Datatypes in XML

Jeni Tennison
www.jenitennison.com
jeni@jenitennison.com

Overview

- Context: DSDL Part 5
- What is data in XML?
- Why do we need to type data?
  - validation, documentation, authoring support, application support
- What kinds of data do we use?
  - survey from different markup languages
- How do we specify datatypes currently?
- Datatype Library Language
- Outstanding issues

DSDL

- ISO-standardised schema language(s)
- W3C XML Schema is for the big boys
  - want to do data binding into databases or programming languages
  - primarily data-oriented content
  - database vendors, web service providers
    - people who jumped on the XML bandwagon, and are distorting it to satisfy their requirements
- DSDL is for the rest of us
  - redresses the balance back towards document-oriented content
  - better for data-oriented content too!
- DSDL Part 5 addresses datatyping...

Data in XML

- Data appears in:
  - attribute values
  - content of text-only elements
  - mixed-content elements, more rarely
- Sequences of Unicode characters (strings)
  - includes control characters in XML 1.1
  - characters, not bytes
    - "255" is three characters, not one byte
- Whitespace weirdness
  - line endings normalised to #xA
  - whitespace in attributes replaced with spaces
  - whitespace chars can be escaped using entities

Data in XML

- Uses in *real* XML
  - human-readable rather than machine-readable
    - for presentation rather than processing
  - abbreviations and text-based formats
    - for ease of writing
    - to control the size of the XML
    - to reuse existing text-based formats
    - to enable reuse in e.g. URIs
  - datatypes rarely match programming language or database datatypes
    - care about characters, not bytes

Reasons for Datatyping

- Validation
  - is the value allowed?
  - does the supplied value equal the fixed/listed/key value?
- Equality testing isn't straightforward

```
<element name="example">
  <attribute name="quality">
    <choice>
      <value>good</value>
      <value>bad</value>
    </choice>
  </parse>
</element>
```

```
<example quality="Good" />

<example quality=" bad " />
```

Reasons for Datatyping

- Application support
  - use datatypes in XPath/XForms etc.
  - data binding
    - translate to appropriate datatype in programming language/database
- General comparisons, not just equality

```
<xsl:for-each select="Order[Total >= 1000]">
  <xsl:sort select="Date"
            data-type="xs:date" />
  …
</xsl:for-each>
```

Reasons for Datatyping

- Documentation
  - so users can understand what kind of value an element/attribute takes
  - so users can understand how they have to format that value
- Authoring support
  - enable applications to prompt user for values rather than using separate validation step
    - pop-ups for enumerated values
    - calendars for dates
    - sliders for numbers

Survey of Data

- Well-known markup languages
  - XML attributes
  - DocBook
  - XHTML
  - SVG
  - MathML
  - Dublin Core
  - XInclude
  - XSLT
  - XSL-FO
  - XML Schema
  - RELAX NG
  - XForms
- Designed by people who should know what they're doing, so bound to be good
  - or "designed by committee, so bound to be bad"
- Used extensively
- Hard to change

Standard Atomic Datatypes

- Strings/text
  - variable whitespace significance
  - variable case significance
  - limited/extended character sets
    - restricted characters in XInclude accept attribute
    - extended characters in MathML values
  - variable length (usually single characters)
- Numbers
  - integers (7)
  - decimals (7.5)
  - scientific format (7.5E3)
- Booleans
  - true/false, 1/0, yes/no

Enumerated Values

- Listings of possible values
  - xml:space - "preserve" | "default"
  - XInclude - "xml" | "text"
- May be case-insensitive
  - XHTML LinkTypes
- May be listed separately, accessible by URI
  - IANA registered media types
  - DCMI type vocabulary
  - ISO 15924 scripts
- May be part of structured value (see later)
  - xml:lang - language and country codes
- May be subset of allowed values
  - SVG color keywords
- Often have expansion/explanation/description

Lists

- Lists of values
  - whitespace-separated
    - most common
    - NMTOKENS, IDREFS
  - comma-separated
    - XHTML URI lists
    - XHTML media-type lists (as in CSS2)
  - whitespace-or-comma-separated
    - SVG lists
  - semi-colon-separated
    - Dublin Core Separated Values

Values with Units

- Number coupled with unit designator
  - lengths (36pt, 3px)
  - frequencies (5Hz, 16kHz)
  - angles (90deg)
  - durations (3s, 150ms)
  - proportions (5*)
  - percentages (25%)
- Some units are absolute, others relative (see later)

Simple Structured Values

- Values conforming to a regular grammar
  - describable using a regular expression
    - though perhaps not easily
  - many examples:
    - dates and times
    - URI references
    - colours in RGB notation
    - SVG path data
    - SVG transformations
    - MathML group alignment
    - XInclude accept, accept-language attributes
    - XPath 2.0 sequence types
    - XPath subsets (as in W3C XML Schema)
    - P3P type names (as in XForms)

Complex Structured Values

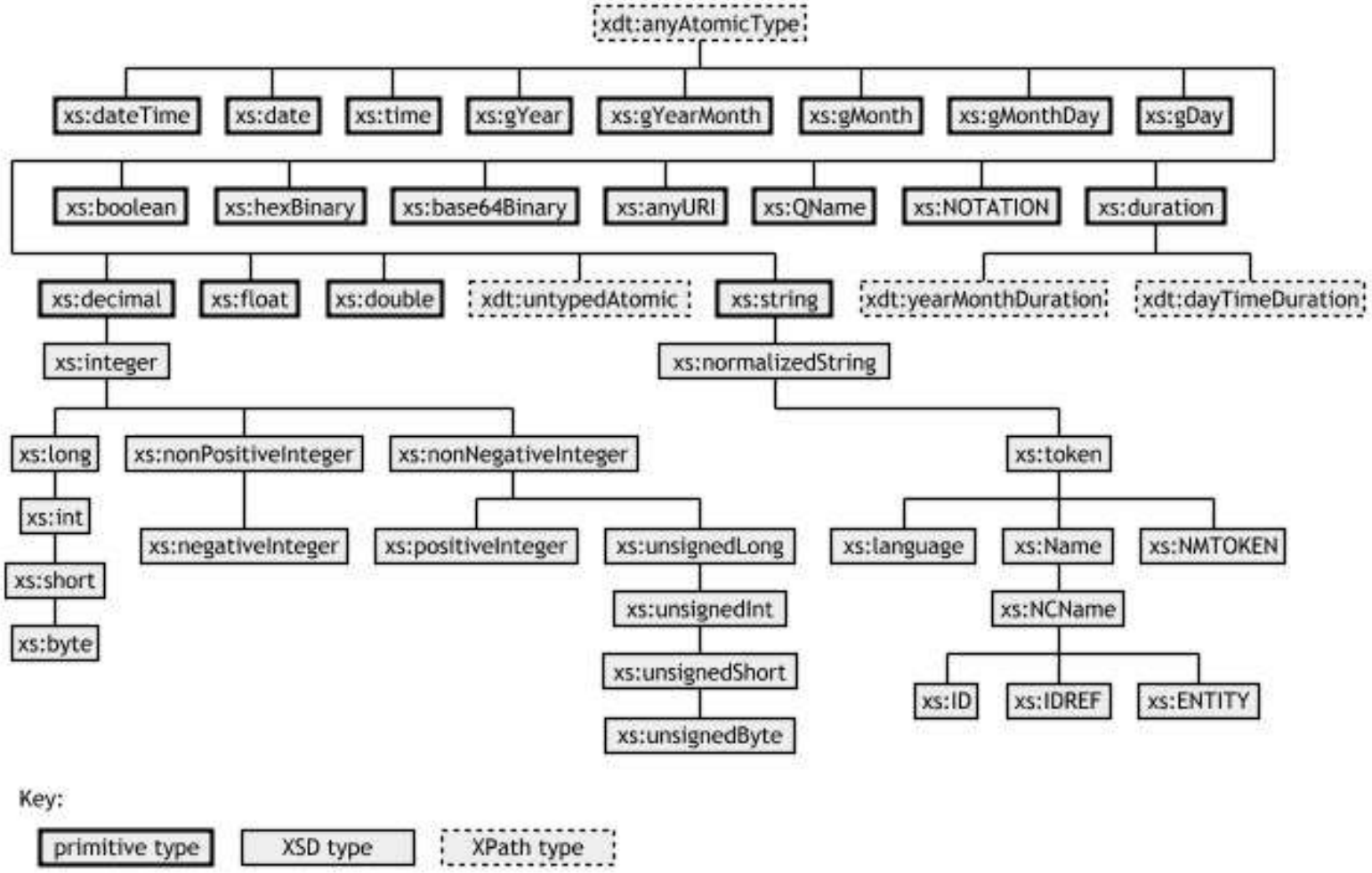
- Values not conforming to a regular grammar
  - XPointers
  - XPaths
  - XSLT patterns
  - XSL-FO expressions
  - regular expressions

Use of Context Information

- Validity/meaning of value depends on where it appears in XML document
  - XML (Infoset) context
    - qualified names
    - namespace prefixes
    - relative URIs
    - declared unparsed entities and notations
    - IDs and IDREFs
  - application context
    - lengths - ems and exs
    - proportions - percentages and *s
    - 'auto'/'inherit' in XSL-FO

Datatype Support in XML Schema

- XML Schema hierarchy + XPath 2.0 datatypes



Datatype Support in XML Schema

- Type has:
  - value space
  - lexical space
  - canonical lexical representation (usually)
- Union types
- Whitespace-separated lists
  - items all have to be the same type (though it can be a union type)
- Subtypes derived using facets
  - pattern facet controls lexical representation
  - other facets control value space

Problems with WXS Datatypes

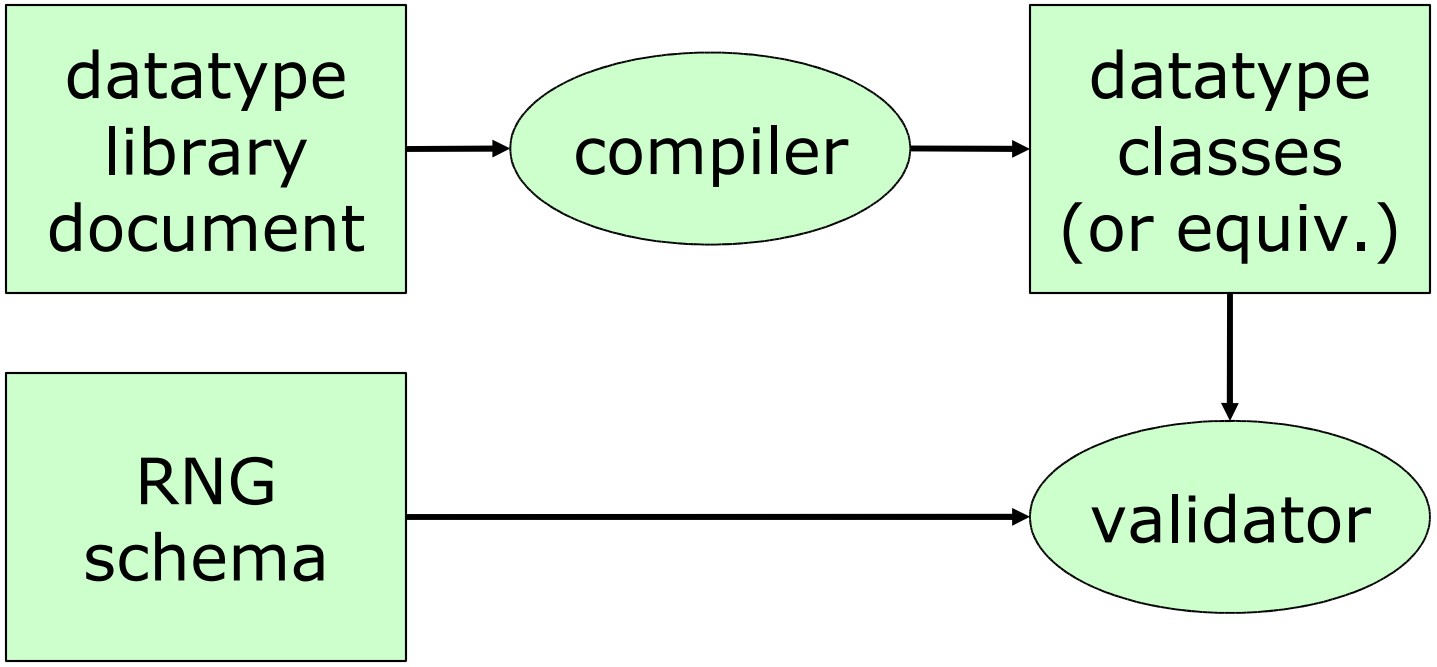
- No way to add primitive datatypes
  - can't represent colours, for example
- No way to change lexical space
  - can't have dates in format DD/MM/YYYY
- Possible for canonical lexical representation to be invalid
  - two-decimal places in price 12.50
  - causes problems with round-tripping
- Lists must be space separated, and items must be of the same type
- Can use regexes for lengths etc., but then comparisons are string comparisons

Datatype Support in RELAX NG

- Two basic types, string and token
  - differ in whitespace treatment
- Supports whitespace-separated lists
  - control over types of items
- Supports enumerated values
- Supports "except" and "choice"
- Datatype libraries can be used
  - test validity of value, and equality of values
  - pass in parameter values and context information
  - usually uses XML Schema datatype library

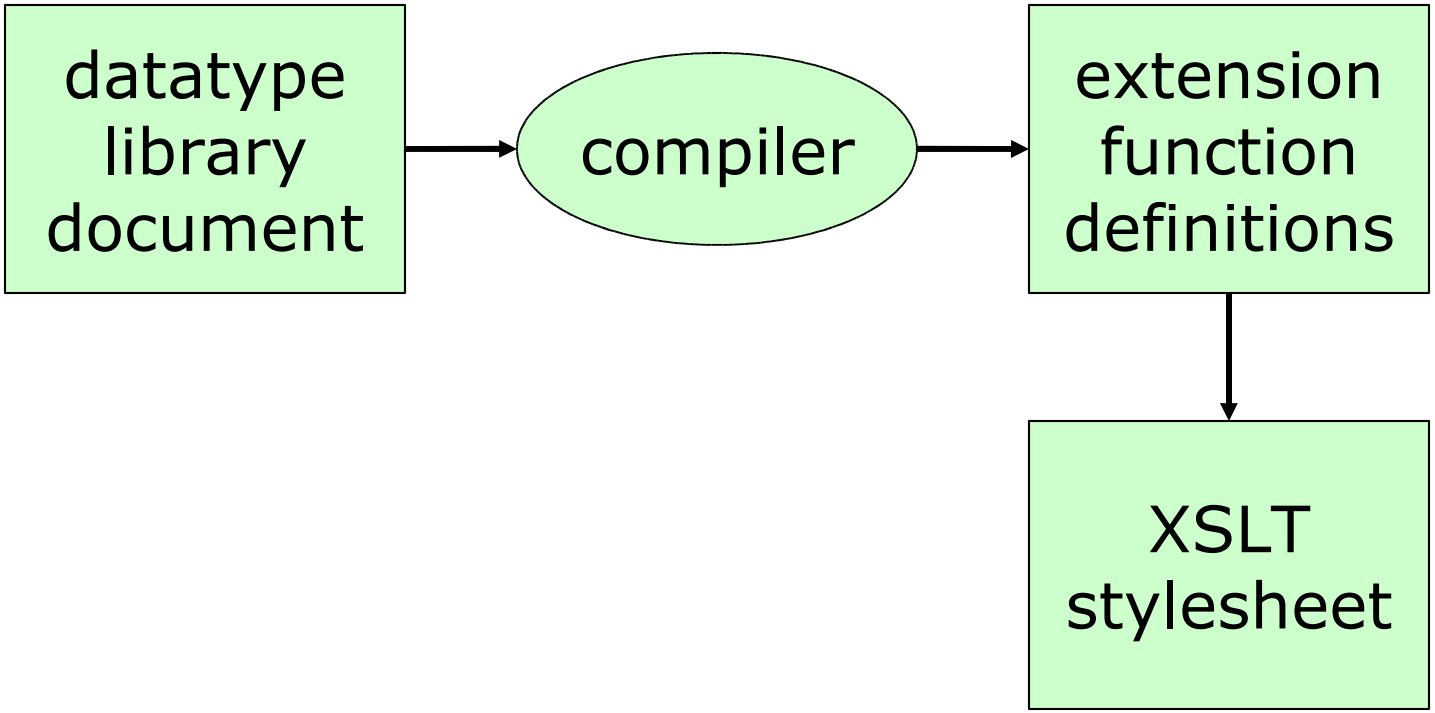
Datatype Library Language

- Language for describing datatypes
- Use in RELAX NG



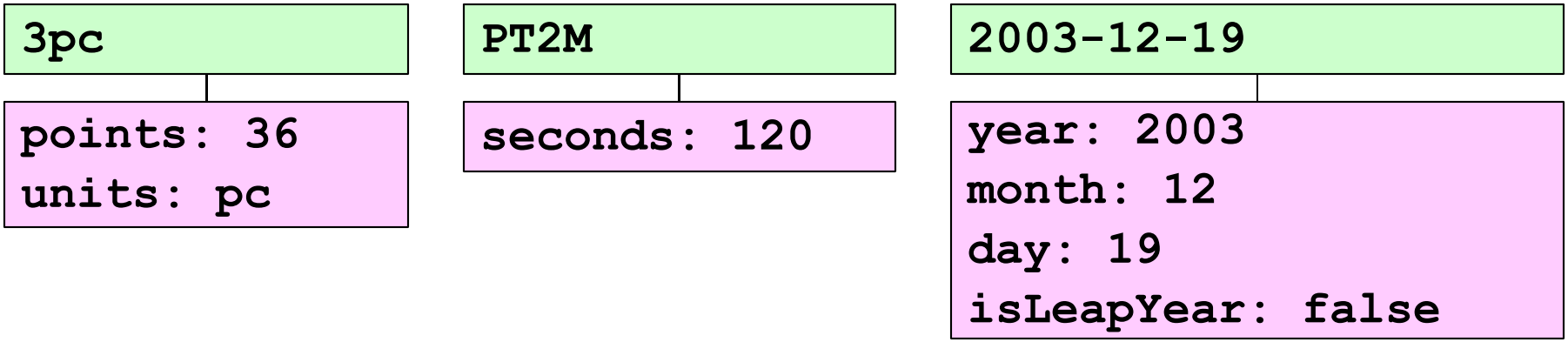
Datatype Library Language

- Use in XSLT 2.0



Lexical Datatyping

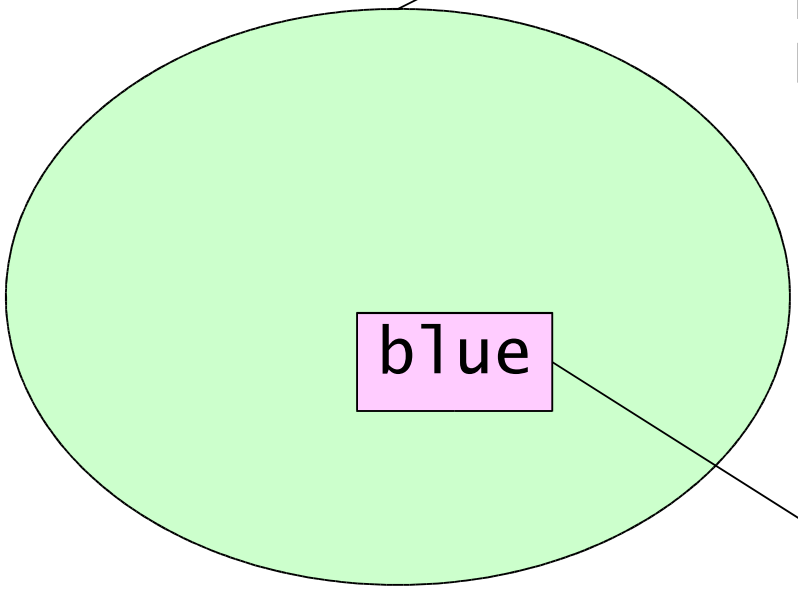
- Values are sequences of characters
- Values have properties
  - accessible via API
  - properties have different types
  - aren't necessarily independent



Datatypes as Annotated Sets

- Datatypes are annotated sets of values
  - annotations include:
    - collations for comparisons
    - datatype parameters
  - datatypes define properties for values of that type
    - abstract datatypes define only properties and constraints on those properties
    - concrete datatypes define lexical structure of strings as well
- Typed value is value + datatype
  - adds property values
  - typed values with same collation can be compared

Example



colourNames
collation: colourCollation
params: lightest, darkest
props: red, green, blue,
hue, saturation, luminance

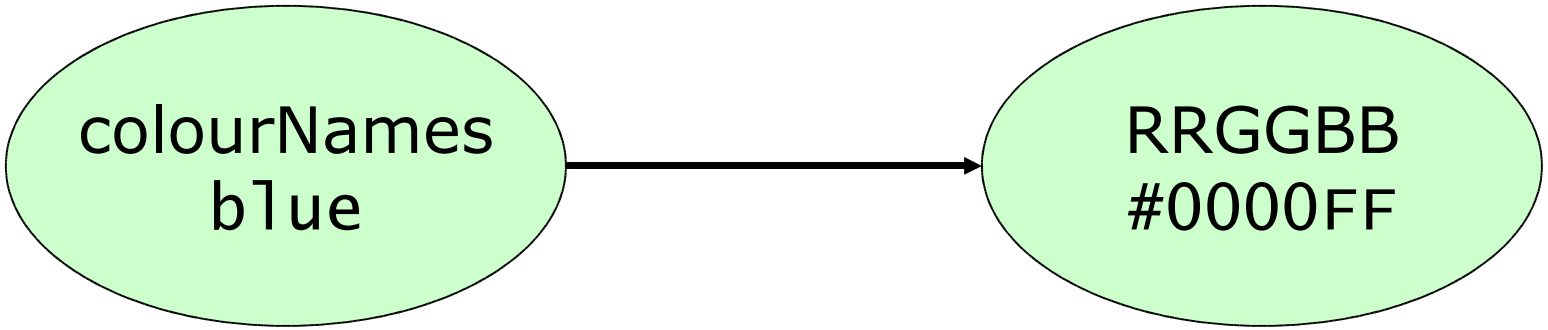
red: 0
green: 0
blue: 255
…

Datatype Definitions

- Extensible set of tests on values
  - valid values must pass all the tests
- Parsing of values via:
  - regex with named subexpressions
  - list definition with particular separators
  - enumeration of values
  - implementation-defined extension methods
    - EBNF, PEGs, …
- Sets of conditions testing property values
  - cross-property conditions
  - testing against parameter values
- Negative conditions

Datatype Mapping

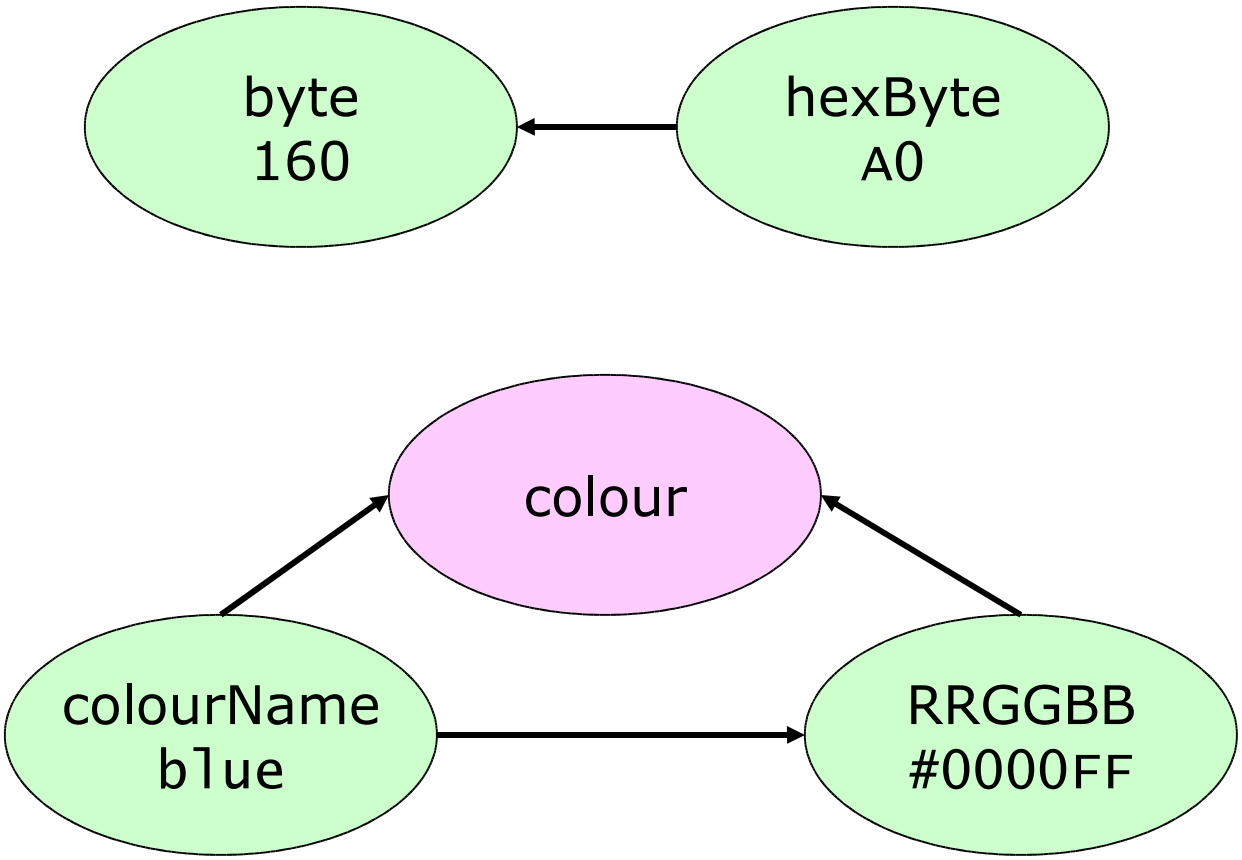
- How value in one datatype maps to value (or properties) in another
  - unidirectional: every source value must be mappable to target
- Supports casting



Supertyping

- Ease definition of types
  - inherit properties and collation
  - can just alter values of parameters
- Supertype doesn't imply superset
  - if supertype is concrete, all values of subtype are legal values of supertype
  - if supertype is abstract, supertyping provides map to (properties defined by) that supertype

Example



Concrete hexByte and byte Types

```
<datatype name="byte">
  <super type="integer">
    <param name="min" value="0" />
    <param name="max" value="255" />
  </super>
</datatype>

<datatype name="hexByte">
  <parse>
    <regex>[0-9A-F]{2}</regex>
  </parse>
  <map to="byte"
    select="my:int(substring($this, 1, 1)) * 16 +
            my:int(substring($this, 2, 1))" />
  <collate type="byte" />
</datatype>
```

Abstract colour Type

```
<datatype name="colour">
  <property name="red" type="byte" />
  <property name="green" type="byte" />
  <property name="blue" type="byte" />
  <property name="hue" type="byte" select="..." />
  <property name="saturation" type="byte" select="..." />
  <property name="luminance" type="byte" select="..." />

  <collate>
    <collate select="$this.hue" type="byte" />
    <collate select="$this.saturation" type="byte" />
    <collate select="$this.luminance" type="byte" />
  </collate>
  ...
</datatype>
```

Abstract colour Type

```
<datatype name="colour">
  ...
  <param name="lightest" type="colour" subtype="le" />
  <param name="darkest" type="colour" subtype="ge" />

  <constraint test="$type.lightest.luminance >=
                    $type.darkest.luminance" />

  <condition test="$type.lightest.luminance >=
                   $this.luminance" />
  <condition test="$this.luminance >=
                   $type.darkest.luminance" />
</datatype>
```

Concrete RRGGBB Type

```
<datatype name="RRGGBB">
  <param name="lightest" type="RRGGBB" subtype="le" />
  <param name="darkest" type="RRGGBB" subtype="ge" />
  <super type="colour">
    <param name="lightest" select="$type.lightest" />
    <param name="darkest" select="$type.darkest" />
  </super>
  <parse name="RRGGBB">
    <regex ignore-whitespace="true">
      #(?[RR][0-9A-F]{2})
       (?[GG][0-9A-F]{2})
       (?[BB][0-9A-F]{2})
    </regex>
  </parse>
  <property name="red" select="hexByte($RRGGBB/RR)" />
  <property name="green" select="hexByte($RRGGBB/GG)" />
  <property name="blue" select="hexByte($RRGGBB/BB)" />
</datatype>
```

Concrete colourName Type

```
<datatype name="colourName">
  <super type="colour" />
  <parse name="colour">
    <enumeration code="@name"
      values="document('colours.xml')/colours/colour"/>
  </parse>
  <property name="red" select="$colour/@red" />
  <property name="green" select="$colour/@green" />
  <property name="blue" select="$colour/@blue" />
</datatype>
```

```
<colours>
  ...
  <colour name="blue" red="0" green="0" blue="255" />
  ...
</colours>
```

Partial Ordering

- Occurs with durations and date/times (due to timezones)

```
xs:duration('P1M') = xs:duration('P30D')
```

- Use min/max collations

```
<collate type="xs:decimal"
         select.min="my:min-seconds($this)"
         select.max="my:max-seconds($this)" />
```

- XPath comparisons based on two-value logic
  - true/false, rather than true/false/unknown
  - map unknown to empty sequence (false)

Context Information

- Standard extension functions for Infoset information:
  - inf:ns-for-prefix($prefix) returns URI
  - inf:prefix-declared($prefix) returns boolean
  - inf:entity-declared($entity) returns boolean
  - …
- Implementations can define additional extension functions for other context information

Complex Structured Values

- No built-in support for complex structures:
  - XPointers
  - XPaths
  - XSLT patterns
  - XSL-FO expressions
  - regular expressions
- But implementations can provide support via extension parse methods
  - standardise these later

XPath Datatyping Problem

- Want to use XPath to express:
  - bindings to properties
  - conditions that have to be met by values
- Want expressions to be datatype aware

```
<condition test="$type.lightest.luminance >=
                 $this.luminance" />
<condition test="$this.luminance >=
                 $type.darkest.luminance" />
```

- Should be possible in XPath 2.0
  - we know what type each value actually is
  - and therefore how they should be compared

XPath Datatyping Problem (cont…)

- But XPath 2.0 assumes WXS datatypes
  - datatypes need to fit into type hierarchy
  - no mechanisms for
    - having different collations for comparison operators
    - defining casts to known datatypes
- Use functions for comparisons

```
<condition test="dt:ge($type.lightest.luminance,
                       $this.luminance)" />
<condition test="dt:ge($this.luminance,
                       $type.darkest.luminance)" />
```

Status

- Draft spec available at: http://www.jenitennison.com/datatypes
  - schemas also available there
  - comments, please!
- No implementations yet
  - help, please!
- That's it
  - questions, please!