XML Schemas express shared vocabularies and allow computers to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents.

A schema defines what one or more XML documents can look like — what elements they contain and in what order, what their content might be, and what attributes they might contain. This sounds like a DTD. In a general sense, a DTD is a schema, but it has several disadvantages:

- DTDs are written in a syntax that have little to do with XML. Thus, the DTD cannot be interpreted with an XML parser.
- DTD declarations are global, meaning you can't define two different elements with the same name even if they appear in separate contexts.
- DTDs cannot control what kind of information a given element or attribute can contain.

The XML Schema language is an attempt to fix these weaknesses. XML Schema, written in XML itself, lets you define both global elements that must be used in the same way throughout the document, and local elements that have specific meanings in particular contexts. In other words, like the DTD, the Schema is a strict set of rules that allows the XML to conduct its business.

XML Schema contains a system of data types that let you specify that a certain element should contain a string, an integer or a date. XML Schema gives you better control over the contents of an XML document than that offered by a DTD.

A schema can divide a document into two types of content — simple and complex. Elements that contain only strings (text) are of the simple type. Elements that contain other elements or that contain attributes are of the complex type.

Building an XML Schema can be tricky, but the effort is worthwhile. With Schema language, you can build XML files that have the power of a database without having database application to develop or run it. Let’s take a look at how it is done.

Starting a Schema file

You first must begin with a well-formed XML file. Remember that your browser will parse a well-formed file, but you need something more for the data in the file to have any meaning. You must define the data for it to be valid, much as you have done in the DTD. Using the Schema technique, you write a separate file following the XML syntax. However, Schema language has a specific vocabulary.

A Schema file is an XML document and must follow the rules of XML carefully. Here are the first things you must do to write a Schema:

- Write the standard XML declaration and save the file with the extension of .xsd.
- The root element in an XSD is <schema>. Type that after the XML declaration.

Copyright © 2010 by Paul Roberts • All rights reserved

- Next you must create a namespace using the xmlns attribute, which provides the prefix that will be added to the Schema element type names in the XSD instance. The current definition can be found at http://www.w3.org/2001/XMLSchema. This is how it will look:

  ```
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ```

- You must also include a namespace declaration in the XML document that will be validated by your schema. This namespace is always placed on the root element of the XML document with an xmlns attribute, as shown in the following example:

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

  This namespace holds the XML Schema elements and properties that might be included in the XML document instance. The prefix xsi is used as a proxy to this namespace and is also affixed to the beginning of all elements and attributes belonging to the namespace, separated by a colon.

- In case the XSD file is not linked with a namespace, use the noNamespaceSchemaLocation attribute with the value referring to the file as follows:

  ```
  xsi:noNamespaceSchemaLocation="customers.xsd"
  ```

  A namespace can be declared along with a filename, in which case the URI (Uniform Resource Identifier) for the namespace and the URI of the schema (file name) will be separated by a white space as part of the same attribute value. Here is an example:

  ```
  xsi:schemaLocation="http://example.org/folder/ filename.xsd"
  ```

Simple Elements

A simple element is one that can contain only text. It cannot contain any other elements or attributes. However, the "only text" restriction is misleading. The text can be of many different types. It can be one of the types that are included in the XML Schema definition (integer, string, date, etc.), or it can be a custom type that you define yourself. You can also add restrictions — known as facets) — to a data type in order to limit its content. Further, and you can require the data to match a defined pattern.

Here is the syntax for defining simple elements:

```
<xs:element name="xxx" type="yyy"/>
```

The `name` attribute refers to the name of the element in the XML file. The `type` element refers to the data type of the element. Following is an example, beginning with some data from an XML document named *"customers.xml"*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<customerList>
<customer>
<firstName>Eric</firstName>
<lastName>Olsen</lastName>
<address>23 Pinecrest Lane</address>
<city>Houston</city>
<state>TX</state>
```

```
</customer>
</customerList>
```

Now here is the Scheme definition, contained in a file named "customers.xsd". Note the nesting of elements within other elements:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="customerList">
<xs:complexType>
<xs:sequence>
<xs:element name="customer" minOccurs="1">
<xs:complexType>
<xs:sequence>
<xs:element name="firstName" type="xs:string" />
<xs:element name="lastName" type="xs:string" />
<xs:element name="address" type="xs:string" />
<xs:element name="city" type="xs:string" />
<xs:element name="state" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

The root element is always xs:schema. The namespace is declared with the attribute xmlns:xs. The xs is a proxy that acts in the place of the URI that follows it. The URI points to the place where the parser will go to make sense out of the code in the file.

Now let's add the necessary links to the XML file, which will make the file valid.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<customerList
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="customers.xsd">
<customer>
<firstName>Eric</firstName>
<lastName>Olsen</lastName>
<address>23 Pinecrest Lane</address>
<city>Houston</city>
<state>TX</state>
</customer>
</customerList>
```

The bold type indicates the addition of two properties to the root element. The first attribute—xmlns:xsi — points to the resource that validates this file as an instance using XML Schema. The xsi component is a proxy for the URL. The next attribute—xsi —points to the location of the XSD file itself. The above example uses on within the same root folder as the XML document. If the XSD file is located at a URL, the following syntax would be used instead. Note the space before the file name:

That's a space.

```
xsi:SchemaLocation="http://www.absentmind.net/XML/ customers.xsd"
```

Attributes

Simple elements do not have attributes. Any element with attributes is complex type. The attribute itself is always declared as a simple type. This means that an element with attributes always has a complex type definition. Here is how to declare attributes, with xxx being the name of the attribute and yyy being the data type:

```
<xs:attribute name="xxx" type="yyy"/>
```

Here is an element from our XML file, with an attribute added, followed by its corresponding attribute definition in the Schema file:

```
<firstName language="EN">Eric</firstName>
```

```
<xs:attribute name="language" type="xs:string" />
```

Here are a few of the most commonly used Schema data types. More on how to use each of these attributes will come later. For now it is enough to be aware of some of the different types of data:

- *xs:string* — The string data type can contain characters, line feeds, carriage returns, and tab characters.
- *xs:decimal* — The decimal data type is used to specify a numeric value. The maximum number of decimal digits you can specify is 18.
- *xs:integer* — The integer data type is used to specify a numeric value without a fractional component.
- *xs:boolean* — The Boolean data type is used to specify a true or false value.
- *xs:date* — The date data type is used to specify a date.
- *xs:time* — The time data type is used to specify a time.

Attributes can have a specified default value or fixed value. A default value is automatically assigned to the attribute when no other value is specified. A fixed value is also automatically assigned to the attribute, but you cannot specify another value. Here are a couple of examples:

```
<xs:attribute name="language" type="xs:string" default="EN" />
```

```
<xs:attribute name="language" type="xs:string" fixed="EN"/>
```

You can also declare required and optional attributes. If not specified, the attribute will be optional. This is what these specifications look like.

```
<xs:attribute name="language" type="xs:string" use="required" />
```

```
<xs:attribute name="language" type="xs:string" use="optional" />
```

Defining a type for an XML element or attribute sets a restriction for the element's or attribute's content. If an XML element is of type "xs:date" and contains a string like "Hello Mother", the element will not validate. With XML Schemas, you can add your own restrictions to your XML elements and attributes. These restrictions are called facets.