

# XML валидиране чрез XML Schema



Цели  
Структура на XML Schema  
Синтаксис  
Особености  
Примери

# Дефиниция на типа на документа (DTD)

- DTD дефиницията задава шаблон за маркиране на XML документ
- Форматът на DTD е наследен от SGML, като значително е опростен
- Както при SGML, така и XML DTD използва формална граматика за описание на структурата и типа на съдържанията на XML документа
- DTD осигурява начин за проверка на неговата структура и съдържание – т.е. за **валидация**

# XML Schema

- XML Schema е специализиран XML-базиран език за описание на XML документи
- XML Schema е на практика XML документ
- Всички правила, които важат за XML – за затваряне на таговете, за влагане и т.н., са в сила и за XML Schema
- XML Schema добавя допълнителни ограничения към правилата на XML – елементите в един XSD документ имат точно определени имена и са с точно определено значение

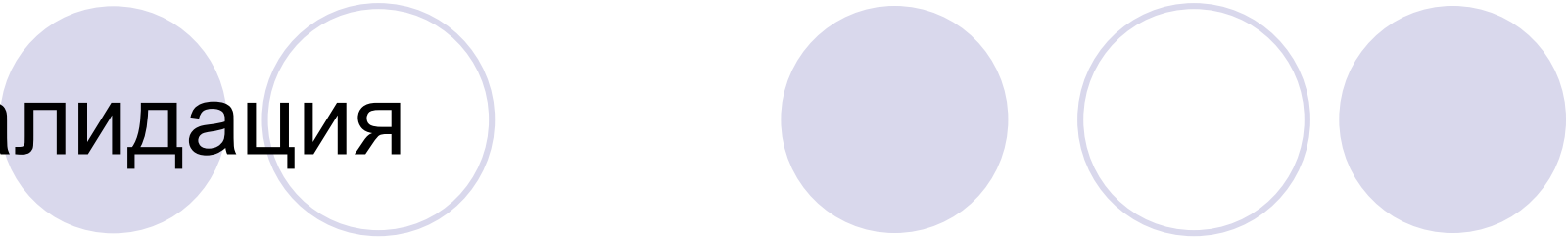
# DTD и XML schema

- DTD (Document Type Definition) и XML schema или XSD (XML Schema Definition) задават правила, съгласно които се определят имената на елементите и атрибутите, тяхната последователност, честота на срещане и др.
- DTD използва по-стегнат и кратък синтаксис в сравнение с XML Schema, но за сметка на това XML Schema предоставя по-богат набор от средства за по-строго дефиниране на структурата на XML и освен това нейните правила се задават в XML формат

# Валиден XML документ

- Всеки отделен документ, отговарящ на даден документен тип, е документен екземпляр (инстанция) на типа. Такъв документ представлява валиден документ за този документен тип.
- Всеки валиден документ е добре конструиран, но обратното не е задължително вярно.
- Всички XML парсери проверяват дали входния документ е добре конструиран XML документ.
- Парсери, които извършват още и проверка за определяне дали съдържанието на XML документите е валидно спрямо зададен тип на документа, се наричат валидиращи парсери.

# Валидация



- Валидацията е времеемък процес, но често спестява много проблеми на външните приложения и се извършва от специализиран процесор (валидиращ парсер)
- DTD описанието (ако е външно!) е споделено от валидизиращия парсер за XML документите-екземпляри на този документен тип – т.е. използва се многократно само едно описание

# Валидиране чрез XML схеми

- XML schema е XML документ, който описва структурата на друг XML документ.
- Тя е наследник на DTD, но предоставя много по-богати възможности за по-прецизна спецификация на XML структура.
- Има множество XML schema езици, като най-разпространените са:
  - **XML Schema** или наричан още XSD (XML Schema Definition), който се препоръчва от W3C консорциума, и
  - **RELAX NG**, дефиниран от OASIS.
- XML schema с малко **s** е общо име за такъв тип езици, а XML Schema с голямо **S** е конкретен XML schema език).

# Валидиране чрез XML схеми (сравнено с DTD) 1/3

- XML Schema използват XML-базиран синтаксис, докато DTD има специфичен синтаксис повлиян от SGML DTD.
- DTD използва по-стегнат и компактен синтаксис в сравнение с XML Schema.
- XML Schema дава възможност да се създават типове данни, които след това да се използват при специфицирането на елементи и атрибути. Върху тези типове данни могат да бъдат специфицирани различни ограничения като например минимална/максимална стойност, изброимо множество от стойности, стойности отговарящи на определен регулярен израз и др.



# Валидиране чрез XML схеми (сравнено с DTD) 2/3

- С DTD за разлика от XML Schema честотата на срещане на даден елемент не може да се зададе с произволно цяло число, а може да приема само стойностите – нула, едно, безкрайност.
- В DTD даден елемент не може да бъде асоцииран с избираем списък от стойности (например така: **<!ELEMENT score ("bad"|"good"|"high")>**), а това е валидно само за атрибутите (напр. **<!ATTLIST score value ("bad"|"good"|"high")>**). В XML Schema това е възможно, като се дефинира такъв тип данни и след това съответният елемент се асоциира с него.

# Валидиране чрез XML схеми (сравнено с DTD) 3/3

- XML Schema поддържа пространства от имена, докато DTD не осигурява такава поддръжка.
- В DTD моделът на съдържание на един елемент се определя изцяло от неговото име и така не може да се използва в различен контекст в зависимост от това къде се намира.
- В XML Schema всеки под-елемент на даден елемент може да се специфицира локално и така отделните под-елементи с едно и също име ще имат различен модел на съдържание.

# И все пак... ползваме DTD:

- ако е необходимо да бъде предефинирана дефиниция на даден елемент (например с използване на ключовите думи **INCLUDE** и **IGNORE**);
- когато съдържанието на елементите е най-вече текстово и не се налага дефинирането на собствени типове и налагането на ограничения;
- XML Schema, както и другите schema езици не предоставят заместващ механизъм за деклариране на **DTD entities**;
- ако е от значение големината на файла, дефиниращ структурата на XML документа.

# Програмни средства за работа със схема

- XML Schema-aware Parser
  - Xerces-J
  - Oracle XML Schema Processor
- XML Schema Validator (XSV, online)
- DTD to Schema Conversion Tools
- XML Schema Editor
  - Extensibility's XML Authority

# Възможности на XML Schema

- Структурни:
  - Пространства от имена
  - Интеграция на примитивни и комплексни типове данни
  - Ограничаване на даден тип данни
  - Наследяване
  - Интегритет
- Типове данни:
  - integers, dates, ... (както в езиците за прогр.)
  - user-defined (ограничаване на свойства)
- Използване:
  - Разлики с MS Schema: *XDR namespace (but not XSD), shown by URN (but not URL), ...*

# Използване на пространства от имена в XML Schema документ

```
<?xml version="1.0"?>  
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
  targetNamespace="http://www.myFirstXMLSchema.com"  
  xmlns="http://www.myFirstXMLSchema.com"  
  elementFormDefault="qualified">  
  <xs:element name="country">  
    <xs:complexType>  
      <xs:sequence>  
        <xs:element name="name" type="xs:string"/>  
        <xs:element name="total_area" type="xs:decimal"/>  
      </xs:sequence>  
    </xs:complexType>  
  </xs:element>  
</xs:schema>
```

XML

# Префиксът xs

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.myFirstXMLSchema.com"
  xmlns="http://www.myFirstXMLSchema.com"
  elementFormDefault="qualified">
  <xs:element name="country">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="total_area" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

**xmlns:xs = http://www.w3.org/2001/XMLSchema**,  
указва, че елементите и типовете данни на схемата  
идват от пространството от имена  
**http://www.w3.org/2001/XMLSchema** и те трябва да  
бъдат с префикс **xs**, т.е. префиксът на директивите на  
схема езика е **xs**. Така директивите на схема езика  
трябва да бъдат използвани по следния начин  
**xs:element, xs:attribute, xs:complexType** и т.н.

# Нуждата от **targetNamespace**

- Аргументът **targetNamespace** определя пространството от имена, за което тази схема е предназначена, а атрибутът **xmlns** определя, че точно **targetNamespace** ще бъде пространството от имена по подразбиране. За тази цел двата атрибута трябва да имат еднаква стойност.
- Пространството от имена дефинирано от **targetNamespace** няма префикс, но това не е задължително и може да има произволен такъв. Така с използване на **targetNamespace** когато дефинираме собствени типове данни те ще бъдат част от пространството, специфицирано от него.
- Аргументът **elementFormDefault="qualified"** декларира, че всички имена на елементи от XML екземплярите на тази схема трябва да бъдат асоциирани с пространство от имена:
  - или чрез префикс,
  - или с пространството от имена по подразбиране.



# XML документ, асоцииран с предишната схема

```
<?xml version="1.0"?>
<xx:country xmlns:xx="http://www.myFirstXMLSchema.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.myFirstXMLSchema.com country.xsd">
  <xx:name>Bulgaria</xx:name>
  <xx:total_area>110993.7</xx:total_area>
</xx:country>
```

# Рефериране на схема в XML документ-екземпляр

NS по подразбиране е targetNamespace

```
<?xml version="1.0"?>
<BookCatalogue xmlns="http://www.somewhere.org/BookCatalogue"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
  xsi:schemaLocation="http://www.somewhere.org/BookCatalogue
    http://www.somewhere.org/BookCatalogue/BookCatalogue1.xsd">

  <Book>
    <Title>My Life and Times</Title>
    <Author>Paul McCartney</Author>
    <Date>July, 1998</Date>
    <ISBN>94303-12021-43892</ISBN>
    <Publisher>McMillin Publishing</Publisher>
  </Book>
  ...
</BookCatalogue>
```

schemaLocation е двойката {NS, XSD файл}

В елемента BookCatalogue декларираме, че атрибута **schemaLocation** идва от пространството XML Schema Instance namespace (xsi). Стойността на **schemaLocation** е двойката (namespace, URI към схемата). Когато XML парсерът обработва този XML документ, той използва двойката в **schemaLocation**, за да извлече файла със схемата (BookCatalogue.xsd) и да провери дали нейният targetNamespace отговаря на пространството в **schemaLocation**. **Пространството по подразбиране** указва, че всички XML елементи идват от същото NS.

# XML Schema и xmlns

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.somewhere.org/BookCatalogue"
  xmlns:cat="http://www.somewhere.org/BookCatalogue">
  <element name="BookCatalogue">
    <annotation>
      <info>A book catalogue contains zero or more books</info>
    </annotation>
    <complexType>
      <sequence>
        <element ref="cat:Book" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="Book">
    <annotation>
      <info>A Book has a Title, Author, Date, ISBN, and a Publisher</info>
    </annotation>
    <complexType>
      <sequence>
        <element ref="cat:Title"/>
        <element ref="cat:Author"/>
        <element ref="cat:Date"/>
        <element ref="cat:ISBN"/>
        <element ref="cat:Publisher"/>
      </sequence>
    </complexType>
  </element>
  <element name="Title" type="string"/>
  <element name="Author" type="string"/>
  <element name="Date" type="string"/>
  <element name="ISBN" type="string"/>
  <element name="Publisher" type="string"/>
</schema>
```

Референция  
(ref) – за пре-  
използване  
на типове

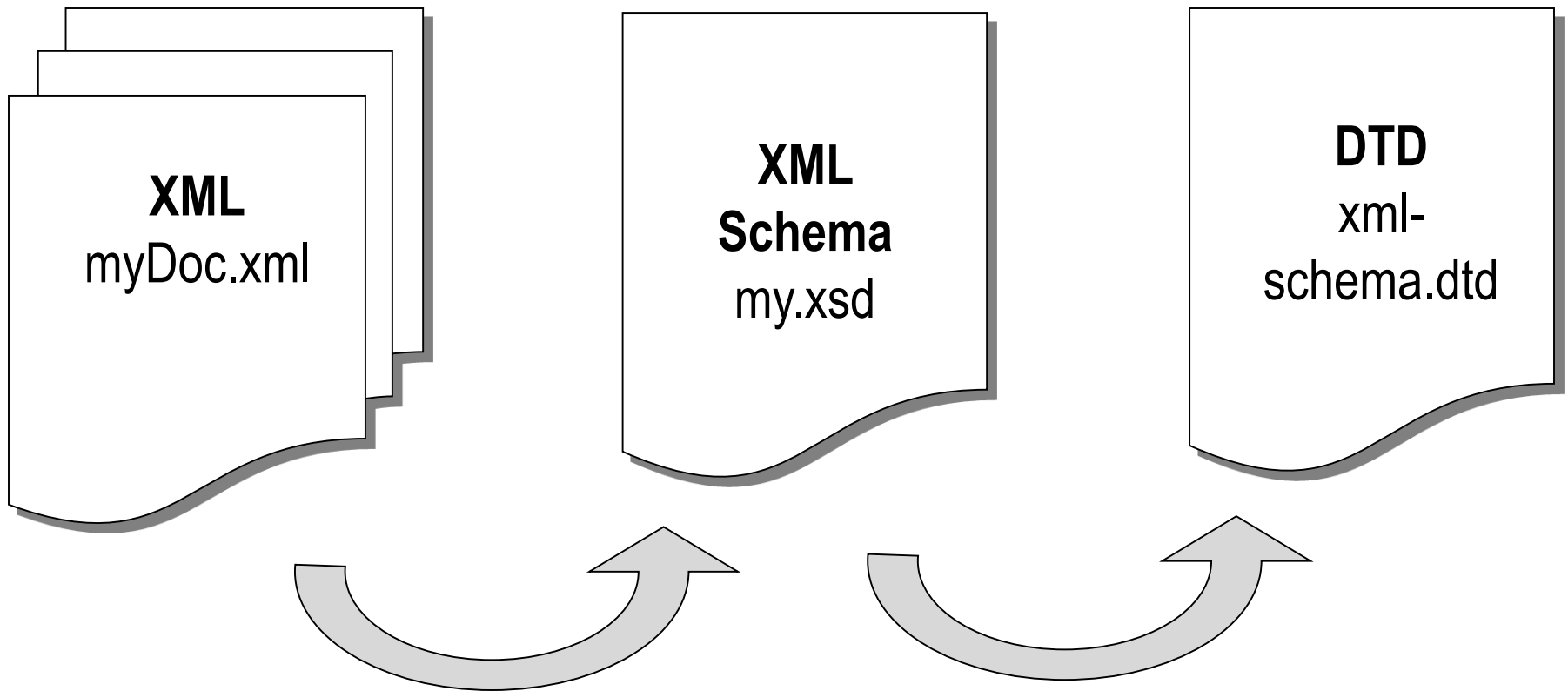
ref **само** към  
**Global types** –  
те са директни  
наследници на  
<schema>

Елементите на  
схемата са с NS  
по подразбиране

Пространството  
cat е  
дефинираното от  
схемата (target  
Namespace).

targetNameSpace  
съвпада с cat.

# Валидиране на документ чрез XML Schema



# XML Schema спецификации

- Part 0: Primer

- въведение

- Part 1: Structures

<http://www.w3.org/TR/xmlschema-1/>

- структури

- ограничения

- Part 2: Data types

- типове данни за елементи и атрибути

<http://www.w3.org/TR/xmlschema-2/>

# Част 1: Структури

- **Type Definitions** `<simpleType>` `<complexType>`  
`<element>` `<group>` `<all>` `<choice>` `<sequence>`  
`<attribute>` `<attributeGroup>`
- **Attribute Declarations** `<attribute>`  
`<simpleType>`
- **Element Declarations** `<element>`  
`<simpleType>` `<complexType>`
- **Attribute Group Definitions** `<attributeGroup>`  
`<attribute>` `<attributeGroup>`
- **Model Group Definitions** `<group>`  
`<element>` `<group>` `<all>` `<choice>` `<sequence>`
- **Notation Declarations** `<notation>`
- **Annotations** `<annotation>`  
`<appinfo>` `<documentation>`

# Структура на DTD спрямо схема

- DTD

- `<!ELEMENT`

- `e1`

- `((e2,e3?)+|e4)>`

- Schema

- `<element name="e1">`

- `<complexType>`

- `<choice>`

- `<sequence`

- `maxOccurs="unbounded">`

- `<element ref="e2"/>`

- `<element ref="e3"`  
`minOccurs="0"/>`

- `</sequence>`

- `<element ref="e4"/>`

- `</choice>`

- `</complexType>`

- `</element>`

# Интегритет по референции и уникалност

*Дефинираме ограничения (Constraints)  
чрез XPath изрази*

- `<unique>`
- `<key>`
- `<keyref>`
- `<selector>`
- `<field>`



# Част 2: Типове данни: `<simpleType>`

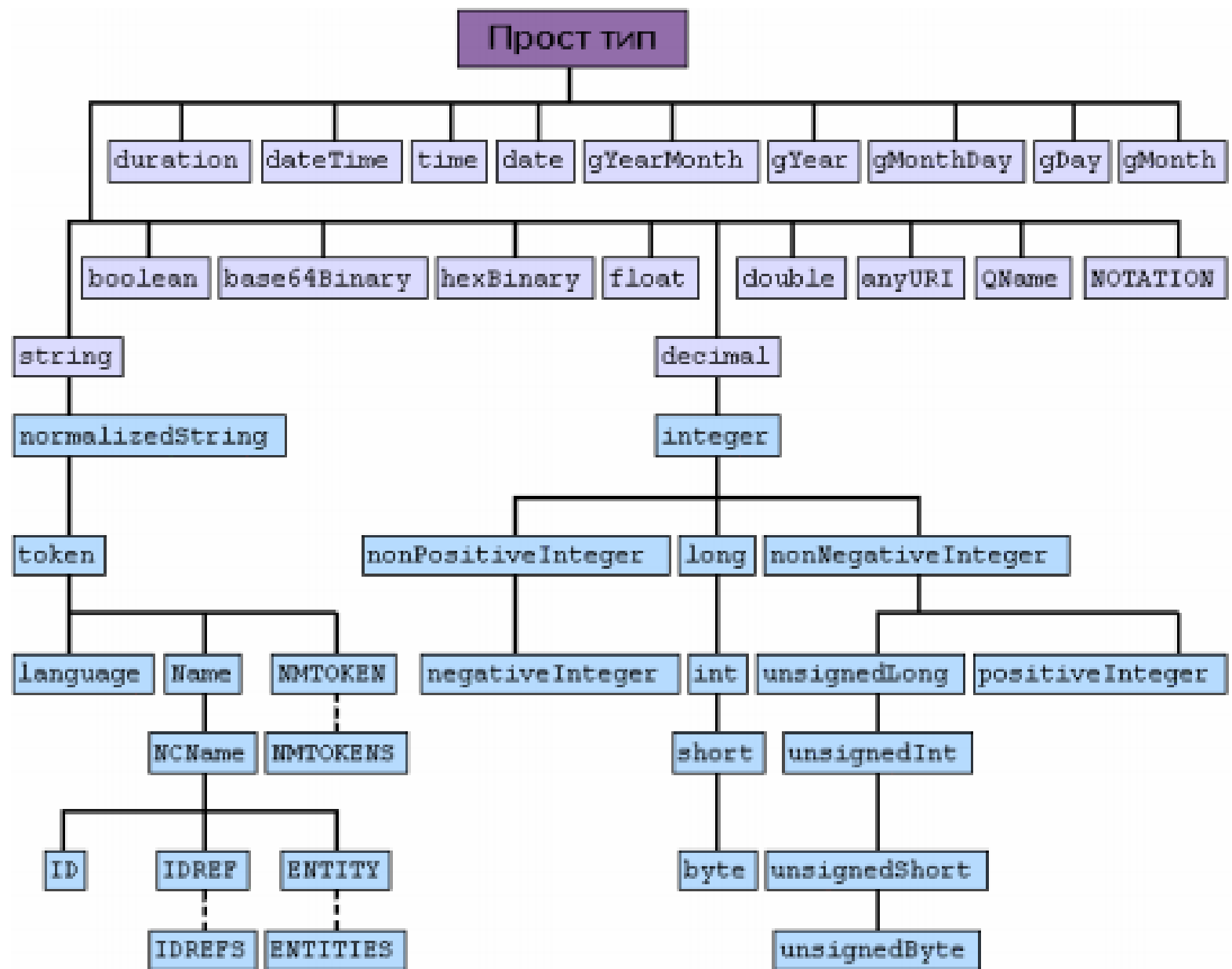
- Пространство на стойности (Value Space)
  - Множество стойности за даден тип данни
  - Дефинира аксиоматично примитивните типове
  - Изброими
  - Ограничими
  - Комбинации от стойности в списък (list)
  - Свойства (*cardinality, equality, ordered, ...*)
- Лексическо пространство
  - Освен value space, всеки тип данни има и lexical space.
  - Дефиниция: „A lexical space is the set of valid literals for a datatype (e.g. 100 and 1.0E2 denote same value)“

# Дефиниране на елементи от прост тип

- Елементите от прост тип в една XML Schema са тези, които:
  - са от даден предефиниран тип, и
  - нямат атрибути и под-елементи:

**<xs:element name="element\_name" type="type\_name"/>**

- Атрибутът **name** задава името на елемента,
- Атрибутът **type** - името на предефинирания тип и може да приема една от стойностите: **float, double, decimal, integer, nonPositiveInteger, nonNegativeInteger, negativeInteger, positiveInteger, long, int, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte, xs:time, dateTime, date, time, gDay, gMonth, gYear, gYearMonth, gMonthDay, duration, string, normal, token, language, NMTOKEN, NMTOKENS, Name, NCName, ID, IDREF, IDREFS, ENTITY, ENTITIES, anyURI, QName, NOTATION, hexBinary, base64Binary, boolean.**



XML

Вградени примитивни типове

Вградени производни типове

получаване с restriction

получаване с list

# Създаване на нови прости типове

Създаваме нови прости типове чрез елемента **<xs:simpleType>** - на база на съществуващите, като обединяваме типове, задаваме списъци от типове, или като добавим ограничения над тях:

- Обединение (union) - чрез елемента **<xs:union />**:  
**<xs:simpleType name="intORfloat">**  
**<xs:union memberTypes="xs:integer xs:float" />**  
**</xs:simpleType>**
- Списък от елементи от даден прост тип, разделени с интервали - чрез елемента **<xs:list />**:  
**<xs:simpleType name="intList">**  
**<xs:list itemType="xs:integer" />**  
**</xs:simpleType>**
- Ограничение (restriction) – на следващите слайдове

# Ограничения върху стойността на елемент от прост тип

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.xmlschema.com"
  xmlns="http://www.xmlschema.com"
  elementFormDefault="qualified">
  <xs:element name="country">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Bulgaria"/>
              <xs:enumeration value="Greece"/>
              <xs:enumeration value="Romania"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="area" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Новият тип наследява **xs:string**, и то с ограничения!

```
<country>
  <name>Bulgaria</name>
  <area>110879</area>
</country>
```

# Простите типове могат да дефинират списък от стойности

- `<xsd:element name="cities" type="xsd:countryCities ">`
- `<xsd:simpleType name="countryCities">`
- `<xsd:list>`
- `<xsd:simpleType>`
- `<xsd:restriction base="xsd:string">`
- `<xsd:enumeration value="Sofia"/>`
- `<xsd:enumeration value="Plovdiv"/>`
- `<xsd:enumeration value="Varna"/>`
- `</xsd:restriction>`
- `</xsd:simpleType>`
- `</xsd:list>`
- `</xsd:simpleType>`
- `</xsd:element>`

- Примерна инстанция:

- `<cities>Sofia Plovdiv</cities >`

# Фасети (аспекти)

- Дефиниция (<http://www.w3.org/TR/xmlschema-2/#facets>)
  - Фасет е определящ аспект на пространството от стойности за даден тип.
  - Фасетите на даден тип данни служат да се разграничат онези аспекти на този тип, които се различават от други типове данни.
- Видове фасети
  - Фундаментални фасети (дефинират типа)
  - Ограничаващи фасети (ограничават стойностите на типа)

# Фундаментални фасети

- Equal
  - Важи за всички типове данни
- Order
  - За някои типове
- Bounds
  - upper bound и lower bound
- Cardinality
  - finite, infinite
- Numeric
  - yes или no



# Ограничаващи фасети

- length
- minLength
- maxLength
- pattern
- enumeration
- maxInclusive / maxExclusive
- minInclusive / minExclusive
- precision
- scale
- encoding
- duration
- period

# Примитивни спрямо дериватни типове

## ● А. Примитивни типове

- string
- boolean
- float
- double
- decimal
- timeDuration
- recurringDuration
- binary
- uriReference
- ID
- IDREF
- ENTITY
- NOTATION
- QName

## ● В. Дериватни типове

### ○ Чрез ограничения

- С използване на фасети

```
<simpleType name="sku"
  base="xsd:string">
  <pattern
    value="\d{3}-[A-D]{4}"/>
</simpleType>
```

### ○ Чрез списъци →

*Съществуват ab initio*

# Вградени спрямо производни типове

- Вградени типове

- А. примитивни

- В. производни

- language

- IDREFS

- long

- int

- short

- positiveInteger

- time

- month

- recurringDay

- ...

- Потребителски  
типове

- Само производни

# Атомарни спрямо списъчни типове

- Атомарни

- Неделими стойности

- `<simpleType  
name="ShoeSize"  
base="xsd:decimal"/>`

- `<element name="shoe"  
type="ShoeSize"/>`

- `<shoe>10.5</shoe>`

- Списъци

- Последователност от атомарни стойности

- `<simpleType  
name="ShoeSizes"  
base="ShoeSize"  
derivedBy="list"/>`

- `<element name="shoes"  
type="ShoeSizes"/>`

- `<shoes>8 10  
10.5</shoes>`

# Създаване на потребителски тип данни

Използваме **datatype** елемента за деклариране на производен типове (на базата на string, integer, ...)

```
<datatype name="TelephoneNumber" source="string">  
  <length value="8"/>  
  <pattern value="\d{3}-\d{4}"/>  
</datatype>
```

Новият тип 'TelephoneNumber' се задава от осем-символен шаблон, съдържащ : ddd-dddd, където 'd' представя цифра ('digit').

# Фасети за тип данни Integer

- maxInclusive
- maxExclusive
- minInclusive
- minExclusive



```
<datatype name= "EarthSurfaceElevation"  
source="integer">  
  <minInclusive value="-430"/>  
  <maxInclusive value="8848"/>  
</datatype>
```

# Начин за използване на фасети

`<datatype name= "name" source= "source">`

`<facet value= "value"/>`

`<facet value= "value"/>`

...

`</datatype>`

## Facets:

- **minInclusive**
- **maxInclusive**
- **minExclusive**
- **maxExclusive**
- **length**
- **minlength**
- **maxlength**
- **pattern**
- **enumeration**

...

## Sources:

- **string**
- **boolean**
- **float**
- **double**
- **decimal**
- **timeInstant**
- **timeDuration**
- **recurringInstant**
- **binary**
- **uri**

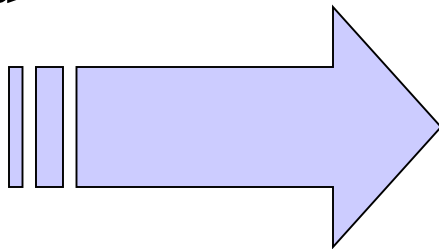
...

# Регулярни изрази 1/2

- Стойността на фасета **pattern** е регулярен израз:

## *Regular Expression*

- Chapter \d
- a\*b
- [xyz]b
- a?b
- a+b
- [a-c]x



## *Example*

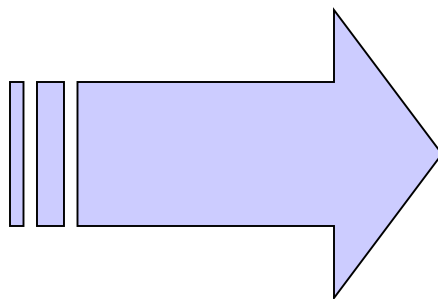
- Chapter 1
- b, ab, aab, aaab, ...
- xb, yb, zb
- b, ab
- ab, aab, aaab, ...
- ax, bx, cx



# Регулярни изрази 2/2

## ● Регулярен израз

- [a-c]x
- [-ac]x
- [ac-]x
- [^0-9]x
- \Dx
- Chapter\s\d
- (ho){2} there
- (ho\s){2} there
- .abc
- (a|b)+x



## ● Пример

- ax, bx, cx
- -x, ax, cx
- ax, cx, -x
- any non-digit char followed by x
- any non-digit char followed by x
- Chapter followed by blank followed by digit
- hoho there
- ho ho there
- any char followed by abc
- ax, bx, aax, bbx, abx, bax,...

# Регулярен израз за интервала 0..255

**[1-9]?[0-9] | 1[0-9][0-9] | 2[0-4][0-9] | 25[0-5]**

0 to 99	100 to 199	200 to 249	250 to 255
---------	------------	------------	------------

# IP тип данни

```
<datatype name="IP" source="string">  
  <pattern value="((([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}  
    ([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])">  
    <annotation>  
      <info>  
        Datatype for representing IP addresses. Examples,  
        129.83.64.255, 64.128.2.71, etc.  
        This datatype restricts each field of the IP address  
        to have a value between zero and 255, i.e.,  
        [0-255].[0-255].[0-255].[0-255]  
        Note: in the value attribute (above) the regular  
        expression has been split over two lines. This is  
        for readability purposes only. In practice the R.E.  
        would all be on one line.  
      </info>  
    </annotation>  
  </pattern>  
</datatype>
```

# Сложни типове

Сложни елементите са тези, асоциирани с т.нар. сложен тип и могат да бъдат празни или да съдържат под-елементи и атрибути. За целта съответният сложен тип трябва да бъде:

- дефиниран и именуван извън елемента или
- да бъде специфициран като под-елемент.

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

```
  <xs:complexType>
```

```
    <xs:sequence>
```

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

```
      <xs:element name="population" type="xs:integer"/>
```

```
    </xs:sequence>
```

```
  </xs:complexType>
```

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

XML валидиране чрез XML Schema

# Сложен тип, дефиниран извън елемента

Сложен тип, дефиниран извън елемента, подобрява четимостта на XML Schema документа:

```
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>
```

# Видове елементи от сложен тип

- празни елементи – те са без съдържание, под-елементи и атрибути подобно на html елемента **<br/>**.
- елементи с атрибути без под-елементи – те имат атрибути и могат да имат също текстово съдържание, но нямат под-елементи.
- елементи, съдържащи под-елементи - те имат под-елементи и могат да имат също текстово съдържание, но нямат атрибути.
- елементи с атрибути с под-елементи – те имат както атрибути, така и под-елементи, а също така може да имат текстово съдържание.

# Разширения или ограничения на типове

- Използваме **simple/complex-type** елементи за деклариране на типове на елементи и/или атрибути
- Използваме **datatype** елемента за деклариране на производен типове (на базата на string, integer, ...)
- XML Schema разрешава разширения или ограничения на типове
- Разширения или ограничения на типове са възможни с използване на елементите:
  - **simpleContent**
  - **complexContent**

# Елементи от тип **simpleContent**

- XML Schema елементът с име **simpleContent** задава разширения или ограничения върху:
  - complex type, който е само текст
  - simple type
- XML елементът от тип **simpleContent** не съдържа елементи, но може да съдържа атрибути

```
<xs:complexType name="image">  
  <xs:simpleContent>  
    <xs:extension base="xs:base64Binary">  
      <xs:attribute name="src" type="xs:string" use="required"/>  
    </xs:extension>  
  </xs:simpleContent>  
</xs:complexType>
```



# Елементи от тип **complexContent**

- Елементът **complexContent** определя разширения или ограничения за сложен тип, който съдържа:

- смесено съдържание или
- само елементи

```
<xs:complexType name="customerExt">
  <xs:complexContent>
    <xs:extension base="customer">
      <xs:sequence>
        <xs:element ref="contacts" minOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType
name="customer">
  <xs:sequence>
    <xs:element
ref="name"/>
    <xs:element
ref="address"/>
  </xs:sequence>
  <xs:attribute name="id"
type="xs:ID" use="required"/>
</xs:complexType>
```

# Елементи с атрибути

```
<xs:element name="population">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string"
          use="required"/>
        <xs:attribute name="year" type="xs:integer"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

XML документът ще има елемент **population**, който ще изглежда така:

```
<population country="bulgaria" year="2011"> 7500441
</population>
```

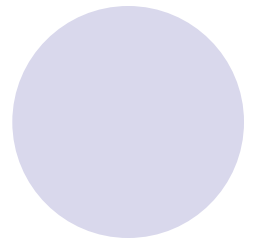
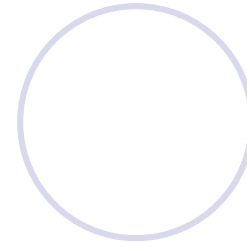
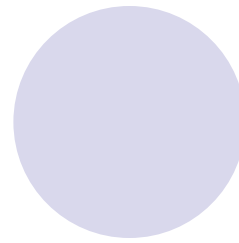
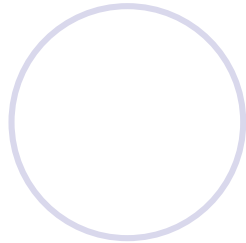
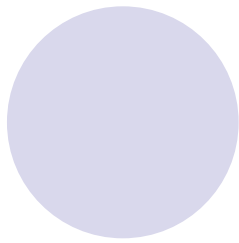
# Производни типове

- Производни типове ("derived types") се създават чрез наследяване по два начина:
  - derive by **extension**: *разширяване на родителския тип с повече елементи*
  - derive by **restriction**: *ограничаване на родителския тип посредством:*
    - **по-ограничен обхват на стойности (range of values), и/или**
    - **по-ограничен брой на повторения.**

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.somewhere.org/BookCatalogue"
        xmlns:cat="http://www.somewhere.org/BookCatalogue">
  <complexType name="Publication">
    <element name="Title" type="string" maxOccurs="unbounded"/>
    <element name="Author" type="string" maxOccurs="unbounded"/>
    <element name="Date" type="date"/>
  </complexType>
  <complexType name="Book" source="cat:Publication" derivedBy="extension">
    <element name="ISBN" type="string"/>
    <element name="Publisher" type="string"/>
  </complexType>
  <element name="BookCatalogue">
    <complexType>
      <element name="CatalogueEntry" minOccurs="0" maxOccurs="unbounded"
type="cat:Book" />
    </complexType>
  </element>
</schema>

```



```
<complexType name="Publication">
  <element name="Title" type="string" maxOccurs="unbounded"/>
  <element name="Author" type="string" maxOccurs="unbounded"/>
  <element name="Date" type="date"/>
</complexType>
<complexType name="Book" source="cat:Publication" derivedBy="extension">
  <element name="ISBN" type="string"/>
  <element name="Publisher" type="string"/>
</complexType>
```

Като резултат:

*Елементите от тип Book ще имат 5 наследника – Title, Author, Date, ISBN, and Publisher.*

# Използване на дериватни типове

- Ако декларираме даден тип да бъде Publication,
- тогава в XML документа-екземпляр неговото съдържание може да бъде
- или Publication, или Book
- (понеже Book е Publication).

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.somewhere.org/BookCatalogue"
        xmlns:cat="http://www.somewhere.org/BookCatalogue">
  <complexType name="Publication">
    <element name="Title" type="string" maxOccurs="unbounded"/>
    <element name="Author" type="string" maxOccurs="unbounded"/>
    <element name="Date" type="date"/>
  </complexType>
  <complexType name="Book" source="cat:Publication"
derivedBy="extension">
    <element name="ISBN" type="string"/>
    <element name="Publisher" type="string"/>
  </complexType>
  <element name="BookCatalogue">
    <complexType>
      <element name="CatalogueEntry" minOccurs="0"
        maxOccurs="unbounded" type="cat:Publication"/>
    </complexType>
  </element>
</schema>

```

BookCatalogue6.xsd

```

<?xml version="1.0"?>
<Catalogue xmlns="http://www.somewhere.org/Catalogue"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
  xsi:schemaLocation="http://www.somewhere.org/Catalogue
    http://www.somewhere.org/Catalogue/BookCatalogue6.xsd">

  <CatalogueEntry>
    <Title>Staying Young Forever</Title>
    <Author>Karin Granstrom Jordan, M.D.</Author>
    <Date>December, 1999</Date>
  </CatalogueEntry>

  <CatalogueEntry xsi:type="Book">
    <Title>Illusions The Adventures of a Reluctant Messiah</Title>
    <Author>Richard Bach</Author>
    <Date>1977</Date>
    <ISBN>0-440-34319-4</ISBN>
    <Publisher>Dell Publishing Co.</Publisher>
  </CatalogueEntry>

  <CatalogueEntry xsi:type="Book">
    <Title>The First and Last Freedom</Title>
    <Author>J. Krishnamurti</Author>
    <Date>1954</Date>
    <ISBN>0-06-064831-7</ISBN>
    <Publisher>Harper & Row</Publisher>
  </CatalogueEntry>
</Catalogue>

```

Publication

Book



```
<CatalogueEntry xsi:type="Book">
  <Title>Illusions The Adventures of a Reluctant Messiah
</Title>
  <Author>Richard Bach</Author>
  <Date>1977</Date>
  <ISBN>0-440-34319-4</ISBN>
  <Publisher>Dell Publishing Co.</Publisher>
</CatalogueEntry>
```

- Нека елементът CatalogueEntry е от тип Publication. Book е дериват на Publication. Следователно, Book е Publication. Ето защо съдържанието на CatalogueEntry може да е Book.
- За да укажем, че съдържанието е от даден дериватен тип, трябва да зададем от кой точно дериватен тип е то (така избягваме възможни нееднозначности!)
- **Атрибутът 'type' идва от XML Schema Instance (xsi) namespace."**

# Деривация чрез рестрикция

```
<complexType name="Publication">
  <element name="Title" type="string" maxOccurs="unbounded"/>
  <element name="Author" type="string" maxOccurs="unbounded"/>
  <element name="Date" type="date"/>
</complexType>
<complexType name="SingleAuthorPublication" source="cat:Publication"
derivedBy="restriction">
  <restrictions>
    <element name="Author" type="string" maxOccurs="1"/>
  </restrictions>
</complexType>
```

Елементите от типа SingleAuthorPublication ще имат 3 дъщерни елемента – Title, Author, and Date.

Трябва да съществува точно един елемент Author.

# Деривация чрез рестрикция – простият тип *restriction*

```
<attribute name="Title">  
  <simpleType>  
    <restriction base="string">  
      <enumeration value="Sir"/>  
      <enumeration value="Dr."/>  
      <enumeration value="Mr."/>  
    </restriction>  
  </ simpleType >  
</attribute >
```

# Ограничаване на деривациите

Можем да създадем нов тип и да:

- Забраним всички деривации от него, или
- Забраним само рестрикциите от него, или
- Забраним само разширенията от него.

`<type name="Publication" final="#all" ...>`

This type cannot be extended nor restricted

`<type name="Publication" final="restriction" ...>`

This type cannot be restricted

`<type name="Publication" final="extended" ...>`

This type cannot be extended

# exact тип

- Даден тип е ***exact***, ако:
  - Други типове могат да са негови деривати.
  - В документа-екземпляр на схемата дериватните типове не могат да се ползват вместо този exact тип.

Схема:

```
<complexType name="Publication">
  <element name="Title" type="string" maxOccurs="unbounded"/>
  <element name="Author" type="string" maxOccurs="unbounded"/>
  <element name="Date" type="date"/>
</complexType>
<complexType name="Book" source="cat:Publication" derivedBy="extension">
  <element name="ISBN" type="string"/>
  <element name="Publisher" type="string"/>
</complexType>
<element name="Catalogue">
  <complexType>
    <element name="CatalogueEntry" minOccurs="0" maxOccurs="unbounded" type="cat:Publication"/>
  </complexType>
</element>
```

**Това разрешава елементи от тип Publication, както и техните дериватни типове като например Book, да се ползват като деца на Catalogue, напр.**

Документ-  
екземпляр:

```
<CatalogueEntry xsi:type="Book">
  <Title>Illusions The Adventures of a Reluctant Messiah</Title>
  <Author>Richard Bach</Author>
  <Date>1977</Date>
  <ISBN>0-440-34319-4</ISBN>
  <Publisher>Dell Publishing Co.</Publisher>
</CatalogueEntry>
```

Схема:

```

<complexType name="Publication" exact="extension">
  <element name="Title" type="string" maxOccurs="unbounded"/>
  <element name="Author" type="string" maxOccurs="unbounded"/>
  <element name="Date" type="date"/>
</complexType>
<complexType name="Book" source="cat:Publication" derivedBy="extension">
  <element name="ISBN" type="string"/>
  <element name="Publisher" type="string"/>
</complexType>
<element name="Catalogue">
  <complexType>
    <element name="CatalogueEntry" minOccurs="0" maxOccurs="unbounded" type="cat:Publication"/>
  </complexType>
</element>

```


*Това забранява използването на дериватни типове, разширяващи Publication, като деца на CatalogueEntry, напр.*

Екземпляр:

```

<CatalogueEntry xsi:type="Book">
  <Title>Illusions of a Reluctant Messiah</Title>
  <Author>Richard...</Author>
  <Date>1977</Date>
  <ISBN>0-440-343...</ISBN>
  <Publisher>Dell Publishing Co.</Publisher>
</CatalogueEntry>

```



# exact типове

- exact="extension"

- Забранява **използването** на дериватни типове чрез разширение вместо дадения тип в документ – екземпляр на схемата

- exact="restriction"

- Забранява **използването** на дериватни типове чрез рестрикция вместо дадения тип в документ – екземпляр на схемата

- exact="#all"

- Забранява **използването** на всякакви дериватни типове вместо дадения тип



# Еквивалентност

- Често в ежедневното общуване изразяваме нещо по няколко начина:
  - В Бостън думите "Т" (от Tube) и "subway" са взаимозаменяеми и означават метро:
    - "we took the T into town"
    - "we took the subway into town".
  - Така "Т" и "subway" са еквивалентни
- Свойството еквивалентност (equivalence) може да се представи в XML Schema.
  - **Желаем да декларираме еквивалентност между елемента "subway" и елемента "Т", така че в документите-екземпляри на схемата да можем да ползваме "subway" или "Т", според предпочитанията ни.**

# equivClass

- Създаваме елемента **subway** (наричан *exemplar*) и след него други елементи, които са еквивалентни (*equivalent*) на екземпляра.

```
<element name="subway" type="string"/>  
<element name="T" equivClass="boston:subway"  
type="string"/>
```

- **subway** : *exemplar*
- **T** : *equivalent*.

*Можем да използваме екземпляра и еквивалентите му взаимозаменяемо.*

### Schema:

```
<element name="subway" type="string"/>
<element name="T" equivClass="boston:subway" type="string"/>
<element name="transportation">
  <simpleType>
    <element ref="boston:subway"/>
  </simpleType>
</element>
```

### Instance doc:

```
<transportation>
  <subway>Red Line</subway>
</transportation>
```

### Alternative Instance doc:

```
<transportation>
  <T>Red Line</T>
</transportation>
```

Заб.: типът на всеки еквивалентен елемент трябва да бъде същият като типа на елемента-екземпляр, или негов дериват.

# Подпомагане на оперативния обмен на данни (Interoperability) 1/2

- Софтуерни приложения от различни приложни области (domains) трябва да си комуникират ефективно независимо от различията помежду им
- equivClass е начална стъпка в реализацията на оперативния обмен на данни (interoperability).
  - Експлицитна еквивалентност между елементи

# Подпомагане на оперативния обмен на данни (Interoperability) 2/2

- Софтуерно приложение очаква да открие елемента `<subway>`.
- В даден XML документ-екземпляр то открива елемента `<T>`.
- Приложението прочита декларациите в XML Schema и установява, че `<T>` е еквивалент на `<subway>`, така то приема този XML документ, макар че той е с различен речник (*vocabulary*) от очаквания.
- Добавяме и нов елемент към схемата напр.  
`<element name="train" equivClass="boston:subway" type="string"/>`
- Без промяна на приложението, то ще може да обработва XML документи с елемента `<train>`.
- Mike Los предлага термина "***interoperability schema***" с цел да опише тези възможности.

# Abstract елементи

- Даден елемент може да се декларира като абстрактен, напр.:  
**<element name="Publication" type="cat:Pub" abstract="true"/>**
- Абстрактният елемент е вид контейнер или шаблон.
- Той не може да присъства в XML документа-екземпляр:
  - Напр., **<Publication>** не може да бъде използван.
- На негово място обаче могат да се ползват елементи, които са му еквивалентни.

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/Catalogue"
  xmlns:cat="http://www.somewhere.org/Catalogue">
  <complexType name="Pub">
    <element name="Title" type="string" maxOccurs="*" />
    <element name="Author" type="string" maxOccurs="*" />
    <element name="Date" type="date" />
  </complexType>
  <element name="Publication" type="cat:Pub" abstract="true"/>
  <element name="Book" equivClass="cat:Publication">
    <complexType source="cat:Pub" derivedBy="extension">
      <element name="ISBN" type="string" />
      <element name="Publisher" type="string" />
    </complexType>
  </element>
  <element name="Magazine" equivClass="cat:Publication">
    <complexType source="cat:Pub" derivedBy="restriction">
      <restrictions>
        <element name="Author" type="string" maxOccurs="0" />
      </restrictions>
    </complexType>
  </element>
  <element name="Catalogue">
    <complexType>
      <element ref="cat:Publication" minOccurs="0" maxOccurs="unbounded" />
    </complexType>
  </element>
</schema>

```

XML

XML валидиране чрез XML Schema

Елементите Book  
и Magazine са  
еквивалентни на  
**Publication.**

Понеже  
Publication  
е абстрактен, то  
само неговите  
еквиваленти  
могат да бъдат  
деца на  
Catalogue.

# XML документ-экземпляр на предната схема

```
<?xml version="1.0"?>
<Catalogue xmlns="http://www.somewhere.org/Catalogue"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
  xsi:schemaLocation="http://www.somewhere.org/Catalogue
    http://www.somewhere.org/Catalogue/BookCatalogue7.xsd">
  <Magazine>
    <Title>Natural Health</Title>
    <Date>December, 1999</Date>
  </Magazine>
  <Book>
    <Title>Illusions The Adventures of a Reluctant Messiah</Title>
    <Author>Richard Bach</Author>
    <Date>1977</Date>
    <ISBN>0-440-34319-4</ISBN>
    <Publisher>Dell Publishing Co.</Publisher>
  </Book>
  <Book>
    <Title>The First and Last Freedom</Title>
    <Author>J. Krishnamurti</Author>
    <Date>1954</Date>
    <ISBN>0-06-064831-7</ISBN>
    <Publisher>Harper & Row</Publisher>
  </Book>
</Catalogue>
```

XML



# Атрибути



- Използване в BookCatalogue DTD.
- ... и в XML Schema....

```

<!-- A book catalogue contains zero or more books -->
<!ELEMENT BookCatalogue (Book)*>
<!-- A Book has a Title, one or more Authors, a Date, an ISBN, and a Publisher -->
<!ELEMENT Book (Title, Author+, Date, ISBN, Publisher)>
<!-- A Book has three attributes - Category, InStock, and Reviewer. Category must be
either "autobiography", "non-fiction", or "fiction". A value must be supplied for this
attribute whenever a Book element is used within a document. InStock can be either
"true" or "false". If no value is supplied it defaults to "false". Reviewer contains the
name of the reviewer. It defaults to "" if no value is supplied -->

```

### **<!ATTLIST Book**

**Category (autobiography | non-fiction | fiction) #REQUIRED**

**InStock (true | false) "false"**

**Reviewer CDATA "">**

```

<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!-- A Date may have a Month. It must have a Year. -->
<!ELEMENT Date (Month?, Year)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Year (#PCDATA)>

```

Ако декларацията не е  
нито #REQUIRED, нито #IMPLIED,  
тогава се задава стойността по  
подразбиране

BookCatalogue2.dtd

.....

```
<element name="BookCatalogue">
  <annotation>
    <info>A book catalogue contains zero or more books</info>
  </annotation>
  <complexType>
    <element ref="cat:Book" minOccurs="0" maxOccurs="*" />
  </complexType>
</element>

<element name="Book">
  <annotation>
    <info>A Book has a Title, one or more Authors, a Date, an ISBN, and a Publ.</info>
  </annotation>
  <complexType>
    <element ref="cat:Title" />
    <element ref="cat:Author" maxOccurs="unbounded" />
    <element ref="cat:Date" />
    <element ref="cat:ISBN" />
    <element ref="cat:Publisher" />
    <attributeGroup ref="BookAttributes" />
  </complexType>
</element>
```

**<attributeGroup name="BookAttributes">**

**<annotation>**

**<info>**

A Book has three attributes - Category, InStock, and Reviewer. Category must be either "autobiography", "non-fiction", or "fiction". A value must be supplied for this attribute whenever a Book element is used within a document. InStock can be either "yes" or "no". If no value is supplied it defaults to "no". Reviewer contains the name of the reviewer. It defaults to "" if no value is supplied.

**</info>**

**</annotation>**

**<attribute name="Category" minOccurs="1">**

**<datatype source="string">**

**<enumeration value="autobiography"/>**

**<enumeration value="non-fiction"/>**

**<enumeration value="fiction"/>**

**</datatype>**

**</attribute>**

**<attribute name="InStock" type="boolean" default="false"/>**

**<attribute name="Reviewer" type="string" default=""/>**

**</attributeGroup>**

**<element name="Title" type="string"/>**

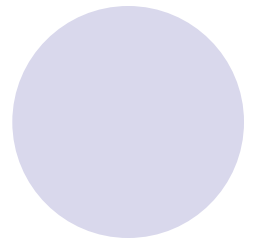
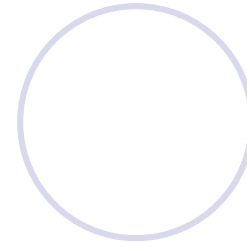
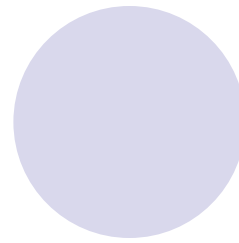
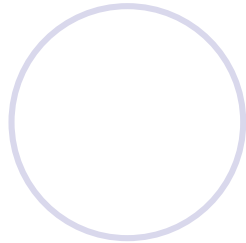
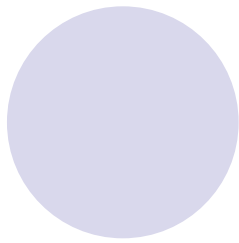
**<element name="Author" type="string"/>**

**<element name="Date" type="string"/>**

**<element name="ISBN" type="string"/>**

**<element name="Publisher" type="string"/>**

**</schema>**



```
<attribute name="Category" minOccurs="1">  
  <datatype source="string">  
    <enumeration value="autobiography"/>  
    <enumeration value="non-fiction"/>  
    <enumeration value="fiction"/>  
  </datatype>  
</attribute>
```



“По подразбиране стойността на  
**maxOccurs** е 1. За **minOccurs="1"**  
=> такъв атрибут е **REQUIRED.**”

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:cat CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/BookCatalogue"
  xmlns:cat="http://www.somewhere.org/BookCatalogue">
  <element name="BookCatalogue">
    <complexType>
      <element name="Book" minOccurs="0" maxOccurs="*">
        <complexType>
          <element name="Title" type="string"/>
          <element name="Author" type="string"/>
          <element name="Date" type="string"/>
          <element name="ISBN" type="string"/>
          <element name="Publisher" type="string"/>
          <attribute name="Category" minOccurs="1">
            <datatype source="string">
              <enumeration value="autobiography"/>
              <enumeration value="non-fiction"/>
              <enumeration value="fiction"/>
            </datatype>
          </attribute>
          <attribute name="InStock" type="boolean" default="false"/>
          <attribute name="Reviewer" type="string" default=""/>
        </complexType>
      </element>
    </complexType>
  </element>
</schema>

```

# Още за атрибутите

- **ДЕКЛАРАЦИИТЕ НА АТРИБУТИТЕ  
ВИНАГИ СА СЛЕД ТЕЗИ НА  
ЕЛЕМЕНТИТЕ, ЗА КОИТО ТЕ СЕ  
ДЕКЛАРИРАТ.**

# Още примери:

<http://www.zvon.org/xxl/XMLSchemaTutorial/Output/highlights.html>

Let's say we want to define a group of common attributes, which will be reused. The root element is named "root", it must contain the "aaa" and "bbb" elements, and these elements must have attributes "x" and "y".

## Valid document

```
<root xsi:noNamespaceSchemaLocation="correct_0.xsd" xmlns=""  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >  
  <aaa x="1" y="2"/>  
  <bbb x="3" y="4"/>  
</root>
```

## Invalid document

Attribute "x" is missing.

```
<root xsi:noNamespaceSchemaLocation="correct_0.xsd" xmlns=""  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >  
  <aaa y="2"/>  
  <bbb x="3" y="4"/>  
</root>
```

## Invalid document

Attribute "z" is not allowed.

```
<root xsi:noNamespaceSchemaLocation="correct_0.xsd" xmlns=""  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >  
  <aaa x="1" y="2" z="2"/>  
  <bbb x="3" y="4"/>  
</root>
```

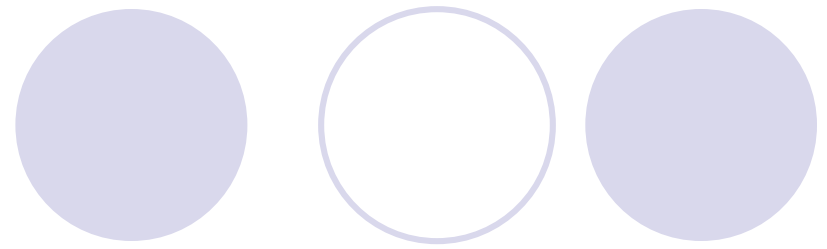
## Correct XML Schema (correct\_0.xsd)

Do not forget to add the attribute "use" set to "required" when declaring attribute (default value is "optional").

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >  
  <xsd:element name="root">  
    <xsd:complexType>  
      <xsd:sequence>  
        <xsd:element name="aaa" minOccurs="1" maxOccurs="1">  
          <xsd:complexType>  
            <xsd:attributeGroup ref="myAttrs"/>  
          </xsd:complexType>  
        </xsd:element>  
        <xsd:element name="bbb" minOccurs="1" maxOccurs="1">  
          <xsd:complexType>  
            <xsd:attributeGroup ref="myAttrs"/>  
          </xsd:complexType>  
        </xsd:element>  
      </xsd:sequence>  
    </xsd:complexType>  
  </xsd:element>  
  <xsd:attributeGroup name="myAttrs">  
    <xsd:attribute name="x" type="xsd:integer" use="required"/>  
    <xsd:attribute name="y" type="xsd:integer" use="required"/>  
  </xsd:attributeGroup>  
</xsd:schema>
```



# Елемент group



- Елементът **group** ни позволява да групираме заедно декларации на елементи.
- Заб.: елементът group разрешава многократно използване на елементни декларации, но без атрибути в тях.

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd" [
<!ATTLIST schema xmlns:cat CDATA #IMPLIED> ]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/BookCatalogue"
  xmlns:cat="http://www.somewhere.org/BookCatalogue">
  <element name="BookCatalogue">
    <complexType>
      <element name="Book" minOccurs="0" maxOccurs="*">
        <complexType>
          <group ref="cat:BookElements"/>
          <attribute name="Category" minOccurs="1">
            <datatype source="string">
              <enumeration value="autobiography"/>
              <enumeration value="non-fiction"/>
              <enumeration value="fiction"/>
            </datatype>
          </attribute>
          <attribute name="InStock" type="boolean" default="false"/>
          <attribute name="Reviewer" type="string" default=""/>
        </complexType>
      </element>
    </complexType>
  </element>
  <group name="BookElements" order="seq">
    <element name="Title" type="string"/>
    <element name="Author" type="string"/>
    <element name="Date" type="string"/>
    <element name="ISBN" type="string"/>
    <element name="Publisher" type="string"/>
  </group>
</schema>

```

# Задаване на алтернативи

DTD: `<!ELEMENT signature (name | (name, date))>`

XML Schema:

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:sig CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/Examples"
  xmlns:sig="http://www.somewhere.org/Examples">
  <element name="signature">
    <complexType>
      <group order="choice">
        <element ref="sig:name"/>
        <group order="seq">
          <element ref="sig:name"/>
          <element name="date" type="date"/>
        </group>
      </group>
    </complexType>
  </element>
  <element name="name" type="string"/>
</schema>
```

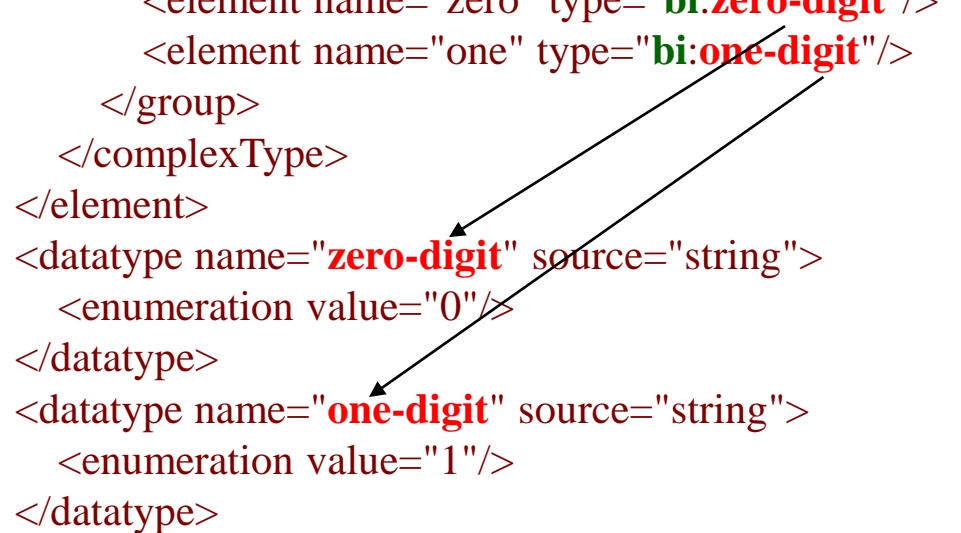
# Задаване на повторения

DTD:

```
<!ELEMENT binary-digit (zero | one)+>
```

XML Schema:

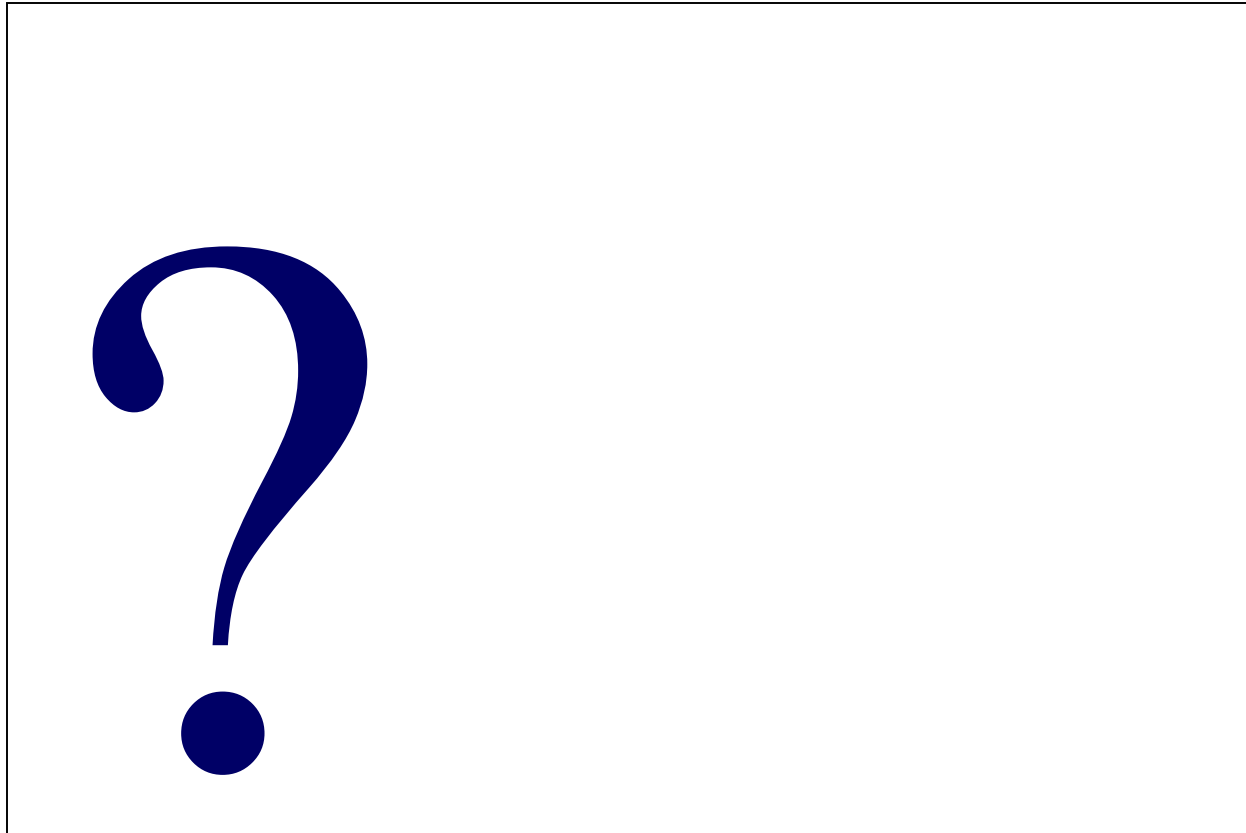
```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd" [
<!ATTLIST schema xmlns:bi CDATA #IMPLIED> ]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/Examples"
  xmlns:bi="http://www.somewhere.org/Examples">
  <element name="binary-digit">
    <complexType>
      <group order="choice" maxOccurs="unbounded">
        <element name="zero" type="bi:zero-digit"/>
        <element name="one" type="bi:one-digit"/>
      </group>
    </complexType>
  </element>
  <datatype name="zero-digit" source="string">
    <enumeration value="0"/>
  </datatype>
  <datatype name="one-digit" source="string">
    <enumeration value="1"/>
  </datatype>
</schema>
```



# Задаване на произволно редуване (Any Order)

Проблем: елементът Book да съдържа Author, Title, Date, ISBN, and Publisher,  
*в any order* (как беше в *DTD?!).*

XML Schema:



***order="all"** -> Book трябва да съдържа петте елемента, но в произволен ред.*  
*Заб.: minOccurs, maxOccurs с ФИКСИРАНИ СЪС СТОЙНОСТ "1".*

# Пример

Елементът Book да съдържа Author, Title, Date, ISBN, and Publisher,  
*в any order*

XML Schema:

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd">
<schema xmlns="http://www.w3.org/1999/XMLSchema"
        targetNamespace="http://www.somewhere.org/Examples">
  <element name="Book">
    <type>
      <group order="all">
        <element name="Title" type="string"/>
        <element name="Author" type="string"/>
        <element name="Date" type="date"/>
        <element name="ISBN" type="string"/>
        <element name="Publisher" type="string"/>
      </group>
    </type>
  </element>
</schema>
```

# Вместо елемента group:

- DTD

- `<!ELEMENT`

- `e1`

- `( (e2,e3?)+ | e4 ) >`

- Schema

- `<element name="e1">`

- `<complexType>`

- `<choice>`

- `<sequence`

- `maxOccurs="unbounded">`

- `<element ref="e2"/>`

- `<element ref="e3"`

- `minOccurs="0"/>`

- `</sequence>`

- `<element ref="e4"/>`

- `</choice>`

- `</complexType>`

- `</element>`

# Празен елемент

Schema:

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd">
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/Examples">
  <element name="image">
    <simpleType content="empty">
      <attribute name="href" type="uri"/>
    </simpleType>
  </element>
</schema>
```

Instance doc:

```
<image href="http://www.xfront.org/Rog.gif"/>
```



# Уникалност

- DTD предоставя атрибутния тип данни **ID** с цел деклариране на уникалност (*дадена ID атрибутна стойност трябва да бъде уникална за целия документ и XML парсерът следи за това*).
- В XML Schema можем да дефинираме много повече уникалност (*uniqueness*) на съдържанието:
  1. уникално съдържание на елемент.
  2. non-ID атрибути да бъдат уникални.
  3. комбинация между елементи и атрибути да бъде уникална.
  4. разлика между unique и key.
  5. обхват в документа, за който нещо е unique.

# unique спрямо key?

- Key: елемент или атрибут (или комбинация от тях), който е деклариран като ключ (key), трябва:
  - Винаги да е наличен (minOccurs да е по-голямо от 0)
  - Да не може да е null (тоест nullable="false")
  - Да е уникален (unique)
- *Key предполага unique, но unique не предполага непременно key*



The diagram illustrates XML code with annotations for key and keyref attributes. A light blue circle is in the top right corner. Arrows point from text labels to specific parts of the XML code.

```

<?xml version="1.0"?>
<Library xmlns="http://www.somewhere.org/Library">
  <BookCatalogue>
    <Book Category = "fiction" InStock = "yes" Reviewer = "John Doe">
      <Title>Illusions The Adventures of a Reluctant Messiah</Title>
      <Author>Richard Bach</Author>
      <Date>1977</Date>
      <ISBN>0-440-34319-4</ISBN>
      <Publisher>Dell Publishing Co.</Publisher>
    </Book>
    <Book Category = "non-fiction" InStock = "yes" Reviewer = "John Doe">
      <Title>The First and Last Freedom</Title>
      <Author>J. Krishnamurti</Author>
      <Date>1954</Date>
      <ISBN>0-06-064831-7</ISBN>
      <Publisher>Harper & Row</Publisher>
    </Book>
  </BookCatalogue>
  <CheckoutRegister>
    <Person>Roger Costello</Person>
    <Book titleRef="Illusions The Adventures of a Reluctant Messiah"
      categoryRef="fiction"/>
  </CheckoutRegister>
</Library>

```

Annotations:

- + = key**: Points to the `Category = "fiction"` attribute in the first `<Book>` element.
- + = keyref**: Points to the `categoryRef="fiction"` attribute in the `<Book>` element within `<CheckoutRegister>`.

```

<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:lib CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/Library"
  xmlns:lib="http://www.somewhere.org/Library">
  <element name="Library">
    <complexType>
      <element name="BookCatalogue">
        <complexType>
          <element name="Book" minOccurs="0" maxOccurs="unbounded">
            <complexType>
              <element name="Title" type="string"/>
              <element name="Author" type="string"/>
              <element name="Date" type="string"/>
              <element name="ISBN" type="string"/>
              <element name="Publisher" type="string"/>
              <attribute name="Category" minOccurs="1">
                <datatype source="string">
                  <enumeration value="autobiography"/>
                  <enumeration value="non-fiction"/>
                  <enumeration value="fiction"/>
                </datatype>
              </attribute>
              <attribute name="InStock" type="boolean" default="false"/>
              <attribute name="Reviewer" type="string" default=""/>
            </complexType>
          </element>
        </complexType>
      </element>
    </complexType>
  </element>

```

The default value for both the minOccurs and the maxOccurs attributes is 1.

```
<element name="CheckoutRegister">
  <type>
    <element name="Person" type="string"/>
    <element name="Book">
      <type content="empty">
        <attribute name="titleRef" type="string"/>
        <attribute name="categoryRef" type="string"/>
      </type>
    </element>
  </type>
</element>
</type>
</element>
<key name="bookKey">
  <selector>Library/BookCatalogue/Book</selector>
  <field>Title</field>
  <field>@Category</field>
</key>
<keyref name="bookRef" refer="bookKey">
  <selector>Library/CheckoutRegister/Book</selector>
  <field>@titleRef</field>
  <field>@categoryRef</field>
</keyref>
</schema>
```

# Дефиниране на ключ (Key)

```
<key name="bookKey">  
  <selector>Library/BookCatalogue/Book</selector>  
  <field>Title</field>  
  <field>@Category</field>  
</key>
```

“Комбинацията от съдържанието на елемента Title плюс стойността на атрибута Category ще бъде уникална за целия XML документ-екземпляр. Елементът Title и атрибутът Category са към елемента **Book**, зададен като селектор чрез XPath израз.”

Заб.: можем да задаваме едно и повече полета (field), като ключът е комбинацията от всички тях в зададения ред.

# Дефиниране на референция (Reference) към ключ (Key)

```
<keyref name="bookRef" refer="bookKey">  
  <selector>Library/CheckoutRegister/Book</selector>  
  <field>@titleRef</field>  
  <field>@categoryRef</field>  
</keyref>
```

“Два атрибута - *titleRef* и *categoryRef*, заедно са референция към ключа, дефиниран от *bookKey*. Те принадлежат към селектора, дефиниран чрез Xpath израз.”

Note: полетата на keyref трябва да съответстват на типа и на позицията на тези в ключа.

# Задаване на уникалност (Uniqueness)

```
<unique name="bookUnique">  
  <selector>Library/BookCatalogue/Book</selector>  
  <field>Title</field>  
  <field>@Category</field>  
</unique>
```



# Обхват на уникалността в XML Schema

- Елементите `key/keyref/unique` могат да се намират където и да е в схемата.
- Местоположението им определя обхвата на уникалността.
- В примера, `key/keyref` елементите се намират на *top-level* (в прекия наследник на елемента `schema`). *Затова уникалността им е за целия документ.*
- Ако обаче разположим елементите `key/keyref` като деца на елемента `Book`, то уникалността им ще е само за елементите `Book` в документа-екземпляр.

# any елемент

- Елементът **any** задава какъв да е well-formed XML
- Пример – произволен брой елементи от което и да е пространство от имена:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
  <xsd:element name="root">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The free-form element can contain any Well-Formed XML (WFXML).

```
<root>
  <comment source="BB">This is great!</comment>
</root>
```

# anyAttribute

- Елементът anyAttribute задава какъв да е атрибут от кое и да е пространство от имена:

```
<xsd:schema  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" >  
  <xsd:element name="root">  
    <xsd:complexType>  
      <xsd:anyAttribute namespace="##any"/>  
    </xsd:complexType>  
  </xsd:element>  
</xsd:schema>
```

```
<root comment="This is great"/>
```

# (Почти) any елемент / атрибут

`<any namespace="##other"/>`

- allows any well-formed XML element, provided the element is in another namespace than the one we're defining.

`<any namespace="http://www.somewhere.com"/>`

- allows any well-formed XML element, provided it's from the specified namespace.

`<any namespace="##targetNamespace"/>`

- allows any well-formed XML element, provided it's from the namespace that we're defining.

```
<Library xmlns:book="http://www.somewhere.org/Examples"
  xmlns:person="http://www.somewhere-else.org/Person"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema/instance"
  xsi:schemaLocation=
    "http://www.somewhere.org/Examples
    http://www.somewhere.org/Examples/Book.xsd
    http://www.somewhere-else.org/Person
    http://www.somewhere-else.org/Person/Person.xsd">
```

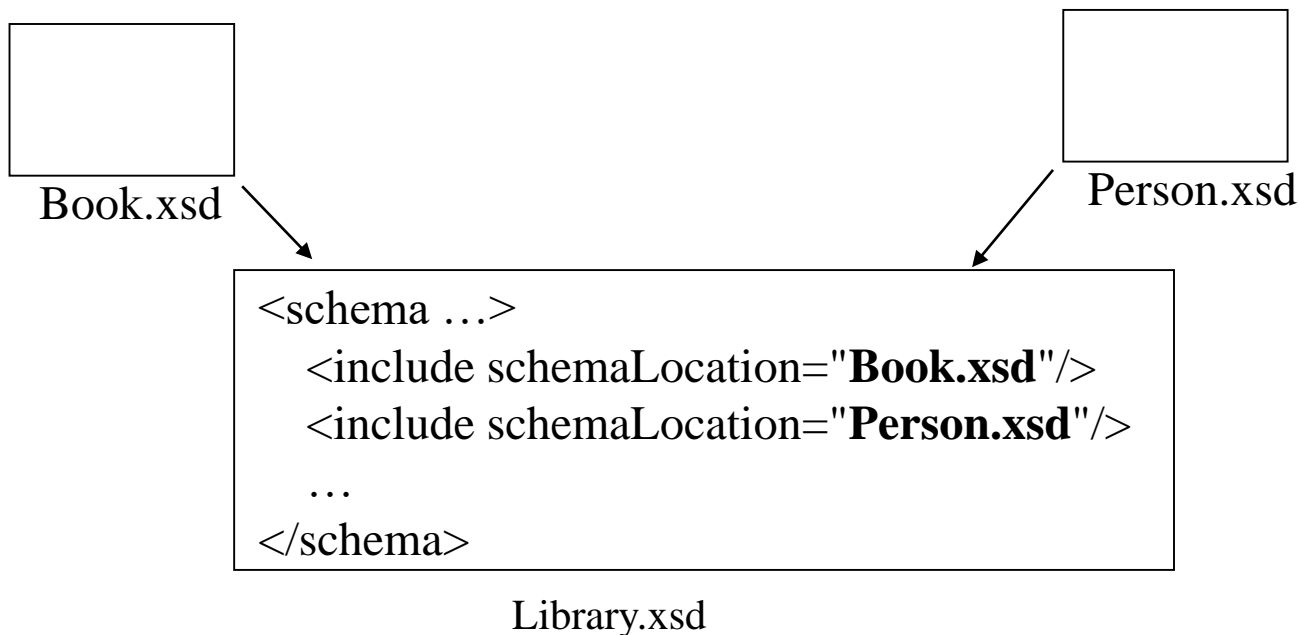
```
<BookCatalogue>
  <book:Book>
    <book:Title>Illusions The Adventures of a Reluctant Messiah</book:Title>
    <book:Author>Richard Bach</book:Author>
    <book:Date>1977</book:Date>
    <book:ISBN>0-440-34319-4</book:ISBN>
    <book:Publisher>Dell Publishing Co.</book:Publisher>
  </book:Book>
  <book:Book>
    <book:Title>The First and Last Freedom</book:Title>
    <book:Author>J. Krishnamurti</book:Author>
    <book:Date>1954</book:Date>
    <book:ISBN>0-06-064831-7</book:ISBN>
    <book:Publisher>Harper & Row</book:Publisher>
  </book:Book>
</BookCatalogue>
<Employees>
  <person:Employee>
    <person:Name>John Doe</person:Name>
    <person:SSN>123-45-6789</person:SSN>
  </person:Employee>
  <person:Employee>
    <person:Name>Sally Smith</person:Name>
    <person:SSN>000-11-2345</person:SSN>
  </person:Employee>
</Employees>
```

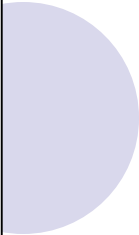
```
</Library>
```

Валидиране  
на документ  
с две схеми

# Съставяне на схема от няколко схеми

- Елементът **include** задава включване в дадена схема на дефиниции от няколко схеми
  - Те трябва да имат едно и също пространство от имена
  - Внасят дефинициите директно в съдържащата ги схема





```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd"[
<!ATTLIST schema xmlns:lib CDATA #IMPLIED>
]>
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  targetNamespace="http://www.somewhere.org/Examples"
  xmlns:lib="http://www.somewhere.org/Examples">
  <include schemaLocation="http://www.somewhere.org/Examples/Book.xsd"/>
  <include schemaLocation="http://www.somewhere.org/Examples/Person.xsd"/>
  <element name="Library">
    <type>
      <element name="BookCatalogue">
        <type>
          <element ref="lib:Book" minOccurs="0" maxOccurs="*" />
        </type>
      </element>
      <element name="Employees">
        <type>
          <element name="lib:Employee" minOccurs="0" maxOccurs="*" />
        </type>
      </element>
    </type>
  </element>
</schema>
```

# Елемент import

- Елементът **import** разрешава да се реферират елементи от друго пространство от имена

```
<schema xmlns="http://www.w3.org/1999/XMLSchema"
        targetNamespace="http://www.somewhere.org/Examples"
        xmlns:html="http://www.w3.org/1999/XHTML">
  <import namespace="http://www.w3.org/1999/XHTML"
          schemaLocation="http://www.w3.org/XHTML/xhtml.xsd"/>
  ...
  <element ref="html:table">
  ...
</schema>
```

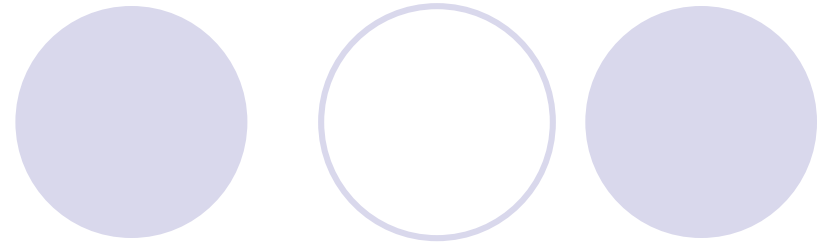


# Декларация или дефиниция

- Използваме декларации (*declarations*) за това, което ще се съдържа в XML документа-екземпляр.
- Използваме дефиниции (*definitions*) за задаване на нови типове и групи – те се използват в схемата

Declarations: <ul style="list-style-type: none"><li>- element declarations</li><li>- attribute declarations</li></ul>	Definitions: <ul style="list-style-type: none"><li>- type definitions</li><li>- attribute group definitions</li><li>- model group definitions</li></ul>
---	---

# xsd:ENTITY 1/2



- XML Schema, както и другите schema езици не предоставят заместващ механизъм за деклариране на **DTD entities**
- Типът **xsd:ENTITY** представя референция към unparsed entity и се ползва за включване на документи, които не са в XML формат.
- Стойността на xsd:ENTITY трябва да бъде **NCName** (non-colonized name, или обратното на **QName**)
- Стойността на xsd:ENTITY трябва да бъде **unparsed entity** от DTD дефиницията за документа-екземпляр.

# xsd:ENTITY 2/2

## Schema:

```
<xs:element name="picture">
  <xs:complexType>
    <xs:attribute name="location"
      type="xs:ENTITY"/>
  </xs:complexType>
</xs:element>
<!--...-->
```

## Instance:

```
<!DOCTYPE catalog SYSTEM "catalog.dtd" [
  <!NOTATION jpeg SYSTEM "JPG">
  <!ENTITY prod557 SYSTEM "prod557.jpg"
    NDATA jpeg>
  <!ENTITY prod563 SYSTEM "prod563.jpg"
    NDATA jpeg> ]>
<catalog>
  <product>
    <number>557</number>
    <picture location="prod557"/>
  </product>
  <product>
    <number>563</number>
    <picture location="prod563"/>
  </product>
</catalog>
```

# Управление на версиите на схемата

- За документиране

```
<?xml version="1.0"?>
<!DOCTYPE schema SYSTEM "xml-schema.dtd">
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.somewhere.org/Examples"
        version="1.0">
    ...
</schema>
```

# <simple/complex-type> или <datatype>?

- Кога да използваме simple/complex-type и datatype?
  - Използваме <simple/complex-type> елементи за деклариране на типове на елементи и/или атрибути
  - Използваме datatype елемента за деклариране на производен типове (на базата на string, integer, ...)
- XML Schema елементът с име simpleContent задава разширения или ограничения върху:
  - complex type, който е само текст
  - simple type
- XML елемент от тип simpleContent няма под-елементи

# null съдържание на елемент

- Empty спрямо null:
  - **empty**: елемент от тип `content="empty"` е празен - не може да има съдържание.
  - **null**: този елемент в документа екземпляр може да бъде недефиниран – тогава `xsi:null` атрибутът му ще е 'true'

XML Schema:

```
<element name="PersonName">
  <complexType><sequence>
    <element name="forename" type="NMTOKEN"/>
    <element name="middle" type="NMTOKEN" nullable="true"/>
```

XML instance document:

```
<PersonName>
  <forename>John</forename>
  <middle xsi:null="true"/>
```

Съдържанието на middle може да е NMTOKEN или да не е дефинирано!

# Заключение

- DTD (Document Type Definition) и XML schema или XSD (XML Schema Definition) задават правила, съгласно които се определят имената на елементите и атрибутите, тяхната последователност, честота на срещане и др.
- DTD използва по-стегнат и кратък синтаксис в сравнение с XML Schema, но за сметка на това XML Schema предоставя по-богат набор от средства за по-строго дефиниране на структурата на XML и освен това нейните правила се задават в XML формат.
- Още онлайн ресурси:
  - [https://en.wikibooks.org/wiki/XML\\_Schema](https://en.wikibooks.org/wiki/XML_Schema)
  - [https://www.w3schools.com/xml/schema\\_intro.asp](https://www.w3schools.com/xml/schema_intro.asp)
  - <https://www.w3.org/XML/Schema>

