



## ***Intro to XML and Schemas (XSDs)***

Author: Aaron Bartell  
 Site: [www.rpg-xml.com](http://www.rpg-xml.com)

XML stands for eXtensible Markup Language and could be considered the next (or this) generation's method for holding data that needs to be defined independently of operating systems and programming languages.

The following is a relative example that demonstrates XML's basic function. Take the below address data structure:

```

D PostAdr      ds
D residential  n
D title        5a
D name         15a
D street       15a
D city         10a
D state        2a
D zip          5a
D phone        12a dim(2)
  
```

If it were filled with data it would look similar to this in memory:

```

.....1.....+.....2.....+.....3.....+.....4.....+.....5.....+.....6.....+.....7.....+...
1Mr.  Aaron Bartell  123 Center Rd  Mankato  MN56001123-123-1234321-321-4321
  
```

If that same data were to be stored in XML, there would not only be the actual variable data but also a begin and end tag around each piece of data that would essentially delimit where the data started and stopped. Look at the following xml and note the name and data relationships to the above data structure:

```

<PostAdr residential="true">
  <name title="Mr.">Aaron Bartell</name>
  <street>123 Center Rd</street>
  <city>Mankato</city>
  <state>MN</state>
  <zip>56001</zip>
  <phone>123-123-1234</phone>
  <phone>321-321-4321</phone>
</PostAdr>
  
```

As you can see, each piece of data is identifiable, as are the start and stop points. It will also be obvious that a large amount of space is utilized in describing each piece of data.

Yes, XML does require a great deal of hard drive space and memory. However, with the growing popularity of the XML standard, this has become the norm.

Note that the xml looks "pretty" in the above formatting. It is frequently stored in memory or files without carriage returns or tabs like the following example:

```
<PostAdr residential="true"><name title="Mr.">Aaron Bartell</name><street>...
```

Each piece of data is delimited by a beginning tag (i.e. <zip>) and an ending tag (i.e. </zip>). The "</" denotes the end of the tag. These tags, as they are often called, would be more appropriately called "elements", but the names are interchangeable. Each element can have children elements, attributes, and/or textual content.

In the case of tag <PostAdr> it has child elements <name>, <street>, <cty>, <state>, <zip>, and <phone> (x2). It also has an attribute named *residential*. An attribute is used to further define an element. (For more information on when to use attributes vs. elements please see the section titled "When to Use Attributes vs. Elements".)

Also note that <PostAdr> encompasses the elements <name>, <street>, <cty>, <state>, <zip>, and <phone> (x2). That is where the eXtensible part comes in. Theoretically one could put as many elements as needed/wanted within the <PostAdr> element OR embed additional elements within one of <PostAdr>'s children elements - using <name> as an example, and further defined it appears as the following:

```
<PostAdr residential="true">
  <name title="Mr.">
    <first>Aaron</first>
    <last>Bartell</last>
  </name>
  <street>123 Center Rd</street>
  <cty>Mankato</cty>
  <state>MN</state>
  <zip>56001</zip>
  <phone>123-123-1234</phone>
  <phone>321-321-4321</phone>
</PostAdr>
```

There are two new children elements within the <name> element - <first> and <last>. The xml has just officially "extended" to facilitate more detailed data.

Some might wonder "what happens to programs that are expecting the first version of <PostAdr> with the simple definition of <name>?" It would be right to assume that those programs would no longer be looking in the right place for the name of the postal address. There is no "magic" with xml. If you are trading xml documents with another program (or business partner) they **both must agree on the format**. This is where the need for XML Schema Definitions (XSDs) come into play.

The PostAdr RPG data structure has definitions for each of its sub-fields. Each sub-field has a data type and a length. If an XML file was being passed back and forth between programs there would also have to be agreements on data types so runtime errors don't occur when programs try to access (say parse) the xml. The creators of XML realized this would be needed and created what's called XSDs or XML Schema Definitions. An XSD is used to define the structure of a document (i.e. parent and child elements) and also what type of data can appear in each of the elements. Take the following XML Schema Definition which defines the <PostAdr> element.

```
<schema>
  <element name="PostAdr">
    <complexType>
      <sequence>
        <element name="name">
          <complexType>
            <sequence>
              <element name="first" type="string"/></element>
              <element name="last" type="string"/></element>
            </sequence>
            <attribute name="title" type="string" />
          </complexType>
        </element>
        <element name="street" type="string"/></element>
        <element name="cty" type="string"/></element>
        <element name="state" type="string"/></element>
        <element name="zip" type="string"/></element>
        <element name="phone" type="string" minOccurs="1" maxOccurs="2"/></element>
      </sequence>
      <attribute name="residential" type="boolean" />
    </complexType>
  </element>
</schema>
```

Oddly, this looks like an XML file. That's because XSDs use XML to define themselves. The first tag is the <schema> tag. Nothing special there except to declare what type of document we are in. Next is <element name="PostAdr">. Essentially that is declaring the name of the element being described. Within the <element...> tag there is a <complexType> tag which states PostAdr either has attributes or children elements, or both. The <name> element needs to be defined within the <PostAdr> tag and to do that the <sequence> tag is used. The contents of the <sequence> tag should contain none other than a sequence of children element definitions. Once arriving at a child that is not defined with a <complexType> there will be the *type="string"* attribute used to define the end data type that is allowed in that element. The *type* attribute could have any number of values ranging from numerical data types to dates, timestamps, Booleans, etc. More on those later.

The next thing to note is the definition of the *phone* element. It states that it can contain a string and must have at least one occurrence of the <phone> element included in the XML document at transaction time. It also states that there can be a total of two <phone> elements sent (i.e. maxOccurs="2")

The last thing to note is the definition of an element's attribute (i.e. the *residential* attribute of element PostAdr). This is located within the PostAdr's <complexType> tag. Using the <attribute> tag defines the name and type - *residential* and *boolean* respectively.

Another way to think of an XML Schema Definition is to view it similar to how iSeries Physical Files relate to one another. For instance consider an order file that holds data pertaining to orders taken over the phone. Those order files would most likely have a header type file that holds when the order was taken, the customer number, when the order will ship, etc. And then there will be a "child" file that holds the detail line items of the order specifying things like item ordered, quantity, unit price, extended price, discounts, etc.

That is a high-level overview of XML and XSD from an RPG programmer's perspective. More platform independent information is available with the below links.

Additional XSD resources

- <http://www.w3schools.com/xml/>
- <http://www.w3schools.com/schema/>
- [http://en.wikipedia.org/wiki/XML\\_schema](http://en.wikipedia.org/wiki/XML_schema)

Copyright Kregel Technology Inc 2006. All rights reserved.