



# Who's Afraid of XML Schema?

---

Neil Graham  
IBM Canada Ltd.





# Objectives

---

- Review some of the “scariest” aspects of XML Schema
  - Focus on broad concepts
  - Heavy use of examples
- Prove that the basics of XML Schema are not difficult to grasp
  - Common scenarios make sense
  - Every language has its warts and gotchas



# Topics

---

- The Power of Datatypes
- Substitution groups/xsi:type
- Overview of complexType Restriction
- Identity Constraints
- The Element Declarations Consistent Constraint
- The Unique Particle Attribution Constraint



# Datatypes: Basics

---

- Each datatype defines a mapping from sequences of characters to an abstract **value space**
  - Value space: what the type “really means”
    - *e.g.*, an integer is a number, not a bit- or character-representation on a machine



# Datatypes: Basics *(Continued)*

---

- Datatypes are just a hierarchy, where the children of the root node (primitive types) and not the root node are all users need be concerned with
  - These children are the **primitive datatypes** (the “nasty nineteen”)
  - They have disjoint value spaces
  - Most are obvious and generally useful (string, decimal, date, time, boolean, base64, double)
  - Others are more specialized (duration, dateTime, gYear and friends, NOTATION, QName)



# Datatypes: Basics *(Continued)*

---

- All other datatypes, including the 25 defined by XML Schema, are **derived** from these
- Derivation is generally by **restricting facets** associated with the base type
  - Excludes strings the base type would have accepted, either directly or by taking their values from the value space
- Many of XML Schemas derived types are obvious and generally useful (integer, long, int, short, byte)
  - Others present for DTD compatibility (NMTOKEN, ID, IDREF, ENTITY)
  - Others are more specialized (language, Name, NCName, token, nonPositiveInteger)



■ You can derive your own types too!



# Datatypes: Primitive

---

```
<xs:element name="myString"  
  type="xs:string"/>
```

```
<xs:element name="myDecimal"  
  type="xs:decimal"/>
```

```
<xs:element name="myDate"  
  type="xs:date"/>
```

```
<xs:element name="myBase64Binary"  
  type="xs:base64Binary"/>
```



## Datatypes: Primitive *(Continued)*

---

```
<myString>Hello world!</myString>
```

```
<myDecimal>3.141526</myDecimal>
```

```
<myDate>1986-04-26</myDate>
```

```
<myBase64Binary>vole mole 3RAT  
Cat5</myBase64Binary>
```



# Predefined Derived Types

---

```
<xs:element name="myNormalizedString"  
  type="xs:normalizedString"/>
```

```
<xs:element name="myLanguage"  
  type="xs:language"/>
```

```
<xs:element name="myLong"  
  type="xs:long"/>
```

```
<xs:element name="myNonPositiveInteger"  
  type="xs:nonPositiveInteger"/>
```



## Predefined Derived *(Continued)*

---

<myNormalizedString>actual value has no  
multiple internal spaces  
</myNormalizedString>

<myLanguage>English-US</myLanguage>

<myLong>-111222333444</myLong>

<myNonPositiveInteger>0</myNonPositiveInteger>



# User-Defined Derived Types

---

```
<xs:element name="derivedString">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z]{4}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<derivedString>buZZ</derivedString>
```



## User-Defined Types *(Continued)*

---

```
<xs:element name="derivedDecimal">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="1.1"/>
      <xs:maxExclusive value="1.2"/>
    </xs:restriction>
  </xs:simpleType> </xs:element>
<derivedDecimal>1.199</derivedDecimal>
```



# Datatypes: Any Gotchas?

---

- There are a lot of XML Schema-defined datatypes
  - If they don't seem useful/comprehensible, ignore them!
- Some facets (like pattern) are applied to the characters in the document
- Most facets are applied after the characters are mapped to a value
  - *e.g.*, minInclusive, maxExclusive, enumeration, length



# substitutionGroups: Basics

---

- A global element (M) may declare that it may be used in place of another global element (H)
  - So long as M's type is the same as or derives from H's
  - M is said to be a member of H's substitutionGroup
- M may be used anywhere in an instance document H may be



# substitutionGroups: Example

---

Schema:

```
<element name="salary" type="tns:salaryType"
  abstract="true"/>
```

```
<element name="lecturerSalary"
  type="tns:lecturerSalaryType"
  substitutionGroup="tns:salary"/>
```

- The salary element can be used in the schema (*e.g.*, in a record for an instructor)
- In the instance, salary cannot be used (because it is abstract)
- But lecturerSalary may be used in its place



# xsi:type: Basics

---

- Consider an element E of type T
- If type D derives from T, then (normally) xsi:type can be used to force validation of an instance of E according to D instead of T
- Analogous to run-time casting in Java



# xsi:type: Example

---

```
<element name="course" type="tns:courseType" >
<complexType name="courseType">...
<complexType name="sciCourseType">
  <simpleContent>
    <extension base="tns:courseType">...
<complexType name="mathCourseType">
  <simpleContent>
    <restriction base="tns:sciCourseType">
<course xsi:type="mathCourseType" courseNum="7654"
  title="Tensor Calculus for Dummies"
  textISBN="0-13136-6666-5">
```



# Related Features

---

- Just like Java classes, complexTypes can declare themselves final w.r.t. extension or restriction
- complexTypes can also block xsi:type replacement in an instance
- Element declarations can also prevent other element declarations from declaring that they are in their substitutionGroup
- Element declarations can also block xsi:type replacement in an instance



# complexType Restriction: Basics

---

- complexType extension: just add attributes and/or elements to the end of the base type's content model
- Restriction:
  - with attributes, gives instance fewer valid options
  - With content, (generally) allows a subset of the base type's content
  - *i.e.*, if it's valid against the restriction, valid against the base



## Restriction: Basics *(Continued)*

---

- Four main aspects of complexTypes can be restricted:
  - Attributes associated with a type can be disallowed or made non-optional
  - Order of elements can be fixed
  - Number of occurrences of elements can be limited
  - Content of elements/values of attributes may be restricted



# Restriction: Example 1

---

```
<complexType name="courseType">
  <simpleContent>
    <extension base="string">
      <attribute name="title" type="string"/>
      <attribute name="courseNum"
type="unsignedInt"/>
      <attribute name="textISBN"
type="tns:isbnType"/>
    </extension>
  </simpleContent> </complexType>
```



## Restriction: Example 1 *(Continued)*

---

```
<complexType name="sciCourseType">  
  <simpleContent>  
    <extension base="tns:courseType">  
      <attribute name="labManual"  
        type="tns:isbnType"/>  
    </extension>  
  </simpleContent>  
</complexType>
```



## Restriction: Example 1 *(Continued)*

---

```
<complexType name="mathCourseType">
  <simpleContent>
    <restriction base="tns:sciCourseType">
      <!-- title was optional in the base -->
      <attribute name="title" type="string"
        use="required"/>
      <!-- labManual was also optional -->
      <attribute name="labManual"
        use="prohibited"/>
    </restriction>
  </simpleContent>
</complexType>
```



# Restriction: Example 1 *(Continued)*

---

```
<attribute name="courseNum">  
  <simpleType>  
    <restriction base="unsignedInt">  
      <minInclusive value="7000"/>  
      <maxExclusive value="8000"/>  
    </restriction>  
  </simpleType>  
</attribute>
```



# Restriction: Example 1 *(Continued)*

---

```
</restriction>
```

```
</simpleContent>
```

```
</complexType>
```

```
<element name = "course"  
  type= "tns:courseType" />
```



## Restriction: Example 1 *(Continued)*

---

```
<course xsi:type="mathCourseType"  
  courseNum="7654"  
  title="Tensor Calculus for Dummies"  
  textISBN="0-13136-6666-5">
```

This course tries to bring tensor calculus down to the level

where the average person can understand it. It seldom works.

```
</course>
```



## Restriction: Example 1 *(Continued)*

---

```
<course xsi:type="sciCourseType"
  courseNum="3279"
  title="Readings in Rabbit Psychology"
  textISBN="0-11223-3445-4" labManual="0-
  11220-0445-6">
```

This course introduces the fascinating subject of rabbit dreams, goals and fears, with especial emphasis on fears.

```
</course>
```



## Restriction: Example 2

---

```
<complexType name="nameType">  
  <all>  
    <element name="given" type="token"/>  
    <element name="middle" type="token"  
      nillable="true"/>  
    <element name="family" type="token"/>  
  </all>  
</complexType>
```



## Restriction: Example 2 *(Continued)*

---

```
<complexType name="orderedNameType">
  <complexContent>
    <restriction base="tns:nameType">
      <sequence>
        <element name="family" type="token"/>
        <element name="given" type="token"/>
        <element name="middle" type="token"
nillable="true"/>
      </sequence></restriction>
    </complexContent>
  </complexType>
```



## Restriction: Example 2 *(Continued)*

---

```
<element name="name" type="tns:nameType"/>
```

```
<name xsi:type="orderedNameType">
```

```
  <family>Parsec</family>
```

```
  <given>Saul</given>
```

```
  <middle xsi:nil="true"/>
```

```
</name>
```



## Restriction: Example 3

---

```
<complexType name="instructorType">
  <sequence>
    <element name="name" type="tns:nameType"
      form="qualified"/>
    <element name="courseNum" type="unsignedInt"
      minOccurs="0" maxOccurs="unbounded"/>
    <element ref="tns:salary"/>
  </sequence>
</complexType>
```



## Restriction: Example 3 *(Continued)*

---

```
<complexType name="professorEmeritusType">
  <complexContent>
    <restriction base="tns:instructorType">
      <sequence>
        <element ref="tns:name"/>
        <element name="courseNum"
          type="unsignedInt"
          minOccurs="0" maxOccurs="0"/>
        <element ref="tns:professorEmeritusSalary"/>
      </sequence>    </restriction>
    </complexContent>  </complexType>
```



## Restriction: Example 3 *(Continued)*

---

```
<element name="professorEmeritusSalary"  
  type="tns:professorEmeritusSalaryType"  
  substitutionGroup="tns:salary"/>
```

- ```
<element name="instructor"  
  type="tns:instructorType"/>
```

```
<element name="professor"  
  type="tns:instructorType"  
  substitutionGroup="tns:instructor"/>
```

```
<element name="professorEmeritus"  
  type="tns:professorEmeritusType"  
  substitutionGroup="tns:instructor"/>
```



## Restriction: Example 3 *(Continued)*

---

```
<professorEmeritus>
```

```
  <name>
```

```
    <family>Day</family>
```

```
    <middle xsi:nil="true"/>
```

```
    <given>Holly</given>
```

```
  </name>
```

```
<professorEmeritusSalary>0</professorEmeritusSalary>
```

```
</professorEmeritus>
```



# Restriction: Complexities

---

- anyType, which all complexTypes in schema implicitly or explicitly restrict, is slightly magical
  - This makes theorists antsy, but has few if any practical consequences
- Schema rules describing restriction tend to scare even schema experts
  - But you shouldn't need to dwell on them



# Identity Constraints: Basics

---

- Provide a means for ensuring that values:
  - Exist in some context
  - Are unique within a context
  - Have other values that correspond to them
- Similar to XML 1.0 ID/IDREF except:
  - May have other than document scope
  - Use schema datatypes for comparisons
  - Multiple values can participate



## IDC: Basics *(Continued)*

---

- Specified on element declarations
- Use a (restricted) XPath syntax to select values to act upon
- These XPaths are relative to elements/attributes in instance documents
  - *i.e.*, the content of the instance of the element validated by the declaration/reference they are defined on



## IDC: Basics *(Continued)*

---

- Two parts to the XPath selection:
  - A “selector” XPath identifies a set of elements in the subtree that are of interest
  - Starting with each of these elements, each “field” XPath then finds an element or attribute value
- Depending on the type of identity constraint, this set of values identified by the “fields” is then compared in some way with the other sets corresponding to nodes found by the “selector”



## IDC: Basics *(Continued)*

---

- For clarity, let  $I$  be an identity constraint on some element declaration  $E$
- Let  $S$  be the set of elements identified by  $I$ 's selector starting from an element instance validated by  $E$
- Let  $F_s$  be the set of values matched by the fields of  $I$  starting with some element  $s$  in  $S$
- $F_s$  exists if and only if all fields identify values in the subtree rooted at  $s$



## IDC: Basics *(Continued)*

---

- There are three kinds of identity constraint:
  - Unique: for distinct elements  $s$  and  $t$  in  $S$ , if  $F_s$  and  $F_t$  exist, they must not be identical
  - Key: Same as unique, but for all elements  $s$  in  $S$ ,  $F_s$  must exist
  - Keyref: Let  $J$  be the identity constraint referred to by the keyref in the schema and  $S'$  be the set of nodes selected by  $J$ 's selector for some element instance
    - Then for every  $F_s$ , there must be an identical  $F_{s'}$  for some  $s'$  in  $S'$



# IDC: Example 1

---

```
<element name="instructorList">  
  <complexType>  
    <sequence>  
      <element ref="tns:instructor"  
maxOccurs="unbounded"/>  
    </sequence>  
  </complexType>
```



# IDC: Example 1 *(Continued)*

---

```
<unique name="uniqueInstructors">  
  <selector xpath="./*/tns:name"/>  
  <field xpath="tns:given"/>  
  <field xpath="tns:family"/>  
</unique>  
</element>
```



# IDC: Example 1 *(Continued)*

---

```
<instructorList>
  <professor>
    <name xsi:type="orderedNameType">
      <family>Parsec</family>
      <given>Saul</given>
      <middle/>
    </name>
    <courseNum>7654</courseNum>
    <professorSalary>125000</professorSalary>
  </professor>
```



# IDC: Example 1 *(Continued)*

---

```
<lecturer>
  <name>
    <given>Wolfgang</given>
    <middle>Rathouser</middle>
    <family>Katz</family>
  </name>
  <courseNum>3279</courseNum>
  <lecturerSalary>70000</lecturerSalary>
</lecturer>
</instructorList>
```



## IDC: Example 2

---

```
<simpleType name="isbnType">  
  <restriction base="string">  
    <pattern value="\d-\d{5}-\d{4}-[X\d]"/>  
  </restriction>  
</simpleType>
```



## IDC: Example 2 *(Continued)*

---

```
<complexType name="bookType">  
  <attribute name="title" type="string"/>  
  <attribute name="author" type="string"/>  
  <attribute name="publisher" type="string"/>  
  <attribute name="copyright"  
    type="gYear"/>  
  <attribute name="isbn"  
    type="tns:isbnType"/>  
</complexType>
```



## IDC: Example 2 *(Continued)*

---

```
<element name="approvedBookList">  
  <complexType>  
    <choice maxOccurs="unbounded">  
      <element name="textBook"  
type="tns:bookType"/>  
      <element name="labManual"  
type="tns:bookType"/>  
    </choice>  
  </complexType>
```



## IDC: Example 2 *(Continued)*

---

```
<key name="isbnKey">  
  <selector xpath="tns:textBook |  
tns:labManual"/>  
  <field xpath="@isbn"/>  
</key>  
</element>
```



## IDC: Example 2 *(Continued)*

---

```
<approvedBookList>
```

```
<textBook title="Tensors Are Fun and Easy!"  
copyright="1983" author="A. Weyl Killing"  
publisher="Springing-Furlongs"  
isbn="0-13136-6666-5"/>
```

```
<textBook title="Studies in Rabbithood"  
copyright="1997" author="Bunny O'Hare"  
publisher="FeelGood Books"  
isbn="0-11223-3445-4"/>
```



## IDC: Example 2 *(Continued)*

---

```
<labManual title="A Practical Guide to Rabbit  
Psychoanalysis"
```

```
  copyright="1957" publisher="Lucy Lapin"
```

```
  isbn="0-11220-0445-6"/>
```

```
</approvedBookList>
```



## IDC: Example 3

---

```
<element name="university">  
  <complexType>  
    <sequence>  
      <element name="courseList">  
        ...  
      <element name="approvedBookList">  
        ...  
      <element name="instructorList">  
        ...  
    </sequence>  
  </complexType>
```



## IDC: Example 3 *(Continued)*

---

```
<keyref name="textISBNExistsKey"
refer="tns:isbnKey">
  <selector xpath="./tns:courseList/tns:course"/>
  <field xpath="@textISBN"/>
</keyref>
<keyref name="labISBNExistsKey"
refer="tns:isbnKey">
  <selector xpath="./tns:courseList/tns:course"/>
  <field xpath="@labManual"/>
</keyref>
</element>
```



## IDC: Example 3 *(Continued)*

---

```
<university
```

```
  xsi:schemaLocation = "http://www.college.edu  
  university.xsd"
```

```
  xmlns:xsi =
```

```
  "http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns = "http://www.college.edu">
```

```
<courseList>
```



## IDC: Example 3 *(Continued)*

---

```
<course xsi:type="sciCourseType"
courseNum="3279"
title="Readings in Rabbit Psychology"
textISBN="0-11223-3445-4" labManual="0-11220-
0445-6">
```

This course introduces the fascinating subject of rabbit dreams, goals and fears, with especial emphasis on fears.

```
</course>
```

```
</courseList> ... </university>
```



# IDC: Gotchas

---

- Remember that default namespaces don't work with XPath statements!
- Depending on the schema, elements declared local to a complexType may not have a namespace --- beware!
- The identity constraint referred to by a keyref must appear on the keyref's element or on a descendant of the keyref's element in the document



# Element Declarations Consistent

---

- This constraint relates to elements used in a particular content model
- It means all elements with the same name must have the same type
- This is highly useful for systems that map schema types to language constructs
  - *e.g.*, Java classes



# EDC: Example 1

---

```
<complexType name="badType1">
  <sequence>
    <element name="text"
      type="tns:isbnType"/>
    <element name="blurb" type="string"/>
    <element name="text" type="string"/>
  </sequence>
</complexType>
```



## EDC: Example 2

---

```
<complexType name="badType2">  
  <sequence>  
    <element name="text">  
      <simpleType>  
        <restriction base="string">  
          <pattern value="\d-\d{5}-\d{4}-[X\d]"/>  
        </restriction>  
      </simpleType>  
    </element>
```



## EDC: Example 2 *(Continued)*

---

```
<element name="blurb" type="string"/>
<element name="text">
  <simpleType>
    <restriction base="string">
      <pattern value="\d-\d{5}-\d{4}-[X\d]"/>
    </restriction>
  </simpleType>
</element>
</sequence>
</complexType>
```



# Unique Particle Attribution

---

- “particles” are used to define content models
- They have a “term” that describes what they permit. Types of term:
  - An element reference or local element declaration
  - A wildcard (<any>)
  - A choice or sequence
- They associate the term with occurrence info
- Useful for validators that there is no ambiguity about which particle to use during validation



# UPA: Invalid Example 1

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  elementFormDefault="qualified">
  <element name="text" type="string"/>
  <!-- "text" and the wildcard are in the same namespace -->
  <complexType name="badType1">
    <sequence>
      <any namespace="##targetNamespace"
        processContents="skip"
        minOccurs="0"/>
      <element ref="tns:text"/>
    </sequence>
  </complexType>
</schema>
```



# UPA: Valid Example 1

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <!-- "text" and wildcard in different namespaces -->
  <complexType name="goodType">
    <sequence>
      <element name="text" type="string" minOccurs="0"/>
      <any namespace="##local" processContents="skip"/>
    </sequence>
  </complexType>
</schema>
```



## UPA: Invalid Example 2a

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema">
<!-- if "text" occurs, is it the elem decl or the wildcard? -->
<complexType name="badType1">
  <sequence>
    <element name="text" type="string" minOccurs="0"/>
    <any namespace="##local" processContents="skip"/>
  </sequence>
</complexType> </schema>
```



## UPA: Invalid Example 2b

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema">
<!-- if "text" occurs, is it the elem decl or the wildcard? -->
<complexType name="badType1">
  <sequence>
    <any namespace="##local" processContents="skip"
      minOccurs="0"/>
    <element name="text" type="string"/>
  </sequence>
</complexType>
</schema>
```



## UPA: Valid Example 2a

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema">
<!-- "text" must be present -->
<complexType name="goodType1">
  <sequence>
    <element name="text" type="string"/>
    <any namespace="##local" processContents="skip"
      minOccurs="0"/>
  </sequence>
</complexType>
</schema>
```



## UPA: Valid Example 2b

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema">
<!-- wildcard is non-optional -->
<complexType name="goodType1">
  <sequence>
    <any namespace="##local" processContents="skip"/>
    <element name="text" type="string" minOccurs="0"/>
  </sequence>
</complexType>
</schema>
```



## UPA: Invalid Example 3

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema">
<!-- if "text" occurs, is it the elem decl or the wildcard? -->
<complexType name="badType1">
  <choice>
    <element name="text" type="string"/>
    <any namespace="##local" processContents="skip"/>
  </choice>
</complexType>
</schema>
```



## UPA: Valid Example 3

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <!-- "text" and wildcard in different namespaces -->
  <complexType name="goodType1">
    <choice>
      <element name="text" type="string"/>
      <any namespace="http://foo.com"
        processContents="lax"/>
    </choice>
  </complexType>
</schema>
```



## UPA: Invalid Example 4

---

```
<schema xmlns:tns="http://www.college.edu"
  targetNamespace="http://www.college.edu"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <!-- when "foo" occurs, which sequence are we in? -->
  <complexType name="badType1">
    <choice>
      <sequence>
        <element name="foo" type="string"/>
        <element name="bar" type="string"/>
      </sequence>
    </choice>
  </complexType>
</schema>
```

# UPA: Invalid Example 4

*(Continued)*

---

```
<sequence>  
  <element name="foo" type="string"/>  
  <element name="baz" type="string"/>  
</sequence>  
</choice>  
</complexType>  
  
</schema>
```



# UPA: Summary

---

- Two particles overlap if
  - One is an element and the other is a wildcard that contains the element's namespace
  - They're both wildcards with overlapping namespaces
  - They're elements with the same name
- If they're in a sequence, the first must occur a set number of times
- Both cannot be initial options in a choice



# Schema: Terminology

---

- If you have to read the Schema specs...
- Descriptions focus on “schema components”
  - What a schema processor turns the XML representation of the schema into
- And validating “information items”
  - Contents of XML documents at an abstract level
- Conceptually, schema components derived from anywhere can validate XML Information items derived from anywhere (not just XML documents)



# Conclusion

---

- XML Schema is very powerful
  - The Schema specs themselves can be positively impenetrable
- But even the more obscure features can be demonstrated with examples covering only a few slides
- Every language has its edge cases and rough spots; Schema's should seldom get in your way



# References — XML Schema

---

- XML Schema 1.0 is now in its 2<sup>nd</sup> edition
- XML Schema Part 0: Primer:  
<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/XML>
- Schema Part 1: Structures:  
<http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>

# References — XML Schema

*(Continued)*

---

- XML Schema Part 2: Datatypes:  
<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>
- The Primer is an easy read and an excellent place to start studying XML Schema!