Theme 2: DTD

Part 1: **Validation and Using DTD**

Schemas

- **Reminder:**
  - XML has no tags of its own.
  - The author designs a tag set, i.e. a **custom XML language**.
  - Others can then use the same tag set to describe similar data.
- When many authors use the same language, use a **schema** to keep all docs consistent.

Schemas

- A schema is a **rule set** governing a custom language.
- Adding a schema is **optional** but crucial for consistency.
- **The schema dictates:**
  - Which elements you may use and their relationships;
  - Which attributes you may use for each element;
  - And whether each element/attribute is required or optional.

Schemas

- Compare (validate) a doc against the language schema.
- If the doc adheres to the schema rules, it is **valid**.
- A valid doc is one that uses the custom language correctly.
- If all authors validate their docs against the schema...
- ...they can ensure that their docs are consistent.

Consistency in XML

- **Why is consistency important?** Because, for example:
  - If an app is designed to read docs of that custom language...
  - ...the app won't be able to read invalid docs.
- Create a schema using a **schema language**.
- The two main ones: **DTD** and **XML Schema**.

DTD

Document **T**ype **D**efinition

What is DTD

- **DTD** stands for **D**ocument **T**ype **D**efinition.
- A DTD doc is **not** an XML doc (unlike XML Schema docs).
- You need a **DTD validator** to validate XML against DTD.
    - It is a standalone app or part of another one.
    - Browsers cannot validate XML docs with DTD.

Validating with a DTD

- An **internal DTD** is written inside an XML doc.
- An **external DTD** is a text file with extension **.dtd**.
  - The XML doc must **declare** (link to) the DTD.
  - An external DTD can be **system-specific** or **public**.

- **System-specific:** the DTD file is on your system.

- **Public:** the file is on a public web server.

System-specific DTD

To declare a **system-specific** external DTD in the XML...

- ...add the following directly after the XML declaration:

```
<!DOCTYPE root SYSTEM "schema.dtd">
```

- Replace **root** with the name of the XML root element.

Public DTD

- Replace **schema.dtd** with the path to and name of the DTD file.
  - In this example, the DTD is in the same folder as the XML.

- To declare a **public** external DTD in the XML:

```
<!DOCTYPE root PUBLIC "FPI" "URL">
```

Public DTD

- Replace **root** with the XML root element's name.
- Replace **FPI** with the DTD's **F**ormal **P**ublic **I**dentifier.
- Replace **URL** with the web address to the DTD file.
- An **FPI** is a special name for a public DTD.
  - The parser uses it to locate the DTD on a public server. If that fails, it uses the URL instead.

Public DTD

- An FPI looks like this:

```
-//owner//DTD description//XX
```

- The minus (**-**) means the DTD is not a standard.

- Replace it with **+** if it's an approved non-ISO standard...
- ...or with **ISO** if it is an approved ISO standard.

Public DTD

- Replace **owner** with text that identifies…
- …the person/organisation responsible for the DTD.

- Replace **DTD description** with a description of the DTD.

- Replace **XX** with the spoken language used (e.g. English).

- E.g.:

```
-//W3C//DTD XHTML 1.0 Transitional//EN
```

Validating with a DTD

- When declaring an external DTD in the XML doc...
- ...add the following attribute to the XML declaration:

```
<?xml version="1.0" standalone="no" ?>
```

- This lets the XML processor know that the XML is dependent on an external file.

Internal DTDs

- An internal DTD is declared and created at the same time.
- Add the following directly after the XML declaration:

```
<!DOCTYPE root [
     <!-- DTD code -->
]>
```

- Once again, replace **root** with the root element’s name.

DTD Syntax

- We'll mostly deal with system-specific external DTDs.

- The first step in a DTD is to define...
- ...the **structure** and **content** of...
- ...the **elements** a valid XML doc would have.

- We need to create an **element rule** for each element.

Defining Elements

Element Rules

- An element rule in a DTD looks like this:

```
<!ELEMENT name content>
```

- Replace **name** with the element's name.
- Replace **content** with DTD code describing the content.
  - We'll look at describing different kinds of content.

Element Rules

- Every element that might appear in a valid XML doc...
- ...**must** have an element rule in the DTD. Children too!
- It doesn't matter in which order you write the rules.
- Write each rule on its own line.
- DTD syntax is **case-sensitive**!

Element Rules – Plain Text

- Most XML elements will contain only text.
- To define an element that may contain only text:

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

- PCDATA stands for **P**arsed **C**haracter **DATA**.
- An element defined like this **may not** have children.

Element Rules – Empty Element

- To define an empty element:

```
<!ELEMENT name EMPTY>
```

- In this case, **do not** add parentheses.

- An element defined like this **may not** have any content.
  - Though they may have attributes, discussed later.

Element Rules – Child Element

- To define an element with **one** child:

```
<!ELEMENT name (child)>
```

- Replace **child** with the name of the child element.
- This element may not contain anything except that child.

- You must write a separate element rule for the child!

Element Rules – Sequential

- To define an element with **two or more** children:

```
<!ELEMENT family (child, child, child1)>
```

- The children must appear **in this sequence** in the XML.
  - So, **child1** must be the first child, **child2** the second, etc.

- You may not use "(#PCDATA)" as part of the sequence.

Defining Occurrences

- Consider the following element rule:

```
<!ELEMENT name (first, middle, last)>
```

- This element must contain **one** of each child in that order.

- Use **quantifiers** (special symbols) to define...
- ...**how many** of each child must appear.

Defining Occurrences

- An asterisk (*) means **zero or more** times.

```
<!ELEMENT garden (flowers*, trees)>
```

- You can now skip **flowers**, or put it in once or more.

- Finish with **flowers** in the XML before adding **trees**.

Defining Occurrences

- A plus (+) means **one or more** times.

```
<!ELEMENT life (failure+, success)>
```

- Now **child1** may appear many times, but at least once.

Defining Occurrences

- A question mark (?) means **zero or one** time.

```
<!ELEMENT health (chickenpoxs?, flu)>
```

- Now you may skip **chickenpoxs**, or add it **once**.
- A child without a quantifier **must** appear **exactly once**.

Defining Occurrences

- You can also apply quantifiers to a sequence. E.g.:

```
<!ELEMENT binary (zero, one)+>
```

- This sequence may now appear one or more times.

- In the XML, create **zero** and **one**, then...
- ...you may create **zero** and **one** again, etc.

Element Rules – Choices

- You can define an element to contain a choice of children.

```
<!ELEMENT airplaneMeal (chicken| beef| pork)>
```

- It may contain either **chicken**, or **beef**, or **pork**.
- If you use one child, you **may not** include the others.
- You can add as many choices as you want.

Nesting Choices

- You can nest choices and sequences. E.g.:

```
<!ELEMENT Thursdays ((wine| whiskey), puke)>
```

- It must contain either **wine** or **whiskey**, then **puke**.

Choices Occurrences

- If you add an asterisk (*) to a list of choices:

```
<!ELEMENT Fridays (beer| cider)*>
```

- ...you can make the choice **zero or more** times.

- The element may contain nothing (zero choices), or...
- ...an unordered list of **beer** and **cider** elements.

Element Rules – Mixed Content

- **Mixed content** = a mix of child elements and text.
- To define an element with mixed content:

```
<!ELEMENT name (#PCDATA | fullname | lastname)*>
```

- You **must** put #PCDATA first in such a choice list, must have an **“asterisk**”
- The element may contain text, **fullname**, or **lastname** elements.

Element Rules – Any

- You can define an element to contain anything:

```
<!ELEMENT eggs ANY>
```

- This element may contain any text and/or elements.
- Use only when absolutely necessary!
  - The point of a schema is to set up more specific rules.

Element Rules – Any

- An **ANY** element may only contain elements...
- ...that have an element rule in the DTD.

- **A final word about element rules:**
  - Each element has **only one** element rule in the DTD...

  - ...even if the element appears in multiple places in the XML.

Theme 2: DTD

TO BE CONTINUED...