Introduction to XML with JAXP

This chapter is an overview of some of the basic things you will need to know before you can understand the processing of XML documents using Java's Java API for XML Processing (JAXP). Although the Java application programming interface (API) is rather straightforward, you will need to understand how XML is constructed before you can clearly understand the sort of things that can be done with the API. The fundamental concepts described in this chapter include the fact that, like HTML, the XML language is derived from SGML. This kinship between XML and HTML has brought about the existence of a hybrid known as XHTML. There are two completely distinct parsers, named DOM and SAX, that can be used to read the contents of an XML document.

Java and XML

Java was designed to make it possible to write completely portable programs. XML was designed to make it possible to create completely portable data. In an ideal situation, using the two together will make for a completely portable software package that can communicate its data with any other completely portable software package. Nothing is absolutely perfect, but these two, used together, come about as close as anything that has been developed so far in terms of the ability to write a program that runs on any type of computer and can swap data with any other type of computer.

The JAXP package is a set of Java classes that implements XML document parsers, supplies methods that can be used to manipulate the parsed data, and has special transformation processors to automate the conversion of data from XML to another form. For example, the other form can be a database record layout ready for storage, an HTML Web page ready for display, or simply a textual layout ready for printing.

One of the outstanding features of XML is its fundamental simplicity. Once you understand how tags are used to create elements, it is easy to manually read and write XML documents. With this basic XML understanding, and with knowledge of the Java language, it is a straightforward process to understand the relationship between XML and the Java API for manipulating XML. There are only a few classes in this API, and it is only a matter of creating the appropriate set of objects, and they will supply the methods you can call to manipulate the contents of an XML document. With these basic concepts understood, and with the simplicity of the constructs involved, you can design and write programs while concentrating mostly on the problem you are trying to solve, not on the mechanics of getting it done.

Java, the Language

Following are some characteristics of Java that make it ideal for use as a language to manipulate XML documents:

- **The JAXP is now a part of standard Java.** It contains all of the classes and interfaces that you need for parsing and processing an XML document. It also contains methods that can be used to automate the transformation process of converting an XML document into an entirely different form.
- **The fundamental Java stream IO can be used for input of XML documents and output of the results of processing.** This means that your application is able to process files stored remotely on the Internet just as easily as the ones on the local disk. Once a stream has been established to a file, the rest of the application can use streaming input and output without having to know anything about the location at the other end of the stream. You can write your application just once and know that it will work no matter how the data needs to be fed to or extracted from it.
- **The majority of installed Web servers are capable of running Java applications to dynamically generate Web pages**. This means that, using the JAXP, the set of Java classes that provide the methods to be used to manage XML documents, it becomes a very simple matter to transform data from an XML format to HTML format as a response to a request made from a remote Web browser. All of the software you need, from receiving the request through formatting the data to transmitting the response, is ready and waiting. About all that is left to do is decide how you want the Web page to appear and then write the Java code to lay it out.
- **Portability applies to anything written in Java**. Using Java and its built-in JAXP allows you to run the application on any computer that has a Java Virtual Ma-

chine installed. And, because XML is also portable, the result is an almost universally portable system and can be used in exactly the same way on any computer.

To fully understand the concepts discussed in the following paragraphs and chapters, you should be familiar with Java, or familiarize yourself with the Java programming language using Java tutorials. To understand how these classes do their jobs, you will only need to understand Java classes, objects, interfaces, and methods. There is nothing more complicated than a static method returning an object that implements an interface; if you understand these fundamentals, you will have no problem with anything in this book.

XML, the Language

XML stands for Extensible Markup Language. These three words are actually a very accurate description. It is a nonprocedural programming *language*, which means that things written in the language are not so much commands as they are descriptions of a condition or state. Like almost all programming languages, XML is written as human-readable text, in such a form that humans as well as programs can read and understand the instructions.

The XML language is used to *mark up* a document so that the reader (usually a program) can identify each piece of the document and determine its characteristics by examining the tags it contains. A tag can be named anything you would like, but it only has meaning if the program reading the document already knows the tag name. XML is also *extensible* because you can invent as many markup tags as you need as you go along; all you need to do is make sure the reader of the document knows the meanings of your tags. In fact, there are no markup tags defined as part of XML. The creator of an XML document invents whatever tags are necessary for a full description of the document being marked up.

There is a common misconception that the purpose of XML is to format and display data. That is not what it is for. Its purpose is to store data in a form that can be easily read and analyzed. It is quite common to use XML to store data and use the descriptive XML tags to specify how it should be displayed, but this not an inherent part of XML. It is also very common to write applications that convert XML data into HTML for display. And, because XML and HTML are so similar in their basic syntax, it is possible to use the tag names defined for HTML in an XML document and then use a Web browser to display it as if it were HTML. A special name for this type of XML document is XHTML, but it is just a special case of XML.

A file, or other entity, containing XML-formatted data is referred to as a *document*. This term carries a broad interpretation because XML is used to format many different kinds of information, some of which is never intended for human use (such as data being transferred from one database to another). The following examples show how tags can be used to mark up documents. An XML document can be for any purpose and can take any form it needs to fit that purpose, but generally speaking, there are two categories of XML documents. An XML document can be storage for text that is in-

tended to be formatted and presented in a readable format. Or it can be a convenient form for packaging data records for transmission from one place to another, or simply for storage, in a portable format. The following is an example of text that can be formatted for display:

```
<paragraph>
The purpose of this type of XML document is to use
<italic>tags</italic> in such a way that the software that
reads the document will be able to <underline>organize</underline>
and <underline>format</underline> the text in such a way that
it is more presentable and easier to read.
</paragraph>
```

This form of XML looks a lot like HTML. In fact, this form of XML and HTML both serve exactly the same purpose: to allow the software reading the document to extract things from it and also to use the tags as formatting instructions to create a display from the extracted text.

The same basic form can be used to package data, as in the following example:

```
<person>
  <name>Karan Dirsham</name>
  <street>8080 Holly Lane</street>
  <city>Anchor Point</city>
  <state>Alaska</state>
  <zip>99603</zip>
</person>
```

This second form is more like a collection of fields that go to make up a data record, and used this way, it can be a very convenient method for storing data and transmitting information among otherwise incompatible systems. All that is necessary for successful data reception of transmitted data is for the recipient to understand the meanings of the tags and be able to extract the data from them. Of course, by using the appropriate application to read and process the data, any XML document can be easily formatted for display. The process of extraction and formatting XML data is the primary subject of this book.

Attributes can be used to specify options that further refine the meaning of the tags to the process reading the document. These attributes can be used both for data definition and for formatting. For example, the following code has attributes:

```
<person font="Courier">
  <name type="first" enhance="bold">Janie</name>
  <name type="last" enhance="underline">Rorick</name>
</person>
```

Any program reading this document can apply its own interpretation to the meanings of the tags and the options. No formatting information is included in an XML document. All formatting is left entirely to the process reading and interpreting the XML document. One program could read a document containing this example and take the

bold option to mean a different font, another could take it to mean a larger font, and another could use it as an instruction to underline the text or display it in a different color. Or the bold option could be ignored altogether. The only thing XML knows about the attribute is the syntax required to include it with a tag.

If you have worked with HTML, you can see the similarities in the syntax of HTML and XML. They are similar enough that it is possible to write an XML document and use only tag names known to a particular Web browser and then have that Web browser read the document and impose its interpretations on the tags and options and result in a displayed page. A displayable XML document is written often enough that a document of this type has a special name; it is called XHTML. There is more information about XHTML later in this chapter.

XML is a *nonprocedural* computer language, as opposed to a *procedural* language such as Java. A procedural language is one that consists of lists of instruction that are expected to be obeyed one by one, usually in order from top to bottom. A nonprocedural language is one that expects all of its instructions to be executed as if they were all being executed simultaneously and, if necessary, to react to one another to create an overall state or set of states. An example of nonprocedural processing is a spreadsheet in which all the cells in the sheet that contain equations have their values calculated at once, creating a static state of constant values displayed in the cells. This same sort of thing happens in a Web browser where the HTML tags define the state (layout, colors, text, pictures, fonts, and so on) that determines the appearance of a Web page.

DTD

DTD stands for Document Type Definition. Although DTD is normally treated separately from XML tags, has a different syntax, and serves a different purpose, it is very much a part of the XML language. The DTD section of an XML document is used to define the names and syntax of the elements that can be used in the document. There are several steps involved in the creation and application of a DTD definition, and Chapter 2 contains explanations of those steps along with a number of examples. Its source can be included inline inside an XML document, or it can be stored in a separate document from the XML text and tags. Because its purpose is to specify the correct format of a marked up document, DTD is most useful if it is made available to several documents and is most often stored in a file separate from any XML document, which enables it to be accessed from any number of XML documents. For the sake of simplicity, however, most of the examples in this book have a simple DTD included as part of the XML document.

DTD enables you to further refine the syntactical requirements of a set of XML markup rules. In a DTD you can specify the allowed and disallowed content of each tag that is to appear in an XML document. That takes at least a third of Chapter 2. DTD has its limitations, but there are many different things that can be done by using it. You can specify which elements are allowed to appear inside other elements as well as which elements are required and where they are required. You can specify which attributes are valid for each tag and even specify the set of possible values for each one. You can

create macro-like objects (called *entities*) that are expanded into text as the XML document is parsed. All of this is explored in Chapter 2.

An XML document that conforms to the fundamental syntax of markup tags is called *well-formed*. To be a well-formed document, all elements must have matching opening and closing tags, and the tags must be nested properly. For example, the expression `<p><b>text</b></p>` is well-formed because the opening tags, `<p><b>`, have closing tags, `</b></p>`, and they are nested properly. To be well-formed every closing tag must be a match with the most recent tag that is still open. In short, all tags must be closed in the exact reverse order in which they were opened. The expressions `<p><b>text</b>` and `<p><b>text</p></b>` are not well-formed.

An XML document that conforms to the rules of its DTD is referred to as a *valid* document. For a document to be successfully tested as being valid, it must also be well-formed. Some parsers can check the document against the DTD definitions and throw an exception if the document is not valid.

Use of DTDs is very important to the portability of XML. If a DTD is well written (that is, if all the tags are defined properly), a process can be written that will be able to read and interpret any XML data from any document that conforms to the rules of the DTD.

A single XML document can use more than one DTD. However, this multiple DTD use can result in a naming collision. If two or more DTDs define a tag by the same name, they will more than likely define that tag as having different characteristics. For example, one could be defined as requiring a font attribute, whereas the other has no such attribute. This problem is solved by using device known as a *namespace*. An element specified as being from one namespace is distinct from one of the same name from another namespace. For example, if a pair of DTDs both include a definition for an element with the tag name `selectable`, one DTD could be declared in the namespace `max` and the other in the namespace `scrim`; then there would be the two distinct tag names `max:selectable` and `scrim:selectable` available for use in the document. Examples using namespaces are explained in Chapter 2.

XSL

XSL stands for XML Stylesheet Language. It is used as a set of instructions for the translation of the content of an XML document into another form—usually a presentation form intended to be displayed to a human. An XSL program is actually, in itself, a document that adheres to the syntax of XML. It contains a set of detailed instructions for extracting data from another XML document and converting it to a new format. Performing such transformation is the subject of Chapter 8.

The process of using XSL to change the format of the data is known as *transformation*. Transformation methods are built into the JAXP that can be used to perform any data format translation you define in an XSL document. These transformations can be programmed directly into Java instead of using XSL, but XSL simplifies things by taking care of some of the underlying mechanics, such as walking through the memory-resident parse tree to examine the source document. It also supplies you with some

built-in methods for doing commonly performed tasks such as configuring the parser and handling error conditions.

XSL performs a function—supplying human-readable data—that is every bit as important as XML itself. With the single exception of robotics, all the software in the world is ultimately used to display data to humans. Nothing is ever stored in a database without the expectation that it will be extracted and presented in some human-readable format. In fact, presenting readable data is the entire purpose of the Internet. Operating systems and computer language compilers only exist to support and create other programs that, in turn, directly or indirectly present data in a form that can be understood by humans.

SAX

SAX stands for Simple API for XML. It is a collection of Java methods that can be used to read an XML document and parse it in such a way that each of the individual pieces of the input are supplied to your program. It is a very rudimentary form of parsing that is not much more than a lexical scan: It reads the input, determines the type of things it encounters (it recognizes the format of the nested tags and separates out the text that is the data portion of the document), and supplies them to your program in the same order in which they appear in the document.

The form of the data coming from a SAX parser can be very useful for streaming operations such as a direct translation of tags or text into another form, with no changes in order. If your application needs to switch things around, however, it will be necessary for it to keep copies of the data so it can be reorganized. In many cases, it would be easier to parse using the DOM parser. The SAX parser has the advantage of being fast and small because it doesn't hold anything in memory once it has moved on to the next input item in the input document.

There are two versions of SAX. The original version is SAX 1.0 (also called SAX1). The current version is SAX 2.0 (also called SAX2). SAX2 is an extension of the definitions of SAX 1.0 to include things such as the ability to specify names using namespaces. Both SAX1 and SAX2 are a part of JAXP. Because SAX1 is still a part of JAXP, programs based on it will work , but much of it has been deprecated in the API to promote use of SAX2 in all newly written programs. Only SAX2 is discussed in the following chapters because it does everything SAX1 does and more.

DOM

DOM stands for Document Object Model. It is a collection of Java methods that enable your program to parse an input document into a memory-resident tree of nodes that maintains the relationships found in the original input document. There are also methods that enable your application to walk freely about the tree and extract the information stored there.

The internal form of the data tree resulting from a DOM parse is quite convenient if you are going to be accessing document content out of order. That is, if your program needs to rearrange the incoming data for its output, or if it needs to move around the document and select data in random-access order, you should find that the DOM document tree will provide what you need for doing this. You can search for things in the tree and pull out what you need without regard to where it appeared in the input document. One disadvantage of DOM is that a large document will take up a lot of space because the entire document is held in memory. With modern operating systems, however, the document would need to be extremely large before it would adversely affect anything. DOM also has the disadvantage of being more complicated to use than SAX. Because DOM can randomly access the stored data, the API for it is necessarily more complex. Although DOM is more complicated to use than SAX, it can be used to do much more. For more details about how DOM works, see Chapters 3, 6, and 7.

Internally in the JAXP, the DOM parser actually uses SAX as its lexical scanner. That is, a SAX parser is used to read the document and break it down into a stream of its components, and the DOM software takes this token stream and constructs a tree from it. This is why it is best to have an understanding of SAX before trying to get a clear idea of how JAXP DOM works. Although you may never use SAX directly, it's a good idea to know how it works and how the incoming document is broken down. At the very least, you will need to be familiar with the meaning of its error messages and how to process them in your application, which means you will need to know how SAX works. For more details, see Chapters 3, 4, and 5

SGML

SGML stands for Standard Generalized Markup Language. This is the parent markup language of XML and HTML, which were both derived as special-purpose subsets of SGML. Included in the 500-page SGML specification document is a definition of the system for organizing and tagging elements in a document. It became a standard with the International Organization of Standards (ISO) in 1986, but the specification had actually been in use some time before that. It was designed to manage large documents so that they could be frequently changed and also printed. It is a large language definition and too difficult to actually implement, which has resulted in the subsets XML and HTML.

XML works well being a subset of SGML because the complexity of SGML isn't necessary to do all of the tagging and transforming that needs to be done. Being a practical subset makes it much easier to write a parser for XML. Because of the reduction in complexity of the language, XML documents are smaller and easier to create than SGML documents would be. For example, where SGML always requires the presence of a DTD, in XML the DTD is largely optional. If you are going to validate the correctness of an XML document, the DTD is necessary, but otherwise it can be omitted.

XML is a bit closer to being like SGML than is HTML. For one thing, HTML is filled with ambiguities because it allows things like an opening tag that has no closing tag to match it. This prevents any attempts to standardize HTML because a parser cannot

predict what it will find. And many HTML extensions and modifications apply in one place but do not apply in another. Although XML is extensible—you can add all the tag types you wish—it is very strict in the way it allows you to do it. Like SGML, the formatting of XML can be controlled by XSL documents used for transformations.

XHTML

XHTML stands for Extensible Hypertext Markup Language. An XHTML document is a hybrid of XML and HTML in such a way that it is syntactically correct for both of them. That is, although an XHTML document can be displayed by a Web browser, it can also be parsed into its component parts by a SAX or DOM parser. Both XML and HTML are subsets of SGML, so the only problem in combining the two into XHTML was in dealing with the places where HTML had departed from the standard format. Most obvious are the many opening tags in HTML that do not have closing tags to match them and the fact that tag nesting is not required.

XHTML was conceived so that, once Web browsers were capable of dealing with the strict and standard forms required for XML, a more standardized form of Web page could evolve. With XML it is relatively easy to introduce new forms by defining additional elements and attributes, and because this same technique is part of XHTML, it will allow the smooth integration of new features with the existing ones. This capability is particularly attractive because alternate ways of accessing the Internet are constantly being developed. The presence of a standard, parsable Web page will allow easier modification to the display format for new demands, such as the special requirements of hand-held computers.

There is a fundamental difference between XML and HTML. XML *is* an SGML, while HTML is an *application* of SGML. That is, SGML does not have any tag names defined and neither does XML. For both XML and SGML, a DTD must be used to define and provide meanings for element names. On the other hand, HTML has a set of element names already defined. The element names of HTML are the ones that have a meaning to the Web browser attempting to format the page. This fundamental difference between XML and HTML can be overcome by the creation of a DTD that defines the syntax for all the elements that are used in HTML. With such a DTD in place, an XML document that adheres to the DTD's definitions will also be an HTML document, and thus it can be displayed using a Web browser.

JAXP

JAXP stands for Java API for XML Processing. It is a set of Java classes and interfaces specifically designed to be used in a program to make it capable of reading, manipulating, and writing XML-formatted data.

It includes complete parsers for SAX1 and SAX2 and the two types of DOM: DOM Level 1 and DOM Level 2. Most of this book explores the use of parsers in extracting

data from an XML document. More information on both of these parsers is the subject of Chapter 3. There are examples of using the SAX parsers in Chapters 4 and 5 and of the DOM parsers in Chapters 6 and 7. All of the parsers check whether a document is well-formed, and the parsers can be used in validating or nonvalidating mode, as described in the DTD section earlier in this chapter. There is also an extensive API that can be used to access the data resulting from any of these parsers. And although SAX1 and DOM Level 1 are both present and working, the API for them is deprecated. Anything you need to do can be accomplished with just the SAX2 and DOM Level 2 API.

The simplest way to get a copy of JAXP is to download and install the latest copy of Java from Sun at http://java.sun.com. Beginning with Java version 1.4, the JAXP API is included as part of Java 2 Standard Edition. It is also a part of the Java 1.3 Enterprise Edition. If you want to use JAXP with a prior version of Java, you can get a copy of it from the Web site http://java.sun.com/xml. You can use JAXP version 1.1 with the Java Software Development Kit (SDK) version 1.1.8 or newer.

If, for some reason, you are staying with an older version of Java and downloading JAXP as a separate API, you will need to download the documentation separately. This documentation is in the form of a set of HTML pages generated from the source code by the standard javadoc utility. It is in the same format as the documentation for the rest of Java. There are two ways to install the standalone JAXP: You can include its jar files in the same directories as your Java installation, or you can install them in their own directories. If you do not elect to include them with Java, you will need to specify the `classpath` settings when either compiling or running a JAXP application.

Ant

Ant is a tool used to compile Java classes. It isn't limited to Java; it can be used for your entire software development project. It performs the same job as the traditional `make` utility by compiling only the programs that need to be compiled, but it has some special features that cause it to work very will with Java. For one thing, it understands the Java package organization and can use it when checking dependencies. It is an XML application and, as such, uses an XML file as its input control file (much like the `make` utility uses a `makefile`) to determine which modules to compile. The Java classes of Ant are available in source code form, so you can, by extending the existing classes, add any new commands and processing that you would like.

When compiling Java, Ant compares the timestamp of the source files to timestamp of the class files to determine which Java source files need to be compiled. Also, Ant knows about the relationship between Java directory trees and packages, so it is capable of descending your source tree properly to create classes within several packages.

For more information about Ant, with examples, see Chapter 9. The Web site for Ant is http://jakarta.apache.org/ant/.

Summary

This chapter is an overview of the basics. You should now have some idea of what JAXP is designed to do and a rough idea of how it does it. Using JAXP, a Java program is capable of reading an XML document and analyzing its various parts for meaning or formatting. There is more than one way to read a document: You can do it in a sequential manner with a SAX parser or browse about the document randomly by using a DOM parser.

The next chapter takes a detailed look at the syntax of an XML document. As you will see, its tags and content are straightforward and easy to read and write. The complexities of the syntax are all contained in the DTD portion. Chapter 2 describes the syntax and is designed to make it easy for you to use as a reference later.