Session [2]

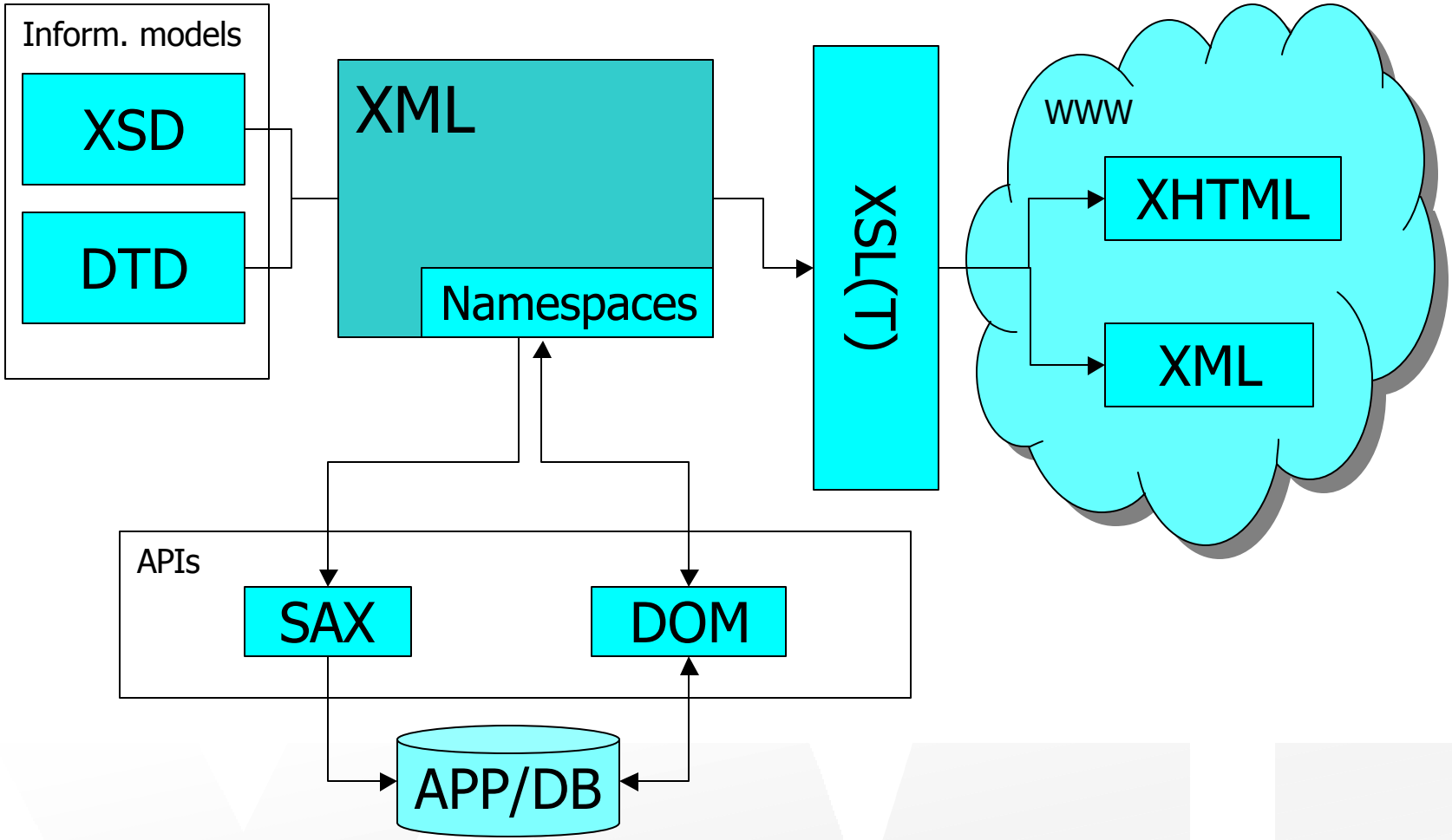
- **Information Modeling with XSD and DTD**

- September 12, 2000
  Horst Rechner

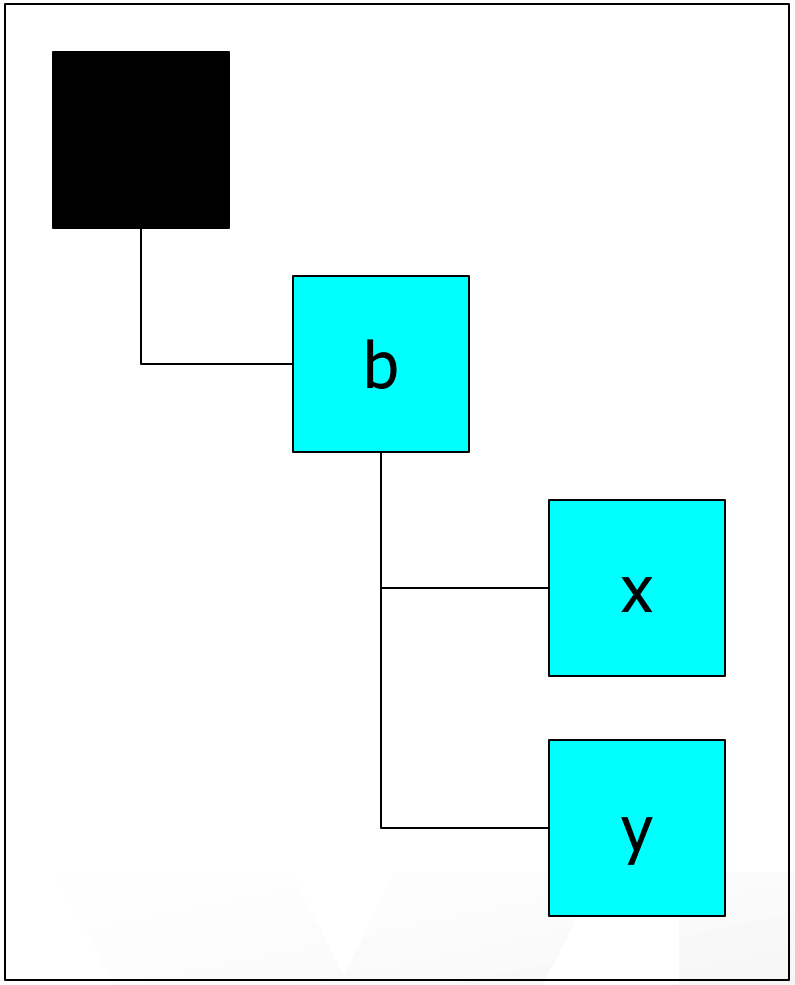
Q&A from Session [1]

- HTML without XML
  - See Code
- HDBMS vs. RDBMS
- What does XDR mean?
  - XML-Data Reduced
  - Utilized in Biztalk

Modeling


Inform. models
XSD
DTD
XML
Namespaces
XSL(T)
www
XHTML
XML
APIs
SAX
DOM
APP/DB

Modeling



Enforce Structure

```
<a>
    <b>
        <x>…</x>
        <y>…</y>
    </b>
</a>
```

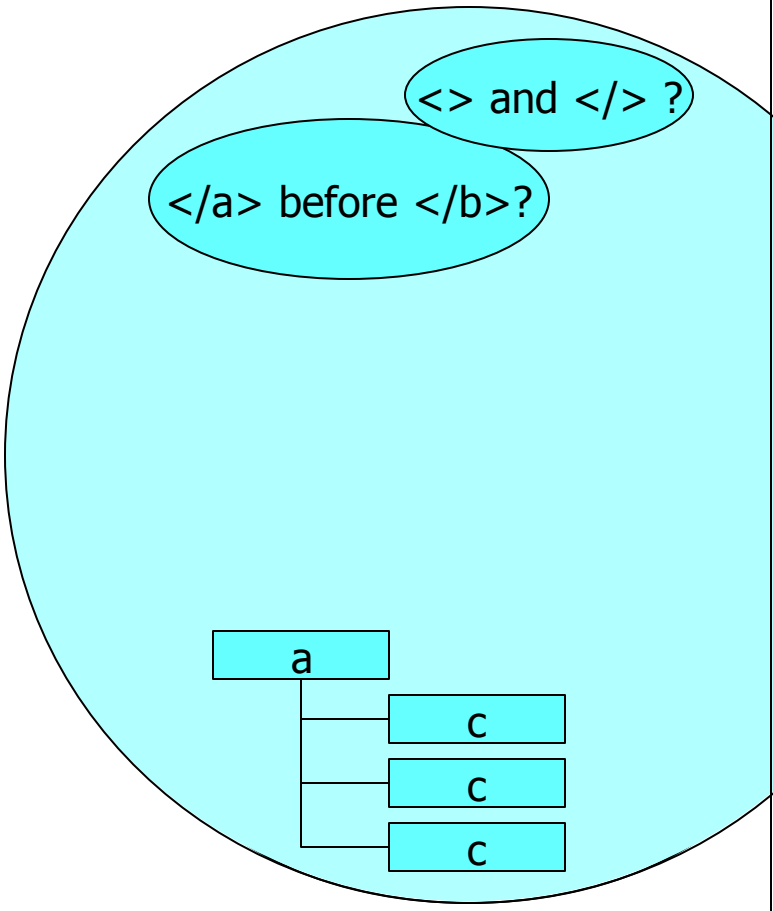
DTD

XSD

XML

XML - Well-Formed and Valid

- Well-Formed:
  - Syntax is correct
    i.e. no nested tags, tags match case
  - A document has to be well formed before it becomes an XML document and can be used any further.
- Valid:
  - Semantic is correct
    i.e. when compared against a information model that says:
    "Name must be a child of address !",
    "Message can have a language attribute"
  - May be valid against one information model document and not valid against another one!



What can you do with Information models?

- Define
  - Elements
    - Approved names and types
    - Ordering and nesting
    - Multiplicity
  - Attributes
    - Approved names and types
    - Optional standard values

What advantage do you get out of it?

- Data control
  - Data is always in a expected order
  - Which allows:
    - Processing
      - Programmers can write standardized processing code
    - Authoring
      - XML tools can help in creating and validating documents

What disadvantage does that have?

- Extra processor time to perform check
- Inconsistent support for schema definitions

The two XML information models

- DTD - document type definition
- XSD - XML Schema definition

DTD - How does it look?

- Uses separate syntax from XML

Address_Book_dtd.xml

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

<!DOCTYPE Address_Book SYSTEM
"Address_Book.dtd">

<Address_Book>
        <Address>
                <Name>Steve</Name>
                <Phone>123</Phone>
        </Address>

        <Address>
        …
```

Adress_Book.dtd

```
<?xml version="1.0"
        encoding="UTF-8"?>

<!ELEMENT Address_Book (Address*)>
<!ELEMENT Address (Name, Phone)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Phone (#PCDATA)>
```

DTD - Internal and external definition

- Internal definition overrides external!

Address_Book_Internal.xml

```
<?xml version="1.0"?>
<!DOCTYPE Address_Book [
        <!ELEMENT Address_Book (Address*)>
        <!ELEMENT Address (Name, Phone)>
        <!ELEMENT Name (#PCDATA)>
        <!ELEMENT Phone (#PCDATA)>
]>
<Address_Book>
        <Address>
        …
```

DTD - declarable Components

- Header (document Type)
- Elements
- Attributes
- Entities
- Notations
- Ignore (not covered here)
- Include (not covered here)

What do we need?

```
<?xml version="1.0"?>
<emailMessage>
     <To>Samuel</To>
     <From>Adam</From>
     <Subject>Party at
        5:00</Subject>
     <Message lang="EN">
          Wanna go?
     </Message>
     <Attachment/>
     <!-- No comment-->
     <?MyPrinter R=128; G=0;
        B=128?>
     <![CDATA[This is not
        parsed!]]>
</emailMessage>
```

DTD - document Type

- `<!DOCTYPE DTDName Identifier Location>`
  - DTDName: Name of document element
    i.e. `Address_Book`
  - Identifier: `SYSTEM` or `PUBLIC` (not implemented)
  - Location: relative or absolute Path or URL

DTD - Elements

- `<!ELEMENT ElementName` **`(Child)`**`>`
  - `(x)` - child must appear once and once only
  - `(x?)` - zero one one times
  - `(x+)` - one or more times
  - `(x*)` - zero or more times
  - `(x|y)` - x or y must appear
  - `(#PCDATA|x)*` - mixed content is allowed
    (this is the minimal syntax for mixed content!)
  - `(#PCDATA)` - parsed text
  - All combinations of the above are possible
  - ~~`(#CDATA)`~~ NO!

Corresponding XML:

```
<ElementName>
        <x>Data</x>
        Text Text
Text
        <x>Data</x>
</ElementName>
```

DTD - Element weaknesses

- No data typing
- This will be resolved in XSD

DTD - Attribute types

- `<!ATTLIST ElementName AttributeName`
  `**AttrType** AttrQualifier defValue>`
  - String: `CDATA`
  - Tokenized: `ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS` (see next slide)
  - Enumerated: `(x|y|z)`

DTD - Tokenized Attributes

- `ID` - unique element
- `IDREF, IDREFS` - reference to ID, that was defined before its reference
- `ENTITY, ENTITIES` - reference to an parsed or unparsed object (see one of next slides)
- `NMTOKEN, NMTOKENS` - Single word, or list of single words stripped of white spaces

DTD - Attribute qualifier

- `<!ATTLIST ElementName AttributeName AttrType` **`AttrQualifier`** `defValue>`
  - `#FIXED:` Default value is fixed
  - `#IMPLIED:` No default value, no one has to be provided
  - `#REQUIRED:` Attribute is required and no default
  - `Enumerated: (x|y|z)`
- Multiples attributes can be specified

DTD - Attribute weaknesses

- No data typing
- This will be resolved in XSD

DTD - Entities

- Parsed
  - Markup characters: `< > & &apos; &quot;`
  - User defined strings: `<!ENTITY SW "Semantic Web">`
  - External ressources: `<!ENTITY resourceName Identifier nameAndLocation>`
  - Parameter Entities: `<!ENTITY % entityName parameterName>`
    Usage (not in .XML, but in .DTD):
    `<!ELEMENT name %entityName;>`
- Unparsed
  - External ressources: data in non-XML format (i.e. images)

Advantages of using external resources: Modular design!

DTD - Parsed external resource entity

Address_Entity.xml

```
<?xml version="1.0"?>
<!DOCTYPE Address_Entity [
        <!ENTITY AOne SYSTEM "Address_EntityOne.xml">
        <!ENTITY ATwo SYSTEM "Address_EntityTwo.xml">
]>
<Address_Entity>
        &AOne;
        &ATwo;
</Address_Entity>
```

Address_EntityOne.xml

```
<Address>
        <Name>Steve</Name>
        <Phone>123</Phone>
</Address>
```

DTD - Unparsed external resource entity

Address_EntityGraphic.xml

```
<?xml version="1.0"?>
<!DOCTYPE Address_Entity [
        <!ELEMENT Address_Entity (Address*)>
        <!ELEMENT Address (Name, Phone, GraphicTag)>
        <!ELEMENT Name (#PCDATA)>
        <!ELEMENT Phone (#PCDATA)>

        <!NOTATION GIF SYSTEM "CompuServer Graphics Interchange
                Format 87a">
        <!ENTITY GraphicFile SYSTEM "Address_GraphicOne.gif">
        <!ELEMENT GraphicTag EMPTY>
                <!ATTLIST GraphicTag FileName ENTITY #REQUIRED>
]>
…
```

DTD - Unparsed external resource entity

Address_EntityGraphic.xml

…

```
<Address_Entity>
        <Address>
                <Name>Steve</Name>
                <Phone>123</Phone>
                <GraphicTag FileName="GraphicFile"/>
        </Address>
</Address_Entity>
```

DTD - Weaknesses

- No data typing
- Elements can not appear in any order
- Uses separate syntax from XML

Elements versus Attributes

- Attributes:
  - Easier to check by processor (see DOM)
  - No tag overhead
  - Default value
  - General: **Describe content**
- Elements:
  - Provide more structure
  - General: **Define content**

```
<?xml version="1.0" encoding="UTF-16"?>
<emailMessage>
    <To>Samuel</To>
    <From>Adam</From>
    <Subject>Party at 5:00</Subject>
    <Message lang="EN">
        Wanna go?
    </Message>
    <Attachment/>
    <!-- No comment-->
    <?MyPrinter R=128; G=0; B=128?>
    <![CDATA[This is not parsed!]]>
</emailMessage>
```

Exercises

- How do I setup the parser?
  - Why do we use XERCES and not IE?
  - XERCES is a validator.
  - IE is only used for rendering.
- How can I validate my code?
  - See `readmefirst.txt`

So...

- We need something better
- XML Schema

XSD - Highlights

- Data typing
  - 41+ datatypes
  - Create your own
- Written in XML
- Any order for elements possible
- Supports namespaces (will be covered later)
- Object-oriented approach
  - Derive new types from old ones

Q&A from Session 2 Part 1

- Can I reference an ID in an attribute before I defined it?
  - Yes, see Code
- Can I change a HTML DTD?
  - Not in the browser. They are hardcoded.
    But in an SGML parser.
    http://www.jclark.com/sp/index.htm
    http://web3.w3.org/TR/html401/sgml/dtd.html

Interesting sites...

- The basics at www.XML101.com

XSD - How does it look?

Address_Book_xsd.xml

```
<?xml version="1.0"?>
<ab:Address_Book
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xsi:schemaLocation="AddressNS Address_Book.xsd"
        xmlns:ab="AddressNS">
        <Address>
                <Name>Steve</Name>
                <Phone>123</Phone>
        </Address>
        <Address>
                <Name>Carol</Name>
                <Phone>321</Phone>
        </Address>
</ab:Address_Book>
```

This version does only validate with XERCES! Change Namespace to validate with XMLSpy

XSD - How does it look?

Address_Book.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        targetNamespace="AddressNS"
        xmlns:ab="AddressNS">
        <xsd:element name="Address_Book" type="ab:Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="ab:AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="AddressType">
                <xsd:element name="Name" type="ab:NameType"/>
                <xsd:element name="Phone" type="ab:PhoneType"/>
        </xsd:complexType>
        <xsd:simpleType name="NameType" base="xsd:string"/>
        <xsd:simpleType name="PhoneType" base="xsd:string"/>
</xsd:schema>
```

XSD - What can you do?

- Only 3 things:
  - Declare elements and attributes
  - Define types that are used by elements/attributes
  - Declare namespaces
- …but in a very powerful and flexible way

Namespaces in DTD

- Used to define where certain elements belong

Address_Book.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        targetNamespace="AddressNS"
        xmlns:ab="AddressNS">
        <xsd:element name="Address_Book" type="ab:Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="ab:AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        …
```

Namespaces in XML

- Used to define where certain elements belong

Address_Book_xsd_spy.xml

```
<?xml version="1.0"?>
<ab:Address_Book
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xsi:schemaLocation="AddressNS Address_Book.xsd"
        xmlns:ab="AddressNS">
        <ab:Address>
                <ab:Name>Steve</ab:Name>
                <ab:Phone>123</ab:Phone>
        </ab:Address>
        …
```

This version does only validate with XMLSpy! Change Namespace to validate with XERCES

If you only have one namespace…

- Make it easy for yourself in XSD

Address_Book_noNamespace.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <xsd:element name="Address_Book" type="Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        …
```

This version validates in both parsers.

If you only have one namespace…

- And make it easy in XML

Address_Book_noNamepace.xml

```
<?xml version="1.0"?>
<Address_Book
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="Address_Book.xsd">
        <Address>
                <Name>Steve</Name>
                <Phone>123</Phone>
        </Address>
        …
```

This version validates in both parsers.

XSD - Elements

- Simple types
  - No children and no attriutes
  - i.e. string, integer
- Complex types
  - Children and attributes allowed
- Anonymous types
  - Used within one element only
  - Cannot be referenced
- Named types
  - Can be referenced

(All combinations are possible i.e. complex anonymous types)

XSD - Named types

Address_Book.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        targetNamespace="AddressNS"
        xmlns:ab="AddressNS">
        <xsd:element name="Address_Book" type="ab:Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="ab:AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="AddressType">
                <xsd:element name="Name" type="ab:NameType"/>
                <xsd:element name="Phone" type="ab:PhoneType"/>
        </xsd:complexType>
        <xsd:simpleType name="NameType" base="xsd:string"/>
        <xsd:simpleType name="PhoneType" base="xsd:string"/>
</xsd:schema>
```

XSD - Anonymous types

Address_Book_Anonymous.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/1999/XMLSchema"
        targetNamespace="AddressNS"
        xmlns:ab="AddressNS">
<xsd:element name="Address_Book">
        <xsd:complexType>
                <xsd:element name="Address" maxOccurs="unbounded">
                        <xsd:complexType>
                                <xsd:element name="Name" type="xsd:string"/>
                                <xsd:element name="Phone" type="xsd:string"/>
                        </xsd:complexType>
                </xsd:element>
        </xsd:complexType>
</xsd:element>
</xsd:schema>
```

XSD - Type references

- Used to reference a defined type

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
          xmlns:xsd="http://www.w3.org/1999/XMLSchema"
          targetNamespace="AddressNS"
          xmlns:ab="AddressNS">
<xsd:element name="Address_Book">
          <xsd:complexType>
                    <xsd:element ref="Address"/>
          </xsd:complexType>
</xsd:element>

<xsd:element name="Address" maxOccurs="unbounded">
          <xsd:complexType>
                    <xsd:element name="Name" type="xsd:string"/>
                    <xsd:element name="Phone" type="xsd:string"/>
          </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Does neither work with XERCES nor XMLSpy!

XSD - Data typing

- All DTD types (i.e. ID, NMTOKEN) are included in XSD
- Users can build their own simple and complex types

XSD - Build in types (excerpt)

• String	"Semantic Web"
• Boolean	true, false, 0, 1
• Float / double	1.34 (single, double precision)
• Integer, long, short	10000
• timeInstant	2000-02-25T13:20:00.000-05:00 (in this case EST)
• timeDuration	P1Y2M3DT10H30M12.3S
• recurringDate	--05-31 (every 31. May)
• Binary	010101
• Uri	http://www.semantic.web
• DTD attribute types	ID, IDREF, ENTITY, …

For a complete list see: http://www.w3.org/TR/xmlschema-0/#CreatDt

XSD - Creating your own types

- We can derive new types from every existing base type (build in type)
- Additional characteristics are called **facets**
- Facets are (not all facets go with each type)
  - Pattern
  - Enumeration
  - Length
  - Precison / scale
  - minInclusive/Exclusive
  - maxInclusive/Exclusive
  - min/maxlength

For a complete facet list see: http://www.w3.org/TR/xmlschema-0/#SimpleTypeFacets

XSD - Creating your own types

- <simpleType name="name" base="source">
  <facet value="value"/>
  <facet value="value"/>
  …
  </simpleType>
- <simpleType name="Tel" base="string">
  <length value="8"/>
  <pattern value="\d{3}-\d{4}"/>
  </simpleType>

Regular Expressions Parser: http://www.xfront.org/xml-schema/

XSD - Creating your own types

- <simpleType name="States"
  base="string">
  <enumeration value="AK"/>
  <enumeration value="ME"/>
  …
  </simpleType>

XSD - Creating your own type

- minOccurs and maxOccurs in elements
  - Default value of minOccurs is one (means required)
  - If maxOccurs is “unbounded”, there is no maximum
  - If there is no maxOccurs then maxOccurs = minOccurs
  - If there is no min/maxOccurs, then the element must occur exactly once

XSD - only Attributes

Address_Book_attributes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema“>
        <xsd:element name="Address_Book" type="Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="AddressType“
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="AddressType“ content=“empty”>
                <xsd:attribute name=“name” type=“NameType”/>
                <xsd:attribute name=“phone" type="PhoneType"/>
        </xsd:complexType>
        <xsd:simpleType name=“NameType” base=“xsd:string”/>
        <xsd:simpleType name="PhoneType" base="xsd:string"/>
</xsd:schema>
```

XSD - only Attributes to link a image

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <xsd:element name="Gallery" type="GalleryType"/>
        <xsd:complexType name="GalleryType">
                <xsd:element name="img" type="imageType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="imageType" content="empty">
                <xsd:attribute name="href" type="uriReference"
                        use="required"/>
        </xsd:complexType>
</xsd:schema>
```

In the XML file: `<img href="http://www.image.com/myimage.gif"/>`

XSD - Attribute qualifier

- You can use the keyword “use” as a additional parameter in an attribute declaration to specify:
  - Required attribute
  - Optional attribute
  - Fixed attribute
  - Prohibited attribute
- i.e. `<attribute name=“abc” type=“abcType” use=“fixed” value=“myValue”>`

XSD - Attributes with text in the Element

Address_Book_attributesWText.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <xsd:element name="Address_Book" type="Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="AddressType"
                derivedBy="extension" base="string">
                <xsd:attribute name="name" type="NameType"/>
                <xsd:attribute name="phone" type="PhoneType"/>
        </xsd:complexType>
        <xsd:simpleType name="NameType" base="xsd:string"/>
        <xsd:simpleType name="PhoneType" base="xsd:string"/>
</xsd:schema>
```

XSD - Derived types (OO-Concept)

- We can form subclasses of types
- Simple data types can be restricted
  - i.e. a more restricted range of values
- You also should be able to restrict complex data types, but…
- Complex data types can be extended

XSD - Restricting simple types

- We have already done that on a previous slide: XSD - Creating your own types
-
```
<simpleType name="name" base="source">
   <facet value="value"/>
   <facet value="value"/>
</simpleType>
```
-
```
<simpleType name="Tel" base="string">
   <length value="8"/>
   <pattern value="\d{3}-\d{4}"/>
</simpleType>
```

Regular Expressions Parser: http://www.xfront.org/xml-schema/

XSD - Extending complex types

- See code: Address_Book_deriving.xml/xsd
- Extension and restriction can be blocked, by using the attribute block in the xsd file (not yet supported)

XSD - Additional concepts

- Mixed content
- Null content
- Groups (won’t be covered here)
- Alternatives with “choice”
- Any order with “all”
- Anotations (won’t be covered here)
- Typecasting (feature not yet supported)
- Equivalent elements
  - Abstract elements (won’t be covered here)

XSD - mixed content

Address_Book_mixed.xml

```
<?xml version="1.0"?>
<Address_Book
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="Address_Book.xsd">
        <Address>Steve
                <Phone>123</Phone>
        </Address>
        <Address>Carol
                <Phone>321</Phone>
        </Address>
</Address_Book>
```

XSD - mixed content

- No type definition for name possible

Address_Book_mixed.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <xsd:element name="Address_Book" type="Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="AddressType" content="mixed">
                <xsd:element name="Phone" type="PhoneType"/>
        </xsd:complexType>
        <xsd:simpleType name="PhoneType" base="xsd:string"/>
</xsd:schema>
```

XSD - null content

- The element is not empty but NULL (databases!)

Definition:

```
<xsd:simpleType name="Name" base="xsd:string" nullable="true"/>
```

Usage:

```
<Name xsi:null="true"/>
```

XSD - Alternatives

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <xsd:element name="Address_Book" type="Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="AddressType">
                <choice minOccurs="0" maxOccurs="unbounded">
                        <xsd:element name="Phone" type="PhoneType"/>
                        <xsd:element name="Name" type="NameType"/>
                </choice>
        </xsd:complexType>
        <xsd:simpleType name="PhoneType" base="xsd:string"/>
        <xsd:simpleType name="NameType" base="xsd:string"/>
</xsd:schema>
```

XSD - Any order

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
        <xsd:element name="Address_Book" type="Address_BookType"/>
        <xsd:complexType name="Address_BookType">
                <xsd:element name="Address" type="AddressType"
                        maxOccurs="unbounded"/>
        </xsd:complexType>
        <xsd:complexType name="AddressType">
                <all>
                        <xsd:element name="Phone" type="PhoneType"/>
                        <xsd:element name="Name" type="NameType"/>
                </all>
        </xsd:complexType>
        <xsd:simpleType name="PhoneType" base="xsd:string"/>
        <xsd:simpleType name="NameType" base="xsd:string"/>
</xsd:schema>
```

XSD - Equivalent elements

- equivClass elements must be declared on the top level
- Types must be the same

```
…
<xsd:element name="Phone" type="PhoneType"/>
<xsd:element name="Tel" equivClass="Phone" type="PhoneType">
<xsd:element name="Name" type="NameType"/>
…
```

XSD - Additional concepts

- Include / import
- Uniqueness / key / reference to a key
- anyAttribute
- Open ContentModel

- Not covered in this seminar.

Exercises

- Write a schema for `juicers_xsd.xml`

XML - Rendering and Transformation

- How do you do it?
- Next session:
  XML Rendering and Transformation with XSL(T)