Introduction to DTDs

June 24, 2003
What is a DTD?

- A DTD is a Document Type Definition
  - The DTD is written using a formal syntax
  - It explains which elements and entities may appear
  - It explains what the element’s content and attributes are
What a DTD is not!

- It does not tell you what the root element is
- Or how many instances of each element should exist
- Or what the character data inside an element looks like
- Or the semantic meaning of an element
Why Use DTDs?

- They’ve been around since SGML
- They are relatively simple to implement
- They provide a means to validate your document

Note: You MUST be using a validating parser if you wish your DTD to be used!
Simple XML Doc

<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
DTD Example for previous XML Doc

<!ELEMENT person (name, profession*)>
<!ELEMENT name (first_name, last_name)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
Q: How do you specify the DTD for an XML document?

A: Through the document type declaration!

```xml
<!DOCTYPE person SYSTEM 
 "http://www.cafeconleche.org/dtlds/person.dtd">
```

A document can only have one document type declaration--it must come after the xml declaration but before the root element.
Internal DTDs

- You can also declare your DTD internally to the document:

```xml
<?xml version="1.0"? standalone="yes">
<!DOCTYPE person [ 
  <!ELEMENT first_name (#PCDATA)>
  <!ELEMENT last_name (#PCDATA)>
  <!ELEMENT profession (#PCDATA)>
  <!ELEMENT name (first_name, last_name)>
  <!ELEMENT person (name, profession*)>
]> 
<person>…</person>
```
Combining Internal and External DTDs

- You can use both internal and external DTDs:

```xml
<!DOCTYPE person SYSTEM "name.dtd" [ 
  <!ELEMENT profession (#PCDATA)> 
  <!ELEMENT person (name, profession*)> ]>
```
Public DTDs

- Some DTDs are well-known and public. Services map names to DTDs, which can be cached locally. To declare that we are using a public DTD:

  ```xml
  <!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
   "http://my.netscape.com/publish/formats/rss-0.91.dtd">
  ```

- In practice, public DTDs are not used often and the backup URI is used to locate the DTD.
The XML declaration can declare the document as ‘standalone’
If you use external DTDs, you should set this value to ‘no’:
`<?xml version="1.0" standalone="no"?>`
Every element in an XML document must be declared in the DTD:

```xml
<!ELEMENT element_name content_specification>
```

*element_name* is the name (or type) of the element
The content specification can be:

- **#PCDATA**: states that the element only contains parsed character data (ie, no elements!)
  ```xml
  <!ELEMENT phone_number (#PCDATA)>
  ```

- Child elements: this specification lists the elements that the parent element can contain:
  ```xml
  <!ELEMENT fax (phone_number)>
  ```
Sequences

• We can specify a sequence of elements by using the `,`. For example:

```xml
<!ELEMENT name (first_name, last_name)>

<name>
    <last_name>Turing</last_name>
    <first_name>Alan</first_name>
</name>

is not valid!
Choices

- We can specify a choice between several elements using a ‘|’ symbol. For example:
  
  ```xml
  <!ELEMENT favorite_color (red | blue | green)>
  ```

- Thus, `<favorite_color>` can contain red, blue, or green sub-elements, but not more than one of them.
Suffixes used in DTDs

- We can gain more expressive power using the suffixes ‘*’, ‘+’, and ‘?’.
- The ‘*’ means “0 or more elements”.
- The ‘+’ means “1 or more elements”.
- The ‘?’ means 0 or 1 element.
Examples with Suffixes

- `<!ELEMENT person (name, profession*)>` has 0 or more profession elements.
- `<!ELEMENT person (name, profesion+)>` has at least one or more profession elements.
- `<!ELEMENT person (name, profession?)>` has 0 or 1 profession.
Parenthesis

- Parenthesis can be used in the content specification to increase expressiveness. For example:

  `<!ELEMENT circle (center, (radius | diameter))>`
Mixed Content

- How can we specify that an element contains both parsed character data and child elements?
- There is only one way! The format is:
  `<!ELEMENT definition (#PCDATA | term)*>`
- You can add as many ‘terms’ as you wish, but #PCDATA must come first and this specification cannot be part of a larger spec.
Empty Content

For elements which have no content, such as an `<image />` tag in XHTML, we can use the EMPTY declaration:

```xml
<!ELEMENT image EMPTY>
```

This definition also means that `<image></image>` is legal.
Unspecified Content

- The ‘catchall’ of content specification is the ANY specification:
  ```xml
  <!ELEMENT page ANY>
  ```
- Useful when developing DTDs and you’re not sure of the structure yet.
- Any content this element contains must still be declared in the DTD!
Attribute Declarations

- Besides elements, DTDs must declare the attributes of elements. These are specified through the attribute declaration:

```xml
<!ATTLIST element_name attribute_name type required/implied>
```
Example Declaration

<!ATTLIST image source CDATA #REQUIRED
width CDATA #REQUIRED
height CDATA #REQUIRED
alt CDATA #IMPLIED>

#REQUIRED means the attribute must be specified in the element
#IMPLIED means that it optionally exists
Attribute Types

- There are numerous attribute types:
  - CDATA
  - NM_TOKEN
  - NM_TOKENS
  - Enumeration
  - ENTITY
  - ENTITIES
  - ID
  - IDREF
  - IDREFS
  - NOTATION
Attribute type semantics:

- **CDATA**: any text that is acceptable in a well-formed attribute
- **NMTOKEN**: An XML Name Token: consists of all of the elements of an XML name, except there is no restriction on the start of the data (hence, “10” is a legal NMTOKEN, while it’s not a legal XML element name.
- **NMTOKENS**: A space separated list of NMTOKENs: “1.0 2.0 3.0”
- **Enumeration**: This is not actually a keyword, but a list of possible values for the attribute: `<!ATTLIST half-year (Jan | Feb | Mar | Apr | May | Jun ) #REQUIRED>`
ID Types

- ID: an attribute that must contain a valid XML name (not name token as previously described), and further, the name must be unique in the document.

```xml
<!ATTLIST employee ssn ID #REQUIRED>
<employee ssn="ss483-20-2989">
```
IDREF Type

- An IDREF type refers to an ID attribute of another element.

```xml
<!ATTLIST member person IDREF #REQUIRED>

<project project_id="p1">
  <member person="ss483-20-2989"/>
</project>
```
More types...

- IDREFS: a space separated list of IDREFs.
- ENTITY: the name of an unparsed entity declared somewhere else in the document (discussed later)
- ENTITIES: a space separated list of entities
- NOTATION: the name of a notation declared elsewhere in a document (discussed later)
Attribute Defaults

- #IMPLIED: the attribute may or may not appear
- #REQUIRED: the attribute must be in the element
- #FIXED: if the attribute appears, it must have a specific value
- *Literal*: The actual default is listed as a quoted string
Attribute Default Examples

- `<!ATTLIST room version CDATA #FIXED “1.0”>`
- `<!ATTLIST web_page protocol NMToken “http”>`
General Entity Declarations

- We can add new entities using an entity declaration:
  
  `<!ENTITY super “supercalifragilisticexpialidocious”>`

  The entity is used in the document like this:
  
  `<MaryPoppins>&super;</MaryPoppins>`

- Note: text in the quote must be well formed!
External Parsed General Entities

- We can also include ‘external’ entities, which an XML parser may include (and must if it is a validating parser). The format is:
  
  `<!ENTITY name SYSTEM “uri”>`

  Where name is the name of the entity, SYSTEM specifies that it’s external, and the quoted uri is a Universal Resource Identifier
External Unparsed Entities

- How can we refer to non-XML data? We use an external unparsed entity declaration.
- Both NOTATION and ENTITY declarations are needed for external unparsed entities.
- If you want to learn about External Unparsed Entities, refer to “XML in a Nutshell”, chapter 3.
- In conclusion: try to avoid attributes that refer to external unparsed entities and require ENTITY, ENTITIES, or NOTATION types.
In Conclusion

- DTDs can be used to help validate your XML document with a validating parser
- Try to make your document type definition flexible--too many requirements encourages users to cheat!
Exercise

• Come up with a DTD for your address book