Tutorial 13: Loading and saving simple XML files

http://www.codemii.com/2009/06/21/tutorial-13-loading-and-saving-simple-xml-files/

XML files are used as an easy way to read and store data which is useful for storing configuration settings from applications. It's always good practice to allow your application to be customised by the user, be it by turning off the Wiimote rumble, enabling background music, disabling prompts, etc. In this tutorial we'll learn how to load and save simple settings by using XML files.

Understanding XML file

If you don't know what an XML file looks like here is an example:

```
<?xml version="1.0" ?>
<app version="1">
      <name>Homebrew Browser</name>
      <coder>teknecal</coder>
      <version>0.3.1</version>
      <release_date>200905010000</release_date>
      <short_description>Browse homebrew apps</short_description>
</app>
```

As you can tell, it's well structured and easily readable. You firstly have the root element which tells the XML parser that this is an XML document. This isn't much importance to us but note that all valid XML files must have this root element.

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

Next we have "app" as an element with an attribute of "version" and the attribute value of "1". Elements are defined as being the text after the <. The attribute is the text that is inside the element and is equal to something, in this case it's version = 1.

```
<app version="1">
```

Moving on, we have an element beneath the "app" element.

```
<name>Homebrew Browser</name>
```

This means that this element belongs to the "app" element. In this case the element is "name" and this time the element has a content of "Homebrew Browser". We need to close off this element by using the / and the elements name.

The next few elements we can skip as they require the exact same processing as above. We then reach the end of our app element.

```
</app>
```

You can either end the file or keep adding more elements at the end such as:

```
<app version="1">
<name>Homebrew Browser 2</name>
        <coder>teknecal2</coder>
        …
</app>
```

You will firstly need to download libxml which can be found here: http://wiichat.googlecode.com/files/mxml-wii.tgz

Extract and then copy mxml-wii\mxml\lib\libmxml.a to C:\devkitPro\libogc\lib\wii and mxml-wii\mxml\lib\include\mxml.h to C:\devkitPro\libogc\include. Thanks to Beardface for porting this XML library.

You’ll need to add –lxmxl to LIBS: in your makefile and also add #include in your main.c file. Also make sure you have #include and the other necessary fat initialisation functions.

Here is our source code for both saving and loading XML files: tutorial13.zip

Saving XML files

So now we have a very basic understanding of how XML files are structured and can now proceed to save our own XML files. We'll be saving our data as attributes.

Going on the simple settings theme, let's assume we only have either a 0 or a 1 to store in our XML file and that we have 3 variables which are setting_background_music, setting_rumble and setting_tips. We have our function below which saves our settings which I'll explain below.

```
void update_settings() {
        mxml_node_t *xml;
        mxml_node_t *data;
        xml = mxmlNewXML("1.0");

        data = mxmlNewElement(xml, "settings");

        char set1[1];
        sprintf(set1, "%i", setting_background_music);
        mxmlElementSetAttr(data, "setting_background_music", set1);
        char set2[1];
        sprintf(set2, "%i", setting_rumble);
        mxmlElementSetAttr(data, "setting_rumble", set2);
        char set3[1];
        sprintf(set3, "%i", setting_tips);
        mxmlElementSetAttr(data, "setting_tips", set3);

        FILE *f;
        f = fopen("sd:/settings.xml", "wb");

        if (f == NULL) {
                fclose(f);
                printf("Settings could not be written.\n");
        } else {
                mxmlSaveFile(xml, f, MXML_NO_CALLBACK);
                fclose(f);
                mxmlDelete(data);
                mxmlDelete(xml);
                printf("Settings Saved\n");
        }
}
```

The first three items are just standard XML things which we do. We will be creating a XML file in memory at this point in time.

```
mxml_node_t *xml;
mxml_node_t *data;
xml = mxmlNewXML("1.0");
```

We now create a new element with the name settings which would look like in the file.

```
data = mxmlNewElement(xml, "settings");
```

Then we start adding our attributes, but before we do so, these attributes are of type strings and our

0 or 1 (integer) needs to be changed into a string, so we use sprintf. We create an empty char array with a length of 1. We then sprintf to set1 the value of setting_background_music.

```
char set1[1];
sprintf(set1, "%i", setting_background_music);
```

Now our char array has the value of either “0” or “1”. We can then set an attribute in our element settings to show this. We’ll call our attribute the same name as our variable which is setting_background_music for simplicity. Our setting element would like: (assuming we had the variable set to 1).

```
mxmlElementSetAttr(data, "setting_background_music", set1);
```

Now we just repeat this to the rest of our variables.

```
char set2[1];
sprintf(set2, "%i", setting_rumble);
mxmlElementSetAttr(data, "setting_rumble", set2);
char set3[1];
sprintf(set3, "%i", setting_tips);
mxmlElementSetAttr(data, "setting_tips", set3);
```

Ok, so now we are done we can begin writing the file. As normal we open the file for writing, check if we can write to the file, etc.

```
FILE *f;
f = fopen("sd:/settings.xml", "wb");

if (f == NULL) {
	fclose(f);
	printf("File could not be written to\n");
}
```

Now we can save our XML to file with just one line. After we have saved the file we close our file and then delete the XML in memory.

```
else {
	mxmlSaveFile(xml, f, MXML_NO_CALLBACK);
	fclose(f);
	mxmlDelete(data);
	mxmlDelete(xml);
	printf("XML Saved\n");
}
```

Loading XML files

So now we know what we've saved in the XML file and what it will look like this:

We can parse our XML file and find out what our settings for each variable are; we can use the below code to do this.

```
void load_settings() {
	mxml_node_t *tree;
	mxml_node_t *data;

	FILE *fp = fopen("sd:/settings.xml", "rb");
	if (fp == NULL) {
		fclose(fp);
	} else {
		fseek (fp , 0, SEEK_END);
		long settings_size = ftell (fp);
		rewind (fp);

		if (settings_size > 0) {

			tree = mxmlLoadFile(NULL, fp, MXML_NO_CALLBACK);
			fclose(fp);

			data = mxmlFindElement(tree, tree, "settings", NULL, NULL,
MXML_DESCEND);

			if (mxmlElementGetAttr(data,"setting_background_music")) {
				setting_background_music =
atoi(mxmlElementGetAttr(data,"setting_background_music"));
				printf("Setting for background music loaded\n");
			}
			if (mxmlElementGetAttr(data,"setting_rumble")) {
				setting_rumble =
atoi(mxmlElementGetAttr(data,"setting_rumble"));
				printf("Setting for rumble loaded\n");
			}
			if (mxmlElementGetAttr(data,"setting_tips")) {
				setting_tips = atoi(mxmlElementGetAttr(data,"setting_tips"));
				printf("Setting for tips loaded\n");
			}

			mxmlDelete(data);
			mxmlDelete(tree);
			printf("Settings loaded.\n");
		} else {
			fclose(fp);
			unlink("sd:/settings.xml");
		}
	}
}
```

We start almost the same as before when saving our XML file, we initialise variables to store the XML and then we just use our standard fopen to read our XML file.

```
void load_settings() {
            mxml_node_t *tree;
            mxml_node_t *data;

            FILE *fp = fopen("sd:/settings.xml", "rb");
            if (fp == NULL) {
                        fclose(fp);
            }
```

It's a good idea to check that this file has content otherwise we'll have some errors, so we can do so using ftell to tell us the file size, if it's over 0 bytes, the file has some content..

```
else {
            fseek (fp , 0, SEEK_END);
            long settings_size = ftell (fp);
            rewind (fp);

            if (settings_size > 0) {
```

Next we'll load our file to our XML variable which we created at the start and then close the file, read all our "settings" elements and store them in the variable data.

```
tree = mxmlLoadFile(NULL, fp, MXML_NO_CALLBACK);
fclose(fp);

data = mxmlFindElement(tree, tree, "settings", NULL, NULL, MXML_DESCEND);
```

Now we start reading our settings one by one. We make reference to our data variable which contains all the settings and say that we only want to read the element value for the element called "setting_check_size".

It's important to have this check, otherwise if our settings XML file didn't contain the element we were trying to read, it would cause some issues. If this element is present, it's stored as a string so we use atoi to convert it to an int and store it in our variable "setting_check_size".

```
if (mxmlElementGetAttr(data,"setting_check_size")) {
            setting_check_size = atoi(mxmlElementGetAttr(data,"setting_check_size"));
}
```

We do the same for the rest of our settings.

```
if (mxmlElementGetAttr(data," setting_rumble ")) {
            setting_rumble = atoi(mxmlElementGetAttr(data,"setting_rumble"));
}
if (mxmlElementGetAttr(data,"setting_tips")) {
            setting_tips = atoi(mxmlElementGetAttr(data,"setting_tips"));
}
```

We can remove the XML file from memory and we are done.

```
                mxmlDelete(data);
                mxmlDelete(tree);
                printf("Settings loaded.\n");
            } else {
                fclose(fp);
            }
        }
}
```

We’ve now successfully read all our settings from the XML we saved and that wraps up this tutorial. You can now save and load your applications simple settings using XML files.

PDF by Pvt Ryan (http://www.ninet.org),
Changes: Formatting.