

DTD and XML Schema



XML Document Type Definitions

- part of the original XML specification
- an XML document may have a DTD
- terminology for XML:
 - well-formed: if tags are correctly closed
 - valid: if it has a DTD and conforms to it
- validation is useful in data exchange

Very Simple DTD

```
<!DOCTYPE company [  
    <!ELEMENT company ((person|product)*)>  
    <!ELEMENT person (ssn, name, office, phone?)>  
    <!ELEMENT ssn (#PCDATA)>  
    <!ELEMENT name (#PCDATA)>  
    <!ELEMENT office (#PCDATA)>  
    <!ELEMENT phone (#PCDATA)>  
    <!ELEMENT product (pid, name, description?)>  
    <!ELEMENT pid (#PCDATA)>  
    <!ELEMENT description (#PCDATA)>  
]>
```

Example of Valid XML document

```
<company>
  <person><ssn> 123456789 </ssn>
    <name> John </name>
    <office> B432 </office>
    <phone> 1234 </phone>
  </person>
  <person><ssn> 987654321 </ssn>
    <name> Jim </name>
    <office> B123 </office>
  </person>
  <product> ... </product>
  ...
</company>
```

Content Model

- Element content:
 - what we can put in an element (aka content model)
- Content model:
 - Complex = a regular expression over other elements
 - Text-only = #PCDATA
 - Empty = EMPTY
 - Any = ANY
 - Mixed content = (#PCDATA | A | B | C)*
 - (i.e. very restricted)

Attributes

```
<!ELEMENT person (ssn, name, office, phone?)>
<!ATTLIS person age CDATA #REQUIRED>
```

```
<person age="25">
  <name>....</name>
  ...
</person>
```

Attributes

```
<!ELEMENT person (ssn, name, office, phone?)>
<!ATTLIS person age      CDATA #REQUIRED
          id          ID      #REQUIRED
          manager    IDREF   #REQUIRED
          manages   IDREFS #REQUIRED
>
```

```
<person age="25",
          id="p29432",
          manager="p48293",
          manages="p34982 p423234">
<ssn> ...
</ssn>
<name> ...
</name>
...
</person>
```

Attribute Types

- **CDATA** : string
- **ID** : key
- **IDREF** : foreign key
- **IDREFS** : foreign keys separated by space
- **(Monday | Wednesday | Friday)** : enumeration
- **NMTOKEN** : must be a valid XML name
- **NMTOKENS** : multiple valid XML names
- **ENTITY** : Reference to, e.g., an external file

Attribute Kind

- #REQUIRED
- #IMPLIED : optional
- *value* : default value
- *value* #FIXED : the only value allowed

Using DTDs

- Must include in the XML document
- Either include the entire DTD:
`<!DOCTYPE rootElement [.....]>`
- Or include a reference to it:
`<!DOCTYPE rootElement SYSTEM
"http://www.mydtd.org/file.dtd">`
- Or mix the two... (e.g. to override the external definition)

DTDs as Grammars

```
<!DOCTYPE paper [  
    <!ELEMENT paper (section*)>  
    <!ELEMENT section ((title, section*) | text)>  
    <!ELEMENT title (#PCDATA)>  
    <!ELEMENT text (#PCDATA)>  
]>
```

```
<paper><section><text></text></section>  
<section><title></title></section>  
<section> ... </section>  
<section> ... </section>  
</section>  
</paper>
```

DTDs as Grammars

- A DTD = a grammar
- A valid XML document = a parse tree for that grammar

XML Schemas

- generalizes DTDs
- uses XML syntax
- two documents: structure and datatypes
 - <http://www.w3.org/TR/xmlschema-1>
 - <http://www.w3.org/TR/xmlschema-2>
- XML-Schema is very complex
 - often criticized
 - some alternative proposals

Why XML Schemas?

- DTDs provide a very weak specification language
 - You can't put any restrictions on text content
 - You have very little control over mixed content (text plus elements)
 - Little control over ordering of elements
- DTDs are written in a strange (non-XML) format
 - Separate parsers for DTDs and XML
- The **XML Schema** **D**efinition language solves these problems
 - XSD gives you much more control over structure and content
 - XSD is written in XML

Why not XML schemas?

- DTDs have been around longer than XSD
 - Therefore they are more widely used
 - Also, more tools support them
- XSD is very verbose, even by XML standards
- More advanced XML Schema instructions can be non-intuitive and confusing
- Nevertheless, XSD is not likely to go away quickly

Referring to a Schema

- To refer to a DTD in an XML document, the reference goes **before** the root element:

```
<?xml version="1.0"?>  
<!DOCTYPE rootElement SYSTEM "$url!">  
<rootElement> ... </rootElement>
```

- To refer to an XML Schema in an XML document, the reference goes **in** the root element:

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

```
<rootElement
```

```
xmllns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
(the XML Schema Instance reference is required)  
xsi:noNamespaceSchemaLocation="$url.xsd">  
(where your XML Schema definition can be found)  
...  
</rootElement>
```

The XSD Document

- Since the XSD is written in XML, it can get confusing which we are talking about
- Except for the additions to the root element of our XML data document, we will discuss the XSD schema document
 - The file extension is .xsd
 - The root element is <schema>
 - The XSD starts like this:

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```
 - The last line specifies where all XSD tags are defined

“Simple” and “Complex” Elements

- A **simple element** is one that contains text and nothing else
 - A simple element cannot have attributes
 - A simple element cannot contain other elements
 - A simple element cannot be empty
 - However, the text can be of many different types, and may have various restrictions applied to it
- If an element isn't simple, it's **complex**
 - A complex element may have attributes
 - A complex element may be empty, or it may contain text, other elements, or both text and other elements

Defining a Simple Element

- A simple element is defined as
`<xs:element name="name" type="type" />`
- where:
 - *name* is the name of the element
 - the most common values for *type* are
 - `xs:boolean`
 - `xs:date`
 - `xs:string`
 - `xs:decimal`
 - `xs:time`
- Other attributes a simple element may have:
 - `default="default value"` if no other value is specified
 - `fixed="value"` no other value may be specified

Defining an Attribute

- Attributes themselves are always declared as simple types
- An attribute is defined as
 - `<xs:attribute name="name" type="type" />`
- where:
 - *name* and *type* are the same as for `xs:element`
- Other attributes a simple element may have:
 - `default="default value"` if no other value is specified
 - `fixed="value"`
 - `use="optional"` the attribute is not required (default)
 - `use="required"` the attribute must be present

Restrictions or "Facets"

- The general form for putting a restriction on a text value is:

```
<xS:element name="name">
  <xS:restriction base="type">
    ...
    ... the restrictions ...
    </xS:restriction>
  </xS:element>
```

- Example

```
<xS:element name="age">
  <xS:restriction base="xS:integer">
    <xS:minInclusive value="0">
    <xS:maxInclusive value="140">
  </xS:restriction>
</xS:element>
```

- Restriction on attribute: use xS:attribute

Restrictions on Numbers

- `minInclusive` -- number must be \geq the given *value*
- `minExclusive` -- number must be $>$ the given *value*
- `maxInclusive` -- number must be \leq the given *value*
- `maxExclusive` -- number must be $<$ the given *value*
- `totalDigits` -- number must have exactly *value* digits
- `fractionDigits` -- number must have no more than *value* digits after the decimal point

Restrictions on Strings

- **length** : the string must contain exactly value characters
- **minLength** : the string must contain at least value characters
- **maxLength** : the string must contain no more than value characters
- **pattern** : the value is a regular expression that the string must match
- **whiteSpace** : not really a "restriction"--tells what to do with whitespace
 - **value="preserve"**: Keep all whitespace
 - **value="replace"**: Change all whitespace characters to spaces
 - **value="collapse"**: Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration

- An enumeration restricts the value to be one of a fixed set of values
- Example:

```
<xs:element name="season">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Spring"/>
      <xs:enumeration value="Summer"/>
      <xs:enumeration value="Autumn"/>
      <xs:enumeration value="Fall"/>
      <xs:enumeration value="Winter"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Complex Elements

- A complex element is defined as

```
<xss:element name="name">  
  <xss:complexType>  
    ...  
    information about the complex type ...  
  </xss:complexType>  
</xss:element>
```

- Attributes are always simple types

Example of a Complex Element

```
<xss:element name="person">
  <xss:complexType>
    <xss:sequence>
      <xss:element name="firstName" type="xs:string"/>
      <xss:element name="lastName" type="xs:string"/>
    </xss:sequence>
  </xss:complexType>
</xss:element>
```

- **<xss:sequence>** says that elements must occur in this order
- Remember that attributes are always simple types

Another Complex Element

```
<xsd:element name="paper" type="papertype"/>
<xsd:complexType name="papertype">
  <xsd:sequence>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="author" minOccurs="0"/>
    <xsd:element name="year"/>
    <xsd: choice><xsd:element name="journal"/>
      <xsd:element name="conference"/>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

DTD:

<!ELEMENT paper (title, author*, year, (journal|conference))>

Declaration and Use

- Types can be declared/defined for later uses
- To use a type, use it as the value of `type="..."`
- Examples:
 - `<x:element name="student" type="person"/>`
 - `<x:element name="professor" type="person"/>`
- Scope is important: you cannot use a type if is local to some other type

Elements v.s. Types in XML Schema

```
<xsd:element  
    name="person">  
    <xsd:complexType>  
        <xsd:sequence>  
            <xsd:element  
                name="name"  
                type="xsd:string" />  
            <xsd:element  
                name="address"  
                type="xsd:string" />  
        </xsd:sequence>  
    </xsd:complexType>  
</xsd:element>
```

```
<xsd:element  
    name="person"  
    type="ttt" />  
    <xsd:complexType  
        name="ttt">  
        <xsd:sequence>  
            <xsd:element  
                name="name"  
                type="xsd:string" />  
            <xsd:element  
                name="address"  
                type="xsd:string" />  
        </xsd:sequence>  
    </xsd:complexType>  
</xsd:element>
```

DTD: <!ELEMENT person (name, address)>

Elements v.s. Types in XML Schema

- **Types:**

- Simple types (integers, strings, ...)
- Complex types (regular expressions, like in DTDs)
- Element-type-element alternation:
 - Root element has a complex type
 - That type is a regular expression of elements
 - Those elements have their complex types...
 - ...
- On the leaves we have simple types

Global and Local Definitions

- Elements declared at the “top level” of a <schema> are available for use throughout the schema
- Elements declared within a <xS:complexType> are local to that type

```
<xS:element name="person">
  <xS:complexType>
    <xS:sequence>
      <xS:element name="firstName" type="xS:string" />
      <xS:element name="lastName" type="xS:string" />
    </xS:sequence>
  </xS:complexType>
</xS:element>
```

- **firstName** and **lastName** elements are locally declared
- The order of declarations at the “top level” of a <schema> do not specify the order in the XML data document

Local and Global Types in XML Schema

- Local type:

```
<xsd:element name="person">  
... define locally the person's type ...  
</xsd:element>
```

- Global type:

```
<xsd:element name="person" type="ttt" />  
  
<xsd:complexType name="ttt">  
... define here the type ttt ...  
</xsd:complexType>
```

- Global types can be reused in other elements

Local v.s. Global Elements in XML Schema

- Local element:

```
<xsd:complexType name="ttt">  
  <xsd:sequence>  
    <xsd:element name="address" type="..." />...  
  </xsd:sequence>  
</xsd:complexType>
```

- Global element:

```
<xsd:element name="address" type="..." />  
  
<xsd:complexType name="ttt">  
  <xsd:sequence>  
    <xsd:element ref="address" /> ...  
  </xsd:sequence>  
</xsd:complexType>
```

- Global elements: like in DTDs

Local Names in XML Schema

name has different meanings in person and in product

```
<xsd:element name="person">
  <xsd:complexType>
    . . .
    <xsd:element name="name">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="lastname" type="xsd:string"/>
        </xsd:sequence>
      </xsd:element>
    . . .
    </xsd:complexType>
  </xsd:element>

<xsd:element name="product">
  <xsd:complexType>
    . . .
    <xsd:element name="name" type="xsd:string"/>
    . . .
    </xsd:complexType>
  </xsd:element>
```

xs:sequence

- We've already seen an example of a complex type whose elements must occur in a specific order:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstName" type="xs:string" />
      <xs:element name="lastName" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

xs:all

- xs:all allows elements to appear in any order

```
<xss:element name="person">
  <xss:complexType>
    <xss:all>
      <xss:element name="firstName" type="xss:string" />
      <xss:element name="lastName" type="xss:string" />
    </xss:all>
  </xss:complexType>
</xss:element>
```

- The members of an xs:all group can occur exactly once
- You can use minOccurs="n" and maxOccurs="n" to specify how many times an element may occur (default value is 1)

All Group

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all> <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items" type="Items"/>
    </xsd:all>
    <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>
```

- A **restricted form of &** in SGML
- Restrictions:
 - Only at top level
 - Has only elements
 - Each element occurs at most once
 - E.g. "comment" occurs 0 or 1 times

Regular Expressions in XML Schema

Recall the element-type-element alternation:

```
<xsd:complexType name="....">  
  [regular expression on elements]  
</xsd:complexType>
```

Regular expressions:

- <xsd:sequence> A B C </...> = A B C
- <xsd:choice> A B C </...> = A | B | C
- <xsd:group> A B C </...> = (A B C)
- <xsd:... minOccurs="0" maxOccurs="unbounded"> ... </...> = (...)*
- <xsd:... minOccurs="0" maxOccurs="1"> ... </...> = (...)?

Referencing

- Once you have defined an element or attribute (with name="..."), you can refer to it with ref="..."
- Example:

```
<xss:element name="person">
  <xss:complexType>
    <xss:all>
      <xss:element name="firstName" type="xs:string" />
      <xss:element name="lastName" type="xs:string" />
    </xss:all>
  </xss:complexType>
</xss:element>
```



```
<xss:element name="student" ref="person">
```

- Or just: <xss:element ref="person">

Attributes Again

- Attributes are associated to the type, not to the element
- Only to complex types; more trouble if we want to add attributes to simple types

```
<xsd:element name="paper" type="xsd:string"/>
<xsd:complexType name="paperType">
  <xsd:sequence>
    <xsd:element name="title" type="xsd:string"/>
    ...
  </xsd:sequence>
  <xsd:attribute name="language" type="xsd:NMTOKEN"
    fixed="English"/>
</xsd:complexType>
```

Text Element with Attributes

- If a text element has attributes, it is no longer a simple type

```
<xss:element name="population">
<xss:complexType>
  <xss:simpleContent>
    <xss:extension base="xss:integer">
      <xss:attribute name="year" type="xss:integer">
        </xss:extension>
      </xss:simpleContent>
    </xss:complexType>
  </xss:element>
```

"Any" Type

- Means anything is permitted there

```
<xsd:element name="anything" type="xsd:anyType"/>  
    ...
```

Empty Elements

- Empty elements are (ridiculously) complex

```
<xsd:complexType name="counter">
  <xsd:complexContent>
    <xsd:extension base="xsd:anyType"/>
      <xsd:attribute name="count" type="xsd:integer"/>
    </xsd:complexContent>
  </xsd:complexType>
```

Mixed Elements

- Mixed elements may contain both text and elements
- We add `mixed="true"` to the `xs:complexType` element
 - The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

```
<xs:complexType name="paragraph" mixed="true">
<xs:sequence>
  <xs:element name="someName" type="xs:anyType"/>
</xs:sequence>
</xs:complexType>
```

Predefined String Types

- Recall that a simple element is defined as:
`<xs:element name="name" type="type" />`
- Here are a few of the possible string types:
 - `xs:string` -- a string
 - `xs:normalizedString` -- a string that doesn't contain **tabs**, **newlines**, or carriage returns
 - `xs:token` -- a string that doesn't contain any whitespace other than single spaces
- Allowable restrictions on strings:
 - `enumeration`, `length`, `maxLength`, `minLength`, `pattern`, `whiteSpace`

Predefined Date and Time Types

- `xs:date` -- A date in the format CCYY-MM-DD, for example, 2002-11-05
- `xs:time` -- A date in the format hh:mm:ss (hours, minutes, seconds)
- `xs:dateTime` -- Format is CCYY-MM-DDThh:mm:ss
- Allowable restrictions on dates and times:
 - `enumeration`, `minInclusive`, `maxExclusive`, `maxInclusive`, `maxExclusive`, `pattern`, `whiteSpace`

Predefined Numeric Types

- Here are some of the predefined numeric types:

xs:decimal xs:positiveInteger
xs:byte xs:negativeInteger
xs:short xs:nonPositiveInteger
xs:int xs:nonNegativeInteger
xs:long

- Allowable restrictions on numeric types:

- enumeration, minInclusive, maxExclusive, maxInclusive,
maxExclusive, fractionDigits, totalDigits, pattern,
whiteSpace

Extensions

- You can base a complex type on another complex type

```
<xss:complexType name="newType">
<xss:complexContent>
  <xss:extension base="otherType">
    ...new stuff...
  </xss:extension>
</xss:complexContent>
</xss:complexType>
```

Derived Types by Extensions (Inheritance)

```
<complexType name="Address">
<sequence> <element name="street" type="string"/>
<element name="city" type="string"/>
</sequence>
</complexType>

<complexType name="USAAddress">
<complexContentContent>
<extension base="ipo:Address">
<sequence> <element name="state" type="ipo:USState"/>
<element name="zip" type="positiveInteger"/>
</sequence>
</extension>
</complexContentContent>
</complexType>
```

Derived Types by Restrictions

```
<complexContent>
  <restriction base="ipo:Items">
    ... [rewrite the entire content, with restrictions]...
  </restriction>
</complexContent>
```

- (*): may restrict cardinalities, e.g. (0,infty) to (1,1);
may restrict choices; other restrictions...

Summary

- Similar role to DTD: define structures for XML documents
- XML Schema definition itself is in XML
- Detailed simple types
 - `String`, `Token`, `Byte`, `unsignedByte`, `Integer`,
`positiveInteger`, `Int` (larger than `Integer`),
`unsignedInt`, `Long`, `Short`, `Time`, `dateTime`,
`Duration`, `Date`, `ID`, `IDREF`, `IDREFS`, ...
- 15 facets