

A Quick Introduction to XML Schemas

by **Jeff Volkheimer**



[Comment on this article](#)



XML ("Extensible Markup Language") is rapidly establishing itself as a useful tool for data exchange because it has the incredible potential to become a universal format for structuring information.

To use XML effectively in a community such as the Internet, there must be some constraints on the valid tags and tag sequences so that the data exchange can actually make sense to someone other than the creator. Still commonly used, DTDs (Document Type Definition) fulfilled this need. DTDs, however, have several disadvantages, such as:

- Creation of DTDs requires the use and knowledge of a completely different syntax from XML.
- Very limited ability to specify custom datatypes
- Desire commonly used database datatypes (such as dates): DTDs support 10 types

The answer to these problems is XML Schemas. Schemas overcome DTD's shortcomings and still provide the user with the power he needs. Here are some advantages of schemas:

- Schemas use the same syntax as XML, so there's less to learn.
- Allowed to specify custom datatypes.
- More predefined types (over 40!).
- Attribute grouping.

In this tutorial, we will cover the basics of writing a schema to validate XML documents.

A Simple Schema

First, we are going to use the following XML as an example data file that is intended to follow the schema that we discuss next.

```
<parks xmlns="x-schema:schema.xml">
  <zoo>
    <name>Bronx Zoo</name>
    <ID>RJx415</ID>
  </zoo>
  <theme>
    <name>Wally World</name>
    <ID>Yp1756</ID>
  </theme>
  <amusement>
    <name>Balanci's Land 'o Fun</name>
    <ID>Poo4u2</ID>
  </amusement>
</parks>
```

While this may not seem like a very useful example, we will flesh it out in a bit. So, we see we have an XML containing a listing of parks of different varieties. You may be unfamiliar with this line:

```
<parks xmlns="x-schema:schema.xml">
```

This line simply tells the validating parser where to look for our schema. In this case, it is telling the parser to retrieve the schema from the same location as the XML file. This may be a local file system directory that holds all your XML stuff or a place on the Internet.

Moving forward, our schema would look something like this:

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name='parks' content='eltOnly'>
    <element type='zoo' />
    <element type='theme' />
    <element type='amusement' />
  </ElementType>
  <ElementType name='zoo' content='eltOnly'>
    <element type='name' />
    <element type='ID' />
  </ElementType>
  <ElementType name='theme' content='eltOnly'>
    <element type='name' />
    <element type='ID' />
  </ElementType>
  <ElementType name='amusement' content='eltOnly'>
    <element type='name' />
    <element type='ID' />
  </ElementType>
  <ElementType name='name' content='textOnly' />
  <ElementType name='ID' content='textOnly' />
</Schema>
```

In this schema we declare each of our tags and how they relate to each other. Let's dissect our schema line by line. First:

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
```

This line starts off the schema with the schema element, declaring the namespace and the datatype namespace (which we will explain later). A namespace is a set of names that are specified as legal elements or attributes within an XML document. Namespaces may be declared using a URI (Universal Resource Identifier), which is in the form of some URL (Uniform Resource Locator) or URN (Uniform Resource Number). It doesn't matter whether you use a URL or a URN, since URLs are unique across the Internet.

In this tutorial, we have chosen to use the Microsoft namespace because it is a little easier to understand than the W3C standard. For more information on the W3C namespace, check [this](#) out.

Our next line:

```
<ElementType name='parks' content='eltOnly'>
```

declares type, conditions, and child elements of our element 'parks'. This line states that we will have an element with the name 'parks' that will only contain other elements and no other values. The next three lines:

```
<element type='zoo' />
<element type='theme' />
<element type='amusement' />
```

actually define what elements can legally be contained within the element 'parks'. Note that using the notation `<element type='zoo' />` is shorthand for `<element type='zoo'></element>`, which would be an empty element in XML terms. This saves time in coding a well-formed document.

We continue doing this for each element, until we get to some element that actually contains data. For example:

```
<ElementType name='name' content='textOnly' />
<ElementType name='ID' content='textOnly' />
```

declares the name element and the ID element. They both contain only text, so these are the tags that will actually contain the data in our document.

Something More Interesting

Let's spice up our XML now to make a more interesting schema. Let's remove the 'theme' and 'amusement' elements for brevity. Here is the XML we'll work with:

```
<parks xmlns="x-schema: schema.xml">
  <zoo>
    <name>Bronx Zoo</name>
    <ID>RJx415</ID>
    <animals>
      <lion species='P.l. persica'>
        <name>Joey</name>
        <gender>male</gender>
        <date_of_birth>1980-01-10</date_of_birth>
        <aggression_rating>7.0</aggression_rating>
      </lion>
      <tiger species='Panthera tigris'>
        <name>Todd</name>
        <gender>male</gender>
        <date_of_birth>1987-04-05</date_of_birth>
        <aggression_rating>9.5</aggression_rating>
      </tiger>
      <bear species='Ursus americanus'>
        <name>Sally</name>
        <gender>female</gender>
        <date_of_birth>1992-12-11</date_of_birth>
        <aggression_rating>5.3</aggression_rating>
      </bear>
    </animals>
  </zoo>
</parks>
```

Assuming everyone is very interested in zoos and their animals, let's create a schema to describe our modified example.

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name='parks' content='eltOnly'>
    <element type='zoo' />
  </ElementType>
  <ElementType name='zoo' content='eltOnly'>
    <element type='name' />
    <element type='ID' />
    <element type='animals' />
  </ElementType>
  <ElementType name='animals' content='eltOnly'>
    <element type='lion' />
    <element type='tiger' />
    <element type='bear' />
  </ElementType>
  <AttributeType name='species' required='yes' />
  <ElementType name='lion' content='mixed' >
    <attribute type='species' />
    <element type='name' />
    <element type='gender' />
    <element type='date_of_birth' />
    <element type='aggression_rating' />
  </ElementType>
  <ElementType name='tiger' content='mixed' >
    <attribute type='species' />
    <element type='name' />
```

```

    <element type='gender' />
    <element type='date_of_birth' />
    <element type='aggression_rating' />
</ElementType>
<ElementType name='bear' content='mixed' >
  <attribute type='species' />
  <element type='name' />
  <element type='gender' />
  <element type='date_of_birth' />
  <element type='aggression_rating' />
</ElementType>
<ElementType name='name' content='textOnly' />
<ElementType name='gender' content='textOnly' />
<ElementType name='date_of_birth' content='textOnly' dt:type='date' />
<ElementType name='aggression_rating'
  content='textOnly' dt:type='decimal' />
<ElementType name='ID' content='textOnly' />
</Schema>

```

It may look long but it really isn't all that different from our previous example. Our schema now describes valid animals for our zoo, outlining their name, gender, date of birth, and "aggression rating". By adding the animal element, we have introduced a number of interesting changes.

```

<AttributeType name='species' required='yes' />
<ElementType name='lion' content='mixed' >
  <attribute type='species' />
  <element type='name' />
  <element type='gender' />
  <element type='date_of_birth' />
  <element type='aggression_rating' />
</ElementType>

```

You can see the differences from our previous example in bold. First, we need to declare the attribute 'species'. It is an attribute of an element, so we use the AttributeType type definition to describe it as an attribute. We want this to be required in every element in which it is included because we may have several different 'tigers' but we want them to be distinguishable by more than just their elements.

We have also changed the content type to 'mixed' indicating that we no longer only contain elements but other types as well. In this case, our element contains attributes and elements. We also need to add the attribute to the 'lion' definition, which is the final bold item.

In addition to adding attributes, datatypes are now used actively. For example:

```

<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  :
<ElementType name='date_of_birth' content='textOnly' dt:type='date' />
<ElementType name='aggression_rating'
  content='textOnly' dt:type='decimal' />

```

In the schema element declaration, we assign the dt prefix to the datatype namespace we desire. This is used later on, as seen in the 'date_of_birth' and 'aggression_rating' elements, where we define those as being of type 'date' and 'decimal' respectively. This is useful because it allows us to impose restrictions on the type of data that can be contained in those elements. After all, it does not make much sense to have "Elephant" as a date of birth or "&" for an aggression rating.

Here is a list of some common primitive "built-in" datatypes you may want to use (this list is not complete):

Datatype	Example
string	"Schemas are great!"
decimal	4.05
float	82, 99.2E4, INF (infinity)
double	82, 99.2E4, INF (infinity)
integer	6
long	-9223372036854775808 to 9223372036854775808
boolean	true, false
date	1978-04-05

Conclusion

XML Schemas are the result of a need for a new way to make XML a viable solution for data exchange and storage. Their power and ease of use make them the clear choice for XML definition.

References

The W3C has all the latest on today's web standards:

<http://www.w3.org/XML/Schema>

Sun's quick XML tutorial:

http://java.sun.com/xml/docs/tutorial/overview/1_xml.html

A cool XML editor and validator...probably one of the best available! Free evaluation demo:

<http://link.xmlspy.com/>

© 2001 [Interface Technologies, Inc.](#) All Rights Reserved

Questions or Comments? devcentral@itcentral.com

[PRIVACY POLICY](#)