Understanding RELAX NG

XML validation that's just right

Skill Level: Introductory

Nicholas Chase (nicholas@nicholaschase.com)
Author
Studio B

04 Dec 2003

Many developers find DTD validation too simple (it doesn't use XML syntax, and doesn't enable developers to specify many of the rules needed in the real world) and XML Schema too complicated. RELAX NG represents a compromise: It uses XML syntax, and it enables developers to create most of the same rules as the W3C XML Schema language, but with a greatly simplified syntax. This tutorial explains the concepts behind RELAX NG in both its XML and compact forms.

Section 1. Introduction

What is this tutorial about?

As far back as the original XML 1.0 Recommendation, XML has included the ability to validate it, or compare it to a predefined structure. The first version of validation was called the Document Type Definition (DTD), but many developers thought this was too simple: It doesn't use XML syntax, and doesn't enable developers to specify many of the rules needed in the real world. The W3C then developed XML Schema, but many developers thought this was too complicated: It uses XML syntax and enables users to create just about any rule under the sun, but at the cost of adding complexity. Others combined the TREX and RELAX proposals into **RELAX NG** and many developers think it's just right. It uses XML syntax, and it enables developers to create most of the same rules as the W3C XML Schema language, but with a greatly simplified syntax.

This tutorial explains the concepts behind RELAX NG in both its XML and compact forms. It covers:

- Creating a RELAX NG schema
- Elements
- Attributes
- Types
- Optional items
- Mixed content
- Restrictions
- Modularizing a schema
- Creating these rules using the compact syntax

The tutorial doesn't actually build an application; it uses a command-line application to simply validate the document against the schema document.

Tools

This tutorial doesn't involve any actual programming, but to verify the schema files you create, you'll need a validator. RELAX NG validators exist for a number of different languages, including:

- Java technology -- Jing (http://www.thaiopensource.com/relaxng/jing.html) and MSV (http://wwws.sun.com/software/xml/developers/multischema/)
- C -- Libxml2 (http://www.xmlsoft.org/)
- C# -- Tenuto (http://sourceforge.net/projects/relaxng)
- Python -- XVIF (http://freshmeat.net/projects/xvif/?topic_id=868)
- ActiveX -- a DLL, VBRELAXNG (http://www.geocities.co.jp/SiliconValley-Bay/4639/vbrelaxng/vbrelaxng.html)

For more information on available tools, check out http://www.relaxng.org/#software.

Before beginning the tutorial, install your chosen software according to its instructions and make sure you know how to actually validate a file.

This tutorial does assume a basic understanding of XML.

Section 2. Getting started

What is RELAX NG (and how do I pronounce it?)

When the XML 1.0 Recommendation was first created it included the definition for DTDs, the validation format for XML's predecessor, SGML. DTDs were useful in that they enabled a developer to create a hierarchy that an XML document must follow.

Unfortunately, DTDs suffered from two serious drawbacks. First, they were not XML documents -- for example, a DTD might look something like this:

```
<!ELEMENT memories (memory+)>
<!ELEMENT memory (media, subdate, donor, subject, location)>
<!ATTLIST memory mediaid CDATA #REQUIRED
                 status CDATA #REQUIRED>
<!ELEMENT subdate (#PCDATA)>
<!ELEMENT donor (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT location (description | place)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT place (#PCDATA)>
```

Second, they had a limited ability to create data types and other restrictions.

In response, groups of developers began creating new, XML-based means for creating the hierarchy (or grammar) that the XML document must follow. The official standard is the W3C XML Schema language, but that also suffers from several handicaps, primarily that creating complex rules can be difficult.

But the W3C doesn't have a lock on creating schema languages. The open source community has combined two different proposals, TREX and RELAX, to form RELAX NG (pronounced "relaxing"), an XML-based schema language that provides many of the same capabilities as the W3C's schema language but is much simpler. (RELAX NG also provides a non-XML *compact* syntax that I'll discuss later, in Compact syntax .)

In the tutorial, you'll learn how to use RELAX NG to validate a single source file.

The source file

This validation source file is from the Millennium Memory project, which takes video tapes and other media showing ordinary life and preserves it for later, when things have changed. Users can then check out a memory tape from the library.

The sample document looks like this:

```
<?xml version="1.0"?>
<memories>
   <memory tapeid="1">
      <media mediaid="1" status="vhs" />
      <subdate>2001-05-23</subdate>
      <donor>John Baker</donor>
      <subject>Fishing off Pier 60</subject>
      <location>
```

```
          <description>Outside in the woods</description>
      </location>
   </memory>
   <memory tapeid="2">
      <media mediaid="2" status="vhs"/>
      <subdate>2001-05-18</subdate>
      <donor>Elizabeth Davison</donor>
      <subject>Beach volleyball</subject>
      <location>
         <place>Clearwater beach</place>
      </location>
   </memory>
</memories>
```

The W3C schema

To give you an idea of what's involved, here's a list of the conditions you would need to specify if you were to build the grammar using the W3C's XML Schema language:

```
<xsd:element name="memories">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="memory" minOccurs = "0" maxOccurs="unbounded"
              type="memoryType"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:complexType name="memoryType">
   <xsd:sequence>
      <xsd:element name="media">
         <xsd:complexType>
            <xsd:attribute name="mediaid" type="xsd:integer" />
            <xsd:attribute name="status" type="mediaType" />
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="subdate" type="xsd:date"/>
      <xsd:element name="donor" type="xsd:string"/>
      <xsd:element name="subject">
         <xsd:complexType mixed="true">
             <xsd:all>
                <xsd:element name="i" minOccurs="0" maxOccurs="1"
                     type="xsd:string" />
                <xsd:element name="b" minOccurs="0" maxOccurs="1"
                     type="xsd:string" />
             </xsd:all>
         </xsd:complexType>
      </xsd:element>
      <xsd:element name="location" type="locationType" />
   </xsd:sequence>
   <xsd:attribute name="tapeid" type="idNumber" />
   <xsd:attribute name="status" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="locationType">
   <xsd:choice>
      <xsd:element name="description" type="xsd:anyType" />
      <xsd:element name="place" type="xsd:string" />
   </xsd:choice>
</xsd:complexType>

<xsd:simpleType name="idNumber">
   <xsd:restriction  base="xsd:integer">
      <xsd:minInclusive value="1" />
      <xsd:maxInclusive value="100000" />
   </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="mediaType">
   <xsd:restriction  base="xsd:string">
      <xsd:enumeration value="8mm" />
      <xsd:enumeration value="vhs" />
      <xsd:enumeration value="vhsc" />
      <xsd:enumeration value="digital" />
      <xsd:enumeration value="audio" />
   </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

I'll talk more about each of these conditions as I show you how to code them into a RELAX NG schema.

Section 3. The basic structure

The skeleton file

I'll start by creating the most basic RELAX NG schema file:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">

<start>
</start>

</grammar>
```

This is, of course, the XML syntax for RELAX NG schemas. The root element is the `grammar` element, with the `start` element showing where the validating parser should begin looking for definitions. In some cases, the entire definition can be contained within the `start` element, but in Internal definitions I'll talk about referring back to definitions that live outside the `start` element.

Save this definition in a file -- I'll call mine `schema.rng` -- and check your validation tool to make sure it's working properly. For example, to run this file through Jing, type:

```
jing schema.rng source.xml
```

You should see lots of errors because you haven't actually defined the content in the document yet. I'll look at that next.

Adding the elements

I'll start by adding the basic elements to the schema. In a W3C XML Schema document, you'd have to start by creating complex types, but here you're just going

to create simple elements, nested as they would be in the document:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
      <element name="memory">
         <element name="media" />
         <element name="subdate">
            <text />
         </element>
         <element name="donor">
            <text />
         </element>
         <element name="subject">
            <text />
         </element>
         <element name="location">
            <text />
         </element>
      </element>
   </element>
</start>

</grammar>
```

Notice that some of the elements, such as `media`, are empty, while others have either element content (such as `memory`) or text (such as `subdate` or `donor`). As you go along, you'll see that you can be much more specific about the type of content that an element can or must contain.

Adding the attributes

Next I'll look at adding attributes:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
      <element name="memory">
          <attribute name="tapeid">
              <text />
          </attribute>
          <element name="media">
              <attribute name="mediaid">
                  <text />
              </attribute>
              <attribute name="status">
                  <text />
              </attribute>
          </element>
          <element name="subdate">
              <text />
          </element>
          <element name="donor">
              <text />
          </element>
          <element name="subject">
              <text />
          </element>
          <element name="location">
              <text />
          </element>
      </element>
```

```
   </element>
</start>

</grammar>
```

If you run the file at this point, the only errors you see should pertain to the `location` element, which I'll look at next.

Notice that attributes are contained within their elements, and that they also specify the type of content they can contain. For attributes, it follows that they can only contain text -- but you can also be more specific about what types of text they can contain, as you'll see when I talk about Types and restrictions . Before you do that, however, take a look at some other structural opportunities.

Section 4. Adding flexibility

Mixed content

Now that you've got the basic document in place, you need to start making allowances for some items that don't necessarily fall into neat little piles, such as multiple choices and optional elements and attributes. One of these conundrums is mixed content.

Defining mixed content, or content that contains both text and elements, has been a problem since XML was defined using DTDs. In a RELAX NG schema, you can solve the problem easily using the `mixed` element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
  <element name="memories">
    <element name="memory">
      <attribute name="tapeid">
         <text />
      </attribute>
      <element name="media">
         <attribute name="mediaid"><text /></attribute>
         <attribute name="status"><text /></attribute>
      </element>
      <element name="subdate">
         <text />
      </element>
      <element name="donor">
         <text />
      </element>
      <element name="subject">
         <mixed>
           <element name="b"><text /></element>
           <element name="i"><text /></element>
         </mixed>
       </element>
       <element name="location">
         <text />
```

```
        </element>
     </element>
  </element>
</start>

</grammar>
```

In this case, the `subject` element contains text as well as a single `b` and a single `i` element. What's more, you can make these elements optional.

Optional items

Sometimes you don't want to require an item, you just want to allow it. For example, I want to allow bold and italic information in the subject, but I don't want them to be mandatory:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
      <element name="memory">
         <attribute name="tapeid">
             <text />
         </attribute>
         <element name="media">
             <attribute name="mediaid"><text /></attribute>
             <attribute name="status"><text /></attribute>
         </element>
         <element name="subdate">
             <text />
         </element>
         <element name="donor">
             <text />
         </element>
         <element name="subject">
             <mixed>
                <optional>
                   <element name="b"><text /></element>
                </optional>
                <optional>
                   <element name="i"><text /></element>
                </optional>
             </mixed>
         </element>
         <element name="location">
             <text />
         </element>
      </element>
   </element>
</start>

</grammar>
```

In this case I've made elements optional, but you can use the same method to create optional attributes.

Choices

In some cases, you want to allow the user one or more options. For example, in the current structure, the `location` element must contain either a `description`

element or a `place` element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
      <element name="memory">
         <attribute name="tapeid">
             <text />
         </attribute>
         <element name="media">
             <attribute name="mediaid"><text /></attribute>
             <attribute name="status"><text /></attribute>
         </element>
         <element name="subdate">
             <text />
         </element>
         <element name="donor">
             <text />
         </element>
         <element name="subject">
             <mixed>
                <optional>
                   <element name="b"><text /></element>
                </optional>
                <optional>
                   <element name="i"><text /></element>
                </optional>
             </mixed>
         </element>
         <element name="location">
             <choice>
                 <element name="description"><text/></element>
                 <element name="place"><text/></element>
             </choice>
         </element>
      </element>
   </element>
</start>

</grammar>
```

You can also use the `group` element to group together more or different choices, if necessary.

Multiple items

In some cases, you want to allow more than one of a particular item. For example, this file wouldn't be very useful if it couldn't hold more than one `memory` element, so you can specify that more are allowed:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
     <oneOrMore>
       <element name="memory">
           <attribute name="tapeid"><text /></attribute>
           <element name="media">
               <attribute name="mediaid"><text /></attribute>
               <attribute name="status"><text /></attribute>
           </element>
           <element name="subdate">
               <text />
```

```
            </element>
            <element name="donor">
                <text />
            </element>
            <element name="subject">
                <mixed>
                   <optional>
                      <element name="b"><text /></element>
                   </optional>
                   <optional>
                      <element name="i"><text /></element>
                   </optional>
                </mixed>
            </element>
            <element name="location">
                <choice>
                    <element name="description"><text/></element>
                    <element name="place"><text/></element>
                </choice>
            </element>
        </element>
      </oneOrMore>
    </element>
</start>

</grammar>
```

Similarly, you can use the `zeroOrMore` element to specify an optional element that can be repeated. Note that because XML doesn't allow multiple attributes with the same name (in the same namespace) on a single element, both `zeroOrMore` and `oneOrMore` are just for elements and groups, rather than for attributes.

This structure does create a challenge, however, for situations in which you want to enable a *maximum* number of items.

Limiting multiple items

One limitation in RELAX NG involves specifying a maximum number of items. For example, if I wanted to specify in the W3C's XML Schema language that a particular memory could have up to three donors, I could simply indicate `maxOccurs="3"`. RELAX NG has no such construct, so if you need it, you'll have to create a workaround by using optional items:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
     <oneOrMore>
       <element name="memory">
            <attribute name="tapeid"><text /></attribute>
            <element name="media">
                <attribute name="mediaid"><text /></attribute>
                <attribute name="status"><text /></attribute>
            </element>
            <element name="subdate">
                <text />
            </element>
            <element name="donor">
                <text />
            </element>
            <optional>
                <element name="donor">
                  <text />
```

```
                </element>
                <optional>
                    <element name="donor">
                        <text />
                    </element>
                </optional>
            </optional>
            <element name="subject">
                <mixed>
                    <optional>
                        <element name="b"><text /></element>
                    </optional>
                    <optional>
                        <element name="i"><text /></element>
                    </optional>
                </mixed>
            </element>
            <element name="location">
                <choice>
                    <element name="description"><text/></element>
                    <element name="place"><text/></element>
                </choice>
            </element>
        </element>
    </oneOrMore>
  </element>
</start>

</grammar>
```

Next I'll look at items that are grouped together, but not necessarily in a particular order.

Unordered items

In some cases, you want particular items to be part of an element, but you're not particularly interested in the order in which they appear. You can take care of this with the `interleave` element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
     <oneOrMore>
        <element name="memory">
            <attribute name="tapeid"><text /></attribute>
            <interleave>
               <element name="media">
                   <attribute name="mediaid"><text /></attribute>
                   <attribute name="status"><text /></attribute>
               </element>
               <element name="subdate">
                   <text />
               </element>
               <element name="donor">
                   <text />
               </element>
               <element name="subject">
                   <mixed>
                      <optional>
                         <element name="b"><text /></element>
                      </optional>
                      <optional>
                         <element name="i"><text /></element>
                      </optional>
                   </mixed>
```

```
                </element>
                <element name="location">
                    <choice>
                        <element name="description"><text/></element>
                        <element name="place"><text/></element>
                    </choice>
                </element>
            </interleave>
        </element>
     </oneOrMore>
   </element>
</start>

</grammar>
```

In this case, the non-optional elements that describe the `memory` must all appear, but the order is insignificant. In this way, `interleave` is similar to a simple group.

Lists

You can also specify that an element or attribute can contain more than one tokenized value. For example, the `memory` itself might be available in more than one format, as specified by a list of values in the `status` attribute:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
     <oneOrMore>
        <element name="memory">
           <attribute name="tapeid"><text /></attribute>
           <element name="media">
               <attribute name="mediaid"><text /></attribute>
               <attribute name="status">
                   <list>
                      <oneOrMore><text /></oneOrMore>
                   </list>
               </attribute>
           </element>
           <element name="subdate">
               <text />
           </element>
           <element name="donor">
               <text />
           </element>
           <element name="subject">
               <mixed>
                  <optional>
                     <element name="b"><text /></element>
                  </optional>
                  <optional>
                     <element name="i"><text /></element>
                  </optional>
               </mixed>
           </element>
           <element name="location">
               <choice>
                  <element name="description"><text/></element>
                  <element name="place"><text/></element>
               </choice>
           </element>
        </element>
     </oneOrMore>
   </element>
</start>
```

```
</grammar>
```

In this case, the `status` attribute must contain a list of one or more text items. Note that in RELAX NG, a list of items is space-delimited.

Completely flexible elements

To conclude this section, I'll look at making things as flexible as possible. You can use the `anyName` element to specify the user's ability to add any element or any attribute (depending on how you use `anyName`) in a particular location. For example:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories">
     <oneOrMore>
        <element name="memory">
           <attribute name="tapeid"><text /></attribute>
           <element name="media">
               <attribute name="mediaid"><text /></attribute>
               <attribute name="status"><text /></attribute>
           </element>
           <element name="subdate">
               <text />
           </element>
           <element name="donor">
               <text />
           </element>
           <element name="subject">
               <mixed>
                  <zeroOrMore>
                      <element>
                          <anyName />
                          <text />
                      </element>
                  </zeroOrMore>
               </mixed>
           </element>
           <element name="location">
               <choice>
                    <element name="description"><text/></element>
                    <element name="place"><text/></element>
               </choice>
           </element>
        </element>
     </oneOrMore>
   </element>
</start>

</grammar>
```

In this case, the user can add, as part of the content of the `subject` element, zero or more elements that can have any name and can contain only text. For example:

```
<subject>This tape chronicles my life in a biker gang.</subject>
<subject>This tape chronicles <em>my life in a biker gang</em>.</subject>
<subject>This tape chronicles <b>my life</b> in a <b>biker gang</b>.</subject>
<subject>This tape chronicles <i>my life</i> in a <b>biker gang</b>.</subject>
```

That covers architectural options, so now I'll look at restricting the content itself.

Section 5. Types and restrictions

Empty elements

Now I'll take a look at the idea of restricting the type of content that an element can contain.

The simplest and most obvious type of restriction is an element that can't contain anything. For example, the existing structure shows the media element as empty, with only attributes. If, however, the media element had no attributes, you'd have a problem, because an `element` element must contain at least one child. To solve that problem, you can explicitly set an element as `empty`:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
  <element name="memories">
    <oneOrMore>
      <element name="memory">
          <attribute name="tapeid"><text /></attribute>
          <element name="media">
            <empty />
        </element>
        <element name="subdate">
            <text />
        </element>
        <element name="donor">
            <text />
        </element>
        <element name="subject">
            <mixed>
              <optional>
                <element name="b"><text /></element>
              </optional>
              <optional>
                <element name="i"><text /></element>
              </optional>
            </mixed>
        </element>
        <element name="location">
            <choice>
              <element name="description"><text/></element>
              <element name="place"><text/></element>
            </choice>
        </element>
      </element>
    </oneOrMore>
  </element>
</start>

</grammar>
```

In most cases, however, when you're limiting content, you're limiting it to a specific type of content.

Linking to W3C schema types

One of the major strengths of W3C XML Schema over DTDs is the addition of data types. Instead of simply indicating that an element can contain text, you can require it to be in the form of a date or a number.

Rather than reinventing the wheel, RELAX NG provides a way to simply refer to XML Schema data types. The first step is to reference W3C XML Schema:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories" datatypeLibrary=
    "http://www.w3.org/2001/XMLSchema-datatypes">
     <oneOrMore>
        <element name="memory">
           <attribute name="tapeid"><text /></attribute>
           <element name="media">
              <attribute name="mediaid"><text /></attribute>
              <attribute name="status"><text /></attribute>
           </element>
           <element name="subdate">
              <data type="date" />
           </element>
           <element name="donor">
              <text />
           </element>
           <element name="subject">
              <mixed>
                 <optional>
                    <element name="b"><text /></element>
                 </optional>
                 <optional>
                    <element name="i"><text /></element>
                 </optional>
              </mixed>
           </element>
           <element name="location">
              <choice>
                 <element name="description"><text/></element>
                 <element name="place"><text/></element>
              </choice>
           </element>
        </element>
     </oneOrMore>
   </element>
</start>

</grammar>
```

The `datatypeLibrary` attribute is inherited by all children of the element on which it's specified, so now you're ready to refer to the specific types in any of the grammar's elements.

Adding types

Because the schema now references XML Schemas as the basis for its type names, you can reference them directly within the document by using the data element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories" datatypeLibrary=
       "http://www.w3.org/2001/XMLSchema-datatypes">
     <oneOrMore>
        <element name="memory">
           <attribute name="tapeid">
              <data type="integer" />
           </attribute>
           <element name="media">
              <attribute name="mediaid"><data type="integer" /></attribute>
              <attribute name="status"><text /></attribute>
           </element>
           <element name="subdate">
              <data type="date" />
           </element>
           <element name="donor">
              <text />
           </element>
           <element name="subject">
              <mixed>
                 <optional>
                    <element name="b"><text /></element>
                 </optional>
                 <optional>
                    <element name="i"><text /></element>
                 </optional>
              </mixed>
           </element>
           <element name="location">
              <choice>
                 <element name="description"><text/></element>
                 <element name="place"><text/></element>
              </choice>
           </element>
        </element>
     </oneOrMore>
   </element>
</start>

</grammar>
```

In some cases, just specifying the type isn't enough -- fortunately, you can also include additional restrictions.

Minimum and maximum values

One of the other strengths of W3C XML Schema is the ability to limit the content of an element or attribute even more explicitly than by simply requiring it to be of a certain type. For example, in an XML Schema document, I could specify that the `tapeid` must be an integer between `1` and `100000` by adding `minInclusive` and `maxInclusive` attributes to the element definition. RELAX NG provides the same flexibility, but makes it much easier for implementers by adding the information as parameters on the data element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories" datatypeLibrary=
       "http://www.w3.org/2001/XMLSchema-datatypes">
     <oneOrMore>
        <element name="memory">
           <attribute name="tapeid">
```

```
                <data type="integer" >
                    <param name="minInclusive">1</param>
                    <param name="maxInclusive">1000000</param>
                </data>
            </attribute>
         <element name="media">
            <attribute name="mediaid"><data type="integer" /></attribute>
            <attribute name="status"><text /></attribute>
         </element>
         <element name="subdate">
            <data type="date" />
         </element>
         <element name="donor">
            <text />
         </element>
         <element name="subject">
            <mixed>
               <optional>
                   <element name="b"><text /></element>
               </optional>
               <optional>
                   <element name="i"><text /></element>
               </optional>
            </mixed>
         </element>
         <element name="location">
            <choice>
               <element name="description"><text/></element>
               <element name="place"><text/></element>
            </choice>
         </element>
       </element>
     </oneOrMore>
   </element>
</start>

</grammar>
```

This flexibility enables you to easily add other restrictions as well. In general, any parameters that you'd normally add to an XML Schema document's element definition can be added to a data element as a parameter. For example, you could limit the description and place elements to 255 characters:

```
...
    <choice>
       <element name="description">
           <data type="string">
              <param name="maxLength">255</param>
           </data>
       </element>
       <element name="place">
           <data type="string">
              <param name="maxLength">255</param>
           </data>
       </element>
    </choice>
...
```

You can also limit the content to a particular set of values.

Enumerated values

Often you want to limit the values that an element or attribute can contain to a particular enumerated set of values. In RELAX NG, you can do this by using the

`choice` element in conjunction with the `value` element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
   <element name="memories"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
     <oneOrMore>
        <element name="memory">
           <attribute name="tapeid">
              <data type="integer" >
                  <param name="minInclusive">1</param>
                  <param name="maxInclusive">1000000</param>
              </data>
           </attribute>
           <element name="media">
              <attribute name="mediaid"><data type="integer" /></attribute>
              <attribute name="status">
                  <choice>
                      <value>8mm</value>
                      <value>vhs</value>
                      <value>vhsc</value>
                      <value>digital</value>
                      <value>audio</value>
                  </choice>
              </attribute>
           </element>
           <element name="subdate">
              <data type="date" />
           </element>
           <element name="donor">
              <text />
           </element>
           <element name="subject">
              <mixed>
                 <optional>
                    <element name="b"><text /></element>
                 </optional>
                 <optional>
                    <element name="i"><text /></element>
                 </optional>
              </mixed>
           </element>
           <element name="location">
              <choice>
                 <element name="description"><text/></element>
                 <element name="place"><text/></element>
              </choice>
           </element>
        </element>
     </oneOrMore>
   </element>
</start>

</grammar>
```

Now that the structure itself is defined, I'll look at modularizing the definitions for better readability and easier maintenance.

Section 6. Modularizing the schema

Internal definitions

As it's defined now, the structure itself is fine, but in general it's a good idea to modularize the definitions. Modularizing them provides several advantages. In addition to making the file easier to read, modularizing provides the opportunity to separate some definitions into separate files, as you'll see in Referencing external definitions.

I'll start by looking at internal definitions. Here I've separated the main definitions into separate entities in the schema. (In the next panel I'll replace each "..." with a reference.)

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
<start>
   <element name="memories">
<oneOrMore>
...
</oneOrMore>
   </element>
</start>

<define name="memoryType">
   <element name="memory">
            <attribute name="tapeid">
               <data type="integer" >
                  <param name="minInclusive">1</param>
                  <param name="maxInclusive">1000000</param>
               </data>
            </attribute>

...

   </element>

</define>

<define name="mediaType">
      <element name="media">
         <attribute name="mediaid">
            <data type="integer" />
         </attribute>
         <attribute name="status">
            <choice>
               <value>8mm</value>
               <value>vhs</value>
               <value>vhsc</value>
               <value>digital</value>
               <value>audio</value>
            </choice>
         </attribute>
      </element>

</define>

<define name="submissionInfo">
    <group>
      <element name="subdate">
         <data type="date" />
      </element>
      <element name="donor">
         <text />
      </element>
    </group>
</define>
```

```
<define name="subjectType">
      <element name="subject">
         <mixed>
            <optional>
               <element name="b"><text /></element>
            </optional>
            <optional>
               <element name="i"><text /></element>
            </optional>
         </mixed>
      </element>
</define>

<define name="locationType">
      <element name="location">
         <choice>
            <element name="description"><text/></element>
            <element name="place"><text/></element>
         </choice>
      </element>
</define>

</grammar>
```

Here I've created the definitions outside the `start` element, but within the `grammar` element. I can then refer to them from within the document.

Referencing internal definitions

Once you've created the definitions, making use of them is a simple matter of referencing them, just as you'd do in a W3C XML Schema document:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
<start>
   <element name="memories">
      <oneOrMore>
         <ref name="memoryType" />
      </oneOrMore>
   </element>
</start>

<define name="memoryType">
      <element name="memory">
         <attribute name="tapeid">
              <data type="integer" >
                   <param name="minInclusive">1</param>
                   <param name="maxInclusive">1000000</param>
              </data>
         </attribute>

         <ref name="mediaType" />
         <ref name="submissionInfo" />
         <ref name="subjectType" />
         <ref name="locationType" />

      </element>

</define>

<define name="mediaType">
         <element name="media">
             <attribute name="mediaid"><data type="integer" /></attribute>
             <attribute name="status">
                 <choice>
```

```
                    <value>8mm</value>
                    <value>vhs</value>
                    <value>vhsc</value>
                    <value>digital</value>
                    <value>audio</value>
                </choice>
            </attribute>
        </element>
</define>

<define name="submissionInfo">
    <group>
        <element name="subdate">
            <data type="date" />
        </element>
        <element name="donor">
            <text />
        </element>
    </group>
</define>

<define name="subjectType">
        <element name="subject">
            <mixed>
               <optional>
                  <element name="b"><text /></element>
               </optional>
               <optional>
                  <element name="i"><text /></element>
               </optional>
            </mixed>
        </element>
</define>

<define name="locationType">
        <element name="location">
             <choice>
                <element name="description"><text/></element>
                <element name="place"><text/></element>
             </choice>
        </element>
</define>

</grammar>
```

You can also reference definitions that are not part of the document itself.

Referencing external definitions

In addition to referencing definitions within the schema file, you can create external files and then reference them. For example, suppose you thought that at some point you might want to replace the `subject` definition. You could pull it into a separate file, perhaps called `subjectDef.rng` :

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<start>
    <ref name="subjectType" />
</start>

<define name="subjectType">
        <element name="subject">
            <mixed>
               <optional>
                  <element name="b"><text /></element>
               </optional>
```

```
                <optional>
                   <element name="i"><text /></element>
                </optional>
            </mixed>
         </element>
</define>
</grammar>
```

You could then reference that definition from within the original schema document by using the `externalRef` element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
        datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
<start>
   <element name="memories">
       <oneOrMore>
           <ref name="memoryType" />
       </oneOrMore>
   </element>
</start>

<define name="memoryType">
    <element name="memory">
         <attribute name="tapeid">
             <data type="integer" >
                 <param name="minInclusive">1</param>
                 <param name="maxInclusive">1000000</param>
             </data>
         </attribute>

         <ref name="mediaType" />
         <ref name="submissionInfo" />
         <externalRef href="subjectDef.rng" />
         <ref name="locationType" />

      </element>

</define>

<define name="mediaType">
         <element name="media">
              <attribute name="mediaid"><data type="integer" /></attribute>
              <attribute name="status">
                  <choice>
                      <value>8mm</value>
                      <value>vhs</value>
                      <value>vhsc</value>
                      <value>digital</value>
                      <value>audio</value>
                  </choice>
              </attribute>
         </element>
</define>

<define name="submissionInfo">
    <group>
         <element name="subdate">
             <data type="date" />
         </element>
         <element name="donor">
             <text />
         </element>
    </group>
</define>


<define name="locationType">
    <element name="location">
         <choice>
             <element name="description"><text/></element>
```

```
            <element name="place"><text/></element>
        </choice>
    </element>
</define>

</grammar>
```

Note that the `externalRef` element simply references the `start` element of the external file and any definitions it creates or references.

Including other files

You can also include other files directly, and then reference their definitions. For example, you could include the `subjectDef.rng` file and then treat the `subjectType` as though it were created within the `schema.rng` file:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="subjectDef.rng" />

<start>
   <element name="memories">
<oneOrMore>
<ref name="memoryType" />
</oneOrMore>
   </element>
</start>

<define name="memoryType">
      <element name="memory">
                <attribute name="tapeid">
                     <data type="integer" >
                         <param name="minInclusive">1</param>
                         <param name="maxInclusive">1000000</param>
                     </data>
                </attribute>

         <ref name="mediaType" />
         <ref name="submissionInfo" />
         <ref name="subjectType" />
         <ref name="locationType" />

      </element>

</define>

<define name="mediaType">
         <element name="media">
                <attribute name="mediaid"><data type="integer" /></attribute>
                <attribute name="status">
<choice>
    <value>8mm</value>
    <value>vhs</value>
    <value>vhsc</value>
    <value>digital</value>
    <value>audio</value>
</choice>
</attribute>
         </element>

</define>

<define name="submissionInfo">
<group>
```

```
          <element name="subdate">
              <data type="date" />
          </element>
          <element name="donor">
              <text />
          </element>
</group>
</define>


<define name="locationType">
          <element name="location">
                  <choice>
                       <element name="description"><text/></element>
                       <element name="place"><text/></element>
                  </choice>
          </element>
</define>

</grammar>
```

Note, however, that if you did it this way, you'd essentially have two `start` elements, so you would have to alter the `subjectDef.rng` file to read:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
<define name="subjectType">
          <element name="subject">
              <mixed>
                 <optional>
                    <element name="b"><text /></element>
                 </optional>
                 <optional>
                    <element name="i"><text /></element>
                 </optional>
              </mixed>
          </element>
</define>
</grammar>
```

RELAX NG also provides a way to override external definitions.

Overriding external definitions

The `subjectDef.rng` file only contains one definition, but in some cases, you might have an external file with many definitions, of which you might want to override one or two. RELAX NG enables you to do this by adding information to the `include` element:

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="subjectDef.rng">
    <define name="subjectType">
        <element name="subject">
           <text />
        </element>
    </define>
</include>
```

```
<start>
   <element name="memories">
       <oneOrMore>
          <ref name="memoryType" />
       </oneOrMore>
   </element>
</start>

<define name="memoryType">
    <element name="memory">
         <attribute name="tapeid">
             <data type="integer" >
                 <param name="minInclusive">1</param>
                 <param name="maxInclusive">1000000</param>
             </data>
         </attribute>

         <ref name="mediaType" />
         <ref name="submissionInfo" />
         <ref name="subjectType" />
         <ref name="locationType" />

      </element>

</define>

<define name="mediaType">
     <element name="media">
         <attribute name="mediaid"><data type=
                     "integer" /></attribute>
         <attribute name="status">
             <choice>
                 <value>8mm</value>
                 <value>vhs</value>
                 <value>vhsc</value>
                 <value>digital</value>
                 <value>audio</value>
             </choice>
         </attribute>
     </element>
</define>

<define name="submissionInfo">
    <group>
         <element name="subdate">
             <data type="date" />
         </element>
         <element name="donor">
             <text />
         </element>
   </group>
</define>


<define name="locationType">
   <element name="location">
       <choice>
           <element name="description"><text/></element>
           <element name="place"><text/></element>
       </choice>
   </element>
</define>

</grammar>
```

This action replaces the definition of the `subjectType` with the version contained in `schema.rng`.

Next I'll look at another way to specify RELAX NG schemas.

Section 7. Compact syntax

Elements

One interesting aspect of RELAX NG is that it doesn't actually require you to use XML to define your structures. Instead, you can use the compact syntax, which is similar to specifying class or table structures.

For example, you can create a compact document that specifies the basic elements and their content:

```
start =
  element memories {
    element memory {
      element media { empty },
      element subdate { text },
      element donor { text },
      element subject { text },
      element location {
        element description { text }
      }
    }
  }
```

Save this file as `schema.rnc`. To validate the source document against it in Jing, use:

```
jing -c schema.rnc source.xml
```

Attributes

Once the elements are in place, go ahead and add the attributes within the element definitions:

```
start =
  element memories {
    element memory {
      attribute tapeid { text },
      element media {
        attribute mediaid { text },
                          attribute status { text }
      },
      element subdate { text },
      element donor { text },
      element subject { text },
      element location {
        element description { text }
      }
    }
  }
```

As before, you can also specify types for both element and attribute content.

Types

To specify types, reference the XML Schemas namespace and add the types just as you did in the XML syntax:

```
                    namespace xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
start =
  element memories {
    element memory {
      attribute tapeid {
xsd:integer
  },
      element media {
        attribute mediaid {
xsd:integer
  },
        attribute status { text }
      },
      element subdate {
xsd:date
  },
      element donor { text },
      element subject { text },
      element location {
        element description { text }
        | element place { text }
      }
    }
  }
```

Connectors

The compact syntax defines three types of **connectors**, which specify how definitions relate to each other. For example, so far you've seen the comma (,) which specifies that items must follow each other, in order. You can also use the pipe connector (|) to specify choices:

```
start =
  element memories {
    element memory {
      attribute tapeid { xsd:integer },
      element media {
        attribute mediaid { xsd:integer },
        attribute status {
          "8mm" | "vhs" | "vhsc" | "digital" | "audio"
        }
      },
      element subdate { xsd:date },
      element donor { text },
      element subject { text },
      element location {
        element description { text }
        | element place { text }
      }
    }
  }
```

Another connector is the interleave connector, the ampersand (&), which enables the user to add information in any order. For example:

```
start =
 element memories {
   element memory {
     attribute tapeid { xsd:integer },
     (element media {
       attribute mediaid { xsd:integer },
       attribute status {
         "8mm" | "vhs" | "vhsc" | "digital" | "audio"
       }
     }&
     element subdate { xsd:date } &
     element donor { text } &
     element subject { text } &
     element location {
       element description { text }
       | element place { text }
     })
   }
 }
```

Note that if you're using the interleave connector, you must enclose the interleaved elements in parentheses.

Modifiers

To include multiple instances of an element, you can use the same modifiers you would normally use with a DTD. For example, to allow one or more memory elements, use the + modifier:

```
start =
  element memories {
    element memory {
      attribute tapeid { xsd:integer },
      element media {
        attribute mediaid { xsd:integer },
        attribute status {
          "8mm" | "vhs" | "vhsc" | "digital" | "audio"
        }
      },
      element subdate { xsd:date },
      element donor { text },
      element subject { text },
      element location {
        element description { text }
        | element place { text }
      }
    }
+
  }
```

You can use the following modifiers:

+ : One or more

* : Zero or more

? : Zero or one

Parameters

You can also add parameters to a specification, just as you could with the XML syntax. In the compact syntax, you add it directly to the definition:

```
start =
  element memories {
    element memory {
      attribute tapeid {
        xsd:integer
{ minInclusive = "1" maxInclusive = "1000000" }
      },
      element media {
        attribute mediaid { xsd:integer },
        attribute status {
          "8mm" | "vhs" | "vhsc" | "digital" | "audio"
        }
      },
      element subdate { xsd:date },
      element donor { text },
      element subject { text },
      element location {
        element description { text }
        | element place { text }
      }
    }+
  }
```

In this way, you can add any necessary parameters.

Modularizing the definitions

Modularizing RELAX NG schemas in the compact syntax looks a lot like defining a DTD. For example, converting the version of the schema in Referencing internal definitions to compact syntax produces this:

```
start = element memories { memoryType+ }

memoryType =
  element memory {
    attribute tapeid {
      xsd:integer { minInclusive = "1" maxInclusive = "1000000" }
    },
    mediaType,
    submissionInfo,
    subjectType,
    locationType
  }

mediaType =
  element media {
    attribute mediaid { xsd:integer },
    attribute status { "8mm" | "vhs" | "vhsc" | "digital" | "audio" }
  }

submissionInfo =
  element subdate { xsd:date },
  element donor { text }

subjectType =
  element subject {
    mixed {
      element b { text }?,
```

```
      element i { text }?
    }
  }

locationType =
  element location {
    element description { text }
    | element place { text }
  }
```

Notice the use of modifiers within the individual definitions to specify not only multiple items, but also optional items.

External files

Compact syntax also enables you to reference external files. For example, the version of the schema in Overriding external definitions translates to:

```
                    include "subjectDef.rnc" {
                     subjectType = element subject { text }
                   }

start = element memories { memoryType+ }

memoryType =
  element memory {
    attribute tapeid {
      xsd:integer { minInclusive = "1" maxInclusive = "1000000" }
    },
    mediaType,
    submissionInfo,
    subjectType,
    locationType
  }

mediaType =
  element media {
    attribute mediaid { xsd:integer },
    attribute status { "8mm" | "vhs" | "vhsc" | "digital" | "audio" }
  }

submissionInfo =
  element subdate { xsd:date },
  element donor { text }

locationType =
  element location {
    element description { text }
    | element place { text }
  }
```

Note that `subjectDef.rnc` becomes a single definition:

```
subjectType =
  element subject {
    mixed {
      element b { text }?,
      element i { text }?
    }
  }
```

Section 8. Summary

Tutorial summary

This tutorial has described the structures and capabilities behind RELAX NG schema files. These schemas provide most of the functionality of W3C XML Schema without the inherent complexity. I have covered the main concepts necessary for using RELAX NG, including elements, attributes, data types, optional items, and modularizing. You should now have a solid base from which to begin exploring and using this powerful new XML-based schema language.

Resources

Learn

- Find links to the specifications for RELAX NG as well as various implementations at the main RELAX NG page (http://www.relaxng.org/).
- Read the specifications for RELAX NG's XML syntax (http://www.relaxng.org/spec-20011203.html) or compact syntax (http://www.relaxng.org/compact-20021121.html).
- Get guidance on using W3C XML Schema datatypes with RELAX NG (http://www.relaxng.org/xsd-20010907.html).
- The ZVON RELAX NG reference is a useful Javadoc-esque look at the various elements in a RELAX NG schema (http://zvon.org/xxl/RelaxNG/Output/index.html).
- James Clark, who created TREX, which merged with RELAX to become RELAX NG, has written about The Design of RELAX NG (http://www.thaiopensource.com/relaxng/design.html).
- Check out David Mertz's three-part series for his "XML Matters" column, "Kicking back with RELAX NG." Part 1 (http://www.ibm.com/developerworks/xml/library/x-matters25.html) looks at the general semantics of RELAX NG, and touches on datatyping. Part 2 (http://www.ibm.com/developerworks/xml/library/x-matters26.html) addresses additional semantic issues and looks at tools for working with RELAX NG. Part 3 (http://www.ibm.com/developerworks/xml/library/x-matters27.html) looks at tools for working with and transforming between the compact and XML syntax forms.
- Find more XML resources on the *developerWorks* XML zone (http://www.ibm.com/developerworks/xml/).
- Find out how you can become an IBM Certified Developer in XML and related technologies (http://www-1.ibm.com/certify/certs/adcdxmlrt.shtml).
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- RELAX NG-related tools are available in multiple programming languages:
  - **Java technology:** Jing (http://www.thaiopensource.com/relaxng/jing.html) and MSV (http://wwws.sun.com/software/xml/developers/multischema/) are RELAX NG validators. Trang (http://thaiopensource.com/relaxng/trang.html) converts W3C XML Schemas and DTDs to RELAX NG documents, and converts between XML and compact syntax within RELAX NG. Relaxer creates Java classes from RELAX NG schemas (http://www.relaxer.org).
  - **C:** Libxml2 supports validation against RELAX NG XML schemas (http://www.xmlsoft.org/).

    - **.NET:** Tenuto (http://sourceforge.net/projects/relaxng) is a RELAX NG validator implemented in C#.
    - **Visual Basic:** VBRELAXNG is an ActiveX DLL for validating RELAX NG (http://www.geocities.co.jp/SiliconValley-Bay/4639/vbrelaxng/vbrelaxng.html).

- Build your next development project with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.

About the author

Nicholas Chase
Nicholas Chase, a Studio B author, has been involved in Web site development for companies such as Lucent Technologies, Sun Microsystems, Oracle, and the Tampa Bay Buccaneers. Nick has been a high school physics teacher, a low-level radioactive waste facility manager, an online science fiction magazine editor, a multimedia engineer, and an Oracle instructor. More recently, he was the Chief Technology Officer of Site Dynamics Interactive Communications in Clearwater, Florida, USA, and is the author of several books on Web development, including *XML Primer Plus* (*Sams*). He loves to hear from readers and can be reached at nicholas@nicholaschase.com.