Resource Manager

The resource manager is a static class which is the central location for where resources are stored.

**Config Values**

The engine contains a set of variables which serve as configuration values for certain objects or systems within the engine. These values are stored in a configuration file in the **assets root** folder. This file is loaded when the resource manager is started, and the values are loaded into the manager. These values can be retrieved throughout the project with the **getConfigValue** function and the values can also be set at runtime with the **setConfigValue** function.

To **add** a brand new configuration variable, the config data enum in resourceManager.h must be edited with the integer value it represents be **explicitly** stated. The **static** function in resourceManager.cpp must convert an integer into a string literal for the variable. Finally, edit the **start** function of the resourceManager to load the value of the variable from the json object.

Current configuration data:

* 3D Batch Capacity (Number of 3D submissions)
* 2D Batch Capacity (Number of Quads)
* Maximum number of 3D vertices
* Maximum number of 3D indices
* Maximum number of render passes per scene
* Maximum number of layers per scene
* Maximum number of SubTextures per material
* Maximum number of lights per render call
* Use bloom for lighting calculations (0/1)
* Bloom blur factor
* Print the resource destructor log messages to console (0/1)

**Resources**

Resources refers to the common generic variables across all scenes. Resources are loaded during the start function of the resource manager. The manager defers the loading to the **ResourceLoader** class which loads resources from file. See that document for more information.

When resources are created somewhere in the project, they can be registered with the resource manager therefore allowing access to the resource elsewhere. The resource manager function **registerResource** does this for you.

The following **types** can be managed by the resource manager:

* Vertex Buffer = 0
* Index Buffer = 1
* Vertex Array = 2
* Indirect Buffer = 3
* UniformBuffers = 4
* FrameBuffers = 5
* Shader Program = 6
* Textures (Texture2D and Cubemaps) = 7
* SubTextures = 8
* Model3D = 9
* Materials = 10

All these resources are stored in a single map stored by names. The function **getResource** is a templated function which requires the name of the resource you want; it is your responsibility to provide the correct resource type. **resourceExists** checks whether the name provided is the name of a resource the manager holds.

Resources can be destroyed with the **destroyResource** function which takes a parameter of the resource name. If no name is provided, **all** resources will be deleted. The resources will be deleted in reverse order of the enum value the resource types convert to (Highest to Lowest).

**getResources** will return the entire resource list as a reference. **getResourcesOfType** will return all the resources of a certain type. **getDefaultFrameBuffer** will return the default framebuffer resource.

**Creating a New Resource Type**

1. Add the resource **type** in the ResourceType enum in resource.h
2. Edit the **toString** function in resource.h to convert the resource type to a string literal
3. Create the new resource class which must **inherit** from the Resource class
   1. Resource constructor requires resource name and type
4. Add the new resource’s include to the resource manager
5. Update the **ResourceLoader** to load the resource from file
6. Update the ResourceManager’s **start** function to call the loader’s load function