Introduction to XML Schema

Extensible Markup Language (XML) is now widely used for interchanging documents and data. Much of this is sent via the Internet, but a large amount is also exchanged using private networks of some sort. There are now many XML subsets designed for particular areas, ranging from the rapid distribution of breaking news (using RSS, Rich Site Summary or Really Simple Syndication) to markup languages tailored specifically for internal company use.

Since XML does not have tags with fixed meanings, the senders and receivers must agree on the tags and know how to interpret them. For example, a list tag could refer either to an html form or a medical form. One way to handle this is to put each into separate *namespaces*. Along with this, both Document Type Definitions (DTDs) and XML Schema are used to certify that the document adheres to certain agreed upon standards.

Namespaces

Distinctions are made between tags with the same names by adding a *prefix* to the beginning of the tag. For example, we could have h:form for html forms and m:form for medical forms. The entire name is said to be the *qualified* name. It consists of the prefix and the *local* part. Since XML tag names may contain only a single colon, the local part must be colon free.

Namespaces are described by a Uniform Resource Identifier (URI). The identifier doesn't actually have to point to a real web page, but it is preferable that it do so. The page only needs to have some explanation about the uses for the prefix. The main one that we will use is

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

This is the namespace for XML schema. The prefix is "xmlns", which stands for xml namespace.

For the form example above, we could have namespaces

xmlns:m="http://csis.pace.edu/~wolf/medical/"

and

xmlns:h="http://www.w3.org/TR/html4/"

The latter is a real website, the W3C HTML 4.01 Specification. However there is no medical folder on my web site. If you try to link to it, you will get a **Not Found** page.

If you put a namespace attribute in a tag, all its children will inherit it. This way you do not have to add the prefix to every tag. This provides a *default* namespace for the tag and its children.

```
<form xmlns="http://csis.pace.edu/~wolf/medical/">
    <doctor>Dr. Stein</doctor>
    <patient>Alice Lee</patient>
</form>
```

Schema and DTDs

Schema and Document Type Definitions (DTDs) are used to make sure that those that use the documents agree on their contents and form. DTDs were developed first. They define the tree structure of a document, but they only provide two data types, CDATA and PCDATA. This was fine when XML was primarily used for marking up documents, such as books and articles. Most of that content consists of character data.

However, now XML is widely used to interchange data from files and databases. This data can have a number of other data types, including integers, decimals, dates and booleans. Also the W3C Recommendation for DTDs came before that for namespaces. A colon is allowed in an XML name, so a qualified name will be accepted by a DTD, but a DTD cannot parse the separate parts. Schema solve both of these problems. In addition, Schema are themselves XML documents. So a new format does not have to be learned.

The W3C Recommendation for Schema only dates to May 2001. However, they are now probably more widely used than DTDs. And some are suggesting that DTDs be retired in favor of Schema. Schema are more complicated than DTDs, but they also do more. We will look at some simple examples and then a few more complicated ones.

First Address Example

We considered the following XML document earlier.

```
<?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>
```

It might represent a row in a database. We had a DTD that described it. The following Schema does also. Note the first two lines of the schema. They are standard and must be copied exactly into the document.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="address">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="email" type="xs:string"/>
            <xs:element name="phone" type="xs:string"/>
            <xs:element name="birthday" type="xs:date"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

This looks more complicated than the DTD we used for it previously. But it also contains more information. It says that address is an element, that its type is complex, and that the elements called name, email, phone, and birthday must occur in the order shown. If <xs:sequence> had been left out, the four elements could appear in any order, but they would all have to be there. Also while three of the elements are strings, the fourth is a date. Date fields in XML are of the form yyyy-mm-dd. If they are not in this form, they are not valid.

Also since schema are XML documents themselves, they mirror the form of the documents they are describing. The one above shows that address is the root node and that name, email, phone, and birthday

are its children. This schema also says that each element must occur once and only once. The default is exactly once. This can be changed by adding a constraint to an element.

```
<xs:element name="phone" type="xs:string" maxOccurs="unbounded"/>
```

This says that there may be one or more phone numbers listed. There must be at least one, however. To change that, we would have to add another constraint, minOccurs="0".

An XML document is known as an *instance* of the schema. To use the schema, the document must contain a link to it. This is put into the root tag.

```
<address
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="address.xsd">
```

Notice that it not only identifies the W3C site for schema, but it also indicates that this is an instance of that schema. Since namespaces are not used inside this document, it says that the location of the schema is in xsi:noNamespaceSchemaLocation. If a namespace had been used, this would change to xsi:schemaLocation.

Schemas are not unique. Another one for address.xml uses a reference in one element to another element. Thus the *address* element has references to the *name, email, phone*, and *birthday* elements. This can be used to divide the schema into manageable parts. Note that comments follow the usual rules for html and xml.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Definition of simple elements. -->
<xs:element name="name" type="xs:string"/>
<xs:element name="email" type="xs:string"/>
<xs:element name="phone" type="xs:string"/>
<xs:element name="birthday" type="xs:date"/>

<!-- Definition of complex elements. -->
<xs:element name="address">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="name"/>
            <xs:element ref="email"/>
            <xs:element ref="phone"/>
            <xs:element ref="birthday"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Dividing the elements up this way will make it easier to handle more complicated xml documents.

Second Address Example

The next example divides the name element up into first, middle and last. However, the middle name is optional.

```
<?xml version = "1.0" ?>
<address
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="address.xsd">
      <name>
            <first>Alice</first>
            <middle>Ann</middle>
            <last>Lee</last>
      </name>
      <email>alee@aol.com</email>
      <phone>123-45-6789</phone>
      <birthday>1983-07-15</birthday>
</address>
```

One possible schema for this xml document follows. Notice that there are now two complex type sections. In developing a schema, it helps to have the sections separated. Otherwise it is difficult to see which tags go with which, and indentation becomes extreme.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Definition of simple elements. -->
<xs:element name="first" type="xs:string"/>
<xs:element name="middle" type="xs:string"/>
<xs:element name="last" type="xs:string"/>
<xs:element name="email" type="xs:string"/>
<xs:element name="phone" type="xs:string"/>
<xs:element name="birthday" type="xs:date"/>

<!-- Definition of complex elements. -->
<xs:element name="name">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="first"/>
                  <xs:element ref="middle" minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element ref="last"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:element name="address">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="name"/>
                  <xs:element ref="email"/>
                  <xs:element ref="phone"/>
                  <xs:element ref="birthday"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>
</xs:schema>
```

These two examples illustrated simple and complex types, string and date data types, sequences and occurrences, as well as how to link an xml document to a schema.

Validation

As before, a document may be well-formed but not valid. Validity means that it conforms to either a DTD or a schema. Parsers are used to check the validity of instance documents. One is available from http://xml.apache.org/. It is open source and is called Xerces. A sample parser comes with it called Writer.java. You can use it to parse and validate an xml document either against a DTD or a schema.[1]

If the document is valid, Writer will simply echo it on the console screen. However, if there is an error, it will first point it out and then echo the document. As with all such software, some of the error messages are easier to understand than others. When you are developing a schema for an xml document, it is wise to check it regularly for validity. Schema are complicated, so errors are common.

Roster Example

Another example involves a roster for a class. It contains elements with attributes. An element with an attribute has a complex type and is listed differently from simple elements. The xml file follows.

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

<roster
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="roster.xsd">
      <student>
            <name>AliceLee</name>
            <midterm weightMidterm = "50">85</midterm>
            <final weightFinal = "50">92</final>
      </student>
      <student>
            <name>Barbara Smith</name>
            <midterm weightMidterm = "50">78</midterm>
            <final weightFinal = "50">84</final>
      </student>
      <student>
            <name>Cathy Jones</name>
            <midterm weightMidterm = "50">82</midterm>
            <final weightFinal = "50">87</final>
      </student>
</roster>
```

---

[1] A version of Writer.java is on my web site. In order to compile it in JCreator, you will need some .jar files. These are also on the website. Get all five of them, resolver.jar, xercesImpl.jar, xercesSamples.jar, xml-apis.jar, and xmlParserAPTs.jar. You can store them with the other .jar files in your jdk folder. Configure JCreator to find them by adding them to the profile for the Java editor. When you execute the file it will first ask you for an option and the name of the xml file. To validate using a DTD, the option is –v, and to validate using a schema, the option is –s.

This is more complicated, so the schema for it is also. The midterm and final cannot be considered simple types because they have attributes. They have to be handled as complex types. First, they are *extensions* of a simple base type, xs:positiveInteger. The attributes themselves are simple types, so they contribute simple content to the element. This part of the schema looks as follows:

```
<xs:element name="midterm">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:positiveInteger">
                <xs:attribute ref="weightMidterm"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

The following is a schema for roster.xml. It has

1. one simple element: name,
2. two attributes: weightMidterm and weightFinal,
3. a root: roster,
4. a child of the root: student, and
5. children of student: name, midterm, and final.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="name" type="xs:string"/>

<!-- definition of attributes -->
<xs:attribute name="weightMidterm" type="xs:positiveInteger"/>
<xs:attribute name="weightFinal" type="xs:positiveInteger"/>

<!-- definition of complex elements -->

<xs:element name="midterm">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:positiveInteger">
                <xs:attribute ref="weightMidterm"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="final">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:positiveInteger">
                <xs:attribute ref="weightFinal"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
```

```
</xs:element>

<xs:element name="student">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="name"/>
            <xs:element ref="midterm"/>
            <xs:element ref="final"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="roster">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="student" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

There is a lot more to know about schema besides the above. For more information, see the References.

References

1. Elliotte Rusty Harold, *Processing XML with Java*, chapter 1, Addison Wesley, 2002.
2. Elliotte Rusty Harold and Scott Means, *XML Programming*, chapter 16, O'Reilly & Associates, Inc., 2002.
3. W3Schools Online Web Tutorials, http://www.w3schools.com.