Sexy Kanji Resource Management

Using Resource Manifests

Overview

The SexyKanji resource manifest scheme provides a way to specify common images, fonts, sounds, music and movies that an application needs to load in a simple XML file format. This also allows the user to specify common modifiers such as whether an image has alpha 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 via the ResourceManager class. By using the ResourceGenerator tool, to convert your XML into C++ code, the variable names that you give to the resources will then be accessible by any file that includes your generated resource header file.

The XML File Format

The following is an example of a resource XML File:

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

<ResourceManifest>

  <!-- Resources for the title screen -->
  <Resources id="Init">

    <Image id="IMAGE_TITLE_SCREEN"  path="images/titlescreen"/>
    <Image id="IMAGE_TITLE_BAR"     path="images/titlebar"/>
    <Font  id="FONT_SMALL"          path="fonts/smallfont.txt"/>

  </Resources>

  <!-- Resources for the game -->
  <Resource id="Game">

    <SetDefaults path="images"      idprefix="IMAGE_"/>

      <Image id="_BKG"        path="backdrop"/>
      <Image id="_GEMS"       path="gemsheet"/>
      <Image id="PARROT"      path="parrot" rows="1" cols="15" total="15" anim="loop" framedelay="5"/>

    <SetDefaults path="sounds"      idprefix="SOUND_"/>

      <Sound id="_COMBO_1"    path="combo1"/>
      <Sound id="_COMBO_2"    path="combo2"/>

    <Font  id="FONT_DIGIT"    path="fonts/digitfont.txt"/>
    <Music id="MUSIC_BACK"    path="music/back"/>

  </Resources>

</ResourceManifest>
```

The required top level tag is called ResourceManifest. Under this tag you specify groups of resources. In this example are the groups “Init” and “Game”. Resource groups can be loaded/deleted at any time. Thus, it's advantageous to split your resources into groups if you don't need to have every single resource in memory at the same time. A common use for grouping is to split the title screen, main menu, and main game elements up. Then, when the main game is playing, you

can unload the title screen data. Similarly, when you are back to the main menu, you can unload the game resources. How you use the grouping is up to you, it's provided for your convenience.

For instance, the programmer usually loads the images for the title screen and progress bar in the GameApp::InitHook() method and then loads the rest of the images in the loading thread (method GameApp::LoadingThreadProc(), see Demo 4).

Under the Resources tag you specify the resources that you want to load in that group. There are five different resource types: Image, Font, Sound, Music and Movie. Note that you can type comments in HTML style everywhere in your resource file.

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 resource on the HDD or the resource pack.

To avoid extraneous typing you can use the <SetDefaults> tag to specify a default path prefix and id prefix, see example in group “Game”: <SetDefaults path=”images” idprefix=”IMAGE_” /> would put “images/” before every path and “IMAGE_” before every id.

You do not have to specify the extensions for images (but you must add an extension for the other resource types). Following are the supported file formats for the different resource types:

Image	Font	Sound	Music	Movie
jpg png tga bmp jpf jp2 j2k	xml ttf (kfont)	ogg wav	ogg	ogg/ogv

Loading resources with the ResourceManager also provides a modifier system. For example, if we had an image we wanted to load that contained more than 1 row or column, we could specify optional modifiers, e.g. cols and rows. See the parrot image as an example:

```
<Image id="PARROT" path="parrot" rows="1" cols="15" total="15" anim="loop" framedelay="5"/>
```

Then it's automatically set for you when you load the resource. See next chapter “Resource Modifiers” for a complete list of supported modifiers.

Resource Modifiers

Types:

1 Preload attributes, needed to load the resource.

2 Postload attributes, set after resource is loaded to modify behavior.

3 Postload transformations: when image is loaded, the ImageManager class is used to manipulate the bits.

value in resource.xml	parameter	saved in/used by	type
Image:			
nopurge	true	SexyImage::PurgeBits	2
noalpha	true	ImageManager::GetSharedImage	1
alphaimage	path to image	used in ResourceManager:: LoadAlphaImage	1
variant	your ID	SharedImageRef	1
alphagrid	path to image	used in ResourceManager::LoadAlphaGridImage	1
noblend	true	SexyImage::SetNoBlendOverride	2
wrapmode	clamp = default, repeat, mirror	SexyImage::SetTextureWrappingMode	2
rows	number of rows	SexyImage	2
cols	number of cols	SexyImage	2
total	total frames	AnimInfo::mNumCels	2
anim	none once loop pingpong	SexyImage::AnimInfo	2
framedelay	delay for each frame	SexyImage::AnimInfo	2
begindelay	begin delay	used for mAnimInfo.Compute	2
enddelay	end delay	used for mAnimInfo.Compute	2
perframedelay	vector with delay for each frame	SexyImage::AnimInfo	2
framemap	vector to replace default framemap	SexyImage::AnimInfo	2
colorize	the color (r, g, b, a)	ImageManager::ColorizeImage	3
rotation	the angle (degrees)	ImageManager::RotateImage	3
mirrory	true	ImageManager::FlipImage	3
mirrorx	true	ImageManager::MirrorImage	3
hue	the deltahue	ImageManager::RotateImageHue	3
scaley	the y scale	ImageManager::ScaleImage	3
scalex	the x scale	ImageManager::ScaleImage	3

value in resource.xml	parameter	saved in/used by	type
Font:			
variant	your ID	SharedFontRef	1
size	the fontsize	SexyFont/KTrueText	2
Sound:			
volume	the volume	SoundManager::LoadSample	1
Music:			
volume	the volume	MusicManager::LoadStream	1
forceload	true	MusicManager::LoadStream	1
Movie:			
preload	true	SexyMovie::SexyMovie	1

Modifier descriptions

value in resource.xml	description
Image:	
nopurge	The bits will automatically be purged when first drawn unless nopurge is set.
noalpha	The image loader doesn't look for an alpha mask file to use with the image.
alphaimage	If a path to alphaimage is specified, noalpha is assumed and the alpha image is loaded and applied to the image resource.
variant	Since the Resource Manager uses the shared image mechanism, an image is loaded only once and all subsequent loads refer to that same image. The variant tag tells the shared image loader to load another copy of the image.
alphagrid	If a path to alphagrid image is specified, noalpha is assumed and the alpha grid image is loaded and applied to the image resource.
noblend	Sometimes an image will run up to the edge of the surface and linear blending will cause it to use the border color. Try setting noblend if the image has black line artifacts, this will turn off linear blending.
wrapmode	Set texture repeat mode, possible values (default = clamp): repeat, mirror Clamp (default): Texture coordinates outside the range [0.0, 1.0] are set to the texture color at 0.0 or 1.0, respectively. Repeat: Tile the texture at every integer junction. For example, for u values between 0 and 3, the texture is repeated three times; no mirroring is performed. Mirror: Similar to Repeat, except that the texture is flipped at every integer junction. For u values between 0 and 1, for example, the texture is addressed normally; between 1 and 2, the texture is flipped (mirrored); between 2 and 3, the texture is normal again; and so on.
rows	Set the number of rows for an image with multiple frames arranged in rows.
cols	Set the number of columns for an image with multiple frames arranged in columns.
total	Total number of frames for an image with multiple frames arranged in columns and/or rows.
anim	Set animation type: once, loop, pingpong
framedelay	Set delay for all frames in 1/100 seconds.
begindelay	Set delay for the first frame.
enddelay	Set delay for the last frame.

perframedelay	Int-vector with delay for each frame, e.g. perframedelay=”5, 5, 3, 3, 5, 5”
framemap	Int-vector to replace default framemap
colorize	Colorize (tint) the image with the passed color.
rotation	Rotate the image, valid angles in degrees are: 90, 180, 270.
mirrory	Reverse the image vertically.
mirrorx	Reverse the image horizontally.
hue	Modify the hue of the image by passed delta.
scaley	Resample the image in y direction using bilinear filtering, scale values can be specified in %, px or scalar value, e.g. scaley="50%" scaley="200px" scaley="0.65"
scalex	Resample the image in x direction using bilinear filtering, scale values can be specified in %, px or scalar value, e.g. scalex="50%" scalex="200px" scalex="0.65"
Font:	
variant	Since the Resource Manager uses the shared font mechanism, a font is loaded only once and all subsequent loads refer to that same font. The variant tag tells the shared font loader to load another copy of the font.
size	Specifying a font size is necessary for Free Type fonts.
Sound:	
volume	Specify initial volume of the sample in range [0..100].
Music:	
volume	Specify initial volume of the stream in range [0..100].
forceload	Force to load the stream, even if it’s already loaded.
Movie:	
preload	Entirely load the compressed video file in memory to avoid disk access during playback.

Here are some examples on how to set the modifiers in your resource.XML file:

```
...
<SetDefaults path="images"    idprefix="IMAGE_"/>

  <Image id="PARROT"   path="parrot" rows="1" cols="15" total="15" anim="loop" framedelay="5"/>
  <Image id="FLOWER"   path="flower" hue="100" rotation="90" mirrorx="true" scalex="50%"/>
  <Image id="PARTICLE" path="particle" colorize="0, 255, 0, 255"/> <!-- r, g, b, a -->

<SetDefaults path="sounds"    idprefix="SOUND_" />

  <Sound id="LASER"    path="laser.ogg" volume="80"/>

<SetDefaults path="music"  idprefix="MUSIC_"/>

  <Music id="GROOVE"   path="juniorgroove.ogg" volume="100"/>

<SetDefaults path="videos" idprefix="MOVIE_"/>

  <Movie id="FISH"     path="video.ogv" preload="true"/>
...
```

Using the ResourceManager 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 GetResourceManager()->LoadResourceManifest() (if you choose a different name for your XML file, you can call GetResourceManager()->ParseResourceFile("YourResourceFileName")).

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 modifiers are. Also be sure to check the return value in case parsing fails.

To load the resources for a group you need to call GetResourceManager()->LoadResources() 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 GetResourceManager()->LoadResources("Init"). If this returns false then you should call the GetResourceManager()->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 mLoadingFailed to true and then return.

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

```
mNumLoadingThreadTasks = GetResourceManager()->GetNumResources("LoadingThread");

GetResourceManager()->StartLoadResources("LoadingThread");
while (GetResourceManager->LoadNextResource())
{
        mCompletedLoadingThreadTasks++;

        if (mShutdown)
          return;
}
```

This is how you would be able to update the progress bar while loading resources in the LoadingThreadProc method, see Demo 4.

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

```
mTitleScreenImage = GetResourceManager()->GetImage("IMAGE_TITLE_SCREEN");
```

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

The ResourceGenerator Tool

Use the ResourceGenerator program to convert XML into C++ code. The variable names that you give to the resources will then be accessible by any file that includes your generated resource header file. The ResourceGenerator reads in a resources.xml file and automatically generates source code to call the Get-methods from ResourceManager and store the results in variables which you can then easily use in your program. So, for instance, instead of having to do this:

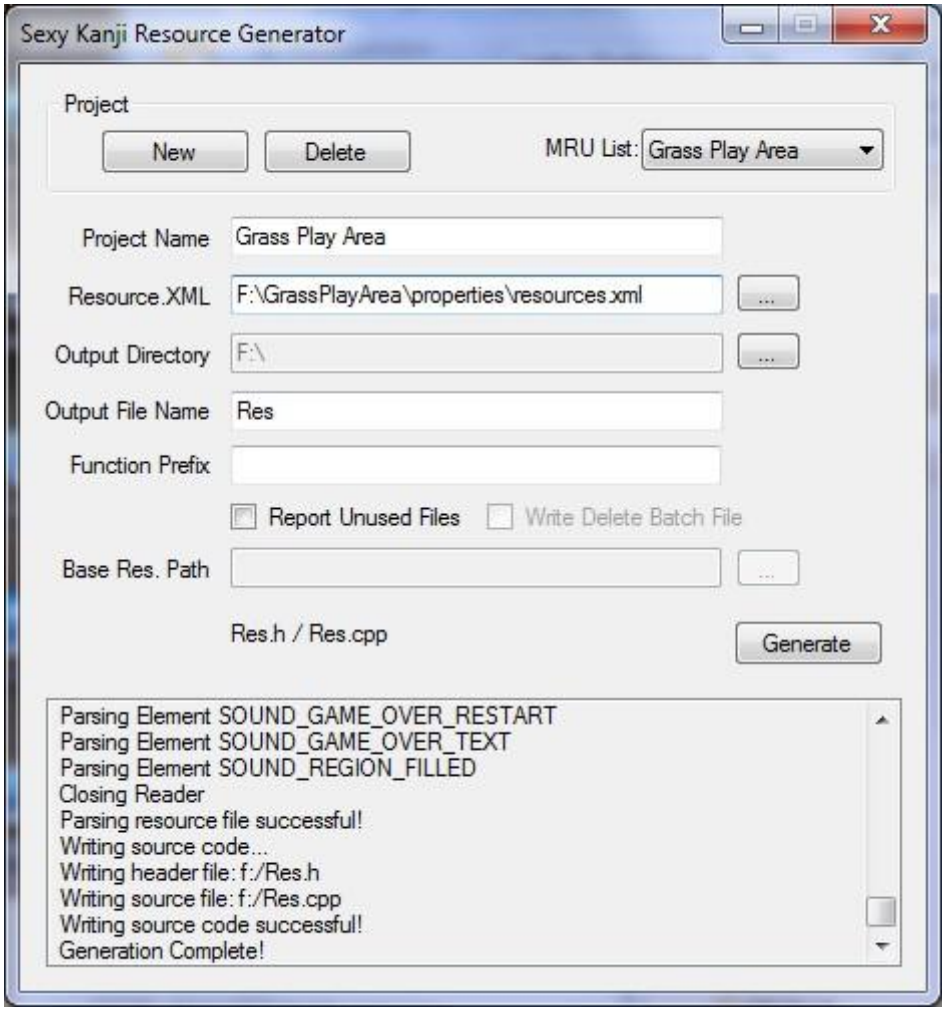
```
g.DrawImage(GetResourceManager()->GetImage("IMAGE_TITLE_SCREEN"), 0, 0);
```

you could just do this:

```
g.DrawImage(IMAGE_TITLE_SCREEN, 0, 0);
```

The code that the ResourceGenerator 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 ResourceGenerator, enter the name of the project, where to find the resource.xml file, and the output path to the C++ code. For example:



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

To generate the code for the project, click the "Generate" button.

For additional options, hover over the GUI elements and take a look at the tooltips.

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 (GetResourceManager()->HadError() || !ExtractLoadingThreadResources(GetResourceManager()))
{
        GetResourceManager()->ShowResourceError(false);
        mLoadingFailed = true;

        return;
}
```

Freeing Resources in Your Program

To delete resources that you are no longer using, simply call GetResourceManager()->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:

```
GetResourceManager()->DeleteResources("");
```