Week 7 Lecture : Schemas - Part 1

Screencast of Recorded Lecture (2022):

- **https://sjeccd-edu.zoom.us/rec/play/AaVej6RGEoflvh7-nl0tsi5CwU_U2HtK5BOG1Jie9w2mb53tC0W0YyUMExto-E1GKNcPgjFy9YdGxH2k.UO2AApC9rr9ybONO** (https://sjeccd-edu.zoom.us/rec/play/AaVej6RGEoflvh7-nl0tsi5CwU_U2HtK5BOG1Jie9w2mb53tC0W0YyUMExto-E1GKNcPgjFy9YdGxH2k.UO2AApC9rr9ybONO)

Screencast of Recorded Lecture (2021):

- **https://sjeccd-edu.zoom.us/rec/play/mkZH-YNvdcry_bKyvvHPNqoxPo8bL27QoK37nlWLs4tfjtNlG9uJubBNPOXk_4bhZS4w6okOrMFDFgAy.wyQsTYmMqKhTMBaP** (https://sjeccd-edu.zoom.us/rec/play/mkZH-YNvdcry_bKyvvHPNqoxPo8bL27QoK37nlWLs4tfjtNlG9uJubBNPOXk_4bhZS4w6okOrMFDFgAy.wyQsTYmMqKhTMBaP)

XML Schemas

This week and next week I want to go over the concept of **XML Schemas**.

Both Schemas and DTDs let us define the "rulebook" that the XML file will abide by.

DTDs are written in a language totally different from XML.
Schemas, on the other hand, are written in XML format.

Let's take a look at what's wrong with DTDs.

Document Type Definitions (DTDs) are an outgrowth of XML's heritage in Standardized General Markup Language (SGML). SGML was always intended for narrative style documents: books, reports, technical manuals, brochures, etc. DTDs were designed to serve the needs of these sorts of documents, and they serve them well. DTDs let you state very simply that every book must have an author, that every song must have a title, that every PERSON must have an ID attribute, etc. For narrative documents that are intended for people to read from start to finish, that are composed of words in a row, there's no need for anything beyond a DTD. However, XML has gone well beyond this. It is being used for object serialization, stock trading, remote procedure calls, graphic file formats, and many more things that look nothing like narrative documents, and in these new areas DTDs have many limitations.

A major limitation of DTDs is the lack of data typing, especially for element content. When thinking about types of data in most computer languages, we have things like integer numbers (like 0,1,2,3, -4, -5), floating point numbers (decimal numbers -- like 4.2, 1.0, -42.7), strings (characters, like "hello"), and booleans (True/False).

DTDs can't say that a PRICE element must contain a number, much less a number that's greater than zero with two decimal digits of precision and a dollar sign. There's no way to say that a MONTH element must be an integer between 1 and 12. This is a major limitation to using DTDs to verify and enforce the data integrity of an XML file.

Another problem is that DTDs have an unusual non-XML syntax. You actually need separate parsers and APIs to handle DTDs than you do to handle XML documents. For example, consider this common element declaration:

```
<!ELEMENT TITLE (#PCDATA)>
```

The above is not a legal XML element. Using XML rules, you can't begin an element name with an exclamation point. **TITLE** is not an attribute, and neither is **(#PCDATA)**. This is a very different way of describing information than is used in XML documents.

Some other limitations of DTDs are:

- DTDs are only marginally extensible. It's very difficult to combine separate DTDs in a sensible way.
- DTDs are only marginally compatible with namespaces.
- DTDs can't enforce the order or number of child elements in mixed content.

Schemas, then, are an attempt to solve all these problems be defining an XML-based syntax for describing the permissible contents of XML documents that includes:

- Powerful data typing including range checking (can throw an error if you use a number when you're supposed to be using text, for example)
- Namespace-aware validation based on namespace URIs rather than on just prefixes
- Extensibility and scalability

XML Schema Syntax

Let's look briefly at some terminology. A W3C XML Schema document defines the allowed content for a class of XML documents. A single document of that class is called an **instance** document.

Elements and attributes are said to be 'declared' in a W3C XML Schema document.

The content of elements and attributes has a 'type'.

There are 2 kinds of 'type':

- **simpleType** and
- **complexType**

Types can be built-in (defined in the W3C Schema specification itself) or can be defined by a schema developer. Elements and attributes have 'declarations'. Simple type and complex types have 'definitions'.

Creating Schemas

xs: vs. xsd:

When defining elements using a Schema, you need to start the element with the prefix **xs** or **xsd**. Either one is fine; generally, Microsoft generally prefers *xsd* -- others prefer *xs*. Although either one can be used, you must be consistent within the Schema (if you start the tag **<xs:schema>**, you can't close it with **</xsd:schema>** and vice-versa).

<xs:schema> *or* <xsd:schema>

The schema element is the root element for every Schema.

Simple Elements vs. Complex Elements

A simple element is one that only contains text. It cannot contain any other sub-elements or attributes.

If the element has a sub-element or attribute, it's a complex element.

Here's an example showing the difference between a simple and complex elements:

```
<name>
  <firstname>Mark</firstname>
  <lastname>Branom</lastname>
</name>
```

Here, both **firstname** and **lastname** are simple elements. They only contain text. But the **name** element is a complex element, because its content has the sub-elements **firstname** and **lastname**.

Defining Simple Elements

```
<xs:element name="ElementName" type="ElementDataType" />
```

You can declare that the element must use a particular data type. Some of the most commonly used data types are:

- **xs:string** – string of characters and numbers
- **xs:decimal** – string of only decimal numbers (but no spaces or other characters)
- **xs:integer** – string of any type of number (but no spaces or other characters)
- **xs:boolean** – true/false
- **xs:date** – date in the YYYY-MM-DD format
- **xs:time** – time in the HH:MM:SS format

For more on data types, visit the W3C's page on datatypes:

https://www.w3.org/TR/xmlschema-2/#datatype (https://www.w3.org/TR/xmlschema-2/#datatype)

In the name example from above, the elements **firstname** and **lastname** could be defined thusly:

```
<xs:element name="firstname" type="xs:string" />
<xs:element name="lastname" type="xs:string" />
```

Declaring Elements

In W3C XML Schema terminology, elements and attributes are 'declared'. Both elements and attributes can occur in an instance document.

Given this simple example of an XML instance document (simpledocument.xml):

```
<?xml version="1.0"?>
<simpledocument>
   Some text here
</simpledocument>
```

This might be a W3C XML Schema document against which the **simpledocument.xml** can be validated (**simpledocument.xsd**):

```
<?xml  version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="simpledocument" type="xsd:string" />
</xsd:schema>
```

The indicative namespace prefix in the W3C XML Schema document is *xsd*. The namespace declaration in the start tag of the *xsd:schema* element associates the *xsd* namespace with the URI *http://www.w3.org/2001/XMLSchema*.

An *xsd:element* element is used to declare the *simpledocument* element that is found in the instance document. The content of the *simpledocument* element is text content only and is declared, using the type attribute of the *xsd:element* element, to be of type *xsd:string*. The *xsd:string* type is one of many built-in types in the W3C XML Schema Recommendations.

As I said before, you could either use the prefix **xsd** or **xs**, but you must be consistent throughout.

Valid Schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="simpledocument" type="xsd:string" />
</xsd:schema>
```

Valid Schema:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="simpledocument" type="xs:string" />
</xs:schema>
```

Invalid Schema (you can't mix and match 'xsd' and 'xs'):

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xs:element name="simpledocument" type="xs:string" />
</xsd:schema>
```

Invalid Schema (the schema is using 'xs' for everything except for the type):

```
<?xml version="1.0"?>
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xs:element name="simpledocument" type="xsd:string" />
</xs:schema>
```

Now let's look at a more complex XML file, and a more complex schema.

Suppose I have that same *starwars.xml* file I worked on when discussing DTDs.

```
<?xml version="1.0" encoding="utf-8"?>
<starwars>
  <movie ranking="2">
    <title>Star Wars: A New Hope</title>
    <episode>4</episode>
    <year>1977</year>
    <director>
       <firstname>George</firstname>
       <lastname>Lucas</lastname>
    </director>
  </movie>
  <movie ranking="1">
     <title>Empire Strikes Back</title>
     <episode>5</episode>
     <year>1980</year>
     <director>
        <firstname>Irvin</firstname>
        <lastname>Kershner</lastname>
     </director>
  </movie>
  <movie ranking="3">
     <title>Return of the Jedi</title>
     <episode>6</episode>
     <year>1983</year>
     <director>
       <firstname>Richard</firstname>
       <lastname>Marquand</lastname>
     </director>
  </movie>
</starwars>
```

If I wanted to create a schema rulebook for this XML file, here's how I might go about creating it:

As with the DTD, I would map out what I want to have in my XML. In this case, we have the following elements:

- starwars
    - movie (with an attribute "ranking")
        - title
        - episode

        - year
        - director
            - firstname
            - lastname

So the elements that need to be declared are **starwars**, **movie**, **title**, **episode**, **year**, **director**, **firstname**, and **lastname**. There is one attribute: **ranking**, and it's modifying the **movie** element.

Again, the order in which they are declared is important. Also, since there are multiple instances of the movie element, we need to indicate that as well.

Let's go through a potential Schema.

First, I'd declare that I'm creating a schema:

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

Then, I'd declare the root element **starwars**, and state that it is a **complexType**. I use **complexType** instead of simpleType because it is not an empty element (there are sub-elements of *starwars*):

```
 <xs:element name="starwars">
   <xs:complexType>
```

Next, because I want the data to be listed in a particular order of sequence, I need to declare that I'm using a **sequence**.

```
   <xs:sequence>
```

Then, I'd want to declare the **movie** element. Note that I'm declaring that there must be at least one movie element (minOccurs="1") but there can be as many movie elements as the XML creator desires (maxOccurs="unbounded"). I again use **complexType** (since there are sub-elements of *movie*).

```
      <xs:element name="movie" maxOccurs="unbounded" minOccurs="1">
          <xs:complexType>
```

Next, because I want to declare the order in which the elements appear, I need to declare that there is a **sequence**. If I didn't care about the order, I could choose **all**. If some of the sub-elements were optional, I could use **choice**. But because I want them to be in the order of title, episode, year, and director, I need to choose **xs:sequence**.

The **title**, **episode**, and **year** elements are simpleType elements, so they just need to declared as a simpleType element. I've chosen to use the **string** type for each, although I could have used a numeric type for the episode and year (**int**, for example). The **director** element is a **complexType** because it has children (**firstname** and **lastname**).

Note that although the **movie** element has an attribute (*ranking*), I have not yet declared it -- the declaration of an attribute comes just before the closing tag for its element.

```
          <xs:sequence>
               <xs:element type="xs:string" name="title" />
               <xs:element type="xs:string" name="episode" />
               <xs:element type="xs:string" name="year" />
               <xs:element name="director">
                 <xs:complexType>
```

Next, because I want to declare the order in which the elements appear, I need to declare that there is a sequence. The **firstname** and **lastname** elements are simpleType elements, so they just need to declared as an element. I've chosen to use the **string** type for each.

```
                  <xs:sequence>
                     <xs:element type="xs:string" name="firstname" />
                     <xs:element type="xs:string" name="lastname" />
                  </xs:sequence>
```

Next, I want to close the **complexType** and **element** tags related to the **director** element; and the the **sequence** tag related to the **movie** element:

```
           </xs:complexType>
         </xs:element>
       </xs:sequence>
```

Next, I want to add the attribute *ranking* and declare that it is a **required** attribute using the **attribute** element (simpleType). I'm using the *string* type, although I could have used *int*.

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

Finally, I want to close the **complexType** tag related to the **movie** element; the **element** tag related to the **starwars** element; and the **schema** element:

```
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Here's an entire valid XSD for this XML:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="starwars">
   <xs:complexType>
     <xs:sequence>
       <xs:element name="movie" maxOccurs="unbounded" minOccurs="1">
         <xs:complexType>
           <xs:sequence>
             <xs:element type="xs:string" name="title" />
             <xs:element type="xs:string" name="episode" />
             <xs:element type="xs:string" name="year" />
             <xs:element name="director">
               <xs:complexType>
                 <xs:sequence>
                    <xs:element type="xs:string" name="firstname" />
                    <xs:element type="xs:string" name="lastname" />
                 </xs:sequence>
               </xs:complexType>
             </xs:element>
           </xs:sequence>
         <xs:attribute type="xs:string" name="ranking" use="required" />
        </xs:complexType>
      </xs:element>
     </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:schema>
```

More about Complex and Simple Types

As I said earlier, in the W3C XML Schema there are two basic types of element content:

- **simple types** (designated as **simpleType**) and
- **complex types** (designated as **complexType**)

Simple Types

- contains **only** text content

- has **no child elements** (sub-elements)
- has **no attributes**
- can be empty

Given a document with the following structure:

```
<memo>
<from>John Smith</from>
<email>JSmith@anyplace.com</email>
<to>Peter Jones</to>
<email>PJones@anyplace.com</email>
<message>Remember to use proper structure</message>
</memo>
```

The *memo* element would be a **complex** element (since it contains sub-elements).

But *from*, *email*, *to*, and *message* would be considered **simple** elements.

Most of these simple elements have simple string content of type **xsd:string**. You can define a simple type, such as for the **email** element, as shown above, if the allowed content is a built-in W3C XML Schema datatype, as follows:

```
 <xsd:element   name="email"   type="xsd:string"  />
```

If you want to constrain allowed values for element content, you can use an anonymous simple type definition (that is, using no name attribute on the **xsd:complexType** element), coupled with a *restriction*:

```
<xsd:element name="from">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="John Smith" />
      <xsd:enumeration value="Jane Smith" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The allowed values for the above element content are the strings "John Smith" and "Jane Smith". This would ensure that only authorized senders of email could be identified in the **from** element. If the XML file had `<from>J Smith</from>`, the XML would be invalid; only if it had `<from>John Smith</from>` or `<from>Jane Smith</from>` would the XML be valid.

Complex Types

- elements that contain either:
  - **sub-elements** or
  - **attributes**

If an element has either sub-element content, attribute(s), or both, it is of complex type.

A named complex type definition is referenced using the type attribute of an **xsd:element** element.

An anonymous complex type definition is nested inside an **xsd:element**.

For example, suppose we have this set of XML code:

```
<parent>
   <firstChild>content</firstChild>
   <secondChild>more content</secondChild>
</parent>
```

parent is a *complexType* element (because it contains the elements **firstChild** and **secondChild**).

Both **firstChild** and **secondChild** elements are *simpleType* elements.

Here is how the complete XSD might be constructed for this example:

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="firstChild" type="xsd:string"  />
    <xsd:element name="secondChild" type="xsd:string"  />
  </xsd:sequence>
</xsd:complexType>
```

Alternatively, if the allowed structures in the instance document were:

```
<parent>
  <firstChoice>some content</firstChoice>
</parent>
```

OR

```
<parent>
  <secondChoice>some content</secondChoice>
</parent>
```

this can be expressed in a schema using the **xsd:choice** element, as follows:

```
<xsd:complexType>
  <xsd:choice>
    <xsd:element name="firstChild" type="xsd:string" />
    <xsd:element name="secondChild type="xsd:string" />
  </xsd:choice>
</xsd:complexType>
```

As with DTDs, I'm going to post next week's lecture today (for those who just can't wait to do more with Schemas); in this next lecture, we will delve more in-depth in the schema topic.