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:

File Edit Load and Save

New Level...
 Open Level...
 Save Current
 Save Current As...
 Save All Levels
 Open Asset...

🗐 Save All

Project

Choose Files to Save... Submit to Source Control...

₩ New Project... W Open Project... W C++ Class... Package Project

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

Add C++ Class

| | Show All Classes |
|--|------------------|

Refresh Visual Studio Code Project

Open Visual Studio Code Cook Content for Windows

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

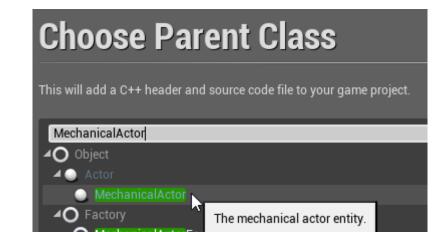


Window

Help

Ctrl+P

X



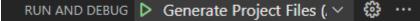
4. Click "Next" and you should see a name choosing dialog. Adjust the name of the class as needed and proceed by pressing the green "Create Class" button at the bottom:

| 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 | I and a (Runtime) - | Public Private | |
| Name Path | MyMechanicalActor | lana a (Runtime) √ | Public Private Choose Folder | |
| Path | | kan a kantime) ▼ | | |

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:

| ✓ Source | |
|-------------------------|---|
| | |
| C Build.cs | |
| ۥ | |
| C Harden and the | |
| € Foo.cpp | |
| C Foo.h | |
| G MyMechanicalActor.cpp | А |
| C MyMechanicalActor.h | А |

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 APPARATUSMARK_API AMyMechanicalActor : public
AMechanicalActor
{
    GENERATED_BODY()
};
```

```
o MyMechanicalActor.cpp:
```

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

#include "MyMechanicalActor.h"

8. Now you can override one (or multiple) of the Tick methods and implement your actual Mechanics as usually in C++...

```
\circ \ \ldots in the header:
```

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

 \circ ... and the source file:

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

From: http://turbanov.ru/wiki/ - **Turbopedia**

Permanent link: http://turbanov.ru/wiki/en/toolworks/docs/apparatus/mechanical?rev=1623415715



Last update: 2021/06/11 15:48