

NPRG036

# XML Technologies

---



Lecture 5

## XML Schema

26. 3. 2018

Author: **Irena Holubová**

Lecturer: **Martin Svoboda**

<http://www.ksi.mff.cuni.cz/~svoboda/courses/172-NPRG036/>

---

# Lecture Outline

---

- XML schema languages
  - **XML Schema**
    - Elements, attributes
    - Simple types, complex types
    - Advanced constructs
-


# A Schema of an XML Document

---

- Well-formedness vs. validity
  - A schema of an XML document
    - Description of allowed structure of XML data
    - Description of elements, attributes and their mutual relationship
  - Tools for definition of the structure:
    - DTD (Document Type Definition)
    - XML Schema – W3C
    - Schematron, RELAX NG, ... – ISO standards
  - Tools for validity checking: XML validators
-

# Terminology

---

- XML schema = allowed structure of XML data in any available language
    - DTD, XML Schema, RELAX NG, Schematron, ...
  - XML Schema = one of the languages
    - „XML schema in XML Schema”
    - XSD = XML Schema definition
      - Counterpart for DTD
  - There are other options: XML-schema, XML-Schema, XML-schema, Xschema, ...
    - In Czech: much more options
-



# XML Schema – Advantages

---

- Does not require special syntax
  - XSDs = XML documents
- Strong support for data types
  - A set of built-in data types (e.g. Boolean, date, ...)
  - User-defined data types
- We can (easily) express number of occurrences of elements

```
<!ELEMENT person (name, e-mail, e-mail?, e-mail?, e-mail?,  
e-mail?, relations?)>
```

# XML Schema – Advantages

---

- We can define elements with the same name but different content
  - In DTD we can not – all elements are defined at the same level
- We can define empty elements and elements which can be specified without content
- We can specify the exact structure of mixed-content elements
- We can (easily) express unordered sequences

```
<!ELEMENT name ((first, surname)|(surname, first))>
```

---

# XML Schema – Advantages

---

- We can re-use various parts of the schema
    - Data types, sets of elements, sets of attributes, ...
    - Object-oriented features
  - Keys and references
    - Specification of context
    - Combination of elements and/or attributes
  - We can define the same thing in various ways
  - Preserves DTD structures
    - Except for entities
-

# XML Schema – Advantages = Disadvantages

---

- ❑ Does not require special syntax
  - XSDs are long and less lucid than DTDs
  - More complex schemas are difficult to understand
  - Complex description
    - ❑ Elements and attributes are defined using elements and attributes
- ❑ We can define the same thing in various ways
  - An advantage for the user
  - A disadvantage for processing

xhtml.xsd





# XML Schema – Specification

---

- Version 1.0:
    - Part 0: Primer <http://www.w3.org/TR/xmlschema-0/>
      - Not a specification, a set of examples and explanations
    - Part 1: Structures <http://www.w3.org/TR/xmlschema-1/>
      - Structures of the language
    - Part 2: Datatypes <http://www.w3.org/TR/xmlschema-2/>
      - Built-in data types
  - Version 1.1:
    - Part 1: Structures
      - <http://www.w3.org/TR/xmlschema11-1/>
    - Part 2: Datatypes
      - <http://www.w3.org/TR/xmlschema11-2/>
-

# XML Schema – Basics

---

- XSD = a well-formed XML document
  - XML declaration, root element, ...
  - Validity against XSD of XML Schema language

```
<?xml version="1.0" encoding="windows-1250"?>  
<schema ...>  
  ... <!-- XML schema definition --> ...  
</schema>
```

- Components of the language = elements
    - Features – subelements/attributes
    - Defined in XML Schema namespace
-

# XSD vs. XML Document

---

```
<?xml version="1.0" encoding="windows-1250"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ... <!-- XML schema definition --> ...
</xs:schema>
```

```
<?xml version="1.0" encoding="windows-1250"?>
<root_element_of_XML_document
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schema2.xsd">
  ... <!-- XML document --> ...
</root_element_of_XML_document>
```

- XML Schema namespace
  - Namespace of XML Schema instances = XML documents valid against an XSD
  - URL of XSD file
-

# Root element of XML document?

---

- Any **globally defined element** can be the root element of an XML document
    - Globally defined = direct subelement of element **schema**
  - Globally defined components have in XML Schema special behaviour
    - Elements, attributes, data types, sets of elements, sets of attributes
      - Elements → root elements, ...
      - Other → can be used repeatedly, ... (see later)
-

# Root Elements – Example (1)

---

```
<?xml version="1.0" encoding="windows-1250"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="employees">
    <!-- definition of content -->
  </xs:element>

  <xs:element name="person">
    <!-- definition of content -->
  </xs:element>

  <!-- definition of other elements -->
</xs:schema>
```

# Root Elements – Example (2)

---

```
<?xml version="1.0" encoding="windows-1250"?>
<employees
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="emp.xsd">
  <!-- element content -->
</employees>
```

```
<?xml version="1.0" encoding="windows-1250"?>
<person
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="emp.xsd">
  <!-- element content -->
</person>
```

---

# How to Work with XML Schema

---

- XML schema definition:
    - Definition of data types
    - Definition of elements and attributes
      - Name + data type
  - Components of the language:
    - **Basic** – simple data type, complex data type, element, attribute, set of elements, set of attributes
    - **Advanced** – identity restriction, substitution groups, wildcards, external schemas, ...
  - „Kit“ – we build complex components from simpler ones
    - We can extend, restrict, refer, ...
-

---

# Basic Components

---



Note: In the following examples we omit XML declaration and element schema.

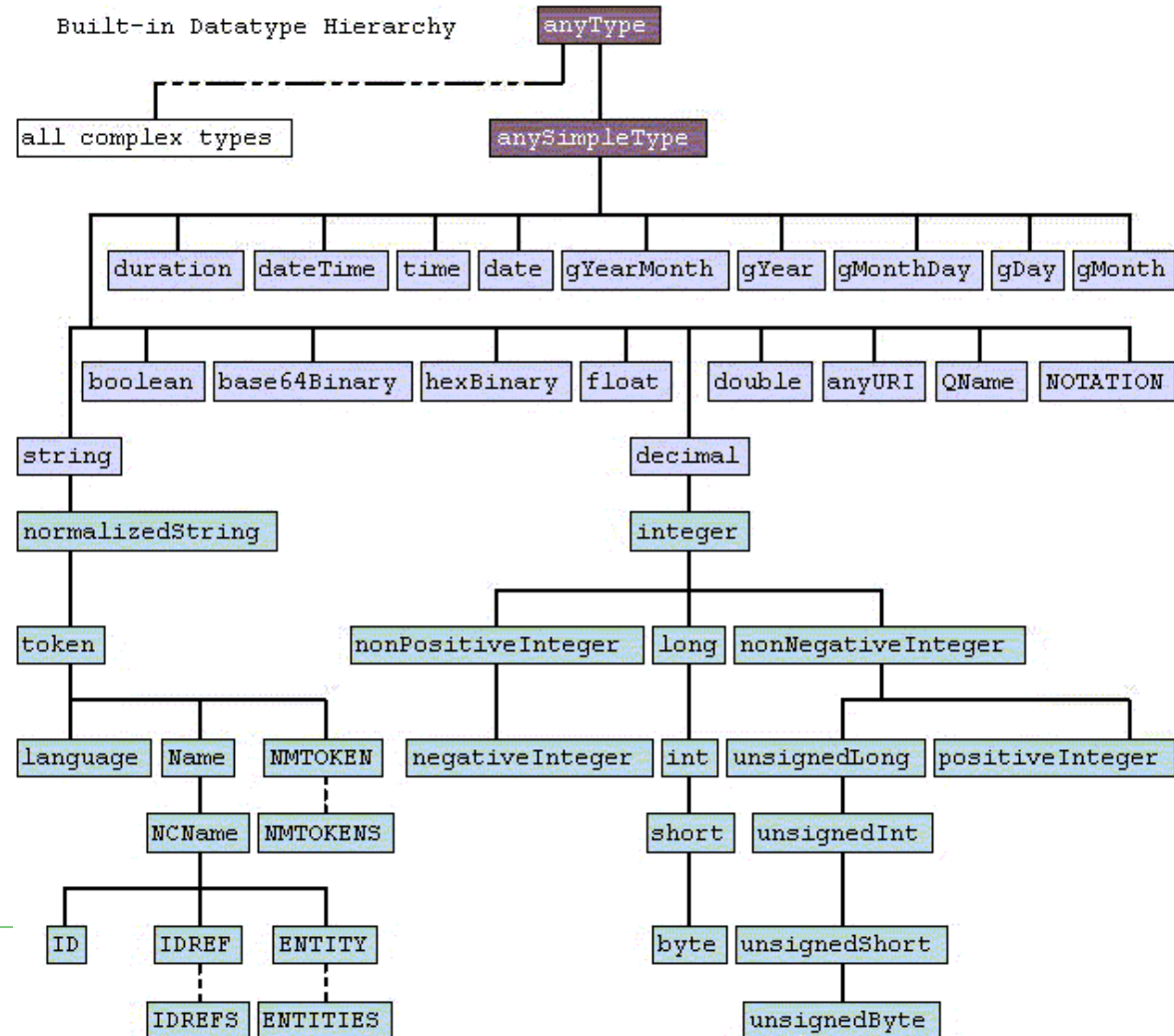
# Simple Data Types

---

- Element content or attribute value is always textual
  - Simple data type = restriction of textual values to a particular subset
  - Types:
    - Built-in – pre-defined
      - see Specification Part 2: Datatypes
    - User-defined – specified by the user
      - Derived from other data types
-

# Built-in Data Types

- Direct part of XML Schema
- Hierarchy:



# Primitive Data Types (1)

---

- **string** – a sequence of characters
  - **boolean** – true, false, 1, 0
  - **decimal** – 0, positive or negative real value
    - e.g. -1.23, 1267.5433, 210
  - **float** – 32-bit floating point data type
    - Values:  $m \times 2^e$ , where  $|m| < 2^{24}$ ,  $-149 \leq e \leq 104$
    - e.g. -1E4, 1267.43233E12, 12
    - Special values: 0, -0, Inf, -Inf, +Inf, NaN
  - **double** – 64-bit floating point data type
    - Values:  $m \times 2^e$ , where  $|m| < 2^{53}$ ,  $-1075 \leq e \leq 970$
-

# Primitive Data Types (2)

---

- **duration** – time interval of the form  $PnYnMnDTnHnMnS$ , where  $P$  and  $T$  are delimiters,  $nY$  means  $n$  years,  $nM$  means  $n$  months etc.
    - e.g. -P13Y7M, P2Y1MT2H
  - **dateTime** – date and time of the form  $YYYY-MM-DDT\text{hh:mm:ss.ss}$ , where  $T$  is a delimiter
  - **time** – time of the form  $\text{hh:mm:ss.ss}$
  - **date** – date of the form  $YYYY-MM-DD$
  - **gYearMonth** – year and month of the form  $YYYY-MM$
  - **gYear** – year of the form  $YYYY$
  - **gMonthDay** – month and day of the form  $--MM-DD$
  - **gMonth** – month of the form  $--MM$
  - **gDay** – day of the form  $---DD$
-

# Primitive Data Types (3)

---

- **hexBinary** – hexadecimal number
  - **base64Binary** – binary data in Base64 encoding (idea „similar" to hexadecimal encoding)
  - **anyURI** – absolute or relative URI
  - **QName** – XML Qualified Name, i.e. string of the form prefix:local\_name
    - see namespaces
  - **NOTATION** – reference to notation
    - see notations
-

# String Simple Data Types (1) – Derived from `string`

---

- `normalizedString` – string which does not contain characters CR, LF a tabulator
  - `token` – `normalizedString` which does not contain leading or trailing spaces or internal sequences of two or more spaces
  - `language` – language identifier
    - Allowed values are given by a standard
    - e.g. en, en-GB
  - `Name` – XML Name, i.e. string which can contain letters, numbers and characters '-', '\_', ':' and '.'
  - `NCName` – XML Name without character ':'
-

# String Simple Data Types (2) – Derived from **string**

---

- NMTOKEN
  - NMTOKENS
  - ID
  - IDREF
  - IDREFS
  - ENTITY
  - ENTITIES
    - Note: Entities can be defined only in DTD
-

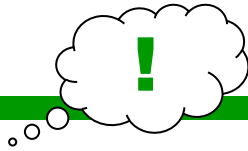
# String Simple Data Types – Derived from `decimal`

---

- `integer`
  - `positiveInteger`
  - `negativeInteger`
  - `nonPositiveInteger`
  - `nonNegativeInteger`
  - `long` – integer from  $\langle -2^{63}, 2^{63}-1 \rangle$
  - `int` – integer from  $\langle -2^{31}, 2^{31}-1 \rangle$
  - `short` – integer from  $\langle -2^{15}, 2^{15}-1 \rangle$
  - `byte` – integer from  $\langle -2^7, 2^7-1 \rangle$
  - `unsignedLong` – non negative integer less than  $2^{64}$
  - `unsignedInt` – non negative integer less than  $2^{32}$
  - `unsignedShort` – non negative integer less than  $2^{16}$
  - `unsignedByte` – non negative integer less than  $2^8$
-



# Simple Data Types – Notes



- Time data types can be defined in **UTC** (Coordinated Universal Time) possibly with offset
  - e.g. 15:30:25Z, 09:30:25+06:00
- Time data types can be **negative**
- Built-in data types with capital letters
  - Correspond to respective DTD data types
  - Can be assigned just with attributes

# User-defined Simple Data Types (simpleType)

---

- Enables to define own data types
- Attributes:
  - name – (optional) name of the data type
  - final – forbids further derivation
    - restriction, union, list, #all
- Derived from another (built-in / user-defined) data type via
  - restriction
  - union
  - list

---

`(annotation?, (restriction | list | union))`

# Derivation using restriction

---

- Restricts values of the original data type using a specified rule
- The restriction must make sense for the original data type, i.e. not everything is allowed
- Attributes:
  - base – restricted data type
    - Or specified using subelement simpleType

```
(annotation?, (simpleType?, (minExclusive |  
minInclusive | maxExclusive | maxInclusive |  
totalDigits | fractionDigits |  
length | minLength | maxLength |  
enumeration | whiteSpace | pattern)*))
```

---

# Derivation using **restriction** – Example

---

```
<xs:simpleType name="Ports">
  <xs:restriction base="xs:integer">
    <xs:enumeration value="111"/>
    <xs:enumeration value="21"/>
    <xs:enumeration value="80"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="NonEmptyString" final="#all">
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
    <xs:maxLength value="10"/>
  </xs:restriction>
</xs:simpleType>
```

---

# Derivation using restriction – Example

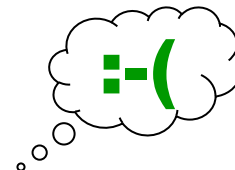
---

```
<xs:element name="PortNumber" type="Ports"/>
```

```
<xs:element name="ServerName" type="NonEmptyString"/>
```

```
<PortNumber>111</PortNumber>
```

```
<ServerName>kocour</ServerName>
```



```
<PortNumber>112</PortNumber>
```

```
<PortNumber>hi</PortNumber>
```

```
<ServerName/>
```

```
<ServerName>kocour.ms.mff.cuni.cz</ServerName>
```

---

# Allowed Types of Restrictions (1)

---

- **length** – the number of items of a particular data type
  - **minLength** – the minimum number of items of a particular data type
  - **maxLength** – the maximum number of items of a particular data type
  - **pattern** – regular expression describing items of the data type
    - Operators: . (any character) \ (escape or meta character) ? \* + | () (group) {} (repetition) [] (interval), \s (white space) \S (non white space) \d (number) \n \t
    - Example. `\*d*\*` ... `"*1234*"`, `a{2,4}` ... `"aaa"`, `(\d|[A-Z])+` ... `"3"`, `"U2"`
  - **enumeration** – a set of values
  - **maxInclusive** – values  $\leq$  specified value
  - **minInclusive** – values  $\geq$  specified value
  - **maxExclusive** – values  $<$  specified value
  - **minExclusive** – values  $>$  specified value
  - **totalDigits** – maximum number of digits
  - **fractionDigits** – maximum number of fraction digits
-

# Allowed Types of Restrictions (2)

---

- **whiteSpace** – processing of whitespaces
  - preserve – no changes
  - replace – characters CR, LF and tabulator are replaced with a space
  - collapse – in addition, all leading and trailing whitespaces are removed and sequences of whitespaces are replaced with a single one

```
<xs:simpleType name="nameWithCapitalLetters">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"/>
    <xs:pattern value="([A-Z]([a-z])*)?"/>
  </xs:restriction>
</xs:simpleType>
```

---

# Derivation using **list**

---

- Creates a list of values of the original data type delimited using whitespaces
  - Problem: list of strings vs. white space delimiters
- Attributes:
  - itemType – original type
    - Or specified using subelement simpleType
- Multivalue data types
  - We cannot derive from other multivalue data types
    - i.e. create a list of lists
  - NMTOKENS, IDREFS, ENTITIES

---

(annotation?, simpleType?)



# Derivation using **list** – Example

---

```
<xs:simpleType name="ListOfFloats">
  <xs:list itemType="xs:float"/>
</xs:simpleType>

<xs:element name="Temperatures"
  type="ListOfFloats"/>
```

```
<Temperatures>11 12.5 10.2</Temperatures>
<Temperatures>-3.14 0 -1.5</Temperatures>
```

---

# Derivation using union

---

- ❑ Creates a union of values of original data types
- ❑ Attributes:
  - memberTypes – original data types
    - ❑ Or specified using subelements simpleType

```
<xs:simpleType name="NonZeroIntegers">  
  <xs:union memberTypes="xs:positiveInteger  
    xs:negativeInteger"/>  
</xs:simpleType>  
  
<xs:element name="Temperature" type="NonZeroIntegers"/>
```

```
<Temperature>11</Temperature>  
<Temperature>-3</Temperature>  
<Temperature>10</Temperature>
```

---

(annotation?, simpleType\*)

# Globally vs. Locally Defined Simple Types

---

```
<xs:simpleType name="TypeZeroTo100">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:positiveInteger">
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="100"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="zero"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

# Attributes (**attribute**)

---

- Name + simple data type
  - Built-in – value of attribute **type**
  - Globally defined – value of attribute **type**
  - Locally defined – subelement **simpleType**

---

`(annotation?, simpleType?)`

# Attributes – Example

---

```
<xs:attribute name="Age" type="xs:positiveInteger"/>
```

```
<xs:attribute name="Name" type="NonEmptyString"/>
```

```
<xs:attribute name="PhoneNumber">
```

```
  <xs:simpleType>
```

```
    <xs:restriction base="xs:string">
```

```
      <xs:pattern value="\d{3}-\d{6}"/>
```

```
    </xs:restriction>
```

```
  </xs:simpleType>
```

```
</xs:attribute>
```

```
<person Age="30"
```

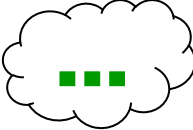
```
  Name="H. Simpson"
```

```
  PhoneNumber="123-445566"/>
```

---

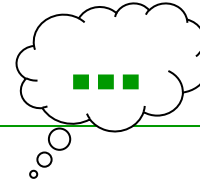
# Attributes

---

- Attributes:
    - default – implicit value
    - fixed – fixed value
    - use – occurrence
      - optional, required, prohibited (?)
  - Attributes can be also defined globally / locally
    - In both the cases it has a name
    - Globally – element **attribute** is a subelement of element **schema**
      - We can refer to it using references° ○ ○ 
    - Locally – within a definition of a complex type or a set of attributes
      - Just local usage
-

# Elements (**element**)

---



- Name + simple / complex data type
  - Simple type – element without attributes with a text content
    - Built-in – value of attribute **type**
    - Globally defined – value of attribute **type**
    - Locally defined – subelement **simpleType**
  - Complex type – other types of elements
    - Globally defined – value of attribute **type**
    - Locally defined – subelement **complexType**
- Enables to define keys/references
  - unique, key, keyref – see later

---

```
(annotation?, ((simpleType | complexType)?,  
                (unique | key | keyref)*))
```

# Elements – Example

---

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

```
<xs:element name="surname">
```

```
  <xs:simpleType>
```

```
    <xs:restriction base="xs:string">
```

```
      <xs:minLength value="2"/>
```

```
    </xs:restriction>
```

```
  </xs:simpleType>
```

```
</xs:element>
```

```
<name>Marge</name>
```

```
<surname>Simpson</surname>
```

---



# Elements

---

- Attributes:
  - nillable – possible empty content
  - default – implicit value
    - Only for elements with text content
  - fixed – fixed value
    - Only for elements with text content
- Elements can be also defined globally / locally
  - In both the cases it has a name
  - Globally – element **element** is a subelement of element **schema**
    - We can refer to it using references
    - Root elements of XML documents
  - Locally – within a definition of a complex type
    - Just local usage



# Complex Data Types (**complexType**)

---

- For definition of more complex types of elements
  - Relations element-subelement and element-attribute
  - Numbers and order of subelements
  - Since version 1.1: Conditions for values of subelements/attributes
    - Using XPath
    - assert – see later
- Consists of:
  - Specification of content
    - empty = an empty element
  - Specification of attributes
    - empty = an elements without attributes

```
(annotation?, (simpleContent | complexContent |  
                ((group | all | choice | sequence)?,  
                ((attribute | attributeGroup)*, anyAttribute?),  
                assert* )))
```

# Complex Data Types – Example

---

```
<xs:complexType name="TypeAddress">
  <!-- specification of content -->
  <xs:sequence>
    <xs:element name="Street" type="xs:string"/>
    <xs:element name="Number" type="xs:integer"/>
    <xs:element name="City" type="xs:string"/>
  </xs:sequence>

  <!-- specification of attributes -->
  <xs:attribute name="Country" type="xs:NMTOKEN"
                default="CZ"/>
</xs:complexType>

<xs:element name="Address" type="TypeAddress"/>
```

# Complex Data Types – Example

---

```
<Address>  
  <Street>Blue Street</Street>  
  <Number>25</Number>  
  <City>Praha 1</City>  
</Address>
```

```
<Address Country="SK">  
  <Street>Red Street</Street>  
  <Number>6</Number>  
  <City>Bratislava 16</City>  
</Address>
```

# Complex Data Types

---

- Attributes:
    - mixed – an element with mixed content
  - Can be also defined globally / locally
    - Usage same as in case of simple types
  - Types of content:
    - I. with a simple textual content (**simpleContent**)
    - II. sequence of components (**sequence**)
    - III. choice of components (**choice**)
    - IV. unordered sequence of elements (**all**)
    - V. model group (**group**)
    - VI. with a complex content (**complexContent**)
-

# I. Simple Content (**simpleContent**)

---

- The content of element is a simple type + attributes
- Derivation:
  - **extension** – adding attributes
  - **restriction** – adding attributes + type restriction

```
<xs:complexType name="Type">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="Subtype" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

---

(annotation?, (restriction | extension))

```
<xs:complexType name="CarType">
  <xs:simpleContent>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="VW"/>
      <xs:enumeration value="BMW"/>
      <xs:attribute name="Subtype" type="xs:string"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

```
<xs:element name="Vehicle" type="Type"/>
<xs:element name="Car" type="CarType"/>
```

```
<Vehicle Subtype="mountain">bicycle</Vehicle>
<Car Subtype="TT">Audi</Car>
```

---

## II. Sequence of Items (**sequence**)

- The content is formed by all the specified items in the given order

```
<xs:complexType name="person">
  <xs:sequence>
    <xs:element name="name" type="xs:string"
      maxOccurs="5"/>
    <xs:element name="surname" type="xs:string"/>
    <xs:element name="born" type="xs:date"/>
    <xs:element name="note" type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Id" type="xs:ID"/>
</xs:complexType>
```

(annotation?, (element | group | choice | sequence | any)\*)

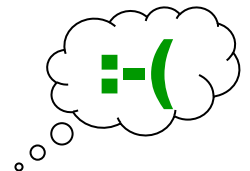


# Sequence of Items

---

```
<xs:element name="Attendee" type="person"/>
```

```
<Attendee Id="1234">  
  <name>Charles</name>  
  <surname>De Gaulle</surname>  
  <born>1848-07-04</born>  
  <note>leader</note>  
</Attendee>
```



```
<Attendee Id="4567">  
  <born>1850-12-12</born>  
  <name>Jean</name>  
  <surname>Moulin</surname>  
</Attendee>
```

# III. Choice of Items (**choice**)



- The content is formed by one of the specified items

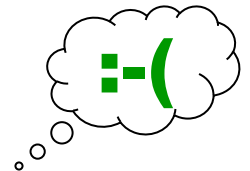
```
<xs:complexType name="TypePriceList">
  <xs:choice>
    <xs:sequence>
      <xs:element name="BasicPrice" type="xs:string"/>
      <xs:element name="FullPrice" type="xs:string"/>
    </xs:sequence>
    <xs:element name="BargainPrice" type="xs:string"/>
    <xs:element name="StudentPrice" type="xs:string"/>
  </xs:choice>
</xs:complexType>
```

(annotation?, (element | group | choice | sequence | any)\*)

# Choice of Items

```
<xs:element name="Price" type="TypePriceList"/>
```

```
<Price>  
  <BasicPrice>1170EUR</BasicPrice>  
  <FullPrice>1500EUR</FullPrice>  
</Price>  
<Price>  
  <StudentPrice>1000EUR</StudentPrice>  
</Price>
```



```
<Price>  
  <FullPrice>1500EUR</FullPrice>  
  <BasicPrice>1170EUR</BasicPrice>  
</Price>  
<Price>  
  <FullPrice>1500EUR</FullPrice>  
</Price>
```

# IV. Unordered Sequence of Elements (all)

---

- The content is formed by the specified elements in an arbitrary order



```
<xs:complexType name="TypeBook">
  <xs:all>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Author" type="xs:string"/>
    <xs:element name="ISBN" type="xs:string"
      minOccurs="0"/>
  </xs:all>
</xs:complexType>
```

---

(annotation?, (element | any)\*)

# Unordered Sequence of Elements

---

```
<xs:element name="Book" type="TypeBook"/>
```

```
<Book>  
  <Name>The King's Speech</Name>  
  <Author>Mark Logue</Author>  
  <ISBN>123-456-789</ISBN>  
</Book>  
<Book>  
  <Author>Sally Bedell Smith</Author>  
  <Name>Elizabeth the Queen</Name>  
</Book>
```

# Unordered Sequence of Elements

---

- Version 1.0: maxOccurs of elements and the whole set is  $\leq 1$ 
    - What if we want maxOccurs  $> 1$ ?
      - Idea: Combination of choice and maxOccurs  $> 1$
      - Can lead to a non-deterministic data model
        - Not allowed by specification, but there can exist a parser which supports it
  - Version 1.1: maxOccurs of elements  $> 1$ 
    - In general, not everything is allowed, but the rules are not so strict
-

# V. Model Group (**group**)

---

- Contains a sequence / choice / set of items (elements)
- Always declared globally and has a name
  - Repeating usage of the content using **references**
- References in general:
  - We declare them using the same construct as the referenced item
    - Instead of attribute **name** we use attribute **ref**
  - The same principle can be used for model groups, elements, attributes and groups of attributes (see later)
    - In case of elements and attributes only the globally defined ones can be referenced

---

`(annotation?, (all | choice | sequence)?)`

# Model Group + References

---

```
<xs:group name="CommonElements">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Author" type="xs:string"/>
    <xs:element name="Date" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:complexType name="TypeBook">
  <xs:sequence>
    <xs:group ref="CommonElements" minOccurs="0"/>
    <xs:element name="ISBN" type="xs:string"/>
    <xs:element name="Publisher" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```



# Model Group

---

```
<xs:element name="Book" type="TypeBook"/>
```

```
<Book>  
  <Name>Elizabeth the Queen</Name>  
  <Author>Sally Bedell Smith</Author>  
  <Date>2006-02-30</Date>  
  <ISBN>987-654-321</ISBN>  
  <Publisher>Random House Trade Paperbacks</Publisher>  
</Book>
```

---

# Note: References and Elements

---

```
<xs:element name="Name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:complexType name="Book">
  <xs:sequence>
    <xs:element ref="Name"/>
    <xs:element name="Publisher" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

## VI. Complex Content (**complexContent**)

---

- Inference of new types from already existing ones
- **restriction** – the new type is a subset of the original one
  - Restriction of occurrences of an element/attribute
    - Removing of elements: `maxOccurs="0"`
    - Removing of attributes: `use="prohibited"`
  - Restriction of allowed values of simple types (of attribute values, content of text elements)
- **extension** – the new type contains original and new data (in this order)
  - A kind of inheritance

---

`(annotation?, (restriction | extension))`

# Complex Content – Example

---

```
<xs:complexType name="Publication">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Author" type="xs:string"
                minOccurs="1" maxOccurs="unbounded"/>
    <xs:element name="Published" type="xs:gYear"/>
  </xs:sequence>
</xs:complexType>
```

# Complex Content – Example of Restriction

---

```
<xs:complexType name="PublicationWithOneAuthor">
  <xs:complexContent>
    <xs:restriction base="Publication">
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Author" type="xs:string"/>
        <xs:element name="Published" type="xs:gYear"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

---

# Complex Content – Example of Restriction

---

```
<xs:element name="Book" type="Publication"/>
<xs:element name="Book1" type="PublicationWithOneAuthor"/>
```

```
<Book>
  <Name>XML technologie</Name>
  <Author>Irena Mlýnková</Author>
  <Author>Jaroslav Pokorný</Author>
  <Author>Karel Richta</Author>
  <Author>Kamil Toman</Author>
  <Author>Vojtěch Toman</Author>
  <Published>2006</Published>
</Book>
```

```
<Book1>
  <Name>Elizabeth the Queen</Name>
  <Author>Sally Bedell Smith</Author>
  <Published>2006</Published>
</Book1>
```

# Complex Content – Example of Extension

---

```
<xs:complexType name="TypeBook">
  <xs:complexContent>
    <xs:extension base="Publication">
      <xs:sequence>
        <xs:element name="Publisher" type="xs:string"/>
        <xs:element name="ISBN" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

# Complex Content – Example of Extension

---

```
<xs:element name="Book" type="TypeBook"/>
```

```
<Book>  
  <Name>XML technologie</Name>  
  <Author>Irena Mlýnková</Author>  
  <Author>Jaroslav Pokorný</Author>  
  <Author>Karel Richta</Author>  
  <Author>Kamil Toman</Author>  
  <Author>Vojtěch Toman</Author>  
  <Published>2006</Published>  
  <Publisher>Karolinum</Publisher>  
  <ISBN>80-246-1272-0</ISBN>  
</Book>
```



# Complex Content

---

- Related attributes of complexType:
    - abstract – abstract data type
      - Cannot be assigned to any element
      - First we must derive a new type
    - final – forbids further derivation
      - Values: restriction, extension, #all
      - Like with simple types
-

# Invariants (**assert**)

---

- Version 1.1: Possibility of specification of conditions for existence or values of subelements / attributes
    - Using XPath
  - Similar to CHECK constraint in databases
  - Attributes:
    - test – XPath expression which must hold true
  - Meaning:
    - assert – error, when the expression does not return true
-

# Invariants

---

```
<xs:complexType name="Interval">
  <xs:attribute name="min" type="xs:integer"/>
  <xs:attribute name="max" type="xs:integer"/>
  <xs:assert test="@min < @max"/>
</xs:complexType>
```

```
<xs:complexType name="Array">
  <xs:sequence>
    <xs:element name="Item" minOccurs="0"
                maxOccurs="unbounded"
                type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="NumberOfItems" type="xs:integer"/>
  <xs:assert test="@NumberOfItems = fn:count(./Item)"/>
</xs:complexType>
```

# Set of Attributes (**attributeGroup**)

---

- Contains a set / group of attributes
  - Similar to a model group of elements
- Always declared globally and always has a name
  - Repeating usage of a set of attributes
    - Using references
    - The same principle can be used for globally defined attributes

---

`(annotation?, ((attribute | attributeGroup)*, anyAttribute?))`

# Set of Attributes – Example

---

```
<xs:attributeGroup name="CommonAttributes">
  <xs:attribute name="Borrowed" type="xs:boolean"/>
  <xs:attribute name="Id" type="xs:ID"/>
</xs:attributeGroup>

<xs:complexType name="TypeBook">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Publisher" type="xs:string"/>
  </xs:sequence>
  <xs:attributeGroup ref="CommonAttributes"/>
</xs:complexType>
```

---

# Set of Attributes – Example

---

```
<xs:element name="Book" type="TypeBook"/>
```

```
<Book Borrowed="true" Id="1234">  
  <Name>XML technologie</Name>  
  <Publisher>Karolinum</Publisher>  
</Book>
```

---

# Note: References and Attributes

---

```
<xs:attribute name="Borrowed">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<xs:complexType name="Book">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Publisher" type="xs:string"/>
  </xs:sequence>
  <xs:attribute ref="Borrowed"/>
</xs:complexType>
```

---

# Advanced Components

---



# XML Schema and Namespaces

---

- XML Schema enables to define a namespace
    - Target namespace
  - Parts of a namespace vs. XML Schema constructs:
    - All element partition
      - Globally defined elements
    - Per element type partitions
      - Attributes of elements
    - Global attribute partition
      - Globally defined attributes
  - Element schema has two special attributes:
    - `elementFormDefault`, `attributeFormDefault`
    - Values: qualified/unqualified
      - Default: unqualified
    - Denote the necessity of qualification of element/attribute names with namespace prefixes
-

# XML Schema – Namespace Declaration

---

- Namespaces:
  - Namespace of XML Schema language
  - Target namespace
  - Implicit namespace
    - We do not have to use a prefix for defined items
      - If we do not define a target namespace, this holds implicitly

```
<?xml version="1.0" encoding="windows-1250"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.mff.cuni.cz/MySchema"
  xmlns="http://www.mff.cuni.cz/MySchema"
  elementFormDefault="qualified">
  ... <!-- definition of XML schema --> ...
</xs:schema>
```

# XSD vs. XML Document – Usage of Namespaces

---

- XSD has a target namespace
  - Namespace of (all) instances of XML Schema (i.e. XML documents)
  - Namespace of XSD of the XML document + URL of the XSD file
  - Implicit namespace

```
<?xml version="1.0" encoding="windows-1250"?>
<root_element_of_XML_document
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.mff.cuni.cz/MySchema schema1.xsd"
  xmlns="http://www.mff.cuni.cz/MySchema">
  ... <!-- XML document --> ...
</root_element_of_XML_document>
```

# Note: DTD and Namespaces

---

- ❑ DTD does not support namespaces
- ❑ We can add prefixes to element / attribute names
- ❑ In all the respective XML documents we must use the same prefixes!
  - Change of prefix means a change everywhere (DTD + all documents)
  - Namespaces do not define the prefix, it is defined **locally**

```
<!ELEMENT emp:employee (emp:name, emp:address, emp:phone,
emp:phone?, emp:phone?)>
<!ELEMENT emp:phone (#PCDATA)>
<!ATTLIST emp:employee
    emp:ID CDATA #REQUIRED
    emp:DID CDATA #REQUIRED
>
```

# External Schemas (**include**)

---

- Including of a schema with the same / none target namespace
  - The components of the included schema become parts of the current target name space
  - Like if we just copy the content of the schema

```
<xs:schema ...>
  <!-- including of components of an external schema -->
  <xs:include schemaLocation="MySchema1.xsd" />

  <!-- definition of other schema components -->
</xs:schema>
```

---

# External Schemas (**import**)

---

- Import of components with any namespace
  - Globally defined items can be then used for definition of the current schema
    - It is not a copy of the imported schema!
- Attributes:
  - schemaLocation – URI of imported schema
  - namespace – namespace of imported schema
    - If specified, we use qualified names

```
<xs:schema ...
  xmlns:types="http://www.priklady.cz/xml/MySchema2">
  <!-- import of components of an external schema -->
  <xs:import
    namespace="http://www.priklady.cz/xml/MySchema2"/>

  <element name="ElementWithExternalType"
    type="types:ExternalType"/>
</xs:schema>
```

# External Schemas (**redefine**)

---

- Redefinition of an existing component
  - Simple type – restriction
  - Complex type – restriction / extension
  - Group of elements
    - Superset – includes the original set using attribute **ref**
    - Subset – **minOccurs** and **maxOccurs**
  - Group of attributes
    - Superset – includes the original set using attribute **ref**
    - Subset – modification of attribute **use**

---

```
(annotation | (simpleType | complexType | group |  
attributeGroup) ) *
```

# External Schemas (**redefine**) – Example

---

```
<xs:redefine schemaLocation="MySchema2.xsd">
  <xs:complexType name="ExternalType">
    <xs:complexContent>
      <xs:extension base="ExternalType">
        <xs:sequence>
          <xs:element name="NewElement"
            type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>
```



# Identity Restriction

---

- ID, IDREF, IDREFS – taken from DTD
    - Just for attributes
    - Must hold within the whole document
  - XSD identity restrictions:
    - Key – compulsory, not-null, unique value (**key**)
    - Unique – not-null, unique value (**unique**)
    - Reference to key / unique value (**keyref**)
      - Similar to keys and foreign keys in relational databases
  - Based on a (subset of) XPath
-

# Subset of XPath

---

- Steps = elements / attributes (@)
- Can contain:
  - . – current element
  - / – child element / attribute
  - // – descendant in any depth
  - \* – any name

```
Selector ::= PathS ( '|' PathS )*
Field    ::= PathF ( '|' PathF )*
PathS   ::= ( './/' )? Step ( '/' Step )*
PathF   ::= ( './/' )? ( Step '/' )* ( Step | '@' NameTest )
Step    ::= '.' | NameTest
NameTest ::= QName | '*' | NCName ':' '*'
```

# Identity Restriction

---

- Attributes:
  - name – name of identity restriction
  - refer – reference to an existing identity restriction
    - Just for keyref
- Content:
  - **selector** – a set of elements within which the restriction must hold
    - Can be used only once
  - **field** – a set of subelements or attributes (relatively to the set from selector) bearing the restriction
    - At least one
    - Can be a combination of elements / attributes

---

`(annotation?, (selector, field+))`

# unique – Example

---

```
<xs:element name="Library">
  ...
  <xs:element name="Book" maxOccurs="unbounded">
    ...
    <xs:element name="ISBN" type="xs:string"
                 minOccurs="0"/>
    ...
  </xs:element>

  <xs:unique name="UniqueISBN">
    <xs:selector xpath="./Book"/>
    <xs:field    xpath="./ISBN"/>
  </xs:unique>
</xs:element>
```

# key – Example

---

```
<xs:element name="Library">
  ...
  <xs:element name="Book" maxOccurs="unbounded">
    ...
    <xs:element name="ISBN" type="xs:string"/>
    ...
  </xs:element>

  <xs:key name="PrimaryKey">
    <xs:selector xpath=" ./Book" />
    <xs:field    xpath=" ./ISBN" />
  </xs:key>
</xs:element>
```

---

# keyref (1) – Example

---

```
<xs:element name="Library">
  <!-- The previously defined element and constraint -->
  <xs:element name="Author" maxOccurs="unbounded">
    ...
    <xs:element name="BestBook">
      ...
      <xs:element name="ISBN" type="xs:string"/>
      ...
    </xs:element>
    ...
  </xs:element>

  <xs:keyref name="ForeignKey" refer="PrimaryKey">
    <xs:selector xpath=" ./Author/BestBook" />
    <xs:field    xpath=" ./ISBN" />
  </xs:keyref>
</xs:element>
```

# keyref (2) – Example

---

```
<Library>
  <!-- books in library -->
  <Book>
    <ISBN>111-222-333</ISBN>
    <Name>M. Logue - The King's Speech</Name>
  </Book>
  <Book>
    <ISBN>444-555-666</ISBN>
    <Name>D. Brown - The Lost Symbol</Name>
  </Book>
  <Book>
    <ISBN>123-456-789</ISBN>
    <Name>S. B. Smith - Elizabeth the Queen</Name>
  </Book>
```

...

# keyref (3) – Example

---

```
...
<!-- information on authors in library -->
<Author>
  <name>Mark Logue</name>
  <BestBook>
    <ISBN>111-222-333</ISBN>
    <NumberOfEditions>123<NumberOfEditions>
  </BestBook>
</Author>
<Author>
  <name>Sally Bedell Smith</name>
  <BestBook>
    <ISBN>123-456-789</ISBN>
    <NumberOfEditions>0<NumberOfEditions>
  </BestBook>
</Author>
</Library>
```



# Implicit Substitutability

## (Substitutability of Data Types)

---

- Implicit = we do not need to specify anything in the schema
- In the document we specify the data type
  - Derived from the original
- Using attribute `xsi:type`

```
<xs:element name="Publication" type="TypePublication"/>
```

- Attribute `block` of element `complexType`
  - Values: restriction, extension, #all

```
<Publication>
  <Name>The King's Speech</Name>
  <Author>M. Logue</Author>
</Publication>
<Publication xsi:type="TypeBook">
  <Name>Elizabeth the Queen</Name>
  <Author>S. B. Smith</Author>
  <ISBN>123-456-789</ISBN>
</Publication>
```

# Substitution Groups

## (Substitutability of Elements)

---

- ❑ Extension of substitutability
  - ❑ Mechanism of explicit allowing / forbidding of substitution of whole elements (i.e., not only their data types)
  - ❑ Idea: Elements are assigned to a substitution group of a **leading element** denoted using its name
    - The leading element can be then substituted with elements in its substitution group
  - ❑ Conditions:
    - All elements must be defined globally
    - An element in a substitution group must have the same data type as the leading element or a type derived from its data type
    - Relation „to be in a substitution group" is transitive
-

# Substitution Groups – Example

---

```
<xs:element name="Publication" type="TypePublication"/>
<xs:element name="Book" type="TypeBook"
  substitutionGroup="Publication"/>
<xs:element name="Journal" type="TypeJournal"
  substitutionGroup="Publication"/>

<xs:element name="Library">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Publication" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Substitution Groups – Example

---

```
<Library>
  <Publication>
    <Name>The King's Speech</Name>
    <Author>M. Logue</Author>
  </Publication>
  <Book>
    <Name>Elizabeth the Queen</Name>
    <Author>S. B. Smith</Author>
    <ISBN>123-456-789</ISBN>
  </Book>
  <!-- other elements Publication, Book or Journal -->
</Library>
```

---

# Substitution Groups

---

- The features of the groups are given by attributes of **element**
    - **substitutionGroup** – name of the leading element, i.e. the group to which the element is assigned
    - **abstract** – abstract element
      - The element cannot be used in a document, it must be always substituted with an element from its substitution group
    - **final** – blocking of adding of elements to the substitution group of the element
      - Values: extension, restriction, #all
    - **block** – blocking of substitution of the element (there can be elements in its substitution group, but we cannot substitute for it at the particular position)
      - Values: extension, restriction, #all
-

# Wildcards

---

- Enable to use at a particular position any item
- Element **anyAttribute**
  - Attributes:
    - namespace – namespace(s) of allowed items
      - A list of URIs of namespaces
      - `##any` – any known namespace
      - `##targetNamespace` – target namespace
      - `##other` – other than target namespace
      - `##local` – no specific namespace

# Wildcards

---

- processContents – the way of validation of the content
    - strict – strict validation
    - lax – validation in case the parser finds the component
    - skip – no validation
  - notNamespace – list of namespaces from which we cannot use items
    - `##targetNamespace`, `##local`
    - Since version 1.1
  - notQName – list of elements / attributes we cannot use
    - Since version 1.1
-

# Wildcards

---

- Element *any*
    - Attributes:
      - nameSpace, processContents, notNamespace, notQName – the same meaning
      - minOccurs
      - maxOccurs
-



# Wildcards

---

```
<xs:complexType name="AnyHtmlText">
  <xs:sequence>
    <xs:any namespace="http://www.w3.org/1999/xhtml"
      minOccurs="1"
      maxOccurs="unbounded"
      processContents="strict"/>
  </xs:sequence>
</xs:complexType>
```

---

# notation

---

- Link to an external executable program
    - Like in DTD
    - Processing depends on another software
  - Attributes:
    - name – name of notation
    - system – system identifier of the executable program
    - public – public identifier of the executable program
  - References to notation – data type **NOTATION**
    - Can be used only via restriction **enumeration**
    - For each enumerated value there must exist a **notation**
      - Similar to DTD
-

# notation – Example

---

```
<xs:notation name="EPS" system="C:\eps\eps.exe"/>

<xs:attribute name="Program">
  <xs:simpleType>
    <xs:restriction base="xs:NOTATION">
      <xs:enumeration value="EPS">
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

---

# Annotation

```
<xs:annotation>
  <xs:documentation xml:lang="cs">
    Toto je příklad anotace pro člověka.
  </xs:documentation>
</xs:annotation>
```

- Denoted for documentation / comments of the schema
  - XML comments can be used as well
  - Part of any schema component
- Element **appinfo** – information for a program
  - Attributes:
    - source – URI of an external file, where the information is stored
- Element **documentation** – information for a human
  - Attributes:
    - source – URI of an external file, where the information is stored
    - xml:lang – language of the information, when provided directly in the schema