Introduction to XML

Extensible Markup Language (XML) was standardized in 1998 after 2 years of work. However, it developed out of SGML (Standard Generalized Markup Language), a product of the 1970s and 1980s. SGML was standardized in 1986. It is very complicated, so a more useful subset was needed, and XML now handles many of the problems that motivated SGML.

XML is a markup language. That means that it annotates or marks up documents. Its main purpose is to provide information (semantics) about the contents of the document. It uses tags similar to those in html, but these tags are not specified in advance, rather they are created by the user.

An Example XML Document

The following is a very simple XML document.

```
<?xml version = "1.0" ?>
<address>
    <name>Alice Lee</name>
    <email>alee@aol.com</email>
    <phone>123-45-6789</phone>
    <birthday>1983-07-15</birthday>
</address>
```

Like in xhtml, all start tags must have matching end tags. However, while XML is case sensitive, it is not restricted to lower case. Most applications mix cases, such as in fullName or lastName. The naming requirements are similar to those in Java, but along with letters, digits, and underscores, names may include hyphens and periods. An XML document must follow these rules. If it does so, it is said to be *well-formed*. If it does not, it is not a true XML document.

Tree Structure

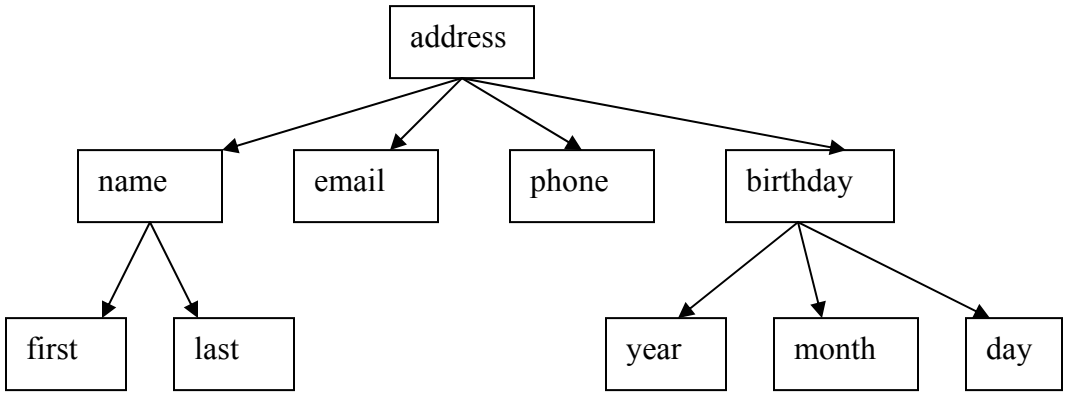
An XML document exhibits a tree structure. It has a single root node, <address> in the example above. The tree is a general ordered tree. There is a first child, a next sibling, etc. Nodes have parents and children. There are leaf nodes at the bottom of the tree. The declaration at the top is not part of the tree, but the rest of the document is.

We could expand the document above so that name and birthday have child nodes.

```
<?xml version = "1.0" ?>
<address>
    <name>
        <first>Alice</first>
        <last>Lee</last>
     </name>
    <email>alee@aol.com</email>
    <phone>123-45-6789</phone>
    <birthday>
        <year>1983</year>
        <month>07</month>
        <day>15</day>
    </birthday>
```

</address>

Now <name> has two children and <birthday> has three. Most processing on the tree is done with a preorder traversal.



Entities

As in html, certain characters are not allowed in the data. The most obvious ones are the less than signs and quotation marks. Also, ampersands are used to start the escape string, so they too have a substitution. These are escaped with the following substitutions with the greater than sign thrown in for symmetry.

<	<
>	>
&	&
“	"
‘	'

CDATA Sections

CDATA stands for character data. XML can have sections that contain characters of any kind that are not *parsed*. This means that they will be ignored by the XML parser that is used to put the document into a tree. These sections are similar to the *pre* sections in html. The browser simply displays them unchanged.

CDATA sections begin with <![CDATA[and end with]]>. An example might be an equation like the following:

```
<![CDATA[
    x + 2*y = 3
]]>
```

Attributes

As in html, tags can have attributes. These are name-value pairs such as width = "300". We have seen these in applet and image tags. They can be used in XML and are required in some places.

An example from the preceding might be

<name first = "Alice" last = "Lee" />

However this is not very useful for data. It makes it harder to see the structure of the document.

There are places where attributes are necessary. One that we will be using in the future is for giving a reference to the location of a stylesheet.

```
<link rel="stylesheet" type="text/css" href="address.css" />
```

Unicode

Like Java, XML uses Unicode to code for character data. There are a number of different versions of Unicode, but all have ASCII as the first 128 characters. After that they may differ. The most common version used in the west, and the XML default, is UTF-8. It is a variable length code that encodes some characters in a byte, some in two bytes and even some in four bytes.

Since many applications just use ASCII, this is the most efficient way to work with them and wastes the least space. The remaining 128 characters from code 128 to code 255 are used for some of the more common non-ascii characters used in western nations. The two-byte codes are used for some other language systems including some Asian ideographs. And finally the four-byte codes are used for more complicated ideographs.

There are a number of other flavors of Unicode. If you expect to be coding languages other than the common western ones, you should investigate all the possibilities. We will use UTF-8 for our documents.

Declarations

XML documents do not require a declaration at the top, but it is always a good idea to put one there. The simplest declaration is the one used above:

```
<?xml version = “1.0”?>
```

To this we can add two attributes. These are encoding and standalone. The result might be

```
<?xml version="1.0" encoding="UTF-8" standalone ="no"?>
```

The meaning of encoding was given above, and standalone is explained below.

There are many predefined declarations in use. The one that follows has been provided by the Apache Tomcat project to be used in configuring the *web application deployment descriptor*, web.xml. The encoding here is ISO-8859-1, known as Latin-1. ISO stands for International Standards Organization. And the 8859 standard contains a number of encodings. Latin-1 is the only one that is the same as UTF-8 in the first 256 characters.

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

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
```

The DOCTYPE declaration is similar to the one that we used in xhtml. But this one refers to a DTD (Document Type Definition) that has been created for Tomcat. It can be found as the only contents of the web.xml file, stored in the WEB-INF folder, in Tomcat. Instructions for configuring it for a web application are in the next document.

Document Type Definitions

A DTD is used to describe a particular XML application. It is used to specify the names, order, and data of elements in the application. It also shows the tree structure, identifying which elements are children,

siblings, or parents of other elements. A document that adheres to the specifications is said to be *valid.* A document may be well-formed but not valid. This can occur for example if the elements are out of order.

A DTD for the address example might be:

```
<!ELEMENT address (name, email, phone, birthday)>
<!ELEMENT name (first, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT birthday (year, month, day)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT day (#PCDATA)>
```

This says that address is the root of the document. It has four children: name, email, phone, and birthday. The element name also has two children, first and last. And the element birthday has three children, year, month, and day. All the rest of the elements consist of PCDATA, parsed character data. A DTD can only make a distinction between PCDATA and CDATA, unparsed character data. For finer distinctions, you have to use schema.

A DTD can be either in-line or external. The standalone attribute in the declaration above has the value "no", meaning that the DTD is external. The value "yes" means that it is in-line. No is the default.

If the above definitions are contained in a file called *address.dtd*, the following declaration should be added to the top of the xml file.

```
<!DOCTYPE  address SYSTEM "address.dtd">
```

This assumes that the file, address.dtd, is in the same folder as the xml file. This is the best way to handle finished DTDs, however when developing a DTD, it is more convenient to have it in-line. In that case, the entire DTD is placed at the top of the xml file enclosed by <!DOCTYPE address …]>. The entire in-line example for the preceding xml file follows. It has been validated by the DTD validator, found at http://www.w3schools.com.

```
<?xml version = "1.0" ?>
<!DOCTYPE address
[<!ELEMENT address (name, email, phone, birthday)>
<!ELEMENT name (first, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT birthday (year, month, day)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT day (#PCDATA)>
]>
<address>
      <name>
            <first>Alice</first>
            <last>Lee</last>
```

```
    </name>
    <email>alee@aol.com</email>
    <phone>123-45-6789</phone>
    <birthday>
        <year>1983</year>
        <month>07</month>
        <day>15</day>
    </birthday>
</address>
```

Reference

Elliotte Rusty Harold and Scott Means, *XML Programming*, chapters 1, 2, 3, and 5, O'Reilly & Associates, Inc., 2004.