Theme 1: XML

WRITING XML

Writing XML

Reminder: An XML doc is...

- ...written in a custom markup language (tag set) that...
- ...follows the grammar rules in the XML spec.

The tags and structure depend on the kind of data.

- E.g. genealogical data, chemical data, business data, etc.

Tools for Writing XML

An XML doc is a text file.

Use any text editor or word processor to edit it.

Save XML docs with the **.xml** extension.

XML editors exist with specific XML editing functionality.

We will use text editors in this module (e.g. Notepad++, Scite, Sublime).

Building Blocks

Elements, Attributes & Values

The XML building blocks:

- Tags that define **elements** (opening and closing tags);
  - An **element** can contain text, graphics paths, table data, and even other elements.
- **Values** of those elements (a.k.a. element content);
- **Attributes** (a name and a value).

An **element** is the most basic unit of your doc.

Elements, Attributes & Values

We use these elements to define an **information hierarchy**

- The hierarchy describes the way in which elements can be used and how they can fit together.
- Elements **nested** in the declaration take on a relationship to the element that contains them…
- …this relationship can then be later used when retrieving or sorting the data.

Elements, Attribute & Values

Each element has an **opening tag**.

- It is the element name written between **<** and **>** symbols.
- **The element name is…**
  - …**author-defined**, and should…
  - …describe the element's **purpose** and **contents**.

Elements, Attribute & Values

Each non-empty element has a **closing tag**.

- It is the **same element name** as the opening tag...
- ...also enclosed in **<** and **>** symbols, but...
- ...with a **/** before the **>**.

```
<this_is_a_tag>  DATA  </this_is_a_tag>
```

Elements, Attributes & Values

Elements may have **attributes**.

Attributes…

- …are contained within an element’s opening tag and…
- …have values in quotation marks.

Attributes further describe an element’s purpose/content.

Elements, Attributes & Values

Info in an attribute is considered **metadata**.

- It is info about the data in the element...
- ...not the data itself.

An element can have an unlimited number of attributes.

XML grammar rule:

- Each attribute of a single element **must** have a unique name.

White Space

You can add white space around elements in your code.

- For example, indentation and line breaks.
- White spaces are not **XML rules**.

White spaces makes code easier to edit and view.

Apps reading your XML will ignore the white space.

Rules of XML

Rules for Writing XML

XML structure is extremely regular and predictable.

It **must** follow **all** the grammar rules in the XML spec.

A doc that follows the rules is **well-formed**.

Docs that are not well-formed are useless (they are not XML).

Every XML doc you write for this module **must** be well-formed.

(or you will receive zero for that xml document)

The Most Important Rules

Rule 1: A root element is required.

- Every XML doc must contain **one** and **only one** root element.
- The root element contains all other elements.
- The only XML allowed outside (preceding) the root element:
  - **Comments** (we'll cover them later);
  - **Processing instructions** (e.g. the XML declaration).

The Most Important Rules

Rule 2: Closing tags are required.

- Every element **must** have a closing tag.

- Element with no contents (empty elements) can be closed in one of two ways...

  - ...an opening and closing tag: `<empty></empty>`

    or

  - ...an all-in-one opening and closing tag: `<empty/>`

The Most Important Rules

Rule 3: Elements must be properly nested.

- If you start element A, then start element B...
- ...you **must** close element B before closing element A.

```
<A>
  <B>
    Correct
  </B>
</A>
```

```
<A>
  <B>
    Incorrect
</A>
  </B>
```

The Most Important Rules

Rule 4: Case matters.

- XML is **case sensitive**.
- Elements named **wonder**, **WONDER**, and **Wonder**...
- ...are considered **separate** and **unrelated**.

The Most Important Rules

Rule 5: Attribute values must be in quotation marks.

- An attribute's value **must** be enclosed in either...
  - ...matching **single** quotation marks: **'value'** or...
  - ...matching **double** quotation marks: **"value"**.

Writing XML

XML Processing

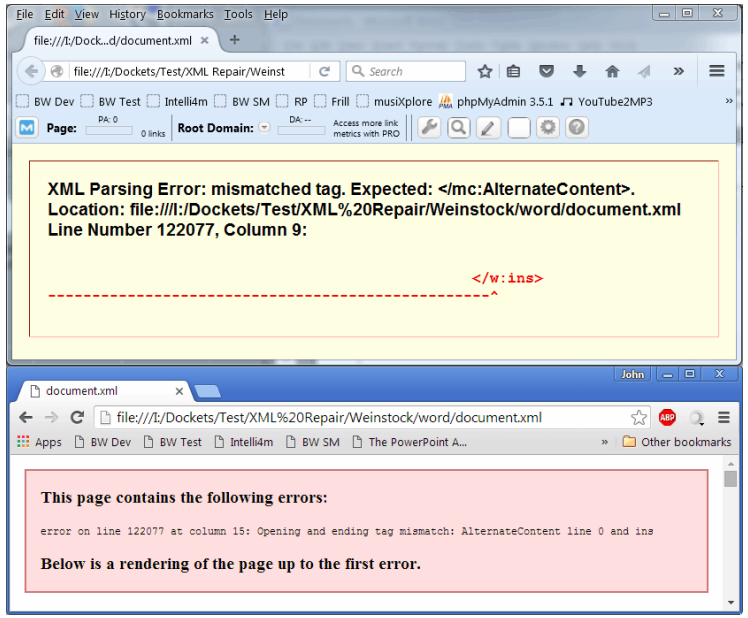
Any app that contains an XML processor (a.k.a. parser)...

...can read (parse) and retrieve data from an XML doc.

It will display errors if the XML is not well-formed.

E.g. most modern browsers contain XML processors...

...that can read XML docs and display their structure.



How to Begin

Begin each XML doc with an **XML declaration**.

```
<?xml version="1.0"?>
```

There is an XML 1.1, but we will be using version 1.0 for this module.

Tags with <? and ?> are called **processing instructions**.

You **may not** put anything before the XML declaration!

Creating the Root Element

After the XML declaration, create the **root element**.

```
<rootAnything>

</rootAnything>
```

Reminder: There must be one and only one in the doc.

The root element will contain all other elements.

Replace the word **root** with an acceptable **element name**.

Creating the Root Element

The opening and closing tag name must match **exactly**.

- **Reminder:** Case matters!

Element names should be short and descriptive.

Reminder: Only comments and PIs may precede the root.

- **PI** = processing instruction.

XML Grammar Rules

Names **must** begin with a letter, underscore, or colon.

- <Hello/> <_hello_/> <:hello:/>
- Many parser will render colons as incorrect, as they interpret colons as a use of namespaces.

You **may not** begin a name with any variation of “xml”.

- <xmlHello/>

The name may contain letters, digits, and underscores.

- <hello1goodbye/> <hello_goodbye/>
- You may use colons, hyphens, and periods, but avoid them.

Writing Child Elements

Create any child element you like within the root element.

A child element shows its relationship to its parent.

Use element names that clearly identify the content.

- It makes elements easier to process later.

```
< myCreepyAunt >content</ myCreepyAunt >
```

Reminder: Every element must be inside the root.

Writing Child Elements

Reminder: The closing tag is never optional!

Child element names must follow the XML naming rules.

The more descriptive your element names are...

...the more useful the XML will be.

Nesting Elements

You'll want to break your data down into smaller pieces.

You can create children of children of children, etc.

The ability to nest multiple levels of child elements...

...enables you to identify individual parts of your data...

...and establish a **hierarchy** between these parts

Nesting Elements

Example:

```
<outer>
      <inner>content</inner>
      ...
</outer>
```

Replace **outer** and **inner** with appropriate names.

Replace **...** with zero or more additional children for **outer**.

Nesting Elements

Reminder: Close `<inner>` before closing `<outer>`!

You may nest as many levels of elements as you like.

Best practice: Indent child elements for readability.

Adding Attributes

Reminder: An attribute adds metadata to an element.

- Use nested elements if it’s data, not metadata.

Attributes are **name-value pairs**.

Attributes appear in an element’s opening tag.

Reminder: Put the value in matching quotation marks.

Adding Attributes

Example:

```
<element attribute="value">content</element>
```

Replace **attribute** with a descriptive name.

- It must follow the XML naming rules.

Replace **value** with the metadata.

Adding Attributes

Reminder:

- All attribute names in a single element must be unique.

If the value itself contains quotes...

...use the quotes that are **opposite** from the outer quotes.

- **For example:** `comments="She said, 'Sup.'"`

Element vs Attribute

Elements are useful when…

- …the data is not a **simple type**…
- …when items may need to be repeated…
- …data need to be ordered.

Attributes are usually a good choice when…

- …there is only one piece of data to be shown…
- …when the data is a simple type…

Simple type are…

- …some text or data that can be represented as a string.

Using Empty Elements

An empty element has no content.

- It may have attributes, though.

Reminder: Write an empty element in one of two ways:

```
<element>  </element>
       -OR-

<element/>
```

If you don’t close it, the doc is **not well-formed**.

Writing Comments

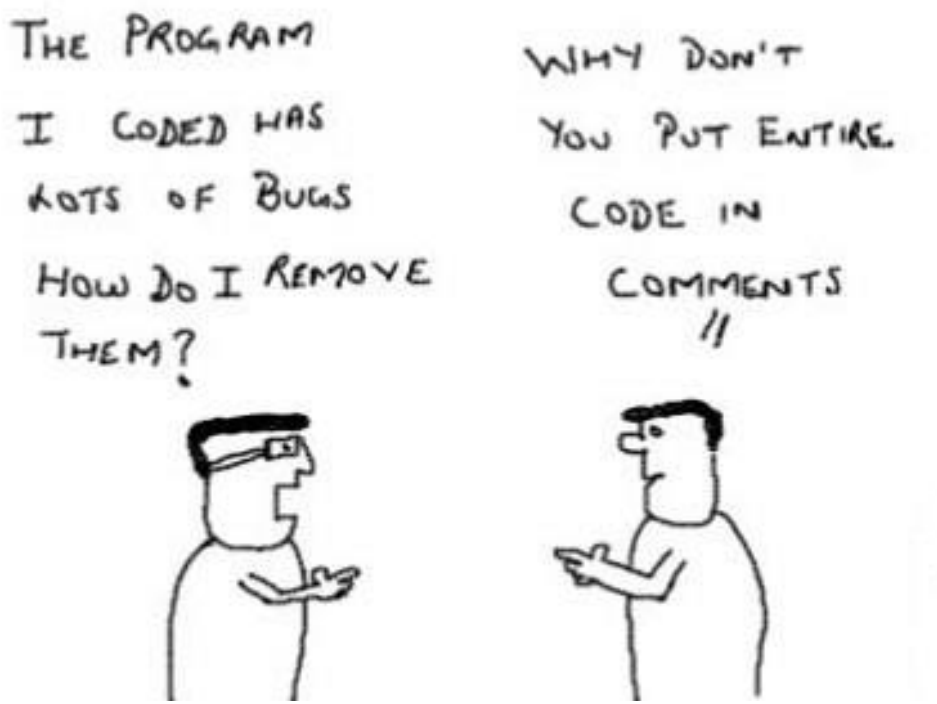
You can add **comments** to your XML doc.

They're used to add notes for a human reader.

XML processors will ignore comments.

Example:

```
<!-- this is a comment -->
```



Writing Comments

You can use comments to “comment out” pieces of code.

XML grammar rules for comments:

- Comments may contain spaces, text, elements and line breaks.
- Comments **may not** contain double hyphens (**--**).
- You **may not** nest comments.

```
<!-- THIS IS NOT CORECT <!--correct--> -->
<!-- NOR--IS--THIS -->
```

Predefined Entities

Entities are a way of entering text into an XML doc...

...without typing it all out. Similar to **constants**.

In XML, there are five **predefined entities**.

Each predefined entity represents a **special character**.

These characters have specific meanings in XML code.

Predefined Entities

To use the special character in your data...

...and to prevent processors from interpreting it as code...

...use the **entity** instead.

```
<cartoon>Tom &amp;amp; Jerry</cartoon>
```

Predefined Entities

The five predefined entities:

- Type **&** to create an ampersand (&);
- Type **<** to create a less than sign (<);
- Type **>** to create a greater than sign (>);
- Type **"** to create a double quotation mark (“);
- Type **'** to create an apostrophe (‘).

Predefined Entities

You **may not** use (**&**) or (**<**) anywhere in your XML...

...except to begin a tag or entity.

You **must** use their entities to use them in text data.

Using the predefined entities for the remaining three special characters is **optional** but **recommended**.

Displaying Elements as Text

If you want to use XML code as data in your XML doc...

...you'll want to keep it from being interpreted as XML.

Display XML code as text by using **CDATA sections**.

```
<declaration>
        <![CDATA[<?xml version="1.0" ?>]]>
</declaration>
```

Displaying Elements as Text

You may use them anywhere within the root element.

CDATA stands for (unparsed) **C**haracter **DATA**.

You may use any special characters in CDATA sections.

- Using the predefined entities is unnecessary and won't work.

You **may not** nest CDATA sections.

Theme 1: XML

End of Section 1