Grammars for XML Documents

XML Schema, Part 1

Lecture "XML in Communication Systems"
Chapter 4

Dr.-Ing. Jesper Zedlitz
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel

Acknowledgement

- This chapter is based on:

  Roger L. Costello: XML Technologies Course

  http://www.xfront.com/files/tutorials.html
  Copyright (c) 2000. Roger L. Costello. All Rights Reserved.

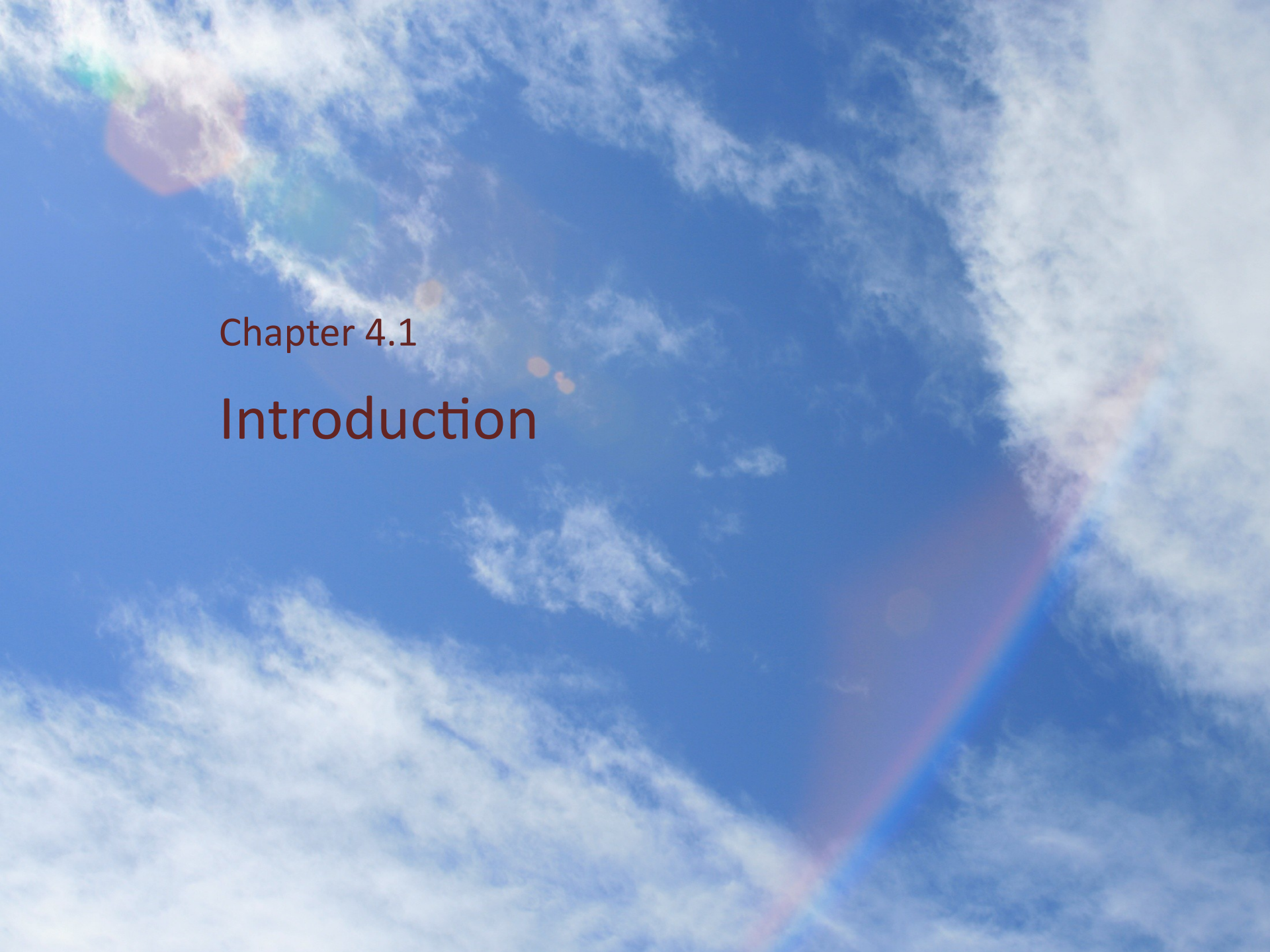
Recommended Reading

- David C. Fallside, Priscilla Walmsley (ed.):
  XML Schema Part 0: Primer. Second Edition.
  W3C Recommendation 28 October 2004.
  http://www.w3.org/TR/xmlschema-0/

- Paul V. Biron, Ashok Malhotra (ed.):
  XML Schema Part 2: Datatypes. Second Edition.
  W3C Recommendation 28 October 2004.
  http://www.w3.org/TR/xmlschema-2/

Overview

1. Introduction
2. Getting started
3. Three XML Schema flavours
4. Simple types
5. List and Union datatypes
6. Annotations



Chapter 4.1

Introduction

Motivation for XML Schema

- People are dissatisfied with DTDs
  - It's a different syntax
    - You write your XML (instance) document using one syntax and the DTD using another syntax → bad, inconsistent
  - Limited datatype capability
    - DTDs supports a single datatype: text
    - Desired: a set of datatypes compatible with those in databases
    - Create your own datatypes: "This is a new type based on the string type and elements of this type must follow this pattern: `ddd-dddd`, where 'd' represents a digit".

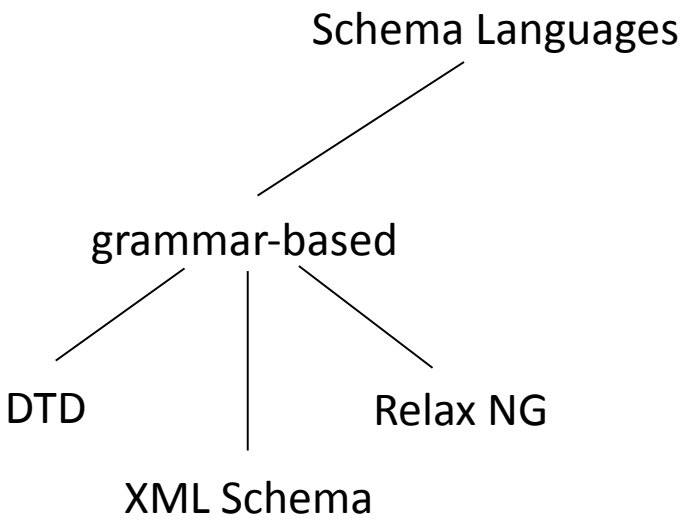
Introduction

- What is XML Schema?

An XML vocabulary and grammar
for expressing your data's business rules.

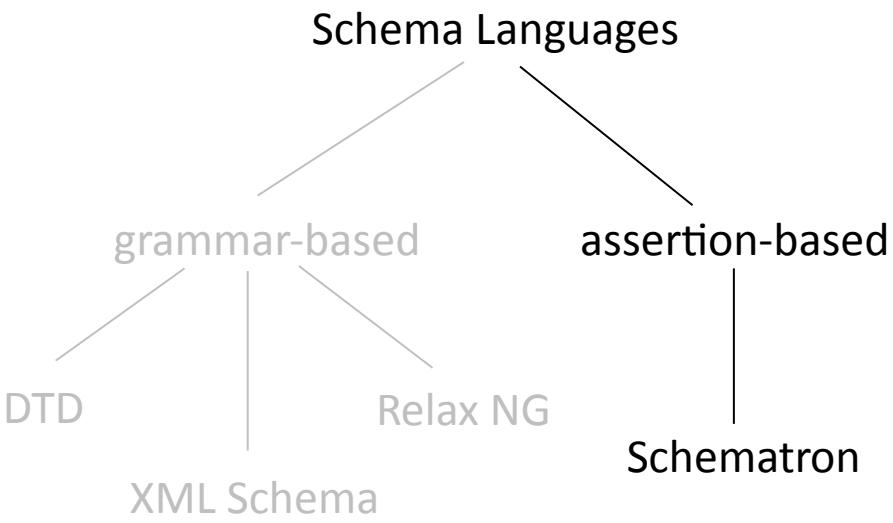
Introduction

- Taxonomy for schema languages
  - A grammar-based schema specifies
    - what elements may be used in an XML instance document,
    - the order of the elements,
    - the number of occurrences of each element, and
    - the datatype of each element and its attributes.



Introduction

- Taxonomy for schema languages
  - An assertion-based schema makes assertions about the relationships that must hold between the elements and attributes in an XML instance document.



In this chapter we discuss the grammar-based W3C XML Schema language.

Introduction

- Introductory example
  - sample instance document

```
<location>
     <latitude>32.904237</latitude>
     <longitude>73.620290</longitude>
     <uncertainty units="meters">2</uncertainty>
</location>
```

Introduction

- To be valid, an instance document must meet the following constraints:
  1. The location must be comprised of a latitude, followed by a longitude, followed by an indication of the uncertainty of the lat/lon measurements.
  2. The latitude must be a decimal with a value between -90 to +90.
  3. The longitude must be a decimal with a value between -180 to +180.
  4. For both latitude and longitude the number of digits to the right of the decimal point must be exactly six digits.
  5. The value of uncertainty must be a non-negative integer.
  6. The uncertainty units must be either meters or feet.

Introduction


```
<location>
      <latitude>32.904237</latitude>
      <longitude>73.620290</longitude>
      <uncertainty units="meters">2</uncertainty>
</location>
```

XML instance doc

XML Schema validator

Ok!

Declare a location element. Require that its content be latitude, longitude, and uncertainty.
Declare a latitude element. Require that its value be between -90 and +90.
Declare a longitude element. Require that its value be between -180 and +180.
Declare a uncertainty element with a units attribute.
Require that the element's value be between 0 and 10.
Require that the attribute's value be either feet or meters.

Informal grammar
→ XML Schema

Introduction

- What does an XML Schema accomplish?
  - Answer: It creates an XML vocabulary:

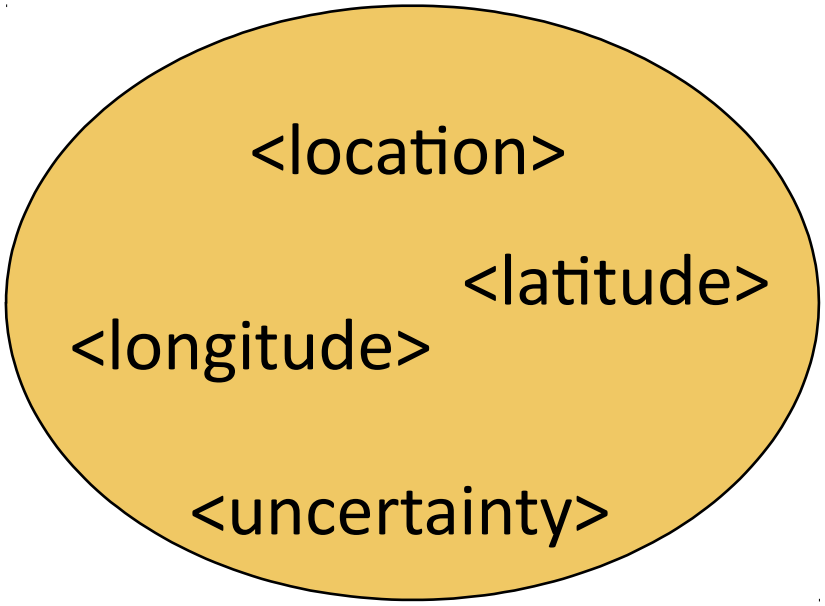
<location>, <latitude>, <longitude>, <uncertainty>

  - It specifies the contents of each element, and the restrictions on the content.
  - It specifies the attributes of each element.
  - An XML Schema specifies that the XML vocabulary that is being created shall be in a "namespace".

Introduction

- Namespace

http://www.example.org/target



Introduction

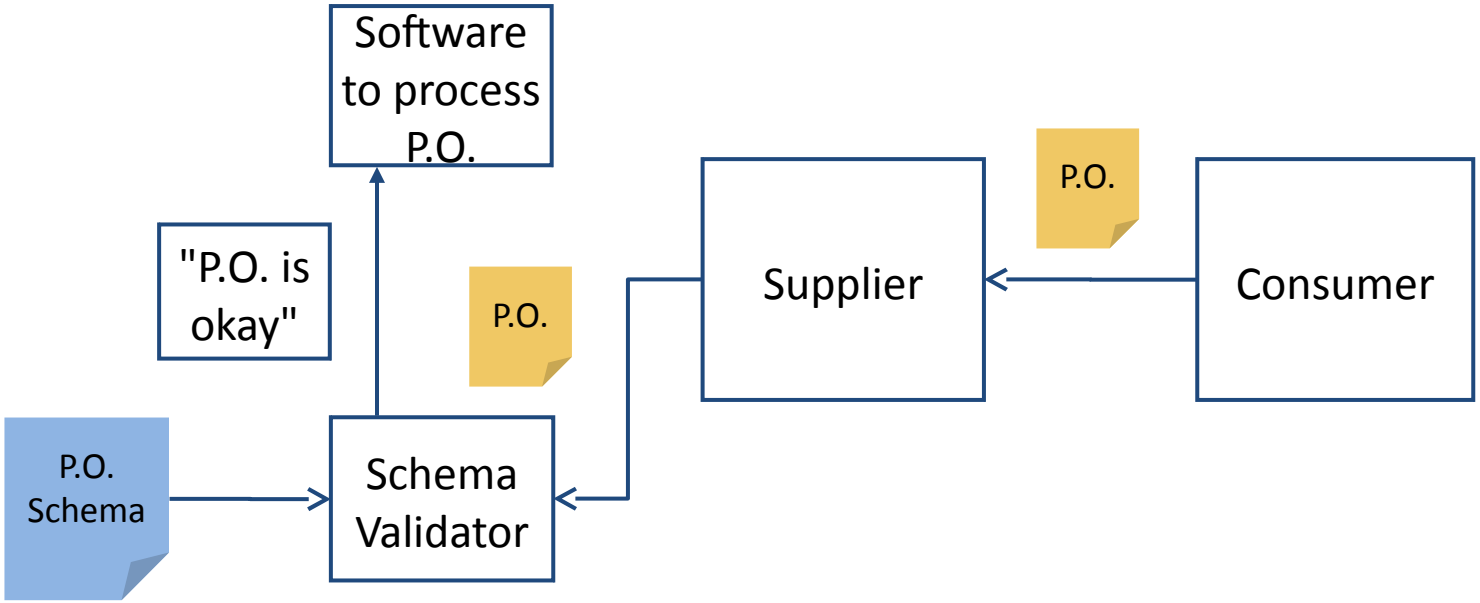
- Constraints: in a document or in code?
  - An XML Schema is an XML document.
    The constraints on the data are expressed in a document.
  - All of the constraints could be expressed in a programming language as well in your system's middleware.

☞ Why in a document?

  - By expressing the data constraints in a document,
    the schema itself becomes information!
  - The schema can be shipped around, mined, morphed, searched, etc.
  - Don't bury your data constraints within code in middleware.
    Information is king!

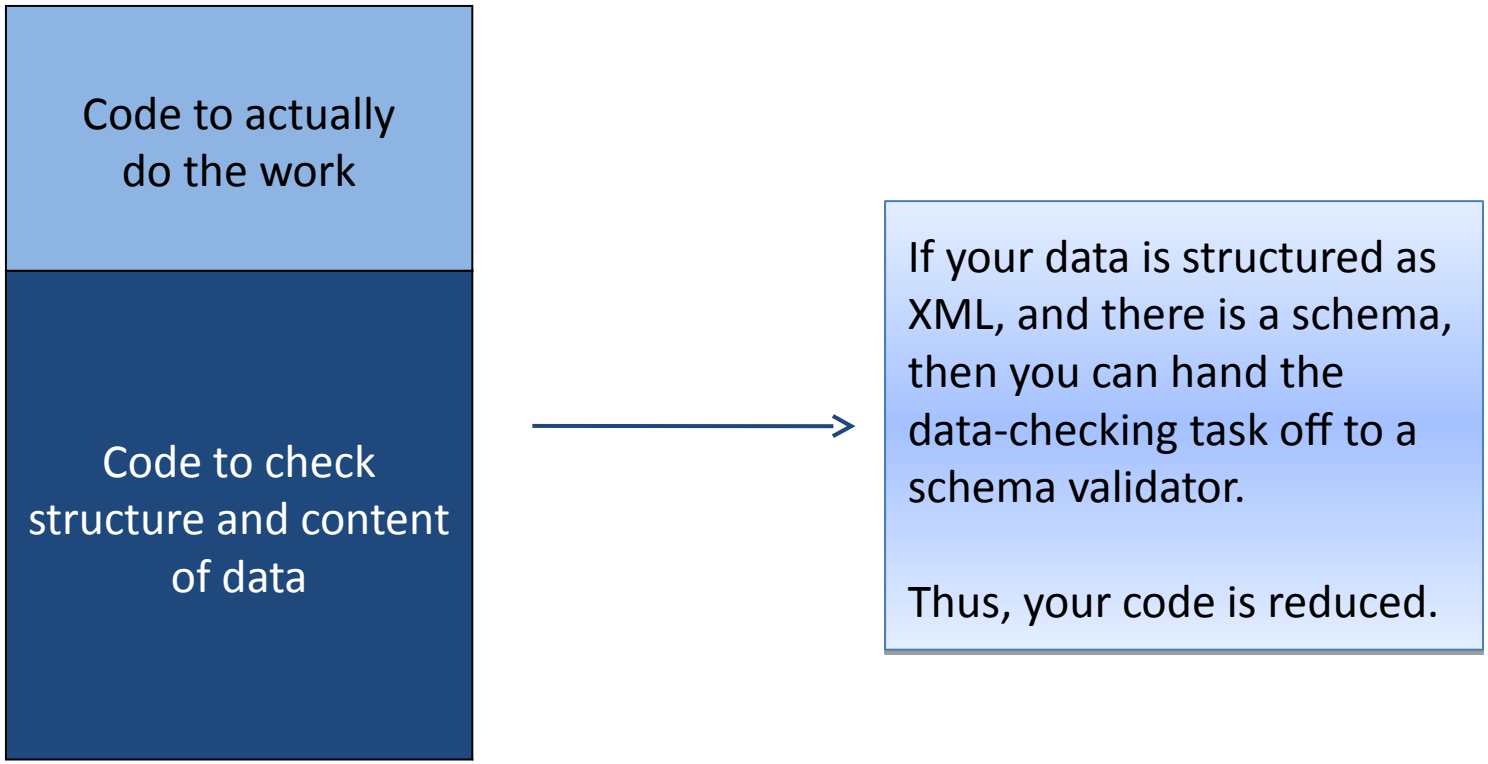
Introduction

- Use of XML Schemas
  - example: purchase order processing



Introduction

- Save $$$ using XML Schemas



Introduction

- What are XML Schemas?
  - Data Model
    - With XML Schemas you specify how your XML data will be organized, and the datatypes of your data. That is, with XML Schemas you model how your data is to be represented in an instance document.
  - A Contract
    - Organizations agree to structure their XML documents in conformance with an XML Schema. Thus, the XML Schema acts as a contract between the organizations.

Introduction

- What are XML Schemas? (cont'd.)
  - A rich source of metadata
    - An XML Schema document contains lots of data about the data in the XML instance documents, such as the datatype of the data, the data's range of values, how the data is related to another piece of data (parent/child, sibling relationship), i.e., XML Schemas contain metadata

Highlights of XML Schema

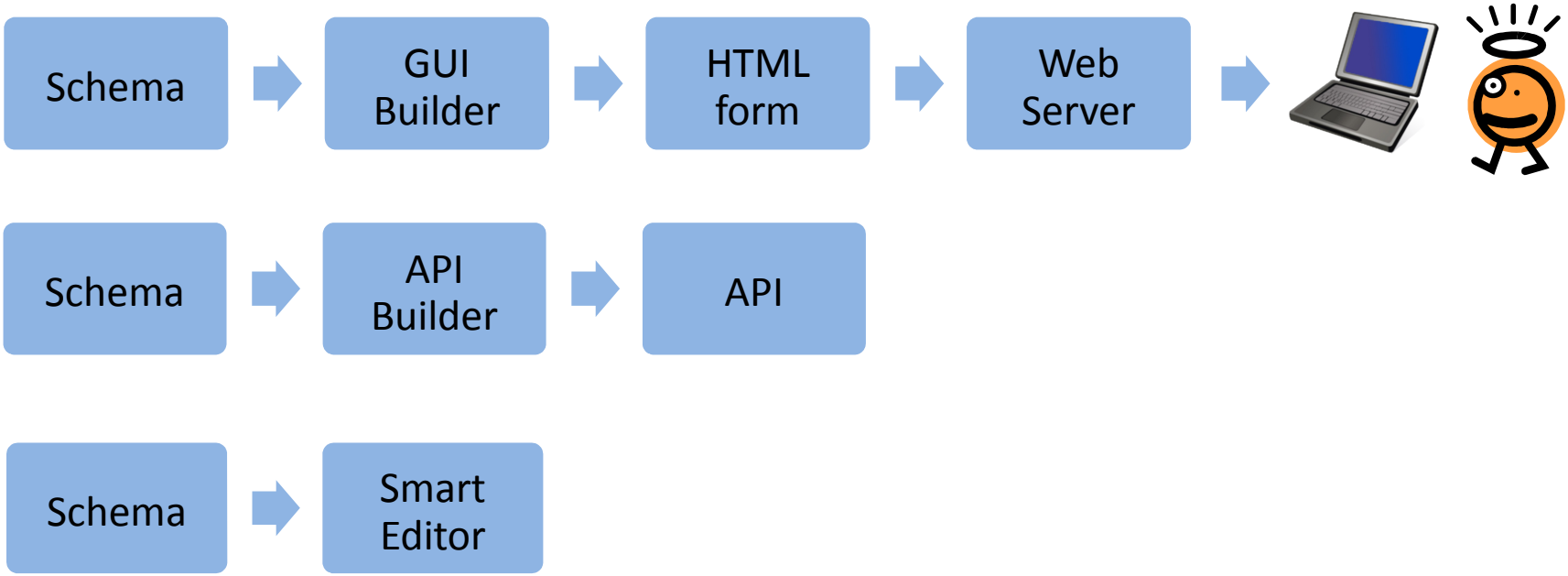
- XML Schema is a tremendous advancement over DTDs
  - Can express sets, i.e., can define the child elements to occur in any order
  - Can specify element content as being unique (keys on content) and uniqueness within a region
  - Can define multiple elements with the same name but different content
  - Object-oriented'ish: can extend or restrict a type: derive new type definitions on the basis of old ones

Highlights of XML Schema

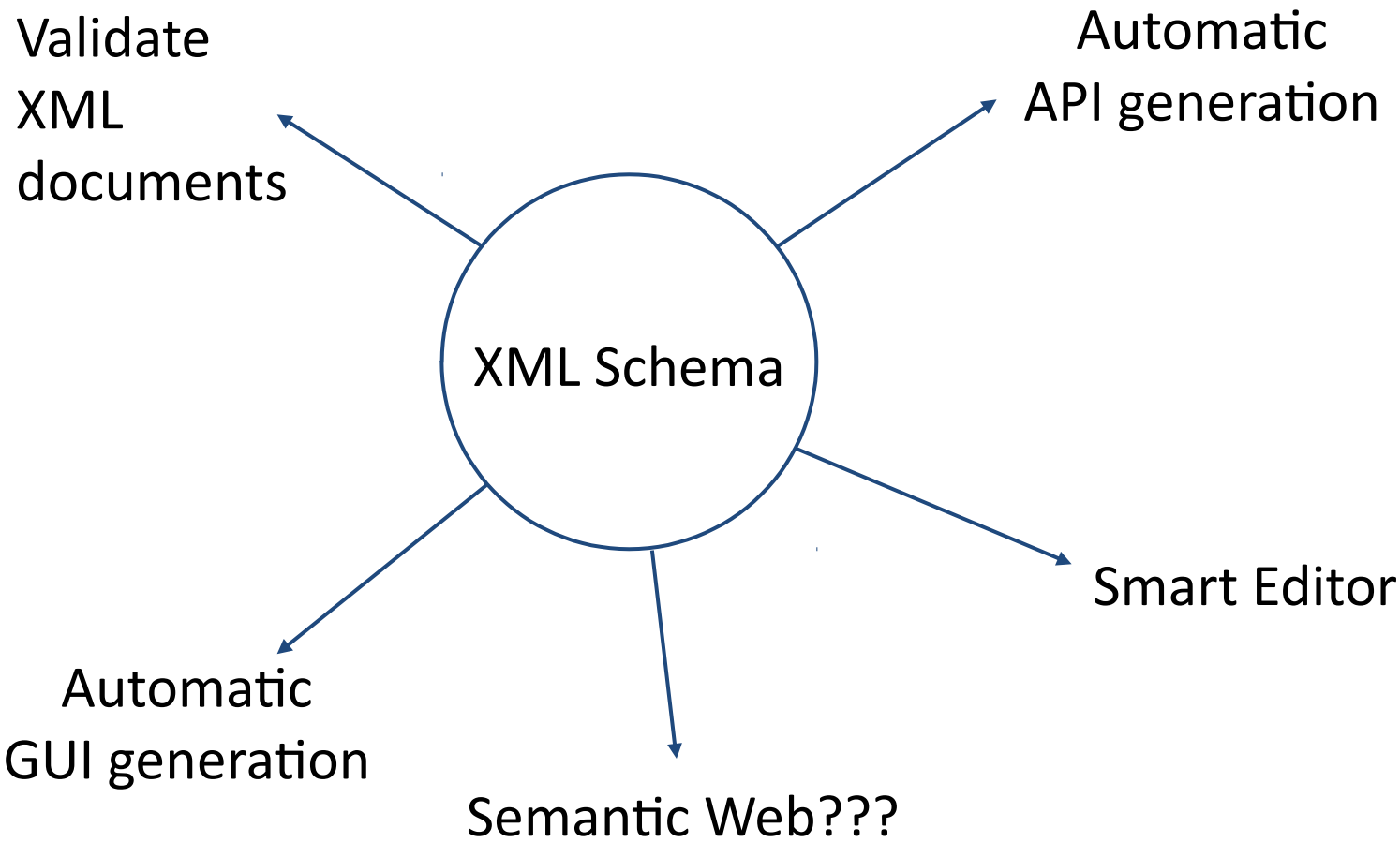
- XML Schema is a tremendous advancement over DTDs
  - Can define elements with nil content
  - Can define substitutable elements, e.g., the "Book" element is substitutable for the "Publication" element.

Highlights of XML Schema

- No Limits
  - There are many other uses for XML Schemas. Schemas are a wonderful source of metadata.



Highlights of XML Schema



Chapter 4.2

Getting started

Let's Get Started!

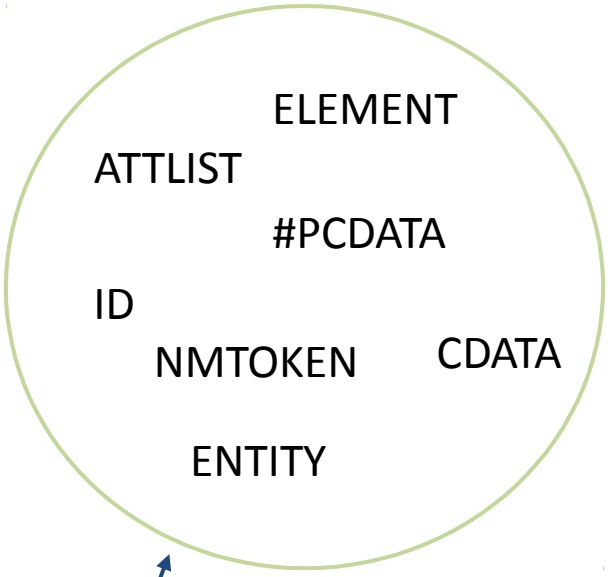
- Convert the `BookStore.dtd` to the XML Schema syntax

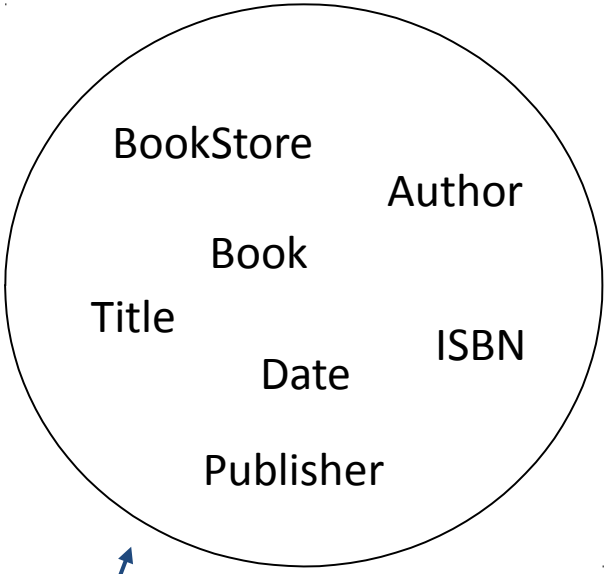
```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

  - Straight, one-to-one conversion, i.e.,
    Title, Author, Date, ISBN, and Publisher will hold strings
  - We will gradually modify the XML Schema to use stronger types

Bookstore Schema

- Namespaces again

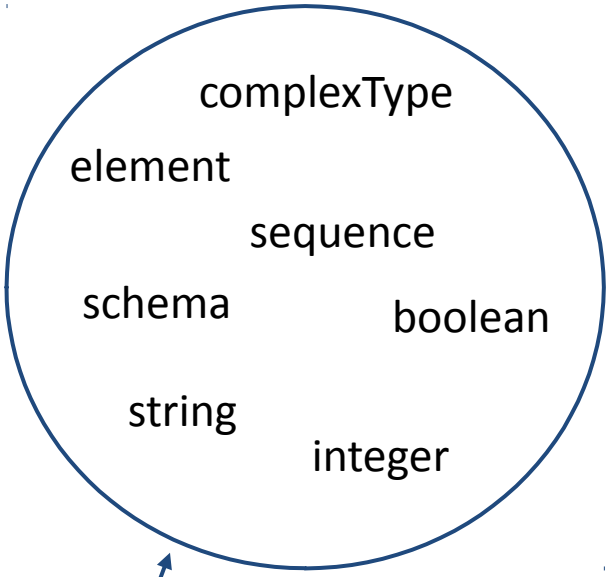




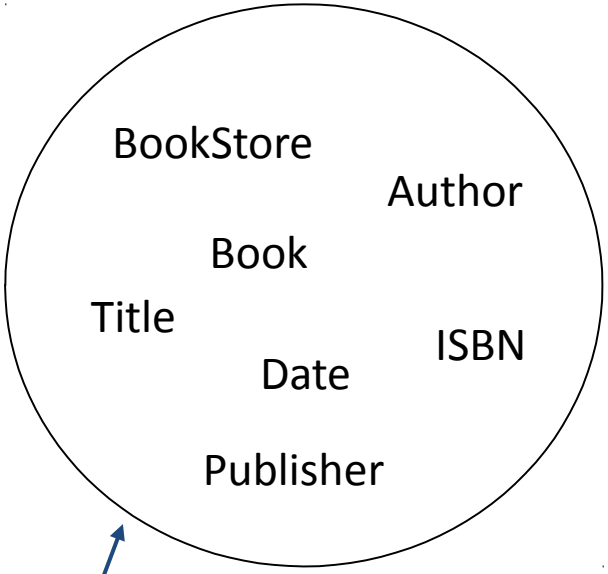
This is the vocabulary that DTDs provide to define your new vocabulary

Bookstore Schema

http://www.w3.org/2001/XMLSchema



http://www.books.org (*targetNamespace*)



This is the vocabulary that
XML Schemas provide to define your
new vocabulary

Bookstore Schema

- Notice:
  - XML Schema vocabulary is associated with a namespace.
  - Likewise, the new vocabulary that you define must be associated with a namespace.
  - With DTDs neither set of vocabulary is associated with a namespace [because DTDs pre-dated namespaces].

BookStore.xsd

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org"
        xmlns="http://www.books.org"
        elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org"
        xmlns="http://www.books.org"
        elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

<!ELEMENT BookStore (Book+)>

<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>

<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            argetNamespace="http://www.books.org"
            mlns="http://www.books.org"
            lementFormDefault="qualified">
            lement name="BookStore">
            d:complexType>
            xsd:sequence>
              <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Book">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
              <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
              <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
              <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
              <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:string"/>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

All XML Schemas have "schema" as the document element.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.book
        xmlns="http://www.books.org"
        elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Book" minOccurs="1" maxOccu
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

The elements and datatypes that are used to construct a schema, e.g.

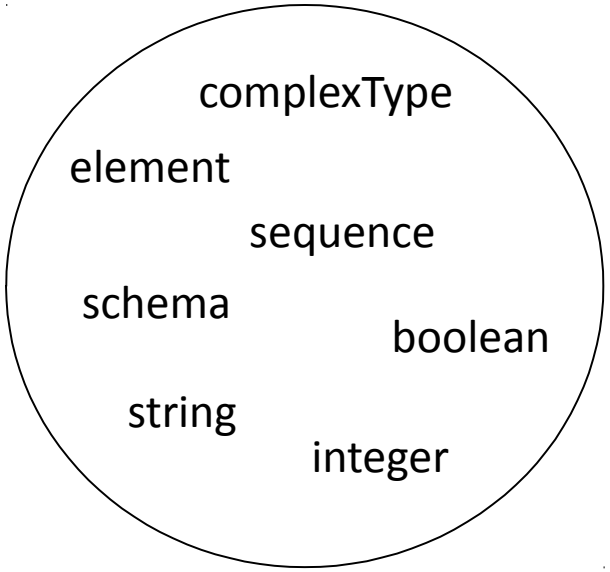
- schema
- element
- complexType
- sequence
- string

come from the http://…/XMLSchema namespace.

Bookstore Schema

- XMLSchema Namespace

http://www.w3.org/2001/XMLSchema



```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema
        targetNamespace="http://www.books.org"
        xmlns="http://www.book
        elementFormDefault="qu d">
   <xsd:element name="BookStor
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Bo minOccurs="1" maxOccu
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

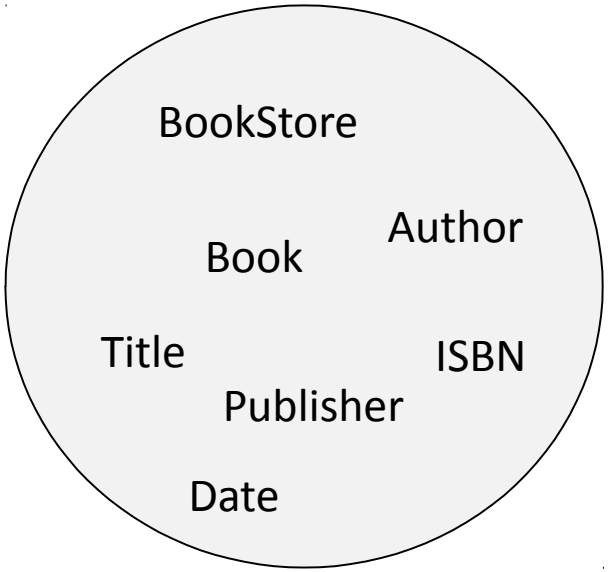
Indicates that the elements defined by this schema
- BookStore
- Book
- Title
- Author
- Date
- ISBN
- Publisher

are to go in the http://www.books.org namespace

Bookstore Schema

- Book Namespace (targetNamespace)

http://www.books.org (*targetNamespace*)



```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org"
        xmlns="http://www.books.org"
        elementFormDe[illegible]="qualified">
   <xsd:element name=[illegible]Store">
      <xsd:complexTyp[illegible]
         <xsd:sequence[illegible]
            <xsd:eleme[illegible]="Book" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequenc[illegible]
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOc[illegible]
            <xsd:element ref="Author" minOccurs="1" maxO[illegible]
            <xsd:element ref="Date" minOccurs="1" maxOcc[illegible]
            <xsd:element ref="ISBN" minOccurs="1" maxOcc[illegible]
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

The default namespace is http://www.books.org which is the targetNamespace!

Reference to Book in what namespace?

Since there is no namespace qualifier it is referencing the Book element in the default namespace, which is the targetNamespace! Thus, this is a reference to the Book element declaration in this schema.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchem
        targetNamespace="http://www.books.org"
        xmlns="http://www.books.org"
        elementFormDefault="qualified">
   <xsd:element name="BookSto
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="                minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Directive to any instance documents which conform to this schema:
Any elements used by the instance document which were declared in this schema must be namespace qualified.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.or
        targetNamespace="http://www.boo
        xmlns="http://www.books.org"
        elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Occurrence constraints
The default value for minOccurs is "1"
The default value for maxOccurs is "1"
e.g.
<xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
equivalent to
<xsd:element ref="Title"/>

Bookstore Schema

- Referencing a schema in an XML i

```
<?xml version="1.0"?>
<BookStore xmlns="http://www.books.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.book
   <Book>
      <Title>
      <Author
      <Date>J
      <ISBN>9
      <Publisher>McMillin Publishing</Publisher>
   </Book>
   …
</BookStore>
```

Tell the schema-validator that all of the elements used in this instance document come from the http://www.books.org namespace.

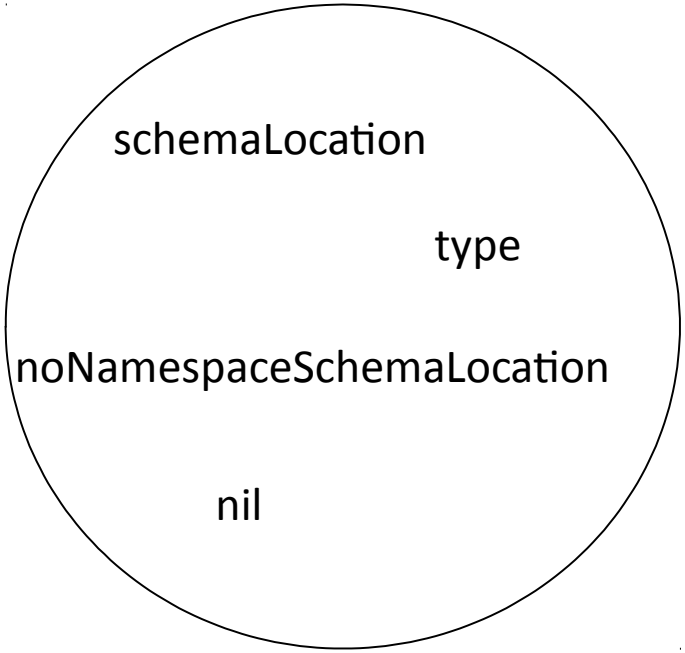
Tell the schema-validator that the schemaLocation attribute is in the XMLSchema-instance namespace.

Tell the schema-validator that the http://www.books.org namespace is defined by BookStore.xsd; schemaLocation attribute has a pair of values.

Bookstore Schema

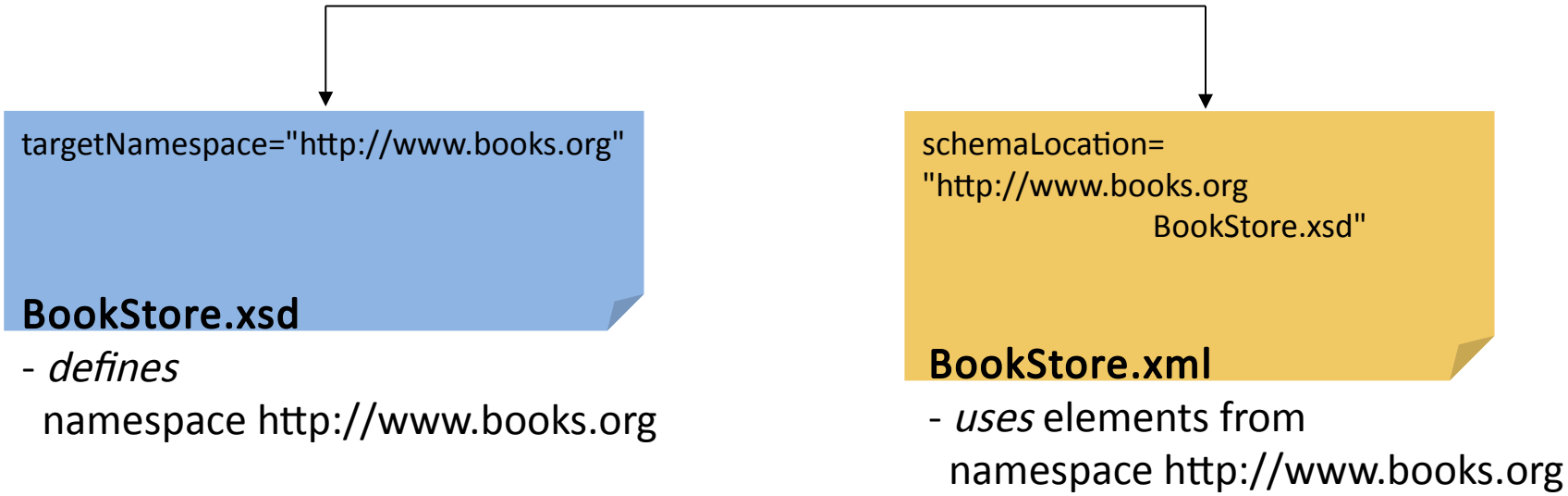
- XMLSchema-instance Namespace

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



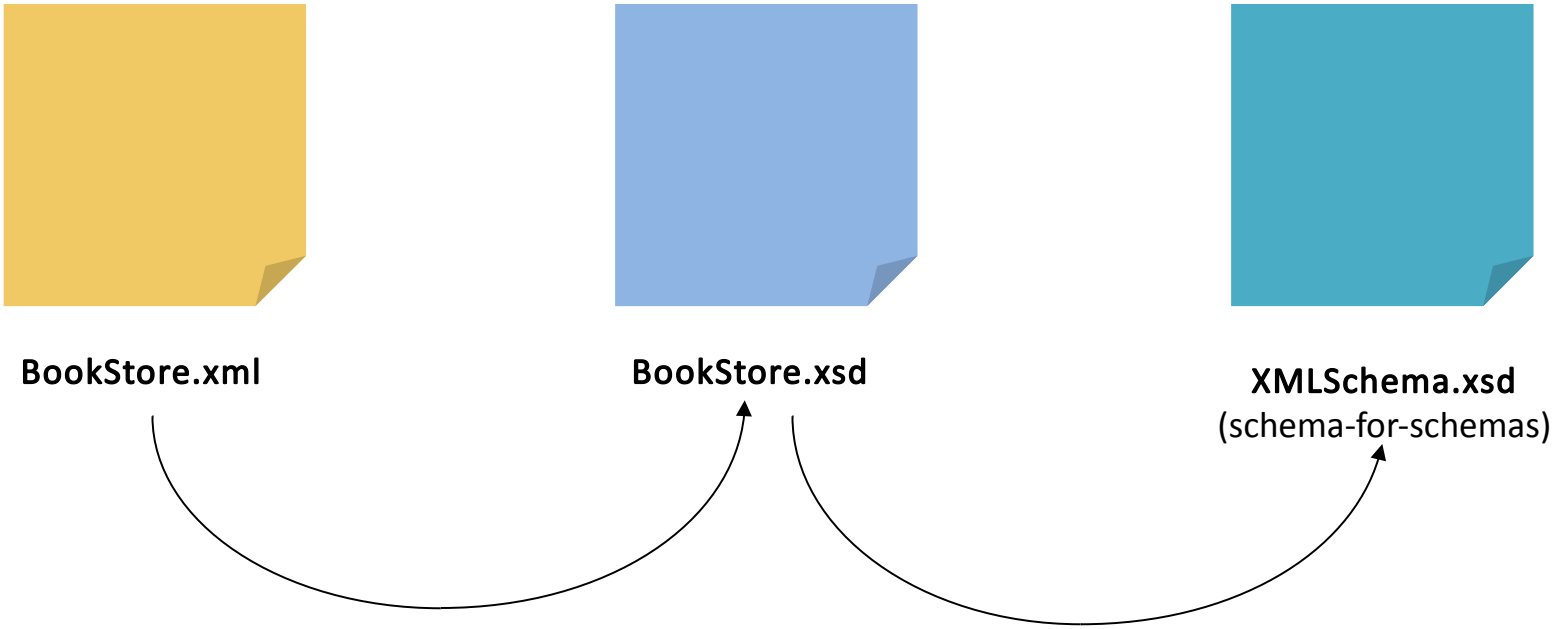
Bookstore Schema

- XML schema and instance document



  - A schema defines a new vocabulary.
  - Instance documents use that new vocabulary.

Multiple Levels of Checking



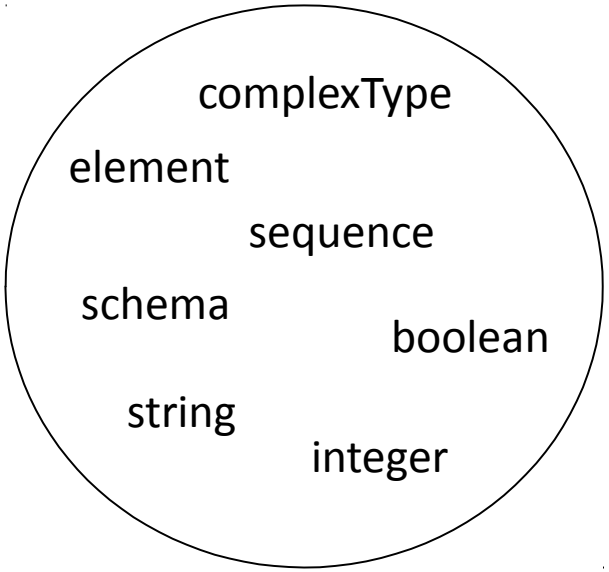
Validate that the xml document conforms to the rules described in BookStore.xsd

Validate that BookStore.xsd is a valid schema document, i.e., it conforms to the rules described in the schema-for-schemas

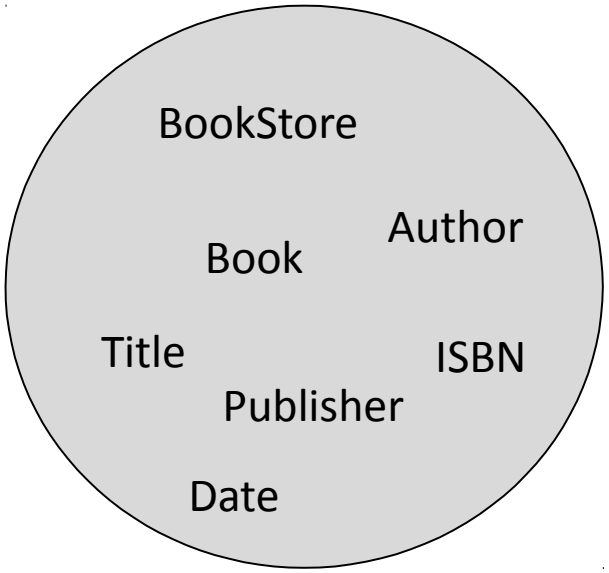
Namespaces Again

- targetNamespace may be the default namespace.

http://www.w3.org/2001/XMLSchema



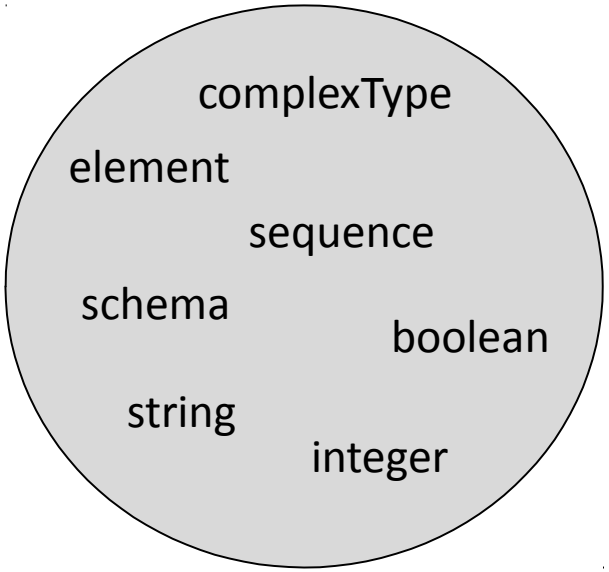
http://www.books.org (*targetNamespace*)



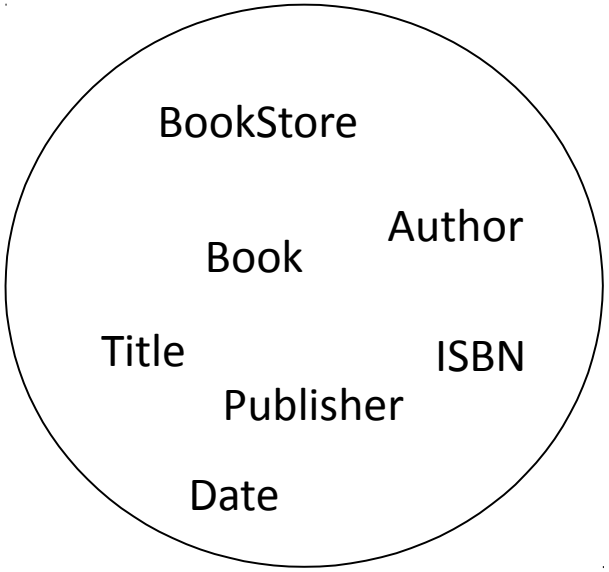
Namespaces Again

- Alternatively (equivalently), we can design our schema so that XMLSchema is the default namespace.

http://www.w3.org/2001/XMLSchema



http://www.books.org (*targetNamespace*)



```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://w      ks.org"
               xmlns:bk="http://www.book      '
               elementFormDefault="quali      >
   <element name="BookStore">
      <complexType>
         <sequence>
            <element ref="bk:Book" minOccurs="1" maxOccurs="ur
         </sequence>
      </complexType>
   </element>
   <element name="Book">
      <complexType>
         <sequence>
            <element ref="bk:Title"/>
            <element ref="bk:Author"/>
            <element ref="bk:Date"/>
            <element ref="bk:ISBN"/>
            <element ref="bk: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>
```

Note that http://
.../XMLSchema
is the default namespace.
Consequently, there are no
prefixes on
- schema
- element
- complexType
- sequence
- string

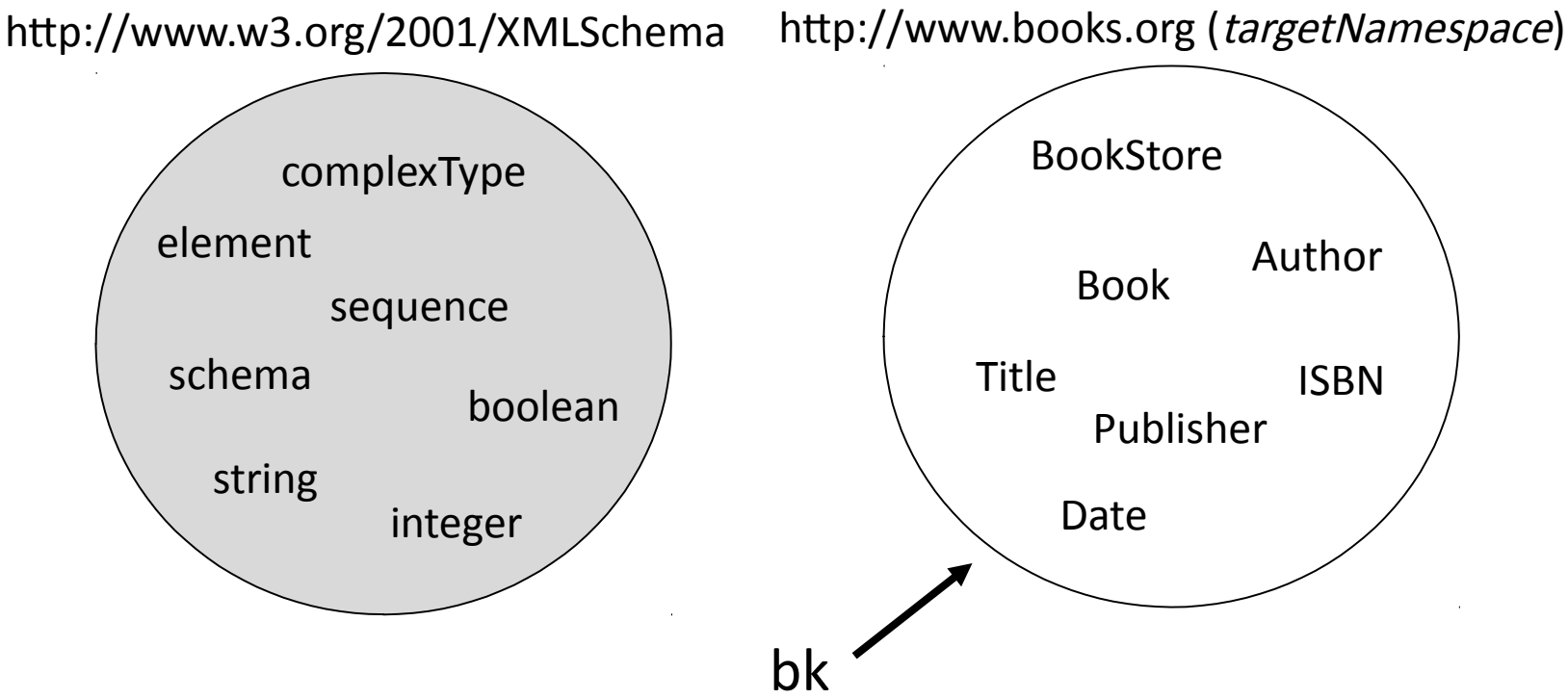
```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://www.books.org"
               xmlns:bk="http://www.books.org"
               elementFormDefault="q      ed">
   <element name="BookStore">
      <complexType>
         <sequence>
            <element ref="bk:Book" min      ="1" maxOccurs="unbounded"/>
         </sequence>
      </complexType>
   </element>
   <element name="Book">
      <complexType>
         <sequence>
            <element ref="bk:Title"/>
            <element ref="bk:Author"/>
            <element ref="bk:Date"/>
            <element ref="bk:ISBN"/>
            <element ref="bk: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>
```

Here we are referencing a Book element. In what namespace is that Book element defined??

The bk: prefix indicates what namespace this element is in. bk: has been set to be the same as the targetNamespace.

Namespaces Again

- "bk:" references the targetNamespace
  → bk:Book refers to the Book element in the targetNamespace.



Chapter 4.3

Three XML Schema Flavours

Three XML Schema Flavours

- Three different "flavours" for writing an XML schema
  - **embedded types**:
    types are defined where they are used in the document hierarchy
  - **named types**:
    each element has a name and a named type,
    and each named type is defined separately
  - **flat catalogue**:
    each element is defined by reference to another element declaration
- Schemas may not be equivalent!

Embedded types

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        targetNamespace="http://www.books.org"
                        xmlns="http://www.books.org"
                        elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Book" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Title" type="xsd:string"/>
                     <xsd:element name="Author" type="xsd:string"/>
                     <xsd:element name="Date" type="xsd:string"/>
                     <xsd:element name="ISBN" type="xsd:string"/>
                     <xsd:element name="Publisher" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Types are anonymous.

Named types

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        targetNamespace="http://www.books.org"
                        xmlns="http://www.books.org"
                        elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Book" type="BookPublication"
                        maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="BookPublication">
      <xsd:sequence>
         <xsd:element name="Title" type="xsd:string"/>
         <xsd:element name="Author" type="xsd:string"/>
         <xsd:element name="Date" type="xsd:string"/>
         <xsd:element name="ISBN" type="xsd:string"/>
         <xsd:element name="Publisher" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

The advantage of splitting out Book's element declarations and wrapping them in a named type is that now this type can be *reused* by other elements.

Three XML Schema Flavours

- Please note that

This

```
<xsd:element name="A" type="foo"/>
<xsd:complexType name="foo">
   <xsd:sequence>
      <xsd:element name="B" …/>
      <xsd:element name="C" …/>
   </xsd:sequence>
</xsd:complexType>
```

is equivalent to:

```
<xsd:element name="A">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="B" …/>
         <xsd:element name="C" …/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Element A *references* the complexType foo.

Element A has the complexType definition *inlined* in the element declaration.

Three XML Schema Flavours

- An element definition has either
  - a `type` attribute or
  - a `complexType` child element, but not both!

```
<xsd:element name="A" type="foo">
    <xsd:complexType>
            …
    </xsd:complexType>
</xsd:element>
```

Flat catalogue

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.books.org"
        xmlns="http://www.books.org"
        elementFormDefault="qualified">
   <xsd:element name="BookStore">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Book">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Author" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="ISBN" type="xsd:string"/>
   <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Embedded types?

```
<schema>
    <element name="weather">
        <complexType>
            <sequence>
                <element name="location">
                    <complexType>
                        <sequence>
                            <element name="city"> <simpleType>
                                <restriction base="string">
                                    <pattern value="[a-zA-Z]"/>
                                </restriction>
                            </simpleType> </element>
                            <element name="country" type="string"/>
                        </sequence>
                    </complexType>
                </element>
                <element name="temperature" type="integer"/>
                <element name="barometric_pressure" type="integer"/>
                <element name="conditions" type="string"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

Embedded types?

```
<schema>
    <complexType name="weatherType">
        <sequence>
            <element name="location" type="locationType"/>
            <element name="temperature" type="integer"/>
            <element name="barometric_pressure" type="integer"/>
            <element name="conditions" type="string"/>
        </sequence>
    </complexType>
    <complexType name="locationType" >
        <sequence>
            <element name="city" type="cityType"/>
            <element name="country" type="string"/>
        </sequence>
    </complexType>
    <simpleType name="cityType">
        <restriction base="string">
            <pattern value="[a-zA-Z]"/>
        </restriction>
    </simpleType>
    <element name="weather" type="weatherType"/>
</schema>
```

Embedded types?

```
<schema>
    <element name="temperature" type="integer"/>
    <element name="barometric_pressure" type="integer"/>
    <element name="conditions" type="string"/>
    <element name="country" type="string"/>
    <element name="city">
        <simpleType>
            <restriction base="string">
                <pattern value="[a-zA-Z]"/>
            </restriction>
        </simpleType>
    </element>
    <element name="location">
        <complexType> <sequence>
            <element ref="city"/>
            <element ref="country"/> </sequence>
        </complexType>
    </element>
    <element name="weather">
        <complexType> <sequence>
                <element ref="location"/>
                <element ref="temperature"/>
                <element ref="barometric_pressure"/>
                <element ref="conditions"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

Summary of Declaring Elements

1

<xsd:element name="*name*" type="*type*" minOccurs="*int*" maxOccurs="*int*"/>

A simple type (e.g., xsd:string) or the name of a complexType (e.g., BookPublication)

A non-negative integer

A non-negative integer or "unbounded"

Note: *minOccurs* and *maxOccurs* can only be used in nested (local) element declarations.

2

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:complexType>
        …
    </xsd:complexType>
</xsd:element>
```

3

flat catalogue

Chapter 4.4

Simple Types

Problem

- Declaring the `Date` element of `BookStore.xsd`
  - type `string` is unsatisfactory
    - it allows any string value to be input as the content of the `Date` element, including non-date strings
  - constrain the allowable content that the `Date` element can have, e.g. restrict the content of `Date` to just year values.
- Similar for the `ISBN` element

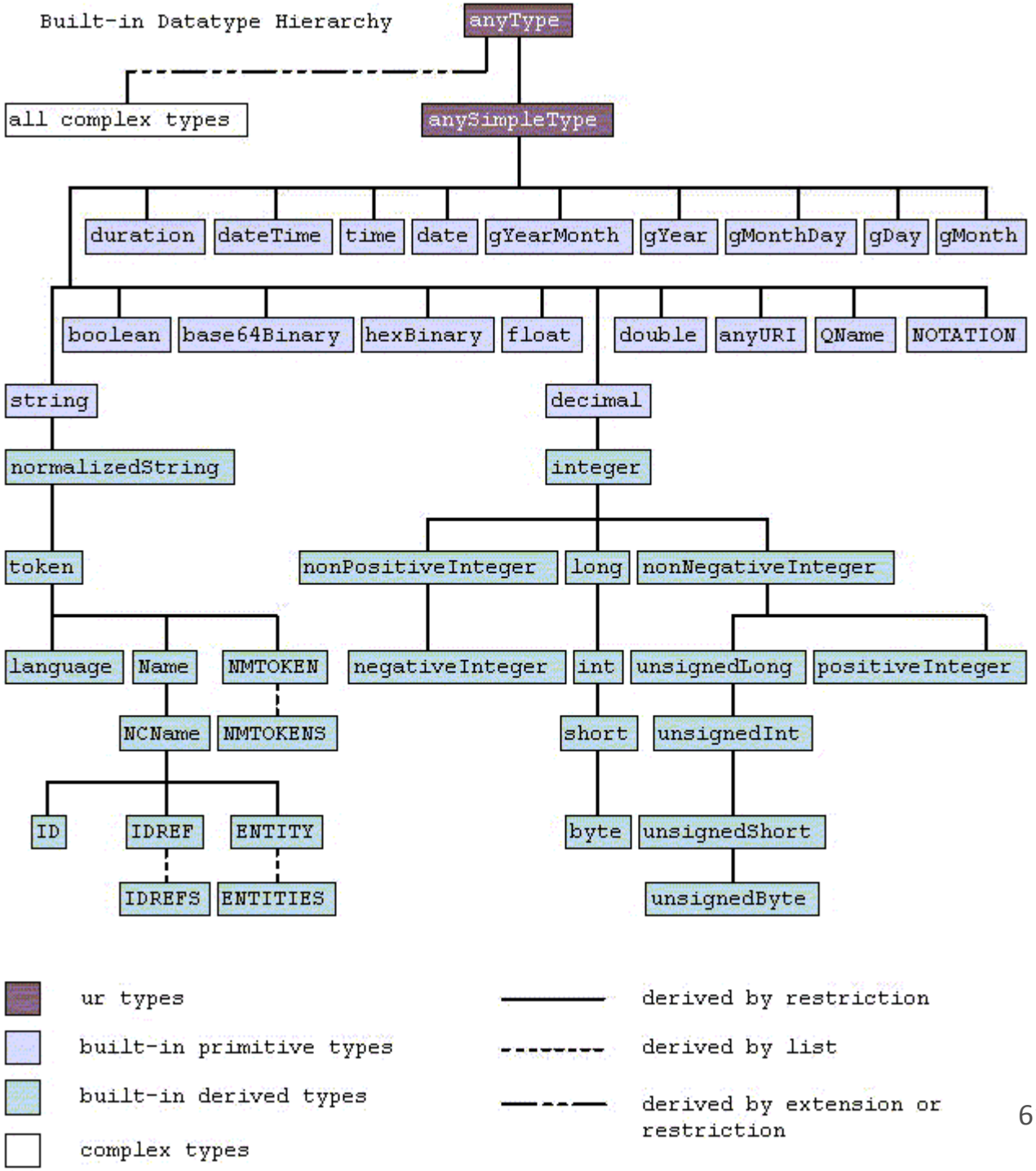
W3C XML Schema

Hierarchy of built-in types

from:

XML Schema Part 2: Datatypes Second Ed.

W3C Recommendation 28 October 2004



Datatypes: Definitions

- "A datatype is a 3-tuple, consisting
  - a set of distinct values, called its value space,
  - a set of lexical representations, called its lexical space, and
  - a set of facets that characterize properties of the value space, individual values or lexical items."

Datatypes: Definitions

- Value space
  - A value space is the set of values for a given datatype. Each value in the value space of a datatype is denoted by one or more literals in its lexical space.
  - The value space of a given datatype can be defined:
    - axiomatically from fundamental notions (intensional definition)
    - by enumeration (extensional definition)
    - by restricting the value space of an already defined datatype to a particular subset with a given set of properties
    - by combining the values from one or more already defined value space(s) by a specific construction procedure [list and union]

Datatypes: Definitions

- Lexical space
  - A lexical space is the set of valid literals for a datatype.

    For example, "100" and "1.0E2" are two different literals from the lexical space of float which both denote the same value. The type system defined in this specification provides a mechanism for schema designers to control the set of values and the corresponding set of acceptable literals of those values for a datatype.

Datatypes: Definitions

- Primitive datatypes
  - Primitive datatypes are those that are not defined in terms of other datatypes; they exist *ab initio*.
- Derived datatypes
  - Derived datatypes are those that are defined in terms of other datatypes.

For example, `float` is a well-defined mathematical concept that cannot be defined in terms of other datatypes, while a `integer` is a special case of the more general datatype `decimal`.

Datatypes: Definitions

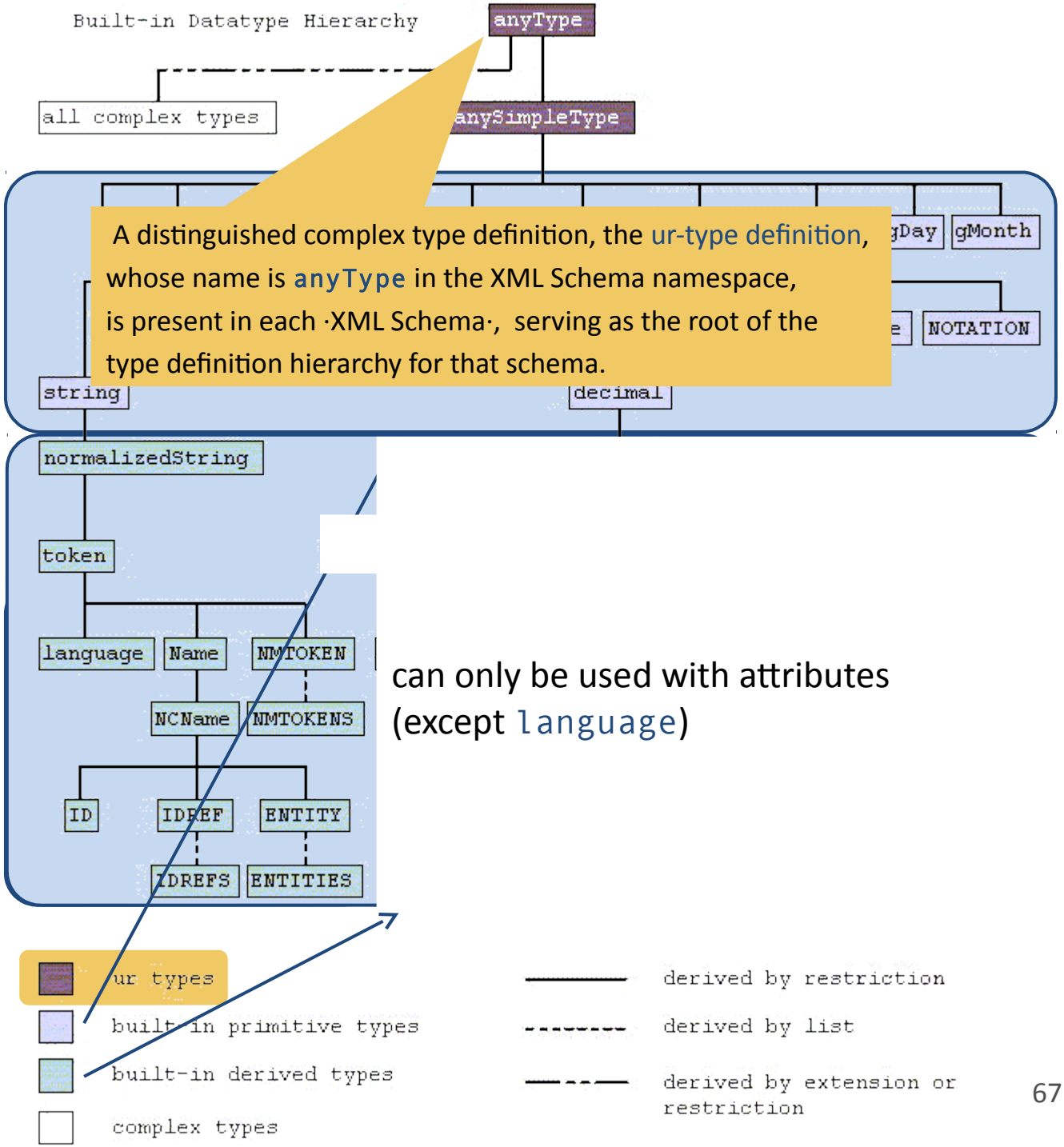
- Facets
  - A facet is a single defining aspect of a value space. Generally speaking, each facet characterizes a value space along independent axes or dimensions.
  - Facets are of two types:
    - fundamental facets
      that define the datatype and
    - non-fundamental or constraining facets
      that constrain the permitted values of a datatype.

W3C XML Schema

Hierarchy of built-in types

from:

XML Schema Part 2: Datatypes Second Ed.
W3C Recommendation 28 October 2004



A distinguished complex type definition, the ur-type definition, whose name is `anyType` in the XML Schema namespace, is present in each ·XML Schema·, serving as the root of the type definition hierarchy for that schema.

can only be used with attributes (except `language`)

Built-in Datatypes

- Primitive datatypes: atomic, built-in
  - string → "Hello World"
  - boolean → {true, false, 1, 0}
  - decimal → 7.08
  - float → 12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
  - double → 12.56E3, 12, 12560, 0, -0, INF, -INF, NAN
  - duration → P1Y2M3DT10H30M12.3S
  - dateTime → format: *CCYY-MM-DDThh:mm:ss*
  - time → format: *hh:mm:ss.sss*
  - date → format: *CCYY-MM-DD*
  - gYearMonth → format: *CCYY-MM*
  - gYear → format: *CCYY*
  - gMonthDay → format: --*MM-DD*

Note: 'T' is the date/time separator
INF = infinity
NAN = Not-A-Number

Built-in Datatypes (cont'd.)

- Primitive datatypes: atomic, built-in
  - gDay → format: ---*DD* (note the 3 dashes)
  - gMonth → format: --*MM*
  - hexBinary → a hex string
  - base64Binary → a base64 string
  - anyURI → http://www.xfront.com
  - QName → a namespace qualified name
  - NOTATION → a NOTATION from the XML spec

Built-in Datatypes (cont'd.)

- Derived from built-in types
  - normalizedString → A string without tabs, line feeds, or carriage returns
  - token → String w/o tabs, l/f, leading/trailing/consecutive spaces
  - language → any valid xml:lang value, e.g., **EN**, **FR**, ...
  - ID → must be used only with attributes
  - IDREF → must be used only with attributes
  - IDREFS → must be used only with attributes
  - ENTITY → must be used only with attributes
  - ENTITIES → must be used only with attributes
  - NMTOKEN → must be used only with attributes
  - NMTOKENS → must be used only with attributes
  - Name →
  - NCName → **part** (no namespace qualifier)
  - integer → **456**
  - nonPositiveInteger → negative infinity to 0

Built-in Datatypes (cont'd.)

- Derived from built-in types
  - negativeInteger → negative infinity to -1
  - long → -9223372036854775808 to 9223372036854775807
  - int → -2147483648 to 2147483647
  - short → -32768 to 32767
  - byte → -127 to 128
  - nonNegativeInteger → 0 to infinity
  - unsignedLong → 0 to 18446744073709551615
  - unsignedInt → 0 to 4294967295
  - unsignedShort → 0 to 65535
  - unsignedByte → 0 to 255
  - positiveInteger → 1 to infinity

The `date` Datatype

- The `date` datatype
  - a *built-in* datatype
  - used to represent a specific day in notation: CCYY-MM-DD
    - range for CC is: 00-99
    - range for YY is: 00-99
    - range for MM is: 01-12
    - range for DD is:
      - 01-28 if month is 2
      - 01-29 if month is 2 and the gYear is a leap gYear
      - 01-30 if month is 4, 6, 9, or 11
      - 01-31 if month is 1, 3, 5, 7, 8, 10, or 12

Pope Gregory XIII, 1552 – 1614

The `gYear` Datatype

- The `gYear` datatype
  - a *built-in* datatype
  - Elements declared to be of type gYear must follow this form: CCYY
    - range for CC is: 00-99
    - range for YY is: 00-99

Datatypes for Attributes

- Special datatypes for attributes only
  - ID
    - unique value within the entire document
    - at most one ID attribute per element
    - no default value
  - IDREF
    - its value must be some other element's ID value in the document
  - IDREFS
    - its value is a space-separated set, each element of the set is the ID value of some other element in the document

Datatypes for Attributes

- Special datatypes for attributes only (cont'd.)
  - Example

```
<person id="o555">
   <name> Jane </name>
</person>
<person id="o456">
   <name> Mary </name>
   <children ref="o123 o555"/>
</person>
<person id="o123" mother="o456">
   <name>John</name>
</person>
```

User-defined `simpleTypes`

- Creating your own `simpleTypes`
  - A new datatype can be derived from an existing datatype, called the "base" type
  - Specify values for one or more of the *facets* for the base type.
  - Example: The string primitive datatype has six optional facets:
    - length
    - minLength
    - maxLength
    - pattern
    - enumeration
    - whitespace (legal values: preserve, replace, collapse)

The `ISBN` Datatype

- No built-in ISBNType, so ISBNType to be defined in schema

  1. restriction of the `string` type,
     using *pattern* facet:
     - first pattern: d-ddddd-ddd-d
     - second pattern: d-ddd-ddddd-d
     - third pattern: d-dd-dddddd-d

     where 'd' stands for 'digit'
  2. use the `simpleType` Schema element to create a new type
     that is a refinement of a built-in type

The ISBN Datatype


```
<xsd:simpleType name="ISBNType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="http://www.books.org"
                      xmlns="http://www.books.org"
                      elementFormDefault="qualified">
  <xsd:simpleType name="ISBNType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book"  maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:gYear"/>
              <xsd:element name="ISBN" type="ISBNType"/>
              <xsd:element name="Publisher" type="xsd:string"/
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Here we are defining a new data-type, called ISBNType.

Declaring Date to be of type gYear, and ISBN to be of type ISBNType (defined above)

The ISBN Datatype

- Equivalent Expressions

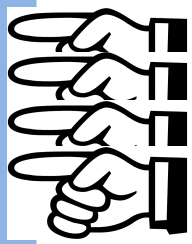
```
<xsd:simpleType name="ISBNType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
      <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="ISBNType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern  value="\d{1}-\d{5}-\d{3}-\d{1}|
                           \d{1}-\d{3}-\d{5}-\d{1}|
                           \d{1}-\d{2}-\d{6}-\d{1}"/>
   </xsd:restriction>
</xsd:simpleType>
```

User-defined `simpleTypes`

- Specifying facet values

```
<xsd:simpleType name="TelephoneNumber">
   <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
      <xsd:pattern value="\d{3}-\d{4}"/>
   </xsd:restriction>
</xsd:simpleType>
```



This creates a new

Elements of this type can hold string values

follow the pattern: ddd-dddd, where 'd' represents a 'digit'

  – Obviously, in this example the regular expression makes the length facet redundant.

Pattern Facet

- Regular Expressions
  - Pattern facets defined by regular expressions
  - examples:

Regular expressions	Facet instances
Chapter \d	Chapter 1
Chapter \d	Chapter 1
a*b	b, ab, aab, aaab, …
[xyz]b	xb, yb, zb
a?b	b, ab
a+b	ab, aab, aaab, …
[a-c]x	ax, bx, cx

Pattern Facet

- Regular Expressions (cont'd.)

Regular Expression	Explained
[a-c]x	x, bx, cx
[\-ac]x	-x, ax, cx (Backslash causes metacharacter "-" to be treated as literal char.)
[ac\-]x	ax, cx, -x
[^0-9]x	any non-digit char followed by x
\Dx	any non-digit char followed by x
Chapter\s\d	"Chapter" followed by a blank followed by a digit
(ho){2} there	hoho there
(ho\s){2} there	ho ho there
.abc	any (*one*) char followed by abc
(a|b)+x	ax, bx, aax, bbx, abx, bax,...

Pattern Facet

- Regular Expressions (cont'd.)

a{1,3}x	ax, aax, aaax
a{2,}x	aax, aaax, aaaax, …
\w\s\w	word **character** (alphanumeric plus dash) followed by a space followed by a word character
[a-zA-Z-[Ol]]*	A string comprised of any lower and upper case letters, except "O" and "l“
\.	The period "."

Pattern Facet

- Regular Expressions (concluded)
  - with definitions from Unicode

\p{L}	A letter, from any language
\p{Lu}	An uppercase letter, from any language
\p{Ll}	A lowercase letter, from any language
\p{N}	A number - Roman, fractions, etc
\p{Nd}	A digit from any language
\p{P}	A punctuation symbol
\p{Sc}	A currency sign, from any language

Pattern Facet

- Sample regular expression

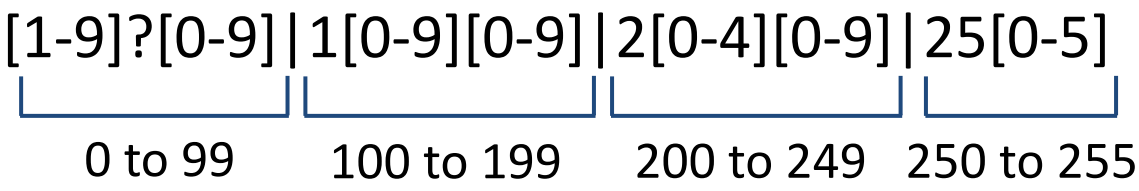
```
<xsd:simpleType name="money">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="cost" type="money"/>
```

```
<cost>$45.99</cost>
<cost>¥300</cost>
```

"currency sign from any language, followed by one or more digits from any language, optionally followed by a period and two digits from any language"

Pattern Facet

- Sample regular expression



  - This regular expression restricts a *string* to have values between 0 and 255.
  - Such a R.E. might be useful in describing an IPv4 address ...

Pattern Facet

- IP Datatype Definition

```
<xsd:simpleType name="IP">
  <xsd:restriction base="xsd:string">
    <xsd: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])">
      <xsd:annotation>
        <xsd:documentation>
          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.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:pattern>
  </xsd:restriction>
</xsd:simpleType>
```

User-defined `simpleTypes`

- Enumeration facet

```
<xsd:simpleType name="shape">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="circle"/>
    <xsd:enumeration value="triangle"/>
    <xsd:enumeration value="square"/>
  </xsd:restriction>
</xsd:simpleType>
```

  - This creates a new type called shape.
  - An element declared to be of this type must have either the value circle, or triangle, or square.

User-defined `simpleTypes`

- Facets of the integer datatype
  - The integer datatype has 8 optional facets:
    - totalDigits
    - pattern
    - whitespace
    - enumeration
    - maxInclusive
    - maxExclusive
    - minInclusive
    - minExclusive

User-defined `simpleTypes`

- min/max facets

```
<xsd:simpleType name="EarthSurfaceElevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-1290"/>
    <xsd:maxInclusive value="29035"/>
  </xsd:restriction>
</xsd:simpleType>
```

  - This creates a new datatype called 'EarthSurfaceElevation'.
  - Elements declared to be of this type can hold an integer.
  - However, the integer is restricted to have a value between -1290 and 29035, inclusive.

User-defined `simpleTypes`

- Specifying facet values

```
<xsd:simpleType name="name">
  <xsd:restriction base="xsd:source">
    <xsd:facet value="value"/>
    <xsd:facet value="value"/>
    …
  </xsd:restriction>
</xsd:simpleType>
```

Sources:
- string
- boolean
- number
- float
- double
- duration
- dateTime
- time
...

Facets:
- length
- minlength
- maxlength
- pattern
- enumeration
- minInclusive
- maxInclusive
- minExclusive
- maxExclusive
...

User-defined `simpleTypes`

- Multiple facets
  - "*and*" them together, or
  - "*or*" them together?

```
<xsd:simpleType name="TelephoneNumber">
   <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
      <xsd:pattern value="\d{3}-\d{4}"/>
   </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type TelephoneNumber must be a string of length=8 *and* the string must follow the pattern: 3 digits, dash, 4 digits.

User-defined `simpleTypes`

- Multiple facets

```
<xsd:simpleType name="TelephoneNumber">
   <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
      <xsd:pattern value="\d{3}-\d{4}"/>
   </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type TelephoneNumber must be a string of length=8 *and* the string must follow the pattern: 3 digits, dash, 4 digits.

```
<xsd:simpleType name="shape">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="circle"/>
      <xsd:enumeration value="triangle"/>
      <xsd:enumeration value="square"/>
   </xsd:restriction>
</xsd:simpleType>
```

An element declared to be of type shape must be a string with a value of *either* circle, *or* triangle, *or* square.

  - Patterns, enumerations: "or" them together
  - All other facets: "and" them together

User-defined `simpleTypes`

- Creating a simpleType from another simpleType
  - Thus far we have created a simpleType using one of the built-in datatypes as our base type.
  - However, we can create a simpleType that uses another simpleType as the base.

User-defined `simpleTypes`

- This simpleType uses EarthSurfaceElevation as its base type.

```
<xsd:simpleType name="EarthSurfaceElevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="-1290"/>
    <xsd:maxInclusive value="29035"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="BostonAreaSurfaceElevation">
  <xsd:restriction base="EarthSurfaceElevation">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="120"/>
  </xsd:restriction>
</xsd:simpleType>
```

User-defined `simpleTypes`

- Fixing a facet value
  - Sometimes it might be required that one (or more) facets have an unchanging value: facet is a constant.

```
<xsd:simpleType name="ClassSize">
  <xsd:restriction base="xsd:nonNegativeInteger">
    <xsd:minInclusive value="10" fixed="true"/>
    <xsd:maxInclusive value="60"/>
  </xsd:restriction>
</xsd:simpleType>
```

simpleTypes which derive from this simpleType may not change this facet.

User-defined `simpleTypes`


```
<xsd:simpleType name="ClassSize">
   <xsd:restriction base="xsd:nonNegativeInteger">
      <xsd:minInclusive value="10" fixed="true"/>
      <xsd:maxInclusive value="60"/>
   </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="BostonIEEEClassSize">
   <xsd:restriction base="ClassSize">
      <xsd:minInclusive value="15"/>
      <xsd:maxInclusive value="60"/>
   </xsd:restriction>
</xsd:simpleType>
```

Error! Cannot change the value of a fixed facet!

User-defined `simpleTypes`

- Create an element declaration for an elevation element.
  - Declare the elevation element to be an integer with a range -1290 to 29035
  - Sample instance document:

```
<elevation>5240</elevation>
```

User-defined `simpleTypes`

- Element Containing a User-Defined Simple Type
  - Here's one way of declaring the elevation element:

```
<xsd:simpleType name="EarthSurfaceElevation">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="-1290"/>
      <xsd:maxInclusive value="29035"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:element name="elevation" type="EarthSurfaceElevation"/>
```

User-defined `simpleTypes`

- Element Containing a User-Defined Simple Type
  - Here's an alternative method for declaring elevation:

```
<xsd:element name="elevation">
   <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
         <xsd:minInclusive value="-1290"/>
         <xsd:maxInclusive value="29035"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

The simpleType definition is defined inline, it is an *anonymous* simpleType definition.
The disadvantage of this approach is that this simpleType may not be reused by other elements.

Chapter 4.5

List and Union Datatypes

Datatypes: Definitions

- Atomic datatypes
  - Atomic datatypes are those having values which are regarded by this specification as being indivisible.
- List datatypes
  - List datatypes are those having values each of which consists of a finite-length (possibly empty) sequence of values of an atomic datatype.
- Union datatypes
  - Union datatypes are those whose value spaces and lexical spaces are the union of the value spaces and lexical spaces of one or more other datatypes.

List Datatypes

- Creating lists
  - There are times when you will want an element to contain a list of values, e.g., "The contents of the `Numbers` element is a list of numbers".

Example: For a document containing a Lottery drawing we might have

<Numbers>12 49 37 99 20 67</Numbers>

How do we declare the element Numbers ...

(1) To contain a list of integers, and
(2) Each integer is restricted to be between 1 and 99, and
(3) The total number of integers in the list is exactly six.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.lottery.org"
          xmlns="http://www.lottery.org" elementFormDefault="qualified">
   <xsd:simpleType name="LotteryNumbers">
      <xsd:list itemType="xsd:positiveInteger"/>
   </xsd:simpleType>
   <xsd:element name="LotteryDrawings">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Drawing" minOccurs="0" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Week" type="xsd:string"/>
                     <xsd:element name="Numbers" type="LotteryNumbers"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

```
<?xml version="1.0"?>
<LotteryDrawings xmlns="http://www.lottery.org"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.lottery.org Lottery.xsd">
   <Drawing>
     <Week>July 1</Week>
     <Numbers>21 3 67 8 90 12</Numbers>
   </Drawing>
   <Drawing>
     <Week>July 8</Week>
     <Numbers>55 31 4 57 98 22</Numbers>
   </Drawing>
   <Drawing>
     <Week>July 15</Week>
     <Numbers>70 77 19 35 44 11</Numbers>
   </Drawing>
</LotteryDrawings>
```

List Datatypes

- List datatypes
  - stronger typing:
    - Restrict the list to length value="6"
    - Restrict the numbers to maxInclusive value="49"

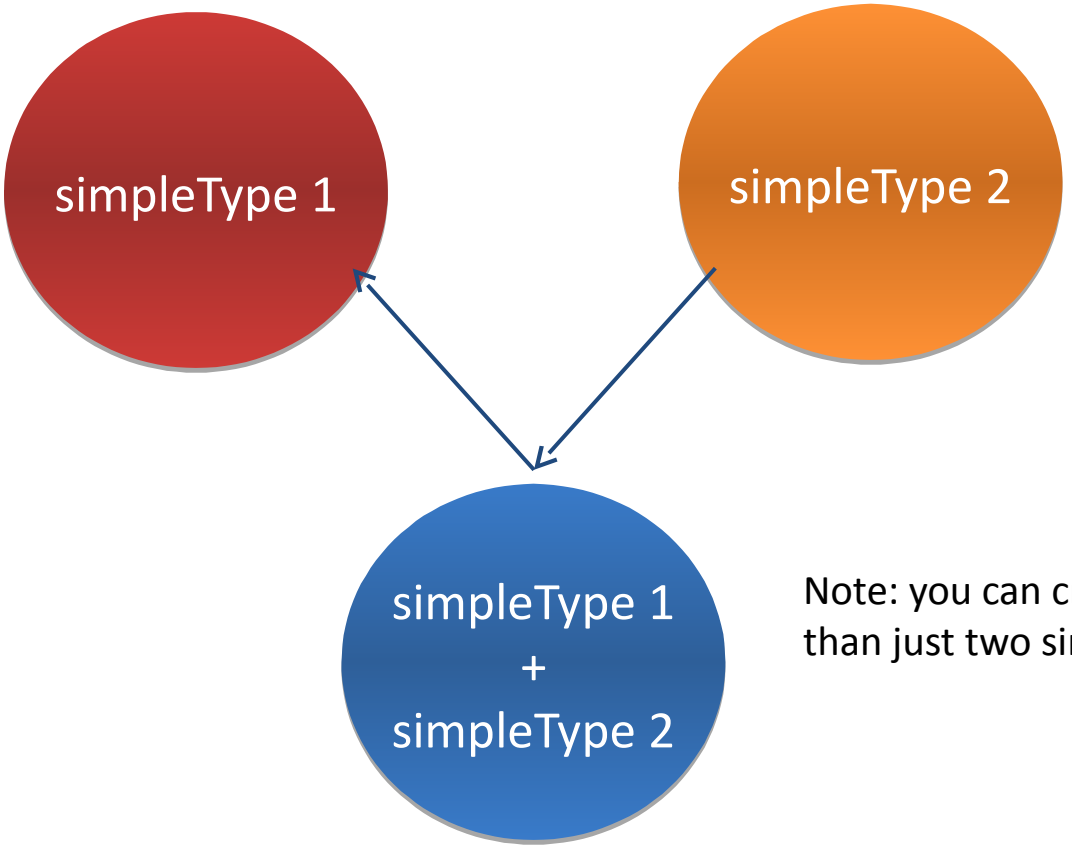
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://www.lottery.org"
     xmlns="http://www.lottery.org" elementFormDefault="qualified">
   <xsd:simpleType name="OneToFortyNine">
     <xsd:restriction base="xsd:positiveInteger">
       <xsd:maxInclusive value="49"/>
     </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="NumbersList">
     <xsd:list itemType="OneToFortyNine"/>
   </xsd:simpleType>
   <xsd:simpleType name="LotteryNumbers">
     <xsd:restriction base="NumbersList">
       <xsd:length value="6"/>
     </xsd:restriction>
   </xsd:simpleType>
   <xsd:element name="LotteryDrawings">
   …
</xsd:schema>
```

List Datatypes

- Notes about the list type
  - You cannot create a list of lists.
  - You cannot create a list of complexTypes.
  - In the instance document, list items are separated by white space chars (blank space, tab, or carriage return)
  - Facets allowed with a list type
    - length: use this to specify the length of the list
    - minLength: use this to specify the minimum length of the list
    - maxLength: use this to specify the maximum length of the list
    - enumeration: use this to specify the values that the list may have
    - pattern: use this to specify the values that the list may have

Union Datatypes

- Creating a simpleType that is a union of types



Note: you can create a union of more than just two simpleTypes

Union Datatypes

- Union type definition

```
<xsd:simpleType name="name">
    <xsd:union memberTypes="space-delimited simpleTypes"/>
</xsd:simpleType>
```

  – or

```
<xsd:simpleType name="name">
    <xsd:union>
        <xsd:simpleType>
            …
        </xsd:simpleType>
        <xsd:simpleType>
            …
        </xsd:simpleType>
            …
    </xsd:union>
</xsd:simpleType>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="http://www.CostelloReunion.org"
                     xmlns="http://www.CostelloReunion.org"
                     elementFormDefault="qualified">
  <xsd:simpleType name="Parent">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Mary"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="PatsFamily">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Pat"/>
      <xsd:enumeration value="Patti"/>
      <xsd:enumeration value="Christopher"/>
      <xsd:enumeration value="Elizabeth"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="BarbsFamily">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Barb"/>
      <xsd:enumeration value="Greg"/>
      <xsd:enumeration value="Dan"/>
      <xsd:enumeration value="Kimberly"/>
    </xsd:restriction>
  </xsd:simpleType>
```

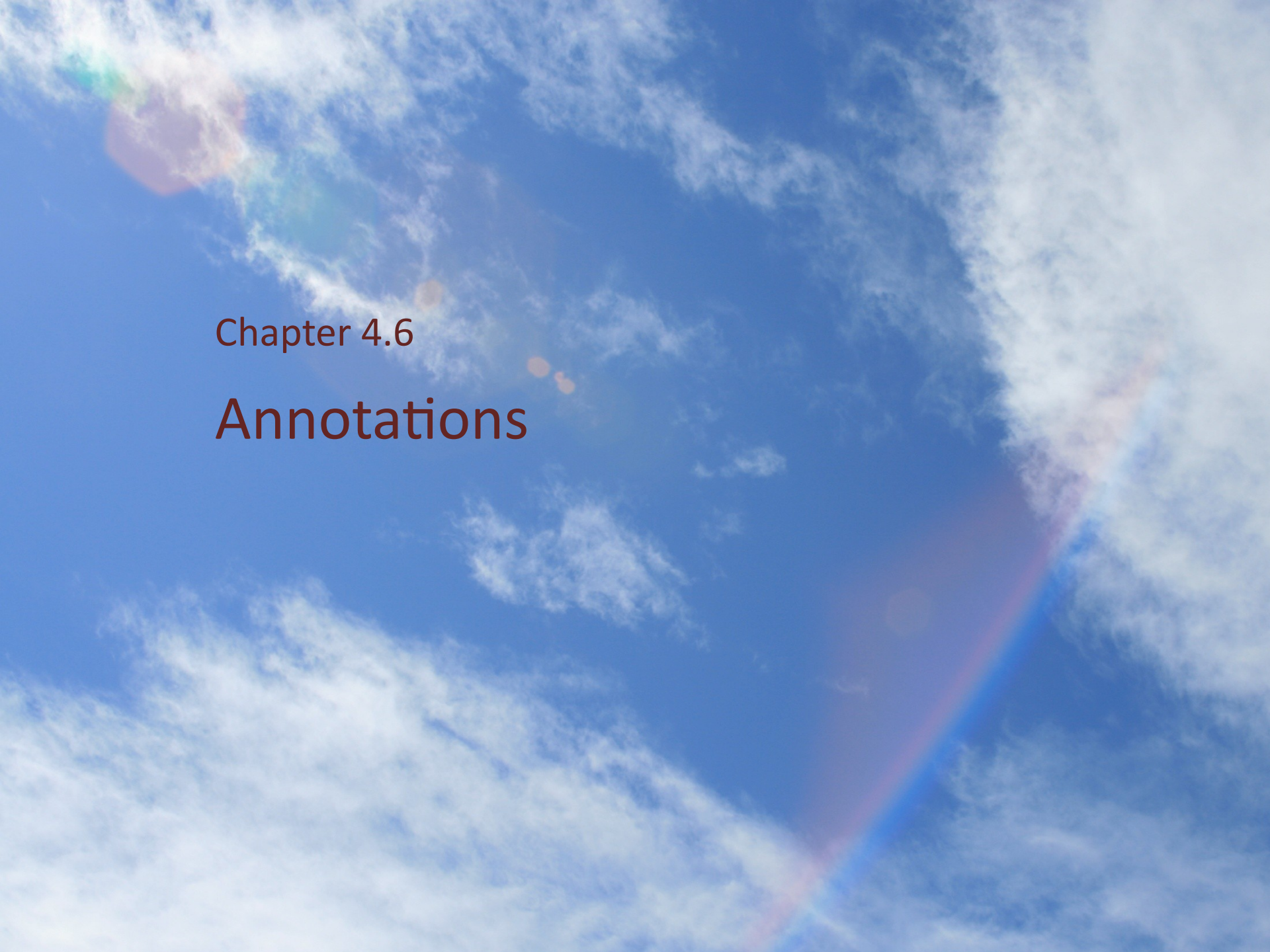
```
<xsd:simpleType name="JudysFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Judy"/>
      <xsd:enumeration value="Peter"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="TomsFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Tom"/>
      <xsd:enumeration value="Cheryl"/>
      <xsd:enumeration value="Marc"/>
      <xsd:enumeration value="Joe"/>
      <xsd:enumeration value="Brian"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="RogersFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Roger"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="JohnsFamily">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="John"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="CostelloFamily">
   <xsd:union memberTypes= "Parent PatsFamily BarbsFamily
                            JudysFamily TomsFamily RogersFamily
                              JohnsFamily"/>
</xsd:simpleType>
```

```
  <xsd:element name="Y2KFamilyReunion">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Participants">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Name" type="CostelloFamily"
                           minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Union Datatypes

- "maxOccurs" is a union type!
  - The value space for maxOccurs is the union of the value space for nonNegativeInteger with the value space of a simpleType which contains only one enumeration value—"unbounded".

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.maxOccurs.org"
                    xmlns="http://www.maxOccurs.org"
                    elementFormDefault="qualified">
  <xsd:simpleType name="unbounded_type">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="unbounded"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="maxOccurs_type">
    <xsd:union memberTypes="unbounded_type xsd:nonNegativeInteger"/>
  </xsd:simpleType>
  <xsd:element name="schema">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="element">
          <xsd:complexType>
            <xsd:attribute name="maxOccurs" type="maxOccurs_type" default="1"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



Chapter 4.6

Annotations

Annotating Schemas

- Three schema elements for schema annotation
  - <annotation> element:
    - used for documenting the schema,
      both for humans and for machines.
  - <documentation>
    - used for providing a comment to humans
  - <appinfo>
    - used for providing a comment to machines
    - content is any well-formed XML
  - Note that annotations have no effect on schema validation

Annotating Schemas

- Example

```
<xsd:annotation>
  <xsd:documentation>
       The following constraint is not expressible with XML Schema:
    The value of element A should be greater than the value of element B.
    So, we need to use a separate tool (e.g., Schematron) to check
    his constraint. We will express this constraint in the
    appinfo section (below).
  </xsd:documentation>
  <xsd:appinfo>
    <assert test="A &gt; B">A should be greater than B</assert>
  </xsd:appinfo>
</xsd:annotation>
```

Annotating Schemas

- Where can you put annotations?
  - Annotations may occur before and after any global component.
  - Annotations may occur only at the beginning of non-global components.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="http://www.books.org"
                    xmlns="http://www.books.org"
                    elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book"  maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Can put annotations only at these locations

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                      targetNamespace="http://www.books.org"
                     xmlns="http://www.books.org"
                     elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book"  maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string">
                <xsd:annotation>
                  <xsd:documentation>This is how to annotate the
Date                                  element!</xsd:documentation>
                </xsd:annotation>
              </xsd:element>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Inline the annotation within the Date element declaration.

Annotating Schemas

- Two optional attributes for the <documentation> element

```
<xsd:documentation  source="http://www.xfront.com/BookReview.txt"
                    xml:lang="FR"/>
```

  - `source`
    this attribute contains a URL to a file
    which contains supplemental information
  - `xml:lang`
    this attribute specifies the language
    that the documentation was written in

Annotating Schemas

- One optional attribute for the <appinfo> element

```
<xsd:appinfo source="http://www.xfront.com/Assertions.xml"/>
```

  - `source`
    this attribute contains a URL to a file
    which contains supplemental information