

# Interact And Possession System

Hello and welcome! Here you can read about the Interact And Possession System and learn more about it. Whether you just bought it or are still unsure if it's worth the purchase, you will find this to be a useful read. The Actor Possession System is a component that can be dropped into virtually any pawn actor, and with a few naming conventions followed allows you to let other actors possess this actor by interacting with various components in it, and then to exit it also using various components in it. Also included is an interface for interacting with other actors, and some demonstration code showing how this can be plugged into an actor so they can interact with things they are looking at. There is also a Look At Info system included that can be used to provide an actor with various information about what it is looking at, in case you want some better feedback in your UI. This system is also replicated to allow for use in multiplayer games should you require it. In order to best utilize some parts of this like the Look At Info system you will need to have some basic familiarity with blueprints to hook it into your UI. Read on to see examples on how to use these systems and what will actually be involved in making them work.

Among other things, this pack will also provide you with...

- A Camera Controller that allows a player to look around using a camera on any actor.
- An Actor Component and Blueprint Interface for sending interaction and look at info notifications between actors.
- A Possession component that can handle letting an actor be entered or exited (such as getting into a tank or helicopter or porta potty).
- A Look At Info system that lets you set up various types of information to be given to other actors on request, such as telling them what type of interaction they will be making should they interact with a given object/surface/component.
- A series of demonstration actors set up to help you understand and learn how to make proper use of these provided components and blueprint interface.

Additional Tutorials and Demo vids for other Packages can be found here:

<https://www.youtube.com/playlist?list=PLAgvMMfMQH2O1-YfsKXSwcFK2bvVg7UrP>

For info on **Multi Possession System** version (allowing multiple players in one vehicle) look towards the end for its area in the “updates” section.

If you get warnings for invalid input axis events, then you can create then or replace them with your projects own on the demo character in the project settings, inputs. This project demo character uses default third person content axis events.

## How To Use - Interact System

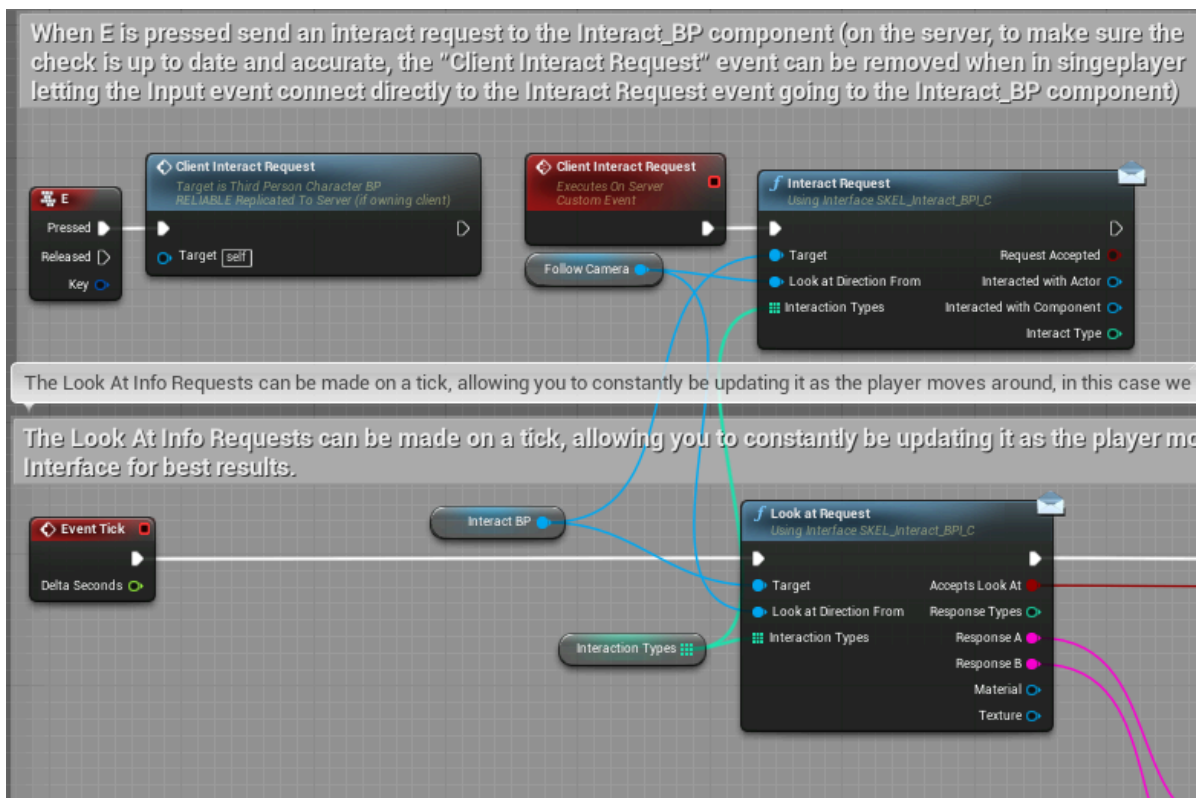
To be able to interact with anything, you'll need to set up an actor to make interact and look at requests first. Start by opening that actor, and then adding the Interact\_BP actor component into it.

Now you'll need to add a component (or use an already existing one) as a starting point for any interact traces. I'd recommend creating an arrow and putting it around chest or head height on your characters in most cases. Then you'll want to make sure it's got "Interact\_From" in its name. You can change the keyword used in the Interact\_BP components options if you want.

Now, go into the event graph, and graph and with a reference to the Interact\_BP component, create an Interact Request message, and if needed a Look at Request message node as well. Plug in a form of input into each one (interact event/button for the interact request in most cases, and a tick event into the look at request in most cases).

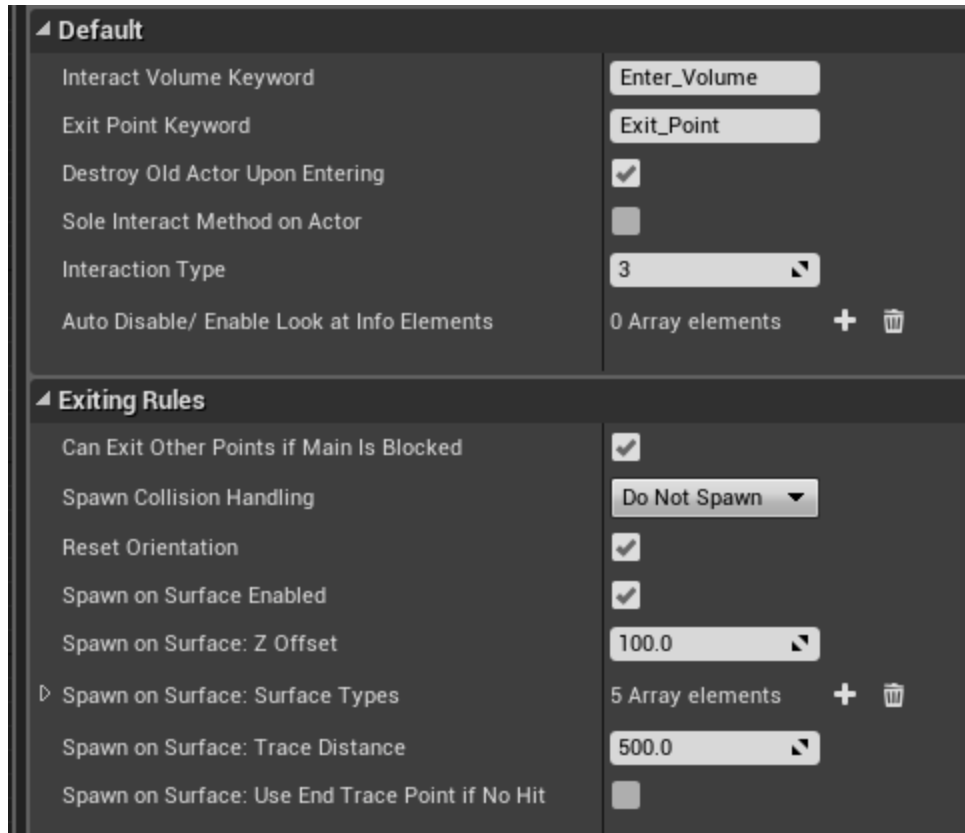
After that you'll need to give them a direction to make the traces in. In most cases you'll use the camera the player will be looking through, but you also might want to use the actors own rotation so it interacts with what's in front of it rather than where the player looks. I prefer the former option of the camera so I can potentially interact with things to my characters side just by looking at them though. The Interact\_BP component will use the forward vector from that component to figure out the direction to make the trace in.

And lastly, you'll want to plug in the interaction types this actor can make. It's just an array of integers that will only interact with other objects that accept one of those integers. So 1 could mean open doors, 2 could mean pick up or carry something heavy, 3 could mean get inside of something, etc... A normal character might only do 1 and 3, while a character that's gotten into an exoskeletal suit might only do 1 and 2 as he'd be too big to fit inside of anything else now, but strong enough to grab or move heavy objects.



## How To Use - Actor Possession System

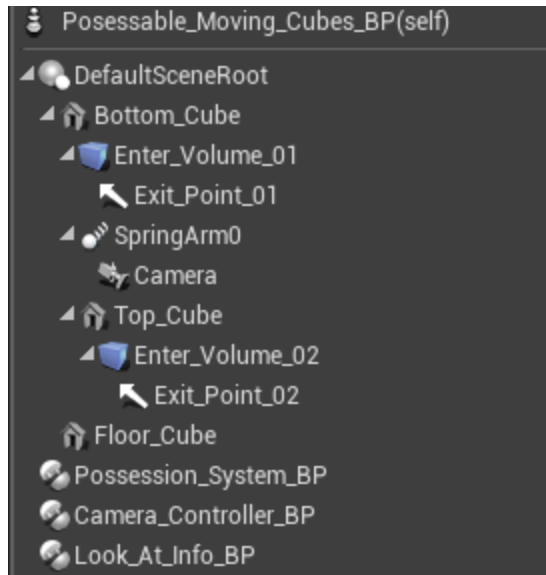
First, open the blueprint you wish to use the Actor Possession System in, and then add the Possession\_System\_BP component into it. Then, go and look at its details panel and set it up as you want.



Here is the default values of the main options. The system will look for any scene components (meshes, volumes, arrows, stuff that has a location in world space) with the given keyword in their name and assign them to the given category of components. Interact volumes are volumes that when interacted with, will allow someone to enter the parent actor. Exit points are components that mark the location that the player should spawn at when leaving the actor. I tend to use volumes for the Interact Volumes, and arrows for the Exit Points.

It is also worth noting that an interact volume should be linked to an exit point by following the keywords with “\_0X” where x should be a given number. So if you want someone who enters the left door of an actor to exit with the exit point in that area, you could add \_01 to both of their keywords as a suffix.

If you want someone to always leave from the same point first (like the driver's door in a car) regardless of where they entered from, you can use the same suffix on all the interact volumes. You can also enable the “Can Exit Other Points If Main Is Blocked” variable and then if you have other exit points it will start trying those if the initial one is not viable even though they don't have a matching suffix.



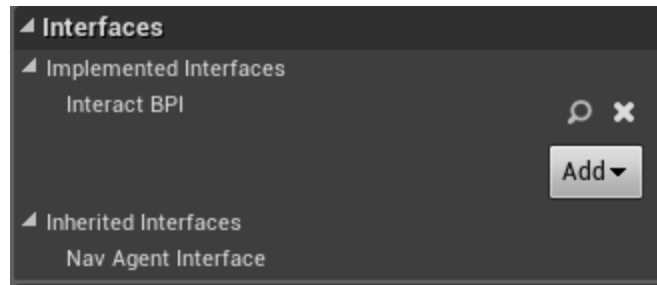
(Here is an example, showing an actor with two enter volumes and exit points, each prioritizing the exit point that was placed closest to it.)

There are a variety of other options to mess with for how you will spawn when exiting the actor, but setting up our interact volumes and exit points is the main part to focus on here. With a component or components that can be interacted with, such as a vehicles door or a box volume around it, and exit points such as arrows, we are mostly done setting up the components.

Now we need to let this component receive messages from others. At this point there are two ways to go. The first approach is, if you are only interacting with this component using the Interact\_BP that I made then you can go into the options and enable “Sole Interact Method On Actor”. This will make the interact system automatically send interact requests directly to the Possession system if it is used on an actor with a Possession system that has this option enabled. This will however keep you from getting that interact notification on the main blueprint and would keep you from using it for other things as well. Still, if the only interaction with this actor will be to possess it, then simply enabling this option can drastically save time. We’ll also need to be sure to set the Interaction Type variable if we enable this option though.

If we do want to have other forms of interaction or need the actor to know more about the interact notification for some reason, then we will need to set up the actor to receive the messages through a Blueprint Interface (and then to forward them on to the Possession system as is needed).

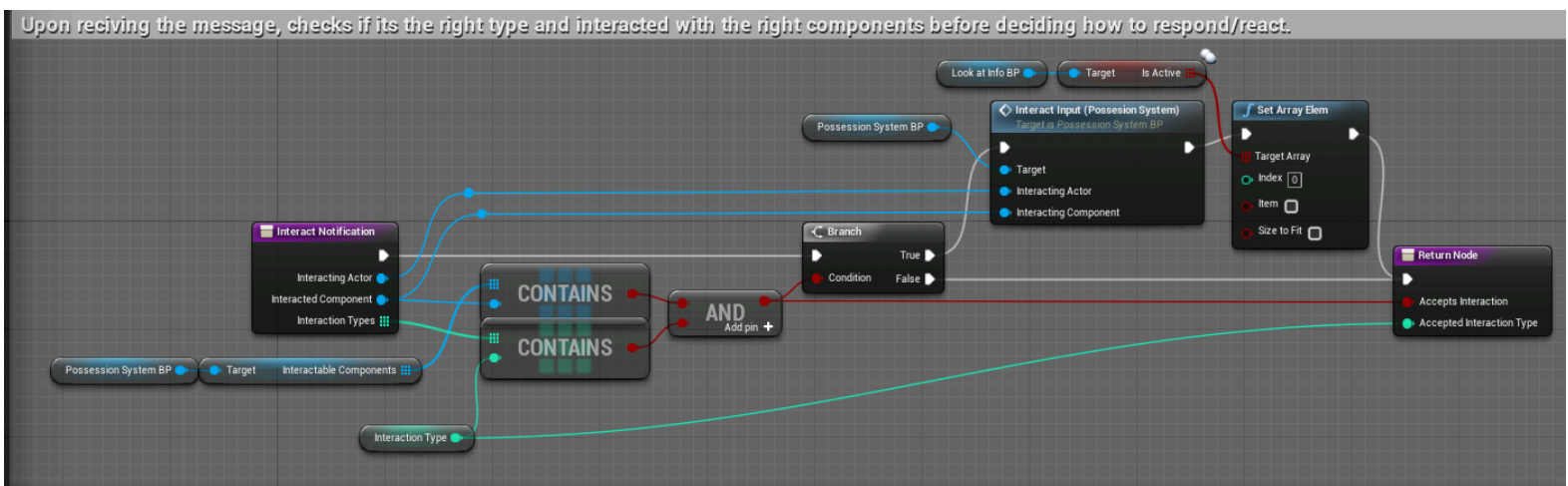
While not required as you can always use other methods of interacting with the actor to possess it, we’ll use it for this example as it’s what the Interact\_BP uses normally. In your actor go to Class Settings (shown in the bar at the top of the screen normally) and then look at the details on the right side after clicking it. Here you’ll see Interfaces, you’ll want to add the Interact BPI here, and then compile.



Now, look down onto the left side of the screen where variables and functions are normally located. You'll now see an area showing Interfaces. Open the Interact\_Notification function from here, and you'll get a new tab that lets you set up how your blueprint will respond to interactions.

You'll want to take the input values and then check them against the possession systems interact components. If the event interacted with an interact volume, then accept it and tell the Possession system about the interacting actor. You can also add other checks such as an interaction type requirement. The idea is you can set up a simple integer to represent a type of interaction, so you could use 1 to represent opening doors, boxes, etc, 2 could represent picking up an object, and 3 could be entering an actor. You can use whatever you want, but the idea is to make sure that the incoming interaction request is made by something that can interact with the Possession Controller (A large mech exosuit might still be able to open doors, but shouldn't be able to fit into a car anymore for example).

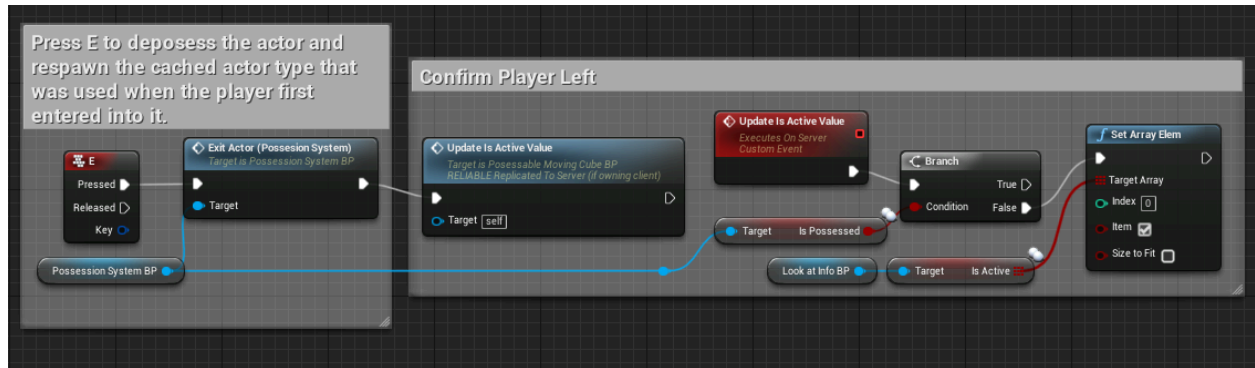
After figuring out if you can accept the interact request, send a reply with the results. You can find this all in the demonstration assets for easy reference and copying. You'll notice near the end it is deactivating an index of the Look At Info controller on this actor as well. That's so that if someone else looks at the actor, they'd not longer get the message telling them they can enter this actor through an interaction event since it's already occupied.



Now, if another actor ever sends an interact message to this actor, it will check to see if it should let them take control of it. Interaction messages can be tied in with a variety of methods,

but I'd recommend using some type of trace plugged into the direction an actors facing or its camera is facing, and then sending the component it was looking at with the Interact Message like the Interact\_BP component can do.

In order to leave, simply send an exit message to the Possession\_System\_BP component. If you have a Look\_At\_Info\_BP component as well, you can also confirm that you have left the possessed actor and then adjust the look at info active values so it will be saying that you can enter that actor again (if appropriate). Don't forget to do the update for the Look\_At\_Info\_BP component on all clients if its a multiplayer game too.



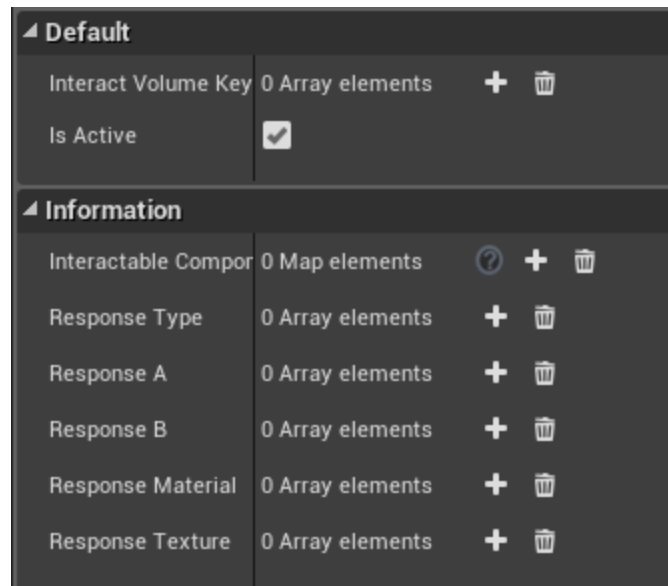
Also, since you can by default only really have one controller in a given actor, if you want to have “passengers” in a car or vehicle, I would recommend creating separate actors that are childed into the main car actor that would have interact volumes placed near the side/rear doors and with a camera set up over them so a player can enter and exit without directly controlling the main vehicle as a passenger.

Also, if you want to have multiple possible interacts on an actor (like interacting with two separate switches, plus an area to enter and possess it), then you’ll want to do a check against the interacted component for each one of these possible interacts should the previous ones fail (in other words the false return on the branch in the above picture would like to a similar check for interacting with another component/etc... and only return false on the return node if all failed to meet the requirements.

It’s worth mentioning that if you also use the Look At Info BP component on this actor you may want to use the “Auto Disable/Enable Look At Info Elements” option. This array lets you add integers as you need. And it will automatically disable/enable the Is Active element that you have added for a Look At Info component on the same actor. So if you want to not tell people they can enter a box anymore after someone else is already inside, then you can use this.

## How To Use - Look At Info System

In order to use the Look At Info component you'll want to take the actor you want to use it on and then add the Look\_At\_Info\_BP component to it. Now you'll want to go into the components details.



Like the Possession System this component will find Scene Components in its parent actor that contain certain keywords and save them into a list. This time you have an array of keywords though, you'll also notice you have an array of everything else as well. This is so that one Look At Info component can handle all the look at responses on an entire actor. The idea is that if you make a keyword in the first index of the Interact Volume Keyword array, then any components with that keyword will be saved with that index value attached to them. And then when getting a reply from that component, it will use that index value to get the response values from the response arrays.

So you could have two keywords, for a car the first could be used on all doors or Interact Volumes that you can enter an actor with, and the second keyword could be used on the trunk. Then all the response arrays would in their first index be given values relating to entering/driving the car, while those in the second index of the arrays could reference opening the trunk.

In order to actually use the Look At Info component now, you'll need to plug it in so other actors can see its contents easily. Just like with the Possession system you have two ways to go about this. The first one involves going into your Interact\_BP component on your interacting actor and enabling the "Send Look At Requests Directly To Look At Info Component" option. This will result in all look at info requests being sent directly to a Look At Info component if its on an actor. This can be much faster to set up, but if you want to access or modify the requests being sent in this actor then you'll want to route the interactions through the actor instead and leave this option disabled.

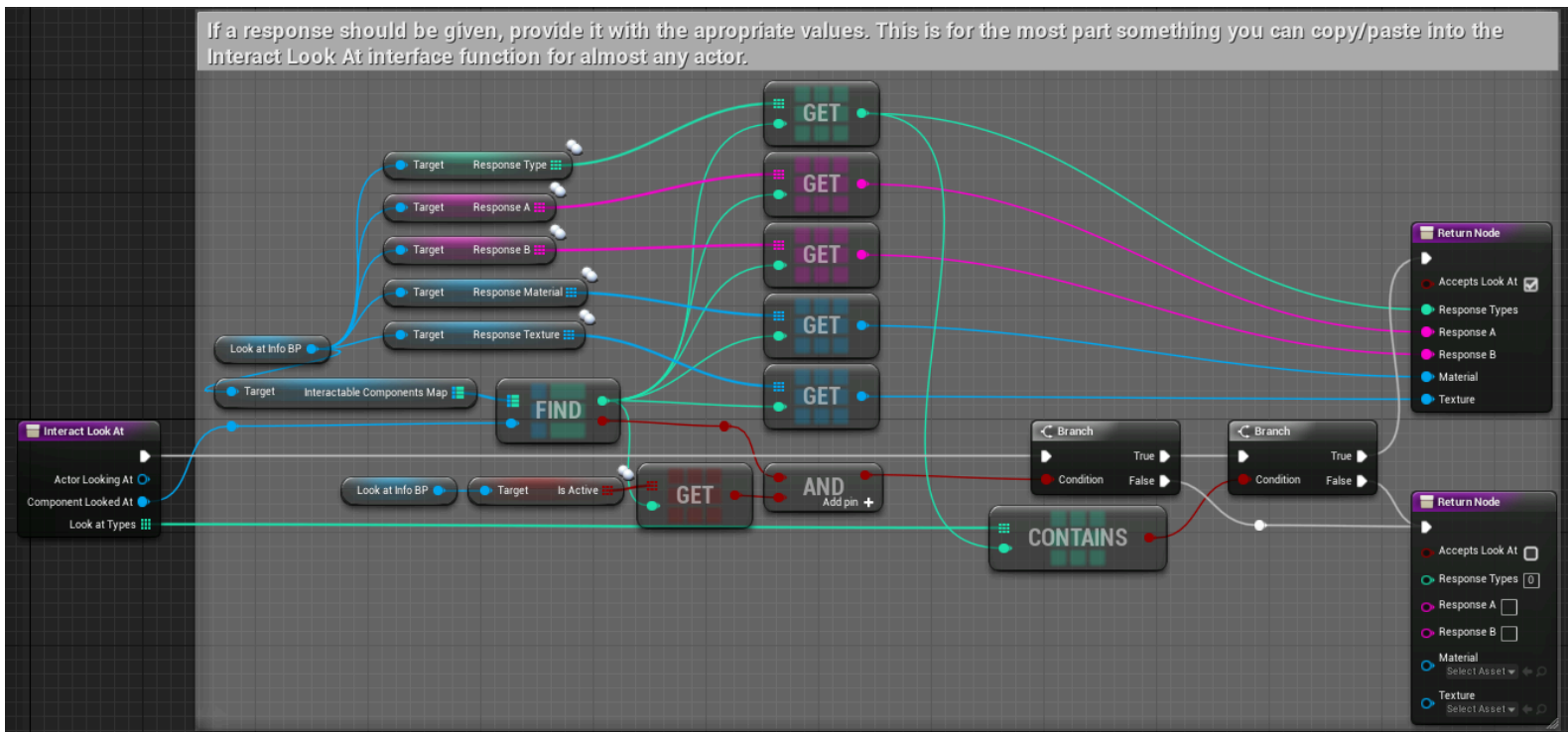


If disabled then you'll need to add the Interact\_BPI Blueprint Interface into the actor (if it's not already added). Instead of the Interact\_Notification function though you'll want the Interact\_Look\_At one. Open that up and then add the following nodes so that you can get the desired information from the Look At Info component and send it as a reply.

Here you can see the look at node having its arrays opened and finding the appropriate response to match for the given component that was looked at. If you are only using textures, or a single string or such in your UI then you don't have any need to add all of these different nodes as it won't matter if you send nothing back on the inputs that won't be used in your game. And should the looked at component not be found, then it will simply send back that it did not accept the look at request.

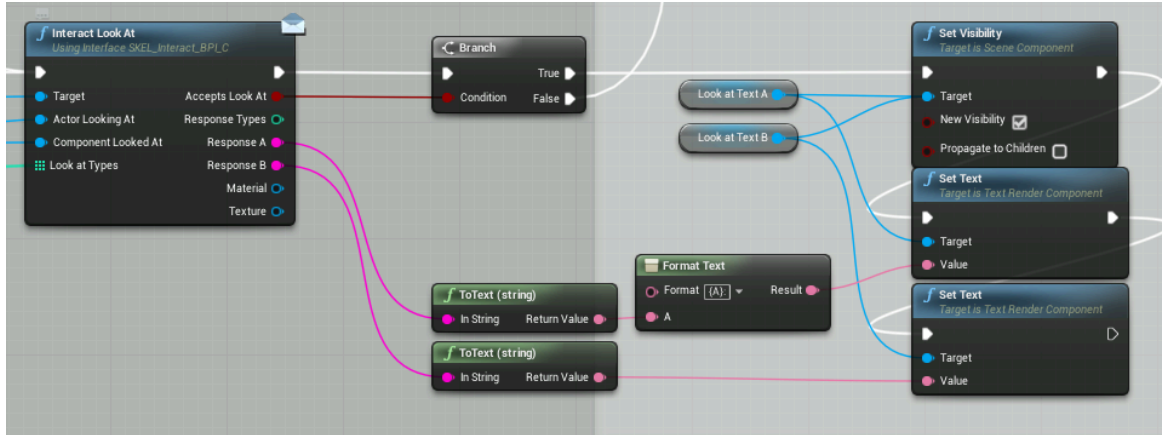
For the most part this section of code can be directly copied and pasted into all other Interact Look At functions on other actors that you may be setting up, so you don't need to worry about redoing them all a bunch of times anyway though.

If a response should be given, provide it with the appropriate values. This is for the most part something you can copy/paste into the Interact Look At interface function for almost any actor.



Setting up the look at on the characters end that sends the message is mostly like the method used to send interact messages. A trace, except this one fires continuously on the tick event and will send the received information to your UI whenever it receives a response.





Here's my little demo approach just making some text components visible with the response (it sets the text to invisible down the false path after checking other hits if they also don't return anything).

As of ~3/13/2018 I have also added a UI Widget to the project, that is also hooked up in the characters blueprint so you can also reference how to create UI Widgets and update values in them (or at least one of many methods to do so).

Short "how to" on basic UI Widgets can be found here: <https://imgur.com/a/7CQso> for those unfamiliar with them.

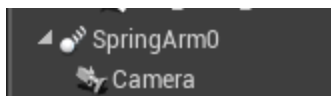
## How To Use - Camera Controller

The camera controller is effectively a component that can be used if you want a player to be able to control a camera (through a spring arm). This can be attached to an actor and, if it has the right components and naming, it will allow for the control of the camera/s. For it to work you should add a spring arm to your actor, and then a camera to that spring arm. The spring arm should have a number in its name, starting with 0 and going up to the total number of spring arms in the actor minus 1 (so count in base 0). For example, adding a spring arm named “SpringArm0” would let the controller move that spring arm around with its default settings.

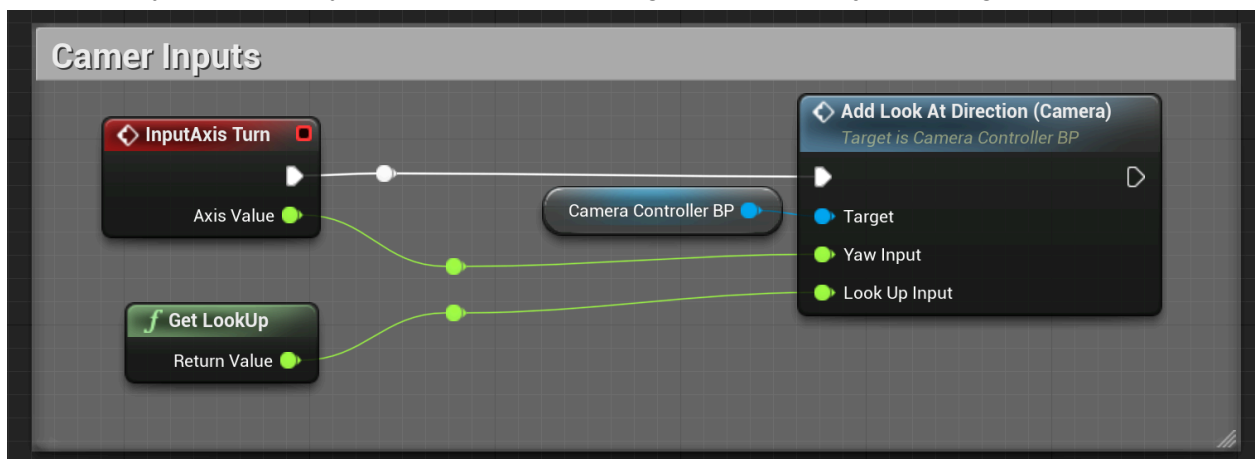
It is plugged in with a single input event (**Add Look At Direction (Camera)**), this event should be plugged into the input axis events needed to look around, such as the mouse y and x input axis events.

There are other options for controlling the sensitivity as well as the method of rotation you may want to look at. The multiplier variable will control sensitivity, and the Rotation Is Relative variable controls if the camera/spring arm will inherit rotation from the actors rotation, or if disabled ignore the actors rotation and only respond to the players inputs.

In other words 90% of the time just add this to an actor, make sure it's got a camera and spring arm like this...



And then plug in the following input event in the actors event graph so it can be told how to rotate based on your desired type of input axis mappings (set up in project settings, Input).



And bam... done. It is also recommended to make sure “Enable Camera Rotation Lag” is enabled on the spring arm to help smooth out movements on vehicles. Especially if “Use Relative Rotation” on the camera controller is set to false.

## Interact System Options

Default	
Interact from Keyword	Interact_From
Interact Trace Distance	300.0
Interact Trace Radius	25.0
Interactable Object Types	3 Array elements + 🗑️
Interact Trace Debug	None ▼
Look at Trace Debug	None ▼
Send Look at Requests Directly to Look at Info Component	<input type="checkbox"/>

### Default

**Interact from Keyword:** A string that must be contained in a components name for it to be identified as the component to make interact and look at traces from.

**Interact Trace Distance:** The distance of an interact trace (how far away you can interact with something, plus half the trace radius value).

**Interact Trace Radius:** The radius of the trace, this means you don't have to look exactly at stuff, but only in its slightly more general area, set it too high and it can become a bit clumsy though.

**Interactable Object Types:** This controls the types of objects the interact trace can hit, if you are using a special collision type for all interact volumes so they don't interfere with other things, be sure to add that here.

**Interact Trace Debug:** Debug option for interact traces.

**Look At Trace Debug:** Debug option for look at info traces.

**\*Send Look At Requests Directly To Look At Info Component:** If true, then look at info requests will be sent directly to a Look At Info component on a given actor, rather than sending them to the looked at actor itself where it can be manually forwarded. By enabling this option you don't need to set up the nodes on the Interact Look At function, but it will also mean you can't tie in any extra functionality through these messages on your own if you want to do that.

## Possession System Options

### Information

Interactable Components	0 Array elements +
Entered by Component	<div>None</div> <div>None </div> <div> </div>
Can Exit Actor	<input checked="" type="checkbox"/>
Exit Point Components	0 Array elements +
Cached Player Pawn Class	<div>None </div> <div>  + </div>
Is Possessed	<input type="checkbox"/>
Allows Entering	<input checked="" type="checkbox"/>
Last Exited from Component	<div>None</div> <div>None </div> <div> </div>

### Default

Interact Volume Keyword	Enter_Volume
Exit Point Keyword	Exit_Point
Destroy Old Actor Upon Entering	<input checked="" type="checkbox"/>
Sole Interact Method on Actor	<input type="checkbox"/>
Interaction Type	3
Auto Disable/ Enable Look at Info Elements	0 Array elements +

### Exiting Rules

Can Exit Other Points if Main Is Blocked	<input checked="" type="checkbox"/>
Spawn Collision Handling	Do Not Spawn
Reset Orientation	<input checked="" type="checkbox"/>
Spawn on Surface Enabled	<input checked="" type="checkbox"/>
Spawn on Surface: Z Offset	100.0
▷ Spawn on Surface: Surface Types	5 Array elements +
Spawn on Surface: Trace Distance	500.0
Spawn on Surface: Use End Trace Point if No Hit	<input type="checkbox"/>

## Information

Some of these are mostly meant to be referenced, rather than being manually set.

**Interactable Components:** An array of components that can be interacted with in order to possess the parent actor.

**Entered By Component:** The component that the currently possessing actor interacted with when it entered.

**Interacting Actor:** The actor that interacted with this actor and has now been destroyed and had its player controller moved into this components parent actor.

**Can Exit Actor:** An option that can be set to keep a player from leaving an actor by means of this system.

**Exit Point Components:** An array of all found exit points.

**Cached Player Pawn Class:** The class of actor that the current possessor previously was controlling, will be spawned when the player leaves this actor.

**Possessing Controller:** The controller that has possessed this actor through this controller

**Is Possessed:** Returns true if there is a player controlling this actor though use of this system. Visible for information/reference purposes only, avoid changing manually when possible (should be set to true if upon spawning in this actor would already be possessed by a player though).

**Allows Entering:** Set to true to allow for players to enter this actor, or false to keep them from entering.

**Last Exited From Component:** The last exit point that was used

## Default

**Interact Volume Keyword:** All components with this string in their name will be considered interact volumes that if interacted with can allow one to possess this actor. In addition adding "\_0X" with X being a number no greater than the total number of interact components will allow you to match an interact component to an exit point by adding the same string to an exit point. The underscore and zero are required prefixes to the used number.

**Exit Point Keyword:** All components with this string in their name will be considered an exit point. Should have “\_0X” as a suffix, with X matching the number used on an interact volume in order to be linked to a specific enter point.

**Destroy Old Actor Upon Entering:** If true then a controllers previous actor will be destroyed when they enter this systems parent actor, if false it will no destroy the old actor that is no longer being possessed but will send it a message "Controller\_Moved" so that it can respond to having its controller moved into the new actor. The actor must have the Interact BPI interface added to it to make use of this event. This event is called on the server, so set up the other actor to receive it appropriately.

**\*Sole Interact Method On Actor:** If true then any interact requests made by the Interact\_BP to an actor with this component will instead of being made to the actor itself will be made directly to this controller. Enable if an actor only can be possessed and nothing else through interactions to avoid needing to setup the nodes in the Interact\_Notification function that would forward an input to this controller, but only if this is the only type of interactions they should be able to make as it will no longer be sent to the parent actor itself.

**\*Interaction Type:** If "Sole Interact Method On Actor" is true then this component will use this value for its interaction type check on incoming interact notifications.

**\*Auto Disable/Enable Look At Info Elements:** You can add the element for a Look At Info Components Is Active array on the same actor as this component that you want to disable when this actor gets possessed through this system.

## Exiting Rules

**Can Exit Other Points If Main Is Blocked:** If true then an exit point other than the one with the same \_0X number in its name can be be used to let the player out should the primary one for that entry point be blocked.

**Spawn Collision Handling:** How to handle collisions when an actor is being spawned for the currently owning player controller to swap its control to while "exiting" this actor.

**Reset Orientation:** If true then when spawning an actor that leaves the possessed object, its orientation will be even with the world horizon, even if the exit points orientation was not.

**Spawn On Surface Enabled:** If true then when an actor is spawned to allow for a player to exit this systems parent actor, it will try to spawn it on a surface beneath the given exit point rather than right at the exit point.

**Spawn On Surface: Z Offset:** A vertical offset for how high to spawn the actor after finding a surface (default third person character for example has its pivot in its center, so it should be offset 100 units (1 meter) to try and spawn it with its feet at the found surfaces level. Adjust this value as is needed.









**Spawn On Surface: Surface Types:** The types of objects that will be traced for when looking for a surface below a spawn point if Spawn On Surface Enabled is set to true.

**Spawn On Surface: Trace Distance:** The maximum distance below the spawn point that the spawn on surface trace will look.

**Spawn On Surface: Use End Trace Point If No Hit:** If the trace does not hit anything, then you can have the player spawned at the end of the trace by setting this to true, or if false they will simply spawn on the exit point itself in this situation.



## Look At Info Options

Default		
Interact Volume Keyword	0 Array elements	+ 
Is Active	0 Array elements	+ 
Information		
Interactable Components Map	0 Map elements	? + 
Response Type	0 Array elements	+ 
Response A	0 Array elements	+ 
Response B	0 Array elements	+ 
Response Material	0 Array elements	+ 
Response Texture	0 Array elements	+ 

### Default

**Interact Volume Keyword:** Keywords for look at components, all components with a keyword will be added to the interact components map variable, showing which array from this index it matched. All replies for that component will give the reply value from that array index as well. (If it contains the first keyword in this array, it will send back the reply values from the first indexes in those reply arrays).

**Is Active:** True if the given index of this component is active and giving replies.

### Information

**Interactable Components map:** This map shows the index value for each component as it would match to the search keyword array, and reply arrays (auto builds on start).

**Response Type:** The type of response being given (matched to components containing the keywords in the keyword array).

**Response A:** Part A of the string response being given (matched to components containing the keywords in the keyword array).

**Response B:** Part B of the string response being given (matched to components containing the keywords in the keyword array).

**Response Material:** An array of materials that can be replied with for a given look at request matching a given keyword.

**Response Texture:** An array of textures that can be replied with for a given look at request matching a given keyword.

## Camera Controller Options

Camera/ Rotation Controls			
Camera Multiplier	0.65		
Spring Arm #	0		
▶ Cached Camera Rotation	X 0.0	Y 0.0	Z 0.0
Default			
Controller Active	<input checked="" type="checkbox"/>		
Use Relative Rotation	<input checked="" type="checkbox"/>		

### Camera/Rotation Controls

**Camera Multiplier:** Controls how fast the camera looks around.

**Spring Arm #:** Sets which spring arm in the scene to use as is determined by the number in its name (0 means use springarm0 for example). Spring arms names must use numbers starting with 0 and only going up to the number of spring arms in the actor minus 1 (so base 0).

**Cached Camera Rotation:** Holds the Current rotation of the camera/spring arm (relative).

### Default

**Controller Active:** Sets the controller to be active or inactive.

**Use Relative Rotation:** If true then rotations are handles relative to the owning actors own rotations (actor turns, camera turns with it), if false then it uses world rotation and the camera will only rotate based on inputs and no its actors own rotations.

## Contact And Things to Keep In Mind

First, always be creative. This package can be used for a lot more than just entering or exiting vehicles. You can use it to hide inside of porta potties, boxes, take control of cranes, defensive emplacements, or a number of other things. It might save you more time than you might originally have planned to use it for if you keep this in mind. Always be creative, for this and everything else you may have at your disposal, your imagination is the limit.

Also do keep in mind what other marketplace products you might want to use this with. Not all marketplace products work together after all, if you plan to use this in certain game mode types of projects such as the Survival Kit, you may find you'll need to tweak those systems to support the players actor being changed from the default type included in these kits themselves. What this may entail may differ heavily based on the particulars of the other system, so if you're uncertain be sure to ask the makers of your other package if it can support the players controller possessing different actors during runtime.

If you ever have any issues, bugs, questions about how something works, or even possibly requests for additional features then feel free to contact me at [Pineconedemon@gmail.com](mailto:Pineconedemon@gmail.com) and I will try to get back to you as quickly as I can. You can also find more of my stuff on the marketplace through my username, Sean Dachtler. I also have some tutorials on my Youtube (Terricon4) for some of my products. And I am for the record always interested in seeing what people might be making with my stuff, so feel free to toss me an email showing whatever cool things you've been doing with it.

I now have a discord channel here as well where you can chat with others or ask questions. <https://discord.gg/d7paEng>

## Limits of Possession System

The possession system can allow for one to fairly easily and quickly swap from one pawn to another, with whatever pawns you happen to have. However... Just because it can swap your control between any pawns, doesn't mean all of your own pawns, or existing controllers, will support having other pawns/controllers being used. If you make your own pawns from scratch or from third person characters or such, you'll probably have zero problems. But if using something from an existing and more complex setup... problems may arise.

For example, two popular products on the Unreal Marketplace are the Survival Game Kit, and the Horror Engine. For the Survival Game Kit, they have a custom class of pawn for the characters, and it must work with a specific player controller. If you don't have the controller, the pawn and many aspects of it won't work, nor the game mode. If you have the controller and game mode, but swap out/remove their specific pawn... the controller and game mode will stop working. That is a system which was pre made to do one specific type of game mode (as of my last experience with it). So if you want to have vehicles... or a different pawn not a child for its own class, too bad. The controller will start throwing constant errors, the game mode will break a bit and try to respawn you thinking you died/were destroyed in a new pawn, and it may take a LONG time to modify that large complex system to work with different pawns in play. The Possession System could be used on the character to let you possess a vehicle with their pawn... but the controller and game mode themselves would have errors and not work without the original pawn being both still in play, and the active player pawn. My one attempt to make this work for someone took multiple hours and incited many a curse words.

For the Horror Engine, you have a case of a custom camera and HUD that reference that original pawn, destroy it/replace it... and those will start having errors. Lots of them constantly. In this case you can set up a "Event Destroyed" node in the pawn to destroy/despawn its own camera actor and HUD from play when it's destroyed, and have it respawn them when it's spawned again, and/or throw in a few IS Valid nodes into those actors/widget so they don't work when no valid pawn is present. In this example you can fix it up in fairly quick order if you know what you're doing in the blueprints, but it does require some work.

This is all to say, be warned that when using existing prebuilt game modes and systems, frameworks, etc... they may not support things like the pawn or something being changed, destroyed, etc... so if you want to use this in one of these, be aware that they might not support the idea of swapping pawns itself, whether you do that on your own, with this system, or any other, it's a limit of how some stuff is designed. And modifying them may be quick, or may be long. Just be aware of that limit in advance if you want to add possession changes and/or vehicles into your game, be it using this product, any other, or just implementing such on your own.

## Updates

3/12/2018:

Added an array variable for auto disabling Look At Info elements when possessed system is possessed/unpossessed. By adding integer values into this array it will automatically disable/enable the look at info Is Active array for those elements when entered/exited, if on the same actor.

Added a "Sole Interaction" variable on the possession system so if enabled you don't have to manually plug in the interaction message on the actor, and the interact component will send it directly to the possession system of that actor if appropriate.

Also added an interaction type variable for the Possession system to properly use this function.

Added an option to directly contact the Look At Info system from the interaction system, once again meaning you don't have to set up those nodes from the connections manually if you are only using my own components and not mixing in other functions or stuff of your own that would need to make use of it on the actor. The overall idea of all of these is to save time when you are only connecting these interaction/look at messages into my own components provided in this package. If tying your own systems into them, you'll likely be leaving some of these disabled as may be needed, but overall it should speed up most people's work with these.

"Send Look At Requests Directly To Look At Info Component" added to Interact\_BP, default category.

"Sole Interact Method On Actor", "Interaction Type", and "Auto Disable/Enable Look At Info Elements" options added to Default category in Possession\_System\_BP.

~3/15/2018:

Added a Demo UI Widget to the project, and incorporated it into the 3rdperson character BP. You can now swap between a UI widget or the in world floaty text using a variable in the 3rdpersn character BP (meant to be set before starting the game). This can help those trying to figure out how to set up a UI using widgets with the Look\_At\_Info\_BP to get started, and also can help them figure out some of the basics for making replicated UIs.

Also fixes an issue with automated Look At Info requests from the component using the option to directly contact the Look At Info components in the Interaction BP always returning element 0 responses.

~11/15/2018:

Updated some tooltips on exit/enter/interact point keywords, and also added ability to automatically pick an exit point even if none was listed for a given enter point.

#### **~4/4/2019: MULTI Possesion System**

**Added a new Multiple Possession blueprint (Possesion\_System\_M\_BP)** for supporting multiple controllers to effectively possess a single actor. This relies on the creation of possession pawns that should be attached to the actor, and are what a player will actually possess when they are in a given vehicle slot, however the Possesion\_System\_M keeps track of the various pawns, players original pawns, and controllers. So each pawn can simply send their requests to be moved to another spot, or to exit the actor to the Possesion\_System\_M component on the main actor, and it can keep track of everything.

This allows for multiple players to occupy and swap out between parts of the same vehicle, like two players in a tank. One driving and one manning the turret.

This system also allows for the old character pawn to be moved to a seat socket, and updates the various actors with info on these changes so you can have a character assume a given animation when in a given location for a vehicle.

The main additional actions required to use this new Possesion component, are to in the construction script, set the Pawn List and the Seat Component List for each of the positions. The seat component is a component (like an arrow) that an original pawn would be attached to when in that position or slot. The Pawn List will contain pawns that the player would end up possessing (in most cases, the slot 0 might as well be the actor in question to cut down on extra pawns needed). The additional pawns can be added as child actors, or spawned during play as may be needed. There is no inherent requirement for a pawn, however you'll generally need to set up events to forward the desired input commands to the parent actor. For example for a Turret in a tank, a pawn made to hold a player for that position would want to forward turret rotation updates and cannon firing events to the main tank actor so it can execute those actions. It would also want to have some events setup for leaving that vehicle position, or swapping to another position, that would be forwarded to the Possesion\_System\_M component on the parent actor.

Here are some options for the Possesion\_System\_M\_BP when in an actor.

<b>Default</b>	
Interact Volume Keyword	EV0
Exit Point Keyword	EP0
Sole Interact Method on Actor	<input checked="" type="checkbox"/>
Interaction Type	3
Auto Disable/ Enable Look at Info Eleme	2 Array elements +
Notify Anim BPs Of Possesion Changes	<input type="checkbox"/>
<b>Exiting Rules</b>	
<b>Positions Info</b>	
<b>Seat Priority Lists</b>	2 Array elements +
0	1 members
<b>Position Priority</b>	2 Array elements +
0	0
1	1
1	1 members
<b>Position Priority</b>	2 Array elements +
0	1
1	0
Exit Component Keyword	2 Array elements +
Space Taken	0 Array elements +
Position Components	0 Array elements +
<b>Seat Type List</b>	2 Array elements +
0	Sitting 1
1	Sitting 1
Sorted Input Priority List	0 Array elements +
<b>Entry Position Keywords</b>	2 Array elements +
0	EV01
1	EV02
Sorted Exit Components	0 Array elements +

The Possesion\_System\_M\_BP has some similar options to the original possession system, but also some different ones. The most notable differences are exiting, now moving an



existing pawn to a new location, doesn't reliably keep the pawn from exiting if that area is blocked (though this may change in future updates).

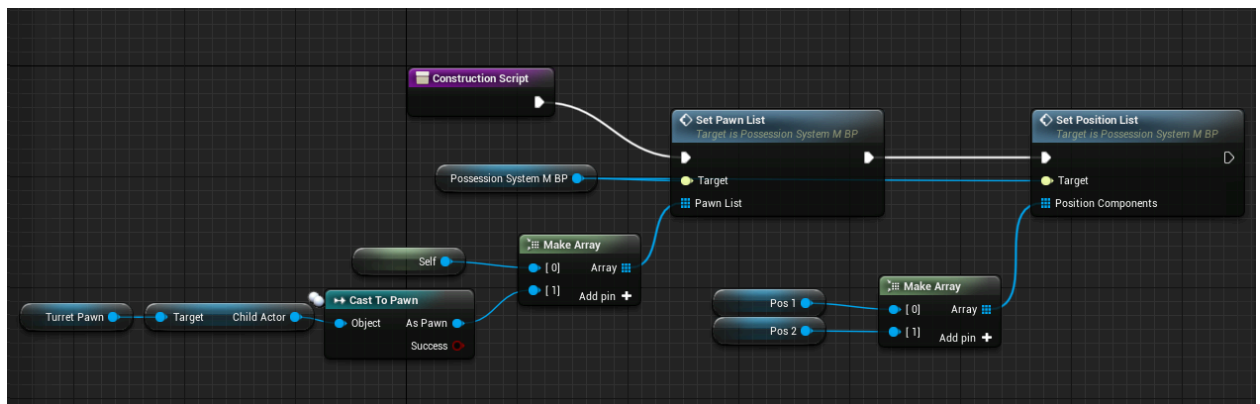
Next, is the Positions Info category. Here you set the finer details on all of the individual positions. Seat Priority Lists, will show the possible positions you can enter from a given entry position keyword of the same array element, and in the order they will be tried for.

Entry and Exit position Keyword arrays will identify components with the given keywords and use them for that position. Note that all interact or exit components must also include the Keyword appropriate to them from the Default category.

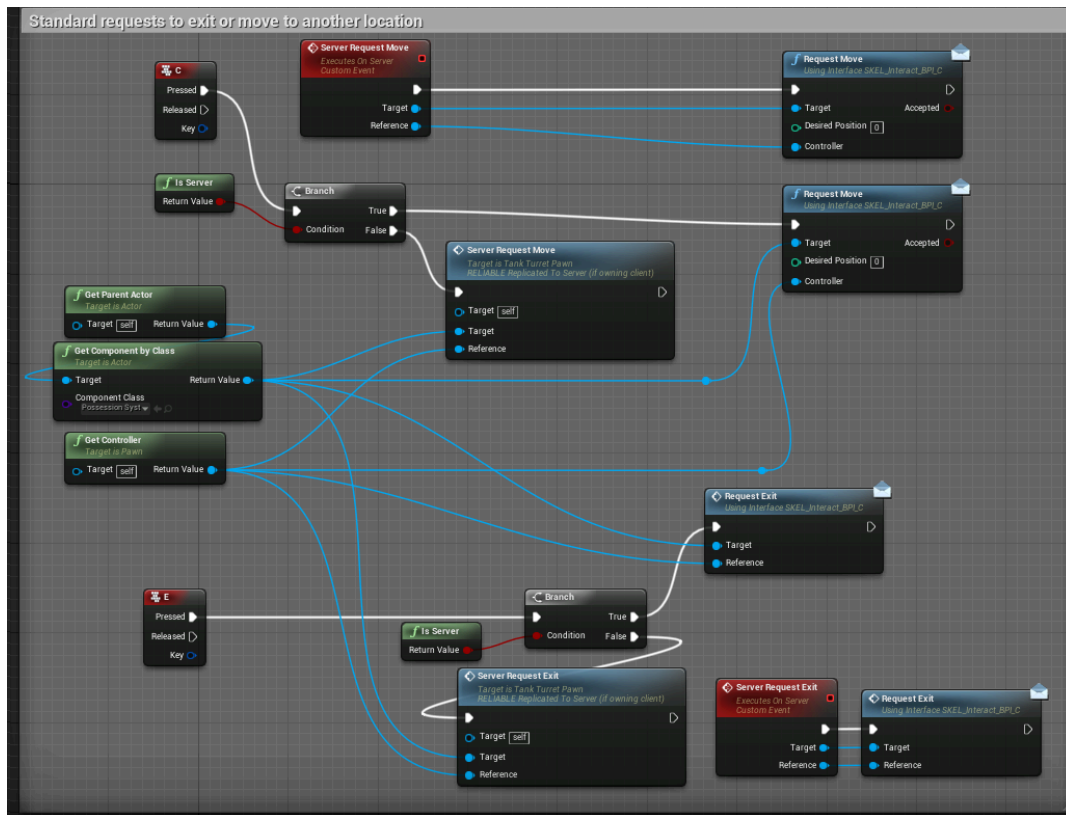
Next is the Seat Type List, that allows you to set a value to identify that positions type. This comes from a custom Enumeration added to the package, and you can modify it to have whatever types of positions you want. The Possesion\_System\_M itself does nothing with this info, other than store it and provide it to all relevant actors upon a possession change with the other possession update info. So if your original pawn receives Possession Info telling it it's now possessing a position in another actor, it will also know the seat type for that position. This can be used to setup how it should animate or pose or behave for a given position.

The sorted list values will be generated by the component itself once it starts play, based on the found components.

Another key part is that you'll want to setup the following in the construction script. Informing the Possesion\_System\_M\_BP component of what all pawns it has for each position, and well as seat positions. In this case the first position (tank driver) is the tank actor itself, and the second position (turret operator) is a Turret Pawn child actor made specifically for this tank actor.



Here is an example of how you can set up a pawn actor to contact the parent actors possession component to request a movement or to exit that position.



Beyond this most functions of a pawn will be unique to your specific pawn and parent actor, such as forwarding when to shoot, rotate a turret, or perform whatever other action you should be able to perform from that position. Keep in mind that to be able to make use of this to its best, you'll need to understand replication and how to send commands over a network to the parent actor, and how to have those actions show up on all clients as is desired. There are many ways to approach this, from some actions being simple one off events, to more constant things like movement where you might want the local client to be able to see its own actions immediately and locally even before the server is aware of the changes, in other cases you might want to confirm with the server before making changes to things like movement. In some cases you'll go for a network heavy but smooth option, and for others you might simplify actions into a series of updates for a desired value that will be lerped towards on the server or clients, so as to allow for a somewhat smooth seeming action with minimal network cost. It all depends on your needs.

For the demo, most actions are on the more expensive but smooth (and simple to execute) side, and should not be taken as ideal examples of how to set up such actions yourself for your own game, but mostly as a proof of concept to show the ability to control the parent demo tank even from the child Turret Pawn.

~6/10/2019:

Added better support for remote control actors, so with multi possession system you can just not set a position component or exit keyword/component and your original actor will remain at its original location upon possessing and exiting.

-7/26/2019:

Added a new option under exiting rules category for multi possession system controlling if the original pawn will be "teleported" or not when moved. Set to true if using a fully simulated skeletal component, false if not and you want hair or other simulations to keep working to at least some extent during the movement.

-8/12/2019:

Fixed issues that could produce a weird bug in CharacterMovement components that could result in faulty animations and some other issues.

Also fixed an issue allowing multiple players to enter the same spot of the multi possession system.

-8/14/2019:

Added more options for managing rotation/location upon unpossessing an actor for the multiple possession system. It can now exit an actor exactly where it entered relative to the possessed actor.

Player controller is also rotated upon exiting to match the actor for dismounting at weird angles in some situations.

-11/30/2021:

Fixed an issue with exiting on slopes with the multi possession system possibly leaving the controller improperly rotated on clients in multiplayer sessions.

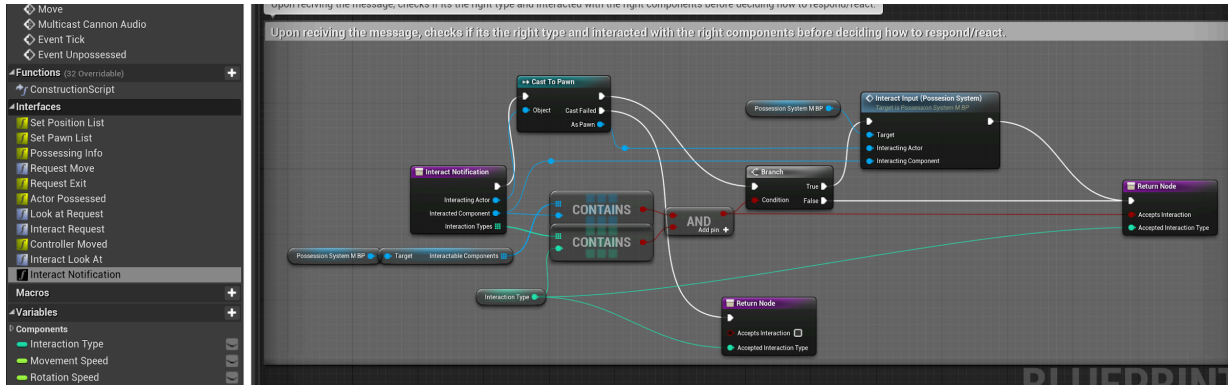
(Planned update when I get around to it (no current eta, just a future idea))

Options for look at info system interaction in the multi possession system components. Rather than just auto disabling the look at info for a seat position/component when its spot is filled, to have a series of options for each seat position in the actor, and to display whatever seat is next available should you enter through a given interaction point (currently if one interaction point allows for use of two seats, if the first seat is full it just would disable that if given the option, this idea is to auto update those values to what they should be at that moment given the current seating position, such as saying it'd send you to the second possible seat that is still open, only disabling it when there are no spots left.

## Q and A

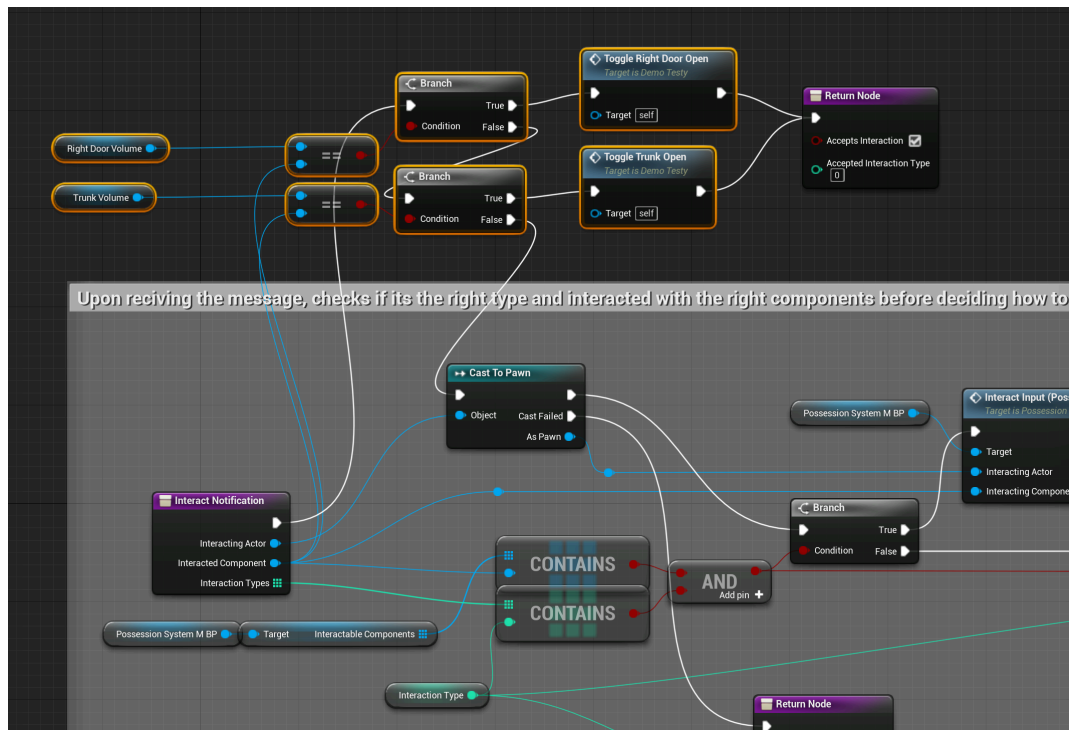
Q:How do I add multiple functions to one vehicle? (like opening doors or stuff)

A: First, make sure that your possession system has “Sole interact method On Actor” set to false. With that on false, the interact system will send a message to the actor using the interact\_BPI interface. So you’ll need that on your actor, then go to the interfaces and interact notification function. By default you’d have it set up something like this.



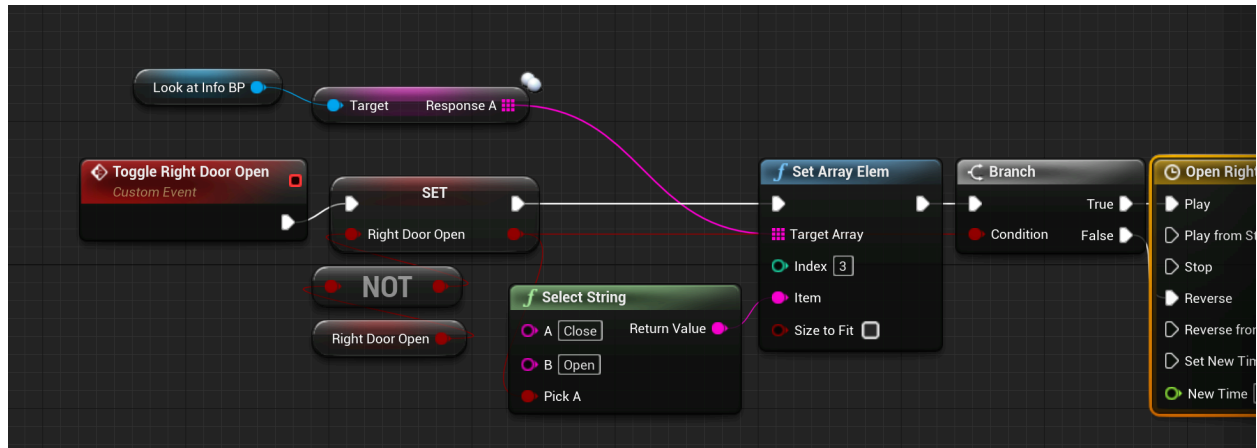
This is where interact requests come through the actor, normally you want it to still forward onto the possession system should it contain an interact volume of the possession system. But to add some extra functions like opening a trunk, opening a door, or an external control panel for an auto turret or whatever, we can plug any such custom behavior in here too.

Here we have some volumes in the actor that relate to each of these, if you interact with them, then it will instead perform that action rather than checking for possession.



Here, we simply check if the interacted component equals the desired volume, and trigger a custom event for that if true. You also might want to check if the given interaction type is there for these actions depending on your game/situation, but this is the general idea. Just make custom events in your actor that handles these various other behaviors, kind of like how the door examples will open/close based on you interacting with their volumes, same basic idea.

For handling the look at requests, you'd need to manually toggle these now, in this case rather than toggling visible look at reponses, we'd probably instead toggle the response type between open/close for right door. So here's an example of how the door event might use Look at info.



Q: My vehicle is despawning/my character respawning at random seeming when driving around.

A: In UE5 in particular if the vehicle has "Is Spatially Loaded" checked in the details panel in the level, when the Player Character passes over a World Partition in the Vehicle, the Vehicle may reload. This can result in it despawning.

Also check for level limits, killboxes, or anything else that may destroy actors upon entering/leaving a given area.

Q: In multiplayer, my vehicle begins jittering around and bouncing weirdly, possibly very stuttery/choppy seeming.

A: Depending on multiplayer settings if you don't set the relevancy/replication settings right on an actor it might unload or start getting really jittery if too far from the host/server. Check net relevancy settings if it is on your vehicle actor/components. This feature makes objects far from the host or players lower their priority for net updates, depending on game settings it may only care about the host meaning other players can be affected by this.

Q: When I try to possess something, my game breaks. Random icons, a bunch of error messages, the camera gets stuck in the ground, etc...

A: Test with a demo possessable cube, if doesn't happen when you try to possess it with your actor in your level, then it might be how you set up possession/the camera on your own vehicle/etc...

If it still happens, then it's most likely from your game mode, player controller, etc... If you are using a pre packaged game system like a third person shooter setup, or Survival Game Kit, then you may need to modify those or simply not use them. The reality is many prebuilt game mode/kits that give you a good starting point for a given type of game, don't work well when you try to add/modify things beyond that specific starting game mode. Adding vehicles for example. When you possess a new vehicle/actor, you are unpossessing your original pawn, possibly despawning it too. Some of these system will throw errors not finding the original pawn to check its health, food, hunger, stamina, inventory, etc... Or might have kill cams that trigger thinking you died because youre original pawn is no longer possessed or present, forcing you out of the newly possessed vehicle and into a death view or such.

The details vary heavily from product to product so I can't comment on them all, and not being the one to make them can't reliably offer support. But in general you'd need to modify them so they don't throw errors when the player reference disappears or the player controller is controlling something that doesn't have their preset components/references... Liberal use of adding "is valid" nodes can stop errors, but often you'll need more work to make them support vehicles than just stopping the errors.

I recommend asking in advance if these game modes/packages would support a player possessing a different pawn when using them, or not, before buying them. Assuming you plan to have vehicle or such in your game. Some do, some don't. Some don't but it's a fairly quick fix to make them work, others don't and it'd be hours and hours to rebuild them so they do... So ask and check in advance if they can support vehicles or possessing other pawns to save yourself some time and trouble.