# Documentation of GCEngine 3.2

## Welcome

Welcome to the official GCEngine documentation. Here you will find a detailed view of all the GCEngine classes and functions.

## Short example

Here is a short example, to show you how simple it is to use GCEngine:

#include "GCEngine.h"

int WinMain(HINSTANCE hInstance, HINSTANCE prevInstance, PSTR cmdLine, int showCmd)

{

// *Create the GameManager*

GCGameManager\* pGameManager = GC::CreateGameManager(hInstance);

// *Create a Scene*

GCScene\* pScene = GCScene::Create();

// *Create a GameObject*

GCGameObject\* pGameObject = pScene->CreateGameObject();

// *Add a sprite to the GameObject*

pGameObject->AddComponent<SpriteRenderer>()->SetSprite("goofyImage.dds");

// *Start the game*

pGameManager->Run();

return 0;

}

# Tutorials for GCEngine 3.2

## Getting started

* [Setup your first Project](#_Setup_your_first_1)
* [Create your first Script](#_Create_your_first)

## Events

* [Inputs](#_Keyboard_Inputs)

# Setup your first Project

## The GameManager

To make a game, you first need to create a GameManager. It is the most important part of the game as it will be the one managing the window, the timer as well as every other manager (physic, render, etc.).

#include "GCEngine.h"

int WinMain(HINSTANCE hInstance, HINSTANCE prevInstance, PSTR cmdLine, int showCmd)

{

// *Create the GameManager*

GCGameManager\* pGameManager = GC::CreateGameManager(hInstance);

// *Your code here*

// *Start the game*

pGameManager->Run();

return 0;

}

The Run() method starts the game loop and keeps the game running until the window is closed.

## The Scene

Now that the GameManager was created, you can create your first Scene. It is necessary if you want to create a game as it will allow for a better organization and a performant way to change the game’s level for instance.

#include "GCEngine.h"

int WinMain(HINSTANCE hInstance, HINSTANCE prevInstance, PSTR cmdLine, int showCmd)

{

// *Create the GameManager*

GCGameManager\* pGameManager = GC::CreateGameManager(hInstance);

// *Create a Scene*

GCScene\* pScene = GCScene::Create();

// *Start the game*

pGameManager->Run();

return 0;

}

## The GameObject

Finally, everything is setup to create our first GameObject. It is an empty object that you are going to do everything with (with the help of Components).

#include "GCEngine.h"

int WinMain(HINSTANCE hInstance, HINSTANCE prevInstance, PSTR cmdLine, int showCmd)

{

// *Create the GameManager*

GCGameManager\* pGameManager = GC::CreateGameManager(hInstance);

// *Create a Scene*

GCScene\* pScene = GCScene::Create();

// *Create a GameObject*

GCGameObject\* pGameObject = pScene->CreateGameObject();

// *Start the game*

pGameManager->Run();

return 0;

}

The only issue with the code above is that, if you run it, nothing is going to appear on your window. This is because, juste like mentionned earlier, GameObjects are empty by default. To make it visible on our window, we’ll need to add a SpriteRenderer to it and load a texture.

#include "GCEngine.h"

int WinMain(HINSTANCE hInstance, HINSTANCE prevInstance, PSTR cmdLine, int showCmd)

{

// *Create the GameManager*

GCGameManager\* pGameManager = GC::CreateGameManager(hInstance);

// *Create a Scene*

GCScene\* pScene = GCScene::Create();

// *Create a GameObject*

GCGameObject\* pGameObject = pScene->CreateGameObject();

// *Add a Sprite to the GameObject*

pGameObject->AddComponent<SpriteRenderer>()->SetSprite("goofyImage.dds");

// *Start the game*

pGameManager->Run();

return 0;

}

The AddComponent() method will add the given Component to the GameObject and will return it so that you’re able to store it or use it directly if you want. In this case, I don’t need to store it as I won’t do anything more to it but I’m calling its SetSprite() method to load my texture.

In this version, you have to give a .dds file if you want to load a texture in your game. There are tons of .png to .dds converters online so it shouldn’t be a problem to convert your images. IMAGE\_DANS\_TEXTURE

# Create your first Script

## Introduction

A Script can be used to run code during the game loop. It can be assigned to a GameObject, a Scene or even to the GameManager. Because Scripts are nothing more than Components with already built-in methods, they are added to the GameObject the exact same way as Components : through the AddComponent() method.

## The Script Setup

Scripts are C++ classes, which means you will need to create two files before starting to code your Script : a header and a source file.

**// *Example.h***

#pragma once

#include "GCEngine.h"

CREATE\_SCRIPT\_START(Example)

// *Your code here*

CREATE\_SCRIPT\_END

**// *Example.cpp***

#include "Example.h"

// *Your code here*

Because of how the Engine works, you’ll also have to add a CopyTo() method in the source file before adding anything to the Script.

// ***Example.cpp***

#include "Example.h"

void ScriptExample::CopyTo(Component\* pDestination)

{

Component::CopyTo(pDestination);

ScriptExample\* pScript = static\_cast<ScriptExample\*>(pDestination);

}

If you don’t plan on having any variables stored in your Script, you can remove the last line of CopyTo() because it will only be useful if you have some.

As you can see, CREATE\_SCRIPT\_START(Example) creates a class named "ScriptExample" and not "Example".

## Coding your Script

Now that you’ve setup your Script, you can start coding. The Start() and Update() built-in methods are probably the one you’re going to use the most so let’s add them to the Script as well as some custom methods and variables.

**// *Example.h***

#pragma once

#include "GCEngine.h"

CREATE\_SCRIPT\_START(Example)

private:

void Start() override; // *This built-in method is called when the Script is created*

void Update() override; // *This built-in method is called every frame*

void ResetPosition();

void TurnToDay();

void TurnToNight();

SpriteRenderer\* m\_pSpriteRenderer;

bool m\_night;

unsigned int m\_days;

CREATE\_SCRIPT\_END

**// *Example.cpp***

#include "Example.h"

void ScriptExample::CopyTo(Component\* pDestination)

{

Component::CopyTo(pDestination);

ScriptExample\* pScript = static\_cast<ScriptExample\*>(pDestination);

pScript->m\_night = m\_night

pScript->m\_days = m\_days

}

void ScriptExample::Start()

{

// *Get the SpriteRenderer and initialize the day counter and the night status*

m\_pSpriteRenderer = m\_pGameObject->GetComponent<SpriteRenderer>();

m\_night = false;

m\_days = 0;

}

void ScriptExample::Update(

{

// *Your code here*

}

void ResetPosition()

{

// *Your code here*

}

void TurnToDay()

{

// *Your code here*

}

void TurnToNight()

{

// *Your code here*

}

As you can see, the CopyTo() method has had some changes : that’s because we added new variables to the Script. Every variable that you would want to duplicate when you duplicate the Script should be added there.

# Inputs

## Keyboard Inputs

if (KEYBOARD::GetKeyDown(KEYBOARD::RIGHT\_ARROW)) {}



### Functions Keys

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 | F11 | F12 | F13 | F14 | F15 | F16 | F17 | F18 | F19 | F20 | F21 | F22 | F23 | F24 |

### Numbers Keys

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Key1  Ampersand | Key2  E\_Acute  Tilde | Key3  Double\_Quotation  Hashtag | Key4  Single\_Quotation  LCurly\_Bracket | Key5  LParenthese  LSquare\_Bracket | Key6  Minus  Vertical\_Bar | Key7  E\_Grave  Grave\_Accent | Key8  Underscore  Backslash | Key9  C\_Cedilla  Circumflex | Key0  A\_Grave  At |

### Letters Keys

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | | Z | | E | | R | | T | | Y | | U | | I | | O | | P | |
|  | Q | | S | | D | | F | | G | | H | | J | | K | | L | | M | |
|  | | W | | X | | C | | V | | B | | N | |  | |  | |  | |  | |

### Special Keys

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ESCAPE  ESC |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | BACKSPACE |
| TAB |  |  |  |  |  |  | ENTER |
| CAPS\_LOCK |  |  |  |  |  |  |
| SHIFT  LSHIFT |  |  |  |  |  | SHIFT  RSHIFT | |
| CTRL  LCTRL | LWINDOW | AltS  LALT  ALT | SPACE  SPACEBAR | ALTs  RALT  ALTGR | RWINDOW | MENU | CTRL  RCTRL |

### Navigation Keys

|  |  |  |
| --- | --- | --- |
|  | SCROLL\_LOCK | PAUSE |
| INSERT | HOME | PAGE\_UP |
| DEL | END | PAGE\_DOWN |
|  | UP\_ARROW  UP |  |
| LEFT\_ARROW  LEFT | DOWN\_ARROW  DOWN | RIGHT\_ARROW  RIGHT |

### Numpad Keys

|  |  |  |  |
| --- | --- | --- | --- |
| Num\_Lock | Numpad\_Divide | Numpad\_Multiply | Numpad\_Substract |
| Numpad7 | Numpad8 | Numpad9 | Numpad\_Add |
| Numpad4 | Numpad5 | Numpad6 |
| Numpad1 | Numpad2 | Numpad3 | Enter |
| Numpad0 | | Numpad\_Period |

## Mouse Inputs

if (MOUSE::GetButtonDown(MOUSE::LEFT)) {}

MIDDLE

LEFT RIGHT

FORWARD

BACKWARD