Intro to XML

History XML (eXtensible Markup Language)

- Based on SGML (Standard Generalized Markup Language)
- Developed by a committee in the W3C consortium on 1996-98
- It was built for the future of the web. Their goals:
    - Internet usability
    - SGML compatibility
    - General purpose stability
    - Formality
    - Conciseness
    - Legibility
    - No authoring
    - Minimization of optional features

Famous uses of XML

- **XHTML**: This is the "XMLization" of HTML 4.0 by W3c.
- **Web Collections:** Web Collections are a meta-data syntax. They fit within the WWW. Web collections are subsequently used for scheduling, HTML Email Threading, content labeling, distributed authoring, etc.
- **Chemical Markup Language (CML)**: CML is used for molecular information management. Its extensive scope covers a wide range of subjects such as inorganic molecules, quantum chemistry and macromolecular sequences.
- **Commerce eXtensible Markup Language (CXML)**: Used for communication of business documents used in e-commerce.
- **Electronic Business XML (EBXML)**: allows the use of electronic business information by everyone consistently and securely.
- **Simple Object Access Protocol (SOAP)**: Communication protocol used to exchange messages in a computer network.

Anatomy of an XML file: Prologue

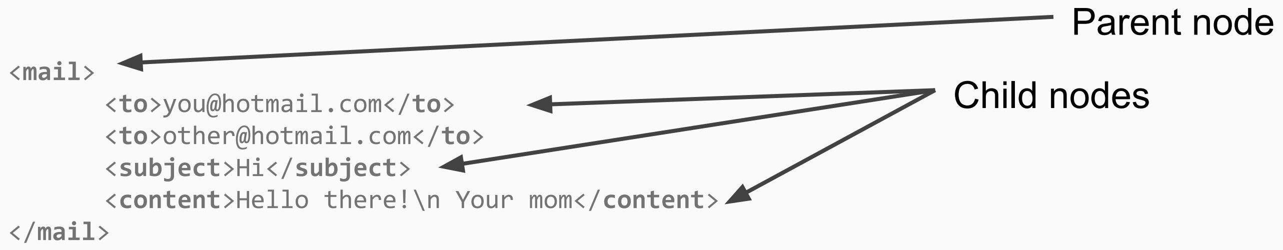
```
<?xml version="1.0" encoding="iso-8859-1"?>
```

- **<?xml** declares to a processor that this is where the XML document begins.
- **version="1.0"** declares which recommended version of XML the document should be evaluated in.
- **encoding="iso-8859-1"** identifies the standardized character set that is being used to write the markup and content of the XML.

Anatomy of an XML file: Content

XML data consist in **elements**, **attributes** and **entities** (meh).

1. Elements: The format is *<element_name>content</element_name>*
   a. Name is case sensitive
   b. Names cannot contain **<**, **>**, **&**, **“** and **:**
2. Elements can be nested:



```
<mail>
      <to>you@hotmail.com</to>
      <to>other@hotmail.com</to>
      <subject>Hi</subject>
      <content>Hello there!\n Your mom</content>
</mail>
```

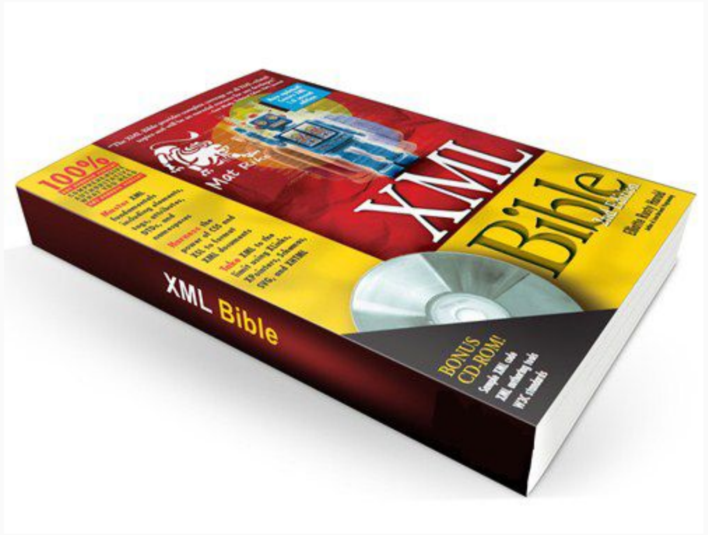
Anatomy of an XML file: Content

3. Elements can contain attributes, using single or double quotes (‘,”):

```
<!-- This is a mail definition -->
<mail>
     <to>you@hotmail.com</to>
     <to status=”cc”>other@hotmail.com</to>
     <subject>Hi</subject>
     <content>Hello there!\n Your mom</content>
</mail>
```

What we do not use

- DTDs: Documents that validates other XML
- XSLT Grammar (some do actually)
- The rest XML advanced features



JSON: Javascript Object Notation

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
 </menu>
```

```
{
   "menu": {
      "id": "file",
      "value": "File",
      "popup": {
         "menuitem": [
            { "value": "New", "onclick": "CreateNewDoc()" },
            { "value": "Open", "onclick": "OpenDoc()" },
            { "value": "Close", "onclick": "CloseDoc()" }
          ]
      }
      }
      }
```

More info

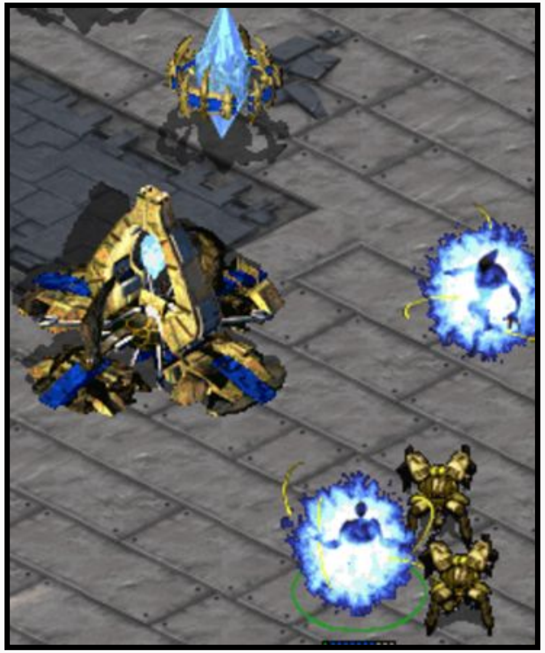
- http://www.w3.org/XML/
- Intro to XML by GameDev.net
- XPath guide
- JSON mainpage

Full example

```
<Ui xmlns="http://www.blizzard.com/wow/ui/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.blizzard.com/wow/ui/ ..\FrameXML\UI.xsd">
   <Frame name="MyAddon_Frame">
      <Anchors>
            <Anchor point="CENTER"/>
      </Anchors>
      <Frames>
            <Button name="MyAddon_Button">
                        <Anchors>
                        <Anchor point="CENTER"/>
                        </Anchors>
            </Button>
      </Frames>
   </Frame>
</Ui>
```

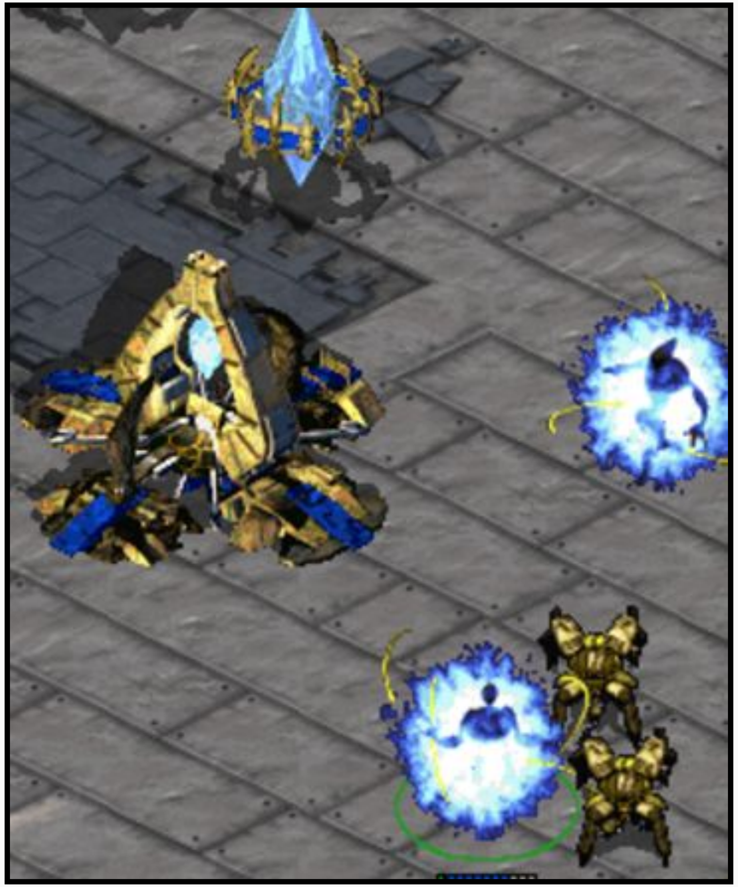
Example from a previous exam question

Come up with an XML structure that would define the game entities seen in this screenshot from Starcraft 1. The XML should define **both** their properties and their current situation. Besides following the XML syntax, avoid property repetition as much as possible.



Solution

```xml
<?xml version="1.0" encoding="utf-8"?>
<entities>
  <static>
    <pylon hp="100">
      <instance coords="50,25"/>
    </pylon>
    <base hp="500">
      <instance coords="40,50"/>
    </base>
  </static>
  <dynamic>
    <archon hp="200" damage="25" distance="1" speed="3">
      <instance coords="80,50"/>
      <instance coords="70,80"/>
    </archon>
    <dragoon hp="75" damage="10" distance="15"
speed="4">
      <instance coords="75,70"/>
      <instance coords="76,80"/>
    </dragoon>
  </dynamic>
</entities>
```



XML libraries for C/C++

- PugiXML (DOM model)
- TinyXML (DOM model)
- Expat (SAX model)
- DOM model loads all in memory
- SAX is faster but more complex to handle
- More info in this gamasutra article

TODO 1 - Creating the config file

“Let’s create config.xml to store configuration data for each module”

- You can edit xml files inside Visual Studio
- For now let’s just add the name of the app
- Come up with any tags you feel appropriate

TODO 1 - Example

```
<!-- Config file for the game -->

<config>

	<title>My super awesome game with XML</title>

</config>
```

TODO 2 - Creating “pugi” variables

*“Create two new variables from pugi namespace: a **xml_document** to store the whole config file and a **xml_node** to read specific branches of the xml”*

- To use a namespace directly use the :: notation
- E.g. pugi::xml_node

TODO 3 - Loading file

*“Load **config.xml** file using load_file() method from the xml_document class. If everything goes well, load the top tag/element inside the xml_node property created in the last TODO”*

- Check if the file was loaded correctly and has the right format
- Pugui::xml_parse_result is all you need to produce a good error message

TODO 4 - Testing the load process

“Read the title of the app from the XML and set the window title using

***win->SetTitle()**”*

- Read pugui documentation to understand how to read data
- Try executing, you should now see the new title

TODO 5 - Filling config info

“Improve config.xml to store all configuration variables that we have as macros.

Use a section with the name of each module (see Module::name)”

- Example:

```
<config>
  <app>
    <title>Game Development Testbed</title>
    <organization>UPC</organization>
  </app>

  <renderer>
    <vsync value="false"/>
  </renderer>
...
```

TODO 6 - Sending config file to all modules

*"Add a new argument to the **Awake()** method to receive a pointer to an **xml_node**.*

*If the section with the module name exists in config.xml, fill the pointer with the valid **xml_node** that can be used to read all variables for that module. Send nullptr if the node does not exist in config.xml"*

TODO 7 - Checking that the code works

*“Move **Todo 4** code to the **Awake()** method on the **window** module”*

- Pass the game title as a variable when creating the window

Homework

- Finish the code so each module receives its set of configuration variables.
- Remove all configuration macros from **p2Defs.h**
- Add music and fx volume as configuration options
    - Use this configuration in the Audio Module