

# RELAXING: DTDs ON WARP DRIVE

John Cowan

Reuters

# Copyright

- Copyright © 2003-04 John Cowan
- Licensed under the GNU General Public License
- ABSOLUTELY NO WARRANTIES; USE AT YOUR OWN RISK
- Portions written by James Clark; used by permission
- Black and white for readability
- The Gentium font available at  
*<http://www.sil.org/~gaultney/gentium>*

# Abstract

In this tutorial you will learn how to use the RELAX NG schema language, an alternative schema language for XML.

RELAX NG allows easy and intuitive descriptions of just what is and what is not allowed in an XML document. It is simple enough to learn in a few hours, and rich and flexible enough to support the design and validation of every kind of document from the very simple to the very complex. Once RELAX NG's concepts have crossed the blood-brain barrier, you will never be able to take any other schema language very seriously again.

# Roadmap

- RELAX NG goals (21 slides)
- The invoice example (9 slides)
- Patterns (22 slides)
- Namespace Routing Language (4 slides)
- Datatypes (12 slides)
- The XLink example (12 slides)
- Tools (14 slides)
- XSD datatypes (16 slides)

# RELAX NG GOALS

# DTDs On Warp Drive

- An evolution/generalization of DTDs
- Shares the same basic paradigm
- Based on experience with SGML, XML
- Adds and subtracts features from DTDs
- DTDs can be automatically converted

# Reusable Knowledge

- Experts in designing SGML and XML DTDs will find their skills transfer easily
- Design patterns commonly used in XML DTDs can be reused
- Much more mature than if based on a completely new and different paradigm
- Higher degree of confidence in its design is possible

# Easy To Learn And Use

- Allows schemas to be patterned after the structure of the documents they describe
- Allows definitions to be composed from other definitions in a variety of ways
- Treats attributes and elements as uniformly as possible

# Namespaces

- DTDs are namespace-blind
- RELAX NG fully supports namespaces for elements and attributes
- Namespace support is purely syntactic, not tied to one schema per namespace
- Name classes support “any name” and “any name in specified namespace”

# Datotyping

- Supports pluggable simple datatype libraries
- Basic library supports strings and tokens
- Full XML Schema Part 2 datatypes available (including facets)
- New libraries can be readily designed and built as needed.

# Composability

- Schema languages provide atomic objects (elements, attributes, text, typed data) and methods of composing them (sequence, repetition, choice)
- All RELAX NG atomic objects can be composed with any available method
- Improves ease of learning, use, power; decreases complexity

# Closure

- RELAX NG is closed under union
  - If two schemas exist describing two document types, then a schema describing the union of the two document types is trivial to create
  - Consequently, the content model of an element can be context-dependent

# Two Syntaxes, One Language

- Provides two interconvertible syntaxes:
  - an XML one for processing
  - a compact non-XML one for human authoring
- We will learn the compact syntax
- One example of the XML syntax is provided to assist in learning it

# Attributes

- “Elements or attributes?”
  - Reasonable people can differ
  - Attributes are treated as much like elements as possible
- Content models include elements as well as attributes
- Attribute defaulting is not done

# Non-goal: Attribute Defaulting

- Attribute defaulting can only be done by DTDs or W3C XML Schema when the value does not depend on context
- Sensible attribute defaults often depend on context (inheritance of `xml:lang`, e.g.)
- Attribute defaulting is trivial for transformation languages such as XSLT

# Non-goal: PSVI

- RELAX NG has no post-validation infoset enhancement
- Infoset enhancement can be done as a separate layer
- Sun's Multi-Schema Validator provides datatype information
- Separation of concerns promotes efficiency, flexibility

# Annotations

- Annotations in the form of elements and attributes can be interspersed in RELAX NG schemas for various purposes:
  - DTD-style attribute defaults
  - documentation
  - embedded Java code
- Conforming RELAX NG validators ignore annotations

# Mixed Content

- SGML had problems with complex mixed-content models
- XML DTDs tightly restrict mixed-content models
- RELAX NG allows character content mixed with any content model

# Unordered Content

- SGML's & operator allows unordered content models
  - A & B means ((A, B) | (B, A))
- XML DTDs removed & to reduce implementation complexity
- RELAX NG restores & with improvements

# Customization

- Definitions included from another schema can be overridden
- Multiple definitions from the same or different schemas can be intelligently combined
  - as if with |
  - as if with &

# A Real Standard

- Being standardized in OASIS by the RELAX NG Technical Committee
- A major component of ISO DSDL, the new Document Schema Definition Languages umbrella-standard

# Non-goal: Inheritance

- Inheritance-based schemas only model single inheritance
- Modeling often requires multiple inheritance (at least for interfaces)
- Schema languages are about syntactic details, not about models

# Non-goal: Identity Constraints

- Identity constraints are not supported
- Identity constraints are still a developing research area
- Different applications have different requirements from simple to complex
- Some RELAX NG tools support DTD-style semantics for ID, IDREF(S)

# Non-goal: Schema Binding

- There is no standard way for a document to specify “its schema”
- Receivers often want to verify against agreed-on schemas, not sender-specified ones
- Documents may be validated against different schemas for different purposes
- The validation model takes two inputs: a document and a schema
- Just part of the XML processing issue

# Interoperability

- You can convert a DTD to RELAX NG, preserving modularity
- You can author in RELAX NG and deliver in DTD or W3C XML Schema or both
- RELAX NG allows embedded Schematron rules

# Pronunciation

- "Relaxing" is the standard way
- Some people say "relax en gee"

AND NOW TO RELAX!



# THE INVOICE EXAMPLE

# An Invoice in XML

```
<invoice number="640959-0" date="2002-03-12">
  <soldTo>
    <name>Reuters Health Information</name>
    <address>45 West 36th St. New York NY 10018</address>
  </soldTo>
  <shipTo>
    <name>Reuters Health Information</name>
    <address>45 West 36th St. New York NY 10018</address>
  </shipTo>
  <terms>Net 10 days</terms>
  <item ordered="6" shipped="6" unitPrice="7.812">
    Binder, D-ring, 1.5"</item>
  <item ordered="4" shipped="2" backOrdered="2"
    unitPrice="3.44">Fork, Plastic, Heavy, Medium</item>
</invoice>
```

# The Invoice Schema (1)

```
element invoice {  
  attribute number { text },  
  attribute date { text },  
  element soldTo {  
    element name { text },  
    element address { text }  
  },  
  element shipTo {  
    element name { text },  
    element address { text }  
  },  
}
```

# The Invoice Schema (2)

```
element terms { text },
element item {
  attribute unitPrice { text },
  attribute ordered { text },
  attribute shipped { text },
  attribute backOrdered { text }?,
  text
}*
}
```

# The XML format

```
<element name="invoice">
  <attribute name="number"/>
  <attribute name="date"/>
  <element name="soldTo">
    <element name="name">
      <text/>
    </element>
    <element name="address">
      <text/>
    </element>
  </element>
  <element name="shipTo">
    <element name="name">
      <text/>
    </element>
    <element name="address">
      <text/>
    </element>
  </element>
</element>
```

```
<element name="terms">
  <text/>
</element>
<zeroOrMore>
  <element name="item">
    <attribute name=
"unitPrice"/>
    <attribute
name="ordered"/>
    <attribute
name="shipped"/>
    <optional>
      <attribute
name="backOrdered"/>
    </optional>
  </element>
</zeroOrMore>
</element>
```

# Things To Note

- The structure of the schema parallels the structure of the document
- Element content models include attributes as well as child elements
- The optional attribute is marked with ?
- "text" is the equivalent of #PCDATA or CDATA

# Things To Note

- Commas separate multiple components of a content model when the components appear in the given order
- Of course, the order of attributes does not matter!
- Consequently, attributes can appear in the schema before, after, or mixed in with child elements

# Definition Form

```
start = element invoice {
  attribute number { text },
  attribute date { text },
  element soldTo { name-addr },
  element shipTo { name-addr },
  element terms { text }
  element item {
    attribute ordered { text },
    attribute shipped { text },
    attribute backOrdered { text }?,
    attribute unitPrice,
    text}*
  }
}
```

name-addr =

```
  element name { text },
```

```
  element address { text }
```

Copyright 2003-04 John Cowan under the GNU GPL

# Definition Form Notes

- In definition form, there must always be a definition of `start`
- You refer to a rule using just its name
- The order of the rules does not matter; use whatever order makes sense to you (top-down, bottom-up, alphabetical)
- Rule names are only relevant to the schema, and *never* appear in the document instance

# DTD-Style Definitions

```
start = element invoice {  
  attribute number { text },  
  attribute date { text },  
  soldTo, shipTo, terms, item  
}
```

```
soldTo = element soldTo { name-addr }
```

```
shipTo = element shipTo { name-addr }
```

```
terms = element terms { text }
```

```
item = element item { ... }
```

```
name-addr = name, address
```

```
name = element name { text }
```

```
address = element address { text }
```

# PATTERNS

# Patterns

- Patterns are the basic building blocks of RELAX NG schemas and rules
- Some kinds of patterns can contain sub-patterns enclosed in braces ( { ... } )

# Element Patterns

- The form is: `element name { ... }`
- The content model (child elements and attributes) is contained within the braces
- Content models consist of one or more patterns

# Attribute Patterns

- The form is: `attribute name { ... }`
- The content model is contained within the braces
- Content models consist of one or more patterns
- You can't have child elements or attributes within attributes, of course!

# Attribute Patterns

- So what patterns can be inside attributes?
  - The `text` pattern - equivalent to `CDATA`
  - Datatypes (next section)
  - Literal strings in quotes:  
`attribute country { "US" }`  
means the `country` attribute must have the value `US`.

# Element Patterns

- So what patterns can be inside elements?
  - The `text` pattern - equivalent to `#PCDATA`
  - Datatypes (next section)
  - Literal strings in quotes:  
`element country { "US" }`  
means the `country` element must have the content `US`.

# The Text Pattern

- Matches any amount of arbitrary text, possibly broken up by child elements
- Equivalent to #PCDATA in elements or CDATA in attributes
- `text*`, `text?`, `text+` all mean the same as `text`

# Namespaces

- To declare elements and attributes in namespaces, use QName in element and attribute patterns
- Namespace prefixes are declared like this:  
`namespace foo = "(some URI)"`
- Namespace declarations must come first in the schema

# Default Namespaces

- You can declare a namespace for unprefixed elements (not attributes) like this:  
default namespace =  
    "(some URI)"
- If you want the default namespace to have a prefix too, use:  
default namespace foo =  
    " (some URI)"

# Namespaces

Here's an example:

```
namespace one = "http://example.com/one"
namespace two = "http://example.com/two"
default namespace = "http://example.com"
element para {
  attribute one:class { text },
  attribute two:class { text },
  element line { text }*
}
```

# Choice

- Two patterns separated by | represent a choice between them; the document can match one pattern or the other, not both
- Arbitrary patterns are allowed in a choice: you can have a choice between attributes, between elements, or even between an element and an attribute!

# Choice

- A useful case:

```
element data {  
    (element id { text } |  
     attribute id { text } ),  
    text  
}
```

- You cannot mix , and | in one list; use parentheses to disambiguate

# Choice

Enumerated values use choice like this:

```
element font {  
    attribute size {  
        "10" | "12" | "14" | "16"  
    }  
}
```

# Interleave

- Interleave is a cross between choice and sequence
- When patterns are combined with &, they all must appear but it can be in any order (as in SGML) ...
- ... or even mixed together!

# Interleave

- So this schema ...

```
element head {  
  element meta { empty }* &  
  element title { text }  
}
```

- ... matches a head element that has any number of meta child elements (including zero) and a required title child element *mixed in anywhere*.

# Interleave

- Note: In the case of attributes, sequence and interleave are the same thing, because attributes don't have ordering
- So you can use either , or & according to what is the most convenient

# Quantifiers

- You can place an `*`, `?`, or `+` after any pattern to allow it to be repeated:
  - `*` means zero or more times
  - `?` means zero or one times
  - `+` means one or more times
- These mean the same as in DTD content models, but can be used after any pattern, not just rule names

# ANY?

- There is no built-in ANY content model, corresponding to `empty` for empty content models or `notAllowed` for forbidden models

- Here's how it can be done:

```
ANY = element * {  
    attribute * {text} *  
    & text & ANY*  
}
```

# Multiple Schemas

- `external` incorporates one pattern document into another
- `include` incorporates one set of rules into another, and allows for overriding any of the included rules by name
- Rules with identical names can also be combined by choice or by interleave

# Context Sensitivity

The first paragraph cannot have footnotes;  
the remainder can:

```
start = element doc {first, other*}  
first = element para { text }  
other = element para {  
  mixed { element footnote {text}* }  
}
```

# Restrictions on Schemas

- The obvious XML ones: no elements or attributes within attributes, only one top-level element, etc. etc.
- Attributes can't have conflicting definitions in a single element
- Wildcard attributes must use \* or +
- Lists of lists and lists of text don't work

# Restrictions on Schemas

- Elements can't group or interleave simple content (data, value, or list) with complex content (child elements and/or text)
- Interleave doesn't allow an element with a given name to be used in more than one subpattern
- Interleave doesn't allow text in more than one subpattern

# Comments

- Ordinary comments begin with #
- Documentation comments begin with ## and are copied (in groups) into the XML syntax as `a:documentation` elements
- The `a:` prefix represents the namespace of the DTD Compatibility extension to RELAX NG

# NAMESPACE ROUTING LANGUAGE

# What it is

- Not a standard yet, but some version of it will become one
- An NRL schema specifies how to apply multiple subschemas to different parts of a document
- Basically it's a map between namespace names and schema URIs

# How it works

- The document is carved up into subtrees whose elements have a common namespace name
- The subschema associated with that name is used to validate the subtree
- Subschemas can be in any schema language including XML Schema and even DTD

# Concurrent validation

- Multiple schemas can be applied to a subtree to cause concurrent validation
- Example: XHTML validation with RELAX NG requires a check that no `a` element has another `a` element as its ancestor
- A small schema that must *fail* validation can be used to enforce this limitation

# Other features

- A subordinate namespace can be attached to a superior namespace so that they will be validated together by the superior namespace's schema
- Attribute-only schemas can be supported using a dummy element name
- Validation can begin at a specific element specified by a simple form of XPath

# DATATYPES

# Datatypes

- A type is a named set of values
- An datatype provides a standardized, machine-checkable representation of a type

# Schema Datatypes

- DTDs have only a few datatypes for attributes and only one datatype for elements
- XML Schema provides a long, but fixed, list of datatypes
- RELAX NG can work with any datatype library, including the XSD (XML Schema Datatypes) library

# RELAX NG Datatypes

- Datatype patterns are written using QNames
- This use of QNames can't be confused with QNames for elements or attributes, because those are only recognized after the words `element` and `attribute`
- The built-in datatypes `string` and `token` don't have prefixes and are recognized by all implementations

# Declaring Datatype Libraries

- A prefix is declared like this:  
`datatypes lib = "(some URI)"`
- Datatype library declarations must come first
- RELAX NG processors recognize a system-dependent list of datatype library URIs

# Useful Datatypes

- The `xsd` prefix is predeclared for XML Schema Datatypes
- `xsd:integer` represents an integer of arbitrary length
- `xsd:ID`, `IDREF`, and `IDREFS` are the same as the corresponding DTD types
- We'll discuss all the `xsd` types later.

# Typed Values

- `"0"` and token `"0"` match a “0” character with possible surrounding whitespace
- `string "0"` matches a “0” character exactly
- `xsd:integer "0"` matches “0” or “00” or “000” or “-0” or ...

# Mixing Datatypes

- Datatypes can't be mixed with anything in the same content model, as in:

```
element foo {                                ## ERROR!  
    xsd:int, xsd:ID,  
    element bar { empty }  
}
```

- Character content that represents a datatype cannot have any siblings

# Mixing Datatypes

- Choices involving datatypes are fine:

```
element foo { xsd:int | xsd: Name }
```

```
element bar { xsd:int |  
              element baz { text } }
```

# Datatype Exceptions

- `xsd:nonNegativeInteger` and `xsd:nonPositiveInteger` are existing types
- How do we say “non-zero integer”?  
`xsd:integer - xsd:integer "0"`
- We can likewise express a token that is not a name:  
`xsd:token - xsd:Name`

# Parameters

- Parameters restrict the values of datatypes
- Each datatype has specific parameters are legal with it
- An integer between 0 and 999 inclusive:

```
xsd:integer {  
    minInclusive = "0"  
    maxInclusive = "999"  
}
```

# Lists

- List patterns specify that character content or an attribute value is to be separated by whitespace into tokens
- The pattern `list {xsd:integer*}` matches a list of zero or more whitespace-separated integers
- This example needs `*` because `list` itself does not imply repetition

# Lists

- The pattern

```
list { (xsd:integer, token)+ }
```

matches the string

```
"32 foo 45 bar 76 baz"
```

# THE XLINK EXAMPLE

# XLink

- A W3C Recommendation for general hyperlinking in XML documents
- Simple links are like HTML `a` or `img` elements
- Extended links specify multiple endpoints, roles, and traversal paths
- XLink is all done by attributes: any element can function in any role

# The XLink Namespace

```
namespace xlink =
```

```
"http://www.w3.org/1999/xlink"
```

# XLink Attribute Rules

href = attribute xlink:href {xsd:anyURI}

role = attribute xlink:role {xsd:anyURI}

arcrole = attribute xlink:arcrole  
{xsd:anyURI}

title.att = attribute xlink:title {text}

label = attribute xlink:label {xsd:NCName}

from = attribute xlink:from {xsd: NCName}

to = attribute xlink:to {xsd: NCName}

# XLink Attribute Rules

```
show = attribute xlink:show  
  {"new" | "replace" | "embed" | "other" |  
  "none" }
```

```
actuate = attribute xlink:actuate  
  {"onLoad" | "onRequest" | "other" |  
  "none" }
```

# Simple Links

```
simple = element * {  
  attribute xlink:type {"simple"},  
  ref?, role?, arcrole?, title.att?,  
  show?, actuate?, anyAttr*,  
  (anyElem | text)*  
}
```

# Extended Links

```
extended = element * {  
  attribute xlink:type {"extended"},  
  role?, title.att?, anyAttr*,  
  (title | resource | locator  
  | arc | anyElem | text)*  
}
```

# Title Elements

Description of link or endpoint:

```
title = element * {  
    attribute xlink:type {"title"},  
    anyAttr*, (anyElem | text)*  
}
```

# Resource Elements

Local endpoints of an extended link:

```
resource = element * {  
  attribute xlink:type {"resource"},  
  role?, title.att?, label?,  
  anyAttr*, (anyElem | text)*  
}
```

# Locator Elements

Remote endpoints of an extended link:

```
locator = element * {  
  attribute xlink:type {"locator"},  
  href, role?, title.att?, label?,  
  anyAttr*, (title | anyElem | text)*  
}
```

# Arc Elements

Traversal paths through an extended link:

```
arc = element * {  
    attribute xlink:type {"arc"},  
    arcrole?, title.att?, show?,  
    actuate?, from?, to?, anyAttr*,  
    (title | anyElem | text)*  
}
```

# Non-XMLink Elements and Attributes

```
anyElem = element * {  
    anyAttr*,  
    (anyElem | text)*  
}
```

```
anyAttr = attribute * - xlink:*  
    {text}
```

# The Whole Document

```
start = element * { anyAttr*,  
  (simple | extended | anyElem)* }
```

# TOOLS

# The Jing validator

- Written by James Clark, principal author of RELAX NG
- Java based command-line tool
  - Validates schemas
  - Validates documents against schemas
- Accepts either compact or XML syntax
- Optionally enforces DTD ID/IDREF

# The Jing validator

- Also usable as a validation library within a Java program
- Provides JAXP (Sun-standard) interface
- Provides native interface
- Validates against other schema languages:
  - W3C XML Schema
  - Schematron
  - Namespace Routing Language (meta)

# The Trang translator

- Another James Clark product
- Translates schemas:
  - Input: XML syntax, compact syntax, DTDs
  - Output: XML syntax, compact syntax, DTDs, W3C XML Schema
- Output schemas may be looser than the input schema (accept a superset of what the input accepts)

# Sun RELAX NG translator

- Translates other schema languages into XML syntax:
  - DTDs
  - RELAX Core/Namespaces
  - TREX (predecessor of RELAX NG)
  - Subset of XML Schema
- Does not preserve schema structure

# Instance to schema

- InstanceToSchema generates a RELAX NG schema from one or more XML instances
- Examplotron ([www.examplotron.org](http://www.examplotron.org)) is a schema language that resembles an instance with optional annotations and is translated into RELAX NG

# Sun Multi-Schema Validator

- By Kohsuke KAWAGUCHI, a major RELAX NG contributor
- Validates documents (command-line or library) using any schema language supported by the Sun RELAX NG Translator
- Also handles stand-alone or embedded Schematron rules

# Validation in .NET

- Tenuto is a C# implementation of validation for the Common Language Runtime environment
- Supports XML syntax, XSD library
- Does not support ID/IDREF semantics
- RelaxngValidatingReader is an unrelated implementation of XMLReader that validates input against a RELAX NG schema

# The Bali validator generator

- Accepts a RELAX NG schema and generates special-purpose code to validate documents against that particular schema
- The generated validator is faster and smaller than a general-purpose validator
- Generates Java, C++, or C#

# The VBRELAXNG validator

- A validator for the XML syntax written in Visual Basic 6.0
- Provides validation as an ActiveX control
- Requires MSXML 4.0 parser
- Topologi Schematron Validator uses this control

# The RelaxNGCC compiler

- Accepts a subset of RELAX NG (no ambiguous grammars) with annotations; RelaxMeter tool checks for ambiguities
- Compiles specified RELAX NG rules into Java classes
- Embedded Java code can refer to the values matched by datatype, text, and list patterns
- Analogous to JavaCC

# The RelaxNGCC compiler

- The generated code requires a source of SAX events (typically a parser)
- Objects of the generated classes are data bindings for RELAX NG rules considered as types
- The killer app for RELAX NG?

# The RELAXER compiler

- Generates Java objects from RELAX NG schemas without embedded code
- Imposes slightly different restrictions on schemas (weaker if DOM available)
- Provides serialization from Java as well as parsing to Java
- Analogous to JAXB
- The killer app for RELAX NG?

# Editors

- oXygen and Topologi editors validate against RELAX NG schemas
- xmloperator and an Emacs mode allow editing guided by a RELAX NG schema
- Vim syntax coloring for writing schemas in the compact syntax

# The libxml2 library

- Provides parsing and (RELAX NG) validation services for C programs
- Packaged with xmllint, an XML parser and RELAX NG validator
- Part of the GNOME desktop, but available separately

# XSD DATATYPES

# Mini-roadmap

- Types (11 slides)
- Facets (5 slides)

# XML Schema Datatypes

- A type is a named set of values
- An XML Schema datatype provides a standardized, machine-checkable representation of a type
- XML Schema types can be grouped:
  - numeric, date, boolean, string, misc.

# Numeric Types

- Decimal types
- Floating-point types

# Decimal Types

- decimal
  - integer
    - nonPositiveInteger
      - negativeInteger
    - nonNegativeInteger
      - positiveInteger
      - unsigned{Long, Int, Short, Byte}
    - long, int, short, byte

# Decimal Types

- `long`, `short`, `int`, and `byte` are the same as in Java: 64, 32, 16, 8 bits
- `unsignedLong`, `unsignedShort`, `unsignedInt`, and `unsignedByte` are the obvious unsigned analogues
- All other numeric types are unbounded

# Floating-point Types

- Only two floating-point types
  - float
  - double
- IEEE ranges (same as Java, all modern hardware)

# Date Types

- `duration`
- `date`, `time`, `dateTime`
- `gYear`, `gMonth`, `gDay`,  
`gYearMonth`, `gMonthDay`

# Date Types

- Duration
  - duration
- Single Time Interval
  - dateTime, date, gYear, gYearMonth
- Recurring Time Interval
  - time, gMonth, gDay, gMonthDay

# Date Type Examples

duration	P1D	PT30M	P2M
dateTime	2002-06-17T13:45:00		
Date	1776-07-04		
Time	17:05:00-05:00		
gYear	1984		
gMonth	--12		
gDay	---29		
gYearMonth	1917-11		
gMonthDay	--09-11		

# Boolean Type

- Only two values are legal:
  - `true` (which can also be written `1`)
  - `false` (which can also be written `0`)

# String Types

- `string`
  - `normalizedString`
    - `token`
      - `language`
      - `NMTOKEN(S)`
      - `Name`
        - » `NCName`
          - `ID, IDREF(S), ENTITY(IES)`

# Miscellaneous Types

- Raw octet types
  - `hexBinary`
  - `base64Binary`
- `anyURI`
- `QName`
- `NOTATION`

# Facets

- Allow the creation of new datatypes by restricting the existing ones in one or more ways
- Called `params` in RELAX NG
- Facets can be grouped into families applicable to datatype families:
  - length, value, pattern
  - enumeration, whiteSpace

# Length Facets

- Applicable to string and miscellaneous types
- `length` facet gives exact length
- `minLength` and `maxLength` facets set limits; either or both may be used
- lengths of `hexBinary` and `base64Binary` types are measured in octets, not characters

# Value Facets

- Applicable to numeric and date types
- `minExclusive` and `minInclusive` specify a lower bound; either but not both may be used
- `maxExclusive` and `maxInclusive` specify an upper bound; either but not both may be used

# Value Facets

- `totalDigits` specifies the total number of significant digits in a `decimal`, `integer`, `(non)PositiveInteger`, or `(non)NegativeInteger` value
- `fractionDigits` specifies the number of fractional digits in a `decimal` value

# Pattern Facet

- Applicable to any type
- Specifies a regular expression that the data must match
- XML Schema: If multiple `pattern` facets are present, the data must match at least one of them
- RELAX NG: If multiple `pattern` facets are present, the data must match all of them

# MORE INFORMATION

*<http://www.relaxng.org>*

*<http://www.ccil.org/~cowan/relaxng>  
{.ppt,.sxi,.pdf}*