Theme 2: DTD

Attributes and entity

Defining Attributes

Defining Attributes

- You must define all possible attributes in the DTD.

- The syntax of an **attribute rule**:

```
<!ATTLIST tagName attName type status>
```

- The **tagName** is the element whose attribute this is.
  - This element must have an element rule in the DTD.

Defining Attributes

- The **type** is the type of attribute.
    - Replace it with **CDATA** if it must have a normal text value.
    - We’ll look at other types later.
    - Don’t confuse **CDATA** with **#PCDATA**!

- The **status** indicates if the attribute is optional/required…
- …or has a default/fixed value.

Defining Multiple Attributes

- If **one** element has multiple attributes...
- ...you can combine them into **one** attribute rule.

- An attribute rule with **two** attributes for **one** element:

```
<!ATTLIST tagName
          att1Name type status
          att2Name type status>
```

(White space is ignored, so you may indent or write it on one line.)

Attributes Occurrences

Status

Defining Attributes – Occurrences

- If the XML author may **leave out** the attribute...
- ...replace **status** with **#IMPLIED** (case matters!).

- If the XML author **must** add the attribute...
- ...replace **status** with **#REQUIRED**.

Defining Attributes – Occurrences

- If the XML author may **leave out** the attribute...
- ...and **if so**, the processor adds it with a **default** value...
- ...replace **status** with **“value”**.

  - Replace **value** with whatever the default value must be.

Defining Attributes – Occurrences

- If the XML author may **leave out** the attribute...
- ...and **if so**, the processor adds it with a certain value...
- ...but **if not**, the author **must** give it that same value...
- ...replace **status** with **#FIXED “value”**.
    - Replace **value** with whatever the fixed value must be.

Defining Attributes – Occurrences

- The four possible replacements for **status** are thus:
  - **#IMPLIED**
  - **#REQUIRED**
  - **“value”** (replace **value** with the value you want)
  - or **#FIXED “value”** (replace **value** with the value you want)

- You may pick only **one** of these.

- Every attribute rule **must** have a **status**.

Attributes Data

Types

Attributes with Choices

- For a normal text attribute, replace **type** with **CDATA**.

```
<!ATTLIST tagName attName CDATA status>
```

- You can also define an attribute that must have...
- ...one of several pre-defined **choices** as its value. E.g.:

```
<!ATTLIST time unit(minutes | hours) status>
```

Attributes with Choices

- Replace each choice (e.g. **choice1**) with an XML name.
  - An **XML name** follows XML naming rules (e.g. no spaces).

- **Don’t** enclose the choices in quotation marks in the DTD!

- An XML author may now use any of the choice values...
- ...as the value of that attribute. Nothing else!

Attributes with Unique Values

- You can define an attribute to have a **unique value**.
- These kind of attributes are called **ID attributes**.
- No two ID attributes may have the same value in the XML.
  - Even if the attributes have different names…
  - ...or are in different elements.

Attributes with Unique Values

- How to define an ID attribute:

```
<!ATTLIST name studentNumber ID status>
```

- The value of an ID attribute **must** be an XML name.

- One element **may not** have more than one ID attribute.

- Its status **must** be **#IMPLIED** or **#REQUIRED**.

Referencing ID Attributes

- You can define an attribute that references an ID attribute:

```
<!ATTLIST studentcard id IDREF status>
```

- The value of the attribute **must** be equal to...
- ...the value of another ID attribute in the XML.

- Different IDREF attributes may refer to the same ID value.

Referencing ID Attributes

- If you use **IDREFS** instead of **IDREF**…
- …the attribute can contain a white-space separated list…
- …of ID attribute values in the XML doc.

- E.g. if one ID attribute = **“c1”** and another = **“c2”**…
- …an IDREFS attribute’s value may be: **“c1 c2”**.

XML Name Attributes

- Name can have restriction on the initial character…
- … **NMTOKEN** value does not have these restrictions…
- …but your value still must be a valid XML name.

```
<!ATTLIST tagName attName NMTOKEN status>
```

XML Name Attributes

- If you use **NMTOKENS** instead of **NMTOKEN**...
- ...the attribute may contain a white-space separated list...
- ...of valid XML names.

Attribute Types

- In short, you can replace **type** in an attribute rule with:
  - **CDATA**
  - (**choice1** | **choice2** | etc.)
  - **ID**
  - **IDREF** or **IDREFS**
  - **NMTOKEN** or **NMTOKENS**

- You **must** specify a **type** for every attribute in the DTD.

Defining Entities

Entities

- An **entity** in XML is like a **constant** in Java/C++/etc.

- In a DTD, you define the entity’s name...
- ...and give it a text value.

- When the XML references the entity in the DTD...
- ...the processor replaces the reference with its value.

Entities

- **There are two main types of entities:**
  - **General** entities, and
  - **Parameter** entities.

- **The main difference:**
  - General entities are referenced in the XML, while...
  - Parameter entities are referenced in the DTD.

General Entities

- **General entities can be:**
  - Internal or external, and
  - Parsed (read by the processor) or unparsed (ignored).

- **Parameter entities can be:**
  - Internal or external, but
  - They are always parsed.

General Entities

General Entities

- **How to define an internal general entity in the DTD:**

```
<!ENTITY ent_name “content”>
```

- Replace **ent_name** with a name for the entity.
    - It must be a **valid XML name** (e.g. no spaces).

- Replace **content** with text you want to reuse in the XML.

General Entities

- **Reminder:**
    - The XML language has five built-in general entities.
    - You don’t need to define them in the DTD.
    - They are **amp** (&), **lt** (<), **gt** (>), **quot** (“), and **apos** (‘).
    - Any other entity **must** be defined in the DTD before using it.

General Entities

- **How to use (reference) the entity in the XML:**

```
&ent_name;
```

- You can use this inside any text data of the XML doc.

- The processor replaces it with the **content** you defined for the entity, **ent_name**, in the DTD.

- The ampersand (&) and semicolon (;) are required!

General Entities

- The **content** of one entity can reference another entity.

- E.g., in the DTD:

```
<!ENTITY ent1_name “content”>
<!ENTITY ent2_name “more &ent1_name;”>
```

- You **cannot** reference an entity within its own **content**.

General Entities

- **How to create and define an external general entity:**
  - Create a new text file with extension, **.ent**.
  - Put the **content** of the entity inside that text file.
  - In the DTD:

```
<!ENTITY ent_name SYSTEM “filename.ent”>
```

General Entities

- Referring to an external general entity in the XML...
- ...works the same as for internal entities. However...
- ...add **standalone="no"** to the XML declaration.

Entities for Unparsed Content

Notations

Entities for Unparsed Content

- **Unparsed content:**
  - Any content that the XML processor must ignore.
  - Usually non-text data like images, videos, a PDF, etc.
- You can embed this content into an XML doc...
- ...by defining an entity for it in the DTD.

Entities for Unparsed Content

- First, the entity must know how to interpret the content.
- Do this with a **notation** with processing instructions.
- In the DTD:

```
<!NOTATION n_name SYSTEM “notation.instr”>
```

Entities for Unparsed Content

- Replace **n_name** with a description of the type of content.
  - E.g., for a JPEG image, it could be **jpg**.

- **notation.instr** will be some file or instruction that explains how to process this content.
  - E.g. a MIME type, a URI to an application, etc.

  - The format of this depends on the processor.

Entities for Unparsed Content

- After defining the notation, define the entity:

```
<!ENTITY ent_name SYSTEM “entity.uri” NDATA n_name>
```

- For example, to define an entity for a JPEG image:

```
<!NOTATION jpg SYSTEM “image/jpeg”>
<!ENTITY chicken_pic SYSTEM “chicken.jpg” NDATA jpg>
```

Entities for Unparsed Content

- You **don’t** reference unparsed content entities in XML...
- ...in the same way you reference parsed content entities.

- You need to define an **ENTITY** type attribute in the DTD...
- ...that references the unparsed content entity in the XML.

Entities for Unparsed Content

- E.g. in the DTD, after defining the notation and entity:

```
<!ELEMENT photo EMPTY>
<!ATTLIST photo source ENTITY #REQUIRED>
```

- Then, in the XML, you can do this:

```
<photo source=“chicken_pic” />
```

Entities for Unparsed Content

- **Note:**
  - You can use **ENTITIES** instead of **ENTITY** in the ATTLIST...
  - ...for a white-space separated list of entity references.

  - The processor is supposed to view/display/run the unparsed content using the notation info...
  - ...but current browser-based processors can’t do it yet.

Parameter Entities

Parameter Entities

- **Parameter entities:**
  - Entities referenced in the DTD itself, not in the XML.

- **Defining a parameter entity in the DTD:**

```
<!ENTITY % ent_name “content”>
```

Parameter Entities

- **How to reference a parameter entity in the DTD:**

```
%ent_name;
```

- The order of rules in the DTD normally doesn’t matter...
- ...but in this case you must **define** a parameter entity...
- ...before any rules that **reference** that entity.

Parameter Entities

- **An example using a parameter entity in the DTD:**

```
<!ENTITY % p “(#PCDATA)”>
<!ELEMENT theRoot (aChild, anotherChild)>
<!ELEMENT aChild %p;>
<!ELEMENT anotherChild %p;>
```

- You can’t reference a parameter entity in the **XML**.

Parameter Entities

- **How to define an external parameter entity:**

```
<!ENTITY % ent_name SYSTEM “entity.ent”>
```

- “entity.ent” is a text file with extension ENT...
- ...that contains the **content** of the parameter entity.

- Referencing it is the same as for internal entities.

DTD vs XML Schema

DTD over Schema

- A "DTD" is a schema doc written in the DTD language.

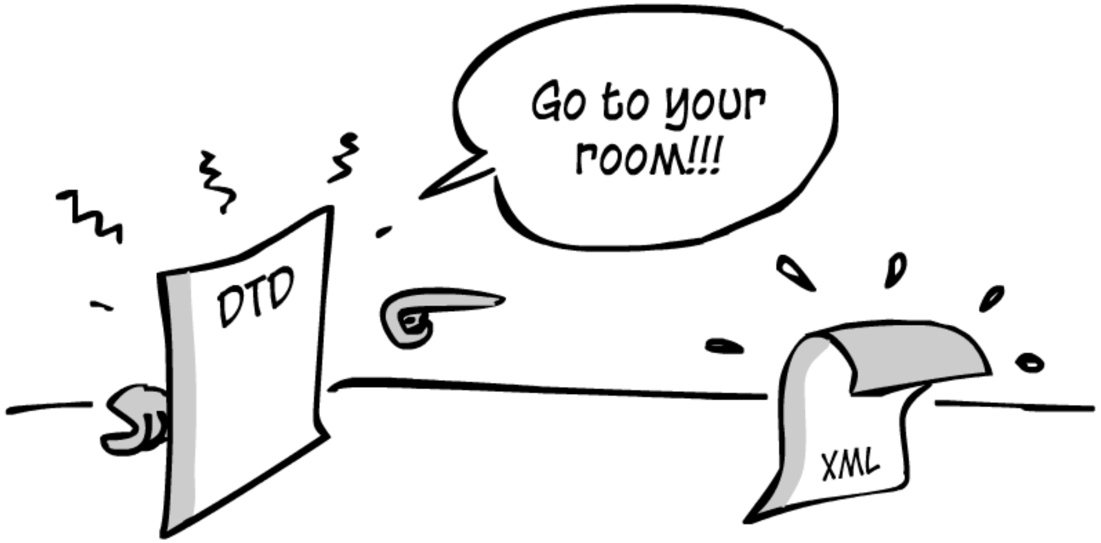
- **Pros of using DTD instead of XML Schema:**
  - DTD is more compact and easier to understand;
  - You can embed a DTD inside an XML doc;
  - You can define custom entities in a DTD;
  - It's supported by most XML parsers and more widely accepted.

DTD over Schema

- **Cons of using DTD instead of XML Schema:**
    - DTD is not an XML language, so you need extra support for it;
    - You cannot use namespaces when using DTD;
    - You cannot enforce specific data types on XML values;
    - You cannot declare a specific number of children for elements.

- Which language you use depends on your situation.

Theme 2: DTD



END OF THEME 2