area. A document is produced as a Working Draft, which is then made available for implementers. These are the people and companies that will actually be building software based on the eventual document. Discussion takes place among members and in forums such as the xml-dev mailing list, and the feedback goes into producing subsequent drafts.

Finally, the group produces a Candidate Recommendation, which is voted on and either accepted or rejected. If rejected, it may move back to Working Draft status. If accepted, it becomes a Proposed Recommendation, and finally a W3C Recommendation.

The Structure of an XML File

The XML 1.0 Recommendation defines a set of conditions that must be followed for XML data to be considered well-formed. Well-formedness is the basis for an XML document; if it's not well-formed, it's not XML.

Well-formedness in XML is based on two types of information contained within a document: character data and markup. In general, the markup is any information that is part of a tag, and the content is any information that isn't.

For example, in the following text, only the word open is actually character data:

```xml
<airlock id="level1">status=open</status></airlock>
```

Everything else on the line is markup that provides information about the content of the document. Parsing is the process of extracting the information represented by the combination of markup and character data.

Tags are not the only type of markup, however. Let's look at the overall structure of the simple file in Listing 1.1 before we move on to discuss the different types of information a document can contain.

**Listing 1.1** A Simple XML Document

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE airlocksysten SYSTEM 'airlocks.dtd'>
<airlocksysten>
  <airlock lockid="A230">
    <size height="520" width="860" />
    <type>Bronson</type>
    <location>Level 2 aft</location>
    <status>open</status>
  </airlock>
  <airlock lockid="0060">
    <size height="200" width="300" />
    <type>Perch</type>
    <location>Level 15 starboard</location>
    <status>closed</status>
  </airlock>
</airlocksysten>
```

The XML Declaration

In most cases, the very first line of an XML file is the XML declaration. The declaration can specify a number of different pieces of information. Take the declaration from our sample file:

```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

The use of the question mark tells any application that this is not simply a tag. This declaration has four separate and distinct pieces of information:

<table>
<thead>
<tr>
<th>Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;?xml</code></td>
<td>The first five characters notify the application that this is, in fact, the XML declaration, and not the start of other content. (This comes in handy when using other encodings, as you'll see shortly.)</td>
</tr>
<tr>
<td><code>version=&quot;1.0&quot;</code></td>
<td>Next comes the version information. This information is included in anticipation of a time when more than one version may exist. Although there are no plans at this time to rework XML and produce a new version, there may be a time when this happens; if it does, including the version information will enable an application to reject a document written in a version it doesn't understand.</td>
</tr>
<tr>
<td><code>encoding=&quot;ISO-8859-1&quot;</code></td>
<td>After the version declaration comes the encoding information. XML has been designed specifically to allow the use of various international character sets. It requires that all applications support the Unicode standard, which provides encoding information for most of the world's major languages. The encoding needs to know how to interpret the characters it sees. In this case, we've specified the Latin-1 character set, ISO 8859-1. Perhaps the most common encoding used with XML files is UTF-8, which includes encodings for most of the world's common alphabets. Of course, this raises the question: How does the application read the encoding information if it doesn't know what encoding to use? The answer lies in the first five characters of the XML declaration. If it doesn't understand the encoding, it goes back to the beginning and tries to determine what encoding would make the first five characters &lt;xml. From there, it can make an educated guess about the encoding.</td>
</tr>
<tr>
<td><code>standalone=&quot;no&quot;</code></td>
<td>Finally, we have the standalone declaration. This declaration determines whether the document contains any external entities. These entities are part of the Document Type Definition (DTD), which we'll discuss briefly later in this chapter, and fully in Chapter 7.</td>
</tr>
</tbody>
</table>
The name of the element is the prominent feature of the markup that defines an element. In this example, the name of the element is type. XML element names have simple rules:

- The name must start with a letter or the underscore character (_).
- The end tag must contain exactly the same name as the start tag. This includes case; XML is case sensitive.
- Names should not contain colons. Officially, they're allowed, but as you'll see when we get to namespaces later in the chapter, the colon has a special meaning and should not be used under other circumstances.
- Names must not contain spaces.

Understandable Names

Although it's not a technical requirement that XML elements have names that fit their functions, part of the advantage of XML is the fact that humans can understand the data. Make sure that unless you're deliberately trying to obscure the data, you name your elements in such a way that an outsider can understand your intent.

Nesting Elements

Of course, a single element would be of limited use. In an XML document, elements are nested, meaning that each element is contained within other elements—except for the root element, of course!

If you're going to nest XML elements, you must do it properly. If the start tag of an element is contained within another element, the end tag must be within that element as well. Let's compare two examples.

Correct:
<sentence>These elements are nested <adverb>properly</adverb></sentence>.

Incorrect:
<sentence>These elements are nested <adverb>improperly</sentence></adverb>.

Elements must be nested properly for the document to be well-formed.

When one element is nested within another, it's said to be the child of that element, and the element that contains it is said to be its parent. In the preceding example, the sentence element is the parent, and the properly nested address element is its child.

Handling Whitespace

XML whitespace is often added to make a document more readable. For example, a document may be formatted as shown in Listing 1.2, or in the more readable format shown in Listing 1.3. This is known as pretty printing.
LISTING 1.2 The XML Data Without Convenient Spacing

```xml
<?xml version="1.0"?>
<airlocksysten<airlock lockid="A23b">
  <location>Level 2 aft</location>
  <status>open</status>
</airlock>
</airlocksysten>
```

LISTING 1.3 The Pretty Printed Document

```xml
<xml version="1.0">
<airlocksysten>
  <airlock lockid="A23b">
    <location>Level 2 aft</location>
    <status>open</status>
  </airlock>
</airlocksysten>
```

Although they contain the same data, Listings 1.2 and 1.3 are not equivalent. The spaces and tabs are considered character data within the elements. In actual processing, however, the spaces and tabs in the `airlocksysten` and `airlock` elements in Listing 1.3 are not really relevant, and although an application is required to pass on these spaces, tabs, and line feeds, they're typically ignored. For example, consider the HTML page in Listing 1.4.

LISTING 1.4 A Sample HTML Page

```html
<html>
  <head>
    <title>Preserving space</title>
  </head>
  <body>
    <p>
      Impressionist paintings are just that:
      <b>Impressions</b>
      of a single moment in time.
    </p>
  </body>
</html>
```

When this page is displayed in a browser, all the whitespace within its elements is collapsed, including the whitespace within the paragraph element `<p>` as shown in Figure 1.1. Collapsing whitespace consists of converting tabs, line feeds, carriage returns, line feed/carriage return combinations, and multiple spaces into a single space.

FIGURE 1.1
Whitespace within elements is collapsed.

It makes sense to collapse the space between elements, but why within them? Well, it turns out that there's really little distinction between within and between. All elements are "within" the root element, so all whitespace is "within" an element.

In the `<p>` element, for example, we have character data, then markup (the `<b>` and `</b>` tags) indicating another element and its character data, then more character data within the `<p>` element. In the `<body>` element, we have character data (the whitespace), then markup (the `<p>` and `</p>` tags) indicating another element and its character data, then more character data (the whitespace).

So what happens if we really want the whitespace? In HTML, we can use a special set of tags, `<pre>` and `</pre>`, to indicate to the browser that the text is preformatted, and the browser should preserve the spaces. For example, if we made the additions shown in bold in Listing 1.5, we would get the result shown in Figure 1.2.

LISTING 1.5 Preserving Whitespace in an HTML Document

```html
<html>
  <head>
    <title>Preserving space</title>
  </head>
  <body>
    <p>
      Impressionist paintings are just that:
      <b>Impressions</b>
      of a single moment in time.
    </p>
  </body>
</html>
```

In XML, we don't have the advantage (or disadvantage) of a single predefined element to inform the application that the whitespace within a particular element is significant. Instead, we can tell the application that the spaces within a particular element are significant by adding an attribute called `xml:space`. We'll deal more with attributes in the next section, but for now you just need to understand that an attribute is a name-value pair added to the start tag of an element.