Using PopCap Resource Manifests

1 Overview

The PopCap resource manifest scheme provides a way to specify all of the images, sounds, and fonts that an application needs to load in a simple xml file format. This allows the user to specify common parameters such as whether an image will be palletized or not and how many frames and image strip has. It also provides an easy way to load and refer to resources in a program.

2 The Resource XML File Format

The XML File format looks like this:

```
<ResourceManifest>
	<Resources id=”Init”>
		<Image id="IMAGE_TITLE_SCREEN" path="images/title" />
		<Image id="IMAGE_TITLE_BAR" path="images/goldbar" />
		<Font id="FONT_TINY" path="data/tinyfont.txt"/>
		<Font id="FONT_SMALL" path="data/smallfont.txt"/>
	</Resources>

	<Resource id=”LoadingThread”>
		<Image id="IMAGE_BKG" path="images/backdrop"/>
		<Image id="IMAGE_GEMS" path="images/gemsheet"/>
		<Font id="FONT_DIGIT" path="data/floaterfont.txt"/>
		<Font id="FONT_FLYER" path="data/flyerfont.txt"/>
		<Sound id="SOUND_COMBO_1" path="sounds/combo22"/>
		<Sound id="SOUND_COMBO_2" path="sounds/combo32"/>
	</Resources>
</ResourceManifest>
```

The top level tag is called ResourceManifest. Under this tag you specify groups of resources. You can load and unload different groups of resources at different times. For instance, in a PopCap game, the programmer usually loads the images for the title screen and progress bar in the Init method and then loads the rest of the images in the loading thread. Under the Resources tag you specify the actual resources that you want to load in that group.

You must give all resources an “id” attribute. This is how you refer to them from within your program. All resources also take a “path” parameter. This specifies where to find the actual sound, image, or font file. To avoid extraneous typing you can use the <SetDefaults> tag to specify a default path prefix and id prefix. For example:

```
<SetDefaults path=”images” idprefix=”IMAGE_” />
```

would put “images/” before every path and “IMAGE_” before every id.

Additionally, the Image tag can have the following attributes:

Attribute	Parameter	Description
nopal	None	Instructs the program not to try to palletize the image
ddsurface	None	Instructs the program to store the image in memory as a DirectDraw

		surface. This works well in circumstances where the image has no alpha component.
a4r4g4b4	None	In hardware acceleration mode this instructs the program to use the A4R4G4B4 texture format to store the image. This helps save video memory and works well when the image doesn't have close gradients of colors.
a8r8g8b8	None	In hardware acceleration mode this instructs the program to use the A8R8G8B8 texture format to store the image. Bu default, non-alpha images will be stored in R5G6B5 format if possible so this option specifies not to do that.
rows	# of rows	Specifies number of vertical images in an image strip. This can be used in conjunction with the Graphics::DrawImageCel methods which draw images out of image strips.
cols	# of cols	Specifies the number of horizontal images in an image strip. This can be used in conjunction with the Graphics::DrawImageCel methods which draw images out of image strips.
alphaimage	path	Specifies the path of an image to use as the alpha channel for this image.

So, for example, if you wanted to have a non-palletized image with 4 rows and 5 columns and have it stored as a ddsurface in software mode and stored as a4r4g4b4 in hardware mode then you would specify something like this:

<Image id="TEST_IMAGE" path="images/testimage" nopal ddsurface a4r4g4b4 rows="4" cols="5" />

3 Using the Resource Manifest in your Program

In order to use the resource manifest, you must call your resource xml file resources.xml and put it in a directory named "properties" underneath your app. To load the file call SexyApp::LoadResourceManifest. This initializes the mResoureManager object which is a member of SexyApp. Note that this does not actually load the resources. It simply parses the xml file so the ResourceManager knows where all the resources are and what their attributes are.

To actually load the resources for a group you need to call LoadResources() on mResourceManager and pass in the name of the group of resources to load. So, for instance, to load the resources with id="Init", you would call mResourceManager->LoadResources("Init"). If this returns false then you should call the SexyApp::ShowResourceError method. This will pop up a message box showing what went wrong. If you're in the main thread then you should pass true into the ShowResourceError method so that the app will exit. If you're in the LoadingThreadProc method then you should pass in false to the ShowResourceError method, set mLoadingFailure to true and then return.

To load resources from a group one at a time and update the progress you can use the ResourceManager::StartLoadResources, and ResourceManager::LoadNextResource methods like this:

```
mNumLoadingThreadTasks = mResourceManager->GetNumResources("LoadingThread");
mResourceManager->StartLoadResources("LoadingThread");
while (mResourceManager->LoadNextResource())
	mCompletedLoadingThreadTasks++;

if (mResourceManager->HadError())
{
	ShowResourceError();
	mLoadingFailed = true;
	return;
}
```

This is how you would be able to update the progress bar while loading resources in the LoadingThreadProc method.

To get resources that have been loaded by the ResourceManager, you use the ResourceManager::GetImage, ResourceManager::GetSound, and ResourceManager::GetFont methods and pass in the id of the resource. For example:

mTitleScreenImage = mResourceManager->GetImage("IMAGE_TITLE_SCREEN");

However, there is a utility program which automates this process for you so you don't need to write all of the monotonous code necessary to extract out the resources.

4 Using the ResourceGen Utility

The ResourceGen utility program reads in a resources.xml file and automatically generates source code to call ResourceManager::GetImage(), ResourceManager::GetSound(), and ResourceManager::GetFont() and store the results in variables which you can then easily use in your program. So, for instance, instead of having to do this:

g.DrawImage(mResourceManager->GetImage("IMAGE_TITLE_SCREEN"),0,0);

You could just do this:

g.DrawImage(IMAGE_TITLE_SCREEN,0,0);

The code that the resource manager generates also automatically checks to make sure that every resource you are using actually exists so you don't have to worry about checking for NULL when using the resources. The variables are named according to their id attribute in the xml file.

To use the ResourceGen utility, click the "New Project…" button and then enter the name of the project, where to find the resource.xml file, and the path to the c++ code. For example:

Project Name: Sweet Tooth
Resource XML: c:\gamesrc\cpp\ winsweet\properties\resources.xml
Resource Code: c:\gamesrc\cpp\ winsweet\ResExtract

You don't need to give an extension to the ResourceCode file, because the generator automatically creates a .h and .cpp file with that name.

To generate the code for the project, select the project in the list and click the "Generate Code" button. If you look in the header file of the generated code, you will see that there are variables for all of your resources and there are functions to hook up the variables to the resources. These functions are named ExtractXResources() where X is the name of the resource group. The functions return false if there is a problem.

Here's an example of using the Extract function in the LoadingThreadProc method:

```
if (!ExtractLoadingThreadResources(mResourceManager))
{
        ShowResourceError();
        mLoadingFailed = true;
        return;
}
```

5 Freeing Resources In Your Program

To delete resources that you are no longer using, simply call mResourceManager->DeleteResources and pass in the name of the group of resources which you want to delete. To delete all of the resources that the ResourceManager knows about, just pass in the empty string to this method like this:

mResourceManager->DeleteResources(“”);