Detail

Details are the main building data blocks in Apparatus. They are high-level entities (unlike Traits), which support some additional ECS+ functionality like multi-iterating and inheritance.

Details do derive from **Dobject** and are subject to garbage collecting and Unreal's general memory model (unlike Traits which use their own memory organization).

If you need to change some of the Detail members, you can do it *directly* via members' assignment or calling non-const Detail's methods. No need to re-apply or copy the Detail data again.

As an optimization of some internal logic, Details can't really be removed from Subjectives. They can only be *disabled*. This is effectively the same thing as removal since Filters do respect this enabled/disabled state both in including and excluding contexts.

Creating Details

C++ Workflow

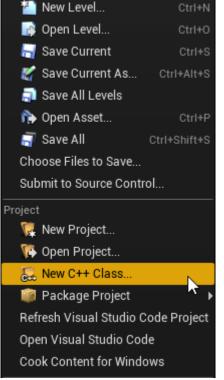
In order to create a Detail visible in your C++ source code you have to do the following:

File Edit

Load and Save

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:



Window

Help

Detail

Add C++ Class	×
	Show All Classes

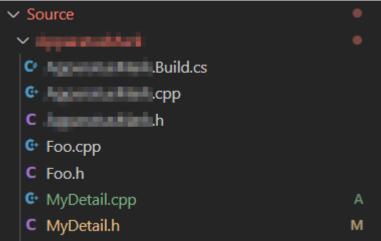
3. Now you can select any of the base classes available including the Apparatus ones. Choose the Detail as a base class:

Detail
▲O Object
▲ 🍚 Actor
⊿🛒 Brush
⊿ 🐺 Volume
🌞 LightmassCharacterIndirectDetailVolume
O Detail
▲O Factory The base subjective data class. O DetailF

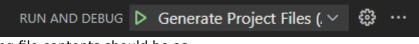
4. Click "Next" and you should see a name choosing dialog. Adjust the name of the class as needed and proceed by pressing the green "Create Class" button at the bottom:

Name Your New Detail						
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	MyDetail	ApparatusMark (Runtime) -	Public Private			
Path	Source/		Choose Folder			
Header File	/Source//MyDetail.h					
Source File	/Source/=/MyOetail.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:

```
∘ MyDetail.h:
```

```
// Fill out your copyright notice in the Description page of
Project Settings.
#pragma once
#include "CoreMinimal.h"
#include "Detail.h"
#include "MyDetail.generated.h"
/**
 *
 */
UCLASS()
class MY_API UMyDetail : public UDetail
{
 GENERATED_BODY()
};
```

• MyDetail.cpp:

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

#include "MyDetail.h"

8. Now you can add some data fields to the class as usually, right in the C++ header:

float X = 0; float Y = 0;

 Perhaps you would also like to expose those fields as properties in order to access them from BPs and even change their initial values through the UI (for more info, please see
 Properties):

```
UPROPERTY(BlueprintReadWrite, EditAnywhere)
float X = 0;
UPROPERTY(BlueprintReadWrite, EditAnywhere)
float Y = 0;
```

10. Your C++-friendly Detail should be ready to use now. Please, refer to API Reference for additional information.

Detail

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

Permanent link: http://turbanov.ru/wiki/en/toolworks/docs/apparatus/detail

Last update: 2022/01/05 13:59



4/4