More XML Schemas, XSLT, and AJAX

CS174

Chris Pollett

Oct 29, 2008.

Outline

- XML Schemas
- XSLT
- AJAX

Overview of data types

- There are two categories of data types in XML Schemas:
  - simple types -- which are restricted to strings and cannot have attributes or nested elements
  - complex types -- which can have attributes and can include other data types as elements.
- There are 44 different types in XML schemas, 19 of which are primitive and the remainder are derived.
- Example primitive types include: string, Boolean, time and anyURI.
- Example predefined derived types include: byte, long, decimal, unsignedInt, positiveInteger, and NMTOKEN.
- User defined types are defined by specifying restrictions on an existing type (called a base type).
- For example, for integer there are 8 so-called "facets" on which restrictions to base types can be made: totalDigits, maxInclusive, maxExclusive minInclusive, minExclusive, pattern, enumeration, and whitespace.
- Types can be named or anonymous. Anonymous elements cannot be used outside of the element in which it was declared
- Elements in DTDs are all global. For schemas one can have local elements. These are elements which defined within the scope of some child of the schema tag. child elements themselves are global.

Simple Types

- The simplest way to define a simple a new tag would be with a command like:

  <xsd:element name="engine" type="xsd:string" />

  <!-- notice namespace applied to the word string. In general namespaces can be applied to lots of things. For example, they also can be applied to attributes. -->

- In an instance of the plane schema we could then have:

  <engine>an example of the content of an engine</engine>.

- You can also give default values or force fixed values with slight variations of this declaration:

  <xsd:element name="engine" type="xsd:string" default="V-6" />

  <xsd:element name="plane" type="xsd:string" fixed="single wing" />

- A simple user-derived type can be defined using the <restriction> tag:

  <xsd:simpleType name="firstName">
  <xsd:restriction base="xsd:string">
  <xsd:maxLength value="10" />
  </xsd:restriction>
  </ xsd:simpleType>

Complex Types

- There are several kinds of complex types that can be used with XML schemas.
- We will only look at the complex types which are restricted to having subelement-only -- not both subelements and text. the complexContent tag can be used to handle other kinds of content.
- To define what subelement occur for an element we can use either sequence or all. sequence -- forces an order on the subelement, all doesn't.

```
<xsd:complexType name="car">
    <xsd:sequence>
      <xsd:element name="make"  type="xsd:string" />
      <xsd:element name="year" type="xsd:decimal" />
    </xsd:sequence><!-- you can use minOccurs, maxOccurs to specify number of occurrences ; you
    can have more than one sequence, all tag here-->
</ xsd:complexType>
```

References

- You can define element to be used as a subelement outside of another element and use a reference to refer to it:

```
<xsd:element name="year">
<xsd:simpleType >
    <xsd:restriction base="xsd:decimal">
        <xsd:minInclusive value="1900" />
        <xsd:maxInclusive value="2010" />
    </xsd:restriction>
</ xsd:simpleType>
</xsd:element>
<xsd:complexType name="car">
        <xsd:sequence>
         <xsd:element name="make"  type="xsd:string" />
          <xsd:element ref="year" />
       </xsd:sequence>
</xsd:complexType>
<xsd:element name="sport_car" type="car">
                              <xsd:attribute name="color" type="xsd:string" />
</xsd:element>
```

XML and CSS

- Most modern browsers are completely happy to style any tag provided there is some style-sheet information given for it:

plane {display:block; border 3px;}

- To associate a stylesheet with an XML document we use the syntax:

<?xml-stylesheet type="text/css" href="mystyles.css" ?>

XSLT

- Sometimes it is useful to transform one XML markup language into some other XML language.
- For instance, suppose you want to display a pure RSS feed nicely and you want the links to work in IE. Since IE does not support the XLink language this is hard unless you do a stylesheet tranformation.
- The basic idea of XSTL (eXtensible stylesheet transformations), is that we can associate a stylesheeet transformation with an XML document and apply this transformation using a processor (for instance, a browser), to get some other XML language.

An Example

```
<?xml version="1.0"
    encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl"
    href="xslplane.xsl" ?>
<plane>
   <year>1970</year>
</plane>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
   xmlns:xsl=
      "http://www.w3.org/1999/XSL/Transf
      orm"
   xmlns =
      "http://www.w3.org/1999/xhtml"
>
<xsl:template match="plane">
<html><head><title>result of applying a
      stylesheet to
      plane</title></head><body><h1>Pla
      ne Description</h1>
    <xsl:apply-templates />
    </body></html>
 </xsl:template>
<xsl:template match="year">
   <p style="color:red">
   <xsl:value-of select="." />
   </p>
</xsl:template>
</xsl:stylesheet>
```

Beginning AJAX

- AJAX stands for Asynchronous Javascript and XML.
- Asynchronous means that one does not need to wait for the response to an HTTP request to do something
  - one can use Javascript to have several outstanding HTTP requests and still make things happen in the browser.
  - Further one doesn't have to reload the entire page when the results of a request are returns.
- The XML in AJAX is because the results of a request are usual XML, JSON, or YAML data which is then formatted by some Javascript code.
- The basic key to AJAX is the ability for Javascript to make an HTTP request. This is done with the XMLHttpRequest object. As an example, look at the Javascript in the file:

http://www.cs.sjsu.edu/faculty/pollett/test/dept3.html

Step by Step

- To create an XMLHttpRequest one could simply write in Javascript:

  request = new XMLHttpRequest().
- This works on modern browsers. For older browsers like IE6 you need to do something like:

  request = new ActiveXObject('MSXML2.XMLHTTP').
- For older still use:

  request = new ActiveXObject('Microsoft.XMLHTTP')
- See the example page for how checking for browsers can be done using a try-catch block.
- At this point if we need to set up HTTP request headers one can use:

  request.setRequestHeader("name", "value").

Step by Step Continued I

- To open a connection back to the server the Javascript came from one can then do:

  request.open(*theHTTPmethod*, *theURL*, *theAsync flag*).

  or

  request.open(*theHTTPmethod*, *theURL*, *theAsync flag*, *username*, *password*)
- For example,

  request.open("GET", "progress.php", true)
- The asynchronous flag says whether or not the Javascript can continue executing (true) or if it must wait for a response (false).
- At this point no data has been sent yet.

Step by Step Continued II

- We next need to set up a callback function which will be called as we get data back from the server:

```
var self = this; /* remember scope of enclosing
                    object */
request.onreadystatechange = function()
{
    switch(request.readyState)
   {
      case 0:// handle uninitialized case.
      case 1: // handle open but no send case.
      case 2: // handle send but no response case .
      case 3: // handle response is being downloaded case.
      case 4: // handle response has completed being downloaded case.
   }
}
```

Step by Step Continued III

- In case 3 or 4 above, request.responseText or request.responseXML will contain the data that has been returned by the server.
- In case 3 or 4 above, request.getAllResponseHeaders() can be used to get the HTTP response headers.
- Sometimes it is useful to start a timer in case 3, using setTimout. If the request is taking to long so that the timer gets called, one can then abort the request using: request.abort.
- Typically, the code in 3 or 4 above would put the response text into some tag in our document. This could be done with:

```
document.getElementById("myDivTag").innerHTML = \ request.responseText; /*not really a
    DOM standard but all browsers happy with */
//or we could do…
myDiv = document.getElementById("myDivTag")
if(myDiv.firstChild)
{
    myDiv.removeChild(myDiv.firstChild);
}
myDiv.appendChild(document.createTextNode(request.responseText));
```

Step by Step Continued IV

- Once we have set up our callback function we are ready to send data to the server.
- To do this we can do:

```
request.send(null);
/* note: send's argument can be used if using
   POST method to send the posted data */
```

- That's it! We just sit back then and wait for our callback to be called.