Lecture 25: DTDs
XML applications

XML is derived as a stricter version of SGML.

XML and SGML are both meta-markup languages. That is they are not markup languages, but a description of how to write a markup language.

When you use the XML rules to write your own tag set you create your own markup language, also called XML application.
Validating XML documents

XML documents can be validated by a validator that does not understand your tag set. The validator will simply check that your document confirms to the basic XML rules.

Such a document is called well-formed.

It is possible to create your own validation rules by using either DTDs or schemas.

If an XML document validates against a DTD or schema, it is called valid.
XML DTDs

DTD stands for Document Type Definition

One of two ways to state the rules for the structure of an XML document (The other is schemas)

An XML document may carry with it a DTD

DTD defines XML documents
  • vocabulary: element and attribute names.
  • grammar: rules for possible arrangements of elements in terms of order, frequency, and nesting.

An XML document that conforms to these rules is said to be valid. A document that is valid is automatically well-formed. Note that a well-formed document need not be valid.
NOTE

DTDs are not themselves XML documents!

DTD expresses the set of rules for the document structure using an EBFN (Extended Backus-Nauer Form) grammar.

EBFN grammars are commonly used to define programming languages. We will not study EBFN grammar in any detail. Refer to Wikipedia if interested.
Why Use a DTD?

With a DTD, each of your XML files can carry a description of its own format.

With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.

Your programs can use a standard DTD to verify that the data you receive from the outside world is valid.

To create your own XML application you need to write a DTD or a schema.
DTDs continued

DTD's can be either

- internal (inside the XML document)
- external (in a separate .dtd text file)

XML documents that use a DTD need a SGML **DOCTYPE declaration** immediately after the XML declaration.

Example:
```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```
Internal DTD DOCTYPE Declaration

<?xml version="1.0" encoding="us-ascii"?>

<!DOCTYPE root_name[

<!--The DTD declarations replace this comment -->

]>

<!--Rest of XML document -->
Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE employees []
<!ELEMENT employees (employee+)>
<!ELEMENT employee (name, flag, species, job)>
<!ATTLIST employee rank CDATA #IMPLIED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT flag EMPTY>
<!ATTLIST flag gender (M|F) "M">
<!ELEMENT species (#PCDATA)>
<!ELEMENT job (#PCDATA)>
>
<employees>
  <employee rank="lowest">
    <name>Philip J. Fry</name>
    <flag gender="M"/>
    <species>Human</species>
    <job>Delivery boy</job>
  </employee>
</employees>
```
External DTD DOCTYPE Declaration (Private DTD)

Syntax:

```xml
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE root_name SYSTEM "dtd_filename">

<!--Rest of XML document -->
```
Example

```xml
<?xml version "1.0" encoding="us-ascii"?>

<!DOCTYPE employees SYSTEM "employees.dtd">

<employees>
    <employee rank="lowest">
        <name>Philip J. Fry</name>
        <flag gender="M"/>
        <species>Human</species>
        <job>Delivery boy</job>
    </employee>
</employees>
```
<?xml version "1.0" encoding="us-ascii"?>
<!DOCTYPE root_name PUBLIC identifier URL>

Example:
<?xml version "1.0" encoding="us-ascii"?>

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

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World Example</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
All modern browsers have a built-in XML parser.

An XML parser converts an XML document into an XML DOM object - which can then be manipulated with JavaScript.
Two important data types in DTDs

**PCDATA**
PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.

Tags inside the text will be treated as markup and entities will be expanded.

However, parsed character data should not contain any &, <, or > characters. These need to be represented by the &amp; &lt; and &gt; entities, respectively.
Two important data types in DTDs

CDATA
CDATA means character data.

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.
DTD Element Declarations

Syntax for simple elements:

```xml
<!ELEMENT element-name (category)>
```

The category can be
- `#PCDATA` parsed character data.
- `#CDATA` parsed character data that will not be parsed.
- `EMPTY` no content
- `ANY` Elements declared with the category keyword `ANY`, can contain any combination of parsable non-parsable data or be empty.
Elements with Children

Elements with one or more children are declared with the name of the children elements inside parentheses.

```xml
<!ELEMENT element-name (child1)>
```
or

```xml
<!ELEMENT element-name (child1,child2,...)>
```

Example:
```xml
<!ELEMENT employee(name, flag, species, job)>
```
Example XML document

```xml
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE employees SYSTEM "employees.dtd">
<employees>
  <employee rank="lowest">
    <name>Philip J. Fry</name>
    <flag gender="M"/>
    <species>Human</species>
    <job>Delivery boy</job>
  </employee>
  <employee rank="low">
    <name>Turanga Leela</name>
    <flag gender="F"/>
    <species>Mutant</species>
    <job>Captain</job>
  </employee>
</employees>
```
Corresponding DTD

<!-- File employees.dtd -->
<!-- DTD for employees.xml -->

<!ELEMENT employees (employee+)>
<!ELEMENT employee (name, flag, species, job)>
<!ATTLIST employee rank CDATA #IMPLIED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT flag EMPTY>
<!ATTLIST flag gender (M|F) "M">
<!ELEMENT species (#PCDATA)>
<!ELEMENT job (#PCDATA)>
Detailed run through of the DTD

In the XML file:

```
<!DOCTYPE employees SYSTEM "employees.dtd">
```

Contains the name of the root element employees. The keyword SYSTEM states that the DTD is contained in an external file called employees.dtd.

In the file `employees.dtd`:

ELEMENT declarations define the rules for the elements.

```
<!ELEMENT employees (employee+)>
```

- Root tag employees consists of one or more employee elements
- The occurrence indicator + dictates that employees must have at least one occurrence of a employee element.
Detailed run through of the DTD

<!ELEMENT employee(name, flag, species, job)>

-employee element consists of name, flag, species and job elements, each occurring once.

<!ATTLIST employee rank CDATA #IMPLIED>

-Defines employee elements attribute called rank
-CDATA indicates that the rank contains character data
-The keyword #IMPLIED means that no error is produced if the attribute is missing.
Detailed run through of the DTD

<!ELEMENT name (#PCDATA)>
-Element name content is parsed character data

<!ELEMENT flag EMPTY>
-Defines the empty element flag

<!ATTLIST flag gender (M|F) "M">
-Defines the attribute gender for element flag
-Gender is specified to be either M or F
-If no attribute is present M is chosen by default.
Content models

Single child elements (precisely one)
<!ELEMENT element-name (child-name)>

Example:
<!ELEMENT race (time)>

Time must occur once, and only once.

Minimum One Occurrence of an Element
<!ELEMENT element-name (child-name+)>

Example:
<!ELEMENT employees(employee+)>

employees must have at least one occurrence of employee.
Content models

Zero or More Occurrences of an Element

<!ELEMENT element-name (child-name*)>

Example:
<!ELEMENT currency (country*)>

country may or may not occur as a sub-element of currency.

Zero or One Occurrences of an Element

<!ELEMENT element-name (child-name?)>

Example:
<!ELEMENT phone (cell_phone?)>

cell_phone may appear at most once as a sub-element of phone.
Content models

Sequences (precisely one of each in order)

```xml
<!ELEMENT element-name (child1-name, child2-name, ..., childN-name)>
```

Example:

```xml
<!ELEMENT date (month, day, yr)>
```

date must contain month, day and a yr exactly in that order.

Either/or Content

```xml
<!ELEMENT element-name (child-name1|child-name2)>
```

Example:

```xml
<!ELEMENT contact_info (name, address, (home_phone|cell_phone))>
```

The example above declares that the contact_info element must contain a name element, an address element, and either a home_phone or a cell_phone element.
Content models

List of options (precisely one option)

<!ELEMENT element-name(option-1-|option-2|... |option-n)>

Example:
<!ELEMENT payment(check|credit|cash)>

Exactly one of check, credit, cash can appear as a child of payment.
Content models

Combining sequences and options

**Sequence of options**
Example:

```xml
<!ELEMENT date ((dinner|cocktail)?,(movie|minigolf))>
```

**Option of sequences**
Example:

```xml
<!ELEMENT date ((dinner, movie) | (cocktail, dinner))>
```
Content models

Mixed content

- You must use | and *
- You cannot specify order or number of children.

Example:

```xml
<!ELEMENT payment (#PCDATA|credit)*>
```

payment may contain 0 or more occurrences of parsed character data and 0 or more occurrences of credit element in any order.
Example with Mixed Content

```xml
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE payment [  
<!ELEMENT payment (#PCDATA|cash|credit)*>  
<!ELEMENT cash (#PCDATA)>  
<!ELEMENT credit (#PCDATA)>  
]>  
<payment>
A combination of
<cash>$5.00</cash> and
<credit>$17.00</credit>
</payment>
```
Some examples:

```xml
<!ELEMENT fraction (numerator, denominator)>
fraction has numerator followed by denominator.

<!ELEMENT list (ListItem)+>
list contains at least one ListItem.

<!ELEMENT table (TableCaption?, TableRow+)>
table contains optional TableCaption (it can be omitted) followed by at least one TableRow.

<!ELEMENT matrix (MatrixRow, MatrixRow+)>
matrix has at least two MatrixRows.

<!ELEMENT parent (FirstChild, (AnotherChild|YetAnotherChild)+, LastChild)>
parent must contain FirstChild followed by either AnotherChild or YetAnotherChild one or more times and end with LastChild.

<!ELEMENT shortList (item, item, item?, item?, item?, item?, item?, item?, item?)>
shortList contains at least two and at most nine items.
Attributes

Syntax:
<!ATTLIST element-name attribute-name attribute-type default-value>

DTD example:
<!ATTLIST payment method CDATA #FIXED "check">

XML example:
<payment method="check"/>

For us attribute-type is always CDATA.

default-value is one of: value, #FIXED value, #REQUIRED or #IMPLIED.
Syntax:
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">

Example:
<!ATTLIST file_name file_type CDATA #FIXED "XHTML">
Valid XML:
<file_name file_type="XHTML">hello_world.html</file_name>
Invalid XML:
<file_name file_type="css">hello_world.html</file_name>

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.
### #REQUIRED

**Syntax:**

```xml
<!ATTLIST element-name attribute-name attribute-type #REQUIRED>
```

**Example:**

```xml
<!ATTLIST person number CDATA #REQUIRED>
```

**Valid XML:**

```xml
<person number="5677" />
```

**Invalid XML:**

```xml
<person />
```

Use when you require an attribute, but you allow it to have any value.
#IMPLIED

Syntax:
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

Example:
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:
<contact fax="555-667788" />

Valid XML:
<contact />

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.
Default specified

<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:

<square width="100" />

In the example above, the square element is defined to be an empty element with a width attribute of type CDATA. If no width is specified, it has a default value of 0.
Enumerated attribute values

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

<!ATTLIST element-name attribute-name (en1|en2|..) default-value>

DTD example:
<!ATTLIST payment method (check|cash) "cash" >

XML example:
<payment method="check" />

or

<payment method="cash" />
There are two types of entities: **General entities** and **parameter entities**.

**General entities**
Are defined in the DTD and can be referenced in the XML document.

**Syntax:**
```
<!ENTITY e_name "e_value">
```

**Example:**
```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
<!ELEMENT author (#PCDATA)>
<!ENTITY jv "Jukka Virtanen">
]>
<author>&jv;</author>
```
Parameter entities

Parameter entities are referenced in the DTD itself.

Syntax:
<!ENTITY % e_name "e_value">

Example:
<!ENTITY % p "(#PCDATA)">
<!ELEMENT student (id,surname,firstname,dob,(subject)*)>
<!ELEMENT id %p;>
<!ELEMENT surname %p;>
<!ELEMENT firstname %p;>
<!ELEMENT dob %p;>
<!ELEMENT subject %p;>
See examples for bruinml.dtd and corresponding xml file