XML and DTDs
by Jacob Cleary
2004

Welcome to XML & DTDs

This tutorial assumes that you have been introduced to the possibilities of XML and want to learn more about the nuts and bolts of creating an XML document or DTD.

If you're unsure of what exactly XML is, we encourage you to look over the Introduction to XML tutorial.

Objectives:

This tutorial aims to show you how to create both a "well formed" XML document and a DTD that validates the XML.

Explain what "well formed" and valid mean when talking about XML and describe anatomy and structure of XML.

Take you through why you would want to create a DTD, the steps of creating a DTD, and some examples of DTDs currently used.

Note: All the example DTD and XML files used in this tutorial are available in zipped format for download.

Definitions:

When talking about XML, here are some terms that would be helpful:

- **XML:** e**X**tensible **M**arkup **Language**, a standard created by the W3Group for marking up data.
- **DTD:** **D**ocument **T**ype **D**efinition, a set of rules defining relationships within a document; DTDs can be "internal" (within a document) or "external" (links to another document).
- **XML Parser:** Software that reads XML documents and interprets or "parse" the code according to the XML standard. A parser is needed to perform actions on XML, such as comparing an XML document to a DTD.

XML Anatomy

If you have ever done HTML coding, creating an XML document will seem very familiar. Like HTML, XML is based on SGML, Standard Generalized Markup Language, and designed for use with the Web. If you haven't coded in HTML before, after creating an XML document, you should find creating HTML documents easy.

Note: If you are interested in learning HTML, please visit one of our following HTML tutorials:

- Basic HTML
- Even More HTML

XML documents, at a minimum, are made of two parts: the prolog and the content. The prolog or head of the document usually contains the administrative metadata about the rest of document. It will have information such as what version of XML is used, the character set standard used, and the DTD, either through a link to an external file or internally. Content is usually divided into two parts, that of the structural markup and content contained in the markup, which is usually plain text.

Let's take a look at a simple prologue for an XML document:

<?xml version="1.0" encoding="iso-8859-1"?>

<?xml declares to a processor that this is where the XML document begins.

version="1.0" declares which recommended version of XML the document should be evaluated in.

encoding="iso-8859-1" identifies the standardized character set that is being used to write the markup and content of the XML.

Note: XML currently has two versions out: 1.0 and 1.1. For more information, visit the W3C group, which developed the XML standard. This tutorial deals with primarily with XML version 1.0.

Note: For more information about standard character sets, see http://www.iana.org/assignments/character-sets

The structural markup consists of elements, attributes, and entities; however, this tutorial will primarily focus on elements and attributes.

Elements have a few particular rules:

1. Element names can be any mixture of characters, with a few exceptions. However, element names are case sensitive, unlike HTML. For instance, <elementname> is different from <ELEMENTNAME>, which is different from <ElementName>.

Note: The characters that are excluded from element names in XML are `&, <, ", and >,` which are used by XML to indicate markup. The character **:** should be avoided as it has been used for special extensions in XML. If you want to use these restricted characters as part of the content within elements but do not want to create new elements, then you would need to use the following entities to have them displayed in XML:

XML Entity Names for Restricted Characters

Use	For
&amp;	&
&lt;	<
&gt;	>
&quot;	"

2. Elements containing content must have closing and opening tags.

<elementName> (opening) </elementName> (closing)

Note that the closing tag is the exact same as the opening tag, but with a backslash in front of it.

The content within elements can be either elements or character data. If an element has additional elements within it, then it is considered a parent element; those contained within it are called child elements.

For example,

<elementName>This is a sample of <anotherElement> simple XML</anotherElement>coding</elementName>.

So in this example, <elementName> is the parent element. <anotherElement> is the child of elementName, because it is nested within elementName.

Elements can have attributes attached to them in the following format:

<elementName attributeName="attributeValue" >

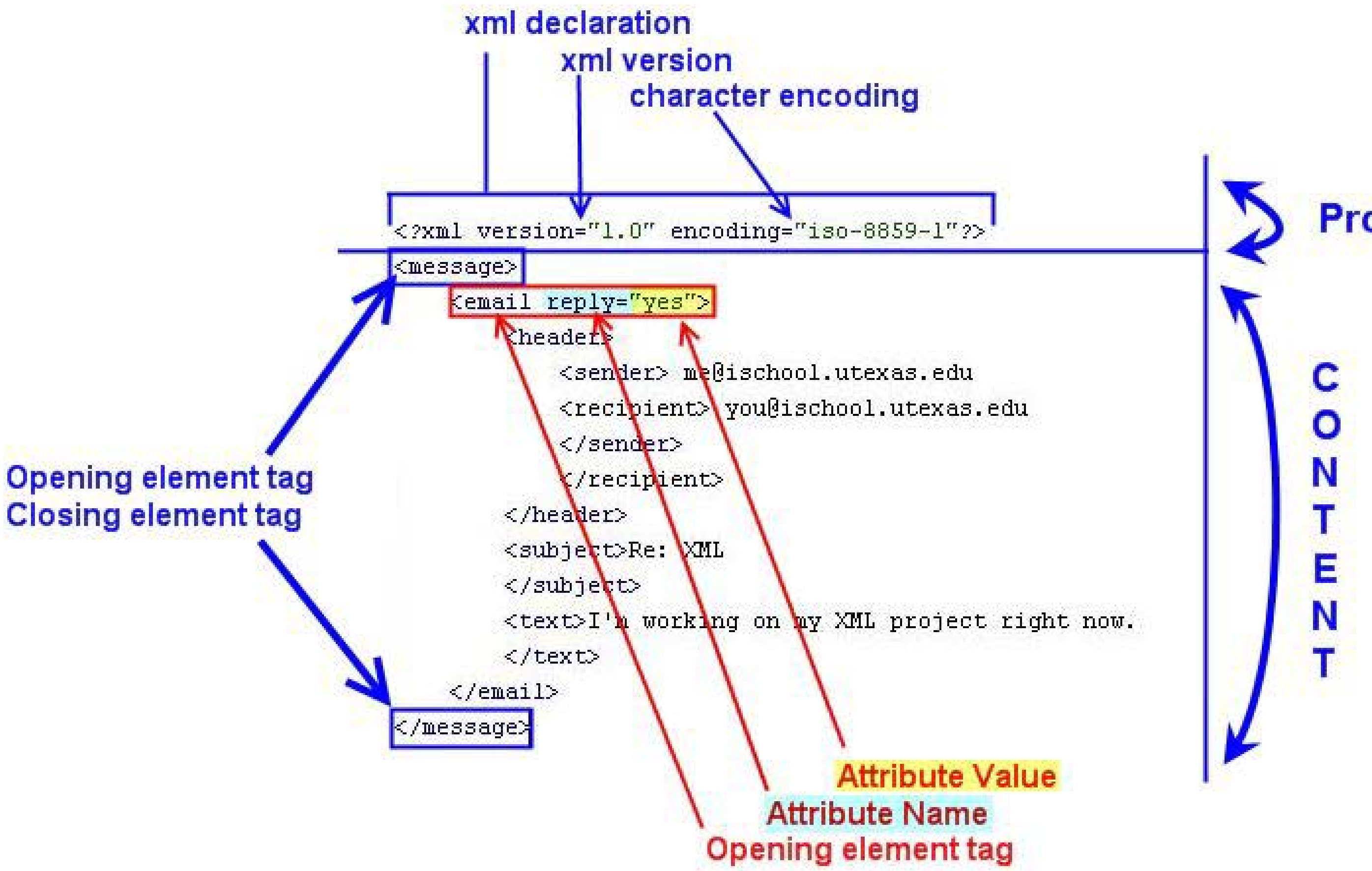
While attributes can be added to elements in XML, there are a couple of reasons to use attributes sparingly:

- XML parsers have a harder time checking attributes against DTDs.
- If the information in the attribute is valuable, why not contain that information in an element?

- Since some attributes can only have predefined categories, you can't go back and easily add new categories.

We recommend using attributes for information that isn't absolutely necessary for interpreting the document or that has a predefined number of options that will not change in the future.

When using attributes in XML, the value of the attributes must always be contained in quotes. The quotes can be either single or double quotes. For example, the attribute version=”1.0” in the opening XML declaration could be written version=’1.0’ and would be interpreted the same way by the XML parser. However, if the attribute value contains quotes, it is necessary to use the other style of quotation marks to indicate the value. For example, if there was an attribute name with a value of John “Q.” Public then it would need to be marked up in XML as name=‘John “Q” Public’, using the symbols for quotes to enclose the attribute value that is not being used in the value
itself.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<message>
    <email reply="yes">
        <header>
            <sender> me@ischool.utexas.edu
            <recipient> you@ischool.utexas.edu
            </sender>
            </recipient>
        </header>
        <subject>Re: XML
        </subject>
        <text>I'm working on my XML project right now.
        </text>
    </email>
</message>
```

There are some rules regarding the order of opening and closing elements, but that will be covered later in the tutorial. For now, let's try creating a simple XML document.

Creating a Simple XML Document

Now that you know the basic rules for creating an XML document, let's try them out.

Like most, if not all, standards developed by the W3Group, you can create XML documents using a plain text editor like Notepad (PC), TextEdit (Mac), or pico (Unix). You can also use programs like Dreamweaver and Cooktop, but all that is necessary to create the document is a text editor.

Let's say we have two types of documents we would like to wrap in XML: emails and letters. We want to encode the emails and letters because we are creating an online repository of archival messages within an organization or by an individual. By encoding them in XML, we hope to encode their content once and be able to translate it to a variety of outputs, like HTML, PDFs, or types not yet created.

To begin, we need to declare an XML version:

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

Now, after declaring the XML version, we need to determine the root element for the documents. Let's use message as the root element, since both email and letters can be classified as messages.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<message>
</message>
```

Note: You might have noticed that I created both the opening and closing tags for the message element. When creating XML documents, it is useful to create both the opening and closing elements at the same time. After creating the tags, you would then fill in the content. Since one of the fatal errors for XML is forgetting to close an element, if you make the opening and closing tags each time you create an element, you won't accidentally forget to do so.

Parent and child relationships

A way of describing relationships in XML is the terminology of parent and child. In our examples, the parent or "root" element is <message>, which then has two child elements, <email>, and <letter>.

An easy way of showing how elements are related in XML is to indent the code to show that an element is a child of another. For example,

```
<?xml version="1.0" encoding="iso-8859-1"?>
<message>

<email>
</email>
</message>
```

Now that we have the XML declaration, the root element, and the child element (email), let's determine the information we want to break out in an email. Say we want to keep information about the sender, recipients, subject, and the body of the text. Since the information about the sender and recipients are generally in the head of the document, let's consider them children elements of a parent element that we will call <header>. In addition to <header>, the other child elements of <email> will be <subject> and <text>. So our XML will look something like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<message>

<email>
<header>
<sender>me@ischool.utexas.edu</sender>
<recipient>you@ischool.utexas.edu</recipient>

</header>
<subject>Re: XML
</subject>
<text>I'm working on my XML project right now.
</text>

</email>

</message>
```

Now, let's create an XML document for a letter. Some of the information in a letter we want to know include the sender, the recipient, and the text of the letter. Additionally, we want to know the date that it was sent and what salutation was used to start off the message. Let's see what this would look like in XML:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<message>

<letter>
<letterhead>
<sender>Margaret</sender>
<recipient>God</recipient>
<date>1970</date>
```

```
</letterhead>
<text>
<salutation>Are you there God?</salutation>
It's me Margaret...
</text>
</letter>

</message>
```

Now say we wanted to keep track of whether or not these messages were replies or not. Instead of creating an additional element called <reply>, let's assign an attribute to the elements <email> and <letter> indicating whether that document was a reply to a previous message.

In XML, it would look something like this:

```
<email reply="yes">
```

or

```
<letter reply="no">
```

When creating XML documents, it's always useful to spend a little time thinking about what information you want to store, as well as what relationships the elements will have. Now that we've made some XML documents, let's talk about "well formed" XML and valid XML.

"Well Formed" vs. Valid

When talking about XML documents, two commonly-used terms are "well formed" and "valid." As in "Is your document marked up in valid and well formed XML?"

Well formed in relation to XML means that it has no syntax, spelling, punctuation, grammar errors, etc. in its markup. These kinds of errors can cause your XML document to not parse.

Note: An XML Parser is software that reads XML documents and interprets or "parses" the code according to the XML standard. A parser is needed to perform actions on XML. For example, a parser would be needed to compare an XML document to a DTD.

In the next section, we will talk about some common errors that prevent an XML document from being well formed.

When you say an XML document is valid, you're saying that the element structure and markup of the XML document matches a defined standard of relationships, in addition to having well formed markup. In other words, is this XML document a quality document?

One standard used to validate XML is a DTD, or **D**ocument **T**ype **D**eclaration, although XML Schemas are also used.

These standards are useful when dealing with the creation of a number of XML documents for they provide a quality control measure to ensure that all the documents meet a minimum standard. Another benefit is that it allows for errors to be detected in the process of creating the XML document, rather than at the end. Later, we'll create a sample DTD for our email and letter XML documents.

Note: An important thing to remember is that when a document is valid it is also "well formed," but a "well formed" document is not necessarily valid. Additionally, you can create XML documents without a DTD, but the XML document can't be considered valid without a document type.

Is Your Markup Well-Formed?

So now that we've created some XML documents, we want to make sure they are well formed documents. To determine whether or not XML documents are well formed, we need to use an XML parser.

Programs such as Dreamweaver or Cooktop have XML parsers built into the software application, but you can also check for well formed XML with most Internet browsers. The most recent versions of Internet Explorer, Netscape, Mozilla or Firefox have at least some simple XML parsing functionality built in and can check XML documents for well formed markup.

Each parser can have different error messages for the same mistake, but the most common errors are not having closing tags, element names not matching up, not closing quotation marks for attributes, and incorrect order.

Since XML requires elements to have opening and closing tags, missing a closing tag will cause what XML calls a "fatal" error - the parser will shut down and give an error message.

To see what message you might get with different parsers for this error, use this modified email XML document. You will see the error message that your browser generates. To see the code, go ahead and view the source (from your browser menu, choose View > Source or Page Source, depending on the browser). Notice that the <sender> element is missing a closing tag.

When creating elements, you can use a mixture of characters, both upper and lower case, as well as symbols and numbers. However, XML is less forgiving then HTML; in XML, case matters. For example, <sender> and <Sender> are two separate types of elements. HTML would read them as the same element. So another common error when coding in XML is accidentally mixing cases between opening and closing tags. Take a look at this example of mixed capitalization to see what error messages your parser might give for that mistake. To see the code that generated this error message, view the source.

Another common error is forgetting to close your quotations around the attributes value. Take a look at this example of forgetting to close quotations to see what error messages the parser you're using might give. To see the code that generated this error message, view the source.

Finally, XML parsers expect to see tags opened and closed in a certain order. XML requires the most recently opened XML element to be closed. One way to remember this is the ABBA rule. No, we're not talking about the "Dancing Queen"; rather, if element A is opened first, and element B is the child of element A, you must close element B before closing A. Here's what it looks like:

```
<A>

        <B>
        </B>
</A>
```

For an example of the error messages this mistake might generate, look at this example of order error in XML. To see the code that generated this error message, view the source.

These are some of the common errors that are seen when making sure that your XML is "well formed." Now let's talk about DTDs and validating your XML documents.

Creating a DTD

Why would you want to create a DTD?

The benefits of DTDs are that it allows you to create numerous documents and make sure that the information contained in them will be comparable. For example, all the information about dates are in tags called <date> rather than <time>, <dates>, <Date> or <DATE>. By creating XML documents that meet a DTD's requirements, you can also share information between institutions.

Here's a "real life" example of using DTDs. The Society of American Archivists and the Library of Congress created **E**ncoded **A**rchival **D**escription (EAD) for the purpose of encoding finding aids. The first version of EAD adhered to SMGL standards; however, with the popularity of XML, newer versions are XML-compliant.

Archival institutions like the Center for American History or Nettie Lee Benson Latin American Collection have created their finding aids using an EAD DTD. Automatic harvesters can use these finding aids to generate an online catalog of what is available at these archives. For an example of this, see the Texas Archival Resources Online or

TARO. TARO is a collection of all the finding aids of archival repositories in Texas that are encoded in EAD.

Note: This section shows you how to create an external DTD file. However, DTDs can also be placed internally in an XML document.

Rules for Creating DTDs

When creating a DTD, you need to define all the elements and attributes you'll have in the XML documents. So let's create a DTD for our message XML documents.

Some syntax to remember when creating DTDs are the following:

Symbol	Meaning	Example
,	AND	header (sender, recipient*, date)
|	OR	message (email | letter)
()	Occurs only Once	(email | letter)
+	must occur at least once	(header, subject?, text+)
?	occurs either once or not at all	(header, recipient* , date?)
*	can occur zero or more times	(sender, recipient*, date)

Elements are declared in the following manner:

```
<!ELEMENT elementName ( elementParts ) >
```

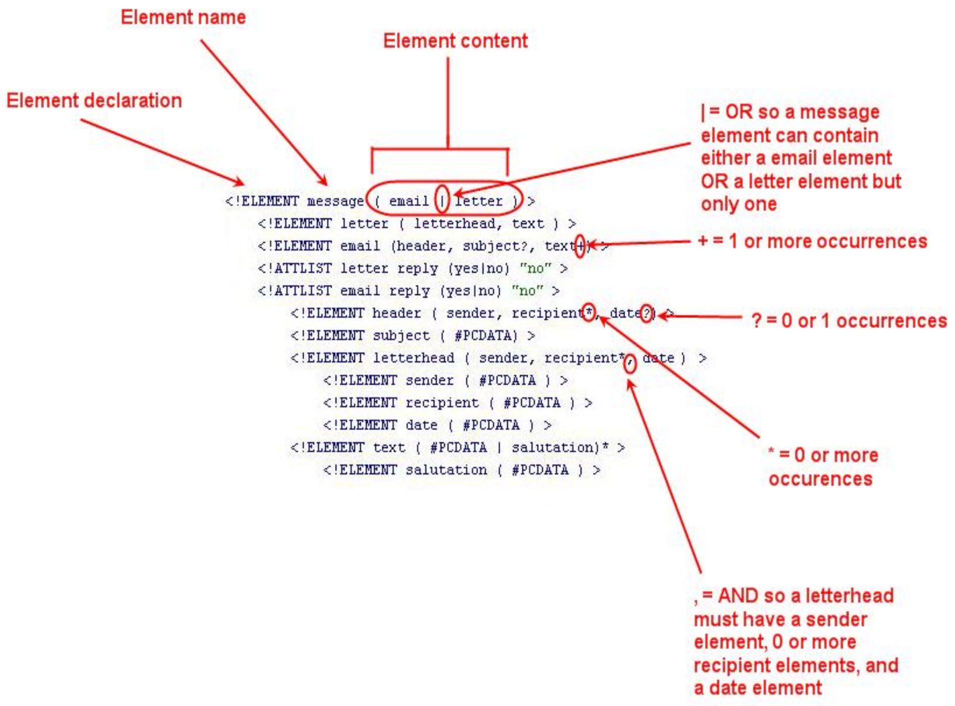
Attributes are declared like this:

```
<!ATTLIST elementName attributeName attributeType attributeDefault >
```

So when creating a DTD for our message XML files, we would have something like this:

```
<!ELEMENT message ( email | letter ) >
<!ELEMENT letter ( letterhead, text ) >
<!ELEMENT email (header, subject?, text+) >
<!ATTLIST letter reply ( yes | no ) "no" >
<!ATTLIST email reply ( yes | no ) "no" >
<!ELEMENT header ( sender, recipient*, date?) >
<!ELEMENT subject ( #PCDATA) >
<!ELEMENT letterhead ( sender, recipient*, date ) >
<!ELEMENT sender ( #PCDATA ) >
<!ELEMENT recipient ( #PCDATA ) >
```

<!ELEMENT date (#PCDATA) >
<!ELEMENT text (#PCDATA | salutation)* >
<!ELEMENT salutation (#PCDATA) >


Element name
Element content
Element declaration
| = OR so a message element can contain either a email element OR a letter element but only one
+ = 1 or more occurrences
? = 0 or 1 occurrences
* = 0 or more occurences
, = AND so a letterhead must have a sender element, 0 or more recipient elements, and a date element

```
<!ELEMENT message ( email | letter ) >
    <!ELEMENT letter ( letterhead, text ) >
    <!ELEMENT email (header, subject?, text+) >
    <!ATTLIST letter reply (yes|no) "no" >
    <!ATTLIST email reply (yes|no) "no" >
        <!ELEMENT header ( sender, recipient*, date?) >
        <!ELEMENT subject ( #PCDATA) >
        <!ELEMENT letterhead ( sender, recipient*, date ) >
            <!ELEMENT sender ( #PCDATA ) >
            <!ELEMENT recipient ( #PCDATA ) >
            <!ELEMENT date ( #PCDATA ) >
        <!ELEMENT text ( #PCDATA | salutation)* >
            <!ELEMENT salutation ( #PCDATA ) >
```

An important thing to remember when making DTDs is that unless you use **|** when defining the element parts, the order of the elements you have within that section is **required** in your XML. So in a letter, the element <letterhead> must occur before the element <text >.

There are some tools that will automatically generate DTDs from XML. HitSoftware, a W3Group member, has created this XML to DTD tool. So now that you've created a DTD, how do you validate your XML against it?

Validating with a DTD

Linking your XML document to a DTD

Now that you've created a DTD for an XML document, you need to link the XML to the DTD. This takes place in the prolog of the XML document.

Remember that the prolog starts off with the XML declaration:

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

Immediately following the XML declaration, you would then either link to a DTD or write an internal DTD. While DTDs can be both internal or external, if you are using a DTD for multiple documents, it makes more sense to have the DTD in a separate "external" file. Otherwise, you will have to put the full DTD in the prolog of every XML document, rather than just one line of code.

To link to an external DTD, the declaration goes like this:

```
<!DOCTYPE RootElementName SYSTEM "DTDfileLocation">
```

Note: SYSTEM in the DTD declaration can be replaced by PUBLIC if the DTD is available via the Internet. You would then need to have a public name for the DTD in the file. For example, the W3Group uses DTDs for the various markup languages they recommend. Here is the recommended DTD for strict XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Assuming that the message.dtd file we created was in the same folder as our XML files for our email and letters, we would add the following line to the XML code:

```
<!DOCTYPE message SYSTEM "message.dtd">
```

Compare the files email.xml and email2.xml to see the differences.

Validating with a DTD

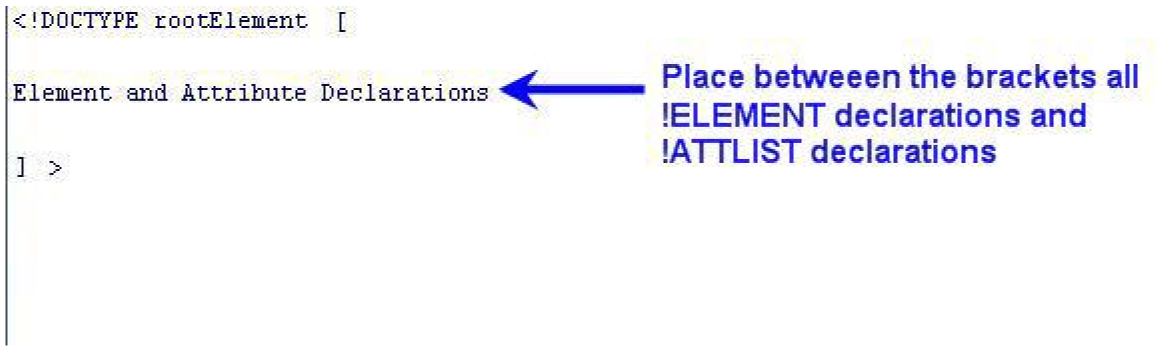
Now that you have the XML linked, you'll need a full parser to validate the XML files. While most browsers can check for well formed XML, only Internet Explorer 5.0 and higher has a complete XML parser built in to the program that checks against DTDs. You can also use programs such as Dreamweaver, Cooktop, and a variety of other XML authoring software.

There are also online resources, such as this online XML Validator, but you need an internal DTD or a DTD available on the Internet in order to compare it against.

An internal DTD is located in the same place as a link to an external DTD but follows the following structure.

```
<!DOCTYPE rootElement [

Element and attribute declarations go here between the brackets ]>
```



```
<!DOCTYPE rootElement  [

Element and Attribute Declarations

] >
```

Place betweeen the brackets all !ELEMENT declarations and !ATTLIST declarations

To link to a DTD on the Internet, you first need to have an account that can serve items to the web, such as an iSchool account. Then you would need to link to the external DTD with the full URL address. So if we were hosting message.dtd in your personal iSchool account, its link would appear as the following: <!DOCTYPE message SYSTEM http://www.ischool.utexas.edu/~youraccountname/message.dtd >.

For more information about using an iSchool account to post items on the Internet, see the How to Use Your School of Information Account tutorial.

Hopefully, you now feel comfortable creating some basic XML and DTD documents. For more resources on XML and DTDs and XML editors, continue on.

XML Resources

Free or Open Source XML Editors

While XML documents can be created by plain text editors, there are also numerous free and open source XML editors available.

One XML editor is Xerlin, which is based on Java and so is platform independent. Xerlin is available from http://www.xerlin.org/. However, to create and expand XML documents, Xerlin requires a DTD but does not allow you to create a DTD. Without a DTD, Xerlin only allows the editing of existing elements and attributes of an XML document, but does not allow you to add additional elements or attributes.

Another XML editor is Cooktop. Cooktop is a freeware Windows-only XML editor available from http://xmlcooktop.com. Cooktop allows for the creation of XML, DTD and XSLT documents, as well as using XPATH. Cooktop differentiates by color the code for XML and DTD documents. In addition, Cooktop provides numerous tools to help in the creation and modification of those documents, such as a DTD extractor to generate a DTD based on a sample XML document, an HTML to XSL converter, and more.

The following are some additional XML resources:

W3Schools, a source of various tutorials on W3Group recommendations:
http://www.w3schools.com/

Webmonkey has an XML tutorial:
http://webmonkey.wired.com/webmonkey/authoring/xml/

XML.com, a website run by O'Reilly Publishers, regarding XML:
http://www.xml.com/

The W3Group's XML page:
http://www.w3.org/XML/

HitSoftware's Automatic DTD Generator:
http://www.hitsw.com/xml_utilites/

Brown University- Scholarly Technology Group's Online XML Validator:
http://www.stg.brown.edu/service/xmlvalid/