XML

XML Schemas
An introduction

What are XML Schemas?

- Like DTDs, XML Schemas are used for defining XML vocabularies.
- They describe the structure and content of XML documents in more detail than DTDs, and therefore allow more precise validation.
- XML Schemas are a mature technology used in a variety of applications, like XML validation; XQuery and SOAP.

Benefits of XML Schemas

- Created using basic XML, whereas DTDs utilise a separate syntax.
- Fully support namespaces.
- Enable you to validate text element content based on built-in and user-defined data types.
- Create complex and reusable content models easily.
- Allow modelling of programming concepts such as object inheritance and type substitution.

Schemas use XML syntax

- Contrary to DTDs that use their own syntax, schemas use XML.
- However, the way of defining rules in schemas has many similarities with the process in DTDs.
- For example, in a DTD you would write

```
<!ELEMENT first (#PCDATA)>
```

- The same rule in XML Schemas (approximately)

```
<element name=“first” type=“string” />
```

XML Schema data types

- In DTD you can specify that an element has
  - Mixed content
  - Element content
  - Empty content
- If your elements contain only text, you cannot add constraints on the format of the text.
  - Attribute declarations give you some control, but even then the data types you can use in attribute declarations are very limited.

XML Schema data types

- XML Schemas divide data types into two categories:
  - Complex
  - Simple
- Elements that may contain attributes or other elements are declared as complex types.
- Attribute values and text content within elements are declared using simple types.

XML Schema data types

- By utilising these types you can specify if an element may contain only positive numbers, dates or numbers within a certain range.
- Many commonly-used simple types are built into XML Schemas. This is the most important feature within XML Schemas.

List of common data types

Here are some common data types. This is not an extensive list, only some basic data types are shown:

TYPE	DESCRIPTION
string	Any character data
token	A string that does not contain sequences of two or more spaces, tabs, carriage returns or linefeed characters.
integer	A numeric value representing a whole number
positiveInteger / negativeInteger	An integer whose value is greater/less than 0
decimal	A numeric value that may or may not include a fractional part.
anyURI	A valid Uniform Resource Identifier (URI)

XML and DTD samples

```
<?xml version="1.0" encoding="UTF-8"?>
<employee type="permanent">
  <firstname>Mark</firstname>
  <surname>Collins</surname>
</employee>
```

Filename: Employee.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT employee (firstname, surname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
```

Filename: Employee.dtd

Schema that validates the XML doc

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema”> *
   <element name="employee">
      <complexType>
         <sequence>
            <element name="firstname" type="string"/>
            <element name="surname" type="string"/>
         </sequence>
      </complexType>
   </element>
</schema>
```

Filename: Employee_alt_noNS.xsd

* For the complete schema element see slide 12 or the example file.

Defining XML Schemas

- The <schema> element is the root element within an XML schema and it enables you to declare namespace information as well as defaults for declarations throughout the document. You can optionally add a version.

  ```
  <schema targetNamespace="URI"
  attributeFormDefault="qualified or
  unqualified"
  elementFormDefault="qualified or
  unqualified"
  version="version number">
  ```

Example XML Schema element

```
<schema xmlns=“http://www.w3.org/2001/XMLSchema”
targetNamespace=“http://www.dcs.bbk.ac.uk/ns”
elementFormDefault=“qualified”>
```

- `xmlns`: namespace declaration for this schema file. Usually the xs or xsd prefix is used: `xmlns:xs`
- `targetNamespace:` indicates that you are developing a vocabulary for the namespace http://www.dcs.bbk.ac.uk/ns. This needs to be a unique URI
- `elementFormDefault` is best to have the value qualified. If it is not included it is assumed to have the value unqualified. It controls the way namespaces are used within the corresponding XML document.

Schema elements

- `element`: This declares an element that exists in the XML document. Its name is given in the name attribute.
  - `<element name="employee" />`
- `complexType`: This definition enables you to specify the allowable elements and their order as well as any attribute declarations.
- `sequence`: This definition means that the elements declared in the Schema appear in the declared sequence inside the XML file.

Schema data type declaration

- If an element is going to have only textual content, you can set the type attribute:

  ```
  <element name=“firstname” type=“string”/>
  ```

- If an element has attributes and/or child elements it is then a `complexType`.

Schema attributes declaration

```
   <attribute name=“contract” type=“string” />
```

- This is a declaration of an attribute that can contain an alphanumeric value.
- This declaration is added in the complexType:

```
<complexType>
   <sequence>
      <!– Element declarations emitted -->
   </sequence>
   <attribute name=“contract” type=“string” />
</complexType>
```

Connecting an XML document to an XML Schema

- You need to add a default namespace to the root element of your XML document that matches the value of the `targetNamespace` attribute in the Schema.
- You also declare the namespace http://`www.w3.org/2001/XMLSchema-instance` usually with the `xsi` prefix.
- The `schemaLocation` attribute contains a *namespace-location* value pair. It contains two values separated by a space.
  - The first value is the namespace of your XML document, which needs to be the same with the targetNamespace in the Schema
  - The second value is the URL that points to the XML Schema that describes your namespace. The URL can be either relative or absolute.

XML Schema example (final)

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="http://
www.dcs.bbk.ac.uk/~dionisis/ns/employee">
    <element name="employee">
        <complexType>
            <sequence>
                <element name="firstname" type="string" />
                <element name="surname" type="string"/>
            </sequence>
            <attribute name="type" type="string" />
        </complexType>
    </element>
</schema>
```

Filename: employee.xsd

XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<employee xmlns="http://www.dcs.bbk.ac.uk/
~dionisis/ns/employee" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dcs.bbk.ac.uk/
~dionisis/ns/employee employee.xsd"
    type="permanent">
    <firstname>John</firstname>
    <surname>Doe</surname>
</employee>
```

Filename: Employee.xml

Alternative XML Schema using XML namespaces

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="http://
www.dcs.bbk.ac.uk/~dionisis/ns/employee" xmlns="http://
www.dcs.bbk.ac.uk/~dionisis/ns/employee">
  <xs:element name="employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string" />
        <xs:element name="surname" type="xs:string" />
      </xs:sequence>
      <xs:attribute name="type" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Filename: Employee-alternative.xsd

XML Schema example with XML namespaces and references

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="http://
www.dcs.bbk.ac.uk/~dionisis/ns/employee" xmlns="http://
www.dcs.bbk.ac.uk/~dionisis/ns/employee">
    <xs:element name="employee">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="firstname" />
                <xs:element ref="surname" />
            </xs:sequence>
            <xs:attribute name="type" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <xs:element name="firstname" type="xs:string" />
    <xs:element name="surname" type="xs:string" />
</xs:schema>
```

Filename: Employee-full.xsd

Hands on exercise

- **Now do hands on exercise 11**
- Write in a comment problems that you might encounter making an XML Schema that corresponds to the DTD you already created.

Repeating and optional elements

- Until now we implicitly specified that the elements should appear once in our XML document.

<element name=”firstname” />

is equivalent to

<element name=“firstname” minOccurs=“1” maxOccurs=“1” />

- Setting minOccurs to zero makes the element optional.
- Giving maxOccurs the value *unbounded* allows the element to be repeated.

Repeating and optional elements

Occurrence requirement	minOccurs	maxOccurs	DTD equivalent
required (the default)	1	1	firstname
optional	0	1	firstname?
optional and repeatable	0	unbounded	name*
required and repeatable	1	unbounded	name+

```
<xs:sequence>
   <xs:element ref="firstname” minOccurs=“0” maxOccurs=“1”/>
   <xs:element ref=“middlename” minOccurs=“0” maxOccurs=“unbounded”
   <xs:element ref="surname" />
</xs:sequence>
```

Valid XML:

```
<employee><firstname>…</firstname><surname>…</surname></employee>
    <!-- or -->
<employee>
    <middlename>…</middlename>   <middlename>…</middlename>
    <surname>…</surname>
</employee>
```

Choices

- Sometimes a choice of sub-elements is needed.
  - e.g. an employee could be represented as an individual with firstname and lastname or as a company with only a company name
- This is represented by the **Choice** element (instead of the Sequence element)
- This is equivalent to a DTD declaration

```
<!ELEMENT employee (personal | company)>
```

- and would validate the XML

```
<employee><personal>…</personal></employee>
     <!-- or -->
<employee><company>…</company></employee>
```

Choices example

```
<xs:element name=“employee”>
  <xs:complexType>
    <xs:choice>
      <xs:element ref=“personal” />
      <xs:element ref=“company” />
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Filename: choicesExample.xsd