Birkbeck
UNIVERSITY OF LONDON

XML

Document Type Definition (DTD)
and introduction to XML Schemas

Entities

- General Entities
  - Can specify abbreviations, short-cuts or aliases
    - <!ENTITY defaultTitle “Photo Album”>
    - <!ENTITY currency “#x0024”>
    - <!ENTITY nbsp “#x00A0”>
    - <text>&defaultTitle;</text>
    - <price>&currency;34,000,000</price>

Example DTD

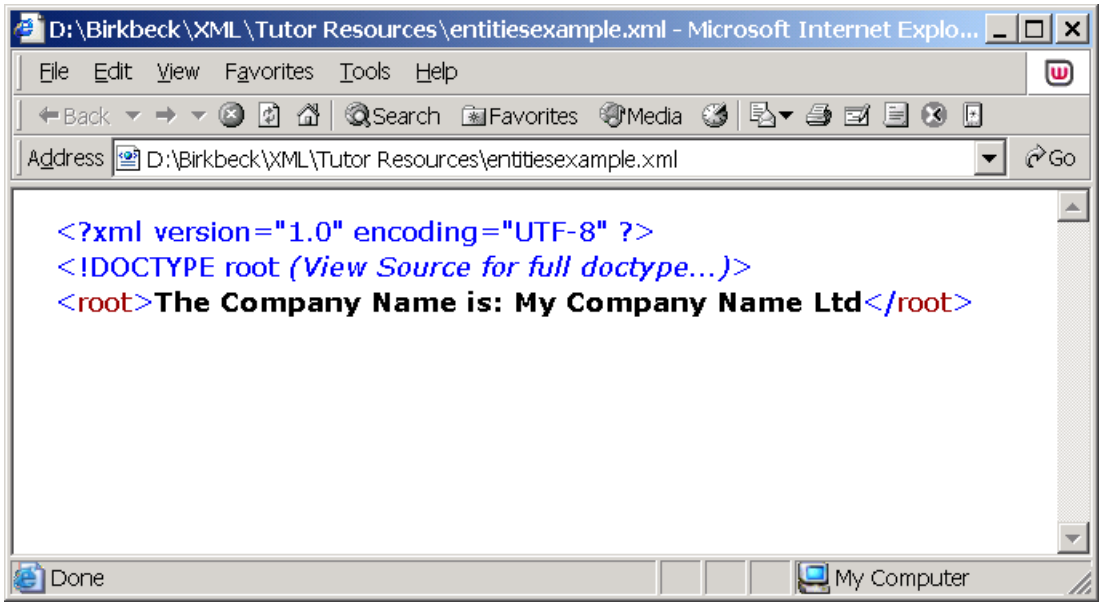
```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT root (#PCDATA)>
<!ENTITY companyname "My Company Name Ltd">
```

Example XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root SYSTEM "entitiesexample.dtd">
<root>The Company Name is:  &companyname;</root>
```

#x0024 and #x00A0 are hexadecimal Unicode values. You can find charts listing the numeric values that correspond to different characters at http://www.unicode.org/charts/

Example of the output using Microsoft Internet Explorer



Please note that the latest browsers have the capability of displaying entities **disabled** for security reasons.

- Parameter Entities
  - Shortcut for repeating syntax within a DTD

Example of an Employee DTD:

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

Filename: Employee.dtd

Let's now create an in-line DTD that uses the above DTD as a parameter entity.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE payroll [
   <!ENTITY % employeedtd SYSTEM "Employee.dtd">
   %employeedtd;
   <!ELEMENT payroll (employee, salary, mailingaddress)>
   <!ELEMENT salary (#PCDATA)>
   <!ELEMENT mailingaddress (employee, address)>
   <!ELEMENT address (#PCDATA)>
]>
<payroll>
   <employee>
      <firstname>Mark</firstname>
      <surname>Collins</surname>
   </employee>
   <salary>£35,000.00</salary>
   <mailingaddress>
      <employee>
         <firstname>Mark</firstname>
         <surname>Collins</surname>
      </employee>
      <address>34, Narrow Lane, SE3 6DY, London</address>
   </mailingaddress>
</payroll>
```

Filename: Payroll.xml

- To link an XML document to a DTD
  - It can be embedded (as in the previous slide).
  - Or the XML document can specify the DTD location as in the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE customerorder SYSTEM “CustomerOrders.dtd">
```

  - The location is usually a URL, local or another domain, such as the following example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Note: The above declaration is the doctype used to create XHTML 1.0 Strict web pages.

Validating a DTD

- A DTD is used to validate an XML document but DTDs themselves can be validated.
- You can validate a DTD by using third-party validation tools such Netbeans, Eclipse, jEdit, Oxygen XML etc.
- Netbeans is already installed on the lab computers and it is also free to download and install at home for all operating systems.
  - Alternatively, you can use Oxygen XML, which is also installed on the lab computers.

Validating a DTD example

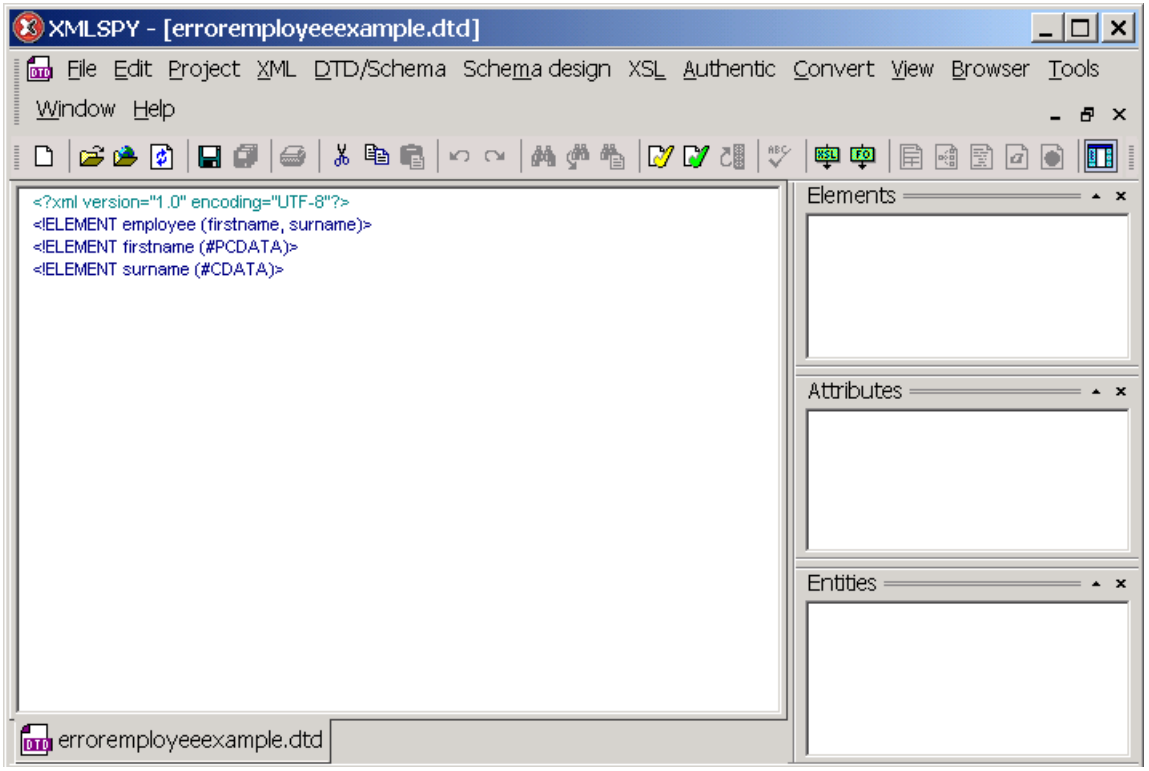
- Can you spot the error in the code? Let's open the following DTD in XMLSpy:

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

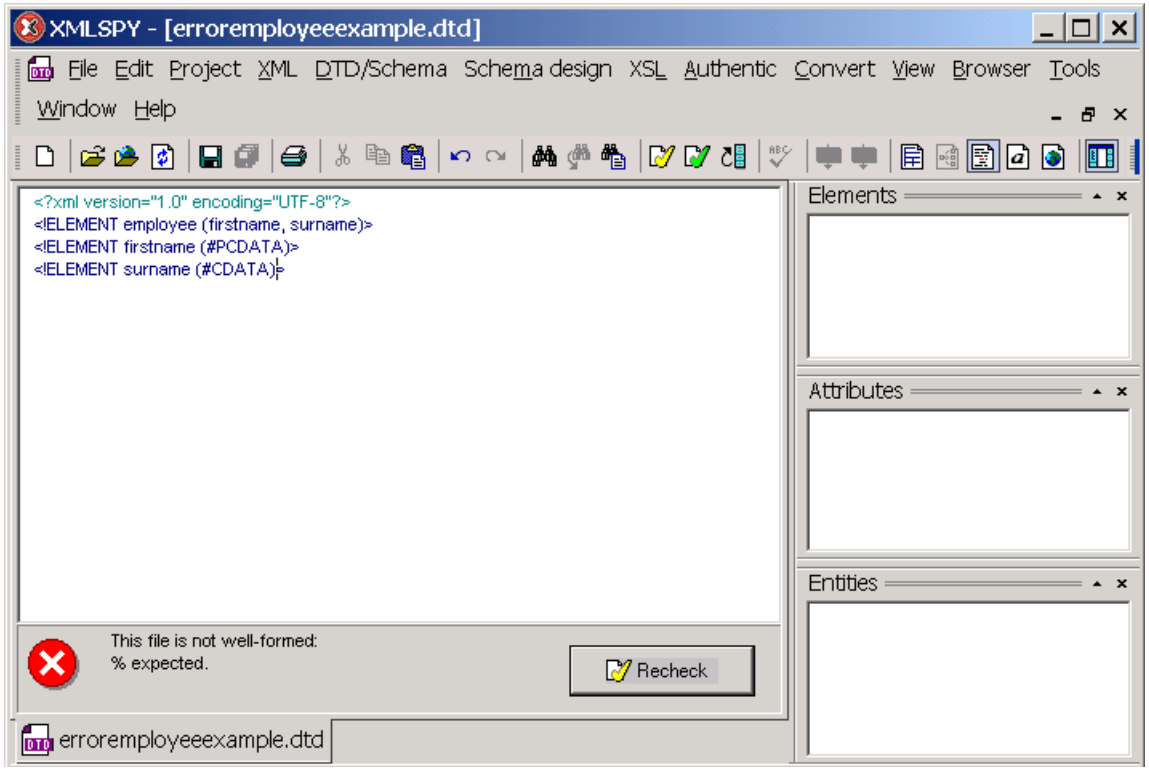
Filename: DTDErrorEmployeeExample.dtd

Open it in your DTD validator.

- Open the DTD in the previous example with XMLSpy as illustrated below:

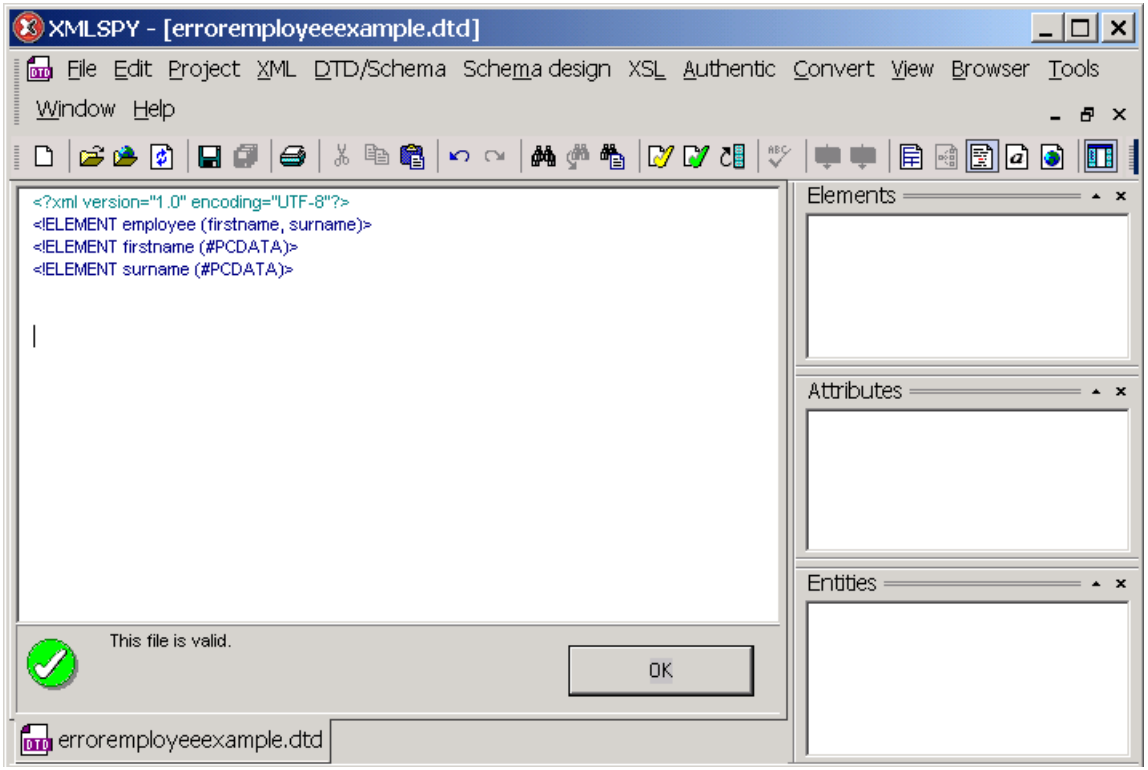


- Click on the green tick button in order to validate the DTD and the following screen should appear:



Note that XMLSpy first checks that the file is well-formed and then performs the validation check.

- Rectify the error in the DTD and click on the Recheck tick button and you should see the following screen:



You can now use your DTD in order to validate your XML file.

- Limitations of DTD's:
  - Weakly-typed data.
  - The values of the data contained in elements can be anything. Hence, the application has to validate the data.
  - A DTD does not make clear what is the root element of the associated XML. The root element is only specified in XML document
  - DTD's are written in a different syntax than XML and thus can not be parsed with an XML parser.
  - All declarations in a DTD are global which means that two elements with the same name can not be defined in different contexts. This presents integration problems when using several different third-party DTDs.

XML Schemas an XML based alternative to DTDs

- Brief overview of xml schemas:
  XML Schema was developed by W3C and attempts to address the limitations of DTDs.
  - XML Schemas are written in XML.
  - You can define global and local elements.
  - XML Schemas have data types such as: string, integer, decimal (with precision), boolean, date, etc…

XML Namespaces

- XML Namespaces were created to solve two problems:
  1. Context identification: The first problem is to identify which DTD or XML Schema each element belongs to.
  2. Duplication conflicts: The second problem is to avoid duplication of element and attribute names, as there is nothing to prevent different models defining objects with the same names. XML does not allow to elements or attributes to have the same name, as it would be impossible to identify which is indicated when one of them is used in a document.
- Reasons for using namespaces:
  - Distinguish between elements and attributes from different applications with the same name.
  - Group all related elements and attributes from one application to recognise them easier.
  - One of the main feature of XML is to combine markup from various XML applications. Hence, XML Namespaces make it easier for software to understand the nature of the markup been used.

XML Namespace Example

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://somedomain.com">
<body xmlns:html="http://www.w3.org/1999/xhtml">
    <p>
        <html:p>
            This is <html:em>content</html:em>and <b>so is this</b>.
        </html:p>
    </p>
</body>
</root>
```

Filename: NamespaceExample.xml

In the above example, the default namespace is declared in the root element and applies to the whole XML document.

However, the body declares another namespace with a prefix (html) which can be used throughout the document to indicate that individual elements belong to the html namespace.

Notice the presence of two paragraph elements: and <html:p></html:p>.

XML Namespace Example 2

```
<competitionEntry>
    <competition>Piano</competition>
    <competitor>J Smith</competitor>
    <score>57</score>
    <composer>George Gershwin</composer>
    <composition>Rhapsody in Blue</composition>
    <score>Ferde Grofe</score>
</competitionEntry>
```

NamespaceMusicExample_invalid.xml

Consider the possibility of a DTD describing musical compositions and includes elements that identify the composer, the title of the composition and the name of the person who provided a particular score (arrangement).

Another DTD describes the performance of a musician at a competition, and includes the score that the individual obtained for the performance.

In this scenario both DTDs have an element named **score**. This is a side effect of the first problem set to be solved by namespaces (identification of context). While it is easy in this example for people to distinguish the origin by reading the content, it is far less easy for software to make such distinction.

XML Namespace Example 2 corrected

```
<competitionEntry xmlns:M="…">
   <competition>Piano</competition>
   <competitor>J Smith</competitor>
   <score>57</score>
   <M:composer>George Gershwin</M:composer>
   <M:composition>Rhapsody in Blue</M:composition>
   <M:score>Ferde Grofe</M:score>
</competitionEntry>
```

NamespaceMusicExample.xml

Points to consider when creating a namespace

- Namespaces do not refer to an exact location.
- Designate a unique URI that determines the scope of the markup.

  Example:

  http://www.somedomain.com/namespace

- Specify a namespace reference in the parent node of the markup or root node of the XML document.

Points to consider when creating a namespace

- Declare a prefix to use for relevant markup elements.
  - This is done by giving a *suffix* to the xmlns attribute.
- Add the prefix to element names to ensure the distinction between elements used in different contexts.
- Be aware that a prefix can not start with x, m, and l (xml) in that order, in upper or lowercase.
- The XML processor considers elements without a prefix to belong to the default namespace if there is one.
- You can add version information to your namespace.
  Example:

  http://www.somedomain.com/namespace/version-2.0

Namespace URIs

- Some namespace URIs are standard, and may be recognised by some browsers
  Example:

  http://www.w3.org/1999/xhtml
  http://www.w3.org/1999/XSL/Transform
  http://www.w3.org/2001/svg
  http://www.w3.org/2001/MathML

- Some prefixes are used by convention
  Example:

  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:svg="http://www.w3.org/2001/svg"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"

  However, prefixes can be user-defined.

  Hence, you can change it but make sure it is meaningful to both a human reader and an XML parser.