# Introduction

In this document I will explain how you can get started using C++ as a runtime recompiled scripting language. I will go through the template, which is set up to be an empty compilable project. How you can add your own classes and how you can spawn them. The various ways in which you can debug your projects and the various components. As well as a link to a pong demo, which has implementations of a lot of the principles explained in this document.

# Template

In the same folder as this document you can find the Template folder. This folder is set up as a bare bones project for you to start in. After you have built the Engine you can run the “RunGame.bat” file and you will be greeted by a black screen.

If you have a look in the Scripts folder you’ll see that there are two files which contain the “Game class”. This class is the starting point of all projects. The engine knows about this class and will invoke it at the start of the game. All of your future classes will be spawned by the game in one way or another.

If you want to edit any of your scripts, you can open the “Project” folder. In here is a solution with the “Game class” files already added. You can also edit them with a basic text editor if you so desire.

# Adding Classes

As stated above the “Game class” gets spawned at the start of the game. To spawn other classes we first need to create a new class. Normally this would be done by “Adding” a new class in the solution explorer. This can still be done, as this already sets up the class in the files. However, it adds it to the “Project” folder. This is not the correct folder. To migrate it to the correct folder, first save it, then remove it from the project (don’t delete it). Navigate to the “Project” folder, cut the two files and paste them in the “Scripts” folder. Now you can use the “add exisitin item…” option to add them back to the project.

Alternatively you can add an empty .h and .cpp file directly to the “Scripts” folder. After which you can use the “add existing item…” option to add your scripts.

Now that you have a class, you have to make it a child class of the “Object class”. This is a requirement for all scripts. This makes it possible for the engine to interface with your script. To be able to spawn the class you’ll need to implement all of the pure virtual functions of the “Object class”. See [here](https://swarm1.nhtv.nl/view/Projects/Resul130134/P1/Engine/Doxygen/html/class_object.html) for a list of the pure virtual functions of the “Object class”.

Now that you have set up the class correctly you have to add a few macros so that the engine has everything it needs. Starting in the header file add the INTERFACE\_START and INTERFACE\_END macros to the project in the start macro provide your class name as argument.

In the cpp file add the SCRIPT\_START and SCRIPT\_END macros. Providing your class name again in both macros.

Now that you have everything set up correctly you can spawn your class. This is accomplished using the Level class. Each Object class has a pointer to this class. To spawn your class we need to invoke the CreateObject function using the correct parameters. The first parameter is the name of your class. The second is the identifying name. This name is used when searching for it in the Level’s GetObjectWithName function.



Snippet 1: Spawning an instance of a class.

# Debugging

Once we start creating a game we will inevitably run into bugs. We have multiple ways to debug them so that we can fix them.

Firstly is the logging. Anywhere in the scripts you can use the Logger class to print out text to the console. For example to print out the delta time in the Update function we would add this line to the Update function:



Snippet 2: Logging a debug line in the console window.

The first argument can always be “console”; The second argument is your message, with the variables replaced with “{}”; And the rest of the arguments are the variables in order they appear in the text. This function will print out the following in the console:



Snippet 3: Output of Snippet 2 in the console window.

For a more detailed description and the different logging levels click [here](https://swarm1.nhtv.nl/view/Projects/Resul130134/P1/Engine/Doxygen/html/class_logger.html).

Secondly is being able to use the visual studio suite of debugging tools. This takes a bit more work to accomplish, but is a far more powerful tool in the right hands. Start by opening the engine by going to “Engine/Main/main.sln”. Now go into the “Solution Explorer” and right click on the “Solution”. Select “Add->Existing project…” and navigate to your project and select it. Now that your project is loaded into the engine you will be able to use the debugging tools.

However, the engine will not yet load your project. To accomplish this, right-click the “Main” project and open the properties. Navigate to the “Debugging” Header and change the “Command Arguments” variable to point to your project. Now the engine will compile and load your project. I recommend building in the Debug configuration to utilize all of the debugging tools visual studio provides. Now you can place breakpoints anywhere in your project to see it working. You can edit your scripts right in the engine project, even if the engine is running.

# Components

You can interact with the engine using Components. There are multiple components, each components is a group of functions that affect the same thing. The Components currently in the engine are:

* AudioComponent
* InputComponent
* PhysicsComponent
* AnimationComponent
* SpriteComponent
* WindowComponent
* RenderingComponent

Most of the Components have one or more Add functions. These functions are used to initialize and add a new component of that type. They will also always return a CompHandle. This Handle is then used with all other functions, with a few exceptions, in the same class.

All of the components are well documented, the documentation can be found [here](https://swarm1.nhtv.nl/view/Projects/Resul130134/P1/Engine/Doxygen/html/annotated.html).

# Examples

For an example you can load the Pong demo by navigating to “Pong/Project/Project.sln”.