# Beginning XML with DOM and Ajax

From Novice to Professional

■■■

Sas Jacobs

Apress®

**Beginning XML with DOM and Ajax: From Novice to Professional**

**Copyright © 2006 by Sas Jacobs**

ISBN-13 (pbk): 978-1-59059-676-0

ISBN-10 (pbk): 1-59059-676-5

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Charles Brown, Chris Mills
Technical Reviewer: Allan Kent
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editor: Nicole Abramowitz
Assistant Production Director: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Dina Quan
Proofreader: Dan Shaw
Indexer: Brenda Miller
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

CHAPTER 3

# Web Vocabularies

As XML grows in popularity, the number of XML vocabularies used within various industry and community sectors increases. These groups use XML to store database information, exchange structured information, and even describe concepts.

XML is a mechanism for storing data. When first applied to the web, XML addressed many of the shortcomings associated with HTML. Although you can view any XML document on the web, some vocabularies were created specifically for this medium.

In this chapter, I'll focus on web vocabularies such as

- XHTML
- Mathematical Markup Language (MathML)
- Scalable Vector Graphics (SVG)
- Web services (WSDL and SOAP)

You can use these vocabularies in web browsers and other web-enabled devices.

You can download the files referred to in this chapter from the Source Code area of the Apress web site (`http://www.apress.com`).

Let's start with a closer look at XHTML.

## XHTML

XHTML is probably the most widespread web vocabulary of all; web developers have been using it for several years. XHTML enjoys support in modern web browsers such as Internet Explorer 6 for Windows, Mozilla Firefox 1.x for Windows, and Safari 1.x for Macintosh.

The W3C states that XHTML is HTML *reformulated* in XML. XHTML 1.0 is nothing other than HTML 4.01 in XML syntax. It's an XML-compliant version of HTML. XHTML is a great starting point for a discussion of XML vocabularies.

XHTML provides a number of benefits compared with HTML. First, XHTML separates presentation from content. In XHTML, content is made up of data as well as the structural elements that organize that data. HTML was concerned with both information and its display, whereas its replacement, XHTML, is concerned with both information and the way it's structured. XHTML also uses much stricter construction rules compared with HTML, as XHTML web pages must be well formed. You learned about well-formed documents in Chapter 1.

Because XHTML is based on XML, you can use XML-specific tools and technologies to create modular documents. Throughout the chapter, I'll show you how to merge other vocabularies into XHTML.

Let's begin by looking more closely at the benefits of XHTML.

## Separation of Presentation and Content

The separation of content from presentation is perhaps the single most important concept in web development today. This fundamental principle underpins most modern web specifications.

Content refers to the basic data and structures that make up a document. Within XHTML, this includes elements such as headings, paragraphs, tables, and lists. Presentation determines how these structures appear within the viewing device and might include font faces, colors, borders, and other visual information. Cascading Style Sheets (CSS) control the presentation of a document.

---

■**Note** When working with XML applications, you can separate the content into both data and data structures. In XML applications, an XML document supplies the data, while XSLT stylesheets provide the structure. You still apply styling through CSS stylesheets.

---

It's important to separate content from presentation because it allows you to repackage the content for different audiences. If you want to provide the same information to a web browser, a mobile phone, and a screen reader, the presentation layer must be different for each device. You can achieve this by excluding the presentation of information from web documents.

Separating presentation from content has four major benefits:

- Accessibility
- Targeted presentation using stylesheets
- Streamlined maintenance
- Improved processing

Let's look at each of these benefits in more detail.

### Accessibility

In recent times, the W3C has focused on making XHTML more accessible to people with disabilities. For example, people with visual impairments can use screen readers and voice browsers when working with XHTML documents. Documents that follow the XHTML construction rules often require little or no change, so users can access them with a screen reader.

Many countries have legislation requiring web sites to be accessible to people with disabilities. In the United States, Section 508 of the Rehabilitation Act of 1973 requires people with disabilities to have access to federal agency electronic information. You can find out more about this regulation at `http://www.usability.gov/accessibility/`.

The W3C Web Accessibility Initiative web site (`http://www.w3.org/WAI/`) provides information about how to make web sites accessible. The site includes quick tips for accessibility (`http://www.w3.org/WAI/References/QuickTips/`), as well as a list of tools to help you evaluate whether your site is currently accessible (`http://www.w3.org/WAI/ER/existingtools.html`).

By separating the visual elements from the actual content of your page, you make the content instantly more accessible. Screen readers and other text-based browsers, such as Lynx for Unix and Linux, can interpret the flow of the document easily. Ultimately, users of your site will have a better experience.

### Targeted Presentation

If you separate the presentation layer from your content, you'll be able to target its appearance for specific devices. You can do this by storing all style information within a stylesheet and linking a specific stylesheet for each device that you want to support. Storing the style information in one place makes it easier to reuse stylesheets and maintain a consistent look.

Several types of stylesheets exist, but the most popular are CSS and XSLT. I'll explain these stylesheets in detail in Chapters 5 to 7.

### Streamlined Site Maintenance

Storing the content and structure separately from the presentation layer makes it easier to maintain your web site. Pages no longer contain presentational elements mixed in with the XHTML structures and data. When working through long blocks of code, you only need to concern yourself with the structural elements because the presentation layer exists elsewhere. This streamlines the site maintenance process and speeds up workflow.

### Improved Processing

Accessibility and targeted presentation were important concerns in HTML even before XHTML was introduced. XHTML, however, directly addresses the need for an improved processing model. Because the rules for XML are so strict, processing XHTML documents becomes easier than processing its predecessor HTML. Software programs can perform XML-related tasks, such as designing XSLT stylesheets. See the WYSIWYG XSLT Designer by Stylus Studio (`http://www.stylusstudio.com/xhtml.html`) for one such example.

Because the rules for constructing HTML were less strict than XHTML rules, it was possible for HTML pages to contain mistakes that didn't affect their display. For example, you could leave out a closing `</body>` tag but still be able to view the page within a browser.

In addition, some web browsers rendered elements slightly differently, so browser manufacturers started adding proprietary extensions to their browsers. Ultimately, this led to incompatible browsers and lack of compliance with the HTML specification.

You can instruct more recent browser versions and software tools to discard XHTML documents that aren't authored correctly and don't use valid, well-formed XHTML. Modern browsers feature improved page processing because they don't need to deal with malformed documents.

Cell phones and personal digital assistants (PDAs) are capable of viewing web documents using either Wireless Markup Language (WML) or XHTML Basic. WML is an XML vocabulary for Wireless Application Protocol (WAP)-enabled phones, and XHTML Basic is a cut-down version of XHTML that includes only basic markup and text. XHTML Basic was created using XHTML's modularization framework, which I'll discuss in more detail in the "XHTML Modularization" section.

## XHTML Construction Rules

The rules for constructing XHTML pages are a little different compared with HTML pages. You must follow these rules in XHTML:

- Include a DOCTYPE declaration specifying that the document is an XHTML document.
- Optionally include an XML declaration.
- Write all tags in lowercase.
- Close all elements.
- Enclose all attributes in quotation marks.
- Write attributes in full (i.e., don't minimize attributes).
- Use the `id` attribute instead of `name`.
- Nest all tags correctly.
- Specify character encoding.
- Specify language.

You'll see how these rules are applied in the following section, which covers DOCTYPE declarations. I'll also work through some sample XHTML documents.

### DOCTYPE Declarations

In any web vocabulary, you need to determine which elements and attributes are valid. In Chapter 2, you saw how you can do this using Document Type Definitions (DTD) and XML schemas. XHTML 1.0 allows for three different DOCTYPE declarations that determine which DTD to use. You can write the following XHTML documents:

- Transitional
- Strict
- Frameset

A DOCTYPE declaration tells a validator how to check your web page. It also instructs a web browser to render your page in standards-compliant mode. Using an outdated or incorrect DOCTYPE makes browsers operate in "Quirks" mode, where they assume that you're writing old-style HTML.

**XHTML 2.0**

At the time of writing, the W3C had prepared a working draft of the XHTML 2.0 specification (`http://www.w3.org/TR/xhtml2/`). This vocabulary removes backward compatibility and all presentation elements in favor of stylesheets. It allows for more flexible organization using sections and headers, and it introduces separator and line elements, as well as navigation lists. It introduces links to every element and overhauls tables and forms.

The most recent XHTML specification, XHTML 1.1, became a recommendation in May 2001. It has only one document type to choose from: XHTML 1.1, which is very similar to XHTML 1.0 strict.

Each of these four document types has a slightly different set of allowable elements. Choosing the right type of document should be the first step in building your XHTML page. I'll explain each of these document types in more detail.

The examples in this chapter show you how to create pages for an imaginary web site called "Mars Travel." I'll keep the examples simple so you can focus on the XHTML content.

### Transitional XHTML Documents

You use the transitional document type for web sites that need to work in many different web browsers, because it supports the deprecated elements not allowed in the strict DTD. If you're not ready or able to remove all presentation from your documents, you should use the transitional DTD.

Let's look at an example of some transitional markup:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Mars Travel</title>
  </head>
  <body bgcolor="#FFFFFF">
    <h1 align="center">Mars Travel<br />
      <i>Visits to a faraway place </i>
    </h1>
    <hr width="100%" />
    <h2 align="center">Your spacecraft</h2>
    <p align="center">
      Your spacecraft is the Mars Explorer, which provides the latest in
      passenger luxury and travel speed.
    </p>
    <hr width="100%" />
    <p align="center">XHTML 1.0 Transitional Document</p>
  </body>
</html>
```

You can find this document saved as `marstransitional.htm` with your code download.

The document begins with an XML declaration:

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

XHTML documents don't require an XML declaration, but it's recommended that you include one. If you include the declaration, web browsers can check that the document is well formed.

Immediately following the XML declaration, a DOCTYPE declaration tells the web browser exactly what kind of document you're writing:

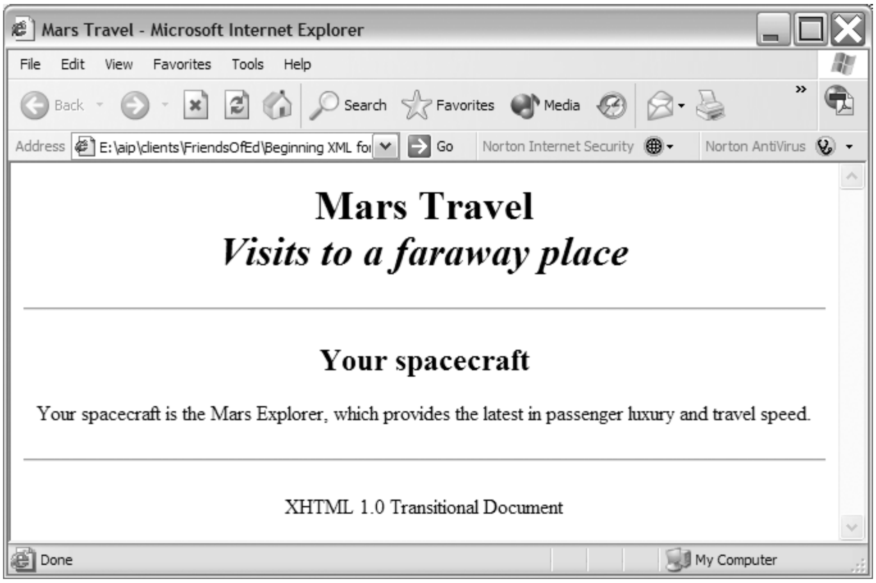
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

The document root `<html>` contains a reference to the XHTML namespace:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

The markup is well-formed XML, but it still contains some presentational information—in particular, `align` and `bgcolor` attributes.

You can download this example, called `marstransitional.htm`, in the Source Code area of the Apress web site (`http://www.apress.com`). If you open the file in a web browser, you should see something like the screen shot shown in Figure 3-1.


Mars Travel - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address E:\aip\clients\FriendsOfEd\Beginning XML fo Go Norton Internet Security Norton AntiVirus
Done My Computer

Mars Travel
*Visits to a faraway place*

Your spacecraft

Your spacecraft is the Mars Explorer, which provides the latest in passenger luxury and travel speed.

XHTML 1.0 Transitional Document

**Figure 3-1.** *The marstransitional.htm page displayed in Internet Explorer*

The XHTML transitional DTD can be useful if you need to support older browsers. Otherwise, you should try to use the strict or XHTML 1.1 document types.

### Strict XHTML Documents

Strict XHTML documents allow you to work with only structural tags, such as headings (`<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`), paragraphs (`<p>`), and lists (`<ul>`, `<ol>`, `<dl>`). All of the presentational elements and attributes, such as `align` and `bgcolor`, are removed. The XHTML 1.1 specification has also completely removed presentational markup. In both strict and XHTML 1.1 document types, you should always use stylesheets to control how your document appears in various browsers.

Let's look at a sample of a strict XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Mars Travel</title>
    <link href="styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1>Mars Travel<br />
      <em>Visits to a faraway place </em>
    </h1>
    <hr />
    <h2>Your spacecraft</h2>
    <p class="centered">
      Your spacecraft is the Mars Explorer, which provides the latest in
      passenger luxury and travel speed.
    </p>
    <hr />
    <p class="footer">XHTML 1.0 Strict Document</p>
  </body>
</html>
```

You can find this file saved as `marsstrict.htm` with your resources.

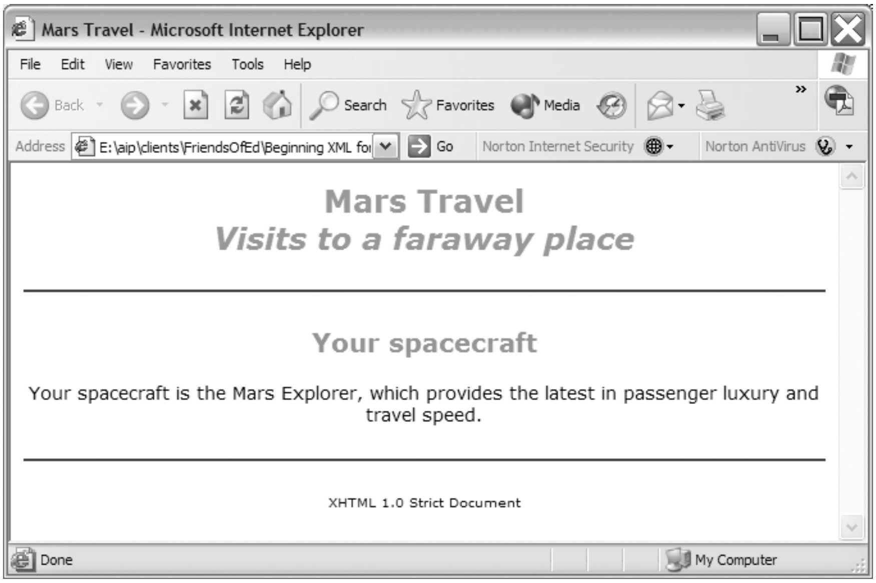
The strict XHTML document is much shorter and doesn't contain any presentational markup. Instead, it contains a link to a stylesheet called `styles.css`, which includes the presentational elements. It also replaces the presentational `<i>` element with the structural `<em>` element. If you view the file in a web browser, it will look much the same as the first XHTML document.

The `styles.css` stylesheet contains the following presentational elements:

```
h1 {
  font-weight: bold;
  font-size: 24px;
  text-align: center;
}
h2 {
  font-weight: bold;
  font-size: 20px;
  text-align: center;
}
hr {
  width: 100%;
}
.centered {
  text-align: center;
}
.footer{
  text-align: center;
}
```

The declarations redefine the `<h1>`, `<h2>`, and `<hr>` elements and create classes called `centered` and `footer`. I'll explain CSS in more detail in Chapter 5.

You can change the look of the web page easily by modifying the CSS. If you apply the same stylesheet to multiple pages, you can update all pages at once by making changes. Figure 3-2 shows the same web page with a modified style sheet.


Mars Travel - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address E:\aip\clients\FriendsOfEd\Beginning XML fo Go
Norton Internet Security Norton AntiVirus
Done
My Computer

Mars Travel
*Visits to a faraway place*

Your spacecraft

Your spacecraft is the Mars Explorer, which provides the latest in passenger luxury and travel speed.

XHTML 1.0 Strict Document

**Figure 3-2.** *A revised presentation of the marsstrict.htm file*

You can find these files saved as `marsstrict2.htm` and `styles2.css`.

The stylesheet tells the browser to set the sizes and colors for the `<h1>` and `<h2>` elements. It also changes the font for the entire page and defines a color for the `<hr>` element. The two classes `centered` and `footer` inherit the default font and center the text. The `footer` class uses a smaller font size.

### Frameset XHTML Documents

XHTML allows you to write a third kind of document called a frameset document. You use frameset documents with web pages that use frames. Frames are no longer recommended for a variety of reasons, so I'll discuss this topic only briefly.

Use the following DOCTYPE declaration to reference a frameset DTD:

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

---

■ **Note** A frameset can include both transitional and strict documents. You can also include one frameset document within another, allowing you to have nested frames.

---

### XHTML 1.1 Documents

XHTML version 1.1 is a modular version of the XHTML 1.0 strict document type. As it's based on the strict document type, you can't include any presentation elements or attributes; you need to declare these in a stylesheet. Frames, which are often presentational, have been moved to a separate "module" that is not enabled by default.

XHTML is modular, which means that parts of the XHTML document have been divided into separate modules that you can add or remove. When I discuss XHTML Modularization later in the chapter, I'll show you how to mix web vocabularies using different XHTML 1.1 modules.

Take a look at this simple XHTML 1.1 document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Mars Travel</title>
    <link href="styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1>Mars Travel<br />
      <em>Visits to a faraway place </em>
    </h1>
    <hr />
    <h2>Your spacecraft</h2>
    <p class="centered">
      Your spacecraft is the Mars Explorer, which provides the latest in
      passenger luxury and travel speed.
    </p>
    <hr />
    <p class="footer">XHTML 1.1 Document</p>
  </body>
</html>
```

This document is saved as `marsxhtm1-1.htm` with your resources.

As you can see, the XHTML 1.0 strict and XHTML 1.1 documents are almost identical. The major difference is the DOCTYPE declaration that specifies which DTD to use. Although most of the internal reorganization is invisible to you, web browsers can understand the modular structure much more easily. Viewing the document gives almost the same results as shown in Figure 3-1.

You could modify the display by changing the stylesheet declarations, exactly as you did with the strict document.

The next requirement for XHTML documents is that tags are written in lowercase.

## Case Sensitivity

Unlike HTML, XHTML is a case-sensitive vocabulary. This means that you must write all elements and attributes in lowercase in order to make them valid. Of course, the text within the element and attribute values is not case-sensitive.

In HTML, you had to write element names in uppercase. However, this wasn't enforced, so any of the following was allowable:

```
<HTML>
<Html>
<html>
```

In XHTML, however, the only allowable element construction is

```
<html>
```

Likewise, you must specify attributes using lowercase names. In HTML, any of the following were allowable:

```
<IMG SRC="images/flower.gif">
<img src="images/flower.gif">
<Img Src="images/flower.gif">
```

In XHTML, all element and attribute names must be lowercase:

```
<img src="images/flower.gif">
```

XHTML is case-sensitive because it's a requirement in XML. Case sensitivity is a major step in internationalization efforts. Although you can easily convert uppercase English characters to lowercase ones, or lowercase characters to uppercase, it's not so easy in other languages. Often there are no equivalent uppercase or lowercase characters, and some case mapping depends on region. Case sensitivity is important in order to allow the specification to use other languages and character sets.

## Closing Elements

In HTML, you didn't need to close some elements, including `<img>`, `<br>`, `<hr>`, and `<input>`. These elements didn't mark up text, so they didn't have a corresponding closing element.

In XML, this type of element, referred to as an empty element, may contain attributes but doesn't mark up text. You must close all elements for an XHTML document to be well formed.

In HTML, empty elements appeared like this:

```
<IMG SRC="flower.gif">
```

In XHTML, empty elements can either appear with an immediate opening and closing tag, such as

```
<img src="flower-.gif"></img>
```

or in the short form, such as

```
<img src="flower.gif"/>
```

In the short form, you add a forward slash (`/`) before the closing angle bracket (`>`). This tells the XML or XHTML parser that the element is empty. Although both forms are legal XHTML, very old browsers have problems reading opening and closing tags for elements that are empty. It's much better to use the short form for empty elements. These browsers also may have difficulty with the forward slash character, so, if you're targeting them, it's also good practice to add a space before the character (`<br />`).

## Attributes

In addition to using the proper case for attribute names, you also need to make sure that you write them correctly. In HTML, you could write attribute values without quotation marks. For example, the following was legal in HTML:

```
<TD colspan=4>
```

HTML also allowed you to minimize attributes:

```
<OPTION selected>An option</OPTION>
```

Neither of these options is acceptable in XHTML. All attributes must have a value, even if it's blank, and you must enclose all values in matching quotation marks:

```
<td colspan="4">
<option selected='selected'>An option</option>
```

In the preceding `<td>` element, you add quotation marks around the attribute value `4`. In the `<option>` element, you remove the minimization of the `selected` attribute and use single quotes around the attribute value. The value for the `selected` attribute is `selected`.

## Names and IDs

In HTML, the `name` attribute identified an element within the document. Later versions also allowed the use of `id` to replace the `name` attribute. In HTML 4.0 and XHTML 1.0, you can use the `name` attribute, the `id` attribute, or both. For example, you can identify the anchor element, `<a>`, with either attribute:

```
<a name="Section1" />
<a id="Section1" />
<a name="Section1" id="Section1" />
```

In XHTML 1.1, however, the W3C permits only the `id` attribute:

```
<a id="Section1" />
```

Again, older browsers expect you to use the `name` attribute. Because of this, some XHTML 1.1 pages don't work in early browser versions.

## Nesting Tags

The HTML language didn't specify how you should nest tags, so writing something like the following didn't cause an error:

```
<H1><EM>A heading</H1></EM>
```

This doesn't work in XHTML; you need to rewrite the code so the tags close in the correct order:

```
<h1><em>A heading</em></h1>
```

## Character Encoding

Specifying the document encoding is very important, and in some cases required, so that the document displays correctly within different web browsers. Document encoding defines a numeric value for each character. Different encoding schemes sometimes use these values in different ways.

Most browsers and computers support ASCII encoding, which assigns values to the 128 most commonly used characters. These characters are compatible across different platforms. If you're using characters with values higher than 128, you must specify the character set so that the browser knows which character to display for a given value.

Within XHTML, you can specify the character set that your document is using in several ways, including

- Using the XML declaration
- Using the `<meta>` element
- Using external means

You can use any of these methods alone or in combination. Using all methods together ensures that the browser understands the document's encoding, even if it doesn't support that encoding. Again, including encoding declarations may confuse some older browsers.

Let's look at each of the methods more closely. Specifying encoding using the XML declaration is very easy, and you've seen it in the examples in Chapter 1:

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

You can specify encoding in a `<meta>` tag by adding the following element to the `<head>` section of your XHTML document:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

## CHOOSING AN ENCODING

UTF-8 is a Unicode character set that supports the first 128 ASCII characters, as well as additional characters. Documents using only simple ASCII characters can use UTF-8 encoding. The basic ASCII character set doesn't include European characters that include accents, and the numeric values for each character may vary depending on the specified encoding.

If you're running an English version of Windows, your default encoding is compatible with ISO-8859-1. This encoding is supported widely, so changing the encoding declaration to ISO-8859-1 allows European characters to display correctly.

Encoding rules are often complex. XML supports UTF-8 and UTF-16 encoding by default. UTF-16 is a large character set that includes many Chinese and Japanese characters, among others. In order to have numeric values for all of the characters, it uses two or more bytes for each character, instead of one byte as in UTF-8 and ASCII. Simple text editors may not support encoding other than UTF-8 or ASCII. For more information about different encoding specifications, visit `http://www.unicode.org/`.

Again, this line tells the browser what type of content the document contains. In the preceding `<meta>` tag, you specify `text/html` as the document type and `ISO-8859-1` as the encoding. If a document contains both the XML declaration and the `<meta>` element, the browser uses the encoding value in the XML declaration. Browsers that don't support the XML declaration use the `<meta>` value.

You can also use the HTTP header `Content-Type` to specify encoding on the web server. This approach provides the most reliable way to specify the encoding in an XHTML document. You can set the header using any server-side technology.

## Specifying Language

HTML 4.0 and XHTML 1.0 allow you to specify the language for a document or element using the `lang` attribute. Web browsers can use this information to display elements in language-specific ways. For example, hyphenation may change depending on the language in use. Additionally, screen readers may read the text using different voices, depending on the language specified. The following `lang` attribute specifies the U.S. version of English as the language for the document:

```
<body lang="en-US">
```

You can find out more about which attribute values to use at `http://www.w3.org/TR/REC-html40/struct/dirlang.html`.

XHTML 1.1 replaces the `lang` attribute with `xml:lang`. In addition to XHTML, many other web vocabularies use this attribute from the `xml` namespace. This makes XHTML much more compatible with other XML applications. If you want a quick refresher on namespaces, see the section, "Understanding the Role of XML Namespaces," in Chapter 2.

## XHTML Tools

You can use three kinds of tools to edit your XHTML documents:

- Simple text editors
- XML editors
- XHTML editors

Each of these tool types offers different benefits. Let's explore these types in more detail.

### Text Editors

Because XHTML is a text-based format, you can create document markup in text editors, including Notepad on Windows, SimpleText on Macintosh, and Vim on Linux. These editors aren't specifically designed to create XHTML or XML documents, so they have very few features that can assist with authoring. They can't provide information about whether a document is well formed or valid, and they don't provide any type of color-coding for the text.

Although they have significant limitations, text editors are often useful because they exist on almost all computers and start up very quickly. The most useful text editors can display line numbers, which are invaluable for tracking down parser errors.

### XML Editors

Many XML editors are designed to work specifically with XML documents. These editors offer many advantages over text editors, not the least of which is automatic color coding for elements within the document.

Although not written specifically for XHTML, XML editors can still provide tag completion so your elements close automatically. In addition, XML editors allow you to check that your document is well formed and valid, based on its DTD or XML schema.

Some popular XML editors include

- Altova's XMLSpy: `http://www.altova.com/products_ide.html`
- Stylus Studio's XML Editor: `http://www.stylusstudio.com/xml/editor/`
- Topologi's Markup Editor: `http://www.topologi.com/products/tme/index.html`
- TIBCO's Turbo XML: `http://www.tibco.com/software/business_integration/turboxml.jsp`
- SyncRO Soft's <oXygen/>: `http://www.oxygenxml.com/index.html/`
- Blast Radius' XMetal: `http://www.xmetal.com/en_us/products/xmetal_author/index.x`
- Wattle Software's XMLwriter: `http://www.xmlwriter.net/`

Most of these products offer a trial version so that you can test whether they'll suit your needs.

### XHTML Editors

Editors written specifically for XHTML documents can provide the most features. These tools often come with XHTML document templates and can warn you about potential display problems. Most importantly, many XHTML editors allow you to design XHTML visually without needing to see the markup. This can be very useful when designing complex layouts.

Some common XHTML editors include

- Adobe's (formerly Macromedia) Dreamweaver: `http://www.macromedia.com/software/dreamweaver/`
- Microsoft's FrontPage: `http://www.microsoft.com/frontpage/`
- W3C's Amaya: `http://www.w3.org/Amaya/`
- Chami.com's HTML-Kit: `http://www.chami.com/html-kit/`
- Adobe's (formerly Macromedia) HomeSite: `http://www.macromedia.com/software/homesite/`
- Belus Technology's XStandard: `http://xstandard.com/?program=google1`
- Bare Bones Software's BBEdit: `http://www.barebones.com/products/bbedit/index.shtml`
- NewsGator Technologies' TopStyle: `http://www.bradsoft.com/topstyle/`

Again, you can often download a trial version so you can test the software against your needs.

## Well-Formed and Valid XHTML Documents

Even if you follow the XHTML construction rules, you need to make sure that the document is both well formed and valid. These concepts are critical regardless of which XML vocabulary you use.

In Chapter 1, you learned that an XML document must be well formed before it can be processed by an XML parser. Well-formed means that

- The document contains one or more elements.
- The document contains a single document element, which may contain other elements.
- Each element closes correctly.
- Elements are case-sensitive.
- Attribute values are enclosed in quotation marks and cannot be empty.

A document is valid if, in addition to being well formed, it uses the correct elements and attributes for the specified vocabulary. In XHTML, the DOCTYPE declaration determines which DTD is used and hence, the validity of elements and attributes.

Validity is an important concept for web developers because creating valid documents guarantees that your web site is interoperable with virtually any XML application. A number of online tools can check XHTML documents for validity.

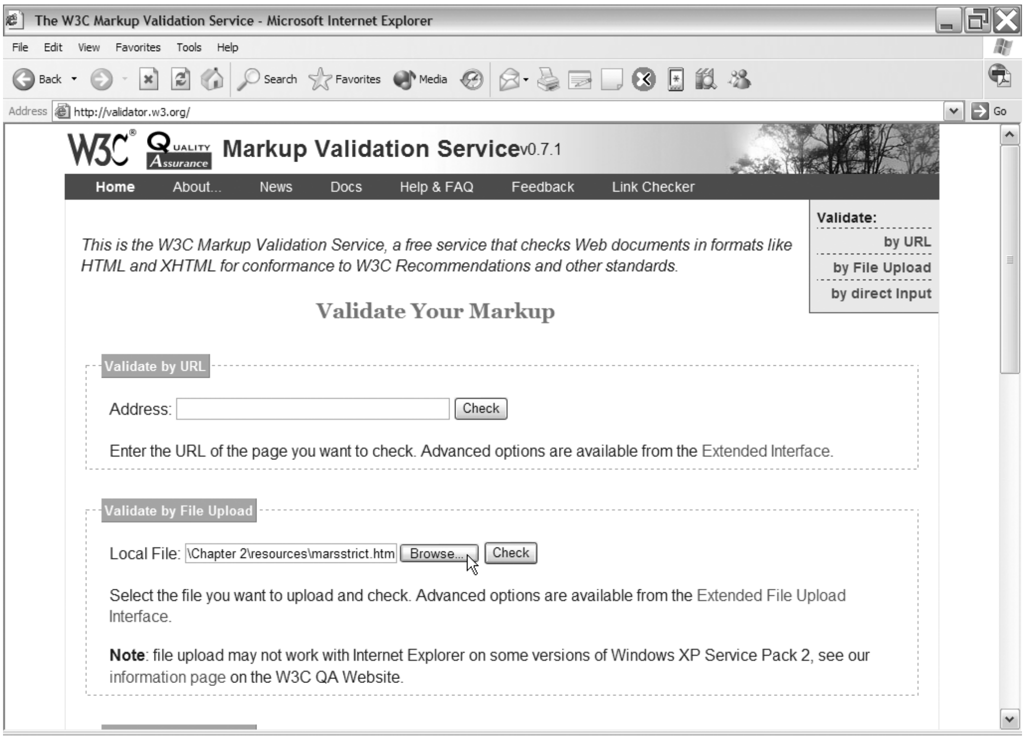
## Online Validators

In addition to the tools I mentioned previously, several web sites offer free online validation services. You can use them to check that your document is valid against specific versions of the XHTML specification. Two popular online validators include

- W3C Markup Validation Service: `http://validator.w3.org/`
- WDG HTML Validator: `http://www.htmlhelp.com/tools/validator/`

I'll validate one of the XHTML documents that you saw previously to show you how the W3C Markup Validation Service works. You need to use the `Validate by File Upload` option to validate an offline file.

Open the web site (`http://validator.w3.org/`) and click the `Browse` button to select your file. In Figure 3-3, I'm validating the file `marsstrict.htm`.


The W3C Markup Validation Service - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://validator.w3.org/ Go
W3C Quality Assurance Markup Validation Service v0.7.1
Home About... News Docs Help & FAQ Feedback Link Checker
Validate: by URL by File Upload by direct Input

*This is the W3C Markup Validation Service, a free service that checks Web documents in formats like HTML and XHTML for conformance to W3C Recommendations and other standards.*

Validate Your Markup

Validate by URL

Address: [ ] Check

Enter the URL of the page you want to check. Advanced options are available from the Extended Interface.

Validate by File Upload

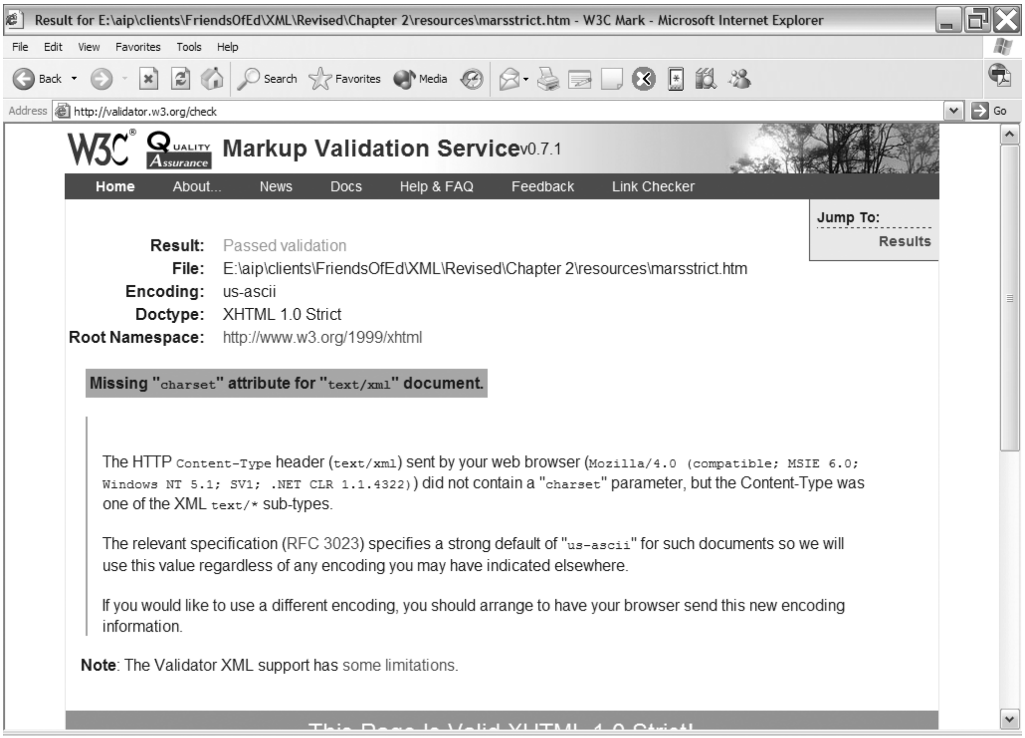
Local File: \Chapter 2\resources\marsstrict.htm Browse... Check

Select the file you want to upload and check. Advanced options are available from the Extended File Upload Interface.

**Note**: file upload may not work with Internet Explorer on some versions of Windows XP Service Pack 2, see our information page on the W3C QA Website.

**Figure 3-3.** *Uploading a file for validation at the W3C Markup Validation Service*

Click the `Check` button to validate the document. After validating, you can see whether the document is valid. You also might see some other messages about the page, as shown in Figure 3-4.


Result for E:\aip\clients\FriendsOfEd\XML\Revised\Chapter 2\resources\marsstrict.htm - W3C Mark - Microsoft Internet Explorer

File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://validator.w3.org/check
Go
W3C Quality Assurance Markup Validation Service v0.7.1
Home About... News Docs Help & FAQ Feedback Link Checker
Jump To: Results

Result: Passed validation
File: E:\aip\clients\FriendsOfEd\XML\Revised\Chapter 2\resources\marsstrict.htm
Encoding: us-ascii
Doctype: XHTML 1.0 Strict
Root Namespace: http://www.w3.org/1999/xhtml

Missing "charset" attribute for "text/xml" document.

The HTTP Content-Type header (text/xml) sent by your web browser (Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)) did not contain a "charset" parameter, but the Content-Type was one of the XML text/* sub-types.

The relevant specification (RFC 3023) specifies a strong default of "us-ascii" for such documents so we will use this value regardless of any encoding you may have indicated elsewhere.

If you would like to use a different encoding, you should arrange to have your browser send this new encoding information.

Note: The Validator XML support has some limitations.

**Figure 3-4.** *The validation results*

In addition to errors, the W3C validator may return warnings. Often, these warnings refer to possible character encoding or DOCTYPE problems. The warnings normally offer suggestions that allow you to address the issues. If you're able to validate your entire site, you can display the W3C XHTML logo on your web page.

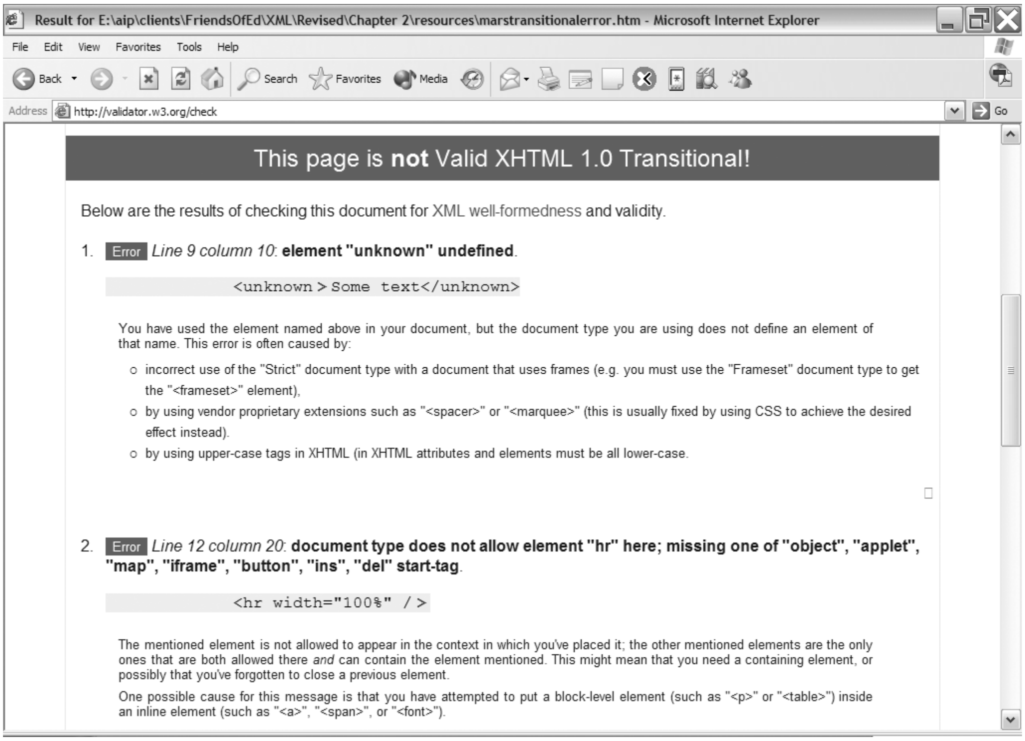
If your validation produces an error message, fix the error and validate the document again. Where you're notified of multiple errors, it's usually easier to revalidate after fixing each error, because a single error can often cause multiple errors later in the document.

I'll deliberately introduce errors into the `marstransitional.htm` page so you can see the effect on validation. I've left out the closing `</h1>` tag and introduced an `<unknown>` element. The document now reads like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Mars Travel</title>
  </head>
  <body bgcolor="#FFFFFF">
    <unknown>Some text</unknown>
    <h1 align="center">Mars Travel<br />
      <i>Visits to a faraway place </i>
    <hr width="100%" />
    <h2 align="center">Your spacecraft</h2>
    <p align="center">
      Your spacecraft is the Mars Explorer, which provides the latest in
      passenger luxury and travel speed.
    </p>
    <hr width="100%" />
    <p align="center">XHTML 1.0 Transitional Document</p>
  </body>
</html>
```

I've saved this document as `marstransitionalerror.htm` if you want to try validating it yourself.

Figure 3-5 shows the effect of validating this document.


Result for E:\aip\clients\FriendsOfEd\XML\Revised\Chapter 2\resources\marstransitionalerror.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://validator.w3.org/check
Go

This page is **not** Valid XHTML 1.0 Transitional!

Below are the results of checking this document for XML well-formedness and validity.

1. Error *Line 9 column 10*: **element "unknown" undefined**.

   `<unknown>Some text</unknown>`

   You have used the element named above in your document, but the document type you are using does not define an element of that name. This error is often caused by:
   - incorrect use of the "Strict" document type with a document that uses frames (e.g. you must use the "Frameset" document type to get the "<frameset>" element),
   - by using vendor proprietary extensions such as "<spacer>" or "<marquee>" (this is usually fixed by using CSS to achieve the desired effect instead).
   - by using upper-case tags in XHTML (in XHTML attributes and elements must be all lower-case.

2. Error *Line 12 column 20*: **document type does not allow element "hr" here; missing one of "object", "applet", "map", "iframe", "button", "ins", "del" start-tag**.

   `<hr width="100%" />`

   The mentioned element is not allowed to appear in the context in which you've placed it; the other mentioned elements are the only ones that are both allowed there *and* can contain the element mentioned. This might mean that you need a containing element, or possibly that you've forgotten to close a previous element.

   One possible cause for this message is that you have attempted to put a block-level element (such as "<p>" or "<table>") inside an inline element (such as "<a>", "<span>", or "<font>").

**Figure 3-5.** *Validation errors*

Validating a web site is an important step. The next section looks at some common practices that can cause validation errors.

## Validation Errors

Unfortunately, the everyday practices of web professionals can cause validation errors. Some common issues involve

- Including JavaScript in your page
- Embedding advertising information
- Including unsupported elements and attributes

In this section, I'll show you some practical tips to address these issues. Many of these tips may be helpful when working with other web vocabularies.

### Including JavaScript in Your Page

For validity, it's best to store your JavaScript in a separate file and refer to it with the `<script>` element:

```
<script type="text/javascript" src="mars.js" />
```

If you can't avoid embedding JavaScript in an XHTML document, place the JavaScript code within a `<![CDATA[...]]>` element so that it is not interpreted as XHTML by the browser. JavaScript can include characters that otherwise cause the document to fail the well-formed test. Instead of using the following code

```
<script type="text/javascript">
<!--
  function maxnumber(a, b) {
    if (a > b) then
      return a;
    if (a < b) then
      return b;
    if (a = b) then
      return a;
  }
-->
</script>
```

rewrite it like this:

```
<script type="text/javascript">
<![CDATA[
  function maxnumber(a, b) {
    if (a > b) then
      return a;
    if (a < b) then
      return b;
    if (a = b) then
      return a;
  }
]]>
</script>
```

### Embedding Advertising Information

Many web sites display advertising information on their pages. If the advertisement isn't valid XHTML, you must make sure that you're using the XHTML 1.0 transitional DTD. You can also add the advertiser information to the page using JavaScript. This ensures that the content displays in the browser, but at the same time, you can ensure that the XHTML page is valid. Make sure that you follow the preceding JavaScript guidelines. I'll cover some advanced JavaScript techniques in Chapter 8.

### Including Unsupported Elements and Attributes

In some cases, you may need to add invalid content to the XHTML page. Using unsupported elements isn't good practice, because it ultimately limits your audience. However, there might be times when you want to add

- Elements or attributes that existed in earlier versions of HTML
- Elements or attributes that are specific to one browser
- New elements or attributes

The first two situations commonly occur when you're trying to build a web site for a specific browser, or when you're trying to convert an older web site to XHTML. You can add this kind of information in several ways. As I discussed in the previous section, you can add the content using JavaScript after the page loads.

Another more complex option is to test for the browser type and version and return appropriate pages to the user. By maintaining templates on the web server, you can quickly transform your web page to support various browsers using XSLT.

## XHTML Modularization

A primary goal of XML is to create a simple markup language that you can extend easily. XHTML 1.1 simplifies the process of extending the XHTML definition. You can add any vocabulary to XHTML through a process called modularization.

Although XHTML modularization is complex, you can still enjoy the benefits. The W3C has released a working draft of a modularization that supports the MathML and SVG vocabularies. These two vocabularies are commonly embedded within XHTML and vice versa. You can find out more at `http://www.w3.org/TR/XHTMLplusMathMLplusSVG/`.

You might need to limit rather than extend the XHTML specification. XHTML Basic provides a subset of the basic modules of XHTML for use on mobile devices; find out more at `http://www.w3.org/TR/xhtml-basic/`.

Using these new vocabularies is very similar to using the other document types you've seen in this chapter. You need to follow the rules of the new document type and declare the appropriate DOCTYPE. The DOCTYPE declaration for XHTML plus MathML plus SVG is

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
"http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
```

The DOCTYPE declaration for XHTML Basic is

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
"http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
```

I've introduced you to the basics of XHTML, examining it as a vocabulary of XML. Now let's move on to examine some of the other popular web vocabularies, starting with MathML and SVG.

## MathML

Mathematical Markup Language (MathML) is a popular XML vocabulary that describes mathematical notation. It was developed to include mathematical expressions on web pages. MathML is an XML vocabulary, so it must be well formed and valid according to the specification. You can find out more about MathML at `http://www.w3.org/Math/`.

While the W3C MathML group was developing the specification, the group realized it actually had two distinct goals. There was a need for a vocabulary that could represent both how mathematic equations were displayed, as well as the meaning of a mathematic equation. The group divided MathML into two types of encoding: presentation and content.

Presentation MathML conveys the notation and structure of mathematical formulas, while Content MathML communicates meaning without being concerned about notation. You can use either or both of these elements, depending on your task, but be aware that each has some web browser limitations.

Firefox supports Presentation MathML, as MathML is part of Mozilla's layout engine. The derived browsers Netscape, Galeon, and Kmeleon also include Presentation MathML, as does the W3C browser Amaya. Internet Explorer 6 supports MathML using plugins such as the free MathPlayer (`http://www.dessci.com/en/products/mathplayer/`) and techexplorer (`http://www.integretechpub.com/techexplorer/`). You can't use MathML within Opera.

### Presentation MathML

Presentation MathML provides control over the display of mathematic notation in a web page. Thirty presentation elements and around 50 attributes allow you to encode mathematical formulas. Presentation MathML tries to map each presentation element to an element.

To start, Presentation MathML divides a formula into vertical rows using `<mrow>` elements. This basic element is used as a wrapper. Rows may contain other nested rows. Each `<mrow>` element usually has a combination of mathematical numbers (`<mn>`), mathematical identifiers (`<mi>`), and mathematical operators (`<mo>`).

This example represents $10 + (x \times y)^4$:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
"http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <mrow>
   <mn>10</mn>
   <mo>+</mo>
   <msup>
```

```
      <mfenced>
        <mrow>
          <mi>x</mi>
          <mo>*</mo>
          <mi>y</mi>
        </mrow>
      </mfenced>
      <mn>4</mn>
    </msup>
  </mrow>
</math>
```

In the preceding document, you start with an XML declaration, adding the DOCTYPE declaration for MathML and including the `<math>` document element. The document includes a default namespace for the MathML vocabulary:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
"http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<math xmlns="http://www.w3.org/1998/Math/MathML">
```

Next, the document includes an `<mrow>` element, which represents the horizontal row of the equation. The row begins with the number `10` and includes a mathematical additional operator +:

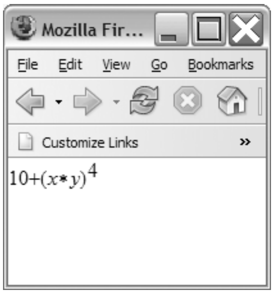
```
<mrow>
  <mn>10</mn>
  <mo>+</mo>
```

It then includes an `<msup>`, or mathematical superscript, section. This section allows the display of exponents and the `<mn>` element before the closing `</msup>` element indicates that the contents are raised to the power of `4`.

The `<msup>` element includes an `<mfenced>` element, which corresponds to the use of brackets in a mathematical equation. Within the brackets, the equation multiplies `x` by `y`:

```
<msup>
  <mfenced>
    <mrow>
      <mi>x</mi>
      <mo>*</mo>
      <mi>y</mi>
    </mrow>
  </mfenced>
  <mn>4</mn>
</msup>
```

You'll find this document saved as `mathml_presentation.mml` with the code download resources. I also could have saved it with a `.xml` file extension. Figure 3-6 shows the effect of opening this document in Firefox 1.5.



$10+(x*y)^4$

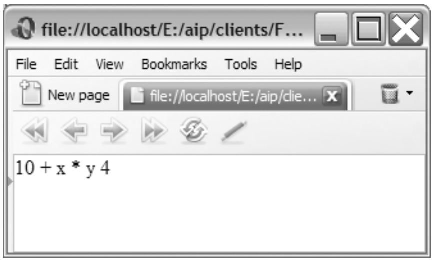
**Figure 3-6.** *A Presentation MathML document displayed in Firefox 1.5*

---

**Note** Firefox may prompt you to install some additional fonts from `http://www.mozilla.org/projects/mathml/fonts/`. Installing these fonts ensures that Firefox can render all mathematical symbols in your MathML document correctly.

---

If you try to view this document in a browser that doesn't support MathML, such as Opera 8.5, you'll see something similar to the image shown in Figure 3-7.

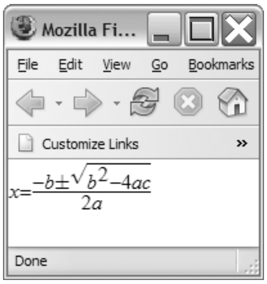

file://localhost/E:/aip/clients/F...
File Edit View Bookmarks Tools Help
New page
file://localhost/E:/aip/die...

```
10 + x * y 4
```

**Figure 3-7.** *A Presentation MathML document displayed in Opera 8.51*

Notice that the browser doesn't render the markup correctly. It doesn't insert the parentheses or raise the exponent. Essentially, it ignores all of the MathML elements and displays only the text within the XML document.

You can find a slightly more advanced example in the file `quadratic_equation_presentation.mml`. You need to install the Firefox MathML-enabled fonts in order to see the square root sign rendered correctly, as shown in Figure 3-8.



$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$

**Figure 3-8.** *Firefox showing a more complicated MathML page*

## Content MathML

Content MathML allows you to be very explicit about the order of operations and primary equation representation. Content markup has around 100 elements and 12 attributes.

Content MathML documents begin in the same way as Presentation MathML documents. They also contain `<mrow>` elements to separate the lines of the equation. However, Content MathML elements don't use the `<mo>` element for mathematical operators. Instead, they use the `<apply>` element and specific operator and function elements. This becomes clearer when you look at the same example written in Content MathML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
"http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <apply>
      <plus/>
      <ci>10</ci>
      <apply>
        <power/>
        <apply>
          <times/>
          <ci>x</ci>
          <ci>y</ci>
        </apply>
        <cn>4</cn>
        </apply>
    </apply>
  </mrow>
</math>
```

You can find the document saved as `mathml_content.mml` with your resources. Let's walk through the example.

The document starts with an XML declaration, a DTD reference, and the document root, including the MathML namespace. Then, like the Presentation XML example, you include an `<mrow>` element:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
"http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
```

From here on, the similarity ends. The example uses the `<apply>` element with `<plus/>` to include the addition operator with the value `10`:

```
<apply>
  <plus/>
  <ci>10</ci>
```

Another `<apply>` element surrounds the `<power/>` element, and the value of `4` is indicated immediately before the corresponding closing element:

```
<cn>4</cn>
```

The x × y section is contained within a third `<apply>` block that uses the `<times/>` element to indicate multiplication:

```
<apply>
  <times/>
  <ci>x</ci>
  <ci>y</ci>
</apply>
```

The differences are obvious. Instead of `<mi>` and `<mn>` elements, the vocabulary uses `<ci>` and `<cn>`. There is no need for the `<mfenced>` element because you can be specific about the order of operations by using the `<apply>` element.

In the preceding example, all of the operators use postfix notation. In postfix notation, you indicate the operation first and then follow that by the operand(s). Some MathML functions use postfix notation, and some don't. For a complete listing, see `http://www.w3.org/TR/MathML2/appendixf.html`.

You can't view this document in the web browser because that's not the purpose of Content MathML. Instead, it's supposed to be processed by a MathML engine, which may also perform the calculation. Most web browsers simply ignore all of the elements and only display the text, as you saw in the earlier Opera example.

## Scalable Vector Graphics

SVG was developed so that designers could represent two-dimensional graphics using an XML vocabulary. Just as MathML provides a detailed model to represent mathematical notation, SVG allows for the display of graphics with a high level of detail and accuracy. Again, because SVG is an XML vocabulary, it must follow the rules of XML. You can find out more about SVG at `http://www.w3.org/Graphics/SVG/`.

SVG has wide acceptance and support with many available viewers and editors. Both Firefox 1.5 and Opera 8 support SVG in some form, as does Amaya. For other browsers, you need to use plugins such as Adobe's SVG Viewer to view SVG documents. You can download the Adobe SVG Viewer plugin from `http://www.adobe.com/svg/`.

You can find the current SVG specification version 1.1 at `http://www.w3.org/TR/SVG11/`. The SVG 1.2 specification is currently under development.

You can break down SVG into three parts:

- Vector graphic shapes
- Images
- Text

Let's look at each of these in more detail.

## Vector Graphic Shapes

Vector graphics allow you to describe an image by listing the shapes involved. In a way, they provide instructions for creating the shapes. This is in contrast to *bitmap* or *raster* graphics, which describe the image one pixel at a time. Because you store vector graphics as a set of instructions, these images are often much smaller than their raster-based counterparts.

In SVG, you can represent vector graphics using either basic shape commands or by specifying a list of points called a *path*. You can also group objects and make complex objects out of more simple ones.

To get an idea about how you can work with shapes, let's look at an SVG document that describes a basic rectangle:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
xmlns="http://www.w3.org/2000/svg">
  <desc>A simple rectangle with a red border</desc>
  <rect x="10"
        y="10"
        width="200"
        height="200"
        fill="none"
        stroke="red"
        stroke-width="10"/>
</svg>
```

This file is saved as `svg_rectangle.svg`. Opening it in an SVG viewer or SVG native browser shows something similar to the image in Figure 3-9.

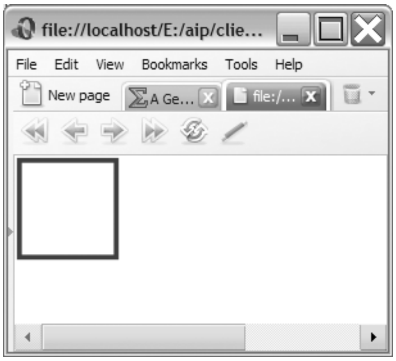

file://localhost/E:/aip/clie...
File Edit View Bookmarks Tools Help
New page
A Ge...
file:/...

**Figure 3-9.** *A simple SVG document displayed in Opera 8.51*

The document starts with an XML and DOCTYPE declaration and includes a document element called `<svg>`. Notice that the document element includes a reference to the SVG namespace, as well as attributes determining the size:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
xmlns="http://www.w3.org/2000/svg">
```

In addition to creating basic shapes, SVG allows you to add complex fill patterns and other effects, as you can see in this example:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
xmlns="http://www.w3.org/2000/svg">
  <desc>A simple rectangle with a red border and a gradient fill</desc>
  <g>
    <defs>
      <linearGradient id="RedGradient" gradientUnits="objectBoundingBox">
        <stop offset="0%" stop-color="#F00" />
        <stop offset="100%" stop-color="#FFF" />
      </linearGradient>
    </defs>
    <rect x="10"
          y="10"
          width="200"
          height="200"
          fill="url(#RedGradient)"
          stroke="red"
          stroke-width="10"/>
  </g>
</svg>
```

I've saved this document as svg_rectangle_fill.svg. When viewed in an appropriate viewer, it appears as shown in Figure 3-10.

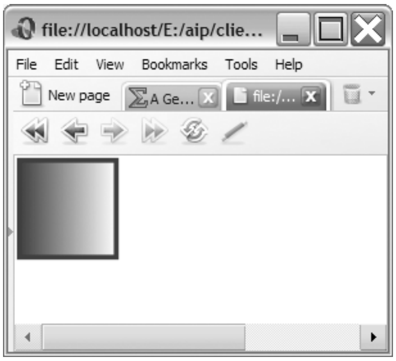

file://localhost/E:/aip/clie...
File Edit View Bookmarks Tools Help
New page
A Ge...
file:/...

**Figure 3-10.** *A shape with a fill shown in Opera 8.51*

This example creates a linear gradient in the <g> graphic object element called RedGradient:

```
<linearGradient id="RedGradient" gradientUnits="objectBoundingBox">
  <stop offset="0%" stop-color="#F00" />
  <stop offset="100%" stop-color="#FFF" />
</linearGradient>
```

The rectangle element then specifies that you should use the RedGradient fill element:

```
<rect x="10"
    y="10"
    width="200"
    height="200"
    fill="url(#RedGradient)"
    stroke="red"
    stroke-width="10"/>
```

The SVG 1.1 specification allows you to create the following basic shapes: <rect>, <circle>, <ellipse>, <line>, <polyline>, and <polygon>.

## Images

You also can include raster graphics in an SVG page. You might need to do this if you want to include an image of a person or landscape, or any other photo-realistic image, that you can't represent adequately as a vector drawing.

Including images in SVG is very simple:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="282px" height="187px" viewBox="0 0 282 187"
```

```
    xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>This SVG document contains lions.jpg</desc>
  <image x="0"
         y="0"
         width="282px"
         height="187px"
         xlink:href="lions.jpg">
    <title>Two lions</title>
  </image>
</svg>
```

This file is saved as `lions.svg`. Figure 3-11 shows how it renders in Firefox.

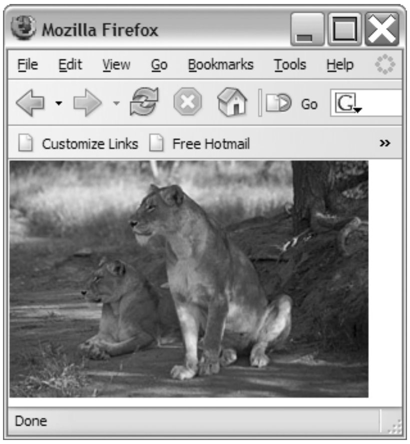


**Figure 3-11.** *An SVG page showing an image of lions*

The markup is self-explanatory. You can control how the image is displayed by changing the attributes in the SVG document. It's important to realize that the image isn't converted to a vector graphic. Instead, it maintains its original raster format and is drawn to the SVG display.

## Text

In addition to creating basic shapes and including images, SVG documents can represent text. This example creates text that has a color gradient outline:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat-20030114.dtd">
<svg width="20cm" height="4cm" viewBox="0 0 400 400"
xmlns="http://www.w3.org/2000/svg">
  <desc>This SVG document contains rainbow text</desc>
  <g>
    <defs>
      <linearGradient id="RedBlueGradient" gradientUnits="objectBoundingBox">
        <stop offset="0%" stop-color="#F00" />
        <stop offset="100%" stop-color="#00F" />
      </linearGradient>
    </defs>
    <text x="-600"
         y="200"
         font-size="128"
         fill="white"
         stroke="url(#RedBlueGradient)"
         stroke-width="5">
      SVG creates gradient text!
    </text>
  </g>
</svg>
```

This file appears as `svg_gradienttext.svg` with your resources. Figure 3-12 shows how it appears when open in an SVG viewer.

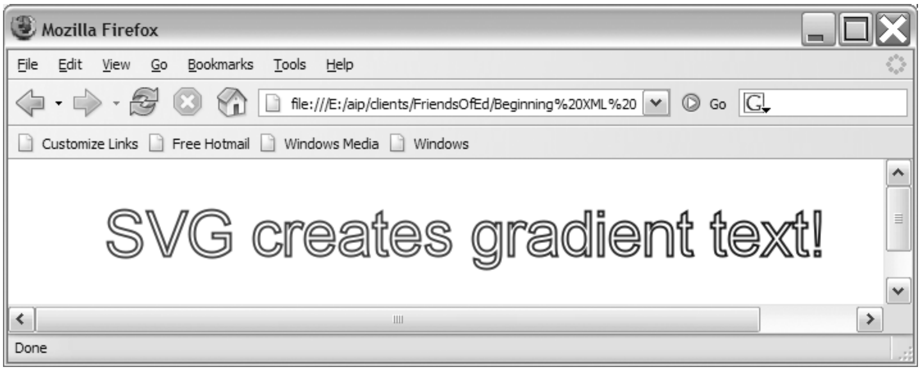

Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///E:/aip/clients/FriendsOfEd/Beginning%20XML%20
Go
Customize Links Free Hotmail Windows Media Windows
SVG creates gradient text!
Done

**Figure 3-12.** *Gradient text created with an SVG document*

The simple examples you've seen so far are only the beginning of what you can achieve with SVG. Let's move on to a more complicated example involving animation.

## Putting It Together

SVG allows you to create animations, and in the next example, I'll create an animation for the imaginary "Mars Travel" web site. The completed file is saved as `marstravel.svg` with your resources. Note that you won't be able to view the page with Mozilla unless you use a plugin. Mozilla's native support doesn't extend to SVG animations.

The page starts with declarations:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat-20030114.dtd">
<svg width="16cm" height="9cm" viewBox="0 0 1000 600"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Mars Travel introduction</desc>
```

These declarations add the XML and DOCTYPE declarations and set the size of the drawing. I've used the `<desc>` element to add a description for the page.

In the next step, I've added an image for the background. Make sure that you save the resource file, `mars.jpg`, in the same location as the svg file:

```
<image x="650" y="100" width="250" height="250" xlink:href="mars.jpg"/>
```

The first animation occurs in the next section of the SVG document:

```
<rect width="300" height="100" fill="rgb(200,200,200)"
  fill-opacity="0.25">
  <animate attributeName="y" attributeType="XML" from="500" to="-100"
  dur="4s" repeatCount="indefinite" fill="freeze" />
</rect>
```

The lines create a `<rect>` object and fill it with a medium-gray color. The `fill-opacity` is set to `0.25`. This attribute accepts values between `0` (completely transparent) and `1` (completely opaque).

The block also includes an `<animate>` element that modifies the `y` attribute from the value `500` to the value `-100`. This moves the block in an up-and-down motion.

The element specifies that the animation lasts for four seconds with the `dur` attribute and that it repeats indefinitely using `repeatCount="indefinite"`. The `fill="freeze"` attribute specifies that the fill doesn't change during the animation.

In this example, I've made the effect more interesting by adding six more moving `<rect>` objects that cross one another:

```
<rect width="300" height="400" fill="rgb(200,200,200)" fill-opacity="0.5">
  <animate attributeName="y" attributeType="XML" from="600" to="-400"
  dur="14s" repeatCount="indefinite" fill="freeze" />
</rect>
<rect width="300" height="14" fill="rgb(200,200,200)" fill-opacity="0.25">
  <animate attributeName="y" attributeType="XML" from="600" to="-40"
  dur="3s" repeatCount="indefinite" fill="freeze" />
</rect>
<rect width="300" height="4" fill="rgb(200,200,200)" fill-opacity="0.75">
  <animate attributeName="y" attributeType="XML" from="500" to="-4"
  dur="2s" repeatCount="indefinite" fill="freeze" />
</rect>
  <rect width="300" height="300" fill="rgb(200,200,200)" fill-opacity="0.75">
  <animate attributeName="y" attributeType="XML" from="-300" to="500"
  dur="8s" repeatCount="indefinite" fill="freeze" />
</rect>
<rect width="300" height="14" fill="rgb(200,200,200)" fill-opacity="0.75">
  <animate attributeName="y" attributeType="XML" from="-90" to="510"
  dur="3s" repeatCount="indefinite" fill="freeze" />
</rect>
<rect width="300" height="4" fill="rgb(200,200,200)" fill-opacity="0.75">
  <animate attributeName="y" attributeType="XML" from="-100" to="500"
  dur="2s" repeatCount="indefinite" fill="freeze" />
</rect>
```

The rectangles are partly transparent, so they produce some interesting effects as they overlap. If you test the document now, you'll see something similar to the screen shot shown in Figure 3-13.

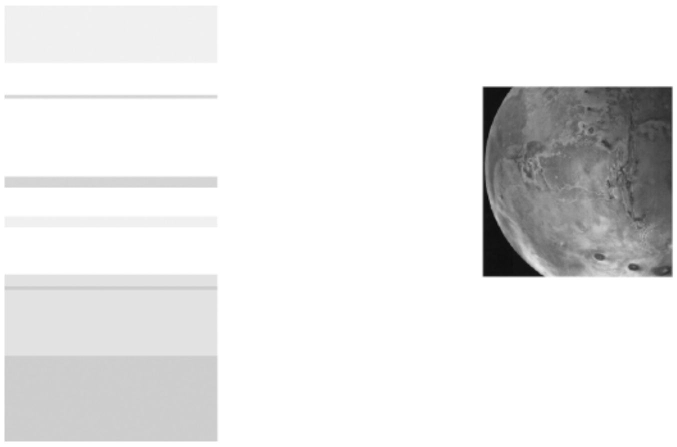


**Figure 3-13.** *The SVG animation so far*

The next block of code adds some text and vertical separators:

```
<!-- Default text -->
<text x="295" y="575" text-anchor="end">Scalable Vector Graphics</text>
<text x="295" y="590" text-anchor="end">by Mars Travel</text>
<!-- Separator -->
<line x1="300" y1="0" x2="300" y2="600" stroke-width="2" stroke="gray"/>
<line x1="305" y1="0" x2="305" y2="600" stroke-width="1" stroke="gray"/>
```

The `<text>` element has the attribute `text-anchor` set to `end`. This is the equivalent of aligning the text to the right. If the SVG viewer you're using has right-to-left reading enabled, the SVG aligns the text to the left. In either case, it aligns it to the "end" of the area.

The following line animates the title of the site so that it flies in from the right side:

```
<text x="1000" y="200" font-size="32" font-style="italic" font-weight="bold"
font-family="verdana" fill="#C65B2E">
  <animate attributeName="x" attributeType="XML" begin="0s" dur="2s"
  fill="freeze" from="1000" to="340"/>
  Mars Travel
</text>
```

The `<text>` element lists the text properties and also includes the `<animate>` element so that the text moves in from the right. It takes two seconds for the text to arrive at its final position.

The next code block adds some more text that enters after the “Mars Travel” text:

```
<text x="1000" y="224" font-size="24" font-style="italic" font-weight="bold"
font-family="verdana" fill="#C65B2E" >
  <animate attributeName="x" attributeType="XML" begin="2.5s" dur="2s"
  fill="freeze" from="1000" to="340" />
  Out of this world!
</text>
```

Finally, the page completes with a `<text>` element and closing `<svg>` tag. The text is linked so that users can visit the rest of the web site:

```
  <a xlink:href="http://www.apress.com/">
    <text x="750" y="467" fill="#C65B2E" font-weight="bold"
    font-family="verdana" font-size="24">ENTER >>></text>
  </a>
</svg>
```

This completes the SVG page. When you view it, you should see an animated version of the screen shot shown in Figure 3-14.

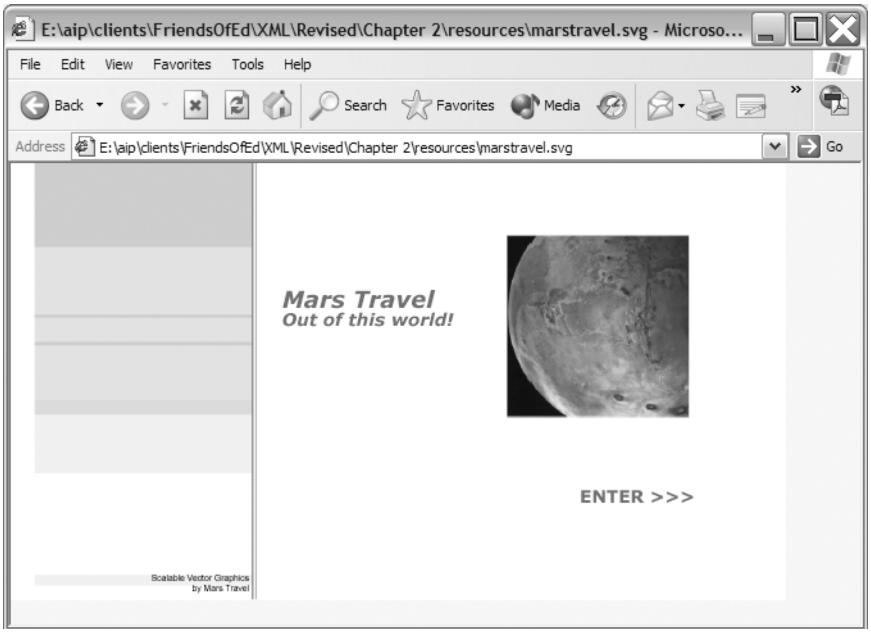

E:\aip\clients\FriendsOfEd\XML\Revised\Chapter 2\resources\marstravel.svg - Microso...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address E:\aip\clients\FriendsOfEd\XML\Revised\Chapter 2\resources\marstravel.svg Go
Mars Travel
Out of this world!
ENTER >>>
Scalable Vector Graphics by Mars Travel

**Figure 3-14.** *The completed SVG animation*

Figure 3-14 shows the page displayed in Internet Explorer; I can view the SVG file in this browser because I have the Adobe SVG Viewer plugin installed. You could also view the page using the native SVG support in Opera 8.5 or in any other browser that has an SVG plugin installed. You should probably provide an alternative image for viewers who don't have this plugin or an appropriate browser.

Even though this SVG introduction is graphically rich, it isn't inaccessible to people with disabilities. As you've seen, SVG documents can include the `<desc>` element, which provides an accessible text-based description of the document.

Let's move on to two more XML vocabularies that you can use with Web services: WSDL and SOAP.

## Web Services

Web services allow organizations to use the Internet to provide information to the public through XML documents. You can see examples of web services at Amazon and Google, where developers can interact with live information from the databases of both companies.

You have a number of different choices for working with web services, but all deliver their content in an XML document. When someone receives this information, it's called "consuming" a web service.

In this section, you'll briefly look at two of the XML vocabularies that impact the area: Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP). Both of these sections are more technical than the previous vocabularies that you've seen in this chapter.

Let's begin with WSDL. You won't need to be able to write this language yourself, as it's usually generated automatically. However, I'll explain the WSDL file, as it's useful to understand its structure.

### WSDL

WSDL is an XML vocabulary that describes web services and how you can access them. A WSDL document lists the operations or functions that a web service can perform. A web programming language usually carries out these operations in an application that isn't accessible to the consumer. The WSDL file describes the data types as well as the protocols used to address the web service.

Microsoft, Ariba, and IBM jointly developed WSDL. They submitted the WSDL 1.1 specification to the W3C as a note. The W3C accepted the note, which you can see at `http://www.w3.org/TR/wsdl`. The W3C is currently working on the WSDL 2.0 recommendation. You can see the primer for the working draft at `http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/`.

You normally don't write the WSDL file yourself using XML tools. Instead, your web services toolkit usually generates the file automatically. However, understanding the structure of the WSDL document can be useful.

## Understanding WSDL Document Structure

WSDL files are stored in locations that are accessible via the web. Anyone consuming the web service accesses these files. For example, you can find the Google web search WSDL at `http://api.google.com/GoogleSearch.wsdl`.

A WSDL document starts with an optional XML declaration and contains the `<types>`, `<message>`, `<binding>`, and `<service>` elements. The following code block shows the file structure of a WSDL file:

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions>
  <types>
    <!-- datatype definitions -->
  </types>
  <message>
    <!-- message definitions -->
  </message>
  <portType>
    <operation>
      <!-- operation definitions -->
    </operation>
  </portType>
  <binding>
    <!-- binding definitions -->
  </binding>
..<service>
..</service>
</definitions>
```

Table 3-1 explains each of the sections.

**Table 3-1.** *The Major Elements Within a WSDL File*

| Element | Explanation |
|---|---|
| `<definitions>` | Provides the root element for the WSDL document and contains the other elements |
| `<types>` | Defines the data types used by the web service |
| `<message>` | Describes the messages used when the web service is consumed |
| `<portType>` | Combines messages to create the library of operations available from the web service |
| `<operation>` | Defines the operations that the web service can carry out |
| `<binding>` | Lists the communication protocols that a user can use to consume the web service and the implementation of the web service |
| `<service>` | Defines the address for invoking the web service—usually a URL to a SOAP service |

## Defining Web Service Data Types

When someone consumes a web service, the service receives the request, queries an application, and sends an XML document containing the results in response. In order to use the web service, the consumer must know how to phrase the request as well as the format for the returned information. It's crucial to understand the data types used by the web service.

The WSDL document defines the data types for both the inputs to and the outputs from the web service. These might equate to the data types listed in the XML schema recommendation, or they could be more complicated, user-defined data types.

If you're only using W3C built-in simple data types, the WSDL file doesn't include the `<types>` element. The XML schema namespace appears in the `<definitions>` element and references data types in the `<message>` elements:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

Custom data type definitions appear in the `<types>` element. The WSDL file can use XML schema declarations or any alternative schema system for defining these data types:

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <!-- schema declarations here -->
..</schema>
</types>
```

## Mapping Data Types to Messages

A consumer calls the web service and provides inputs. These inputs map to `<message>` elements. Each message has `<part>` elements that refer to each of the inputs received:

```
<message name="mName">
  <part name="mInputName" type=" mInputNameType"/>
</message>
```

The types referred to in the `<message>` elements must come from one of the schema namespaces within the document. If the type refers to the simple built-in data types from the XML schema recommendation, the element includes a reference to the XML schema namespace:

```
<message name="mName">
  <part name="mNameIO" type="xsd:string"/>
</message>
```

## Listing Web Service Operations

The most important element in the WSDL document is the `<portType>` element. This element defines all of the operations that are available through the web service. The `<portType>` element is like a library of all of the available operations.

The `<portType>` element contains `<operation>` elements that have `<input>` and `<output>` elements. Inputs pass to an application for processing. The outputs are the responses received from the application that are passed to the consumer:

```
<portType name="ptName">
  <operation name="oName">
    <input message="oNameRequest"/>
    <output message="oNameResponse"/>
  </operation>
</portType>
```

The `<message>` elements define the inputs and outputs. They are normally prefixed with the current document's namespace.

A web service can carry out four types of operations. The most common is the *request-response* type. In this type, the web service receives a request from a consumer and supplies a response. A web service can also carry out a *one-way* operation, where a message is received but no response is returned. In this case, the operation has an `<input>` element.

The other options are *solicit-response*, where the web service sends a message and then receives a response. It is the opposite of a one-way operation. The operation has an `<output>` element followed by an `<input>` element. You can also specify a `<fault>` element. The final option is *notification*, where the service sends a message and only has an `<output>` element.

## Mapping to a Protocol

The `<portType>` element contains all of the operations for a web service. Bindings specify which transport protocol each `portType` uses. Transport protocols include HTTP POST, HTTP GET, and SOAP. You can specify more than one transport protocol for each `portType`. Each binding has a name and associated type that associates with a `portType`.

If you're using SOAP 1.1, WSDL 1.1 includes details specific to SOAP. The binding specifies a `<soap:binding>` element, which indicates that the binding will use SOAP. This element requires style and transport attributes. The style attribute can take values of `rpc` or `document`.

Document style specifies an XML document call style. Both the request and response messages are XML documents. `rpc` style uses a wrapper element for both the request and response XML documents.

The `transport` attribute indicates how to transport the SOAP messages. It uses values such as

```
http://schemas.xmlsoap.org/soap/http
http://schemas.xmlsoap.org/soap/smtp
```

The following example specifies a SOAP 1.1 transport mechanism over HTTP using an `rpc` interaction:

```
<binding name="bName" type="bType">
  <soap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
  <!-- declarations-->
  </soap:binding>
</binding>
```

The web service binds each operation using the following format. The operation name corresponds with the operation defined earlier in the `<portType>` element. The `soapAction` attribute shows the destination URI including a folder, if necessary:

```
<soap:operation name="oName" soapAction="URI">
  <input>
    <soap:body use="literal"/>
  </input>
  <output>
    <soap:body use="literal"/>
  </output>
</soap:operation>
```

You can also specify an optional SOAP encoding for each operation.

### Specifying Processing Software

The `<service>` element shows where to process the requested operation. The service has a `name` attribute and a child `<port>` element. The `<port>` element specifies a `portType` for binding. The `<port>` element also has a `name` attribute.

If you're using SOAP, the `<soap:address>` element specifies the location of the processing application:

```
<service name="sName">
  <port binding="portTypeName" name="pName">
    <soap:address
    location="URI/>
  </port>
</service>
```

The file can also include a `<documentation>` element as a child of `<service>` to provide a human-readable description of the service.

### Viewing a Sample WSDL Document

The concepts behind a WSDL file are easier to understand with an example. The following example shows a simple fictitious WSDL document:

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions name="Author"
targetNamespace="http://www.apress.com/wsdl/Authors.wsdl
xmlns:tns="http://www.apress.com/wsdl/Authors.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <message name="getAuthorRequest">
    <part name="book" type="xsd:string"/>
  </message>
  <message name="getAuthorResponse">
    <part name="author" type="xsd:string"/>
  </message>
  <portType name="authorRequest">
```

```
    <operation name="getAuthor">
      <input message="tns:getAuthorRequest"/>
      <output message="tns:getAuthorResponse"/>
    </operation>
  </portType>
  <binding name="authorSOAPBinding" type="tns:authorRequest">
    <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getAuthor">
      <soap:operation
      soapAction="http://www.apresscom/getAuthor"/>
        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
    </operation>
  </binding>
  <service name="authorSOAPService">
    <port binding="tns:authorSOAPBinding" name="Author_Port">
      <soap:address
      location="http://www.apress.com:8080/soap/servlet/rpcrouter/">
    </port>
  </service>
</definitions>
```

Notice that this WSDL file contains a number of namespaces:

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions name="Author"
targetNamespace="http://www.apress.com/wsdl/Authors.wsdl
xmlns:tns="http://www.apress.com/wsdl/Authors.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

The `targetNamespace` in the document element allows the document to reference itself. It uses a prefix of `tns` for the namespace. The document element includes the default WSDL namespace `http://schemas.xmlsoap.org/wsdl/` as well as a reference to the XML schema namespace `http://www.w3.org/2001/XMLSchema`.

The WSDL document includes two `<message>` elements—one request and one response. The data types are the built-in `xsd:string` types:

```
<message name="getAuthorRequest">
  <part name="book" type="xsd:string"/>
</message>
<message name="getAuthorResponse">
  <part name="author" type="xsd:string"/>
</message>
```

The `<portType>` contains a single operation called `getAuthor`. The `getAuthor` operation has both input and output messages, which correspond to the string `<message>` elements:

```
<portType name="authorRequest">
  <operation name="getAuthor">
    <input message="tns:getAuthorRequest"/>
    <output message="tns:getAuthorResponse"/>
  </operation>
</portType>
```

The binding specifies the SOAP 1.1 protocol over HTTP using the `rpc` style:

```
<binding name="authorSOAPBinding" type="tns:authorRequest">
  <soap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getAuthor">
    <soap:operation soapAction="http://www.apress.com/getAuthor"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
  </operation>
</binding>
```

The application addressed by the web service is located at `http://www.apress.com:8080/soap/servlet/rpcrouter/`:

```
<service name="authorSOAPService">
  <port binding="tns:authorSOAPBinding" name="Author_Port">
    <soap:address
    location="http://www.apress.com:8080/soap/servlet/rpcrouter/">
  </port>
</service>
```

You're not likely to have to write WSDL documents yourself, but understanding how they work can be useful. You can see an example of a more complicated WSDL file at `http://soap.amazon.com/schemas2/AmazonWebServices.wsdl`.

The next section explains the SOAP protocol, one of the most popular ways to consume a web service.

## SOAP

SOAP is another XML vocabulary that works with web services. You can send SOAP messages using HTTP and even email.

When consuming a SOAP web service, the consumer sends a SOAP message to a receiver, who acts upon it in some way. For example, the SOAP message could contain a method name for a remote procedure call. The receiver could run the method on a web application and return the results to the sender.

In the simplest situation, the SOAP message involves a message between two points: the sender and the receiver. The number of messages could increase if the receiver has to send back another SOAP message to clarify the original request. A further SOAP message would then be required to respond to the clarification request. You also can send a SOAP message via an intermediary who acts before sending the message to the receiver.

The SOAP 1.2 primer is available on the W3C web site at `http://www.w3.org/TR/2003/REC-soap12-part0-20030624/`. You also can see the messaging framework at `http://www.w3.org/TR/2003/REC-soap12-part1-20030624/` and the adjuncts at `http://www.w3.org/TR/2003/REC-soap12-part2-20030624/`. The "SOAP Version 1.2 Specification Assertions and Test Collection" document is available at `http://www.w3.org/TR/2003/REC-soap12-testcollection-20030624/`.

## Creating a SOAP Message

SOAP messages are XML documents that conform to the SOAP schema. Because SOAP is an XML vocabulary, a SOAP document must be well formed. A SOAP message can optionally include an XML declaration, but it can't contain a DTD or processing instructions.

The document element of a SOAP message is the `<Envelope>` element. It encloses all other elements in the message and must contain a reference to the `soap-envelope` namespace:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
```

This namespace refers to the SOAP 1.2 specification. If the SOAP processor receiving the message expects a SOAP 1.1 message, it generates an error. You should match the namespace and SOAP version. For SOAP 1.1, use

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
```

Each SOAP message is different. It includes the parameters that are required for the operation. You can include a schema for the SOAP message so that you ensure that the contents are valid. A schema allows both the sender and the receiver to understand the format for the request and response. You can see the schema for a SOAP 1.2 message at `http://www.w3.org/2003/05/soap-envelope/`.

## Understanding the Contents of a SOAP Message

SOAP messages have the following format:

- The root `<Envelope>` element identifies the message as a SOAP message.
- The `<Body>` element contains the content for the end destination.
- The `<Header>` and `<Fault>` elements are optional.

The following code shows the structure of a SOAP message:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <!-- Optional header information -->
  </env:Header>
  <env:Body>
    <!-- Body information -->
    <env:Fault>
      <!-- Optional fault information -->
    </env:Fault>
..</env:Body>
</env:Envelope>
```

## Explaining SOAP Headers

The SOAP `<Header>` element includes information additional to that required by the SOAP receiver. It's optional, but if it's present, it must appear directly after the `<Envelope>` element.

The header often includes machine-generated information such as dates and times and unique session identifiers. Any child element within a `<Header>` element must be qualified with a namespace.

You can include the `mustUnderstand` attribute in a header to require that the receiver must be able to interpret the header:

```
<env:Header>
  <e:Element xmlns:e="http://www.apress.com"
  env:mustUnderstand="True">
    <!--Element content-->
  </e:Element>
</env:Header>
```

You can also use a value of `1`:

```
<e:Element xmlns:e="http://www.apress.com" env:mustUnderstand="1">
```

The processor can only process the message if it understands all elements where the value of the `mustUnderstand` attribute is `True`. If it doesn't, it returns an error message and ignores the rest of the SOAP message.

A SOAP message may pass through other points on the way to its final destination. The intermediate points may need to act on some of the headers in the message. You use the `actor` attribute to address the element to an intermediary:

```
<env:Header>
  <e:Element xmlns:t="http://www.apress.com"
  env:mustUnderstand="True"
  env:actor="http://www.apress.com/wsxml/">
</env:Header>
```

## Understanding the SOAP Body

The `<Body>` element contains the information intended for the final destination. Any information contained in this element is mandatory. Child elements of the `<Body>` element can include a namespace declaration.

The information contained in the body must be well formed and must conform to the WSDL for the web service. In other words, the information must reference the operations set out in the WSDL. The following code shows a sample `<Body>` element:

```
<env:Body>
  <b:getAuthor xmlns:b="http://www.apress.com/bookauthor">
    <b:book>Beginning XML with DOM and Ajax</b:book>
  </b:getAuthor>
</env:Body>
```

In this fictitious example, the `<Body>` element makes a `getAuthor` request. This request takes one parameter `<book>`. In the example, you request the author details for the book `Beginning XML with DOM and Ajax`. The namespace `http://www.apress.com/bookdetails` qualifies the `getAuthor` request.

The body of the returned information might look something like this:

```
<env:Body>
  <b:getAuthorResponse xmlns:b="http://www.apress.com/
  bookauthor">
    <b:Author>Sas Jacobs</b:Author>
  </b:getAuthorResponse>
</env:Body>
```

## Examining the Fault Element

The optional `<Fault>` element provides information on faults that occurred when the message was processed. If present, it must contain two elements: `<Code>` and `<Reason>`. It can also contain an optional `<Detail>` element:

```
<env:Envelope>
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>Value here</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Error reason here</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

If there is a fault, the web service sends a fault message instead of a response. A SOAP processor can't return both a response and a fault.

### Explaining SOAP Encoding

You can include an optional `<encodingStyle>` element in your SOAP message. For SOAP 1.2, use the following:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
xmlns:enc="http://www.w3.org/2003/05/soap-encoding/"
env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
```

You use the following format for SOAP 1.1:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
xmlns:enc=" http://schemas.xmlsoap.org/soap/encoding/"
env:encodingStyle=" http://schemas.xmlsoap.org/soap/encoding/">
```

The namespaces include definitions for the data types that you can use with SOAP encoding.

Let's summarize:

- The WSDL vocabulary describes web services and their operations.
- WSDL isn't a W3C recommendation; rather, it was developed by Microsoft, Ariba, and IBM.
- A WSDL file is usually generated automatically rather than being written by a human.
- SOAP is an XML vocabulary that allows someone to consume a web service.
- There are different versions of SOAP. At the time of writing, the latest is version 1.2.
- SOAP messages request and receive information from web services.

We'll finish this chapter by looking at some of the other web XML vocabularies.

## Other Web Vocabularies

I've given you a brief introduction to some of the most popular web vocabularies: XHTML, MathML, SVG, WSDL, and SOAP. These vocabularies are only the tip of the iceberg, and new vocabularies appear regularly. In this section, I'll list some additional web vocabularies and provide a brief description of their use.

### RSS and News Feeds

Really Simple Syndication or RDF Site Summary (RSS), commonly used in news feeds, is like a web service that works specifically with news. Companies such as The Associated Press (AP) and United Press International (UPI) make international stories available via RSS. You can find news feeds for each of them at `http://www.newsisfree.com/syndicate.php`. Smaller web sites can also provide news in this way.

There are many different versions of the RSS specification. The current version is RSS 3, and you can find out more about at `http://www.rss3.org/main.html`.

## VoiceXML

VoiceXML is a W3C recommendation designed to represent aural communications on the web. VoiceXML includes support for voice-synthesizing software, digitized audio, and command-and-response conversations, among others.

The VoiceXML vocabulary is surprisingly easy to understand:

```
<?xml version="1.0"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form>
    <field name="gender">
      <prompt>Are you female or male?</prompt>
      <grammar src="gender.grxml" type="application/srgs+xml"/>
    </field>
    <block>
      <submit next="gender.asp"/>
    </block>
  </form>
</vxml>
```

Using grammar documents to specify the expected responses to a user's input, you can quickly create verbal forms to interact with users. You can find out more about VoiceXML at `http://www.w3.org/Voice/`.

## SMIL

SMIL (Synchronized Multimedia Integration Language) is an XML vocabulary for authoring interactive multimedia presentations. The acronym, pronounced *smile*, is a W3C recommendation. You can find out more at `http://www.w3.org/AudioVideo/`.

Like VoiceXML, SMIL is a relatively easy vocabulary to understand. It allows you to describe the layout of items on the screen, as well as the timing and synchronization of items in the presentation.

SMIL documents can support the following media types: images, video, audio, animation, text, and textstream. You need a SMIL player or Internet Explorer 6 for Windows to be able to view your presentations.

## Database Output Formats

Although database formats aren't explicitly web vocabularies, you may encounter them in your development. Some popular formats include

- Microsoft Access
- Microsoft SQL Server
- Oracle XML DB
- IBM Informix
- IBM DB2 Universal Database
- Sybase

Each of these formats is different, but stylesheets are available that can handle the conversion from one type of database to another. Most of these databases have the ability to export their data directly as XML. Additionally, some tools can extract the information and format it as XML. I'll show you examples of using XML with databases in Chapters 12 and 13.

## Summary

This chapter presented an introduction to several XML vocabularies. I examined XHTML, the primary vocabulary in use on the web today. I also discussed SVG, MathML, and vocabularies involved with web services, along with some other, less well-known vocabularies.

In the chapters that follow, you'll learn how to use some of the common vocabularies of XML, and learn how they work together to create XML applications.