11a

XSD and XSLT

XSD and Schema Validation

The content of a particular XML document is nearly always domain-specific, such as a Microsoft Word document, an application configuration document, or a web service. For each domain, the XML file conforms to a particular pattern. There are several standards for describing the schema of such a pattern, to standardize and automate the interpretation and validation of XML documents. The most widely accepted standard is *XSD*, short for *XML Schema Definition*. Its precursors, DTD and XDR, are also supported by `System.Xml`.

Consider the following XML document:

```
<?xml version="1.0"?>
<customers>
  <customer id="1" status="active">
    <firstname>Jim</firstname>
    <lastname>Bo</lastname>
  </customer>
  <customer id="1" status="archived">
    <firstname>Thomas</firstname>
    <lastname>Jefferson</lastname>
  </customer>
</customers>
```

We can write an XSD for this document as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
           elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="customers">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="customer">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="firstname" type="xs:string" />
              <xs:element name="lastname" type="xs:string" />
            </xs:sequence>
            <xs:attribute name="id" type="xs:int" use="required" />
            <xs:attribute name="status" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

As you can see, XSD documents are themselves written in XML. Furthermore, an XSD document is describable with XSD—you can find that definition at *http://www.w3.org/2001/xmlschema.xsd*.

Performing Schema Validation

You can validate an XML file or document against one or more schemas before reading or processing it. There are a number of reasons to do so:

- You can get away with less error checking and exception handling.
- Schema validation picks up errors you might otherwise overlook.
- Error messages are detailed and informative.

To perform validation, plug a schema into an `XmlReader`, an `XmlDocument`, or an X-DOM object, and then read or load the XML as you would normally. Schema validation happens automatically as content is read, so the input stream is not read twice.

Validating with an XmlReader

Here's how to plug a schema from the file *customers.xsd* into an `XmlReader`:

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.ValidationType = ValidationType.Schema;
settings.Schemas.Add (null, "customers.xsd");

using XmlReader r = XmlReader.Create ("customers.xml", settings);
...
```

If the schema is inline, set the following flag instead of adding to `Schemas`:

```
settings.ValidationFlags |= XmlSchemaValidationFlags.ProcessInlineSchema;
```

You then `Read` as you would normally. If schema validation fails at any point, an `XmlSchemaValidationException` is thrown.

Calling `Read` on its own validates both elements and attributes: you don't need to navigate to each individual attribute for it to be validated.

If you want *only* to validate the document, you can do this:

```
using (XmlReader r = XmlReader.Create ("customers.xml", settings))
  try { while (r.Read()) ; }
  catch (XmlSchemaValidationException ex)
  {
    ...
  }
```

`XmlSchemaValidationException` has properties for the error `Message`, `LineNumber`, and `LinePosition`. In this case, it only tells you about the first error in the document. If you want to report on all errors in the document, you instead must handle the `ValidationEventHandler` event:

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.ValidationType = ValidationType.Schema;
settings.Schemas.Add (null, "customers.xsd");
settings.ValidationEventHandler += ValidationHandler;
using (XmlReader r = XmlReader.Create ("customers.xml", settings))
  while (r.Read()) ;
```

When you handle this event, schema errors no longer throw exceptions. Instead, they fire your event handler:

```
static void ValidationHandler (object sender, ValidationEventArgs e)
{
  Console.WriteLine ("Error: " + e.Exception.Message);
}
```

The Exception property of ValidationEventArgs contains the XmlSchemaValidationException that would have otherwise been thrown.

> The System.Xml namespace also contains a class called XmlValidatingReader. This was used to perform schema validation prior to Framework 2.0, and it is now deprecated.

Validating an X-DOM

To validate an XML file or stream while reading into an X-DOM, you create an XmlReader, plug in the schemas, and then use the reader to load the DOM:

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.ValidationType = ValidationType.Schema;
settings.Schemas.Add (null, "customers.xsd");

XDocument doc;
using (XmlReader r = XmlReader.Create ("customers.xml", settings))
  try { doc = XDocument.Load (r); }
  catch (XmlSchemaValidationException ex) { ... }
```

You can also validate an XDocument or XElement that's already in memory, by calling extension methods in System.Xml.Schema. These methods accept an XmlSchemaSet (a collection of schemas) and a validation event handler:

```
XDocument doc = XDocument.Load (@"customers.xml");
XmlSchemaSet set = new XmlSchemaSet ();
set.Add (null, @"customers.xsd");
StringBuilder errors = new StringBuilder ();
doc.Validate (set, (sender, args) => { errors.AppendLine
                                       (args.Exception.Message); }
             );
Console.WriteLine (errors.ToString());
```

XSLT

XSLT stands for *Extensible Stylesheet Language Transformations*. It is an XML language that describes how to transform one XML language into another. The quintessential example of such a transformation is transforming an XML document (that typically describes data) into an XHTML document (that describes a formatted document).

Consider the following XML file:

```
<customer>
  <firstname>Jim</firstname>
  <lastname>Bo</lastname>
</customer>
```

The following XSLT file describes such a transformation:

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
    <html>
      <p><xsl:value-of select="//firstname"/></p>
      <p><xsl:value-of select="//lastname"/></p>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

The output is as follows:

```
<html>
  <p>Jim</p>
  <p>Bo</p>
```

```
</html>
```

The `System.Xml.Xsl.XslCompiledTransform` transform class efficiently performs XSLT transforms. It renders `XmlTransform` obsolete. `XslCompiledTransform` works very simply:

```
XslCompiledTransform transform = new XslCompiledTransform();
transform.Load ("customer.xslt");
transform.Transform ("customer.xml", "customer.xhtml");
```

Generally, it's more useful to use the overload of `Transform` that accepts an `XmlWriter` rather than an output file, so you can control the formatting.