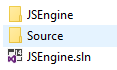
# Engine Set Up

This technical doc will show you how to set up the project for a new game and create a game within it.

## Overview

You’ll start with:

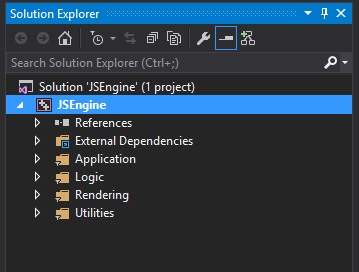


|  |  |
| --- | --- |
| **Folder/File** | **Purpose:** |
| JSEngine | .vcxproj and .vcxproj.folders |
| Source | Engine .h and .cpp as well as 3rd party .libs and .dll |
| JSEngine.sln | The Solution containing the engine. Thats where the game will be created |
| Bin | Will have all the binary files of the engine. .dlls |
| Intermediate | Will have all the .obj files of the engine |

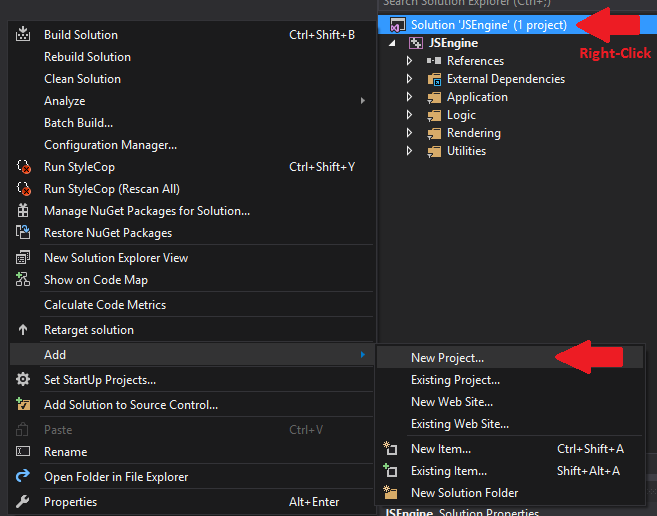
## Creating a Game Project

### New Project

* Start by opening up the JSEngine solution. This is what you’ll see in the Solution Explorer:



* Next up we’ll create a new project under the JSEngine Solution. One of the ways to do it is to right-click on the Solution in the Solution Explorer, go under Add and select the New Project… option.



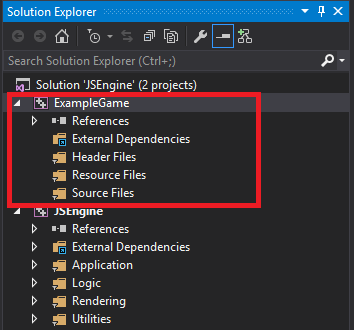
* For this presentation I’ll name my game ExampleGame and its going to be an Empty Project under Visual C++

https://lh4.googleusercontent.com/rObg7N-iGOyjJoJS308sQ8asQSR-EjjQ6XQI1GLRSOXS7-W6w74f--t0DE05jERK4G0Rn58exVYzguMFlYnHEjqcr-278Wp0cBBlrnyYiaxCS9b-9rjmKrJFLLZQRo0abH6A27NB

* Make sure that the location of the game is the same as the location of the Solution.

https://lh3.googleusercontent.com/3ET_cnxNZE5Doc9D5JRUFLPYM7pOXOtZO5Nw_WCetdxx9tiLxI1vOE6v5voaXHAdvtk956ajKZ7YPQmKGdPsOk2JOZYXbBfSGeYkwM7tp-2JzEhGMISzUGZfvoIc6yYziRT4Ei6z

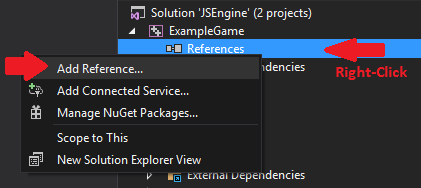
* Our game will now appear under the Solution Explorer



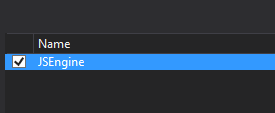
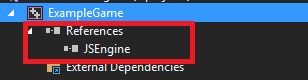
### Project Dependencies

We’re now going to set up project dependencies so the engine is built before the game is.

* Let’s start with Adding reference to our game. We can do that by right clicking on the references under our game project in Solution Explorer and choosing “Add Reference…”



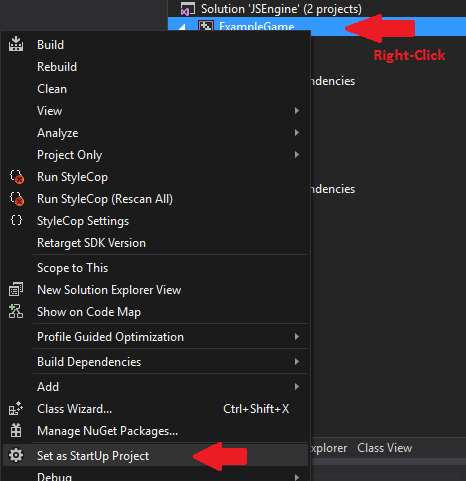
* From there, check a box next to the JSEngine as the dependency for your game, and you should end up with JSEngine reference under the References tab.

### Startup Project

Now let’s set up our game as the startup project, that will ensure that the game is the main project of the solution.

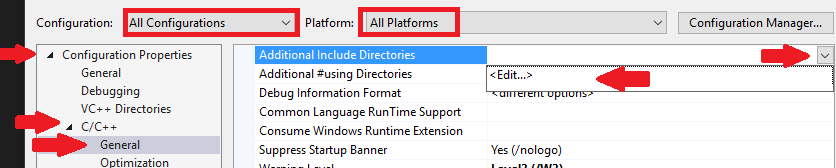
* We can do that by right clicking on the game project and choosing “Set as StartUp Project”



### Engine Includes

We’re now going to include the source code of the engine into our game so we can include the header files and use them.

* Start by going to the Properties of the game project by right clicking on it.
* Set up the Configuration to All Configurations and Platform to All Platforms
* Go to Configuration Properties/CC++/General/ and edit the field for Additional Include Directories.



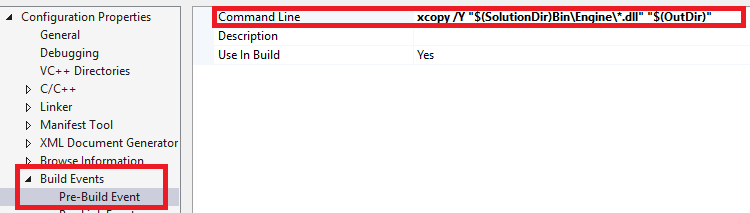
* The source code of the engine lives under our Solution directory under Source. So let’s direct visual studio there by using this line of code: “$(SolutionDir)Source”

https://lh5.googleusercontent.com/kvn5c-605g8kzWHISLTThYiRZw_h6-hzGbrEaqi7iTsRq9Ax0hM83iNpXy4C6BAc90aMjc0ZP1aPDDRkT8YHRIdD9dGTssyJv3Bf3519hj02XP3PCpBJ1FxETy-Fklp7TDJMNQNs

### Copying over engine .dlls

It’s very important that our game has access to engine’s dlls. Let’s set it up so every time the Solution is being built, It’ll copy the dlls from the engine Bin folder into our directory.

* Lets go to our game project Properties and at the very bottom go to Build Events/Pre-Build Event
* In the Command Line add a line *$(SolutionDir)Bin\Engine\\*.dll" "$(OutDir)"*



## Creating a Game

### Setting up folders

Before we start making the game we have to set up a couple of folders. One of them will be for our map and the second one will be for all our textures.

Both of those folders will live under Media under your main game directory.

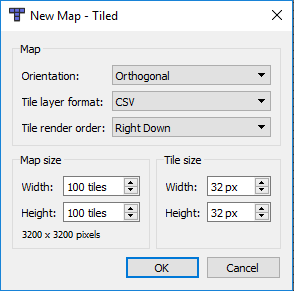




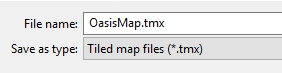
### Creating a Map

We’re going to use Tiled map editor to make a map for our game. This is an editor that the engine supports at the moment.

* Let’s start by creating a new project that’s Orthagonal, CSV Tile layer format, and Right Down Tile render order. Map size and Tile size can be set to anything.



* Let’s save out our empty game map into the Maps folder we have created earlier. I’m going to name my my OasisMap and the extention will be a .tmx

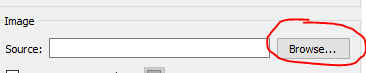


In Tiled under Map/New Tileset we can create new tilesets which will include our texture maps.

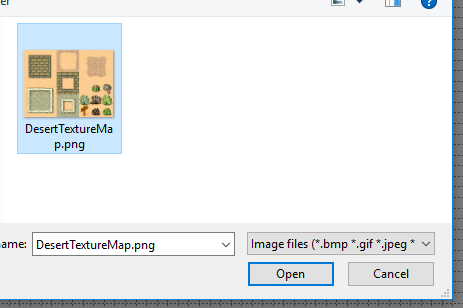
* Click on New Tileset under Map tab at the very top of Tiled editor



* Click Browse right next to the Source file path

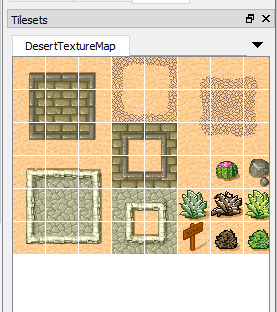


* Select an image that you have placed under the Media/Textures folder

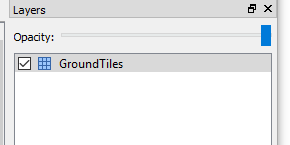
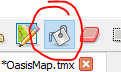


Unfortunately JSEngine only supports .png files and no spacing between the tiles.

After we select the texture, and apply, the tileset will appear under out tilesets.



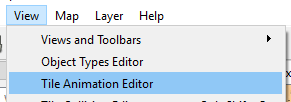
I’m going to start making my map by renaming the tile layer to GroundTiles and filling the entire map with sand tile.

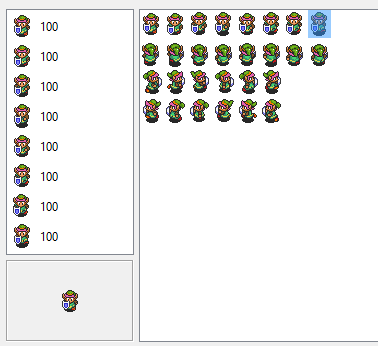
#### Player Placement

I’ll load up one more image. This one is going to be my animated player, that when placed on the map, will appear in the game.

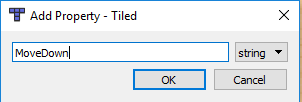
* After the Tileset is loaded. I’ll select one of my walk downwards frame and go to Tile Animation Editor that lives under View/Tile Animation Editor



* Double click all the frames that you want to appear in the walk down animation, and they’ll appear on the left side along with the duration time in milliseconds and a preview of the animation.



* Now we’ll name each animation by clicking on it in the Tileset and adding a new property. Im going to call my animations MoveDown, MoveLeft, MoveRight, and MoveUp



I’m going to repeat the same for move up animation, move right and move left.

I will now create a new object Layer, select an Insert Tile(T) tool and place my player somewhere on the map.

Let’s give some components to our player.

On my player I’m going to need:

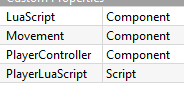
LuaScript-Component

Movement-Component

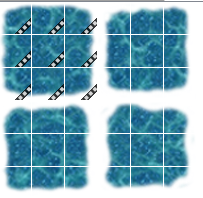
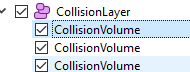
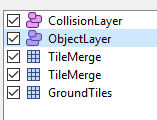
PlayerController-Component

And PlayerLuaScript-Script

That lua script is an example that I wrote which has a simple movement script on it that moves the character around and calls animations on the player.

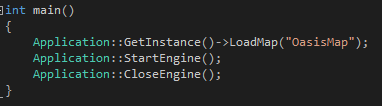


I have created a couple of more layers and placed a few objects around the map.



Added a few more animations, and created empty objects for collision volumes.

Made sure to load my map into the game.



And here’s the result



### Creating Gameplay

To create gameplay logic you’ll be using a main loop that is updated every frame. I have created a simple gameplay logic by creating a lua script and putting it on one of the tiled objects. In that script I check for OnClicked event from cpp, and when that happens I close the engine client.

Using that technique we can do more stuff with it like incrementing players health on click, or changing movement speed etc.

## Engine Usage

### Release and Debug Mode.

Release mode of the engine will compile at the highest setting of optimization.

Debug mode will lack of optimization, draw boxes on the colliders and around the objects themselves.

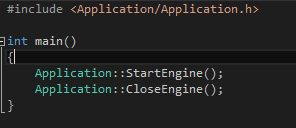
Debug mode will also have a debug console window that will show debug log messages.

### Simple StartUp

Using engine code can be done by including Application.h that lives under Application folder.

I’m going to include it and call a couple of engine functions out of my game’s main function

Those 2 functions are what starts and closes the engine making sure there is no memory leaks behind us.



### Loading a Map

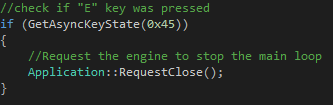
To load a map you can simply call LoadMap(“MapName”);



### Requesting Engine Shutdown

To shut down the engine you can simply call RequestClose();

Here is one variation of how one might close the engine.



Another way to request an engine to close is through Lua.



This is a little lua script I wrote and call it from ButtonComponent.

# Engine Components

Engine is created from multiple subsystems that that needed to run your game.

* GameObject is what each entity is in the engine. By giving GameObject Components you specify what kind of object it is. For example adding PlayerControllerComponent, a MovementComponent, and some lua script that would handle movement will create a player for us.
* Camera. Will follow around an object that has a PlayerControllerComponent on it.
* Renderer. Renders sprites on the screen that have been registered on their creation. If the object gets deleted from the game the renderer will remove it from its list.
* LuaScript Engine uses lua as its scripting language. There is a few public interface functions in it like OnClicked OnKeyboardInput, OnMouseInput etc.
* Multiple calls from lua to cpp like CppSetVelocity(x, y, z) will set the velocity of an objects movement if it exists
* Zlib that can be used to compress the media files and uncompress them.
* Resource Manager that will make sure resources are not being loaded multiple times, instead it will use the same textures that were already loaded.
* XML used to parse maps.
* SDL used as a main renderer layer.