalone="Yes/no"] ?>

Although they are not required to, XML documents typically begin with an XML declaration. An XML declaration must start with the characters <?xml and end with the characters ?>.

Attributes

version

The version attribute specifies the correct version of XML required to process the document, such as "1.0". This attribute cannot be omitted.

encoding

The encoding attribute specifies the character encoding used in the document (e.g., "US-ASCII" or "iso-8859 - 1). This attribute is optional.

standalone

The optional standalone attribute specifies whether a DTD is required to parse the document. The value must be either yes or no. If the value is no, a DTD must be declared with an XML < ! DOCTYPE> instruction.

For example: <?xml version "1.0"?>
<?xml version = "1.0" encoding "US-ASCII" standalone "no"?>

<!DOCTYPE>

<!DOCTYPE root-element SYSTEM | PUBLIC [name] URI-of-DTD>

The < ! DOCTYPE> instruction allows you to specify a DTD for an XML document. This instruction can currently take one of two forms:

<!DOCTYPE root-element: SYSTEM "URI of DTD">

<!DOCTYPE root-element: PUBLIC "name" "URI of DTD">

Keywords

SYSTEM

The SYSTEM variant specifies the URI location of a DTD for private use in the document. The DTD is applied to all elements inside of root-document. For example:

<!DOCTYPE <Book> SYSTEM "http://mycompany.com/dtd/mydoctype.dtd">

PUBLIC

The PUBLIC variant is used in situations where a DTD has been publicized for widespread use. In those cases, the DTD is assigned a unique name, which the XML processor may use by itself to attempt to retrieve the DTD. If that fails, the URI is used:

<!DOCTYPE <Book> PUBLIC "-//O'Reilly//DTD//EN"
"http://www.oReilly.com/dtd/xmlbk.dtd">

Public DTDs follow a specific naming convention. See the XML specification for dctails on naming public DTDs.

<? ... ?>

<?target attribute1 = "value" attribute2="value"..... ?>

A processing instruction allows developers to place information specific to an outside application within the document. Processing instructions always begin with the characters <? and end with the characters ?>. For example:

<?works document "hello.doc" data "hello.wks"?>

You can create your own processing instructions if the XML application processing the document is aware of what the data and acts accordingly.

CDATA

<! [CDATA]] >

You can defme special marked sections of character data or CDATA, which the XML processor will not attempt to interpret as markup. Anything that is included inside a CDATA marked section is treated as plain text.

CDATA marked sections begin with the characters
<! [CDATA [and end with the characters]] >. For example:

```
<! [ CDATA [
  I'm now discussing the <element> tag of documents 5 & 6: "Sales" and
  "Profit and Loss".  Luckily, the XML processor won't apply rules of
  formatting to these sentences!
] ] >
```

Note that you may not use entity references inside a CDATA marked section, as they will not be expanded.

<!-- …-->
<!-- comments -->

You can place comments anywhere in an XML document, except within element tags or before the initial XML processing instructions. Comments in an XML document always start with the characters < ! -- and end with the characters -->.

In addition they may not include double hyphens within the comment. The contents of the comment are ignored by the XML processor:

```
<!-- Sales Figures Start Here -->
<Units>2000</Units>
<Cost>49.95</Cost>
```

Element and Attribute Rules

An element is either bound by its starting and ending tags, or is an empty element. Elements can contain text, other elements or a combination of both. For example:

```
<para> Elements can contain text, other elements, or a combination. For
```

example, a chapter might contain a title and multiple paragraphs, and a paragraph might contain text and <emphasis> emphasis elements </emphasis> : </para>

An element name must start with a letter or an underscore. It can then have any number of letters, numbers, hyphens, periods, or underscores in its name. Elements are case-sensitive: <Para>, <para>, and <pArA> are considered three different element types.

EIement type names may not start with the string ***xml***, in any combination of upper or lowercase. Names beginning with ***xml*** are reserved for special uses by the W3C XML Working Group. Collons are permittcd in element type names only for specifying namespaces; otherwise, colons are forbidden. For example:

<Itallic>	Legal
<_Budget>	Legal
<Punch line>	illegal: has a space
<205Para>	illegal: starts with number
<repair@log>	illegal: contains @ character
<xml>	illegal: starts with xml

Element type names can also include accented Roman characters, letters from other alphabets (e.g. Cyrillic, Greek, Hebrew, Arabic, Thai, hiragana, katakana, or Devanagari), and ideograms from the Chinese, Japanese, and Korean languages.

It you are using a DTD, the content of an element is constrained by its DTD dcclaration. Better XML applications inform you what elements and attributes can appear inside a specific element. Otherwise, you should check the element declaration in the DTD to determine the exact semantics.

Attributes describe additional information about an element. They always consist of a name and a value, as follows:

<price currency "Euro">

The attribute vaIue is aIways quoted, using either single or double quotes. Attribute names are subject to the same restrictions as element type names.

XML Reserved Attributes

The following are reserved attributes in XML.

xml:lang

xml:lang="iso_639_identifier"

The xml: lang attribute can be used on any element. Its value indicates the language of that element. This is useful in a multilingual context. For example, you might have:

<para xml:lang="en">Hello</para>
<para xml:lang="fr">Bonjour</para>

Thisis format allows you to display one or the other, depending
on the users language preference.

The syntax of the xml : lang value is defined by RFC 1766, available at http:ds0.internic.net/rfc1766.txt. A two-letter language code is optionally followed by a hyphen and a twoletter country code. The languages are defined byRFC 1766 atid the countries are defined by ISO 3166. Traditionauy, the language is given in lowercase and the colintry in uppercase (and for safety, this rule should be followed), but processorts are expected to use the values in a case-insensitive manner.

In addition, RFC 1766 also porovides extensions for nonstandartized languages or language variants. Valid xml : lang values include such notations as en, en-US, en-UK, en-cockney. i-navajo, and x-minbari.

xml:space

xml:space="default|preserve"

The xml: space attribbute incticates whether tny whitespace inside the element is significant and should not be altered by the XML processor. The attribute can take one of two enumerted values:

preserve
The XML application honors all whitespace (newlines, spaces, and tabs) present within the element.

default
: The XML processor is free to do whatever it wishes with the whitespace inside the element.

You should set xml:space to preserve only if you have an element you wish to behave similar to the HTML <pre> element, such as documenting source code.

xml:link

xml:link='link_type"

The xinl:link attribute signals an Xlunk processor that an element is a link element. It can take one of the following

values:

simple
: A one-way link, pointing to the area in the target document where the referenced document occurs.

document
: A link that points to a member document of an extended link group.

extended
: An extendect link, which can point to more than one target through the use of multiple locators. Extended links can also support multidirectiotial and out-of-line links (a listing of links stored in a separate document).

group
: A link that contains a group of document links.

The xml: link attribute is always used with other attributes to form an Xlink. See the section "XLink and XPointer" for much more information on the xml : link attribute. This section also has more information on attribute remapping.

(Note that this attribute may change to xlink:form in the future.)

xml:attribute

xml:attribute="existing-attribute
replacement attribute"

The xml :attribute attribute allows you to remap attributes to prevent conflict with other potential uses of Xlink attributes. For example:

```
<Person title="Reverend" title-abbr="Rev."
  given="Kirby" family= "Hensley"
  href="http://www.ulc.org/"
  link-title "Universal Life Church"
  xml:attributes="title link-title"/>
```

In this example, since the title attribute is already taken, the xml:attributes attribute remaps it to use link-
title instead.

See the section "Xlink and Xpointer" for more information on attribute remapping. (Note that this attribute may change to
xlink: attribute in the future.)

Entity References

Entity references are used as substitutions for specific characters in XML. A common use for entity references is to denote document symbols that might otherwise be mistaken for markup by an XML processor. XML predefines five entity references for you, which are substitutions for basic markup symbols. However, you can define as many entity references as you like in your own DTD. (See the next section.)

Entity references always begin with an ampersand (&) and
end with a semicolon (;). They cannot appear inside CDATA sections, but can be used anywhere else. Predefined entities defined in XML are shown in Table 1.

Table 1. Predefined Entities in XML.

Entity	Char	Notes
&amp;	&	Do not use inside processing instructions
&lt;	<	Use inside attribute vakues quoted with "
&gt;	>	Use after]] in normal text and inside processing instructions
&quot;	"	
&apos;	'	Use inside attribute values quoted with '

In addition, you can provide character references for Unicode characters by using a hexadecimal character reference. This consists of the string &#x followed by the hexadecmal number representing the character, and finally a semicolon (;). for example, to represent mc copyright character, you could use the following:

This document is © 1999 by O'Reilly and Assoc.

The entity reference is replaced with the "circled-C" copyright character when the document is formatted.