

# Your First XML Document

---

This chapter teaches you how to create simple XML documents with tags you define that make sense for your document. You'll learn how to write a style sheet for the document that describes how the content of those tags should be displayed. Finally, you'll learn how to load the documents into a Web browser so that they can be viewed.

Since this chapter will teach you by example, and not from first principals, it will not cross all the t's and dot all the i's. Experienced readers may notice a few exceptions and special cases that aren't discussed here. Don't worry about these; you'll get to them over the course of the next several chapters. For the most part, you don't need to worry about the technical rules right up front. As with HTML, you can learn and do a lot by copying simple examples that others have prepared and modifying them to fit your needs.

Toward that end I encourage you to follow along by typing in the examples I give in this chapter and loading them into the different programs discussed. This will give you a basic feel for XML that will make the technical details in future chapters easier to grasp in the context of these specific examples.

## Hello XML

This section follows an old programmer's tradition of introducing a new language with a program that prints "Hello World" on the console. XML is a markup language, not a programming language; but the basic principle still applies. It's easiest to get started if you begin with a complete, working example you can expand on rather than trying to start with more fundamental pieces that by themselves don't do anything. And if you do encounter problems with the basic tools, those problems are



### In This Chapter

Creating a simple XML document

Exploring the Simple XML Document

Assigning meaning to XML tags

Writing style sheets for XML documents

Attaching style sheets to XML documents



a lot easier to debug and fix in the context of the short, simple documents used here rather than in the context of the more complex documents developed in the rest of the book.

In this section, you'll learn how to create a simple XML document and save it in a file. We'll then take a closer look at the code and what it means.

## Creating a Simple XML Document

In this section, you will learn how to type an actual XML document. Let's start with about the simplest XML document I can imagine. Here it is in Listing 3-1:

### Listing 3-1: Hello XML

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

That's not very complicated, but it is a good XML document. To be more precise, it's a *well-formed* XML document. (XML has special terms for documents that it considers "good" depending on exactly which set of rules they satisfy. "Well-formed" is one of those terms, but we'll get to that later in the book.) This document can be typed in any convenient text editor like Notepad, BBEdit, or emacs.



Cross-Reference

Well-formedness is covered in Chapter 6, *Well-Formed XML Documents*.

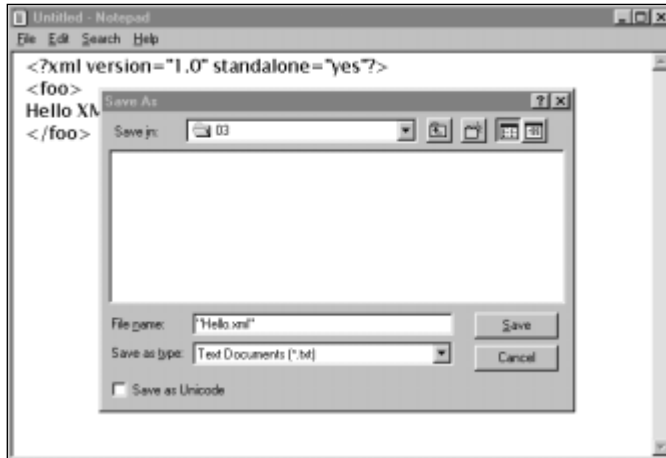
## Saving the XML File

Once you've typed the preceding code, save the document in a file called `hello.xml`, `HelloWorld.xml`, `MyFirstDocument.xml`, or some other name. The three-letter extension `.xml` is fairly standard. However, do make sure that you save it in plain text format, and not in the native format of some word processor like WordPerfect or Microsoft Word.



Note

If you're using Notepad on Windows 95/98 to edit your files, when saving the document be sure to enclose the file name in double quotes, e.g. "Hello.xml", not merely `Hello.xml`, as shown in Figure 3-1. Without the quotes, Notepad will append the `.txt` extension to your file name, naming it `Hello.xml.txt`, which is not what you want at all.



**Figure 3-1:** A saved XML document in Notepad with the file name in quotes

The Windows NT version of Notepad gives you the option to save the file in Unicode. Surprisingly this will work too, though for now you should stick to basic ASCII. XML files may be either Unicode or a compressed version of Unicode called UTF-8, which is a strict superset of ASCII, so pure ASCII files are also valid XML files.

Cross-  
Reference

UTF-8 and ASCII are discussed in more detail in Chapter 7, *Foreign Languages and non-Roman Text*.

## Loading the XML File into a Web Browser

Now that you've created your first XML document, you're going to want to look at it. The file can be opened directly in a browser that supports XML such as Internet Explorer 5.0. Figure 3-2 shows the result.

What you see will vary from browser to browser. In this case it's a nicely formatted and syntax colored view of the document's source code. However, whatever it is, it's likely not to be particularly attractive. The problem is that the browser doesn't really know what to do with the `FOO` element. You have to tell the browser what it's expected to do with each element by using a style sheet. We'll cover that shortly, but first let's look a little more closely at your first XML document.

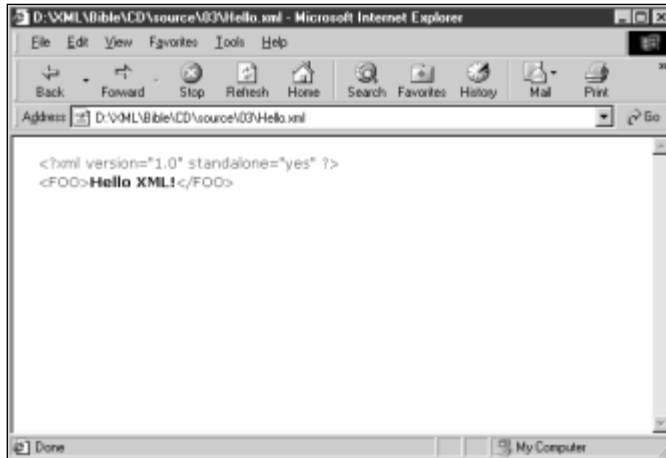


Figure 3-2: hello.xml in Internet Explorer 5.0

## Exploring the Simple XML Document

Let's examine the simple XML document in Listing 3-1 to better understand what each line of code means. The first line is the *XML declaration*:

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

This is an example of an *XML processing instruction*. Processing instructions begin with `<?` and end with `?>`. The first word after the `<?` is the name of the processing instruction, which is `xml` in this example.

The XML declaration has `version` and `standalone` *attributes*. An attribute is a name-value pair separated by an equals sign. The name is on the left-hand side of the equals sign and the value is on the right-hand side with its value given between double quote marks.

Every XML document begins with an XML declaration that specifies the version of XML in use. In the above example, the `version` attribute says this document conforms to XML 1.0. The XML declaration may also have a `standalone` attribute that tells you whether or not the document is complete in this one file or whether it needs to import other files. In this example, and for the next several chapters, all documents will be complete unto themselves so the `standalone` attribute is set to `yes`.

Now let's take a look at the next three lines of Listing 3-1:

```
<FOO>  
Hello XML!  
</FOO>
```

Collectively these three lines form a F00 *element*. Separately, <F00> is a *start tag*; </F00> is an *end tag*; and Hello XML! is the *content* of the F00 element.

You may be asking what the <F00> tag means. The short answer is “whatever you want it to.” Rather than relying on a few hundred predefined tags, XML lets you create the tags that you need. The <F00> tag therefore has whatever meaning you assign it. The same XML document could have been written with different tag names, as shown in Listings 3-2, 3-3, and 3-4, below:

#### Listing 3-2: greeting.xml

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

#### Listing 3-3: paragraph.xml

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

#### Listing 3-4: document.xml

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

The four XML documents in Listings 3-1 through 3-4 have tags with different names. However, they are all equivalent, since they have the same structure and content.

## Assigning Meaning to XML Tags

Markup tags can have three kinds of meaning: structure, semantics, and style. Structure divides documents into a tree of elements. Semantics relates the individual elements to the real world outside of the document itself. Style specifies how an element is displayed.

Structure merely expresses the form of the document, without regard for differences between individual tags and elements. For instance, the four XML documents shown in Listings 3-1 through 3-4 are structurally the same. They all specify documents with a single non-empty, root element. The different names of the tags have no structural significance.

Semantic meaning exists outside the document, in the mind of the author or reader or in some computer program that generates or reads these files. For instance, a Web browser that understands HTML, but not XML, would assign the meaning “paragraph” to the tags `<P>` and `</P>` but not to the tags `<GREETING>` and `</GREETING>`, `<FOO>` and `</FOO>`, or `<DOCUMENT>` and `</DOCUMENT>`. An English-speaking human would be more likely to understand `<GREETING>` and `</GREETING>` or `<DOCUMENT>` and `</DOCUMENT>` than `<FOO>` and `</FOO>` or `<P>` and `</P>`. Meaning, like beauty, is in the mind of the beholder.

Computers, being relatively dumb machines, can’t really be said to understand the meaning of anything. They simply process bits and bytes according to predetermined formula (albeit very quickly). A computer is just as happy to use `<FOO>` or `<P>` as it is to use the more meaningful `<GREETING>` or `<DOCUMENT>` tags. Even a Web browser can’t be said to really understand that what a paragraph is. All the browser knows is that when a paragraph is encountered a blank line should be placed before the next element.

Naturally, it’s better to pick tags that more closely reflect the meaning of the information they contain. Many disciplines like math and chemistry are working on creating industry standard tag sets. These should be used when appropriate. However, most tags are made up as you need them.

Here are some other possible tags:

<code>&lt;MOLECULE&gt;</code>	<code>&lt;INTEGRAL&gt;</code>
<code>&lt;PERSON&gt;</code>	<code>&lt;SALARY&gt;</code>
<code>&lt;author&gt;</code>	<code>&lt;email&gt;</code>
<code>&lt;planet&gt;</code>	<code>&lt;sign&gt;</code>
<code>&lt;Bill&gt;</code>	<code>&lt;plus/&gt;</code>
<code>&lt;Hillary&gt;</code>	<code>&lt;plus/&gt;</code>

```
<Gennifer>    <plus/>
<Paula>      <plus/>
<Monica>     <equals/>
<divorce>
```

The third kind of meaning that can be associated with a tag is style meaning. Style meaning specifies how the content of a tag is to be presented on a computer screen or other output device. Style meaning says whether a particular element is bold, italic, green, 24 points, or what have you. Computers are better at understanding style than semantic meaning. In XML, style meaning is applied through style sheets.

## Writing a Style Sheet for an XML Document

XML allows you to create any tags you need. Of course, since you have almost complete freedom in creating tags, there's no way for a generic browser to anticipate your tags and provide rules for displaying them. Therefore, you also need to write a style sheet for your XML document that tells browsers how to display particular tags. Like tag sets, style sheets can be shared between different documents and different people, and the style sheets you create can be integrated with style sheets others have written.

As discussed in Chapter 1, there is more than one style-sheet language available. The one used here is called Cascading Style Sheets (CSS). CSS has the advantage of being an established W3C standard, being familiar to many people from HTML, and being supported in the first wave of XML-enabled Web browsers.



### Note

As noted in Chapter 1, another possibility is the Extensible Style Language. XSL is currently the most powerful and flexible style-sheet language, and the only one designed specifically for use with XML. However, XSL is more complicated than CSS, not yet as well supported, and not finished either.



### Cross-Reference

XSL will be discussed in Chapters 5, 14, and 15.

The `greeting.xml` example shown in Listing 3-2 only contains one tag, `<GREETING>`, so all you need to do is define the style for the `GREETING` element. Listing 3-5 is a very simple style sheet that specifies that the contents of the `GREETING` element should be rendered as a block-level element in 24-point bold type.

**Listing 3-5: greeting.xsl**

```
GREETING {display: block; font-size: 24pt; font-weight: bold;}
```

---

Listing 3-5 should be typed in a text editor and saved in a new file called `greeting.css` in the same directory as Listing 3-2. The `.css` extension stands for Cascading Style Sheet. Once again the extension, `.css`, is important, although the exact file name is not. However if a style sheet is to be applied only to a single XML document it's often convenient to give it the same name as that document with the extension `.css` instead of `.xml`.

## Attaching a Style Sheet to an XML Document

After you've written an XML document and a CSS style sheet for that document, you need to tell the browser to apply the style sheet to the document. In the long term there are likely to be a number of different ways to do this, including browser-server negotiation via HTTP headers, naming conventions, and browser-side defaults. However, right now the only way that works is to include another processing instruction in the XML document to specify the style sheet to be used.

The processing instruction is `<?xml-stylesheet?>` and it has two attributes, `type` and `href`. The `type` attribute specifies the style-sheet language used, and the `href` attribute specifies a URL, possibly relative, where the style sheet can be found. In Listing 3-6, the `xml-stylesheet` processing instruction specifies that the style sheet named `greeting.css` written in the CSS style-sheet language is to be applied to this document.

**Listing 3-6: styledgreeting.xml with an xml-stylesheet processing instruction**

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css2" href="greeting.css"?>
<GREETING>
Hello XML!
</GREETING>
```

---

Now that you've created your first XML document and style sheet, you're going to want to look at it. All you have to do is load Listing 3-6 into Mozilla or Internet Explorer 5.0. Figure 3-3 shows styledgreeting in Internet Explorer 5.0. Figure 3-4 shows styledgreeting.xml in an early developer build of Mozilla.



Figure 3-3: styledgreeting.xml in Internet Explorer 5.0

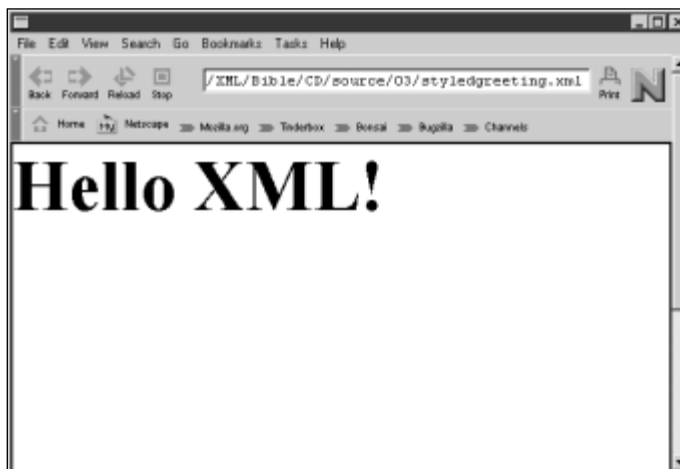


Figure 3-4: styledgreeting.xml in an early developer build of Mozilla

## Summary

In this chapter you learned how to create a simple XML document. In particular you learned:

- ♦ How to write and save simple XML documents.
- ♦ How to assign to XML tags the three kinds of meaning: structure, semantics, and style.
- ♦ How to write a CSS style sheet for an XML document that tells browsers how to display particular tags.
- ♦ How to attach a CSS style sheet to an XML document with an `xml-stylesheet` processing instruction.
- ♦ How to load XML documents into a Web browser.

In the next chapter, we'll develop a much larger example of an XML document that demonstrates more of the practical considerations involved in choosing XML tags.

