# XML: Data and Document Processing

## XML Schema

David Edmond
School of Information Systems
Queensland University of Technology

February 23, 2007

# Introduction

# The Basics of XML Schema

- XML Schema is a language for specifying the *allowable content and structure* of an XML document.
- The XML Schema notation is itself written in the form of an XML document.
- Despite its XML appearance, XML Schema is much more like the data aspect of a conventional programming language – at least compared with the DTD notation.

✓ `http://www.w3.org/TR/xmlschema-0/`

✓ `http://www.w3.org/TR/xmlschema-1/`

✓ `http://www.w3.org/TR/xmlschema-2/`

✓ `http://www.xfront.com/`

✓ `http://msdn2.microsoft.com/en-us/library/ms256235.aspx`

# Topics

In these notes, we discuss the XML Schema language. In particular, we discuss:

- Comparisons with DTDs.
- Built-in data types.
- Custom-built data types.
- Patterns
- Complex and simple types.
- Element declarations.
- Attribute declarations.

# DTD vs XML Schema

Consider the following XML document:

```
<Person>
<Name>Frank</Name><Age>21</Age><Sex>M</Sex>
</Person>
```

We could specify documents of this form using a DTD:

```
<!ELEMENT Person (Name,Age,Sex) >
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Age  (#PCDATA) >
<!ELEMENT Sex  (#PCDATA) >
```

# XML Schema alternative

Alternatively, we could use XML schema:

```
<xsd:element name="Name"  type="xsd:string"/>
<xsd:element name="Age"   type="xsd:integer"/>
<xsd:element name="Sex">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="M"/>
      <xsd:enumeration value="F"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Something is still missing. What do you think it is?

# Some observations

- syntax
  - DTDs are written in . . .
  - Schemas are written in . . .
- length
  - DTDs are . . .
  - XML schemas are . . .
- structure and content
  - DTDs deal, primarily, with . . .
  - Schemas deal with . . .

# Data types

# XML Schema and simple data types

- One of the big differences between DTD and XML Schema is in the area of data typing.
- DTD offers very little support – apart from enumeration of allowable attribute values.
- XML Schema has a rich range of predefined data types – such as we would expect in a programming language or in SQL.
- These are available for use in attributes and in elements.
- XML Schema also offers ways to customise our own data types.

# Primitive datatypes (19)

| | |
|---|---|
| `string` | `Pamela Edgar` |
| `boolean` | `true, false, 1, 0` |
| `decimal` | `10.3, -99.188` |
| `float` | `10.3, .103E2` |
| `double` | `-1E4, 1267.43233E12` |
| `duration` | `P1Y2M3D, P1DT12H` |
| `dateTime` | `2002-04-8T14:00:00.000` |
| `time` | `4:00:00.000` |
| `date` | `2002-04-8` |
| `gYearMonth` | `1997-03` |
| `gYear` | `1997` |
| `gMonthDay` | `--08-06` |
| `gDay` | `mate` |
| `gMonth` | `--10--` |
| `hexBinary` | `0FB7` |
| `base64Binary` | “for encoding binary data” |
| `anyURI` | `http://www.qut.edu.au` |
| `QName` | `xsd:string` |
| `NOTATION` | cf DTD concept |

# Derived datatypes (25)

There area further 25 types which may be viewed as specialisations of the primitive types. Here are *some* of them:

| | |
|---|---|
| token | |
| language | en, fr, de |
| NMTOKEN | (NameChar)+ |
| Name | (Letter\|’_’\|’:’)<br>(NameChar)* |
| NCName | (Letter\|’_’)<br>(NCNameChar)* |
| integer | 1913, -273 |
| nonPositiveInteger | ..., -2, -1, 0 |
| negativeInteger | ..., -2, -1 |
| long | from $-2^{63}$ to $2^{63}-1$ |
| int | from $-2^{31}$ to $2^{63}-1$ |
| short | from $-2^{15}$ to $2^{15}-1$ |
| byte | from -128 to 127 |
| nonNegativeInteger | 0, 1, 2, ... |
| unsignedLong | from 0 to $2^{64}-1$ |
| unsignedInt | from 0 to $2^{32}-1$ |

# Custom-built simple types

# Restriction and extension

- *Restriction*: the creation of a new type by adding conditions either to one of the built-in types or to a previously customised type.
- The way of adding conditions is by means of *facets*: a facet is a specific way of customising a data type.
- *Extension* the creation of a new type by adding attributes or elements to a previously defined (simple or complex) type.

# Facets

The following facets are available:

- `enumeration` of allowable values
- `patterns` of allowable values
- `minInclusive` and `minExclusive` for the low end of a range
- `maxInclusive` and `maxExclusive` for the high end of a range
- `length`, `minLength` and `maxLength` for specifying the allowable number of units
- `whiteSpace` provides a way of saying how white space is to be handled (e.g., preserved or collapsed)

# Example 1: Enumerating allowable values

Suppose we want to use an element called `<state/>` which is to be used to represent Australian states:

```
<state capital="Brisbane">Queensland</state>
```

We can constrain the values used for the `capital` attribute in the following way:

```
<xsd:attribute name="capital" use="required">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Brisbane"/>
      <xsd:enumeration value="Sydney"/>
      <xsd:enumeration value="Melbourne"/>
      <xsd:enumeration value="Adelaide"/>
      <xsd:enumeration value="Perth"/>
      <xsd:enumeration value="Hobart"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

We have introduced an *anonymous* simple type.

# Example 2: Range restriction

A contract is to be valid for a period of between 6 weeks and 6 months inclusive. It is to be used in the following element:

```
<xsd:element name="ContractLength" type="ValidityPeriodType"/>
```

Define a *named* simple type to suit:

```
<xsd:simpleType name="ValidityPeriodType">
  <xsd:restriction base="xsd:...............">
    <xsd:minInclusive value="..............."/>
    <xsd:............... value="P6M"/>
  </xsd:restriction>
</xsd:simpleType>

```

# Example 2: Range restriction

A contract is to be valid for a period of between 6 weeks and 6 months inclusive. It is to be used in the following element:

```
<xsd:element name="ContractLength" type="ValidityPeriodType"/>
```

Define a *named* simple type to suit:

```
<xsd:simpleType name="ValidityPeriodType">
  <xsd:restriction base="xsd:duration">
    <xsd:minInclusive value="P1M14D"/>
    <xsd:maxInclusive value="P6M"/>
  </xsd:restriction>
</xsd:simpleType>
```

# Regular expressions

A *regular expression* is a sequence of characters that denote a set of strings. Here are a number of examples:

| | |
|---|---|
| `[A-D]` | the letters *A*, *B*, *C* and *D* |
| `[f-h]` | the letters *f*, *g* and *h* |
| `[0-9]` | the digits *0* to *9* inclusive |
| `\d` | the digits *0* to *9* inclusive |
| `[p-r0-1]` | the five characters *p*, *q*, *r*, *0* and *1* |
| `[QUT]` | the three letters *Q*, *U* and *T* |
| `Pie` | the string *Pie* |
| `Cutie\|Pie` | the string *Cutie* or the string *Pie* |
| `A+` | one or more *A*s, e.g., *A*, *AA*, *AAA* |
| `Aug-[0-9]{2}` | the strings *Aug-00* to *Aug-99* |
| `Bug{2,4}` | the strings *Bugg*, *Buggg* and *Bugggg* |

✓ `http://www.w3.org/TR/xmlschema-0/#regexAppendix`

# Example 3: Restriction by means of a simple pattern

The `ProductCode` element is to consist of the letter p followed by between two and six digits:

```
<xsd:element name="ProductCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="p\d{2,6}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Example 4: A more complex pattern

Restrict the following `UnitCode` element to consist of the letters `BS` followed by either a `B` or an `N` followed by a hyphen followed by exactly two digits.

```
<xsd:element name="UnitCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:pattern value="..............."/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Example 4: A more complex pattern

Restrict the following `UnitCode` element to consist of the letters `BS` followed by either a `B` or an `N` followed by a hyphen followed by exactly two digits.

```
<xsd:element name="UnitCode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:pattern value="BS(B|N)-\d{2}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Example 5: Specifying allowable SQL names

In a relational database, there may appear a number of different kinds of data object, such as tables, columns, views and constraints. Each of these objects will have a name. The rules for a name in SQL are that it consists of a string of no more than 128 characters made up of letters, digits, _, #, $ and @ symbols. Devise a type called *SQLname* that contains these rules.

```
<xsd:simpleType name="...............">
  <xsd:restriction base="...............">
    <xsd:pattern value="..............."/>
  </xsd:restriction>
</xsd:simpleType>
```

# Example 5: Specifying allowable SQL names

In a relational database, there may appear a number of different kinds of data object, such as tables, columns, views and constraints. Each of these objects will have a name. The rules for a name in SQL are that it consists of a string of no more than 128 characters made up of letters, digits, _, #, $ and @ symbols. Devise a type called *SQLname* that contains these rules.

```
<xsd:simpleType name="SQLname">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[A-Za-z\d_#$@]{1,128}"/>
  </xsd:restriction>
</xsd:simpleType>
```

# Other simple types in XML Schema

# List and Union types

- So far, the assumption has been that simple datatypes in XML Schema are just like those in a programming language.
- By this we mean data objects like strings, integers, dates, and so on. These are called *atomic* types.
- In XML Schema, however, there are two further simple types:
  - List types: for sets of items of the same kind
  - Union types: for items of possibly different kinds

✓ `http://www.w3.org/TR/xmlschema-0/#CreatDt`

# Example 6: A ’List’ type

Suppose we allow students to pick a set of units from those on offer. Something like the following:

```
<choice>ITB011 ITB101 ITB222</choice>
```

The allowable units are encoded as follows:

```
<xsd:simpleType name="UnitCode">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="ITB001"/>
    <xsd:enumeration value="ITB002"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:element name="choice" type="unitCodeList"/>
```

```
<xsd:simpleType name="unitCodeList">
  <xsd:list itemType="UnitCode"/>
</xsd:simpleType>
```

# Example 7: A ’Union’ type

Suppose the rules have changed: a student must now make a choice between one unit to study *or* one company for work experience – something like the following:

```
<choice>Oracle</choice>
<choice>ITB001</choice>
```

The allowable units are encoded as follows:

```
<xsd:simpleType name="Company">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Microsoft"/>
    <xsd:enumeration value="Oracle"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:element name="choice" type="pickJustOne"/>
```

```
<xsd:simpleType name="pickJustOne">
  <xsd:union memberTypes="UnitCode Company"/>
</xsd:simpleType>
```

# Element declarations

# Simple vs complex types

- Elements that contain subelements or carry attributes are said to have *complex* types.
- Elements that only contain numbers (or strings or dates, etc.) but do not contain any subelements are said to have *simple* types.
- Some elements have attributes; attributes always have simple types.

# Example 8: Simple or complex?

Which of the following are simple types and which are complex:

```
<Phonebook>
  <Entry>
    <LastName Title="Miss">Edgar</LastName>
    <FirstName>Pam</FirstName>
    <School>Optometry</School>
    <Campus>GP</Campus>
    <Room>B501</Room>
    <Extension>35695</Extension>
  </Entry>
<!.. and so on. ..>
</Phonebook>
```

1. Campus ...............
2. Entry ...............
3. Extension ...............
4. FirstName ...............
5. LastName ...............
6. Phonebook ...............
7. Room ...............
8. School ...............
9. Title ...............

# Example 8: Simple or complex?

Which of the following are simple types and which are complex:

```
<Phonebook>
  <Entry>
    <LastName Title="Miss">Edgar</LastName>
    <FirstName>Pam</FirstName>
    <School>Optometry</School>
    <Campus>GP</Campus>
    <Room>B501</Room>
    <Extension>35695</Extension>
  </Entry>
<!.. and so on. ..>
</Phonebook>
```

```
Campus     simple
Entry      complex
Extension  simple
FirstName  simple
LastName   complex
Phonebook  complex
Room       simple
School     simple
Title      simple
```

# Content models

“The order and structure of the child elements of a complex type are known as its *content model*.” The possible contents of a type are defined using a combination of the following:

- Model groups
- Element declarations
- Element references
- Wildcards

# Model groups

A model group allows the specification of an element’s children:

- The `sequence` group requires the children to appear in a fixed order.
- The `choice` group specifies a set of children, only one of which may appear in any instance.
- The `all` group specifies a set of children, all of which may appear and in any order.

The requirements specified above may be further refined by means of *occurrence constraints*.

# Example 9: A sequence

An address is always presented in a fixed order:

```
<address>
  <street>121 George St</street>
  <suburb>Paddington</suburb>
  <postcode>4065</postcode>
</address>
```

We might model its content in the following way:

```
<xsd:element name="address">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="street"   type="xsd:string"/>
      <xsd:element name="suburb"   type="xsd:string"/>
      <xsd:element name="postcode" type="xsd:positiveInteger"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

# Example 10: A choice

We can contact people in one of three ways: by phone, email or post:

```
<contact>
  <address>
    <street>121 George St</street>
    <suburb>Paddington</suburb>
    <postcode>4065</postcode>
  </address>
</contact>
```

```
<contact><phone>+61 7 3864 2111</phone></contact>
```

```
<contact><email>andy@gmate.com</email></contact>
```

We might model this choice in the following way:

```
<xsd:element name="contact">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="phone"   type="xsd:string"/>
      <xsd:element name="email"   type="xsd:string"/>
      <xsd:element ref="address"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

# Example 11: An element with mixed content

Mixed content occurs when an element contains a mixture of text and child elements:

```
<message>
You really <bold>must</bold> try this
delicious <bold>new</bold> recipe
 for <italic>sticky date pudding</italic>.
</message>
```

```
<xsd:element name="message">
   <xsd:complexType mixed="true">
      <xsd:choice maxOccurs="unbounded">
         <xsd:element name="bold"   type="xsd:string"/>
         <xsd:element name="italic" type="xsd:string"/>
      </xsd:choice>
   </xsd:complexType>
</xsd:element>
```

```
<!ELEMENT message (#PCDATA | bold | italic)*>
<!ELEMENT bold    (#PCDATA)>
<!ELEMENT italic  (#PCDATA)>
```

# Occurrence constraints

`minOccurs` and `maxOccurs`:

- They are two attributes by which we can specify the number of times an element or a group of elements appears.
- Both attributes are optional. Both have a default value of 1.

In combination, for an element or group `x`:

- `minOccurs="1"  maxOccurs="1"` is like `x`
- `minOccurs="0"  maxOccurs="1"` is like `x?`
- `minOccurs="1"  maxOccurs="unbounded"` is like `x+`
- `minOccurs="0"  maxOccurs="unbounded"` is like `x*`

# Attribute declarations

# About attributes

- Attributes are not treated as second class citizens – at least not to the extent they are in DTDs.
- We can define them seprately.
- We can define a simple type for them.
- Still, there are some odd features . . .

# Example 12: Attributes of simple elements

To attach an attribute to a *simple* element, we must *extend* the simple type. Suppose we have an element to be used like the following:

```
<shipment status="Who knows!">100 pencils</shipment>
```

```
<xsd:element name="shipment">
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base="xsd:string">
    <xsd:attribute name="status"
                   type="xsd:string"
                   use="required" />
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
```

# Example 13: Attributes of complex elements

Specifying the attributes of a complex element is straightforward. Consider the following element:

```
<person age="25" hair="dark" sex="m">
 <name>SI, Yain-Whar</name>
 <home>Macau</home>
</person>
```

We might define this element as follows

```
<xsd:element name="person">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="name"  type="xsd:string">
   <xsd:element name="home"  type="xsd:string">
  </xsd:sequence>
  <xsd:attribute name="age"  type="xsd:integer"/>
  <xsd:attribute name="hair" type="xsd:string"/>
  <xsd:attribute name="sex"  type="xsd:string"/>
 </xsd:complexType>
</xsd:element>
```

# Default and fixed values

# Default and fixed values

- In DTDs, attributes can be provided with fixed and default values.
- XML Schema extends this coverage to include elements as well.

# Attributes: default, fixed or optional

Attributes can express fixed, default and optional values.

```
<xsd:element name="person">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="name"  type="xsd:string">
   <xsd:element name="home"  type="xsd:string">
  </xsd:sequence>
  <xsd:attribute name="age"  type="xsd:integer"
                             use="required"/>
  <xsd:attribute name="hair" type="xsd:string"
                             default="red"/>
  <xsd:attribute name="sex"  type="xsd:string"
                             fixed="M"/>
 </xsd:complexType>
</xsd:element>
```

`use="required"` should not be used together with default/fixed.

# Example 14:

```
<xsd:element name="Printer"
             type="xsd:string"
             fixed="FIT-5009"/>
<xsd:element name="OS"
             type="xsd:string"
             default="Windows"/>
```

- `fixed`: if the element does appear, its value must be `FIT-5009`, and if it does not appear its value is `FIT-5009` (i.e., the element is created).
- `default`: if the element does not appear, it is not provided; if it does appear and it is empty, its value is `Windows`; otherwise its value is that given.

Here is an XML fragment:

1. `<Printer></Printer>`
2. `<OS/>`
3. `<OS>Windows</OS>`

After validation, it becomes:

1. `<Printer>FIT-5009</Printer>`
2. `<OS>Windows</OS>`
3. `<OS>Windows</OS>`

In DTD, it was not possible to achieve this for elements.

# The PhoneBook again!

# The Phonebook DTD

Here is the DTD we defined for the QUT phonebook:

```
<!DOCTYPE Phonebook [
<!ELEMENT Phonebook (Entry)+>
<!ELEMENT Entry (LastName, FirstName, School,
                           Campus, Room, Extension)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT School (#PCDATA)>
<!ELEMENT Campus (#PCDATA)>
<!ELEMENT Room (#PCDATA)>
<!ELEMENT Extension (#PCDATA)>
<!ATTLIST LastName
    Title (Miss | Ms | Mrs | Mr | Dr | Prof) #REQUIRED>
]>
```

# Example 15: A custom type for the ’Title’ attribute

This is a simple type for storing people’s titles:

```
<xsd:simpleType name="TitleType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Miss"/>
    <xsd:enumeration value="Ms"/>
    <xsd:enumeration value="Mrs"/>
    <xsd:enumeration value="Mr"/>
    <xsd:enumeration value="Dr"/>
    <xsd:enumeration value="Prof"/>
  </xsd:restriction>
</xsd:simpleType>
```

# Example 16: The ’LastName’ element

Although the `LastName` element is a *simple* one, we must *extend* its simple type to allow for the `Title` attribute:

```
<xsd:element name="LastName">
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base="xsd:string">
    <xsd:attribute name="Title"
                   type="TitleType"
                   use="required" />
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
```

# Some work still to do

We decided that three of the original eight requirements were not implemented:

1. A Campus must be GP or KG or CA.
2. An Extension must be a five digit number.
3. A Room must consist of a single upper-case letter followed by three digits.

# Example 17: The ’Campus’ element

Define an anonymous simple type for the Campus element. Restrict the campus to KG, GP or CA.

```
<xsd:element name="Campus">
  <xsd:simpleType>
    <xsd:restriction base="...............">
    <xsd:enumeration.............../>
    <xsd:enumeration.............../>
    <xsd:enumeration.............../>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Example 17: The ’Campus’ element

Define an anonymous simple type for the Campus element. Restrict the campus to KG, GP or CA.

```
<xsd:element name="Campus">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
    <xsd:enumeration value="KG"/>
    <xsd:enumeration value="GP"/>
    <xsd:enumeration value="CA"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Example 18: The 'Extension' element

We can require that the Extension element consist of *exactly* five digits in the following way:

```
<xsd:element name="Extension">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{5}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Example 19: The ’Room’ element

Declare the Room element and use an anonymous simple type that restricts the element to being a single alphabetic character followed by three digits.

```
<xsd:element name="Room">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:pattern value="..............."/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Example 19: The ’Room’ element

Declare the Room element and use an anonymous simple type that restricts the element to being a single alphabetic character followed by three digits.

```
<xsd:element name="Room">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:pattern value="[A-Z]\d{3}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

# Conclusions

# This week's topics

This week, we introduced the XML Schema language. In particular, we discussed:

- Built-in data types: an extensive range (44) of simple data formats.
- Custom-built data types using facets.
- Patterns
- Complex and simple types.
- Element declarations.
- Attribute declarations.

We also made some comparisons between XML Schema and DTDs.

# Next week's topics

- Next week, we look at some of the issues surrounding the use of namespaces in XML Schema and in instance documents.
- In particular, we examine four different XML Schema design patterns.
- We also examine a relatively rarely discussed XML Schema concept: the *substitution group*.
- Substitution groups are used extensively in XBRL.
- XBRL (the extensible business reporting language) is an important use of XML.

✓ `http://www.xfront.com/GlobalVersusLocal.html`

✓ `http://www.rpbourret.com/xml/NamespacesFAQ.htm`

✓ `http://devresource.hp.com/drc/resources/xmlSchemaBestPractices.jsp`