**§ 13 – ResourceManager**

The ResourceManager class is used internally to asynchronously load resources, allowing large resource allocations / large numbers of allocations without stalling the main thread or causing significant slowdowns.

There will always be slowdowns while loading resources, especially if you are loading a lot of them or the resource is very large, so loading many of them should not be done in an interactive environment. Instead, a loading screen should be used.

The ResourceManager exists in both the main thread and its own loader thread. It uses a queue to communicate between the two, with requests being pushed into the main queue from the main thread and the queue being emptied and resources loaded inside the loader thread.



There are a plethora of methods available (which may indicate a necessity to redesign):

void RequestTexture(const std::string &Filepath, const std::string &ID,

std::function<void(std::shared\_ptr<sf::Texture>, const std::string &ID)> callback);

void RequestVertexShader(const std::string &Filepath, const std::string &ID,

std::function<void(std::shared\_ptr<sf::Shader>, const std::string &ID)> callback);

void RequestFragmentShader(const std::string &Filepath, const std::string &ID,

std::function<void(std::shared\_ptr<sf::Shader>, const std::string &ID)> callback);

void RequestFont(const std::string &Filepath, const std::string &ID,

std::function<void(std::shared\_ptr<sf::Font>, const std::string &ID)> callback);

void RequestTextureChunk(std::vector<DoubleStringPair> Info,

std::function<void(std::shared\_ptr<sf::Texture>, const std::string &ID)> callback);

void RequestVertexShaderChunk(std::vector<DoubleStringPair> Info,

std::function<void(std::shared\_ptr<sf::Shader>, const std::string &ID)> callback);

void RequestFragmentShaderChunk(std::vector<DoubleStringPair> Info,

std::function<void(std::shared\_ptr<sf::Shader>, const std::string &ID)> callback);

void RequestFontChunk(std::vector<DoubleStringPair> Info,

std::function<void(std::shared\_ptr<sf::Font>, const std::string &ID)> callback);

void GiveTexture(const std::string &ID, std::shared\_ptr<sf::Texture> tex);

void LoadAudio(const std::string &FilePath, const std::string &ID, bool playAfterLoading);

void PlayAudio(const std::string &ID, bool loop);

void PauseAudio(const std::string &ID);

void StopAudio(const std::string &ID);

std::size\_t GetSizeOfTexturePool() const;

std::size\_t GetSizeOfVertexShaderPool() const;

std::size\_t GetSizeOfFragmentShaderPool() const;

std::size\_t GetSizeOfFontPool() const;

void Shutdown();

void Start();

To start with, an explanation on the parameters:

const std::string &Filepath: The file path of the resource, relative to the executable

const std::string &ID: A *unique* ID to identify the resource by. A resource will not be loaded in a resource with the same ID is already loaded.

The next parameter is the function to call when the resource loading is finished.

All of these functions have a return type of void and take different parameters depending on the type of resource being requested. A std::function object is used to store the callback so as to correctly execute the proper code, and to allow the object making the request to have a call to a member method directly

**Requesting a texture:**

The signature of the callback is: void(std::shared\_ptr<sf::Texture>, const std::string &ID)

In other words, it has no return and takes 2 parameters:

* std::shared\_ptr<sf::Texture> - a shared\_ptr to the resource that was just loaded
* const std::string &ID – the ID of the resource that was just loaded

Example of usage:

ResourceManager->RequestTexture("./SomeFilePath.png", "SomeUniqueID",  
 [this](std::shared\_ptr<sf::Texture> tex, const std::string &ID)  
 {  
 this->Textures[ID] = tex;  
 });