

# XML basics for new users

## An introduction to proper markup

Skill Level: Introductory

Kay Whatley ([kaywhatley@aboveandbeyondlearning.com](mailto:kaywhatley@aboveandbeyondlearning.com))

Technology Author  
Freelance

24 Feb 2009

If you're new to XML, this article introduces the basic construction of XML documents as well as the rules that you must follow to create well-formed XML, including naming conventions, proper tag nesting, attribute guidelines, declarations, and entities. You'll also gain an understanding of validation in terms of both DTD and schema usage.

XML stands for *Extensible Markup Language*, with the *markup* bit being the key. You can create content and mark it up with delimiting tags, making each word, phrase, or chunk into identifiable, sortable information. The files, or *document instances*, you create consist of elements (tags) and content, and the elements help the documents to be understood fairly well when read from printouts or even processed electronically. The more descriptive the elements, the more a document's parts can be identified. From the early days of markup to today, one advantage of tagging content is that if a computer system is lost, the *data* in print can still be understood from its tags.

Markup languages evolved from early, private company and government forms into Standard Generalized Markup Language (SGML), Hypertext Markup Language (HTML), and eventually into XML. SGML can seem complex, and HTML (which was really just an element set) was just not powerful enough to identify information. XML is designed as an easy-to-use and easy-to-extend markup language.

With XML, you can create your own elements, giving you the freedom to precisely represent your pieces of information. Rather than treating your documents as headings and paragraphs, you can identify each part within the document. For efficiency, you'll want to define a finite list of your elements and stick to them. (You

can define your elements in a Document Type Definition (DTD) or in a schema, which I will discuss briefly later.) As you start out and get used to XML, feel free to experiment with element names as you build practice files.

## Building XML

As I mentioned, your XML files will consist of content plus markup. You place much of your content in elements by surrounding your content with tags. For example, suppose you need to create an XML cookbook. You have a recipe named *Ice Cream Sundae* to prepare in XML. To mark up the recipe name, you enclose that text in your element by placing the beginning tag before your text and the ending tag after your text. You might call the element `recipeName`. To mark the beginning tag of the element, place the element's name inside angle brackets (`<>`) like this: `<recipeName>`. Then, type your text *Ice Cream Sundae*. After the text, enter the element's ending tag, which is the element's name inside angle brackets plus an ending forward slash (`/`) before the element's name, like this: `</recipeName>`. These tags form an *element*, into which you can enter content or even other elements.

You can create element names for individual documents or for document sets. You can craft the rules for how the elements fit together based on your specific needs. You can be very specific or keep element names more generic. You can create rules for what each element is allowed to contain and make these rules strict, lax, or something in between. Just be sure to create elements that identify the parts of your documents that you feel are important.

### Start your XML file

The first line of your XML document might be an XML declaration. This optional part of the file identifies it as an XML file, which can help tools and humans identify the file as XML rather than SGML or some other markup. The declaration can be written simply as `<?xml?>` or include the XML version (`<?xml version="1.0"?>`) or even the character encoding, such as `<?xml version="1.0" encoding="utf-8"?>` for Unicode. Because this declaration must be first in the file, if you plan to combine smaller XML files into a larger file, you might want to omit this optional information.

### Create your root element

The root element's beginning and end tags surround your XML document's content. Only one root element is in the file, and you need this "wrapper" to contain it all. [Listing 1](#) shows a truncated portion of the example I use here with a root element named `<recipe>`. (See [Download](#) for the full XML file.)

### Listing 1. The root element

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe>
</recipe>
```

As you build your document, your content and additional tags will go between `<recipe>` and `</recipe>`.

## Name your elements

### Matching case in tags

When you create your XML, be sure that your beginning and end tags match in case. If the case doesn't match, you might get an error when you use or view the XML. Internet Explorer, for example, will not display the file content if the case is mismatched. Instead, Internet Explorer displays messages about the beginning and end tags not matching.

So far, you have `<recipe>` as your root element. With XML, you choose the names for your elements, then define the corresponding DTD or schema based on those names. The names you create can contain alphabetic characters, numbers, and special characters such as underscores (`_`). Here are a few things to note about your naming:

- Spaces are not allowed in the element names.
- Names must begin with an alphabetic character, not a number or symbol. (After this first character, you can use any combination of letters, numbers and the allowed symbols.)
- Case does not matter, but be consistent to avoid confusion.

Building on the prior example, if you add an element named `<recipename>`, it will have a beginning tag `<recipename>` and a corresponding end tag `</recipename>`.

## Listing 2. More elements

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe>
<recipename>Ice Cream Sundae</recipename>
<preptime>5 minutes</preptime>
</recipe>
```

An XML document can have some empty tags that do not have anything inside and can be expressed as a single tag instead of as a set of beginning and end tags. To use an HTML-like example, you might have `` as a stand-alone element. It doesn't contain any child elements or text, so it is an empty

element and you can express it as `` (finished off with a space and the familiar ending slash).

## Nest the elements

*Nesting* is the placement of elements inside other elements. These new elements are called *child* elements, and the elements that enclose them are their *parent* elements. Several elements are nested inside the `<recipe>` root element, as in [Listing 3](#). These nested child items include `<recipeName>`, `<ingredlist>`, and `<preptime>`. Inside the `<ingredlist>` element are multiple occurrences of its own child element, `<listitem>`. Nesting can be many levels deep in an XML document.

A common syntax error is improper nesting of parent and child elements. Any child element must be completely enclosed between the starting and end tags of its parent element. Sibling elements must each end before the next sibling begins.

The code in [Listing 3](#) shows proper nesting. The tags begin and end without *intermingling* with other tags.

### Listing 3. Properly nested XML elements

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe>
  <recipeName>Ice Cream Sundae</recipeName>
  <ingredlist>
    <listitem>
      <quantity>3</quantity>
      <itemdescription>chocolate syrup or chocolate fudge</itemdescription>
    </listitem>
    <listitem>
      <quantity>1</quantity>
      <itemdescription>nuts</itemdescription>
    </listitem>
    <listitem>
      <quantity>1</quantity>
      <itemdescription>cherry</itemdescription>
    </listitem>
  </ingredlist>
  <preptime>5 minutes</preptime>
</recipe>
```

## Add attributes

*Attributes* are sometimes added to elements. Attributes consist of a name-value pair, with the value in double quotation marks ("), thus: `type="dessert"`. Attributes provide a way to store additional information each time you use an element, varying the attribute value as needed from one instance of an element to another within the same document.

You type the attribute—or even multiple attributes—within the starting tag of an element: `<recipe type="dessert">`. If you add multiple attributes, separate

them with spaces: `<recipename cuisine="american" servings="1">`.  
[Listing 4](#) shows the XML file as it currently stands.

#### Listing 4. The current XML file with elements and attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe type="dessert">
<recipename cuisine="american" servings="1">Ice Cream Sundae</recipename>
<preptime>5 minutes</preptime>
</recipe>
```

You can use as few or as many attributes as you feel you need. Consider the details you might add to your documents. Attributes are especially helpful if documents will be sorted—for example, by `type` of recipe. Attribute names can include the same characters as element names, with similar rules for omitting spaces and starting names with alphabetic characters.

## Well-formed versus valid XML

If you follow the rules outlined in your structure, you can easily produce well-formed XML. *Well-formed XML* is XML that follows all the rules of XML: proper element naming, nesting, attribute naming, and so on.

Depending on what you do with your XML, you might work with well-formed XML. But consider the aforementioned example of sorting by recipe type. You need to ensure that every `<recipe>` element contains the `type` attribute in order to sort recipes. Being able to properly validate and ensure that this attribute's value is always present can be invaluable (no pun intended).

*Validation* is checking your document's structure against rules for your elements and how you defined child elements for each parent element. You define these rules in a *Document Type Definition* (DTD) or in a schema. This validation requires you to create your DTD or schema, and then reference the DTD or schema file within your XML files.

To enable validation, you include the document type (`DOCTYPE`) in your XML documents near the top. This line refers to the DTD or schema (your list of elements and rules) to be used to validate that document. For example, your `DOCTYPE` might read something like [Listing 5](#).

#### Listing 5. DOCTYPE

```
<!DOCTYPE MyDocs SYSTEM "filename.dtd">
```

This example assumes that your element list file is named *filename.dtd* and resides

on your computer (SYSTEM versus PUBLIC if pointing to a public file location).

## Using entities

*Entities* can be phrases of text or special characters. They can point internally or externally. Entities must be declared and expressed properly to avoid errors and to ensure proper display.

You cannot typed special characters directly into your content. To use a symbol in your text, you must set it up as an entity using its character code. You can set up phrases such as a company name as an entity, then type the entity throughout your content. To set up an entity, create a name for it, and type it within your content, starting with an ampersand (&) and ending with a semicolon (;)—for example, &coname; (or whatever you name it). You then enter code within your DOCTYPE inside square brackets ([ ]), as in [Listing 6](#). This code identifies the text that stands in for the entity.

### Listing 6. ENTITY

```
<!DOCTYPE MyDocs SYSTEM "filename.dtd" [ <!ENTITY coname "Rabid Turtle Industries" ]>
```

Using entities might help you avoid typing the same phrase or information repeatedly. It can also make it easier to adjust the text—perhaps if the company name changes—in many places with a simple adjustment in the entity definition.

## Avoiding errors

As you learn to create your XML files, open them in an XML editor to check for well-formedness and confirm that you're following the rules of XML. If, for example, you have Windows® Internet Explorer®, you can open your XML file in the browser. If it displays your elements, attributes, and content, then the XML is well formed. If instead errors are displayed, you likely have a syntax error and need to review your document carefully for typos or missing tags and punctuation.

As mentioned in [Nest the elements](#), an element that contains another element is the *parent* of that contained element. In the example below, <recipe> is the root element and contains the full content of the file. This parent element, <recipe>, contains child elements <recipename>, <ingredlist>, <directions>, and several others. This structure makes <recipename>, <ingredlist>, and <directions> siblings. Remember to nest your *sibling* elements properly, as well. [Listing 7](#) shows well-formed and properly nested XML.

## Listing 7. Well-formed XML

```
<?xml version="1.0" encoding="UTF-8"?>
<recipe type="dessert">
  <recipename cuisine="american" servings="1">Ice Cream Sundae</recipename>
  <ingredlist>
    <listitem><quantity units="cups">0.5</quantity>
    <itemdescription>vanilla ice cream</itemdescription></listitem>
    <listitem><quantity units="tablespoons">3</quantity>
    <itemdescription>chocolate syrup or chocolate fudge</itemdescription></listitem>
    <listitem><quantity units="tablespoons">1</quantity>
    <itemdescription>nuts</itemdescription></listitem>
    <listitem><quantity units="each">1</quantity>
    <itemdescription>cherry</itemdescription></listitem>
  </ingredlist>
  <utensils>
    <listitem><quantity units="each">1</quantity>
    <utensilname>bowl</utensilname></listitem>
    <listitem><quantity units="each">1</quantity>
    <utensilname>spoons</utensilname></listitem>
    <listitem><quantity units="each">1</quantity>
    <utensilname>ice cream scoop</utensilname></listitem>
  </utensils>
  <directions>
    <step>Using ice cream scoop, place vanilla ice cream into bowl.</step>
    <step>Drizzle chocolate syrup or chocolate fudge over the ice cream.</step>
    <step>Sprinkle nuts over the mound of chocolate and ice cream.</step>
    <step>Place cherry on top of mound with stem pointing upward.</step>
    <step>Serve.</step>
  </directions>
  <variations>
    <option>Replace nuts with raisins.</option>
    <option>Use chocolate ice cream instead of vanilla ice cream.</option>
  </variations>
  <preptime>5 minutes</preptime>
</recipe>
```

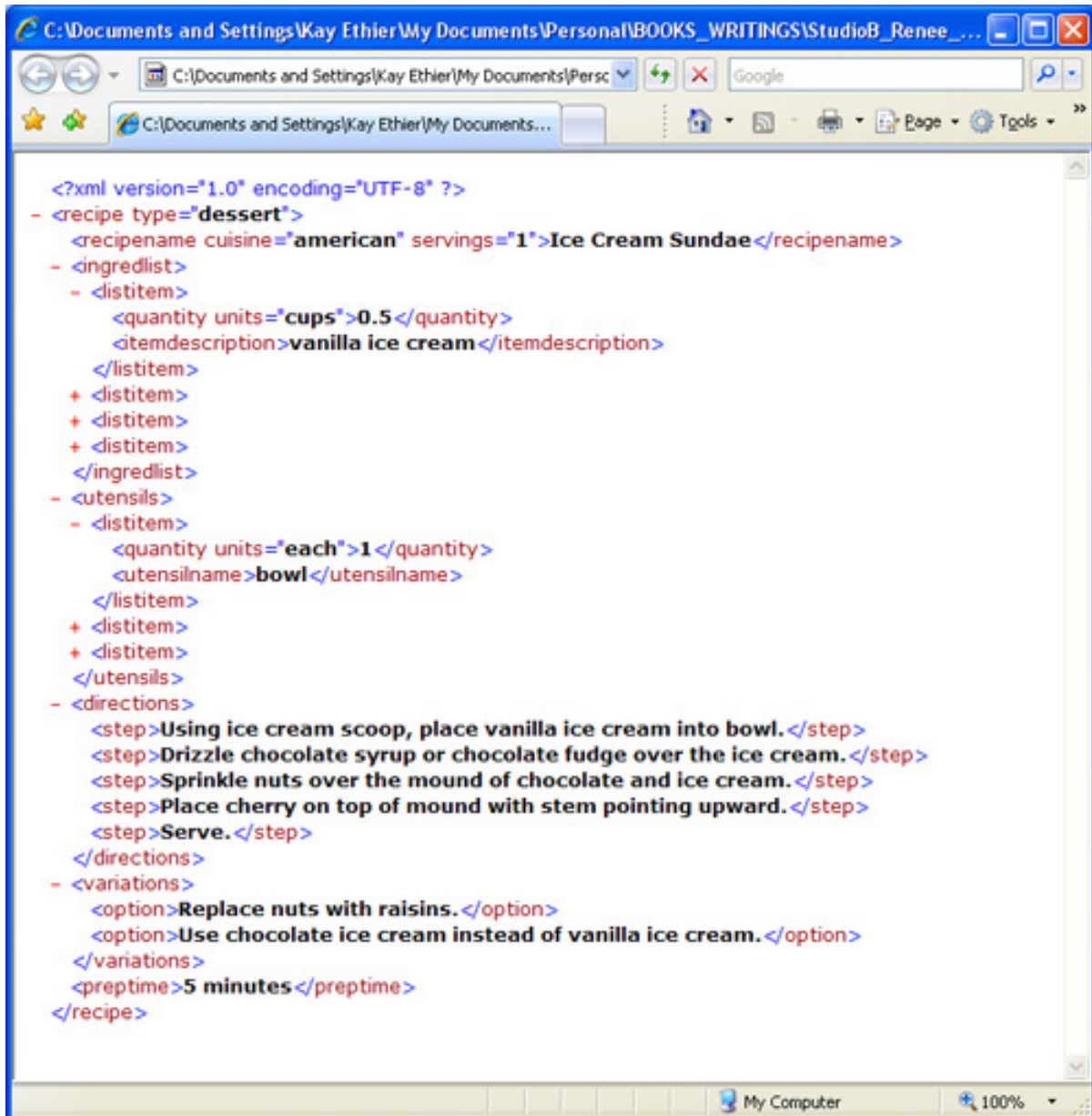
**Note:** The line breaks make it easier for you to read your code and do not affect the XML.

You might wish to experiment with your test files, and move the end tags and beginning tags, to become familiar with the resulting error messages.

## Reviewing XML

In [Figure 1](#), your elements show up clearly when viewed within Internet Explorer. Beginning and end tags surround your content. Small plus (+) and minus (-) symbols are available next to parent elements so you can collapse all elements nested inside them (their *descendants*).

**Figure 1. A sample XML instance (file) with some siblings collapsed**



```

<?xml version="1.0" encoding="UTF-8" ?>
- <recipe type="dessert">
  <recipename cuisine="american" servings="1">Ice Cream Sundae</recipename>
  - <ingredlist>
    - <listitem>
      <quantity units="cups">0.5</quantity>
      <itemdescription>vanilla ice cream</itemdescription>
    </listitem>
    + <listitem>
    + <listitem>
    + <listitem>
  </ingredlist>
  - <utensils>
    - <listitem>
      <quantity units="each">1</quantity>
      <utensilname>bowl</utensilname>
    </listitem>
    + <listitem>
    + <listitem>
  </utensils>
  - <directions>
    <step>Using ice cream scoop, place vanilla ice cream into bowl.</step>
    <step>Drizzle chocolate syrup or chocolate fudge over the ice cream.</step>
    <step>Sprinkle nuts over the mound of chocolate and ice cream.</step>
    <step>Place cherry on top of mound with stem pointing upward.</step>
    <step>Serve.</step>
  </directions>
  - <variations>
    <option>Replace nuts with raisins.</option>
    <option>Use chocolate ice cream instead of vanilla ice cream.</option>
  </variations>
  <preptime>5 minutes</preptime>
</recipe>

```

## Wrapping up

Beyond a few simple rules, you have flexibility in designing your XML elements and attributes. XML's rules are not difficult. Typing an XML document is also not difficult. What *is* difficult is figuring out what you need from your documents in terms of sortability or searchability, then designing elements and attributes to meet your needs.

When you have a good idea of your goals and how to mark up your content, you can build efficient elements and attributes. From that point, careful tagging is all you

need to create well-formed and valid XML.

## Downloads

Description	Name	Size	Download method
Example source code	example.zip	2KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, [tutorials](#), [standards](#), and IBM Redbooks.
- [XML topic on Wikipedia](#): Learn more about XML.
- [XML Tutorials at W3 Schools](#): Expand and test your skills from basic XML through JavaScript and other advanced topics.
- [XML specification](#) from the World Wide Web Consortium: Read more about this flexible text format that works for large-scale electronic publishing as well as the exchange of a wide variety of data on the Web and elsewhere.
- [Introduction to XML](#) (Doug Tidwell, developerWorks, August 2002): Further explore what XML is, why it was developed, and how it shapes electronic commerce in this tutorial. Also, cover various important XML programming interfaces and standards, plus two case studies that show how companies solve business problems with XML.
- [Validating XML](#) (Nicholas Chase, developerWorks, August 2003): In this tutorial, learn what validation is and how to check a document against a Document Type Definition (DTD) or XML Schema document so your data fits model and integrity constraints.
- [New to XML](#) page: For other introductory XML articles and tutorials, check out the XML zone's resource central for XML.
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.

## Get products and technologies

- [IBM trial software for product evaluation](#): Build your next project with trial software available for download directly from developerWorks, including application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

## Discuss

- [Yahoo! Groups](#): Join discussions related to XML.
- [XML zone discussion forums](#): Participate in any of several XML-related

discussions.

- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

## About the author

Kay Whatley

Kay Whatley is a writer, editor, and published author. She is the coauthor of several books, including *XML Weekend Crash Course for Hungry Minds* (Wiley, 2000), lead author of *Advanced FrameMaker* (TIPS, 2004), and author of *XML and FrameMaker* (Apress, 2004). Her latest technology book is *XML: Problem-Design-Solution* (Wiley, 2006). In addition to books, Kay frequently writes articles for industry magazines and Web sites. You can reach her at [kaywhatley@aboveandbeyondlearning.com](mailto:kaywhatley@aboveandbeyondlearning.com).

## Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, and Internet Explorer are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.