

Guide 1

Overview 2

Config 4

SkillTree 4

Node 6

Tooltip 8

Input 9

Material Instances 9

Currency 10

TACTStats Component 10

Modularity 10

Overview 10

Cooked Games 11

Troubleshooting 11

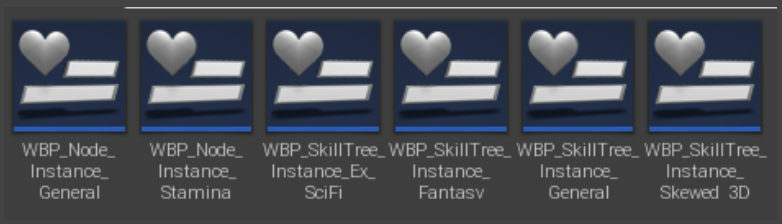
Known issues 12

Guide

- Go to the top of the Widget Hierarchy and set a unique TreeName
- Press the button under the Category EditorCallable, by the name "Construct Clean Save Game From Tree"
- Add Node(s) from the Palette, drag them from the Pallete out into the NodeContainer Canvas space (Or onto the NodeContainer in the Hierarchy)
- You can now move your Node if you want
- You have two default Node(s) to use. But you can make however many types you want
- You can make your own by duplicating Node_Instance_General in the Asset Browser, and set up convenient defaults there
- Once you have some Nodes in the Hierarchy you will want to give them some Connections
- Click a Node and go to the array called Outputs and add an entry. In it's dropdown you can select the node you want to connect to
- Keep doing this until you are satisfied
- Remember to generate a unique GUID for your Nodes. You can do so under the Config-Node Category on the Node itself
- Hit Compile twice and Save (I know it's weird. but works)
- Config-*Suffix* is where you will find most things you can tweak on any of the provided Widgets
- When in PIE(Play In Editor) on the map SkillTree_TestLevel the keyboard key K will enter and exit a SkillTree
- You can get it to display yours by editing what Widget gets spawned in the Level Blueprint
- Have fun!

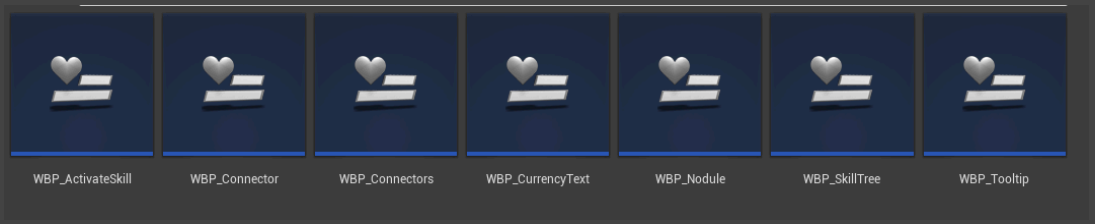
Overview

Examples



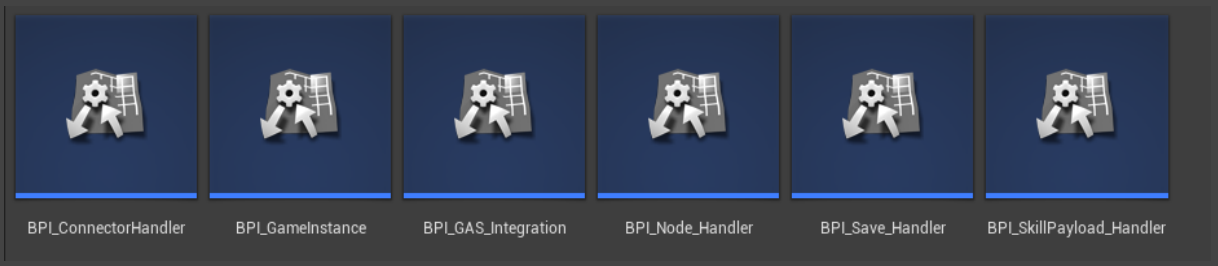
These serve as Examples for you to look at and to learn from.
They are found under TactSkildrasil/Widgets/Examples.
The ones suffixed with _General are the most basic ones. Their main use is for you to duplicate and work from.

Under TactSkilldrasil/Widgets/Base you find Parent widget-classes



These are mainly used by the Developer to deliver updates that propagate through the entire system. Only touch these if you are ok with them being overridden by future updates. Almost all of the logic in child Nodes, SkillTrees, etc are derived from here.
Nodule is named as such to not confuse end-users looking for “Node” to use in the SkillTree, but it is the Base for Node.

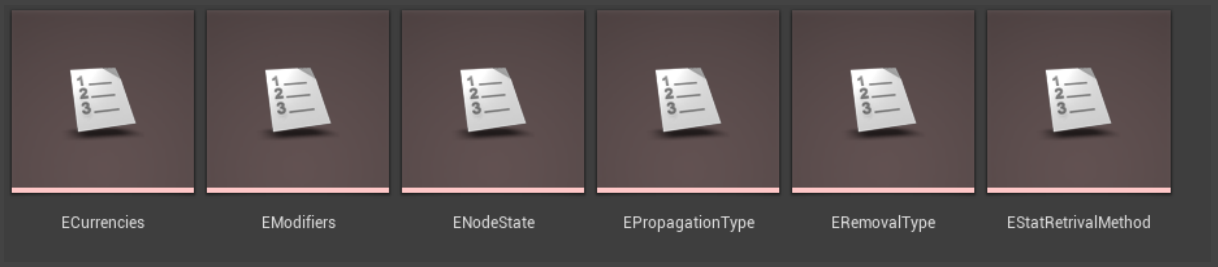
The system has these Interfaces that are used for BP communication.



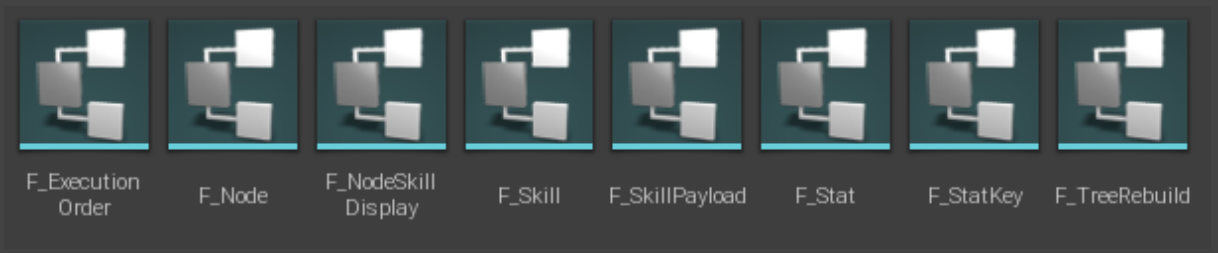
BPI_Node_Handler communicates mostly between the SkillTree and the Nodes.
BPI_SkillPayloadHandler communicates a Payload to external systems. Things such as abilities triggered or activated that need to be communicated to your own systems, such as ABLE, GAS or DCS. There is a Struct called FSkillProxyData you can populate with whatever data-types you need. By Default it sends a GameplayTag.
BPI_Save_Handler communicates between SkillTree, Nodes and SaveGame Objects. Handles Saving/Retrieval of stats, as well as an abstract Hierarchy of GUIDs that correspond to the Node connections, used to reconstruct the SkillTree.
BPI_ConnectorHandler communicates and updates the new Connector type.

Under TactSkilldrasil/Objects you will find the following:
BP_SGO_Progression is the type of SaveGame object that the SkillTree system uses. It contains any data the SkillTree needs to save and load.
BP_BaseUpgrade is the Parent object for any Skill or Ability. Their main use is to encapsulate functionality that is custom to your system.
BP_TACT_Stats is the Component you add to anything you want to give Stats to

The system has these ENUMS



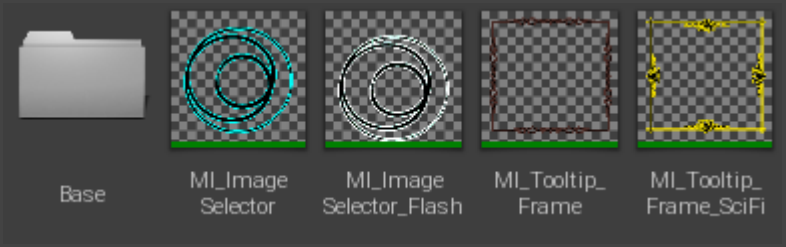
ECurrencies - A list of currencies. You can set this up however you want.
ERemovalType - Used by StatModifiers making config easy.
EStatsRetrievalMethod - Used by StatModifiers making config easy.
EModifiers - A list of modifiers to stats.
Don't touch this unless you know what you are doing.
ENodeState - A list of NodeStates.
Don't touch unless you know what you are doing.



FNodeSkillDisplay - Container for colors to be used in certain NodeStates.
FNodeStruct - Container of final Node data for user entry. Most notably FSkillProperties.
FSkillProperties - Container for properties of a Skill.
FSkillPayload - Used to pass on a Payload, beyond the SkillTree system. This is the Struct where you would add your own custom data.
FStatKey - Used as a key to find a particular Stat with a particular modifier.
FTreeReconstruction - Used as a container for information to rebuild the SkillTree. Important to avoid corruption of references.

Under TACTSkilldrasil/MaterialCollections you will find MPC_Widget_Communicator. This is a Material Parameter collection. Used to pass on information to Dynamic Materials Instances to handle things such as the Frame and Icon color of a Node type Widget.

Under TACTSkilldrasil/Visuals you will find Materials and MaterialInstances.

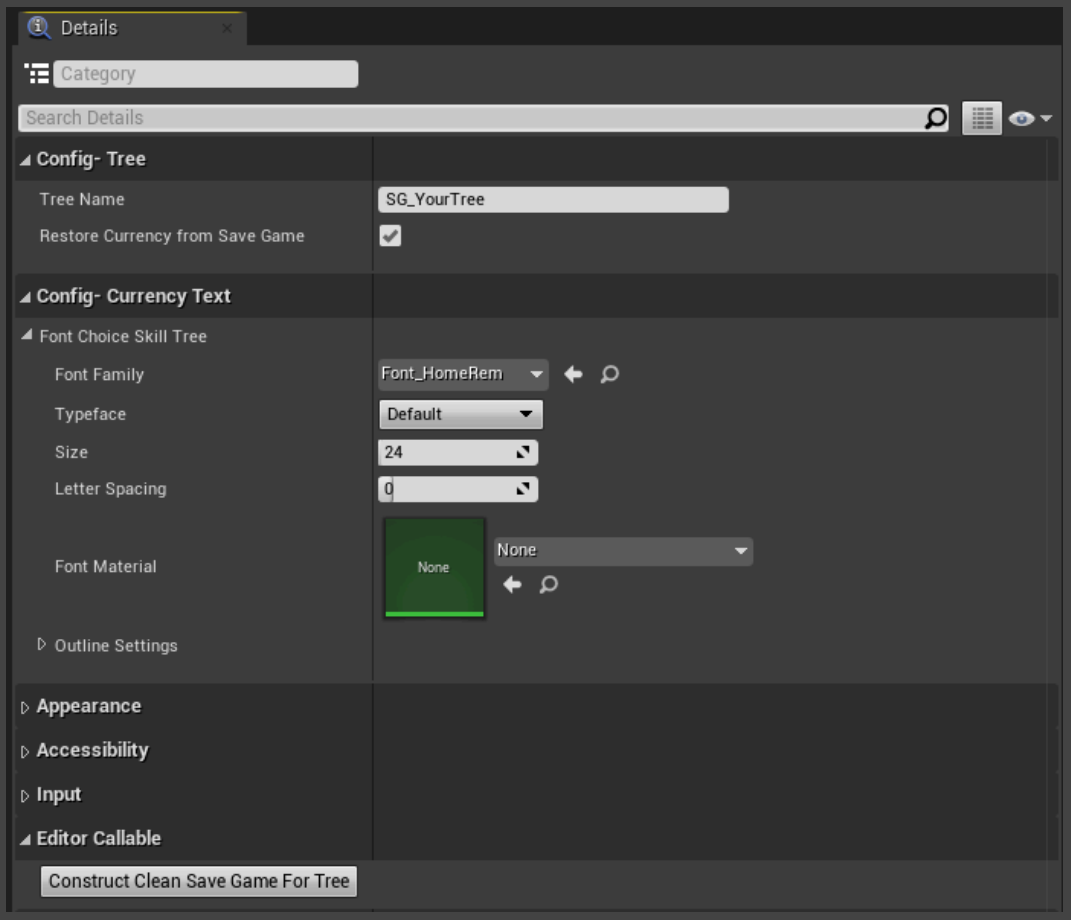


Under /Base you will find your Parent Materials. Tooltips require you to make new MaterialInstances for the style you want. But for Nodes the system creates Dynamic Material Instances you can set up on a per Node basis. MI_ImageSelector and MI_ImageSelector_Flash serve as a foundation for them.
M_Lines is used for the new Connector type, which uses some smooth edges to give much needed fake Anti Aliasing to this type of Connector line.

Config

SkillTree

Depending on where you are in the Hierarchy of SkillTree you will have access to different options to configure and use. In the top of the Hierarchy you have access to these new properties in the Details panel:

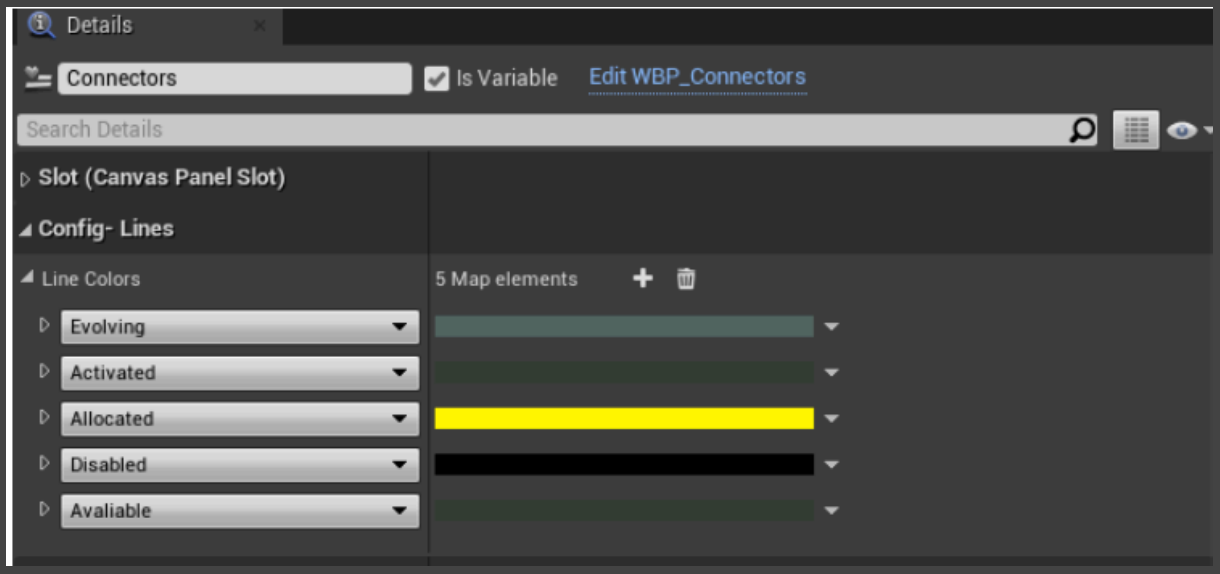


Here you can configure the font that appears in the SkillTree itself. Nodes have their own Config for this!
More importantly when you are done, press the EditorCallable button “ConstructCleanSaveGameForTree”. This will create a unique save game based on your Tree Name.

Moving on, you can tweak the color of NotEnoughCurrency_Text, but the rest gets set by the system. Notice that this Widget is the target for a Widget Animation called NotEnoughCurrency. This Animation only controls Opacity when it’s triggered, to show and hide the text.

You can tweak the color/appearance of any SkillTree buttons as you wish. This includes QuitButton, ResetAllocation and CommitAllocation. Font choice is being handled elsewhere.

Next up is the Connectors widget:

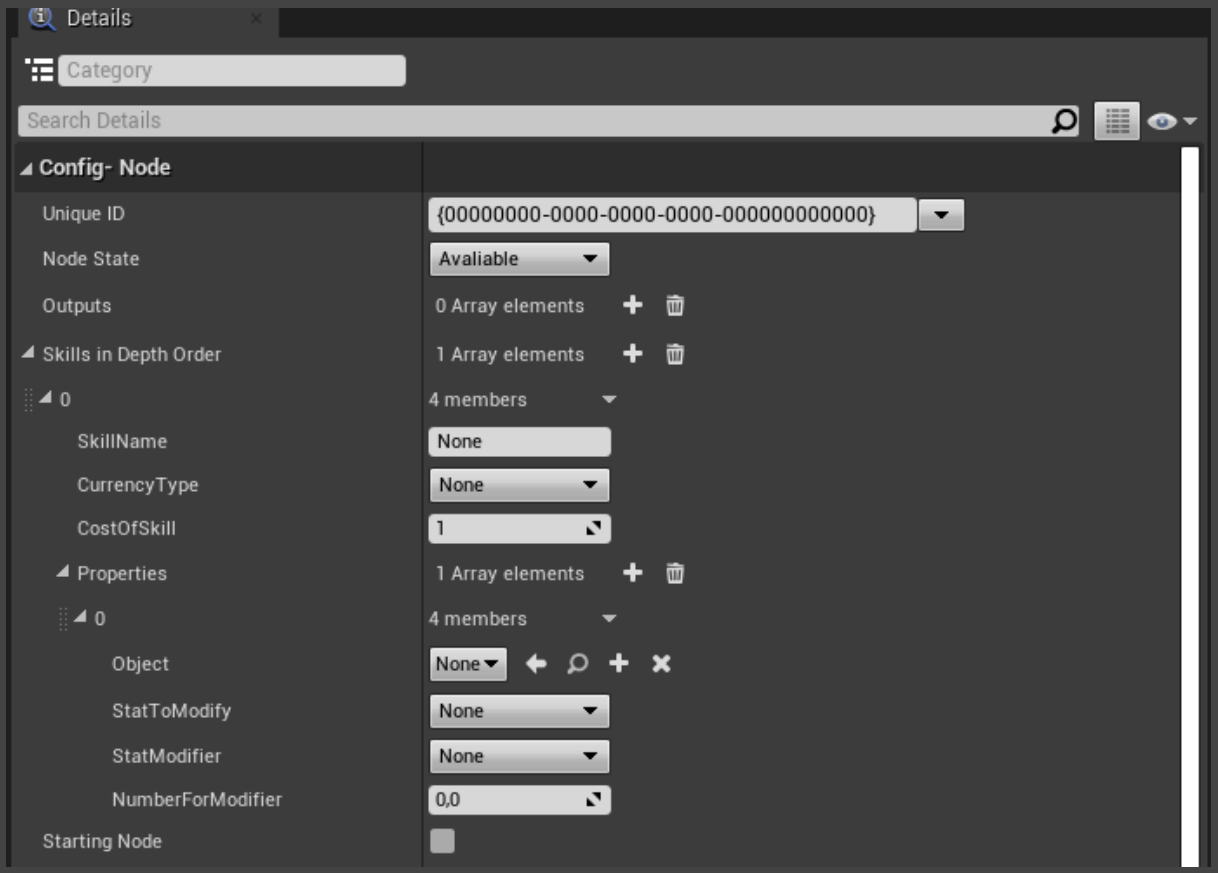


The lines drawn between Nodes change their color depending on what NodeState the adjacent Nodes has. These colors can be tweaked here.

Finally in an empty tree you can tweak Background. You can set any image you want. You'll find that it's just a simple image so edit it as you would a regular Widget.

Node

In a Node we are only concerned with the settings in the top of the Hierarchy.



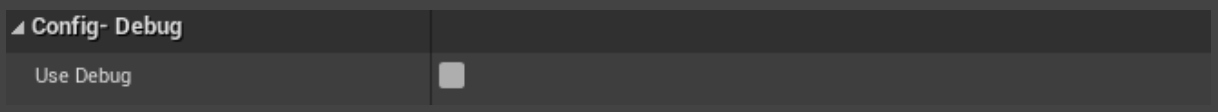
You can leave UniqueID for now and set that when you use the node in the SkillTree.
Let me explain the node states for you.

Available: You can spend Currency to Allocate
Allocated: Ready to Commit to Evolving Node
Activated: Node is purchased and therefore Activated
Evolving: Node will change to another after purchase
Disabled: The Node is Disabled, use this if you want Nodes to only be unlocked through your custom needs.

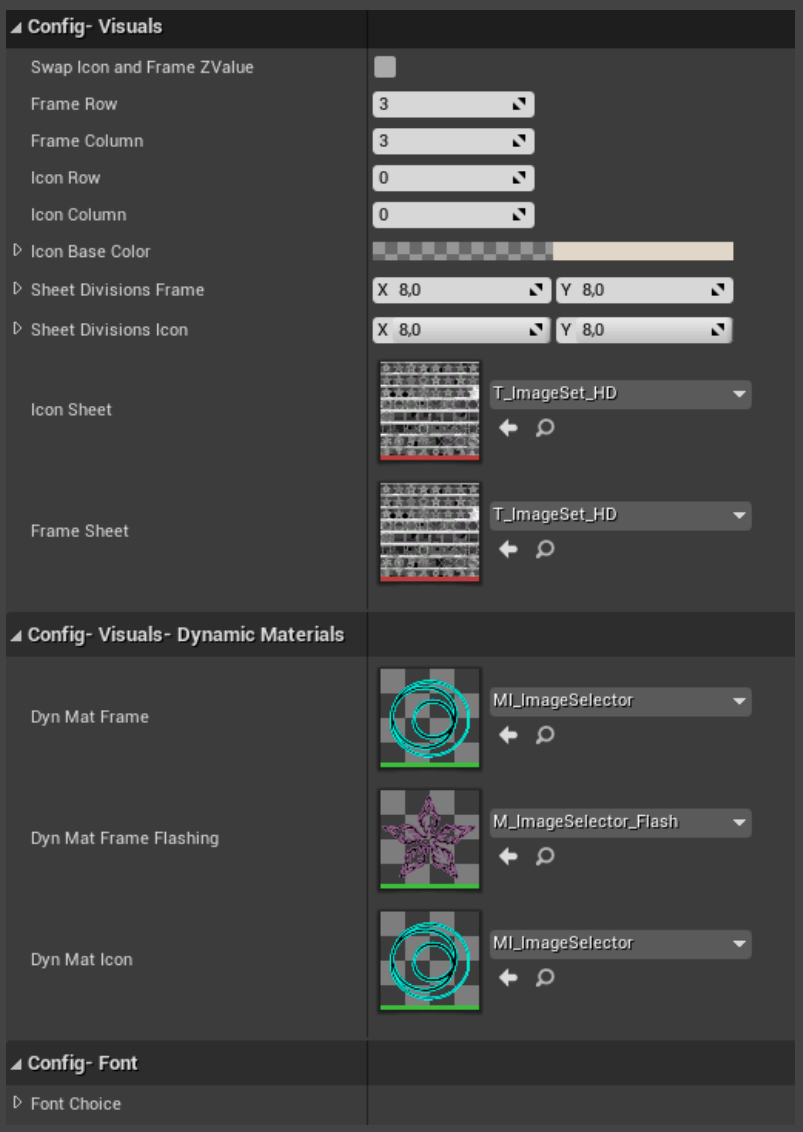
Outputs - This is where you select what other Nodes to connect to in the Skill-Tree. Leave this for now and set it up in the SkillTree later.

SkillsInDepthOrder - Each Depth has the Variables SkillName, Currency type and a cost called CostOfSkill. However each Depth can have any amount of Properties. The properties are as following:
Object - An object that can be called for custom behaviors. Not currently used by any system.
StatToModify - The Stat you want to modify.
StatModifier - How you want to modify the stat.
NumberForModifier - The amount you want to modify your stat by.

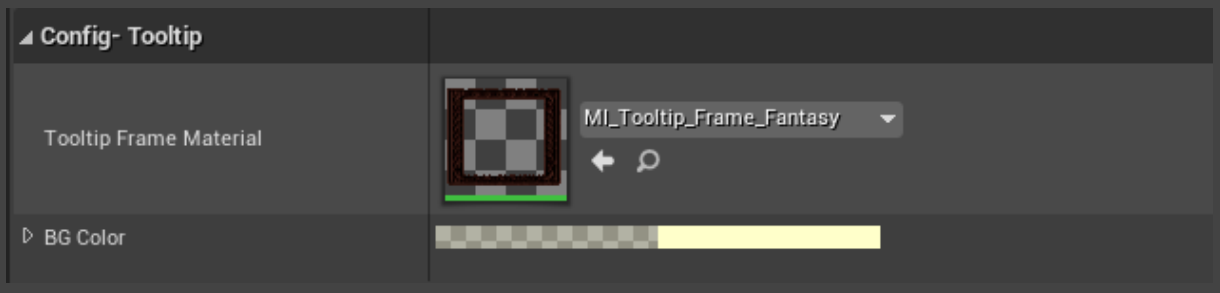
Finally after Properties are set, you can set if the Node should be regarded as a "StartingNode" If it is, this means it needs no Activated connections to it in order to Allocate Currency in it so it can be Activated by Committing it. Next up



Setting this to True will print some Debug info to the Tooltip Widget for the Node. This way you can see in-game if something is configured oddly or wrong. Next up are the visuals of the Node:



Shown here we use an Image-set which you select from using FrameRow & FrameColumn, as well as IconRow & IconColumn. You can use a single image as by setting SheetDivisions to 1 We will cover the DynamicMaterials in another section. Lastly, you have FontChoice under the category Config-Font. Here you can select the Font that will be used for all text on the Node Widget.



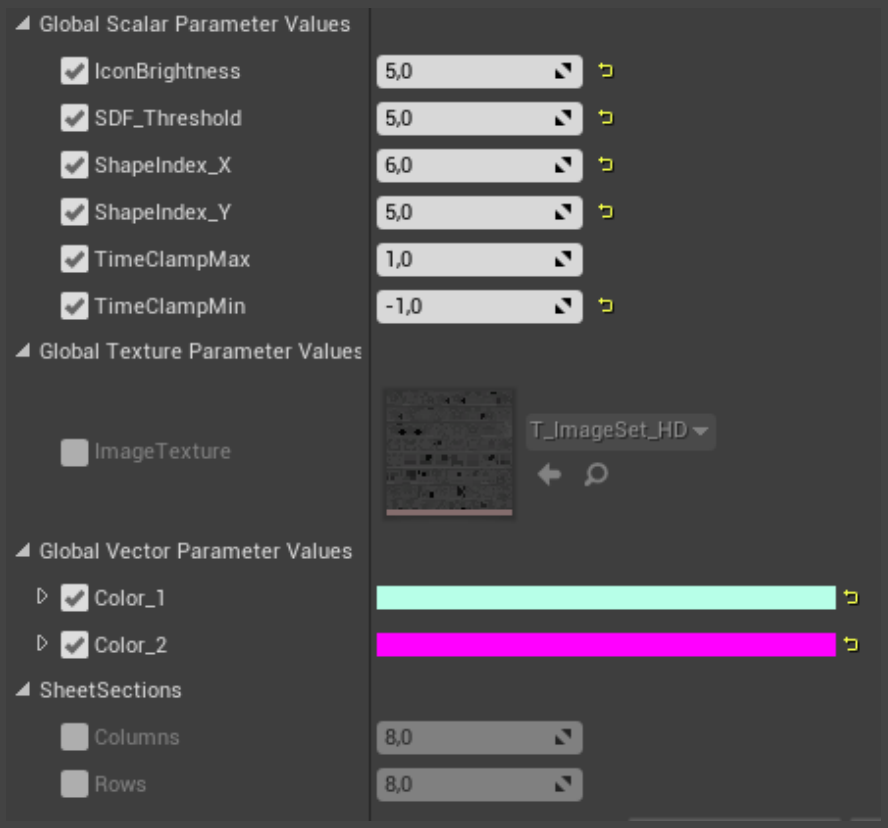
Here you can select the Tooltip Material on a per Node basis. You can also select the color of the background of the Tooltip.

Input

Input is handled in two places in my Demo Level. To enter the SkillTree you press “K” on the Keyboard. Same to exit. We are handling entering the SkillTree through the Level Blueprint. And we handle closing it on the Parent Widget to all SkillTrees, in the Overridable function called OnPreviewKeyDown.

Material Instances

MI_ImageSelector_Flash has these parameters. They are virtually the same as MI_ImageSelector with the difference being that this has more parameters so I will only explain this Material Instance once.



IconBrightness - This dictates the overall output brightness of the Icon. For my Image-set 5 was a good number.
ShapeIndex_X - Controls the Y coordinate to select what Image is currently displayed. Should move in Integers
ShapeIndex_Y - Controls the Y coordinate to select what Image is currently displayed. Should move in Integers
TimeClampMax - Controls how close the pulse will get to Color_2
TimeClampMin - Controls how close the pulse will get to Color_1
ImageTexture - This is where you select what Image or Image-set you want to use for your Icons and Frames.
Color_1 - Color that the Sine Pulse starts with.

Color_2 - Color that the Sine Pulse ends with.
Columns - Divisor for how many Icons exist on the sheet in Columns.
Row - Divisor for how many Icons exist on the sheet in Rows.

Currency

Currency during gameplay is retrieved by the Skilltree from the SaveGame, and then updated to every Node. So in order to set your starting currencies you have to go into the BP_SGO_Progression and set the variable there named CurrencyAmount.

TACTStats Component

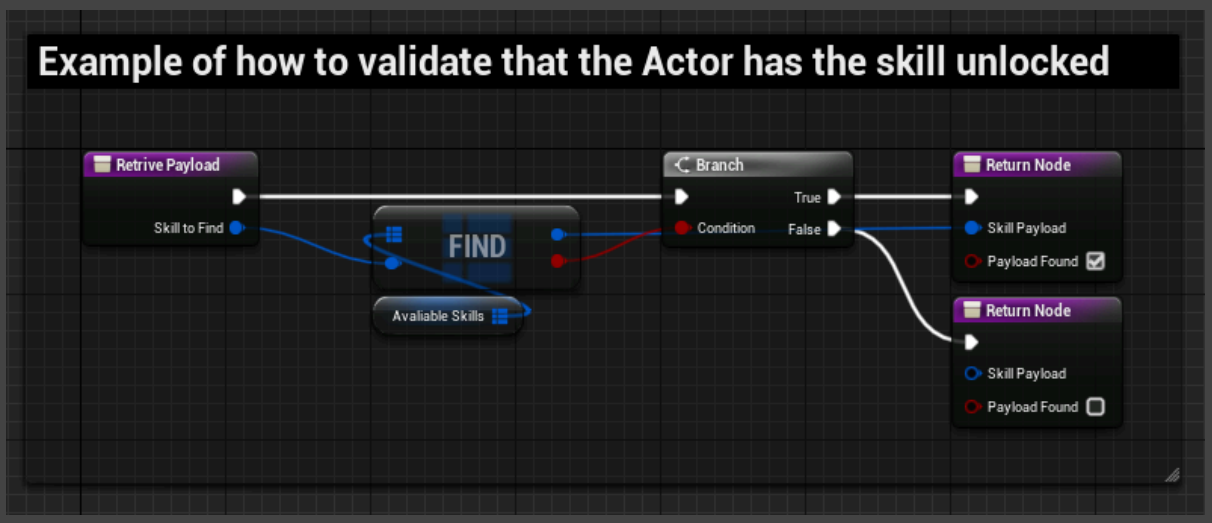
As of Update V07 TACTSkildrasil ships with its own Stats solution. Maintaining compatibility with other systems by good use of its Interfaces.

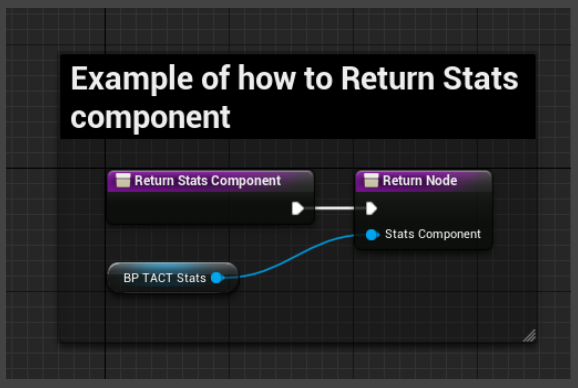
Modularity

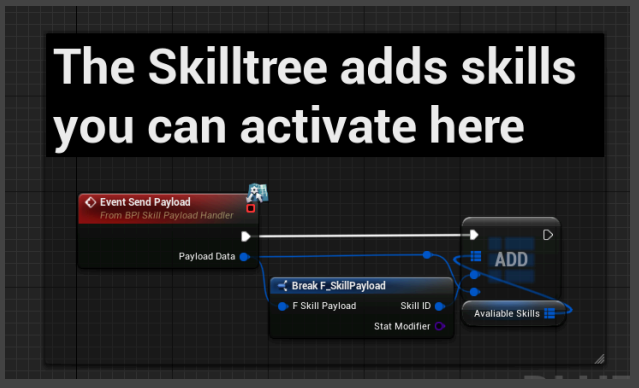
TACTStats Component is entirely optional. And hooking up another Stats system is the same process as before, involving you to pass on a custom Payload that comes in the form of a Payload that you can find on any child class of BP_Base_Upgrade.

Overview

TACTStats is a Component. You can add this to ANY actor! And you can initialize/Instantiate Stats from different sources. Be it from a SaveGame, the Actor itself or from another system such as GAS. For this to function you will have to Inherit the Interface SkillPayloadHander, and then Setup two Interface Functions, this also necessitates you create the Variables needed by the Interface:







At the Event level in BP_TestActor, you have examples of Testing Events. If you need any testing done to your Stats, you can add Events such as these and the Component called BP_TestFunctions to your Actor, this gives you some good basics for testing your Stats.

If you have questions about something Stats related you can look at BP_TestActor to see how things were implemented there.

Cooked Games

In cooked games you will want to move your SkillTree SaveGames from TACT-Skilldrasil\Saved\SaveGames to the SaveGames folder of your build.
Remember to reset the saves in the editor by using the EditorCallable button “ConstructCleanSaveGameForTree" in the top of the relevant SkillTree hierarchy.

Troubleshooting

Q: Why are my SkillTree connections not showing up in Editor?
A: You probably did not compile twice after some change, or you didn’t create a fresh SaveGame by using ConstructCleanSaveGameForTree.

Q: Why do the connections not show up in Cooked games?

A: You probably didn't move over your SaveGames. See the Cooked Games section for further information. Why this is needed at all is a mystery to me as well, I plan to find a way to automate this in the future.

Q: Why does the example with stats not work when creating a project?
A: It is likely due to GameplayTags not carrying over to your new project. Take some time to set up your own Stat GameplayTags. It's used widely across the system to identify both Stats and StatMods.

Q: Will this work with (Insert any Marketplace Asset here)?
A: Yes.. yes most likely. Depending on what it is it will take a bit more work. If you already have skills and abilities you will have to spend some time creating ProxyData that hooks into the system. I tried to make this process easy. You can override CallCustomAbilityObject in the SkillTree and build out your own logic.

Q: When editing sometimes my Nodes have the wrong state.. I didn't set this up!
A: I have some checks in order to make sure the SaveGame side does not affect the Editor out of PIE (Play In Editor), however It's possible something might have slipped.

Known issues

-There is an issue with LegacyConnectors. Any Node that has multiple outputs will Draw Lines in a bit of an odd way, more than once. The new default Connector type does NOT have NodeState colors yet. But they don't have the LegacyConnectors issue. If your Tree is linear, you can use LegacyConnectors.

- The Zoom in and Zoom out acts on the whole screen and does not respect the size of the SkillTree Node Area. I need to consult some of my confidantes on how to fix this.

- Stat restoring after a StatModifier runs out needs some better math to function properly. Especially when dealing with a ModifiedStat. Be careful with division and multiplication. Percentages, plus and minus should work.

-Final interval tick on a StatModifier sometimes does not update the DebugStatBar.

-A long time ago, when the system was new, It would routinely corrupt the load of Widgets and jumble up references. That is the reason why I started using a SaveGame to reconstruct the data. However it is still true that if you remove nodes hap-hazardly and change GUIDs on nodes left and right, you are bound to mess up some of your references. The system is really good at reconstructing from what it believes should be correct. But to be safe, have the editor open, and create a fresh save for your SkillTree to live inside.