Extensible Markup Language (XML)

A Primer

1/3/2013

Table of Contents

Section 1: Introduction

The United States Geoscience Information Network (USGIN) specifies the use of interchange formats to exchange and present data in an interoperable manner. One such interchange format is XML.

This tutorial is designed to familiarize the reader with the basics of XML, thereby permitting the reader to interact more easily with the ubiquitous XML documents used by USGIN.

Section 2: An XML Primer

XML stands for *Extensible Markup Language*, referring to the fact that XML is designed to be readily adapted for a variety of uses.

Because XML is a markup language, both the *content* of an XML document and the *structure* applied to that content are visible to the user. The visible structure of markup language documents is provided markup language elements.

A markup language *element* is demarcated by markup language *tags*. Opening tags *open* the element, and closing tags *close* the element. The *content* of a markup language element exists between the opening and closing tags of the element.

In a markup language document, elements appear like so:

```
<tag>content</tag>
```

For some, the above example may seem reminiscent of HTML. That is because HTML is a markup language that uses the same syntax as XML (HTML stands for "Hypertext Markup Language").

So XML and HTML are both markup languages. Both XML and HTML documents are structured by *elements*, and both XML and HTML elements are demarcated by *tags*.

Markup language elements are *hierarchical*: elements can be nested within elements. Once an element has been opened, everything between the opening tag and the closing tag, including elements, is part of the element demarcated by the surrounding tags.

For example, an XML document containing the contact information for a State Geothermal Data web service might appear as follows:

```
<ContactInfo>

     <Phone>
```

```
            <Facsimile>(987) 654-3210</Facsimile>
            <Voice>(123) 456-7890</Voice>

        </Phone>

    </ContactInfo>
```

The above example of nested markup language elements is analogous to folders within folders on a computer (Figure 1). But markup languages documents don't use folders; they use *elements*.

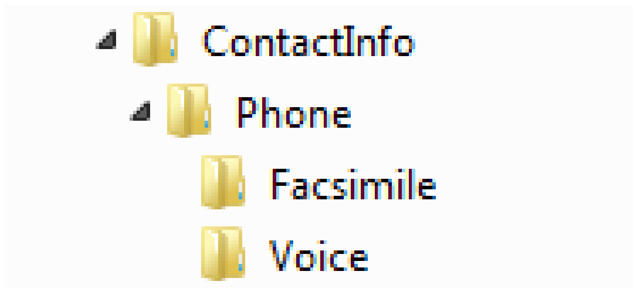

Figure 1: An example of nested folders within a computer filing system

All markup language documents begin with a *root element*: an element that contains all other elements in the document. In the case of HTML documents, this is the **HTML** element (`<html>`).

It is important to remember that, *unless otherwise specified*, elements *inherit* the characteristics of any elements within which they are nested; you will learn more about this in subsequent sections.

Markup language elements can close themselves. For example, the **horizontal rule** (`<hr>`) HTML element can be closed without a closing tag just by adding a **space** followed by a **forward slash (/)** to the opening tag of the element, like so:

```
    <hr />
```

When interpreted by a browser, the **horizontal rule** tag produces a horizontal line, like so:

XML documents sometimes employ self-closing tags, usually when *attributes* modifying the element are more important than any content *within* the element.

Section 3: XML Attributes

Most XML documents tend to look a bit more complicated than previous examples suggest, owing to the presence of attributes.

Markup language attributes specify additional parameters that modify a given element; multiple attributes can modify a single element, and each attribute can specify multiple values.

In general, markup language attributes appear within the opening tag of a given element:

```
<tag attribute="value">content</tag>
```

Attributes can also be present within self-closing elements:

```
<tag attribute="value" />
```

As a practical example, the **paragraph** (`<p>`) element is used to designate a paragraph of text in an HTML document:

```
<p>The quick brown fox jumps over the lazy dog.</p>
```

When interpreted by a web browser, the sentence between the `<p>` tags of the **paragraph** element would appear as its own paragraph, like so:

The quick brown fox jumps over the lazy dog.

But the **paragraph** element can be modified by the **style** attribute, to produce all kinds of stylistic effects:

```
<p style="font-family:cursive; font-weight:bold;">The quick
brown fox jumps over the lazy dog.</p>
```

Here, the **style** attribute (highlighted in red), as applied to the **paragraph** element, specifies multiple values that will cause all content within this particular **paragraph** element to use the **cursive** font family; the text will also be **bolded**. When interpreted by a web browser, the above **paragraph** element would appear like so:

The quick brown fox jumps over the lazy dog.

Recall that elements can be nested; this means that any given element can include other elements.

For example: the **unordered list** (`<ul>`) and **list item** (`<li>`) elements are used to create lists in HTML documents. Once the **unordered list** element is opened, any **list item** elements within are interpreted by web browsers to create bullet points on a list, like so:

```
<ul>

    <li>Item 1</li>

    <li>Item 2</li>

    <li>Item 3</li>

</ul>
```

Interpreted by a web browser, this produces a bulleted list that would appear as follows:

- Item 1
- Item 2
- Item 3

As each of the three **list item** elements are nested within the **unordered list** element, anything that modifies the **unordered list** element *also* modifies the **list item** elements. For example:

```
<ul style="font-family:cursive; font-weight:bold;">

    <li>Item 1</li>

    <li>Item 2</li>

    <li>Item 3</li>

</ul>
```

Again, the **style** attribute has been used to modify the **unordered list** element (and everything nested within it), specifying **bolded** text using the **cursive** font family. This produces the following results

- **Item 1**
- **Item 2**
- **Item 3**

Note that attributes can also be used to further modify or even override the attributes of parent elements on an individual basis. For example, the **style** attribute could be used to make **Item 1** blue or Italicized, or it could be used to specify a different font family. These parameters would take precedence over those specified by the **style** attribute modifying the **unordered list** element.

XML elements can also be modified by attributes in the exact same way. For example:

```
<OperationsMetadata>

    <Operation name='GetCapabilities' />
    </OperationsMetadata>
```

The **Operations Metadata** element (`<OperationsMetadata>`) contains metadata about the operations performed by a web service. Everything within this element, therefore, constitutes web service operation metadata.

In the above example, the **Operation** element (`<Operation>`) is modified by the **name** attribute (`name='GetCapabilities'`).

Here, the **name** attribute indicates that the name of one operation performed by the web service is **GetCapabilities**. Any content within the **Operation** element thereby constitutes metadata about the **GetCapabilities** operation. In this case, though, the **Operation** element is closed by a **forward slash** '/' and therefore contains no metadata.

Having made it this far, you are well on your way to understanding the basics of XML.

Section 4: XML Namespaces and Prefixes

Within XML documents, it sometimes becomes necessary to differentiate one element from another based on the context in which these elements are used.

For example, the `<table>` element is used differently between XML and HTML. Within an XML document in which both XML and HTML `<table>` elements are present, *namespaces* and *prefixes* can be used to differentiate XML table elements from HTML table elements.

Namespaces identify the context in which a given element is used, thereby differentiating that element from otherwise-identical elements.

Theoretically, *any* label can be used as a namespace, but XML namespaces should be URIs. Ideally, an XML namespace is an HTTP-capable URI that dereferences to a web-accessible description of the context identified by the namespace. Consequently, most XML namespaces in common use look like standard web addresses.

Going back to the above example, a namespace identifying HTML5 as the context for a given `<table>` element might appear as follows:

http://www.w3.org/TR/html5/

When entered into a web browser, this namespace dereferences to a description of HTML5.

Before a namespace can be used to differentiate a given element, the namespace must first be *declared* within the desired XML document. This is accomplished by means of the **XML namespace** attribute (`xmlns`); the namespace can be declared in two ways:

1. Declaring the namespace within the desired element, like so:

```
<table xmlns="http://www.w3.org/TR/html5/" />
```

2. Binding the namespace to an XML *prefix* that can be used as a proxy for the bound namespace within the element in which the binding took place (recall that nested elements inherit the characteristics of the parent elements in which they are nested)

Prefixes are short text strings that precede and differentiate markup language *tags*, *attributes*, and even the content of an element, like so:

```
<prefix:tag prefix:attribute="value">prefix:content</prefix:tag>
```

As indicated previously, prefixes serve as a proxy for a full namespace, but they can only do so within an element in which they have been bound to a namespace. In other words, a prefix cannot refer back to a bound namespace unless the prefix is used in an element in which the prefix has been bound to a namespace or which has inherited the binding from a parent element.

To give an example, let us assume that the `<table>` element we wish to differentiate via a bound namespace is nested within a `<body>` element. If we wished to declare a binding between a prefix and an appropriate namespace, we could do so either in the `<table>` element *or* within the `<body>` element within which the `<table>` element is nested, like so:

1. Binding takes place in the `<body>` element:

```
<body xmlns:html5="http://www.w3.org/TR/html5/">
     <html5:table>content</html5:table>

</body>
```

2. Binding takes place in the `<table>` element:

```
<html5:table xmlns:html5="http://www.w3.org/TR/html5/" />
```

In both examples, the **xmlns** attribute is used to bind the chosen prefix, html5, with the desired namespace, http://www.w3.org/TR/html5/; the prefix is then attached to the `<table>` element to invoke the namespace and thereby differentiate the `<table>` element from other `<table>` elements.

Typically, namespaces are bound to prefixes in parent elements and invoked via prefix where necessary in nested elements, except where it is absolutely necessary to bind a prefix to a namespace within the same element in which the namespace is invoked.

To give a more concrete example, color-coded in exactly the same manner as the previous example:

```
<wfs:WFS_Capabilities xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ows="http://www.opengis.net/ows">
```

This is the root element (`<WFS_Capabilities>`) of the capabilities document that describes a National Geothermal Data System web feature service. Here, two namespaces are declared and bound to two different prefixes via the **xmlns** attribute:

- The **wfs** prefix is bound to the following namespace: http://www.opengis.net/wfs
- The **ows** prefix is bound to the following namespace: http://www.opengis.net/ows

Later on in the document, the following elements are nested within the root element in which the binding took place:

```
    <ows:Abstract>Well header information for oil and gas wells
    in Arizona.</ows:Abstract>

    <wfs:Name>aasg:Wellheader</wfs:Name>
```

These prefixes indicate that the **Abstract** element is used in the context of an Open Geospatial Consortium OWS Common Implemetation Specification XML schema; likewise, the **Name** element is used in the context of an Open Geospatial Consortium web feature service. Information about the context of these elements can be found at the web location indicated by the namespaces to which the prefixes have been bound.

Section 5: XML Review

In this tutorial, we have established the following:

- XML is a markup language
- XML uses elements as basic structural components; elements are comprised by an opening tag, content, and a closing tag:

```
        <tag>content</tag>
```

- Markup language elements can close themselves:

```
        <tag />
```

- Elements can be *nested* within each other, much like folders on a computer:

```
        <tag1>
              <tag2>

                    <tag3 />
        </tag2>

    </tag1>
```

- Unless otherwise specified, nested elements inherit the characteristics of elements within which they are nested
- Markup language documents typically begin with a *root element* within which all other elements are nested
- Markup language elements can be modified by **attributes**; there can be multiple attributes per tag and multiple values per attribute:

```
<tag attribute="value" />
```

- Markup language elements can be differentiated from one another by *namespaces,* which identify a specific context in which a given element should be interpreted
- Namespaces must be declared using the **xmlns** attribute, either within a specific element or bound to a *prefix* that acts as a proxy for the full namespace
- Prefixes refer to and invoke a namespace that has been bound within a given element or the parents thereof; prefixes can modify tags, attributes, and content:

```
<prefix:tag prefix:attribute="value">prefix:content</prefix:tag>
```

With this background, it should now be possible to understand and interact with USGIN XML documents more easily.

This concludes the USGIN XML tutorial.