1

Writing XML

The XML specification defines how to write a document in XML format. XML is not a language itself. Rather, an XML document is written in a *custom markup language*, according to the XML specification. For example, there could be custom markup languages describing genealogical, chemical, or business data, and you could write XML documents in each one.

Every custom markup language created using the XML specification must adhere to XML's underlying grammar. Therefore, that is where I will start this book. In this chapter, you will learn the rules for writing XML documents, regardless of the specific custom markup language in which you are writing.

Officially, custom markup languages created with XML are called *XML applications*. In other words, these custom markup languages are applications of XML, such as XSLT, RSS, SOAP, etc. But for me, an application is a full-blown software program, like Photoshop. I find the term so imprecise, I usually try to avoid it.

Tools for Writing XML

XML, like HTML, can be written using any text editor or word processor. There are also many XML editors that have been created since the first edition of this book. These editors have various capabilities, such as validating your XML as you type *(see Appendix A)*.

I'll assume you know how to create new documents, open old ones for editing, and save them when you're done. Just be sure to save all your XML documents with the `.xml` extension.

An XML Sample

XML documents, like HTML documents, are comprised of tags and data. One big difference between the two documents, however, is that the tags used by an XML document are created by the author. Another big difference is that an XML document stores and describes that data; it doesn't do anything more with the data, such as display it, like an HTML document does.

XML documents should be rather self-explanatory in that the tags should describe the data they contain **(Figure 1.1)**.

The first line of the XML document **<?xml version="1.0"?>** is the *XML declaration* which notes which version of XML you are using. The next line **<wonder>** begins the data part of the document and is called the *root element*. In an XML document, there can be only one root element.

The next 3 lines are called *child elements*, and they describe the root element in more detail.

```
<name>Colossus of Rhodes</name>
<location>Rhodes, Greece</location>
<height units="feet">107</height>
```

The last child element, height, contains an *attribute* called **units** which is being used to store the specific units of the height measurement. Attributes are used to include additional information to the element, without adding text to the element itself.

Finally, the XML document ends with the closing tag of the root element **</wonder>**.

This is a complete and valid XML document. Nothing more needs to be written, added, annotated, or complicated. Period.

xml

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

<wonder>

  <name>Colossus of Rhodes</name>

  <location>Rhodes, Greece</location>

  <height units="feet">107</height>

</wonder>
```

Figure 1.1 *An XML document describing one of the Seven Wonders of the World: the Colossus of Rhodes. The document contains the name of the wonder, as well as its location and its height in feet.*

xml

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

<ancient_wonders>

 <wonder>

  <name>Colossus of Rhodes</name>

  <location>Rhodes, Greece</location>

  <height units="feet">107</height>

 </wonder>

 <wonder>

  <name>Great Pyramid of Giza</name>

  <location>Giza, Egypt</location>

  <height units="feet">455</height>

 </wonder>

</ancient_wonders>
```

Figure 1.2 *Here I am extending the XML document in Figure 1.1 above to support multiple* <wonder> *elements. This is done by creating a new root element* <ancient_wonders> *which will contain as many* <wonder> *elements as desired. Now, the XML document contains information about the Colossus of Rhodes along with the Great Pyramid of Giza, which is located in Giza, Egypt, and is 455 feet tall.*

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

<wonder>

 <name>Colossus of Rhodes</name>

</wonder>
```

Figure 1.3 *In a well-formed XML document, there must be one element* (wonder) *that contains all other elements. This is called the* root element. *The first line of an XML document is an exception because it's a* processing instruction *and not part of the XML data.*

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

<wonder>

 <name>Colossus of Rhodes</name>

 <main_image file="colossus.jpg"/>

</wonder>
```

Figure 1.4 *Every element must be enclosed by matching tags such as the* name *element. Empty elements like* main_image *can have an all-in-one opening and closing tag with a final slash. Notice that all elements are properly nested; that is, none are overlapping.*

```xml
<name>Colossus of Rhodes</name>
<Name>Colossus of Rhodes</Name>
```

```xml
<name>Colossus of Rhodes</Name>
```

Figure 1.5 *The top example is valid XML, though it may be confusing. The two elements (*name *and* Name*) are actually considered completely different and independent. The bottom example is incorrect since the opening and closing tags do not match.*

```xml
<main_image file="colossus.jpg"/>
```

Figure 1.6 *The quotation marks are required. They can be single or double, as long as they match each other. Note that the value of the* file *attribute doesn't necessarily refer to an image; it could just as easily say* "The picture from last summer's vacation".

Rules for Writing XML

XML has a structure that is extremely regular and predictable. It is defined by a set of rules, the most important of which are described below. If your document satisfies these rules, it is considered *well-formed*. Once a document is considered well-formed, it can be used in many, many ways.

A root element is required

Every XML document must contain one, and only one, root element. This root element contains all the other elements in the document. The only pieces of XML allowed outside (preceding) the root element are comments and processing instructions **(Figure 1.3)**.

Closing tags are required

Every element must have a closing tag. Empty elements *(see page 12)* can use a separate closing tag, or an all-in-one opening and closing tag with a slash before the final > **(Figure 1.4**, and *Nesting Elements*, later in this chapter**)**.

Elements must be properly nested

If you start element A, then start element B, you must first close element B before closing element A (Figure 1.4).

Case matters

XML is case sensitive. Elements named `wonder`, `WONDER`, and `Wonder` are considered entirely separate and unrelated to each other **(Figure 1.5)**.

Values must be enclosed in quotation marks

An attribute's value must always be enclosed in either matching single or double quotation marks **(Figure 1.6)**.

Elements, Attributes, and Values

XML uses the same building blocks as HTML: tags that define elements, values of those elements, and attributes. An XML *element* is the most basic unit of your document. It can contain text, attributes, and other elements. An element has an opening tag with a name written between less than (<) and greater than (>) signs **(Figure 1.7)**. The name, which you invent yourself, should describe the element's purpose and, in particular, its contents. An element is generally concluded with a closing tag, comprised of the same name preceded with a forward slash, enclosed in the familiar less than and greater than signs. The exception to this is called an empty element which may be "self-closing," and is discussed on page 12.

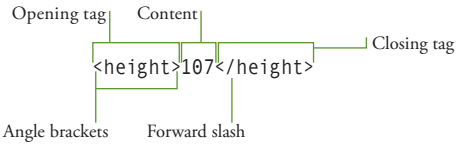

Figure 1.7 *A typical element is comprised of an opening tag, content, and a closing tag. This* height *element contains text.*

Elements may have *attributes*. Attributes, which are contained within an element's opening tag, have quotation-mark delimited *values* that further describe the purpose and content (if any) of the particular element **(Figure 1.8)**. Information contained in an attribute is generally considered metadata; that is, information *about* the data in the element, as opposed to the data itself. An element can have as many attributes as desired, as long as each has a unique name.

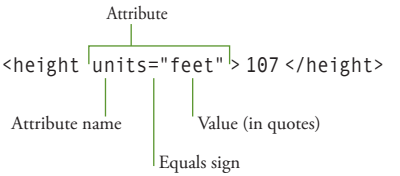

Figure 1.8 *The* height *element now has an attribute called* units *whose value is* feet. *Notice that the word* feet *isn't part of the* height *element's content. This doesn't make the value of* height *equal to* 107 feet. *Rather, the* units *attribute describes the content of the* height *element.*

The rest of this chapter is devoted to writing elements, attributes, and values.

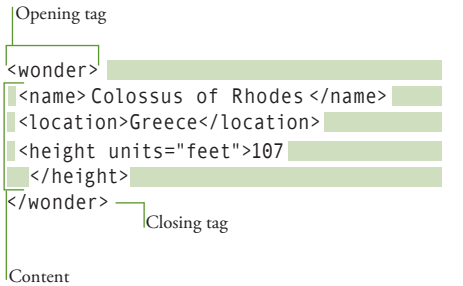

Figure 1.9 *The* wonder *element shown here contains three other elements* (name, location, and height), *but it has no text of its own. The* name, location *and* height *elements contain text, but no other elements. The* height *element is the only element that has an attribute. Notice also that I've added extra white space (green, in this illustration), to make the code easier to read.*

White Space

You can add extra white space, including line breaks, around the elements in your XML code to make it easier to edit and view **(Figure 1.9)**. While extra white space is visible in the file and when passed to other applications, it is ignored by the XML processor, just as it is with HTML in a browser.

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

Figure 1.10 *Because the XML declaration is a processing instruction and not an element, there is no closing tag.*

How To Begin

In general, you should begin each XML document with a declaration that notes what version of XML you're using. This line is called the *XML declaration* **(Figure 1.10)**.

To declare the version of XML that you're using:

1. At the very beginning of your document, before anything else, type **<?xml**.
2. Then, type **version="1.0"**.
3. Finally, type **?>** to complete the declaration.

✔ Tips

- The W3C released a Recommendation for XML Version 1.1 in 2006, but it has few new benefits and little to no support.
- Be sure to enclose the version number in single or double quotation marks. (It doesn't matter which you use, so long as they match.)
- Tags that begin with **<?** and end with **?>** are called *processing instructions*. In addition to declaring the version of XML, processing instructions are also used to specify the style sheet that should be used, among other things. Style sheets are discussed in detail in Part 2, *XSL*.
- This XML processing instruction can also designate the character encoding (UTF-8, ISO-8859-1, etc.), that you're using for the document. Character encodings are discussed in Appendix B.

Creating the Root Element

Every XML document must have one, and only one, element that completely contains all the other elements. This all-encompassing parent element is called the *root element*.

To create the root element:

1. At the beginning of your XML document, type **<root>**, where *root* is the name of the element that will contain the rest of the elements in the document **(Figure 1.11)**.
2. Leave a few empty lines for the rest of your XML document.
3. Finally, type **</root>** exactly matching the name you chose in Step 1.

✔ Tips

- Case matters. <WONDER> is not the same as <Wonder> or <wonder>.
- Element (and attribute) names should be short and descriptive.
- Element and attribute names must begin with a letter, an underscore, or a colon. Names that begin with the letters *xml* (in any combination of upper- and lowercase), are reserved and cannot be used.
- Element and attribute names may contain any number of letters, digits, underscores, and a few other punctuation characters.
- Caveat: Although colons, hyphens, and periods are valid within element and attribute names, I recommend that you avoid including them, as they're often used in specific circumstances (such as for identifying namespaces, subtraction, and object properties, respectively).
- No elements are allowed outside the opening and closing root tags. The only items that are allowed are processing instructions *(see page 7)*.

```
x m l
<?xml version="1.0"?>

<ancient_wonders>




</ancient_wonders>
```

Figure 1.11 *In HTML, the root element is always* <HTML>. *In XML, you can use any valid name for your root element, including* <ancient_wonders>, *as shown here. No content or other elements are allowed before or after the opening and closing root tags, respectively.*

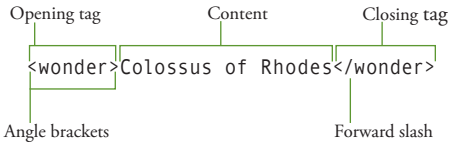

Figure 1.12 *A simple XML element is comprised of an opening tag, content (which might include text, other elements, or be empty), and a closing tag whose only difference with the opening tag is an initial forward slash.*

```
x m l
<?xml version="1.0"?>

<ancient_wonders>

 <wonder>Colossus of Rhodes</wonder>

</ancient_wonders>
```

Figure 1.13 *Every element in your XML document must be contained within the opening and closing tags of the root element.*

Writing Child Elements

Once you have created your root element, you can create any *child element* you like. The idea is that there is a relationship between the root, or parent element, and its child element. When creating child elements, use names that clearly identify the content so that it's easier to process the information at a later date.

To write a child element:

1. Type **<name>**, where *name* identifies the content that is about to appear; the child element's name.
2. Create the content.
3. Finally, type **</name>** matching the word you chose in Step 1 **(Figures 1.12 and 1.13)**.

✔ Tips

- The closing tag is never optional (as it sometimes is in HTML). In XML, elements must always have a closing tag.
- The rules for naming child elements are the same as those for root elements. Case matters. Names must begin with a letter, underscore, or colon, and may contain letters, digits, and underscores. However, although valid, I recommend that you avoid including colons, dashes, and periods within your names. In addition, you may not use names that begin with the letters *xml*, in any combination of upper- and lowercase.
- Names need not be in English or even the Latin alphabet, but if your software doesn't support these characters, they may not display or be processed properly.
- If you use descriptive names for your elements, your XML will be easier to leverage for other uses.

Nesting Elements

Oftentimes when creating your XML document, you'll want to break down your data into smaller pieces. In XML, you can create child elements of child elements of child elements, etc. The ability to nest multiple levels of child elements enables you to identify and work with individual parts of your data and establish a hierarchical relationship between these individual parts.

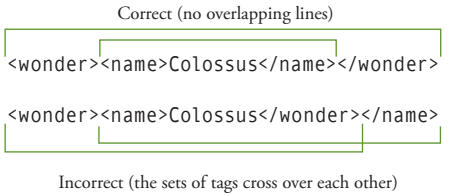

Figure 1.14 *To make sure your tags are correctly nested, connect each set with a line. None of your sets of tags should overlap any other set; each inner set should be completely enclosed within its next outer set.*

To nest elements:

1. Create the opening tag of the outer element as described in Step 1 on page 9.
2. Type **<inner>**, where *inner* is the name of the first individual chunk of data; the first child element.
3. Create the content of the <inner> element, if any.
4. Then, type **</inner>** matching the word chosen in Step 2.
5. Repeat Steps 2–4 as desired.
6. Finally, create the closing tag of the outer element as described in Step 3 on page 9.

✔ Tips

- It is essential that each element be completely enclosed in another. In other words, you may not write the closing tag for the outer element until the inner element is closed. Otherwise, the document will not be considered well-formed, and will generate an error in the XML processor **(Figure 1.14)**.
- You can nest as many levels of elements as you like **(Figure 1.15)**.
- When nesting elements, best practices suggest that you indent the child element. This enables you to easily see parent, child, and sibling relationships. Most XML editors will automatically do this for you.

```
x m l
<?xml version="1.0"?>

<ancient_wonders>

 <wonder>

  <name>Colossus of Rhodes</name>

  <location>Rhodes, Greece</location>

  <height units="feet">107</height>

 </wonder>

</<ancient_wonders>
```

Figure 1.15 *Now the* wonder *element is nested as a child of the* ancient_wonders *element, and* name, location *and* height *are nested as child elements of the* wonder *element.*

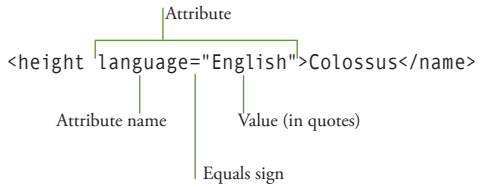

Figure 1.16 *Attributes are* name-value pairs *enclosed within the opening tag of an element. The value must be contained in matched quotation marks (either single or double).*

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

<ancient_wonders>

 <wonder>

  <name language="English">Colossus
    of Rhodes</name>

  <name language="Greek">Κολοσσός της
    Ρόδου</name>

  <location>Rhodes, Greece</location>

  <height units="feet">107</height>

 </wonder>

</ancient_wonders>
```

Figure 1.17 *Attributes let you add information about the contents of an element.*

Adding Attributes

An *attribute* stores additional information about an element, without adding text to the element's content itself. Attributes are known as "name-value pairs," and are contained within the opening tag of an element **(Figure 1.16)**.

To add an attribute:

1. Before the closing > of the opening tag, type **attribute=**, where *attribute* is the word that identifies the additional data.
2. Then, type **"value"**, where *value* is that additional data. The quotes are required.

✔ Tips

- Attribute names must follow the same rules as element names, see the Tips on page 9.
- No two attributes in a given element may have the same name.
- Unlike in HTML, attribute values must, must, **must** be in quotes. You can use either single or double quotes, as long as they match within a single attribute.
- If an attribute's value contains double quotes, use single quotes to contain the value (and vice versa). For example, **comments= 'She said, "The Colossus has fallen!"'**.
- Best practices suggest that attributes should be used as "metadata"; that is, data about data. In other words, attributes should be used to store information about the element's content, and not the content itself **(Figure 1.17)**.
- An additional way to mark and identify distinct information is with nested elements *(see page 10)*.

Using Empty Elements

Empty elements are elements that do not have any content of their own. Instead, they will have attributes to store data about the element. For example, you might have a **main_image** element with an attribute containing the filename of an image, but it has no text content at all.

To write an empty element with a single opening/closing tag:

1. Type **<name**, where *name* identifies the empty element.
2. Create any attributes as necessary, following the instructions on page 11.
3. Finally, type **/>** to complete the element **(Figure 1.18)**.

To write an empty element with separate opening and closing tags:

1. Type **<name**, where *name* identifies the empty element.
2. Create any attributes as necessary, following the instructions on page 11.
3. Finally, type **>** to complete the opening tag.
4. Then, with no spaces, type **</name>** to complete the element, matching the word you chose in Step 1.

✔ Tips

- In XML, both of the above methods are equivalent **(Figure 1.19)**. Which one to use is a stylistic preference; I write elements using a single opening / closing tag.
- In contrast with HTML, you are not allowed to use an opening tag with no corresponding closing tag. A document that contains such a tag is not considered well-formed and will generate an error in the XML processor.

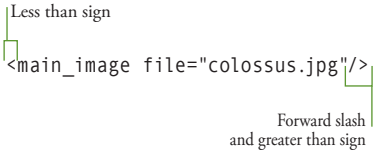

Figure 1.18 *Empty elements can combine the opening and closing tags in one, as shown here, or can consist of an opening tag followed immediately by an independent closing tag as seen in the example below.*

```
x m l
<?xml version="1.0"?>

<wonders_of_the_world>

 <wonder>

  <name language="English">Colossus
    of Rhodes</name>

  <name language="Greek">Κολοσσός της
    Ρόδου</name>

  <location>Rhodes, Greece</location>

  <height units="feet">107</height>

  <main_image file="colossus.jpg"
    w="528" h="349"/>

  <source sectionid="101"
    newspaperid="21"></source>

 </wonder>

</wonders_of_the_world>
```

Figure 1.19 *Typical empty elements are those like* source *and* main_image. *Notice that these elements only contain data in their attributes; the element has no content of its own. I've used both empty element formats in this example: single opening / closing tag and separate opening and closing tags.*

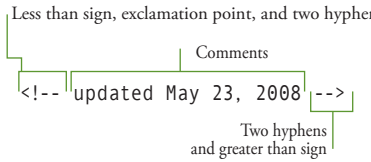

Figure 1.20 *XML comments have the same syntax as HTML comments.*

xml

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

<wonders_of_the_world>

 <wonder>

  <name language="English">Colossus
    of Rhodes</name>

  <name language="Greek">Κολοσσός της
    Ρόδου</name>

  <location>Rhodes, Greece</location>

  <height units="feet">107</height>

  <main_image file="colossus.jpg"
    w="528" h="349"/>

  <!-- the research on this wonder of
    the world came in part from the
    sectionid of the newspaper
    (identified by newspaperid) in
    the source tag below -->

  <source sectionid="101"
    newspaperid="21"/>

 </wonder>

</wonders_of_the_world>
```

Figure 1.21 *Comments let you add information about your code. They can be incredibly useful when you (or someone else) need to go back to a document and understand how it was constructed.*

Writing Comments

It's often useful to annotate your XML documents so that you know why you used a particular element, or what a piece of information specifically means. As with HTML, you can insert comments into your XML documents, and they will not be parsed by the processor **(Figure 1.20)**.

To write comments:

1. Type **<!--**.
2. Write your desired comments.
3. Finally, type **-->** to close the comment.

✔ Tips

- Comments can contain spaces, text, elements, and line breaks, and can therefore span multiple lines of XML.
- No spaces are required between the double hyphens and the content of the comments itself. In other words **<!--this is a comment-->** is perfectly fine.
- You may not use a double hyphen within a comment itself.
- You may not nest comments within other comments.
- You may use comments to hide a piece of your XML code during development or debugging. This is called "commenting out" a section. The elements within a commented out section, along with any errors they may contain, will not be processed by the XML processor.
- Comments are also useful for documenting the structure of an XML document, in order to facilitate changes and updates in the future **(Figure 1.21)**.

Predefined Entities – Five Special Symbols

Entities are a kind of autotext; a way of entering text into an XML document without typing it all out. There are many letters and symbols that can be inserted into HTML documents by using entities. In XML, however, there are only five *predefined entities*.

To write the five predefined entities:

- Type **&** to create an ampersand character (&).
- Type **<** to create a less than sign (<).
- Type **>** to create a greater than sign (>).
- Type **"** to create a double quotation mark (").
- Type **'** to create a single quotation mark or apostrophe (').

✔ Tips

- Predefined entities exist in XML because each of these characters have specific meanings. For example, if you used (<) within the text value of an element or attribute, the XML processor would think you were starting a new element **(Figure 1.22)**.
- You *may not* use (<) or (&) anywhere in your XML document, except to begin a tag or an entity, respectively. If you need to use one of these characters within the text value of an element or attribute, you *must* use one of the predefined entities.
- You may use ("), ('), or (>) within the text value of an element or attribute. However, when using (") or ('), be on the lookout for unintentionally matching existing quotes. Also, I always recommend using the predefined entity for (>) to avoid any possible confusion.
- If you want to create additional entities for your XML documents, you must explicitly declare them *(see Chapter 7)*.

```
x m l
<?xml version="1.0"?>

<wonders_of_the_world>

 <wonder>

  <name language="English">Colossus
    of Rhodes</name>

  <name language="Greek">Κολοσσός της
    Ρόδου</name>

  <location>Rhodes, Greece</location>

  <height units="feet">&lt; 107
    </height>

  <main_image file="colossus.jpg"
    w="528" h="349"/>

  <source sectionid="101"
    newspaperid="21"/>

 </wonder>

</wonders_of_the_world>
```

Figure 1.22 *When this document is parsed, the* > *entity will be displayed as* >. *So when the value of the* height *element is displayed, it will likely read something like* "< 107 ". *How it is displayed will depend on the transformation of the XML, which is discussed in Part 2,* XSL.

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

<xml_book>

 <tags>

  <appearance>

<![CDATA[
<ancient_wonders>
 <wonder>
  <name language="English">
    Colossus of Rhodes</name>
  <name language="Greek">
    Κολοσσός της Ρόδου</name>
  <location>Rhodes, Greece</location>
  <height units="feet">107</height>
  <main_image file="colossus.jpg"
    w="528" h="349"/>
  <source sectionid="101"
    newspaperid="21"/>
 </wonder>
</ancient_wonders>
]]>

  </appearance>

 </tags>

</xml_book>
```

Figure 1.23 *In this example about an example, I use CDATA to display the actual code, without the XML processor parsing it first.*

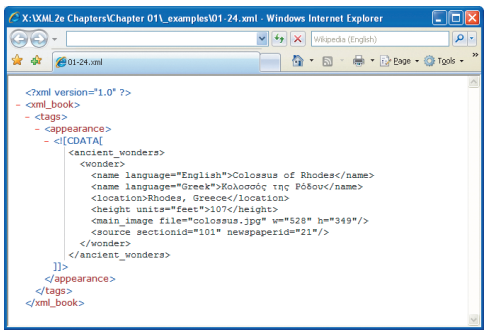

Figure 1.24 *Shown here in Internet Explorer 7 for Windows, you can see how the elements within the CDATA section are treated as text; in contrast with the* xml_book, tags, *and* appearance *elements, which are parsed by the XML processor.*

Displaying Elements as Text

If you want to write about XML elements and attributes in your XML documents, you will want to keep the XML processor from interpreting them, and instead just display them as regular text. To do this, you enclose such information in a CDATA section **(Figure 1.23)**.

To display elements as text:

1. Type **<![CDATA[**.
2. Create the elements, attributes, and content that you would like to display, but not process.
3. Finally, type **]]>** to complete the tag.

✔ Tips

- Two other common uses for the CDATA section are to enclose HTML and JavaScript so that they are not parsed by the XML processor.
- CDATA stands for (unparsed) Character Data, meaning that the CDATA content will not be interpreted by the XML processor. This is opposed to PCDATA, which stands for Parsed Character Data and is discussed in Chapter 6.
- The special meaning that symbols have is ignored in the CDATA section. To display the less than and ampersand symbols, you would write **<** and **&**. If you write **<** and **&**, that's what will display; they will not be replaced with < and &.
- You may not nest CDATA sections.
- CDATA sections can be used anywhere within the root element of an XML document.
- If, for some reason, you want to write **]]>** and you are *not* closing a CDATA section, the > must be written as **>**. See page 14 and Appendix B for more information on writing special symbols.