PB138 — Relax NG

What is Relax NG

- Specified by OASIS under OASIS RELAX NG TC
- "Relax New Generation" modeling language for XML
- Simpler and easier to use than XML Schema
- Based on **Relax** (*REgular LAnguage for XML*) and
- **TREX** (*Tree Regular Expressions for XML* by James Clark)

What is Relax NG (2)

- Having both XML and **non-XML syntax** improving readability
- published first in 2003
- later became ISO/IEC 19757 standard
- uses `.rng` and `.rnc` file extensions

Motivation

XML Schema is an industry standard but:

- too complicated (more than 200 pages of specification)
- ambiguous in some situations
- tries to cover all applications area (documents, databases and all in between)
- hardly fully implemented
- see http://www.xml.com/lpt/a/2002/01/23/relaxng.html for more

More on Relax NG advantages

- is simple
- is easy to learn
- has both an XML syntax (`.rng` files) or a **compact non-XML** syntax (`.rnc` files)
- does not change the **information set** of an XML document
- supports XML **namespaces**
- treats **attributes uniformly** with elements so far as possible
- has unrestricted support for **unordered content**
- has unrestricted support for **mixed content**
- has a solid theoretical basis
- can partner with a separate **datatyping** language (such W3C XML Schema Datatypes)

Specifications, Resources

Based on RELAX designed by OASIS-OPEN:

- http://www.oasis-open.org/committees/relax-ng

Now ISO standard:

- the ISO/IEC 19757 standard can be downloaded

RELAX NG home page (Key resource to Relax NG!):

- http://relaxng.org

RELAX NG — an excellent book by Eric van der Vlist:

- http://books.xmlschemata.org/relaxng/

Relax NG Basic Tools

Validators:

- online https://validator.nu/

Applications:

- Java-written Jing
- mainly for Linux is the C application `libxml2`
- RNV - supports the compact syntax only
- See http://relaxng.org/#validators for more.

Relax NG Editors etc.

Editors, other tools:

- Firedocs — a Firefox plug-in (XML editor)
- XML Operator — OSS (BSD Licence)
- XML editors supporting Relax NG such as <oXygen/> xml editor — commercial
- See Relax NG editors for more

Compact syntax

Following samples are from http://relaxng.org/compact-tutorial-20030326.html:

```
<addressBook>
   <card>
      <name>John Smith</name>
      <email>js@example.com</email>
   </card>
   <card>
      <name>Fred Bloggs</name>
      <email>fb@example.net</email>
   </card>
</addressBook>
```

DTD model

- The same model would be expressed in DTD as follows:

```
<!DOCTYPE addressBook [
<!ELEMENT addressBook (card*)>
<!ELEMENT card (name, email)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
]>
```

RELAX NG pattern

```
element addressBook {
   element card {
      element name { text },
      element email { text }
   }*
}
```

Cardinality

If the addressBook is **required to be non-empty**, then we can use + instead of *:

```
element addressBook {
   element card {
      element name { text },
      element email { text }
   }+
}
```

Optional

Now let's change it to allow each card to have an **optional** `note` element:

- Note that the text pattern matches arbitrary text, including empty text.
- Note also that whitespace separating tags is ignored when matching against a pattern.

```
element addressBook {
   element card {
      element name { text },
      element email { text },
      element note { text }?
   }*
}
# and this is a comment
```

Choice

If we want to have a choice of two (more) alternatives for one content model (eg. in `card`):

```
<addressBook>
   <card>
      <givenName>John</givenName>
      <familyName>Smith</familyName>
      <email>js@example.com</email>
   </card>
   <card>
      <name>Fred Bloggs</name>
      <email>fb@example.net</email>
   </card>
</addressBook>
```

Choice (2)

```
element addressBook {
   element card {
      # here is the option
      (element name { text }
         | (element givenName { text },
            element familyName { text })),
      element email { text },
      element note { text }?
   }*
}
```

Attributes

Specifies that addressBook has attributes instead of elements:

```
element addressBook {
   element card {
      attribute name { text },
      attribute email { text }
   }*
}
```

Alternatively attributes or elements

- The `,` and `|` connectors can combine element and attribute patterns without restriction.
- For example, the following pattern would allow a choice of elements and attributes independently for both the `name` and the `email` part of a `card`:

```
element addressBook {
   element card {
      (element name { text }
         | attribute name { text }),
         (element email { text }
            | attribute email { text })
   }*
}
```

Then all alternatives are valid

```
<card name="John Smith" email="js@example.com"/>
<card email="js@example.com" name="John Smith"/>
<card email="js@example.com"><name>John Smith</name></card>
<card name="John Smith"><email>js@example.com</email></card>
<card><name>John Smith</name><email>js@example.com</email></card>
```

Defining a grammar

```
grammar {
   start = AddressBook
   AddressBook = element addressBook { Card* }
   Card = element card { Name, Email }
   Name = element name { text }
   Email = element email { text }
}
```

- Uppercase names are *type names*, while
- lowercase names are *element names* in this example.
- Elsewhere it could be different but still there can be types and concrete occurencies.

Recursion

- The "inline" content is defined recursively
- This is, of course, allowed for elements only (attributes cannot be nested).

```
inline =
   (text
      | element bold { inline }
      | element italic { inline }
      | element span {
         attribute style { text }?,
         # here is the recursion
         inline
         })*
```

Datatyping

- Relax NG allows the use of external typing systems, such as the one from XML Schema W3C XML Schema Datatypes.

```
element number { xsd:integer }
element numberWithNote {
   xsd:int,
   attribute note { text }
}
element email {
   xsd:string { minLength = "6" maxLength = "127" }
}
```

Enumerations for attributes

```
element card {
   attribute name { text },
   attribute email { text },
   attribute preferredFormat { "html" | "text" }
}
```

Enumerations for text content

```
element card {
   element name { text },
   element email { text },
   element preferredFormat { "html" | "text" }
}
```

Lists

- lists of primitive values separated by white space(s)
- the cardinality of occurencies can be specified

```
element list_of_two_floats {
   list { xsd:float, xsd:float }
}
element list_of_some_doubles {
   list { xsd:double+ }
}
element list_of_some_double_pairs {
   list { (xsd:double, xsd:double)+ }
}
```

Interleave

- for specification of occurence in any order
- example: `name`, `email`

```
element addressBook {
   element card {
      element name { text }
      & element email { text }
      }*
   }
```

Modularity

- we can (re)use some external `.rnc` schema

```
start = inline
inline =
(text
   | element code { inline }
   | element em { inline }
   # etc
)*
```

Modularity (2)

Then we could allow the note element to contain inline HTML markup by using external as follows:

```
element addressBook {
   element card {
      element name { text },
      element email { text },
      element note { external "inline.rnc" }?
   }*
}
```

Namespaces

- prefixed or default namespaces can be used

```
namespace eg = "http://www.example.com"
element eg:foo { empty }
```

```
namespace = "http://www.example.com"
element foo { empty }
```