

Net-Based Applications

Chapter 7: XML Schema

Holger Schwarz
Universität Stuttgart

Winter Term 2016/2017

Overview

- Motivation and introduction
- Predefined data types
- User defined data types: simple and complex
- Uniqueness, keys and references
- Building Blocks: Schema inclusion and import
- OO Features: substitution groups, controlling derivations

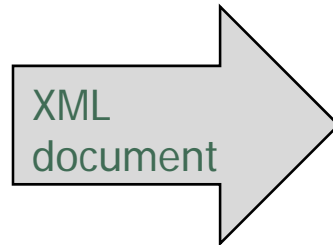
Why XML Schema?

- XML Schema is an alternative to DTD specified in XML syntax
- XML Schema is closer to the general understanding of a (database) schema as it supports ...
 - data types, e.g., integer, string, etc.
 - user defined types
 - constraints for element occurrence (min/max values)
 - typed references (for ID and IDREFS)
 - uniqueness and foreign key constraints

XML Standardization

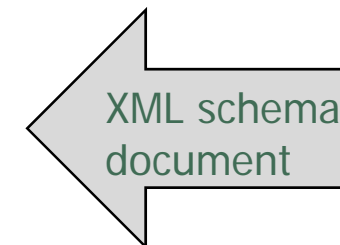
- XML Schema 1.0, W3C Recommendation, 28 October 2004
 - XML Schema Part 0: Primer, Second Edition
 - provides an easily readable description of the XML Schema facilities
 - XML Schema Part 1: Structures, Second Edition
 - defines facilities for describing the structure and constraining the contents of XML 1.0 documents
 - XML Schema Part 2: Datatypes, Second Edition
 - defines a set of built-in data types (primitive and derived)
- XML Schema 1.1, W3C Recommendation, 5 April 2012
 - Part 1: Structures
 - Part 2: Datatypes

Example



```
<?xml version="1.0"?>
<author id="CMS">
  <name> Charles M Schulz </name>
  <born> 1922-11-26 </born>
  <dead> 2000-02-12 </dead>
</author>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="name" type="xs:string"/>
  <xs:element name="born" type="xs:date"/>
  <xs:element name="dead" type="xs:date"/>
  <xs:attribute name="id" type="xs:ID"/>
  <xs:element name="author">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="born"/>
        <xs:element ref="dead" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute ref="id"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



Schema Location

- How to refer to a schema document from an XML document?

```
<?xml version="1.0"?>
```

```
<author id="CMS"
```

```
xmlns="http://dyomedeia.com/ns/library"
```

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

```
xsi:schemaLocation="http://dyomedeia.com/ns/library library.xsd">
```

```
<name> Charles M Schulz </name>
```

```
<born> 1922-11-26 </born>
```

```
<dead> 2000-02-12 </dead>
```

```
</author>
```

Namespace Identifier
for document

Namespace declaration
of XML Schema
for instances

Schema file
location

Namespace
Identifier

Schema Element

- Element **schema** is the root element of any XML schema
- Attribute **xmlns** allows to define ...
 - the namespace to be used for elements and types in the XML schema
 - the prefix that is used to refer to these elements and types
- Commonly used namespace identifier: xs or xsd

```
<?xml version="1.0" encoding="UTF-8"?>  
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  ...  
</xs:schema>
```

Target Namespace

people.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://dyomedea.com/ns/people">
  <xs:element name="author">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="born" type="xs:date"/>
        <xs:element name="dead" type="xs:date" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

namespace for
schema keywords

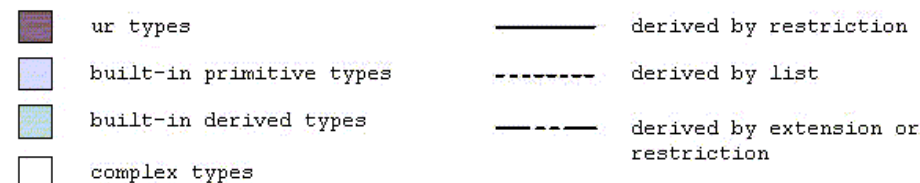
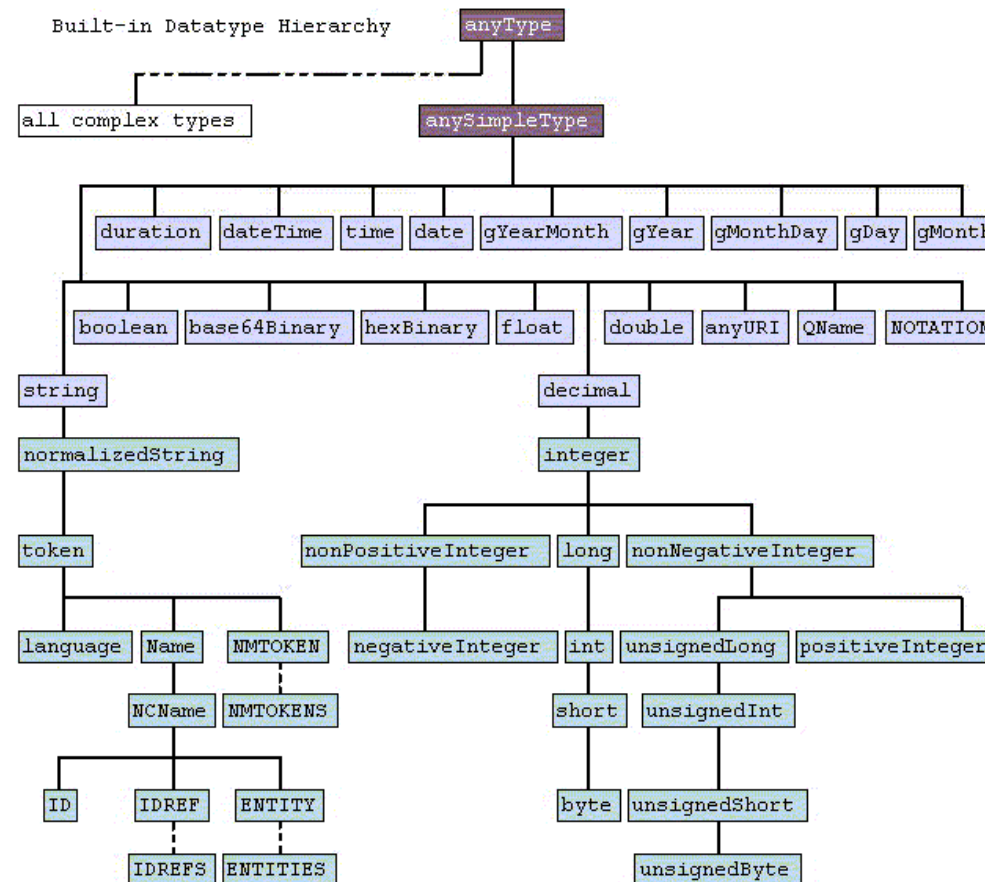
target namespace
of schema

```
<author id="CMS" xmlns="http://dyomedea.com/ns/people">
  <name>Charles M Schulz</name>
  <born>1922-11-26</born>
  <dead>2000-02-12</dead>
</author>
```


Type System

- **Ur-types** (anyType, anySimpleType)
 - root of the type hierarchy
 - rarely required in practice
- Extensive set of **built-in (predefined) data types**
 - smaller set of **primitive datatypes**
 - specific meaning and semantic
 - cannot be derived from other types
 - e.g.: xs:boolean, xs:string, xs:decimal, xs:float, xs:duration, xs:time, ...
 - larger set of **derived datatypes**
 - derived from other datatypes
 - e.g.: xs:integer (from xs:decimal),
xs:nonNegativeInteger (from xs:integer),
xs:name (from xs:string),
...
- **User defined types**

Type System



Overview

- Motivation and introduction
- Predefined data types
- User defined data types: simple and complex
- Uniqueness, keys and references
- Building Blocks: Schema inclusion and import
- OO Features: substitution groups, controlling derivations

Lexical Spaces and Value Spaces

- Each data type has its own lexical and value space
- **Lexical space**: data as read from the instance documents
- **Value space**: value interpreted according to the data type
- Multiple representations (lexical spaces) for the same value
- Example:
 - lexical space:
"3.14116", "03.14116", ".313116E1"
 - value space:
"3.14116" equals "03.14116" if interpreted as xs:float,
and is different if interpreted as xs:string

```
<pi>3.14116</pi>  
<pi>03.14116</pi>  
<pi>.313116E1</pi>
```

Whitespace Processing

- Whitespaces like tab, linefeeds, carriage returns, space, ... are inserted only to 'pretty print'
- Generic two-step algorithm in XML Schema
 - applied to most of the predefined data types

First step: whitespace replacement

- Every whitespace character (LF, CR, TAB, ...) is replaced with a space
- number of characters is not changed by this step
- applied to all data types except for xs:string

Second step: whitespace collapse

- removes leading and trailing spaces
- replaces contiguous occurrences of spaces by a single space char
- applied for all datatypes except for xs:string and xs:normalizedString

String Data Types

- Value space is typically identical to lexical space (except when explicitly described otherwise)
- `xs:string`
 - only predefined data type without whitespace replacement and collapsing
 - set of characters matching the XML 1.0 definition: "tab, carriage return, line feed and legal characters from Unicode and ISO/IEC 10646"
- `xs:normalizedString`
 - whitespace replacement (with space) without collapsing
- Binary string-encoded data types
 - encodings for binary `xs:hexBinary` and `xs:base64Binary`

Collapsed String Datatypes

Type	Description
xs:token	xs:normalizedString with collapsing
xs:language	language codes from RFC1766 (e.g. "fr-FR", "de", ...)
xs:NMTOKEN	'name token', single token (without spaces), allowed in XML name (i.e. as name of a tag)
xs:Name	like NMTOKEN but start with a letter or ":" or "-" (not with a number)
xs:NCName	'noncolonized name', xs:Name without ":"
xs:ID	derived from xs:NCName. No duplicate within a document
xs:IDREF	derived from xs:NCName. Has to match a ID value in the same document
xs:Entity	for compatibility with DTD, derived from xs:NCName, has to match an entity in a DTD
xs:QName	<ul style="list-style-type: none">• 'qualified name', for names with namespaces, Tuple with URI and Name.• lexical space is different from value space!• example: "xs:language" represents {"http://www.w3.org/2001/XMLSchema", "language"}
xs:anyURI	a URI. Also different lexical and value space because of transformation of non-ASCII characters

Numeric Data Types

- Build on four primitive data types
 - `xs:decimal`: decimal numbers of any length (includes integer)
 - `xs:float`: single precision floats
 - `xs:double`: double precision floats
 - `xs:boolean`: booleans (truth values, "0" / "1", "true" / "false")
 - all have collapsed whitespaces
- Some examples
 - `xs:long` : all integers that can be stored in a 64-bit word, i.e. from -9223372036854775808 to 9223372036854775807
 - `xs:int`, `xs:short`, `xs:byte` : same with 32-bit, 16-bit, 8-bit
 - valid values for `xs:float` and `xs:double` include "INF" (infinity, greater than any value), "-INF" (less than any value), "NaN" (not a number)

Point in Time

- `xs:dateTime`
 - point in time, a "specific instant of time"
 - format: CCYY-MM-DDThh:mm:ss (all specified)
optional followed by fractional digits for the seconds and the time zone ("Z" = UTC or +hh:mm)
 - valid examples
 - 2011-10-26T21:32:52
 - 2011-10-26T21:32:52Z
 - 2011-10-26T21:32:52+02:00
 - 2011-10-26T21:32:52.12384
 - invalid examples
 - 2011-10-26 (hours / minutes / seconds missing)
 - 2011-10-26T25:32:52 (25 is no valid hour)

Period of Time

- `xs:date`
 - lexical space: identical to the date part of `xs:dateTime`
 - value space: a date is a period of one day in its time zone
 - two dates in different time zones cannot be identical
 - Example: 2011-10-26, 2011-10-26+02:00
- `xs:gYearMonth`
 - `xs:date` without the day part (g = gregorian)
- `xs:gYear`
 - `xs:gYearMonth` without the month part

Recurring Points and Periods of Time

- `xs:time`
 - lexical space: identical to the time-part in `xs:dateTime`
 - semantics:
 - point in time that recurs every day
 - despite the fact that it is commonly used to represent a duration
 - accepts also an optional time zone definition
 - examples: `19:32:52Z`, `21:32:52.12679`
- `xs:gDay`
 - day, recurring each month
 - Lexical space: `---DD`, optional time zone
 - examples: `---15`, `---01+02:00`
- `xs:gMonth`
 - month, recurring each year
 - lexical space: `--MM`, optional time zone
- `xs:gMonthDay`
 - day, recurring each year
 - lexical space: `--MM-DD`, optional time zone
 - examples: `--05-01`, `--11-01Z`

Duration

- `xs:duration`
 - lexical space: PnYnMnDTnHnMnS (capitals are delimiters)
 - example: P1Y2MT123S – 1 year, 2 months, 123 seconds
 - but years have different number of days
→ such durations are not exactly defined!
 - if needed, define seconds: PT10023981823S

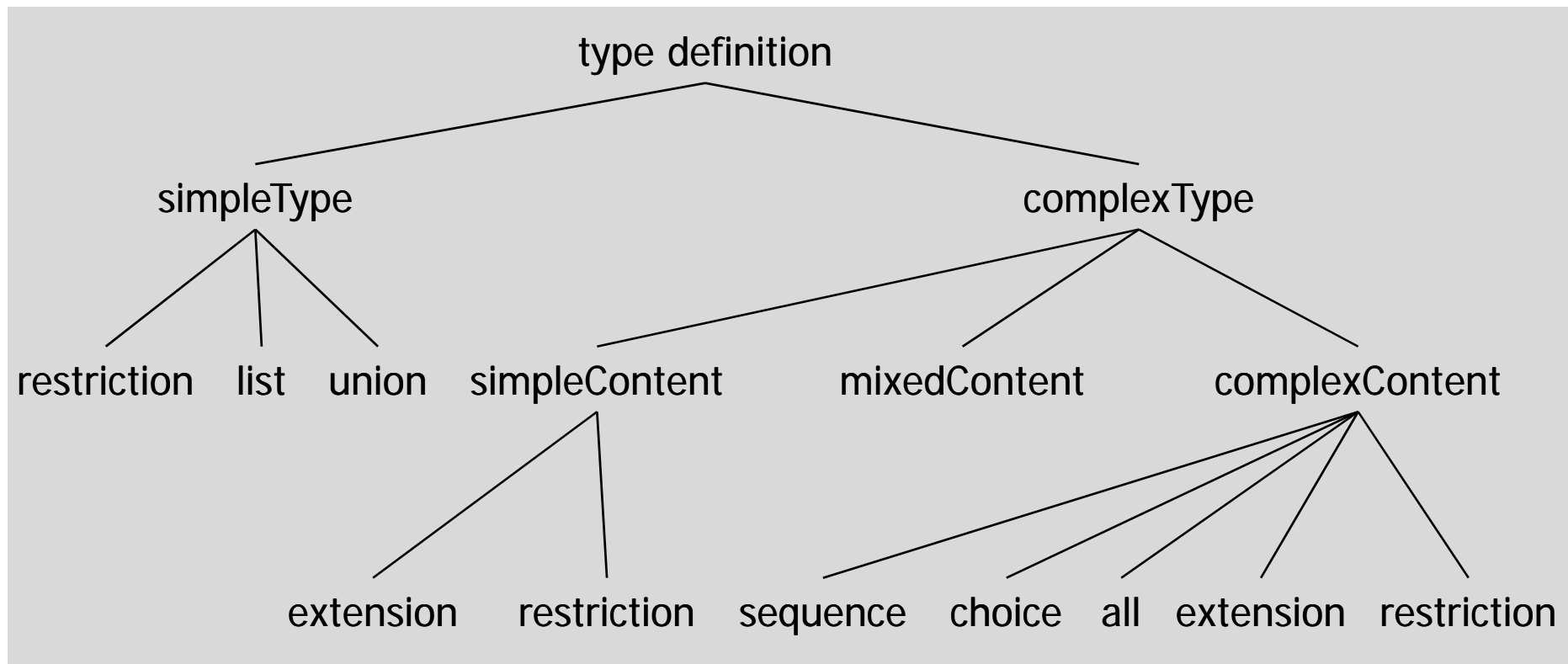
List types and anySimpleType

- List types
 - lists of whitespace-separated items
 - three predefined datatypes are list types
 - xs:NMTOKENS
 - xs:IDREFS
 - xs:ENTITIES
 - each member must be in the corresponding lexical space
- `xs:anySimpleType`
 - special type: wildcard, any value is accepted, no constraint on the lexical space
 - users cannot (directly) derive from this type
 - should be avoided

Overview

- Motivation and introduction
- Predefined data types
- User defined data types: simple and complex
- Uniqueness, keys and references
- Building Blocks: Schema inclusion and import
- OO Features: substitution groups, controlling derivations

User-defined Types – the Big Picture



Creating User-defined Datatypes

- Simple data types
 - describe the content of a text node or attribute value
 - independent from other nodes
- Complex data types
 - description of the markup structure
 - use simple data types for their leaf element nodes and attribute values, but no other links with simple data types
 - different content models:

	simple content	complex content	mixed content	empty content
child elements	no	yes	yes	no
text	yes	no	yes	no

Named vs. Anonymous Types

named types:

- define a data type with a name
- must have global scope
- can be used for multiple elements / attributes using the 'type' attribute

```
<xs:simpleType name="myType">  
  ...  
</xs:simpleType>  
  
<xs:element name="title" type="myType"/>
```

anonymous types:

- define type inside the definition for an element / attribute
- can replace most "type", "base", "itemType", "memberTypes" attributes

```
<xs:element name="title">  
  <xs:simpleType>  
    ...  
  </xs:simpleType>  
</xs:element>
```

references:

- define element / attribute with global scope
- reference it from anywhere (local scope) in the schema

```
<xs:element ref="title"/>
```

Creating Simple Data Types

- Derivation methods
 - derivation by restriction
 - most commonly used method
 - new data type is restricted by adding new constraints on different aspects (facets)
 - derivation by list
 - derive lists from data types
 - derivation by union
 - derive data types by merging the lexical spaces of other types
 - lose the semantic meaning of the types
- Order of derivation methods is significant
 - first do restriction on atomic or member types before derivation by list or union
 - list of unions is different from a union of lists

Derivation by Restriction: Example

```
<xs:simpleType name="myInteger">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="-2"/>
    <xs:maxExclusive value="5"/>
  </xs:restriction>
</xs:simpleType>
```

- definition of a simpleType called "myInteger" that has a range from -2 to 5

```
<xs:simpleType name="myInteger">
  <xs:restriction>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:maxExclusive value="5"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:minInclusive value="-2"/>
  </xs:restriction>
</xs:simpleType>
```

- same definition in two steps
- using an embedded xs:simpleType anonymous definition

Derivation by Restriction: Facets

- **Facets** are restrictions on certain aspects of the lexical or value space.
- Three categories
 1. `xs:whiteSpace` works in between parsing process and lexical space
 - defines the whitespace processing ("preserve", "replace", "collapse")
 - whitespace processing cannot be relaxed
 - example: `<xs:whiteSpace value="replace" />`
 2. `xs:pattern` works on the lexical space
 - defines a pattern that must be matched by the string
 - example: `<xs:pattern value="http://.*" />`
 - can be used to define data types based on **regular expressions**
 3. all other facets constrain the value space
- Some facets only work on certain data types
(e.g. `xs:totalDigits` only of `xs:decimal` and its derivations)

Derivation by Restriction: Facets (2)

- **xs:enumeration**: constrains the value space to certain values

```
<xs:simpleType name="schemaRecommendations">
  <xs:restriction base="xs:anyURI">
    <xs:enumeration value="http://www.w3.org/TR/xmlschema-0/" />
    <xs:enumeration value="http://www.w3.org/TR/xmlschema-1/" />
    <xs:enumeration value="http://www.w3.org/TR/xmlschema-2/" />
  </xs:restriction>
</xs:simpleType>
```

- **xs:length**: defines a fixed length measured in number of characters
- **xs:minLength**, **xs:maxLength**: defines the minimal/maximal length measured in number of characters

```
<xs:simpleType name="binaryImage">
  <xs:restriction base="xs:hexBinary">
    <xs:maxLength value="1024" />
  </xs:restriction>
</xs:simpleType>
```

Derivation by Restriction: Facets (3)

- `xs:minExclusive`, `xs:maxExclusive`, `xs:minInclusive`, `xs:maxInclusive`: define lower and upper bounds

```
<xs:simpleType name="maxExclusive10">
  <xs:restriction base="xs:float">
    <xs:maxExclusive value="10"/>
  </xs:restriction>
</xs:simpleType>
```

- `xs:totalDigits`: defines the maximum number of decimal digits

```
<xs:simpleType name="total5Digits">
  <xs:restriction base="xs:integer">
    <xs:totalDigits value="5"/>
  </xs:restriction>
</xs:simpleType>
```

- `xs:fractionDigits`: defines the maximum number of decimal digits in the fractional part (after the dot)

Multiple Restrictions

- Multiple restrictions on **different** facets
 - Intersection of lexical spaces / value spaces
- Multiple restrictions on the **same** facet
 - Adding more confining restrictions: `xs:enumeration`, `xs:(fraction/total)Digits`, `xs:(min/max)(In/Ex)clusive`, ...
 - `xs:length` cannot be changed
 - `xs:pattern`: intersection of lexical spaces

Attribute “fixed”

- Each facet includes a **fixed** attribute (except for xs:enumeration and xs:pattern)
- Disables the option to modify the facet during further restrictions

```
<xs:simpleType name="minInclusive10">  
  <xs:restriction base="xs:float">  
    <xs:minInclusive value="10" fixed="true"/>  
  </xs:restriction>  
</xs:simpleType>
```

→ all derivations of 'minInclusive10' must include the value of 10

Derivation by List or Union

- List data types
 - whitespace-separated list of values of a simple type
 - structured content in attributes
 - List members cannot be accessed via common XML APIs
 - compatibility with DTD's IDREFS, NMTOKENS, ENTITIES
 - redefines some facets

```
<xs:simpleType name="intList">  
  <xs:list itemType="xs:integer"/>  
</xs:simpleType>
```

- Derivation by union
 - union of simple data types
 - loses semantic meaning and most facets

```
<xs:simpleType name="intOrDate">  
  <xs:union memberTypes="xs:integer xs:date"/>  
</xs:simpleType>
```

- Try to avoid lists and unions

Creating Complex Data Types

- Can have attributes and / or child elements
- Creating complex data types “from scratch”
 - structures based on simple data types
- Derivation by extension
 - adding features (elements and attributes) to an existing type
- Derivation by restriction
 - subset of the possible instances of an existing type

Complex Types with Simple Content (1)

- Elements having only text content and attributes
- Derivation by extension
 - extension of an existing type by adding attributes

```
<xs:complexType name="stringWithLang">  
  <xs:simpleContent>  
    <xs:extension base="xs:string">  
      <xs:attribute name="lang" type="xs:language"/>  
    </xs:extension>  
  </xs:simpleContent>  
</xs:complexType>
```

```
<xs:complexType name="stringWithLangAndID">  
  <xs:simpleContent>  
    <xs:extension base="stringWithLang">  
      <xs:attribute name="id" type="xs:ID"/>  
    </xs:extension>  
  </xs:simpleContent>  
</xs:complexType>
```

```
<xs:element name="title" type="stringWithLangAndID"/>
```

attribute
declaration

```
<title lang="en-US" id="t1">  
  A Title in American English  
</title>
```

Complex Types with Simple Content (2)

- Attributes are declared with xs:attribute
 - "name"
 - "type" (unless declared inline)
 - "ref": alternative for type / name for local scope
 - "default" / "fixed": default / fixed value for the attribute
 - "use": "required", "optional", "prohibited" (only for local scope)

Complex Types with Simple Content (3)

- Derivation by restriction
 - allows to restrict the text content and/or the attributes
 - similar to restriction on simple types
 - same facets as for restrictions on simple types

```
<xs:element name="shortEnglishTitle">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="stringWithLangAndID">
        <xs:maxLength value="100"/>
        <xs:attribute name="lang">
          <xs:simpleType>
            <xs:restriction base="xs:language">
              <xs:pattern value="en.*"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="id" use="prohibited"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

length of text at
most 100

attribute "lang"
must refer to an
English language


attribute "id" may
not be used

Complex Type with Complex Content


- Elements having only child elements and attributes
- Declare possible child elements, their number and order
- **Compositors**
 - xs:sequence: ordered list of particles
 - xs:choice: choice of one particle among several
 - xs:all: unordered list of particles
- **Particles**
 - xs:element: an XML element
 - xs:any: any element (from a given namespace)
 - xs:group: (named) container for reusable content models
 - all compositors except xs:all
- Number of occurrences of particles specified by attributes
 - minOccurs: default "1"
 - maxOccurs: default "1", can be "unbounded"

Example (the "name"-Problem)


```
<xs:element name="author">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element ref="name"/>
        <xs:sequence>
          <xs:element ref="first-name"/>
          <xs:element ref="middle-name" minOccurs="0"/>
          <xs:element ref="last-name"/>
        </xs:sequence>
      </xs:choice>
      <xs:element ref="born"/>
      <xs:element ref="deceased" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute ref="id"/>
  </xs:complexType>
</xs:element>
```



```
<author id="a1">
  <first-name>
    Charles
  </first-name>
  <middle-name>
    M
  </middle-name>
  <last-name>
    Schulz
  </last-name>
  <born>
    1922-11-26
  </born>
</author>
```



```
<author id="a3">
  <name>Snoopy</name>
  <born>1950-02-12</born>
</author>
```



```
<author id="a2">
  <first-name>Peppermint</first-name>
  <last-name>Patty</last-name>
  <born>1950-02-12</born>
</author>
```

Limitations

- Unique particle attribution rule
 - XML Schema processors appear to use some kind of LR(0) algorithm
 - must be able to decide on the particle to use without look-ahead
- Limitations of xs:all
 - cannot be used as a particle, only as compositor
 - cannot have a number of occurrences greater than one
 - all particles must be xs:element
 - no particle may specify a number of occurrences greater than one
- Workarounds
 - adapt the structure of the document
 - use xs:choice instead of xs:all

```
<xs:element name="a">
  <xs:complexType>
    <xs:choice>
      <xs:sequence>
        <xs:element name="b" type="xs:string"/>
        <xs:element name="c" type="xs:string"/>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="b" type="xs:string"/>
        <xs:element name="d" type="xs:string"/>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Error!

Derivation by Extension

- Similar to the extension of simple content complex types
- Joining groups of elements and attributes to create a new complex type
 - Elements: sequence of compositor of base type followed by compositor of new type

```
<xs:complexType name="person">
  <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="born"/>
  </xs:sequence>
  <xs:attribute ref="id"/>
</xs:complexType>
<xs:element name="author">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="person">
        <xs:sequence>
          <xs:element ref="deceased"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

```
<author id="a1">
  <name>Charles M. Schulz</name>
  <born>1922-11-26</born>
  <deceased>2000-02-12</deceased>
</author>
```

Derivation by Restriction

- Restrict the number of instances of the base type
 - instances of the derived type are also instances of the base type
- Full definition of the content model required
- Unchanged attributes can be omitted

```
<xs:complexType name="person">
  <xs:sequence>
    <xs:element ref="name"/>
    <xs:element ref="born"/>
    <xs:element ref="deceased" minOccurs="0"/>
    <xs:element ref="qualification" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute ref="id"/>
</xs:complexType>
```

```
<xs:element name="author">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="person">
        <xs:sequence>
          <xs:element ref="name"/>
          <xs:element ref="born"/>
          <xs:element ref="deceased" minOccurs="0"/>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Mixed and Empty Content Models

- Mixed content model

- add-on to xs:complexType: "mixed" attribute
- allows any text node within the content model

```
<xs:complexType name="markedText" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="em" type="xs:token"/>
    <xs:element ref="a"/>
  </xs:choice>
  <xs:attribute ref="lang"/>
</xs:complexType>
```

- Empty content model: no special support required

- either a complex content model without compositors
- or a simple content model without text nodes

```
<xs:element name="br">
  <xs:complexType>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="class" type="xs:NMTOKEN"/>
  </xs:complexType>
</xs:element>
```

Overview

- Motivation and introduction
- Predefined data types
- User defined data types: simple and complex
- Uniqueness, keys and references
- Building Blocks: Schema inclusion and import
- OO Features: substitution groups, controlling derivations

Identity Constraints

- **unique**
 - asserts uniqueness, with respect to the content identified by a **selector**, of the tuples resulting from the evaluation of the **fields** XPath expression(s)
- **key**
 - asserts uniqueness as for unique. key further asserts that all selected content actually has such tuples
- **keyref**
 - asserts a correspondence, with respect to the content identified by a **selector**, of the tuples resulting from the evaluation of the **fields** XPath expression(s), with those of the **referenced key**
- Each constraint declaration has a name

Unique

```
<xs:element name="items" type="Items">
  <xs:unique name="constraintUniquePartNumAndName">
    <xs:selector xpath="item"/>
    <xs:field xpath="@partNum"/>
    <xs:field xpath="productName"/>
  </xs:unique>
</xs:element>
```

- Selector
 - identifies a node set as the scope of the constraint
 - XPath expression relative to the instance of the element being declared
- Field
 - XPath expression identifying attribute(s) or element(s) relative to each element selected by a selector
 - values of these attribute(s) or element(s) have to be unique
 - combinations of attributes and elements possible

Defining selector and field

- Restricted XPath expressions are used to define selector and field
- Extended BNF for **selector**

```
Selector ::= Path ( '|' Path )*  
Path      ::= ( './' )? Step ( '/' Step )*  
Step      ::= '.' | NameTest  
NameTest  ::= QName | '*' | NCName ':' '*'
```

- Extended BNF for **field** (only definition of Path is extended)

```
Selector ::= Path ( '|' Path )*  
Path      ::= ( './' )? Step ( '/' Step )* ( Step | '@' Nametest )  
Step      ::= '.' | NameTest  
NameTest  ::= QName | '*' | NCName ':' '*'
```

Keys and References

- Keys are special unique constraints
 - all fields must be specified in the instance document
 - key fields are non-nullable
- Reference to `xs:key` or `xs:unique`
 - uses same syntax as `xs:key` and `xs:unique` plus reference to key
 - selector and field denote the element / attribute containing the reference
 - declaration in the same element as the referenced key (or an ancestor)

```
<xs:element name="itemsAndOrders" type="itemsAndOrdersType">
  <xs:key name="itemKey">
    <xs:selector xpath="item"/>
    <xs:field xpath="@partNum"/>
  </xs:key>

  <xs:keyref name="foreignKey" refer="itemKey">
    <xs:selector xpath="order/part"/>
    <xs:field xpath="@partNumber"/>
  </xs:keyref>
</xs:element>
```


Identity Constraints vs. ID/IDREF Attributes

- Keep constraint definition separate from the type definition
- Not just attribute values, but also element content and combinations of values and content can be declared to be unique or key
- Scope of the constraint can be defined
- Keys are not only unique, but must always be present and are non-nillable
- comparison between keyref fields and key or unique fields is by value equality, not by string equality

Overview

- Motivation and introduction
- Predefined data types
- User defined data types: simple and complex
- Uniqueness, keys and references
- Building Blocks: Schema inclusion and import
- OO Features: substitution groups, controlling derivations

Modularised XML Schemas

- Schema inclusion
 - relocate declarations into external schema files
 - reuse external declarations in different schemas
 - all declarations belong to the same namespace
 - logical schema inclusion
 - includes XML schema documents
 - supports modification of included items
 - physical schema inclusion
 - “copy” into document
 - source document does not need to be an XML schema.
- Schema import
 - import items from other namespaces

Logical Schema Inclusion

- Includes all items with global scope into the global scope
- All items belong to the same namespace
 - included items without namespace are copied into the namespace of the including schema.
 - conflicting declarations must be avoided
 - most suitable for usage within one organisation
- Referenced file must be a valid schema
 - not necessarily usable stand-alone

simple.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="shortString">
    <xs:restriction base="xs:string">
      <xs:maxLength value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="simple.xsd"/>
  <xs:element name="text" type="shortString"/>
</xs:schema>
```

Logical Schema Inclusion with Redefinition

- Modify included types
 - new type is a derivation of the original type
 - original type declaration is replaced by new one
 - redefinition uses the same type name as the base type (forbidden anywhere else)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:redefine schemaLocation="simple.xsd">  
    <xs:simpleType name="shortString">  
      <xs:restriction base="shortString"> same name  
        <xs:maxLength value="10"/>  
      </xs:restriction>  
    </xs:simpleType>  
  </xs:redefine>  
  <xs:element name="text" type="shortString"/>  
</xs:schema>
```

Physical Inclusion

- Using XInclude for including
 - XML documents
 - fragments of XML documents
 - plain text files
- Basically copies text / trees into the document

facet.xsd

```
<xs:maxLength value="100" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xi="http://www.w3.org/2001/XInclude">
  <xs:simpleType name="shortString">
    <xs:restriction base="xs:string">
      <xi:include href="facet.xsd" parse="xml"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Schema Import (1)

- Combine components from different namespaces
 - no conflicting names
 - suited for importing schemas from other organisations
- Imports from global scope into global scope

library.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:l="http://dyomedea.com/ns/library"
  xmlns:p="http://dyomedea.com/ns/people"
  targetNamespace="http://dyomedea.com/ns/library"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <xs:import namespace="http://dyomedea.com/ns/people"
    schemaLocation="people.xsd"/>
```

...continued on next slide

prefix for new
namespace

prefix for imported
namespace

target namespace
of this schema

import
declarations
from other
namespace

Schema Import (2)

...continued from last slide

```
<xs:simpleType name="bookID">
  <xs:restriction base="xs:token"/>
</xs:simpleType>

<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isbn" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element ref="p:author"/>
    </xs:sequence>
    <xs:attribute name="id" type="l:bookID"/>
  </xs:complexType>
</xs:element>

<xs:element name="library">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="l:book" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

reference to



external namespace



same namespace

Schema Import (3)

reference to file containing the
schema for the namespace

```
<library xmlns="http://dyomedeia.com/ns/library"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dyomedeia.com/ns/library library.xsd">
  <book id="b0836217462">
    <isbn>0836217462</isbn>
    <title>Being a Dog Is a Full-Time Job</title>
    <author id="CMS" xmlns="http://dyomedeia.com/ns/people">
      <name>Charles M Schulz</name>
      <born>1922-11-26</born>
      <dead>2000-02-12</dead>
    </author>
  </book>
</library>
```

References to XML Schema Files

- Two different “schemaLocation” attributes
- Schema inclusion
 - schemaLocation is required
- Schema import
 - schemaLocation is optional
- Instance document
 - xsi:schemaLocation is optional
 - xsi:schemaLocation is a list of pairs of namespace URIs and schema file locations
 - xsi:noNamespaceSchemaLocation for referencing XML schemas without namespace
- Optional schemaLocation
 - parsers identify schemas by their namespace URI
 - parsers are not required to read schemas from the schemaLocation
 - pros: applications do not depend on the schema being available at a certain location
 - cons: once the schema is released to the public, it cannot be changed

Overview

- Motivation and introduction
- Predefined data types
- User defined data types: simple and complex
- Uniqueness, keys and references
- Building Blocks: Schema inclusion and import
- OO Features: substitution groups, controlling derivations

Substitution Groups (1)

- Set of elements (**members**) that can be used in place of a certain element (**head**)
- All elements must be declared with global scope
- Trees of substitution groups
 - a member of a substitution group can be the head of an other substitution group
- Type of a member must be derived of the type of the head (or be the same type)
 - all types are derived from anyType

Substitution Groups (2)

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="name"/>
  <xs:element name="simple-name" type="xs:string" substitutionGroup="name"/>
  <xs:element name="full-name" substitutionGroup="name">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="first" type="xs:string"/>
        <xs:element name="middle" type="xs:string" minOccurs="0"/>
        <xs:element name="last" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="names">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="name"
          minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

```

<names>
  <simple-name>Charles M Schulz</simple-name>
  <full-name>
    <first>Charles</first>
    <middle>M</middle>
    <last>Schulz</last>
  </full-name>
</names>

```

 head members

Alternative Design: Groups and Choices

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:group name="name">
    <xs:choice>
      <xs:element ref="simple-name"/>
      <xs:element ref="full-name"/>
    </xs:choice>
  </xs:group>
  <xs:element name="simple-name" type="xs:string"/>
  <xs:element name="full-name">
    ... same as previous example...
  </xs:element>
  <xs:element name="names">
    <xs:complexType>
      <xs:sequence>
        <xs:group ref="name" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Deciding on Alternatives

Feature	xs:group + xs:choice	substitution group
Definition	Centralised	Spread over element definitions
Constraints on choices	None	Must be derived from head
Scope of elements	global or local	global
Remove choices	Yes (xs:redefine)	(No)
Add choices	(No)	Yes

Controlling Derivations (1)

- Confining derivations: attribute **final**
 - in type declarations: prohibits derivation by specified type
 - simple types: subset of {"restriction", "list", "union"} or "#all"
 - complex types: subset of {"restriction", "extension"} or "#all"
 - in element declarations: prohibits substitutions
 - similar to complex types declarations: subset of {"restriction", "extension"} or "#all"
- Prohibiting instantiation
 - attribute **abstract** in element declarations (boolean)
 - useful for heads of substitution groups

Controlling Derivations (2)

not reasonable,
but possible

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="name" abstract="true" final="extension"/>
  <xs:element name="simple-name" type="xs:string" substitutionGroup="name"
    final="extension restriction"/>
  <xs:element name="full-name" substitutionGroup="name" final="#all">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="first" type="xs:string"/>
        <xs:element name="middle" type="xs:string" minOccurs="0"/>
        <xs:element name="last" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="names">
    ... same as first example...
  </xs:element>
</xs:schema>
```

Conclusion

- Predefined datatypes:
 - xs:string, xs:boolean, ... only simple datatypes
- User defined datatypes:
 - simple datatypes: elements with attributes
 - complex datatypes: structure of elements
 - different derivation methods! (extension, restriction, ...)
- Building Blocks
 - schema inclusion: within the same namespace, control derivation
 - schema import: from other namespaces
- OO Features: substitution groups, controlling derivations
- Further information
 - W3C: XML Schema Part 0: Primer. <http://www.w3.org/TR/xmlschema-0/>.
 - Eric van der Vlist: XML Schema. O' Reilly.

Literature & Information



- World Wide Web Consortium: XML Schema Part 0: Primer, Second Edition, W3C Recommendation 28 October 2004.
<http://www.w3.org/TR/xmlschema-0/>
- World Wide Web Consortium: XML Schema Part 1: Structures, Second Edition, W3C Recommendation 28 October 2004.
<http://www.w3.org/TR/xmlschema-1/>
- World Wide Web Consortium: XML Schema Part 2: Datatypes, Second Edition, W3C Recommendation 28 October 2004.
<http://www.w3.org/TR/xmlschema-2/>