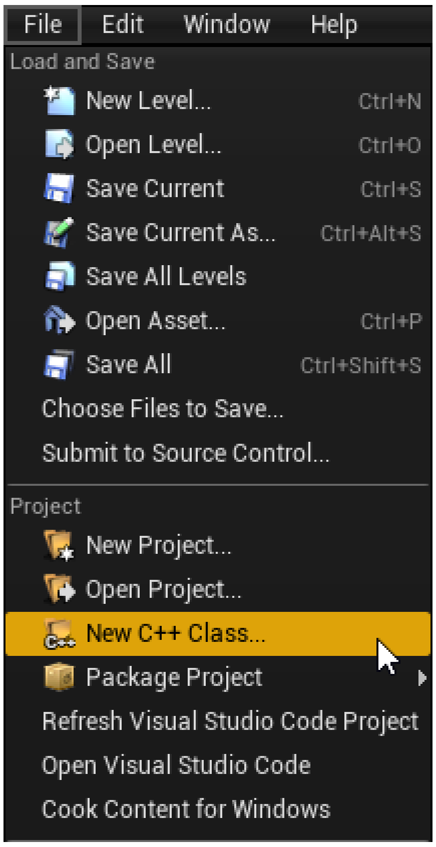
Mechanical

ECS clearly separates the data and the logic operating on that data. This logic in turn is usually executed on an iterative per-frame basis. Apparatus implements this animation-like functionality via a concept called *Mechanical*. Mechanicals are complex in nature and comprise multiple Mechanics that are executed inside of them.

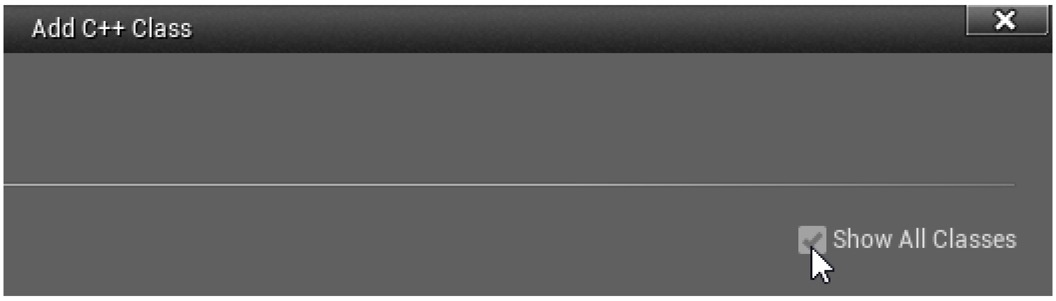
C++ Workflow

If you’re going the C++ way, creating your Mechanicals goes like this.

1. Open the main UE File menu and choose the “New C++ Class…” option:

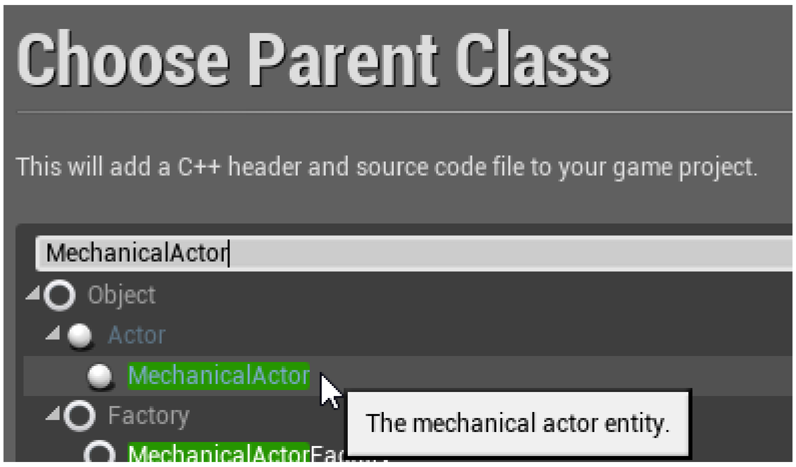


2. In the opened window mark the “Show All Classes” checkbox:



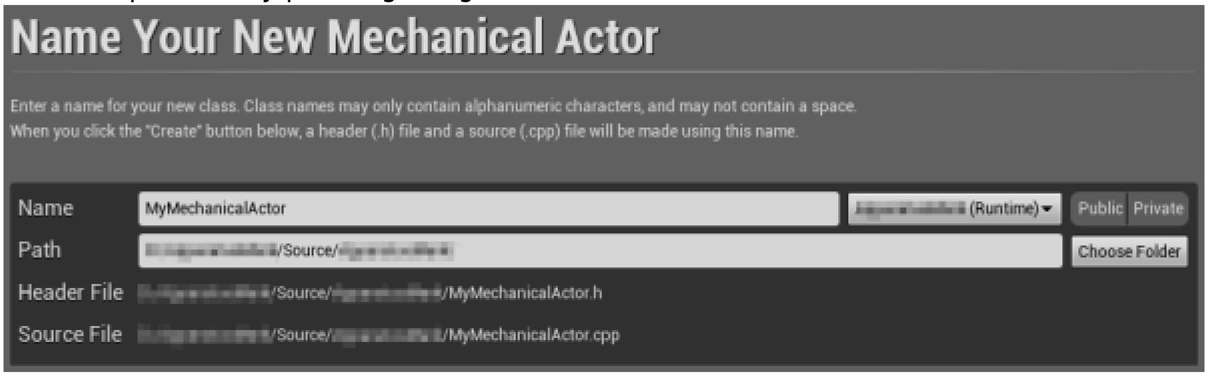
✓ Show All Classes

3. Now you can select any of the base classes available including the Apparatus ones. Choose the Mechanical Actor as a base class:


Choose Parent Class

This will add a C++ header and source code file to your game project.

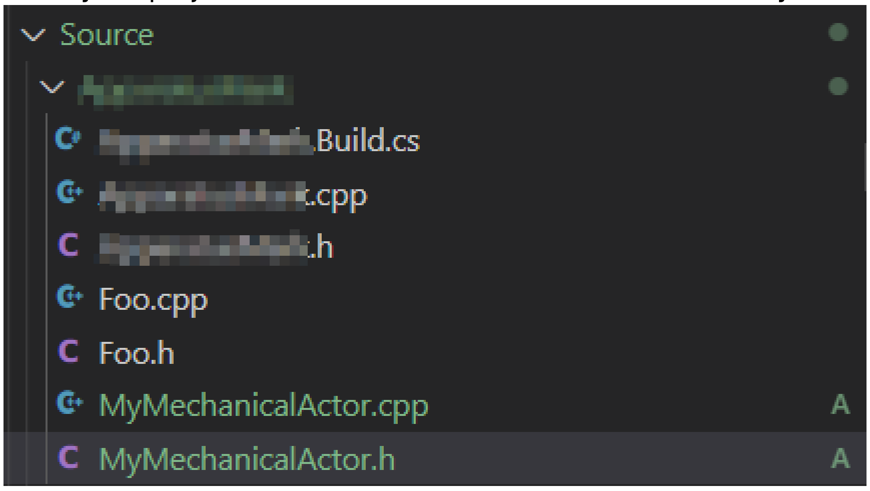
4. Click “Next” and you should see a name choosing dialog. Adjust the name of the class as needed and proceed by pressing the green “Create Class” button at the bottom:


Name Your New Mechanical Actor

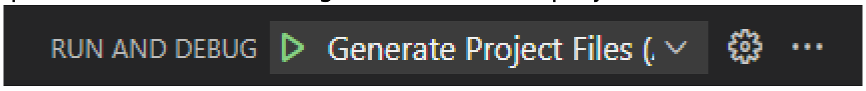
Enter a name for your new class. Class names may only contain alphanumeric characters, and may not contain a space.
When you click the "Create" button below, a header (.h) file and a source (.cpp) file will be made using this name.

Name: MyMechanicalActor (Runtime) Public Private
Path: /Source/ Choose Folder
Header File: /Source/ /MyMechanicalActor.h
Source File: /Source/ /MyMechanicalActor.cpp

5. The new class gets created as a combo of its header (.h) and a source file (.cpp). All in the Source (sub)folder of your project. You should now see them in the IDE of your choice:



6. Note that you may have to recompile the project and/or restart the Editor after that. Don’t be scared by some possible errors here, regenerate the IDE project, rebuild and start again.



7. The corresponding file contents should be as:
   - `MyMechanicalActor.h`:

```
// Fill out your copyright notice in the Description page of
Project Settings.

#pragma once

#include "CoreMinimal.h"
```

```
#include "MechanicalActor.h"
#include "MyMechanicalActor.generated.h"

/**
 *
 */
UCLASS()
class MY_API AMyMechanicalActor : public AMechanicalActor
{
    GENERATED_BODY()
};
```

   - `MyMechanicalActor.cpp`:

```
// Fill out your copyright notice in the Description page of
Project Settings.


#include "MyMechanicalActor.h"
```

8. Now you can override a single (or multiple) Tick method(s) as you usually would do in C++…
   - … in the header:

```
void Tick(float DeltaTime) override;
```

   - … and the source file:

```
void AMyMechanicalActor::Tick(float DeltaTime)
{
    // Your mechanical code here...
}
```

9. Proceed creating a Filter to enchain the Chunks/Belts in order to be iterated upon.