Foundation XML and E4X for Flash and Flex

Sas Jacobs

Foundation XML and E4X for Flash and Flex

ISBN-13 (pbk): 978-1-4302-1634-6

ISBN-13 (electronic): 978-1-4302-1635-3

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

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 www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at http://www.apress.com/info/bulksales.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editor
Ben Renow-Clarke

Technical Reviewer
Kevin Ruse

Editorial Board
Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Project Manager
Beth Christmas

Copy Editor
Marilyn Smith

Associate Production Director
Kari Brooks-Copony

Production Editor
Ellie Fountain

Compositor
Patrick Cunningham

Proofreader
Lisa Hamilton

Indexer
Broccoli Information Management

Artist
April Milne

Cover Image Designer
Corné van Dooren

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski

I'd like to dedicate this book to my Grandma Lucy, who died while I was writing it. You are a beautiful person, and I miss you very much.

CONTENTS AT A GLANCE

CONTENTS

ABOUT THE AUTHOR

Sas Jacobs is a web developer and author who works with Flash, Flex, and XML. Sas has written several books on these topics and has spoken about them at conferences such as Flashforward, webDU, and FlashKit. Nowadays, Sas works as a software developer in the area of e-learning, where she tries to share her passion for all things ActionScript.

When she's not working, Sas loves traveling, photography, running, and her son.

ABOUT THE TECHNICAL REVIEWER

Kevin Ruse is the principal of Kevin Ruse and Associates Inc., a web and print design and consulting firm based in Santa Clara, California. Kevin has been a trainer in web development and graphic design in a variety of environments, including De Anza Community College and the University of California, Santa Cruz. Kevin has also taught the staff and faculty at Stanford University and University of California, Berkeley.

Kevin is an Adobe Certified Instructor and a Certified Training Partner for the Altova XML Suite of software and the <oXygen/> XML editor. He currently teaches the following languages and software: Flex, Fireworks, Flash, Dreamweaver, Photoshop, InDesign, Acrobat, Quark XPress, JavaScript, ActionScript, MXML, XML, XSLT, DTD/Schema, ColdFusion, HTML, XHTML, and CSS.

Kevin is the author of *Web Standards Design Guide*, a college textbook. He is an enthusiastic instructor who maintains a strong belief that with patience, determination, and guidance, all individuals can reach their maximum potential.

ABOUT THE COVER IMAGE DESIGNER

Corné van Dooren designed the front cover image for this book. After taking a brief from friends of ED to create a new design for the Foundation series, he worked at combining technological and organic forms, with the results now appearing on this and other books' covers.

Corné spent his childhood drawing on everything at hand and then began exploring the infinite world of multimedia, and his journey of discovery hasn't stopped since. His mantra has always been, "The only limit to multimedia is the imagination," a saying that keeps him moving forward constantly.

Corné works for many international clients, writes features for multimedia magazines, reviews and tests software, authors multimedia studies, and works on many other friends of ED books. You can see more of his work at and contact him through his web site, `www.cornevandooren.com`.

If you like Corné's work, be sure to check out his chapter in *New Masters of Photoshop: Volume 2* (friends of ED, 2004).

ACKNOWLEDGMENTS

Thanks again to all the people at friends of ED for your hard work in putting this book together. You're a great team and, as always, it has been a pleasure working with you.

INTRODUCTION

This book started out as an update to my first book on Flash and XML. Originally, the idea was to update the content with the changes to XML in ActionScript 3.0. However, when it came to drafting the table of contents, I realized that there was a whole audience of Flex developers who would also benefit from a book about XML and ActionScript 3.0. Hence, this book was born!

So, my plan is for this book to cater to both audiences: Flash designer/developers and Flex developers. I've included common code approaches, as well as topics that are specific to each package. I've tried to show readers how to achieve the same XML results in both software packages.

This book is best suited to people who have limited experience in the areas of XML and ActionScript 3.0. It is really pitched at introductory level users who are keen to learn more about ActionScript 3.0. The book is purposely simple in its approach, showing how to achieve common tasks required for working with XML in Flash and Flex. The Flash sections show function-based approaches, whereas the Flex sections show how to work with custom classes.

I hope that you find this book useful and that it whets your appetite for working with XML in your SWF applications. Hopefully, you'll find that the power and simplicity of XML will inspire you in your Flash and Flex development efforts!

Layout conventions

To keep this book as clear and easy to follow as possible, the following text conventions are used throughout:

- Important words or concepts are normally highlighted on the first appearance in *italics*.
- Code is presented in `fixed-width font`.
- New or changed code is normally presented in **`bold fixed-width font`**.
- Pseudo-code and variable input are written in *`italic fixed-width font`*.
- Menu commands are written in the form Menu ➤ Submenu ➤ Submenu.
- Where I want to draw your attention to something, I've highlighted it like this:

> *Ahem, don't say I didn't warn you.*

- Sometimes code won't fit on a single line in a book. Where this happens, I use an arrow like this: ➥.

```
This is a very, very long section of code that should be written all
➥on the same line without a break.
```

Chapter 1

INTRODUCTION TO XML

If you work in the web design or development area, you've probably heard of XML. It's the basis of all modern web sites, but how many of us actually know what it really means?

This chapter introduces XML and explains why it is such an important standard for exchanging information. I'll cover some of the basic concepts behind XML, including the rules governing the structure of XML documents. I'll also review some of the uses for XML and the reasons you might need to use it in your SWF applications. You'll see some examples of XML documents and, by the end of the chapter, have a solid understanding of XML and its related concepts.

If you're already familiar with XML and are comfortable generating XML documents, feel free to skip forward to Chapter 3. If not, read on! You can download the resources referred to in this chapter from `http://www.friendsofed.com`.

What is XML?

Let's start by answering the most basic question of all: what is XML? It's very difficult to provide a short answer to this question. The people who invented XML, the World Wide Web Consortium (W3C), provide the following definition for XML in their glossary at `http://www.w3.org/TR/DOM-Level-2-Core/glossary.html`.

Extensible Markup Language (XML) is an extremely simple dialect of SGML. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

I think the definition is very accurate if you already know about XML, but it doesn't really explain XML to a novice. Let's go back to basics and see what that definition really means.

Understanding XML

XML describes a format that you can use to share information. By itself, XML doesn't do anything other than store information. It's not a programming language, so you can't use it to create stand-alone applications. XML simply describes information. XML documents need humans or software packages to process the information that they contain.

XML stands for Extensible Markup Language, which is a little misleading, because XML is actually a metalanguage, so you can use it to create other markup languages. That's what the word *extensible* means in the acronym. The term *markup* means that the languages you create use tags to surround or mark up text, which you'll be familiar with if you write Extensible Hypertext Markup Language (XHTML).

In fact, XHTML is one of the languages created by XML. We call XHTML a *vocabulary* of XML. It was created when HTML was redefined according to XML rules. For example, in XHTML, all tags must be in lowercase. This requirement doesn't apply to HTML.

Think about the tags you use in XHTML, such as and <h1></h1>. These tags mark up information that will display on a web page. You use these in a very specific way, according to some predefined rules. For example, one rule says that you can't include tags in the <head> section of a web page.

It's possible to make up your own XML vocabulary or work within a group to create an industry-wide language based on XML. By agreeing on an XML vocabulary, groups can share information using a common set of markup tags and structures. Chemical Markup Language (CML), for example, allows scientists to share molecular information in a standardized way. There are specific rules for structuring CML documents and referring to molecular information. MathML is another vocabulary of XML that describes mathematical operations. But you don't need to work with an existing vocabulary to include XML in your applications. It's possible—in fact, very likely—that you'll create your own XML structures to suit the particular needs of your application. You'll see this as we work through the examples in the book.

So what type of information can you store in an XML document?

Storing information in XML documents

XML documents work best with structured information, similar to what you would find in a database. Examples include lists of names and addresses, product catalogs, sales orders, an iTunes library—anything with a standardized format. Like a database, XML documents can show hierarchical relationships. Instead of breaking the information into tables and fields, elements and tags describe the data. By nesting tags inside each another, you can create a hierarchy of parent and child elements.

The following code block shows some sample XML elements. You can see the hierarchical relationship between the elements, and the tag names describe their contents.

```
<contact>
  <name>Sas Jacobs</name>
  <address>123 Red Street, Redland, Australia</address>
  <phone>123 456</phone>
</contact>
```

The code sample describes contact details, similar to what you would see in an address book. Most of us have an address book that we use to store contact information about our friends and colleagues. You might have the information in a software package like Outlook or Outlook Express. It might also exist on your mobile phone.

Your address book contains many different names, but you store the same information about each contact: name, phone number, and address. The way the information is stored depends on the software package you've chosen. If the manufacturer changes the package or discontinues it, you'll need to find a new way to store information about your contacts.

Transferring the information to a new software program is likely to be difficult. You'll need to export it from the first package, rearrange the contents to suit the second package, and then import the data. Most software applications don't share a standard format for contact data, although some can share information. You must rely on the standards created by each company.

As an alternative, you could use an XML document to store the information. You could create your own tag names to describe the data. Tags like `<contact>`, `<phone>`, and `<address>` would provide clear descriptions for your information. Any human who read the document would be able to understand what information the document held.

Because the address book XML document has a standard format, you could use it to display your contacts on a web page. Web browsers contain an XML parser to process the XML content. You could also print out your contacts, or even build a SWF movie in Flash or Flex to display and manage your contacts.

Your friends could agree on which tags to use and share their address books with each other. You could all save your contacts in the same place and use tags to determine who had contributed each entry. When you use a standardized structure for storage, you have endless choices about how to work with the information.

So if XML is so useful, how did it all start?

XML, in the beginning

XML has been around since 1998. It is based on Standard Generalized Markup Language (SGML), which was in turn was created out of Generalized Markup Language (GML) in the 1960s. XML is actually a simplified version of SGML.

SGML describes how to write languages, specifically those that work with text in electronic documents. It is also an international standard: ISO 8879. SGML was actually one of the considerations for HTML when it was first developed.

The first XML recommendation was released in February 1998. Since then, XML has increased in popularity and is now a worldwide standard for sharing information. Human beings, databases, and many popular software packages use XML documents to store and exchange information. Web services and RSS feeds also use an XML format to share content over the Internet.

The W3C developed the XML specification. The W3C also works with other recommendations such as HTML, XHTML, and Cascading Style Sheets (CSS). Detailed information about the XML specification is available from the W3C's web site at `http://www.w3c.org/XML/`. The current specification is XML 1.1 (Second Edition), dated 16 August 2006. You can view this specification at `http://www.w3.org/TR/2006/REC-xml11-20060816/`.

The W3C has also created a family of related recommendations that work together to create an independent framework for managing markup languages. The other areas covered by recommendations include XML schemas, which describe the structure and syntax of an XML document; XML stylesheets, which allow the transformation and output of XML content; and XPath, which describes how to navigate or locate specific parts of XML documents.

When it created XML, the W3C published the following goals at `http://www.w3.org/TR/REC-xml/#sec-origin-goals`:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

In other words, XML should be easy to use in a variety of settings, by both people and software applications. The rules for XML documents should also be clear so they are easy to create.

An XML example

The following code block shows a simple XML document with address book data containing a single contact. If you've worked with XHTML, you'll see that the elements are written in a similar way.

```
<?xml version="1.0"?>
<phoneBook>
  <contact id="1">
    <name>Sas Jacobs</name>
    <address>123 Some Street, Some City, Some Country</address>
    <phone>123 456</phone>
  </contact>
</phoneBook>
```

The information is stored between tags, and the names of these tags are descriptive—for example, <name>, <address>, and <phone>. The casing of the opening and closing tags is consistent. The hierarchy within the document shows that the information relates to a single <contact> element. You can use any names for these elements, as long as you follow the rules for constructing XML documents. These rules are presented in the "Structuring XML documents" and "Understanding well-formed documents" sections later in the chapter.

Now that you know a little more about XML, you may be wondering why it is so important and why might you want to use XML as a source of data.

Why XML?

Quite simply, XML is simple, flexible, descriptive, accessible, independent, precise, and free! Let's look at each of these advantages of XML in turn.

Simple

The rules for creating XML documents are very simple. You just need a text editor or another software package capable of generating XML formatted data. The only proviso is that you follow some basic rules so that the XML document is well-formed. You'll find out what this means a little later in the chapter, in the "Understanding well-formed documents" section.

Reading an XML document is also simple. Tag names are normally descriptive, so you can figure out what data each element contains. The hierarchical structure of elements allows you to work out the relationships between each piece of information.

Flexible

One key aspect of XML is its flexibility. As long as you follow some simple rules, you can structure an XML document in any way you like. The choice of tag names, attributes, and structures is completely flexible so you can tailor it to suit your data. You can also agree on an XML vocabulary so you can share information with other people. A Document Type Definition or schema describes the "grammar," or rules for the language.

XML documents provide data for use in different applications. You can generate an XML document from a corporate software package, transform it to display on a web site using Extensible Stylesheet Language Transformations (XSLT), share it with staff on portable devices, use it to create PDF files with Extensible Stylesheet Formatting Objects (XSL-FO), and provide it to other software packages. You can reuse the same data in several different settings. The ability to repurpose information is one of XML's key strengths.

> *XSLT and XSL-FO are two related XML recommendations. Both of these recommendations describe how to change or transform an XML document into a different type of output. You might use an XSLT stylesheet to create HTML or text from an XML document. You could use XSL-FO to create a PDF document.*

The way XML information displays is also flexible. You can display any XML document in any XML processor, perhaps a web browser, to see the structure of elements. You can also use the document to display the following:

- A printed list of summary data
- A web page displaying the full details of each element
- A SWF movie that allows you to search the XML content

Descriptive

Because you can choose your own tag names, an XML document becomes a description of your data. Some people call XML documents *self-describing*.

The hierarchy of elements means that XML documents show relationships between information in a similar way to a database. For example, the hierarchies in the address book document tell us that each contact has a name, address, and phone number, and that we can store many different contacts.

Accessible

XML documents separate data from presentation, so you can have access to the information without worrying about how it displays. This makes the data accessible to many different people, devices, and software packages. For example, the sample address book XML document could be accessed in the following ways:

- Read aloud by a screen reader
- Displayed on a web site
- Printed to a PDF file
- Processed automatically by a software package
- Viewed on a mobile phone

XML documents use Unicode for their standard character set, so you can write XML documents in any number of different languages. (The Unicode standard is maintained by the Unicode Consortium; see `http://www.unicode.org/`.) A SWF application could offer multilingual support simply by using different XML documents with equivalent content.

Independent

XML is platform- and device-independent. It doesn't matter if you view the data on a PC, Macintosh, or handheld device. The data is still the same, and people can exchange it seamlessly. Programmers can also use XML to share information between software packages that otherwise couldn't easily communicate with each other.

You don't need a specific software package to work with XML documents. You can type the content in just about any software package capable of receiving text. You can read the document in a web browser, text editor, or any other XML processor. XML documents can provide a text-based alternative to database content. In the case of web services, XML is an intermediary between you and someone else's database.

XML doesn't have "flavors" that are specific to a single web browser (like CSS), version, or operating system. You don't need to create three different versions of your XML document to handle different viewing conditions.

Precise

XML is a precise standard. If you want your XML document to be read by an XML parser, it must be well-formed. Documents that aren't well-formed won't display. You're probably wondering what *well-formed* means. We'll cover that a little later, in the "Understanding well-formed documents" section.

When a schema or Document Type Definition is included within an XML document, an XML processor can validate the content to make sure that the document structure conforms to the structural rules you've established. XML documents with schemas provide standards, so there is only one way that the data they contain can be structured and interpreted.

Free

XML is a specification that isn't owned by any company or commercial enterprise. This means that it's free to use XML—you don't have to buy any special software or other technology. In fact, most major software packages either support XML or plan to support it in the future.

So why should you use XML in your Flash and Flex projects?

Why is XML important in Flash and Flex?

XML is an important tool for all web developers. Many people consider XML the *lingua franca* of the Internet as it provides the standard for data exchange among humans and machines in many different settings.

An understanding of XML is essential for Flash and Flex developers for the following reasons:

- XML is one way for Flash and Flex developers to store content that powers SWF movies.
- Flex uses a vocabulary of XML, called MXML, to describe interfaces.
- XML provides a mechanism for working with data that is disconnected from a database or the Internet.
- ActionScript 3.0 contains features to make working with XML much easier than in previous versions, so it's a sound alternative to plain-text files for content.

XML as a SWF data source

Storing content outside a SWF application means that clients can update their own content without needing to learn either Flash or Flex, or contact the developer each time they want to make a change. It's also possible to provide client tools that make it easy to generate the content automatically.

Developers will understand the importance of storing information for SWF movies in an external data source. Doing so allows the content of a SWF application to change without the need to recompile it each time. Simply update the source document to update the information within the application.

There are many possible sources of external data: text files, databases, and XML documents. While it's possible to create external text files for a SWF file, it's more practical to use an XML document, which can include the data and describe the hierarchical relationships between the data.

Although a developer or client can create an XML document by hand, it's easier to generate the content automatically. With the assistance of a web server and a server-side language like PHP, Visual Basic .NET (VB .NET), or ColdFusion, databases can easily generate XML content suitable for a SWF application. You'll see how this can be done in the next chapter. In terms of security, it's good practice to use an XML layer between a user and database.

Many software packages are capable of exporting their content in an XML format. The most recent versions of Microsoft Office allow you to save Word, Excel, and Access documents using either Microsoft's XML vocabularies or your own. Using standard business tools to generate XML content allows clients to take control of their own application content.

For SWF applications that need to be portable, XML is an excellent choice as a data source. An XML document is usually small in file size, making it easy to include on a CD, DVD, or handheld device.

MXML in Flex

In order to get the most out of Flex, developers need a good understanding of XML. Flex uses an XML vocabulary called MXML to describe application interfaces. MXML is a markup language that provides the same role in Flex applications as XHTML does in web pages.

MXML consists of a set of tags that correspond to ActionScript 3.0 classes. Because MXML is a vocabulary of XML, it must follow the same rules and be well-formed. I'll cover this term in more detail later in the chapter.

ActionScript 3.0 and XML

ActionScript 3.0 greatly simplifies the process of working with XML documents compared with earlier versions. XML is a native data type in this version of ActionScript, making it much easier to work with in both Flash and Flex.

If you've worked with XML in an earlier version of ActionScript, you'll be used to writing complicated expressions and looping through collections of nodes to locate specific information in an XML document. The new process means that you can target content in an XML document by using element names instead. This change is significant and makes working with XML content much easier than in earlier versions of ActionScript. I have found that ActionScript 3.0 has saved me hours of development time when it comes to working with XML content. You'll find out more about the changes to ActionScript in Chapter 3.

ActionScript 3.0 also includes a full implementation of the ECMAScript for XML (E4X) standard, ECMA-357 (see `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-357.pdf`). Because ActionScript 3.0 adheres to international standards, you can take advantage of any existing knowledge you have in this area. Learning E4X means that you'll be able to apply the same skills when working with JavaScript.

Now that you appreciate why XML is important to Flash and Flex developers, let's look more closely inside an XML document.

XML document sections

It's important to understand exactly what the term *XML document* means. This term refers to a collection of content that meets XML construction rules. The *document* part of the term has a more general meaning than with software packages. In Flash or Flex, for example, a document is a physical file. While an XML document can be a physical file, it can also refer to a stream of information that doesn't exist in a physical sense. You can create these streams from a database using a web server, and you'll see how this is done later in the book. As long as the information is structured according to XML rules, it qualifies as an XML document.

An XML document is divided into two sections: the *prolog* and the content, or *document tree*. The content exists inside the *document root* or *root element*.

Document prolog

The document prolog appears at the top of an XML document and contains information about the XML document as a whole. It must appear before the root element in the document. The prolog is a bit like the <head> section of an XHTML document.

The prolog consists of the following:

- An XML declaration
- Processing instructions
- Document Type Definitions (DTDs)

XML declaration

The prolog usually starts with an XML declaration to indicate to humans and computers that the content is an XML document. This declaration is optional but if it is present, it must appear on the first line. At a minimum, the declaration appears as follows:

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

The minimum information that must appear inside an XML declaration is an XML version. The preceding declaration uses version 1.0.

> *At the time of writing, the latest recommendation is XML 1.1. However, you should continue to use the* `version="1.0"` *attribute value for backward-compatibility with XML processors, unless you specifically need version 1.1. For example, XML 1.1 allows characters that can't be used in XML 1.0 and has slightly different requirements for namespaces.*

The XML declaration can also include the `encoding` and `standalone` attributes. The order of these attributes is important.

Encoding determines the character set for the XML document. You can use Unicode character sets UTF-8 and UTF-16 or ISO character sets like ISO 8859-1, Latin-1, or Western European. If no `encoding` attribute is included, it is assumed that the document uses UTF-8 encoding. Languages like Japanese

and Chinese need UTF-16 encoding. Western European languages often use ISO 8859-1 to cope with diacritical characters, such as accent marks, that aren't part of the English language.

The `encoding` attribute must appear after the `version` attribute. Here are some sample declarations that include an `encoding` attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-16"?>
<?xml version="1.0" encoding="ISO-8859-1">
```

The `standalone` attribute indicates whether the XML document uses external information, such as a DTD. A DTD specifies the rules about which elements and attributes to use in the XML document. It also provides information about the number of times each element can appear and whether an element is required or optional.

The `standalone` attribute is optional. When it's included, it must appear as the last attribute in the declaration. The value `standalone="no"` can't be used when you are including an external DTD or stylesheet. Here is an XML declaration that includes this attribute:

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

Processing instructions

The prolog can also include processing instructions (PIs). Processing instructions pass information about the XML document to other applications that may need that information in order to process the XML.

Processing instructions start with the characters `<?` and end with `?>`. You can add your own processing instructions or have them generated automatically by software packages. The first item in a processing instruction is a name, called the processing instruction *target*. Processing instruction names that start with `xml` are reserved.

One common processing instruction is the inclusion of an external XSLT stylesheet. An XSLT stylesheet transforms the content of an XML document into a different structure, and I'll cover this topic in more detail later in this chapter, in the "Understanding XSL" section. A processing instruction that includes an XSLT stylesheet must appear before the document root. The following line shows how this processing instruction might be written:

```
<?xml-stylesheet type="text/xsl" href="listStyle.xsl"?>
```

Processing instructions can also appear in other places in the XML document.

Document Type Definitions

DTDs, or DOCTYPE declarations, can also appear in the prolog. A DTD provides rules about the structure of elements and attributes within the XML document. It explains which elements are legal in the XML document, and tells you which elements are required and which are optional. In other words, a DTD provides the rules for a valid XML document and explains how the document should be constructed.

The prolog can include a set of declarations about the XML document, a bit like an embedded CSS stylesheet in an XHTML document. The prolog can also include a reference to an external DTD as well as or instead of these declarations. The following shows an external reference to a DTD:

```
<?xml version="1.0"?>
<!DOCTYPE phoneBook SYSTEM "phoneBook.dtd">
```

All the other content in an XML document appears within the document tree.

Document tree

Everything that isn't in the prolog is contained within the document tree. The tree contains all of the content within the document. The section "Structuring XML content" explains exactly what items appear here.

The document tree starts with a document root or root element. An XML document can have only one root element. All of the content within the XML document must appear inside the document root. In HTML documents, the `<html>` tag is the root element. This is a rule of a well-formed document.

Whitespace

XML documents include whitespace so that humans can read them more easily. Whitespace refers to spaces, tabs, and returns that space out the content in the document. The XML specification allows you to include whitespace anywhere within an XML document except before the XML declaration.

> *XML processors can interpret whitespace in an XML document, but many won't actually display the spaces. If whitespace is important, there are ways to force an XML processor to display the spaces using the* `xml:space` *attribute in an element. I'll leave you to research that topic on your own if it's something you need to do.*

Namespaces

XML documents can get very complicated. One XML document can reference another XML document, and different rules may apply in each case. XML documents can also summarize content from multiple sources. For example, you might combine several different XML documents into one.

It's possible that an XML document will contain elements that use the same name but that come from different locations and have different meanings. For example, you might use the `<table>` element as part of an XHTML reference in a document about furniture, which also needs to use a `<table>` element as a description of the type of furniture.

In order to overcome this problem, you can use *namespaces* to distinguish between elements. Namespaces associate each XML element with an owner to ensure it is unique within a document, even if there are other elements that use the same name.

Each namespace includes a reference to a Uniform Resource Identifier (URI) as a way to ensure its uniqueness. A URI is an Internet address, and each URI must be unique in the XML document. The URIs used in an XML document don't need to point to anything, although they can.

You can define a namespace using the xmlns attribute within an element. Each namespace usually has a prefix that you use to identify elements belonging to that namespace. You can use any prefix that you like, as long as it doesn't start with xml and doesn't include spaces.

Here is an example of using a namespace:

```
<FOE:details xmlns:FOE="http://www.friendsofed.com/ns/">
  <name>Sas Jacobs</name>
</FOE:details>
```

In the <details> element, the FOE prefix refers to the namespace http://www.friendsofed.com/ns/. You can also use this prefix with other elements and attributes to indicate that they are part of the same namespace, like this:

```
<FOE:address>
  123 Some Street, Some City, Some Country
</FOE:address>
```

This prefix indicates that the <address> element also comes from the http://www.friendsofed.com/ns/ namespace.

You can also define a namespace without using a prefix. If you do this, the namespace will apply to all elements that don't have a prefix or namespace defined. It is referred to as the *default namespace*.

In an XHTML document, the <html> element includes a namespace without a prefix:

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

The namespace then applies to all of the child elements of the <html> element; in other words, all of the remaining elements in the XHTML document.

It isn't compulsory to use namespaces in your XML documents, but it can be a good idea. Namespaces are also important when you start to work with schemas and stylesheets. ActionScript 3.0 provides mechanisms for working with namespaces when dealing with complex XML documents in SWF applications.

You can find out more about namespaces by reading the latest recommendation at the W3C site. At the time of writing, this was the Namespaces in XML 1.1 (Second Edition) recommendation at http://www.w3.org/TR/xml-names11/.

Structuring XML documents

XML documents contain both information and markup. The information about the document appears in the prolog, as discussed in the previous section. You can divide markup into the following:

- Elements
- Attributes
- Text
- Entities