ACE COMBAT 7

Introduction to UE4SS Blueprint Modding

By M4rkoz @ G[B]S

As a new era of AC7 modding has come to a start, I am most honored to introduce you to the UE4SS aspect of the modding scene. With this powerful tool, we have gained potential to create mods that will, without exaggeration, turn the game into something unrecognizable.

If you have any questions, want to learn more about modding or just want to meet a bunch of people who like AC7 and modding it, consider joining our Discord server: https://discord.gg/get-home-be-home-stay-home-280590586321567745

REQUIREMENTS:

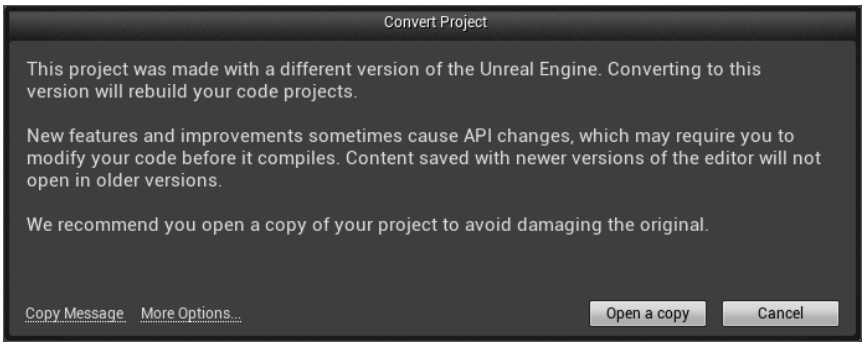
- Unreal Engine 4.18
- UE4SS AC7 build - https://github.com/kosnag/RE-UE4SS/releases/tag/release
- AC7 UE4 project template - https://drive.google.com/file/d/1_zbPS67ONVvKIjDL7JoUbRPNenySS6rP/view
- DefaultConfig.ini, for keybinds - https://drive.google.com/file/d/1EDQi7Z_oCAZdTPRMJYSwjxAHnCUcLO5i/view?usp=sharing

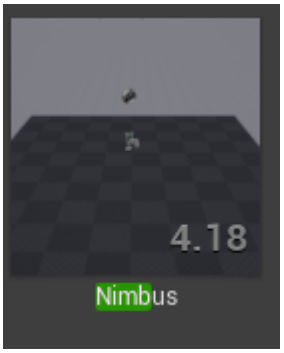
Part One

Getting started

First of all, of course, you need to download Unreal Engine 4.18 from Epic Games Store. Assuming that you know how to do it/already have it, moving on.
Second, you need the AC7 build of UE4SS, linked above. This is the tool that will allow you to load the mods into the game, since they can't be loaded simply by putting them into the ~mods folder. Download UE4SS and unzip it into the game's root folder (**steamapps/common/ACE COMBAT 7**). in **Game/Content/Paks**, create a folder named **LogicMods**. This is the mod folder for UE4SS mods, and all of those go there.

Third, download the UE4 project template linked above. Extract it to your desired location and open the .uproject using UE4.18.

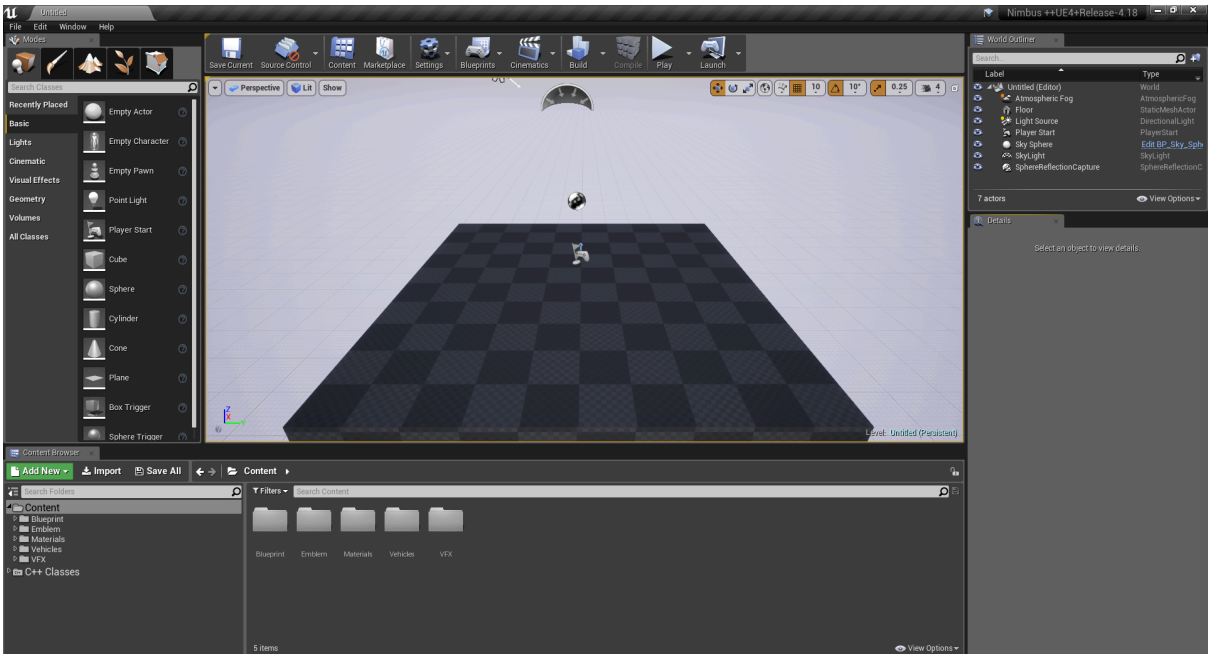




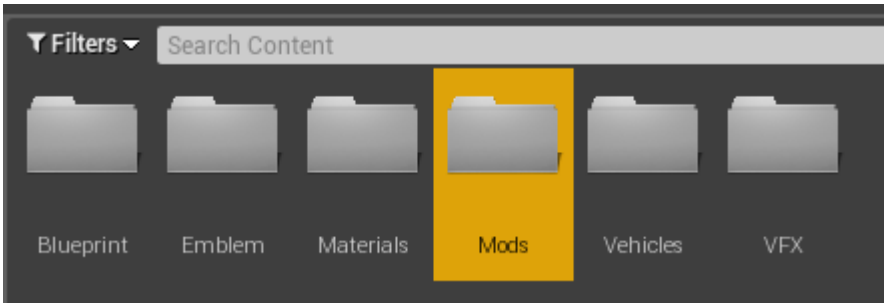
The project icon will be grayed out with a label that says “4.18” and the launcher will ask you to convert the project. Don’t mind the popup and just press “Open a copy”. It will take a couple minutes to create the copy and open it; UE4 is inherently kinda slow, don’t rush it.

The aforementioned template is necessary for UE4SS modding, as it contains all the AC7-specific classes and functions that you’ll need. Special thanks to GreenTrafficLight, Kosnag and Dantofu for this multipurpose template.

And a few minutes later, the project is open.



In the very root of the project, create a folder named **Mods** (yes, not LogicMods). All the work with your mods will be performed in there.



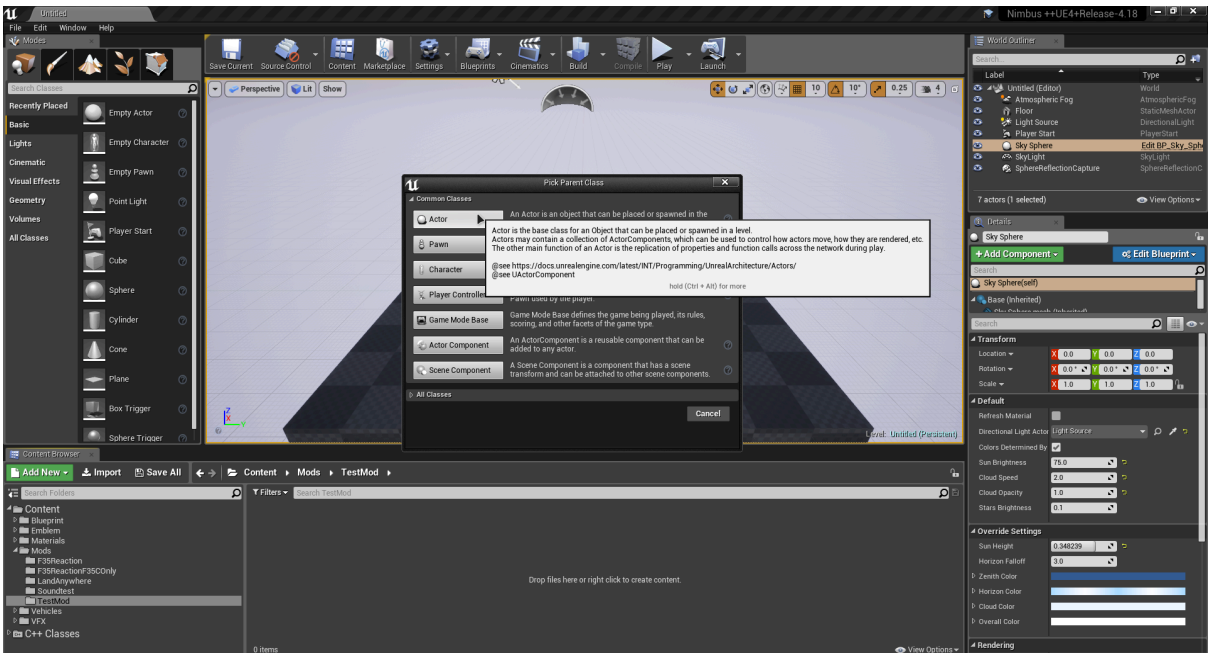
Inside the Mods folder, you will need to create a new folder for each of your mods. The folder name should be the same as the mod name and the .pak name. If the .pak name and the folder name don't match, the mod won't work.

Part Two

Enter ModActor

Now, you can start working on the mod itself. Of course, you need a folder for the mod with its name. For the sake of demonstration, the folder, and consequently the mod, will be named TestMod.

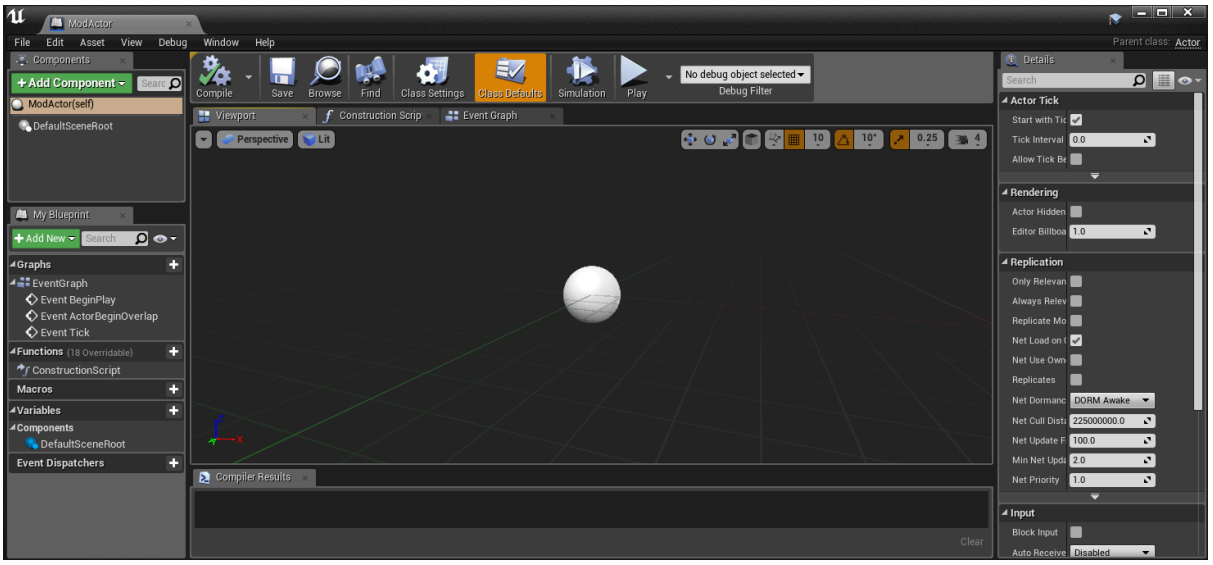
Inside that folder, you need to right click the content browser and create a **Blueprint Class**, then select **Actor**.



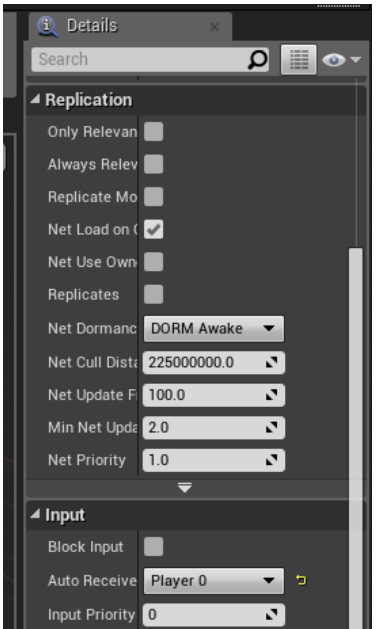
An actor named NewBlueprint will be created. You need to rename it to **ModActor** regardless of the mod name. **If the actor is not named ModActor, UE4SS will ignore it.**

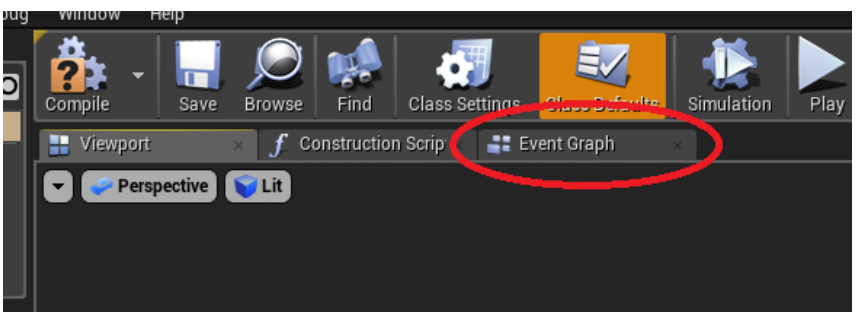
Now, this is where the main part starts. Unlike the other aspects of AC7 modding which we already are familiar with, it is quite intuitive and not too hard to understand, especially if you have prior experience with programming languages. You won't need any of those here, but knowing the basics of, for example, C++ definitely will help.

Upon opening the ModActor, you will be greeted by a new viewport containing a sphere.



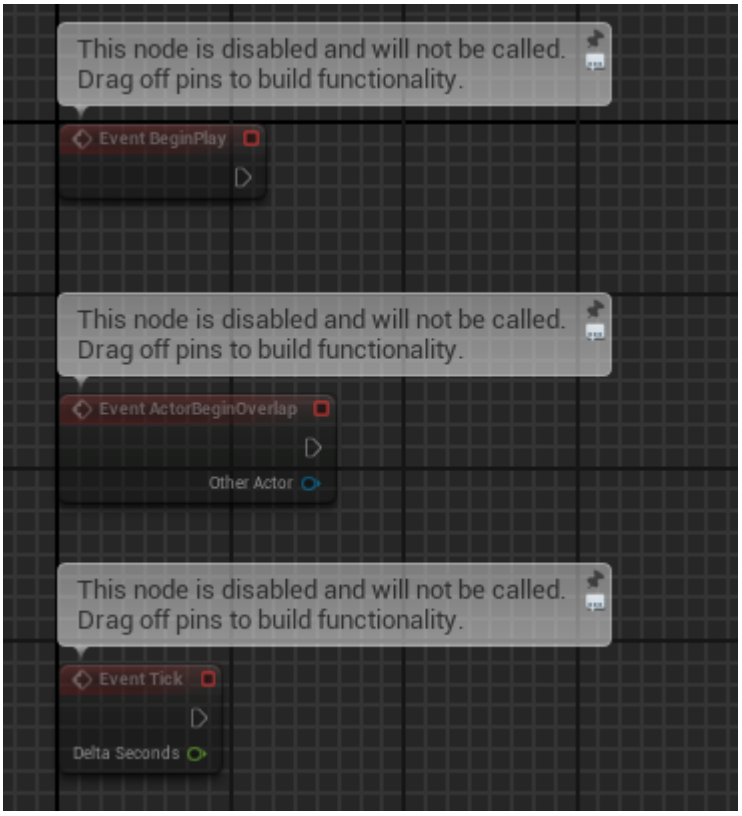
That sphere is not a mesh or a physical object, but the root of the actor. Don't mind it.
In the right part of the window, in the Input section, select Auto Receive to **Player 0** in order to receive keyboard input from the player. That is the local player's internal ID. So, even in multiplayer, your copy of the game will always see you as Player 0.





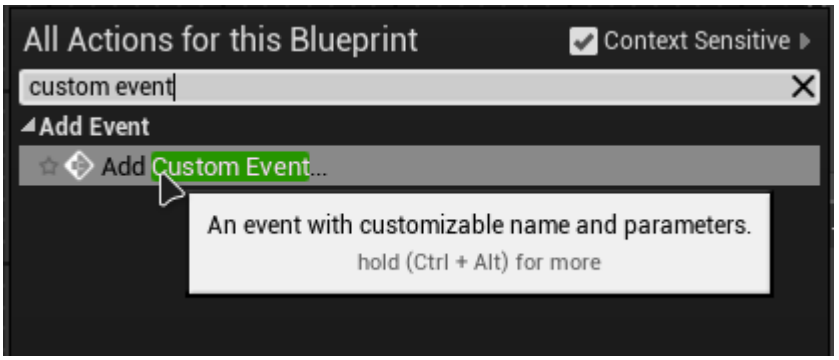
To actually edit the blueprint, near the Viewport tab, open the **Event Graph** tab. That is where the main process happens.

There you will see three nodes: Event BeginPlay, Event ActorBeginOverlap and Event Tick. You don't need ActorBeginOverlap, feel free to delete it.

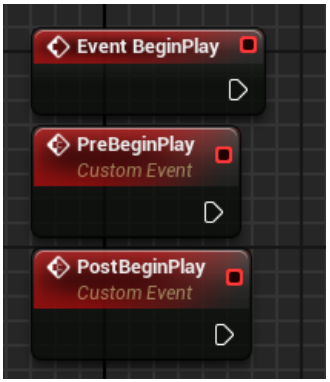


The important ones are **Event BeginPlay** and **Event Tick**. They are needed most often, but you can delete the one that you don't need depending on your mod's purpose.

Before we discuss the purpose of these nodes, which is quite straightforward, I need to mention that together with BeginPlay, you need to create two **Custom Events**. To do that, right click on the empty space and enter "Custom Event" in the search bar.



Create two custom events, and name them **PreBeginPlay** and **PostBeginPlay** respectively.



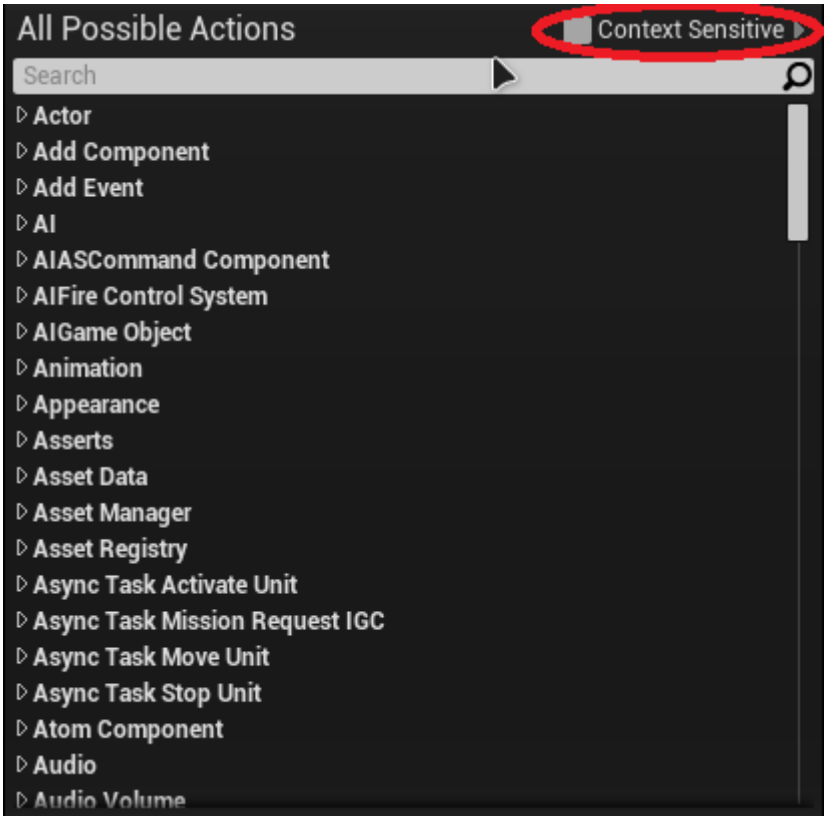
These two are **custom events that will only work with UE4SS. If you use them for non-UE4SS mods, they won't work.**

If you need your mod to perform an action as soon as it loads (when the game opens, when the sortie starts, when the game state changes), connect the action's node to all three of these events.

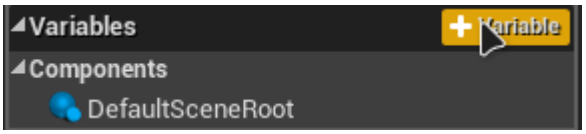
Then there's Event Tick, which is triggered on every in-game tick - basically constantly, without a stop. If you want the game to constantly check for something or similar, this is the one that you want to use.
And of course, you can use both of those in the same Actor.

Now for the actual blueprint scripting, that is the part which one needs to figure out for themselves, using their knowledge of the usual code structure and available references (you can see some in the G[B]S Discord server). Though, I can give some tips that you might find helpful.

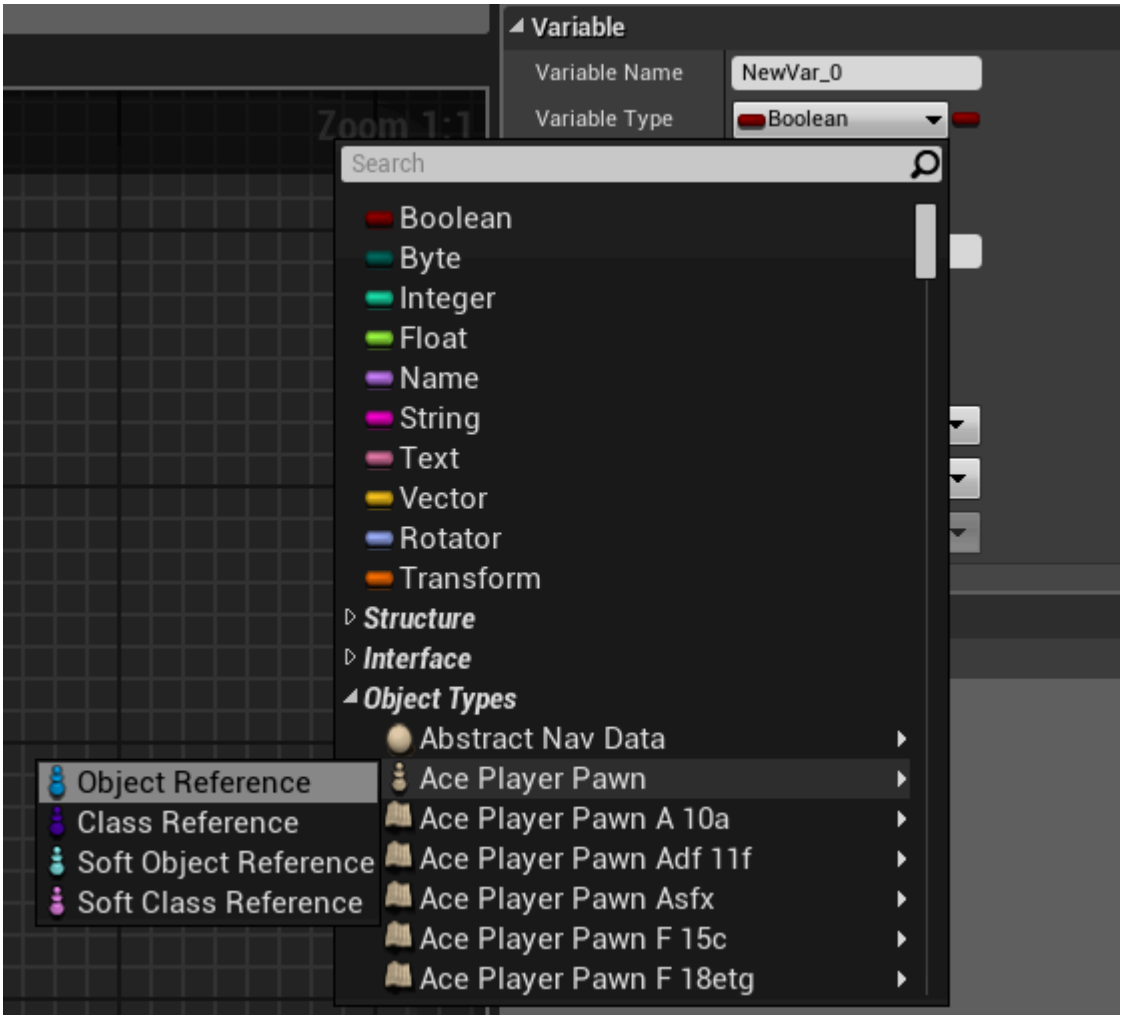
- In the right-click menu, you might have to disable "Context Sensitive" quite often. That will let you see a lot more variables and functions which normally would not be shown.



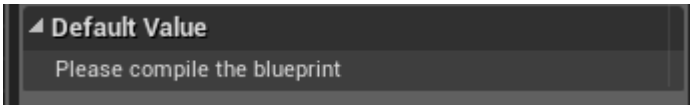
- You can use any kinds of objects in the ModActor: Meshes, sounds, widgets, anything. You can store them as variables too.

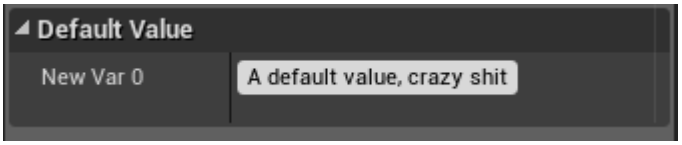


You can also select the variable type and assign any object type to it (image below).

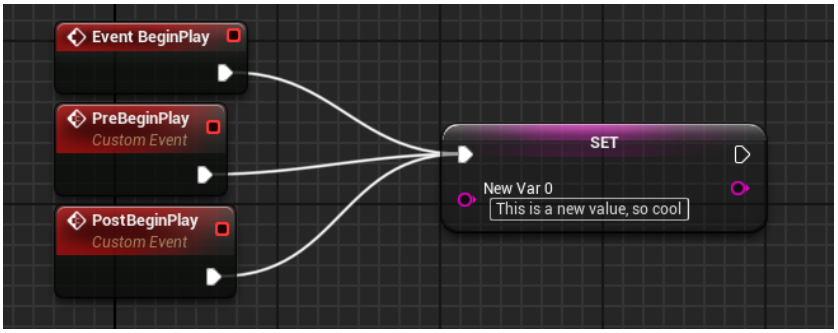


Upon compiling the blueprint (next to the Save button), you'll be able to assign default values to variables.

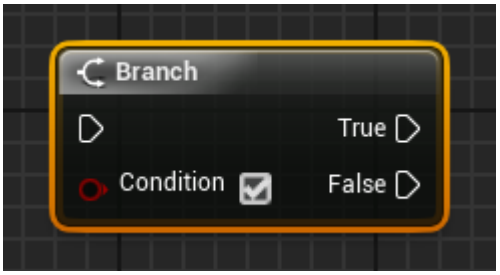




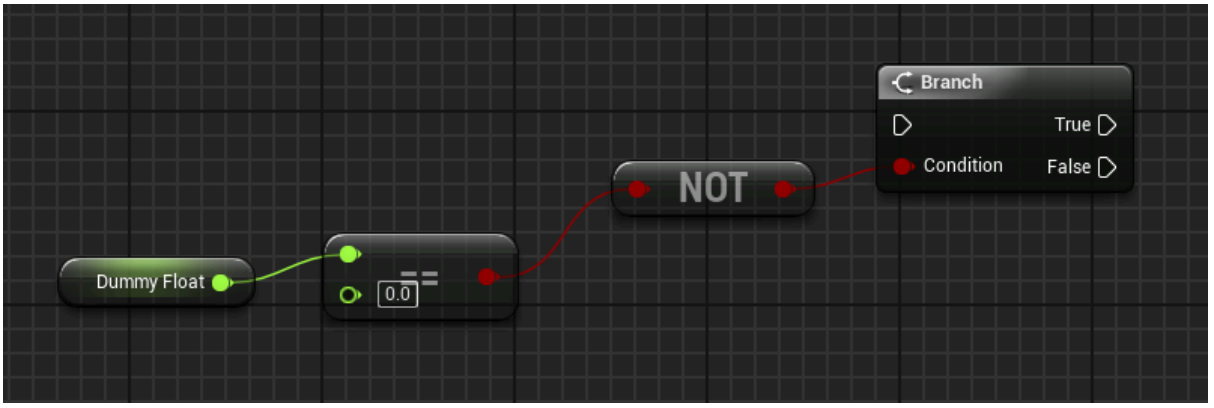
- To use your variables in the graph, drag and drop the variable into the graph area and press **Get [varname]**. To manually change its value, do the same thing, but press **Set [varname]** instead. The setter node should be connected to an event thread.



- An important thing: You can give your mod a description, author name and version. To do so, you need to create 3 string variables: **ModAuthor**, **ModDescription** and **ModVersion**. Compile and edit their respective default values to contain the necessary info.
- Most number values in the game are Float values. You can use float comparison nodes (**Float > Float**, **Float < Float**, **Float == Float**, etc) to compare them.
- If you have an “If…else” scenario, use the Branch node. It requires a condition - a boolean, and outputs two threads, one in case the bool is true, and another if it’s false.

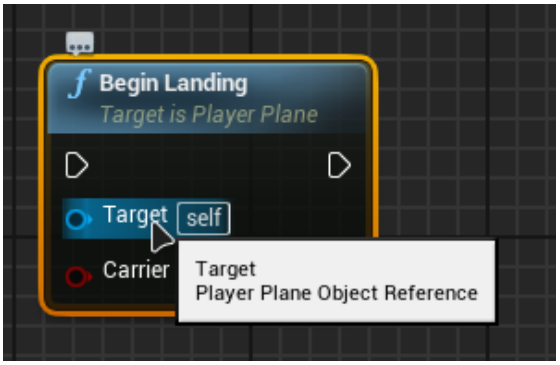


- If you want to get the opposite of a boolean, use the **NOT Boolean** node.



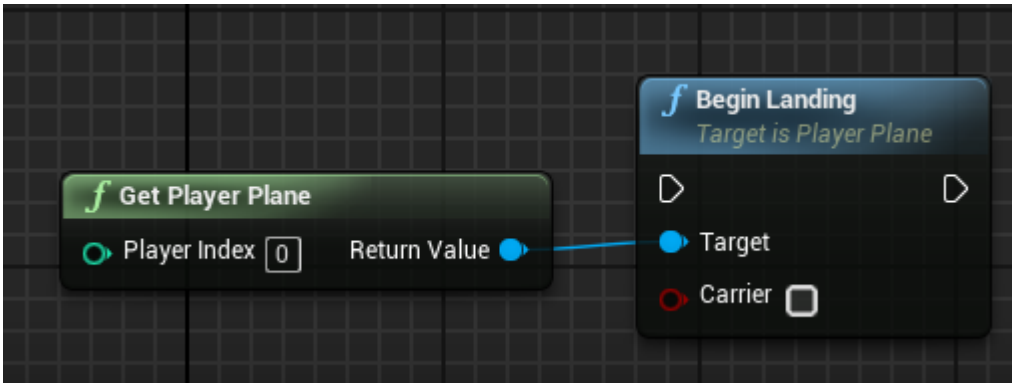
This branch node will check if the dummy float is not zero. If it’s not zero, it will move on to the “True” thread, and if it is zero, it’ll move on to the “False” thread.

- While scripting your ModActor, you will have to use functions along with the variables.
  You can find the function in the right click menu, they are most often denoted with the symbol *f*. If you hover your mouse above the function, you will see its description and what kind of object they need to work with (target). For example, the Begin Landing function, which initiates the landing sequence, requires a Player Plane object.



You can retrieve the Player Plane object by using one of the two methods:
First method - this function in the Nimbus Blueprint Library category:

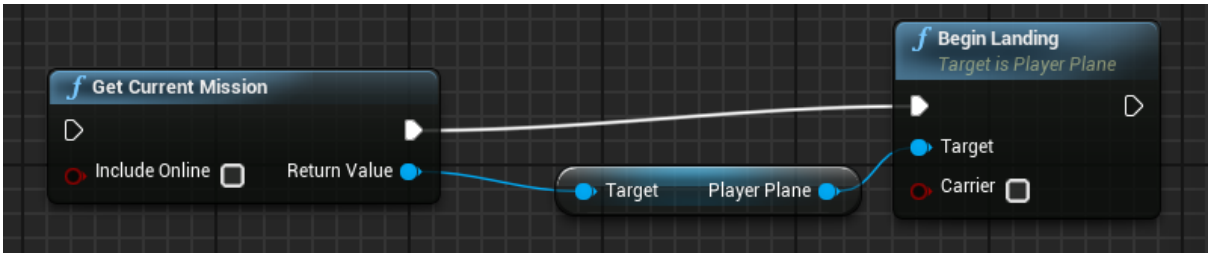
The local player plane's ID is always 0.



Second method - this node in the Mission category:

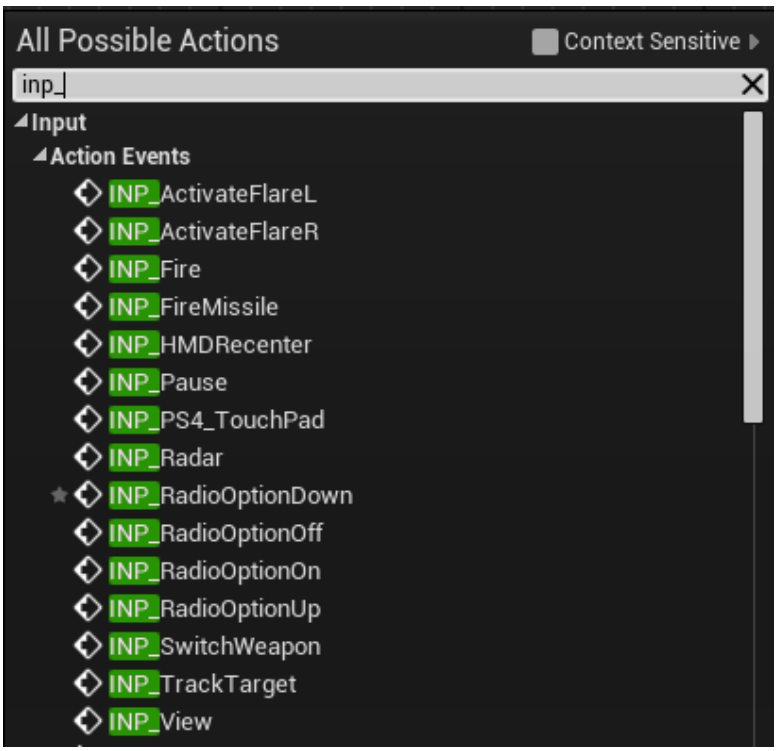
It will need a Mission object, which can be retrieved by the following node:

The nodes would be connected this way:



The game will get the current mission, get the player plane from it, and enable the landing sequence for it. You don't have to tick Include Online for it to work in multiplayer.

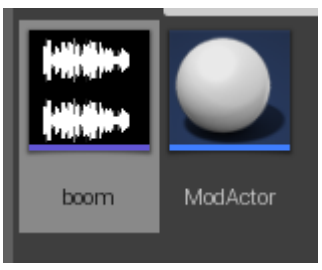
- I suggest not ticking the carrier landing/takeoff bool. Due to how carrier takeoffs work, they won't be possible outside of actual carrier landing minigames.
- To bind actions to player input, you can use the INP_ nodes. They will only show up if you put the DefaultConfig.ini (linked above) in the project's files: **Content/Config**.

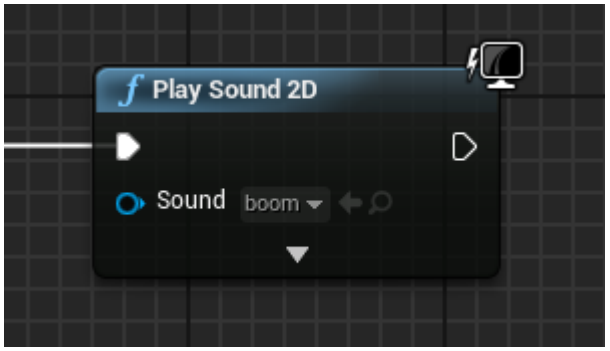


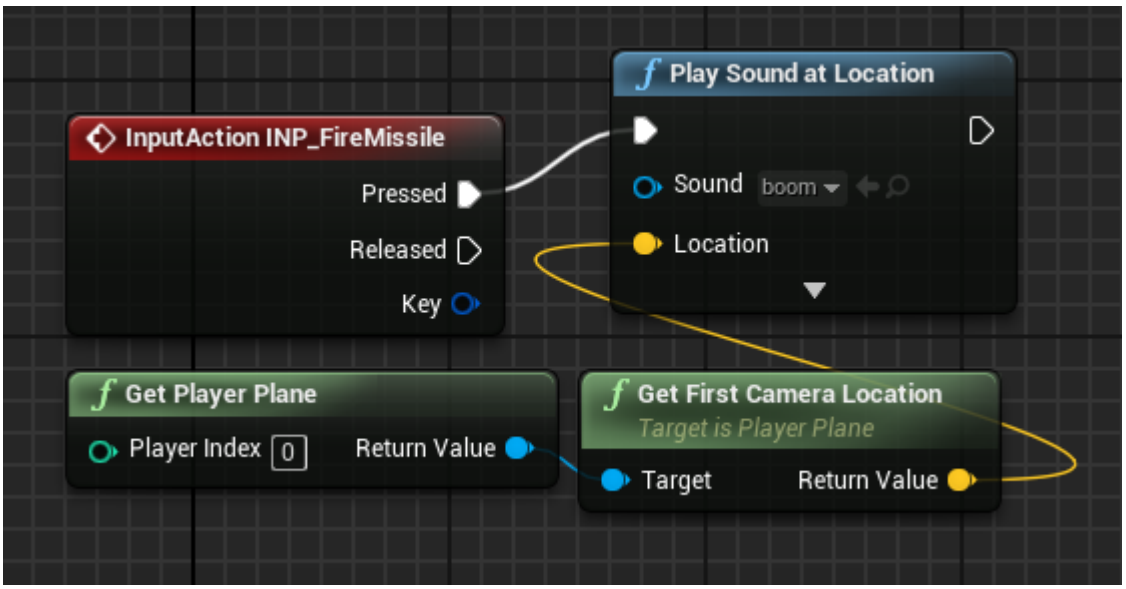
INP_ nodes are events. They don't need to be connected to anything on its left side. They will only activate when the key is pressed and/or released.

Do keep in mind that this will override the keybind's main function, so try to use something that's rarely used, like the RadioOption keys.

- You can also bind new sound effects to events. It's as simple as just importing a sound as a .wav file, setting compression quality to 100 (highest quality), and playing the sound in the ModActor with the **Play Sound 2D** or **Play Sound at Location** node.







- Before saving the ModActor, you need to compile it. So, compile, then save.
- You can find some useful information in Youtube guides.
- Or you can ask around in the aforementioned G[B]S server. You'll find a lot of helpful info in there.

Extra: Function glossary

List of functions

Various functions that you'll encounter in the blueprint will be explained here.

- Get Player Plane under Nimbus Component Library: Get the player plane object by the player ID (local player is player 0).
- Get Player Controller: Get the player controller by player ID.
- Get Player Controller ID: Get the player controller ID from a player controller object.
- Get Player Pawn under Game: Gets the player pawn.
- Get Display Name under Key: There are two functions of the same name; One retrieves an object's display name, other retrieves a class's display name.
- Get Current Mission under Mission: Allows you to retrieve info from the mission.
- Get Condition Checker under Mission: Allows you to check the mission conditions. Needs current mission as a target.
- Everything under the Mission Condition Checker category: Use the aforementioned Condition Checker to check conditions like current MP rank, distance between the player and an actor of choice, etc.
- Kmph to Mps, Kmph to UUps, Meter to UU, Mps to Kmph, Mps to UUps: unit conversion (UU - Unreal Unit, UUps - Unreal Unit per second)

- Create Widget under User Interface: Creates a widget (UI element) of choice and prepares it for enabling.
- Stereo Add Widget to Viewport under Nimbus Blueprint Library: Render a created widget of choice on the screen.
- Stereo Remove Widget from Viewport under Nimbus Blueprint Library: Remove the widget of choice.
- Play Sound 2D under Audio: Just play a sound effect.
- Play Sound at Location under Audio: Play a sound effect at a source location.
- GetActorLocation under Transformation: Retrieve a chosen actor's location.
- Get Speed Kmph/Mps: Get a game object's speed in respective units.
- Enable Player Input: Enable or disable (depending on the In Enable bool) player input; Target is player plane.
- Begin Takeoff under Player Plane: Initiate takeoff phase for a target player plane object. Enabling carrier takeoff is not recommended. Don't mind the float values.
- Begin Landing under Player Plane: Initiate landing phase for a target player plane object.
- Disable Weapon System under Player Plane: Disable player weapons, similarly to the mission 444.
- Is In Search Light under Player Plane: Check if the player has been spotted by a searchlight in Cape Rainy Assault.
- Is Weapon System Disabled under Player Plane: Check if the weapons are disabled.
- Is In Tunnel under Player Plane: Check if the player is inside a tunnel.
- Is On Ground under Player Plane: Check if the player is colliding with the ground.
- Is In Space Elevator under Player Plane: Check if the player is flying up the space elevator.
- Camera View Changed Event under Player Plane: Event that is triggered when player camera view is changed (First person, cockpit, third person, replay, minigame camera, impact camera, VR camera, no camera).
- Set Control Type under PlayerPlane: Switch between Expert and Novice controls.
- Set Stall Altitude under Player Plane: Set stalling altitude for a player plane object.
- Get Player Weapon Activator under Player Plane: Gets the information about the plane's armament; Target is a player plane object.
- Get Flare/Gun/Main Weapon/Special Weapon Count under Player Weapon Activator: Gets the plane's weapon count (Flares, gun, MSL, special weapon).
- On Weapon Fire/On Weapon Fire Pressed/Released under Player Plane: Fire the guns; Target is a player plane object.
- On Weapon Fire Missile/On Weapon Fire Missile Pressed/Released under Player Plane: Fire a missile; Target is a player plane object.
- Get Gun/Weapon/Main Weapon/Special Weapon Name String under Player Weapon Activator: Gets the name of a player's weapon as a string; Target is player weapon activator.
- Toggle Special under Player Weapon Activator: Switch to the special weapon.
- Set Output Mode: Changes filter of most of the sounds in-game during gameplay (available modes: Non VR, VR, VR Airshow).
- Check and Apply Vr Render Settings: Changes some render settings especially for output into VR headset (usable with UEVR injector).
- Get Selected Plane ID: Get integer ID of selected aircraft. Takes value from PPDT

- Plane IDTo Plane String ID: Converts integer ID into string ID of aircraft according to PPDT
- Get Selected Plane SWP: Get integer ID of selected SWP. SWP1 - 0, SWP2 - 1, SWP3 - 2.
- Get Selected Plane Skin No: Get integer ID of selected skin of aircraft. Osean - 0, Erusean - 1 and etc

Part Three

Finishing up

Once you are done with the mod, press File => Cook Content for Windows.
Cooking and packing happens the same way as usual. Create a folder that shares the name with the mod, inside the folder create a folder structure **Nimbus/Content/Mods/[The folder of the mod with the cooked mod files]**, and pack the mod folder. Put the .pak in the LogicMods folder and you're ready to test/use the mod.

And you're done!

You have finished your custom blueprint mod for UE4SS.

You don't have to worry about compatibility with other UE4SS mods as much as when making a regular mod. Though do keep in mind that you should not rename the .pak, or else the mod will not work.

That concludes the guide. Thank you for sticking around, I hope the explanation was clear and understandable.

The guide is subject to change according to on new discoveries and feedback.

Special thanks to Kosnag and GreenTrafficLight for giving AC7 UE4SS modding a headstart and putting great effort in their research.
My Discord: m4rkoza7