Introduction

Welcome to the documentation for the SenseGlove Unreal Engine Plugin (a.k.a. The SenseGlove Unreal Handbook)!

This handbook is an ongoing effort and a work in progress to document the SenseGlove Unreal Engine Plugin. Feel free to visit this handbook on a regular basis.

Due to superior formatting and frequent updates, we recommend the online version of the handbook; nonetheless, it's also available in PDF and ePub formats as well.

> **Tip**
>
> Feel free to check out the SenseGlove Unreal Engine Plugin landing page on Fab as well.

Overview

To help you navigate the SenseGlove Unreal Engine Handbook, we have organized the content into several key sections. This structured layout aims to simplify your journey through the SenseGlove Unreal Engine Plugin, providing clear and detailed guidance at every step.

🚀 Getting Started

This section covers the basics of the SenseGlove Unreal Engine Plugin:

⚙️ Plugin Configuration

This section provides detailed information on configuring the plugin:

- Plugin Settings
    - Initialization
    - Game User Settings
        - Hardware-benchmarking
    - Tracking
        - Glove-tracking
        - Hand-tracking
        - HMD-tracking
        - Wrist-tracking
            - Debugging
    - Virtual Hand
        - Animation
        - Debugging
        - Grab
        - Haptics
        - Mesh
        - Touch
- Overriding Settings

💡 Miscellaneous

Toipcs that do not fall under any specific category:

🛠️ Advanced Topics

For users familiar with the basics, this section explores advanced features of the plugin:

🔌 Low-Level API

This section delves into the SenseGlove low-level API:

📑 Appendix

The appendix contains various extra useful information:

📜 License

Plugin Installation

The SenseGlove Unreal Engine Plugin could be installed using various methods:

- Via the Epic Games Launcher by navigating to the SenseGlove Unreal Engine Plugin landing page on Fab.
- Via the SenseGlove Unreal Engine Plugin Microsoft Azure DevOps repository.

In the following chapters, we discover each of those methods:

- Installation via the Epic Games Launcher
- Installation via Microsoft Azure DevOps Repositories

Video Tutorials

We also have older videos demonstrating both installation methods on Microsoft Windows and GNU Linux in more detail.

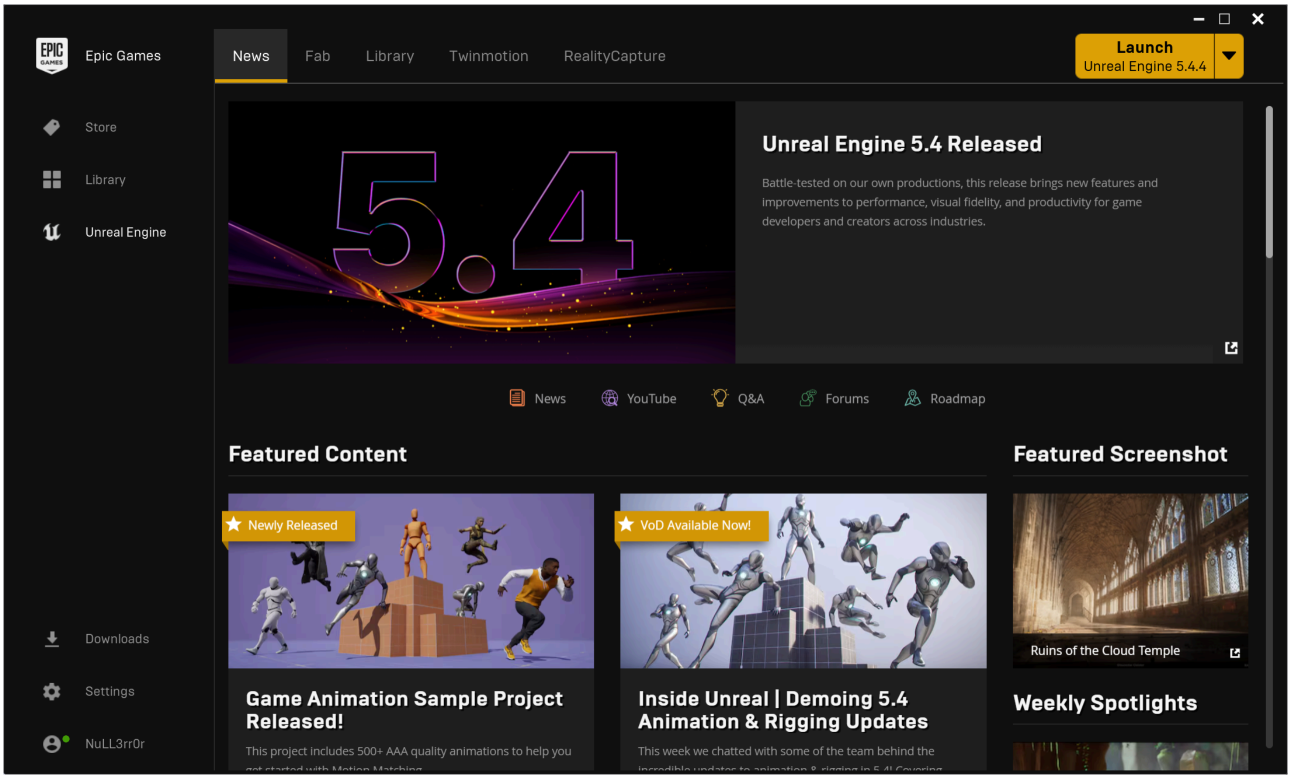
- Plugin installation guide for Microsoft Windows:

- Plugin and examples installation guide for GNU/Linux:

Plugin Installation via the Epic Games Launcher

Before beginning the plugin installation via the Epic Games Launcher, ensure you have signed into your Epic Games account on the Epic Games Launcher and that you have a supported version of Unreal Engine installed. Supported engine versions can be found in the Platform Support Matrix.

1. Run the Epic Games Launcher.



Unreal Engine 5.4 Released

Battle-tested on our own productions, this release brings new features and improvements to performance, visual fidelity, and productivity for game developers and creators across industries.

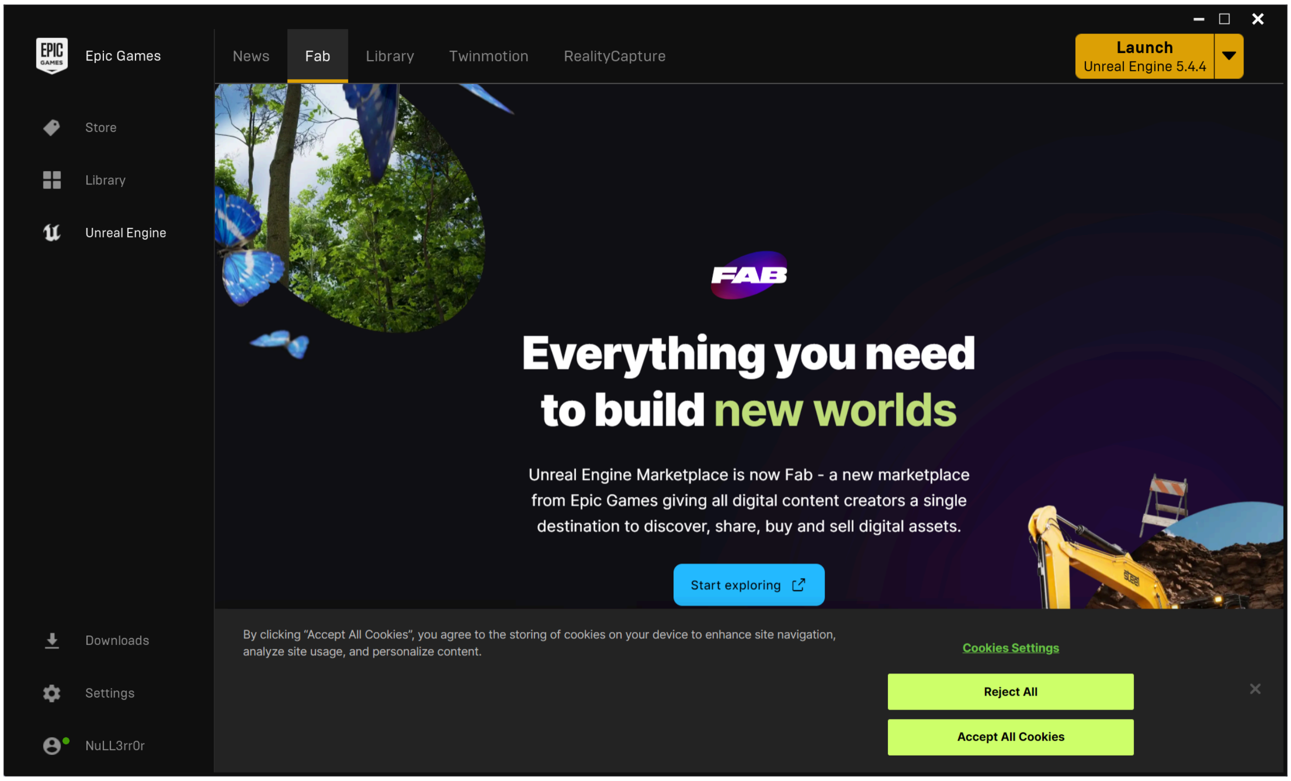
Game Animation Sample Project Released!

This project includes 500+ AAA quality animations to help you

Inside Unreal | Demoing 5.4 Animation & Rigging Updates

This week we chatted with some of the team behind the

2. Navigate to the `Fab` tab and click `Start exploring` button which in turn opens your default web browser pointing to the Fab home page.

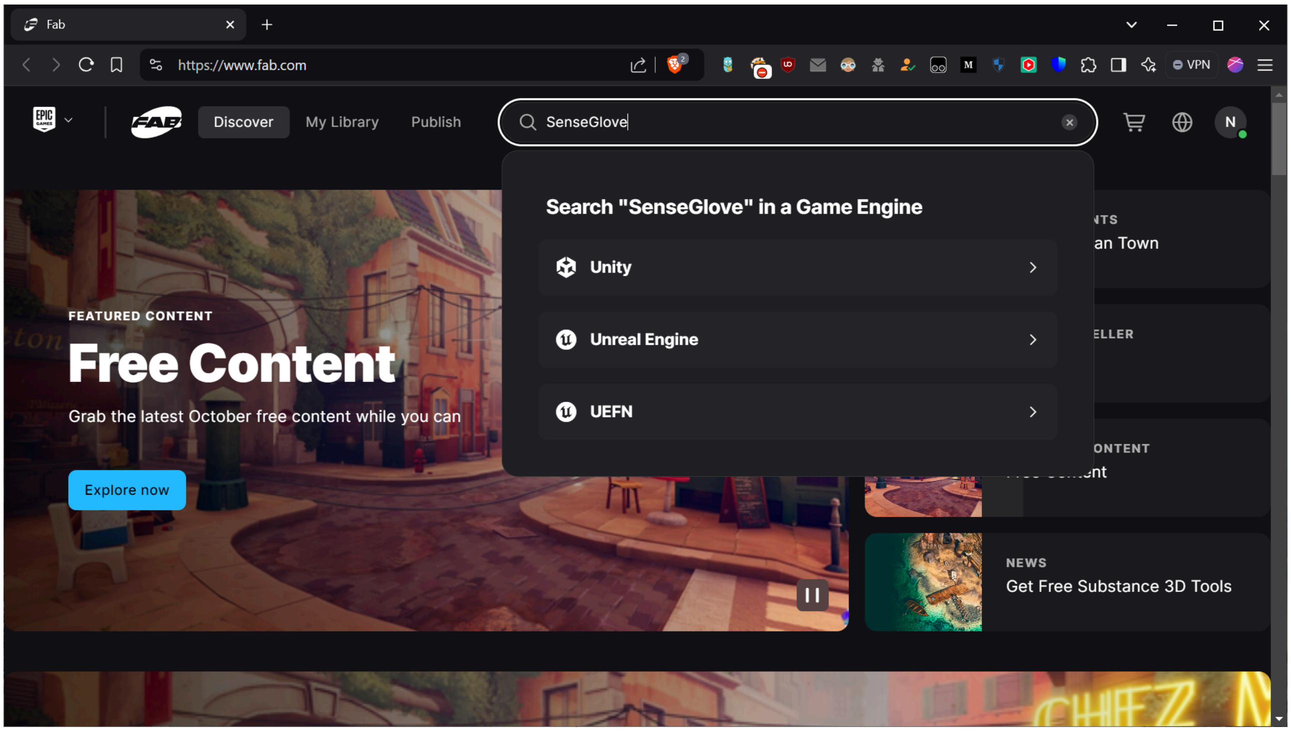


Everything you need to build new worlds

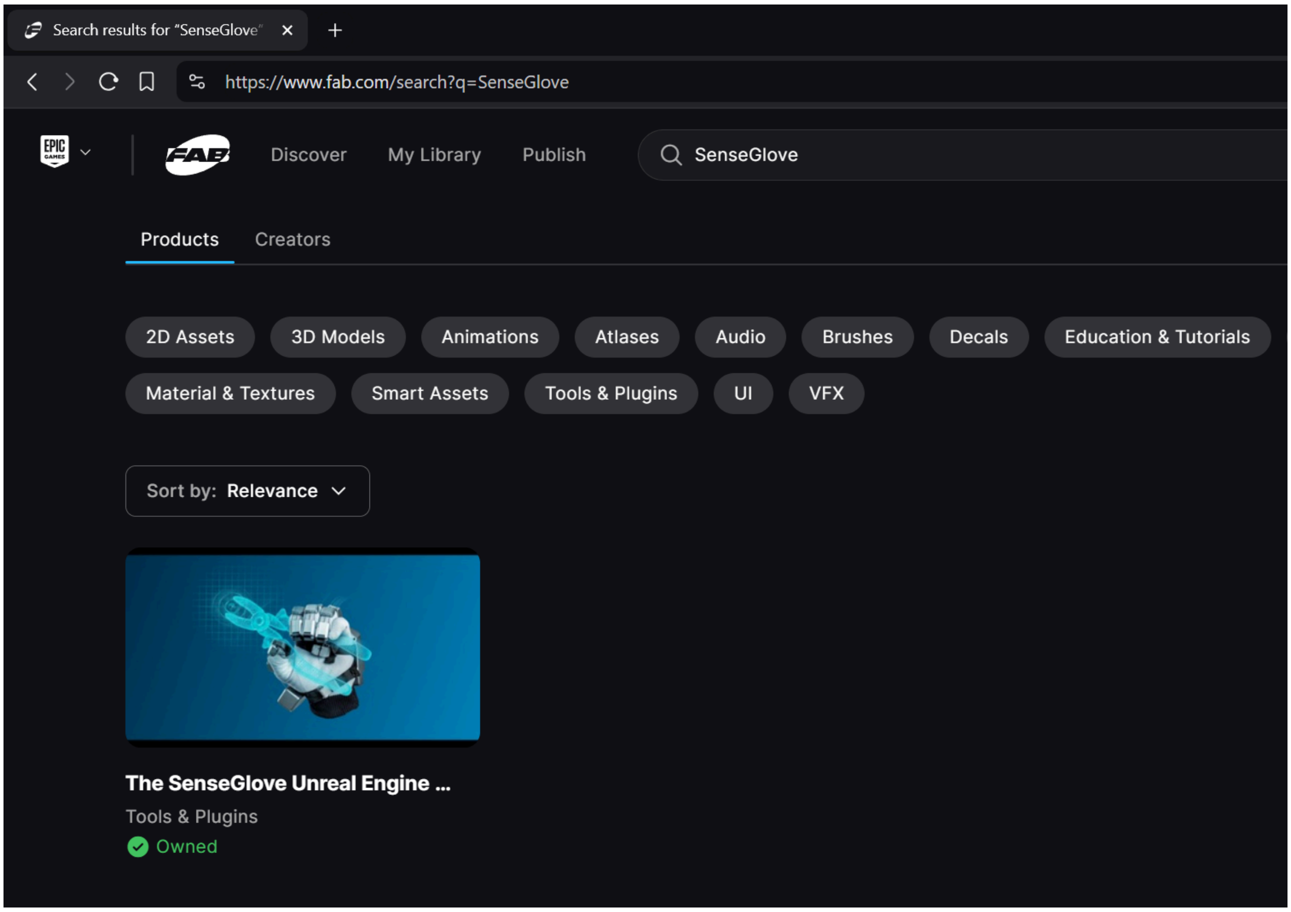
Unreal Engine Marketplace is now Fab - a new marketplace from Epic Games giving all digital content creators a single destination to discover, share, buy and sell digital assets.

By clicking "Accept All Cookies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and personalize content.

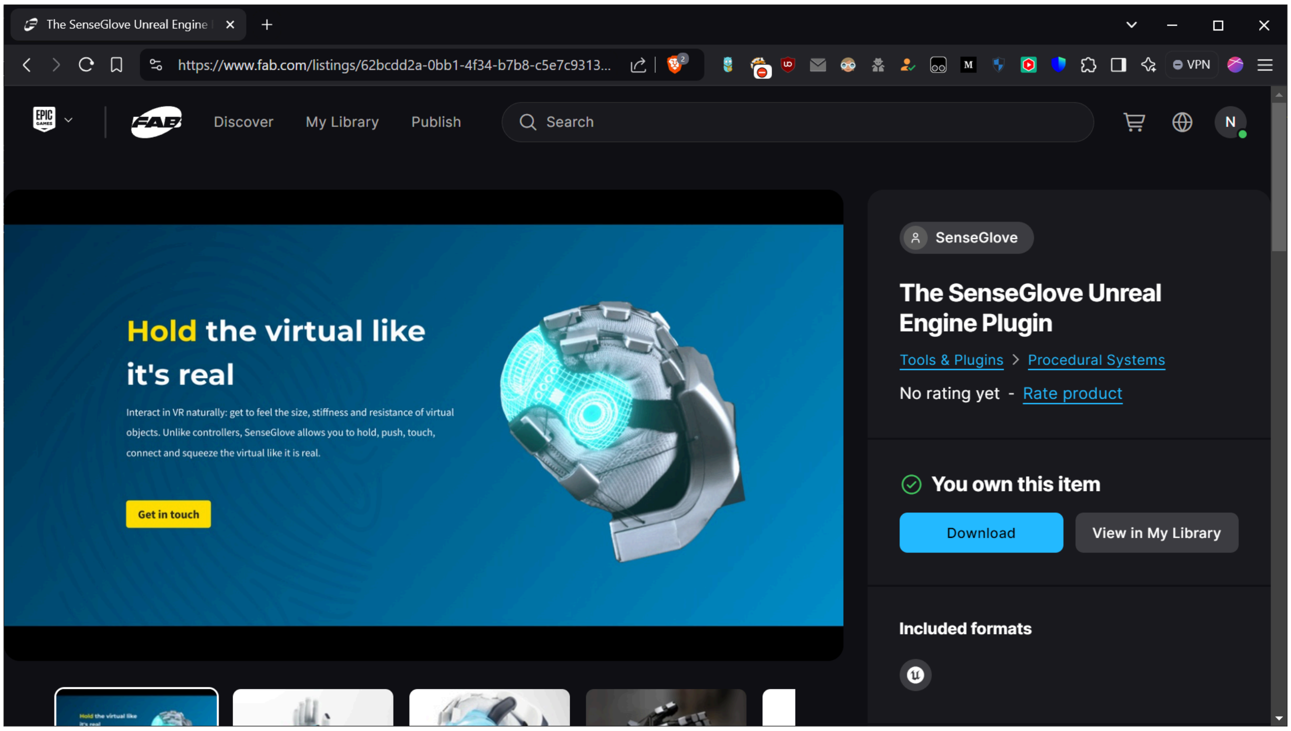
3. On the Fab home page, enter the term `SenseGlove` in the search box and press Enter. Alternatively, you can go directly to the SenseGlove Unreal Engine Plugin landing page on Fab directly instead of taking the above two steps.



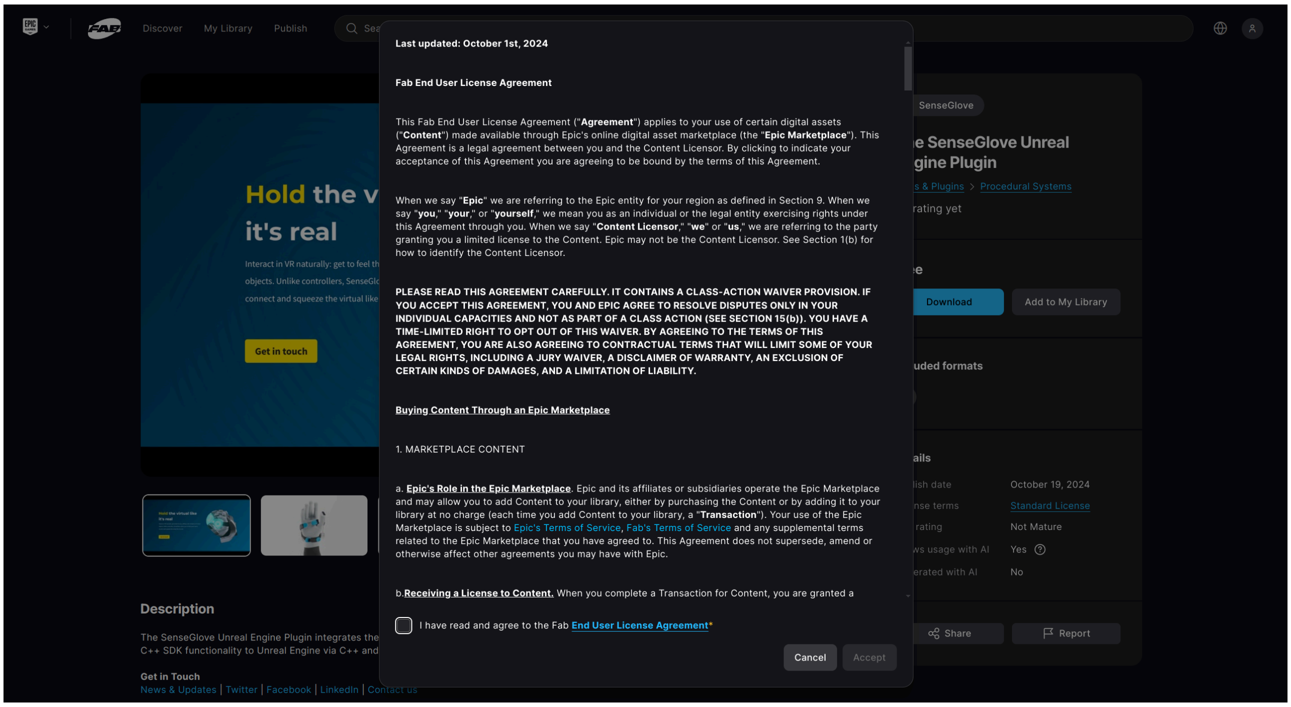
4. Click on the `SenseGlove Unreal Engine Plugin` in the search results to navigate to its dedicated page.



5. On the SenseGlove Unreal Engine Plugin landing page on Fab click the `Download` button.


The SenseGlove Unreal Engine
https://www.fab.com/listings/62bcdd2a-0bb1-4f34-b7b8-c5e7c9313...
Discover
My Library
Publish
Search
Hold the virtual like it's real
Interact in VR naturally: get to feel the size, stiffness and resistance of virtual objects. Unlike controllers, SenseGlove allows you to hold, push, touch, connect and squeeze the virtual like it is real.
Get in touch
SenseGlove
The SenseGlove Unreal Engine Plugin
Tools & Plugins > Procedural Systems
No rating yet - Rate product
You own this item
Download
View in My Library
Included formats

6. If this is your first download from Fab, you will need to agree to the Fab End User License Agreement (EULA) before proceeding.


Last updated: October 1st, 2024

Fab End User License Agreement

This Fab End User License Agreement ("**Agreement**") applies to your use of certain digital assets ("**Content**") made available through Epic's online digital asset marketplace (the "**Epic Marketplace**"). This Agreement is a legal agreement between you and the Content Licensor. By clicking to indicate your acceptance of this Agreement you are agreeing to be bound by the terms of this Agreement.

When we say "**Epic**" we are referring to the Epic entity for your region as defined in Section 9. When we say "**you**," "**your**," or "**yourself**," we mean you as an individual or the legal entity exercising rights under this Agreement through you. When we say "**Content Licensor**," "**we**" or "**us**," we are referring to the party granting you a limited license to the Content. Epic may not be the Content Licensor. See Section 1(b) for how to identify the Content Licensor.

PLEASE READ THIS AGREEMENT CAREFULLY. IT CONTAINS A CLASS-ACTION WAIVER PROVISION. IF YOU ACCEPT THIS AGREEMENT, YOU AND EPIC AGREE TO RESOLVE DISPUTES ONLY IN YOUR INDIVIDUAL CAPACITIES AND NOT AS PART OF A CLASS ACTION (SEE SECTION 15(b)). YOU HAVE A TIME-LIMITED RIGHT TO OPT OUT OF THIS WAIVER. BY AGREEING TO THE TERMS OF THIS AGREEMENT, YOU ARE ALSO AGREEING TO CONTRACTUAL TERMS THAT WILL LIMIT SOME OF YOUR LEGAL RIGHTS, INCLUDING A JURY WAIVER, A DISCLAIMER OF WARRANTY, AN EXCLUSION OF CERTAIN KINDS OF DAMAGES, AND A LIMITATION OF LIABILITY.

Buying Content Through an Epic Marketplace

1. MARKETPLACE CONTENT

a. **Epic's Role in the Epic Marketplace**. Epic and its affiliates or subsidiaries operate the Epic Marketplace and may allow you to add Content to your library, either by purchasing the Content or by adding it to your library at no charge (each time you add Content to your library, a "**Transaction**"). Your use of the Epic Marketplace is subject to Epic's Terms of Service, Fab's Terms of Service and any supplemental terms related to the Epic Marketplace that you have agreed to. This Agreement does not supersede, amend or otherwise affect other agreements you may have with Epic.

b. **Receiving a License to Content.** When you complete a Transaction for Content, you are granted a

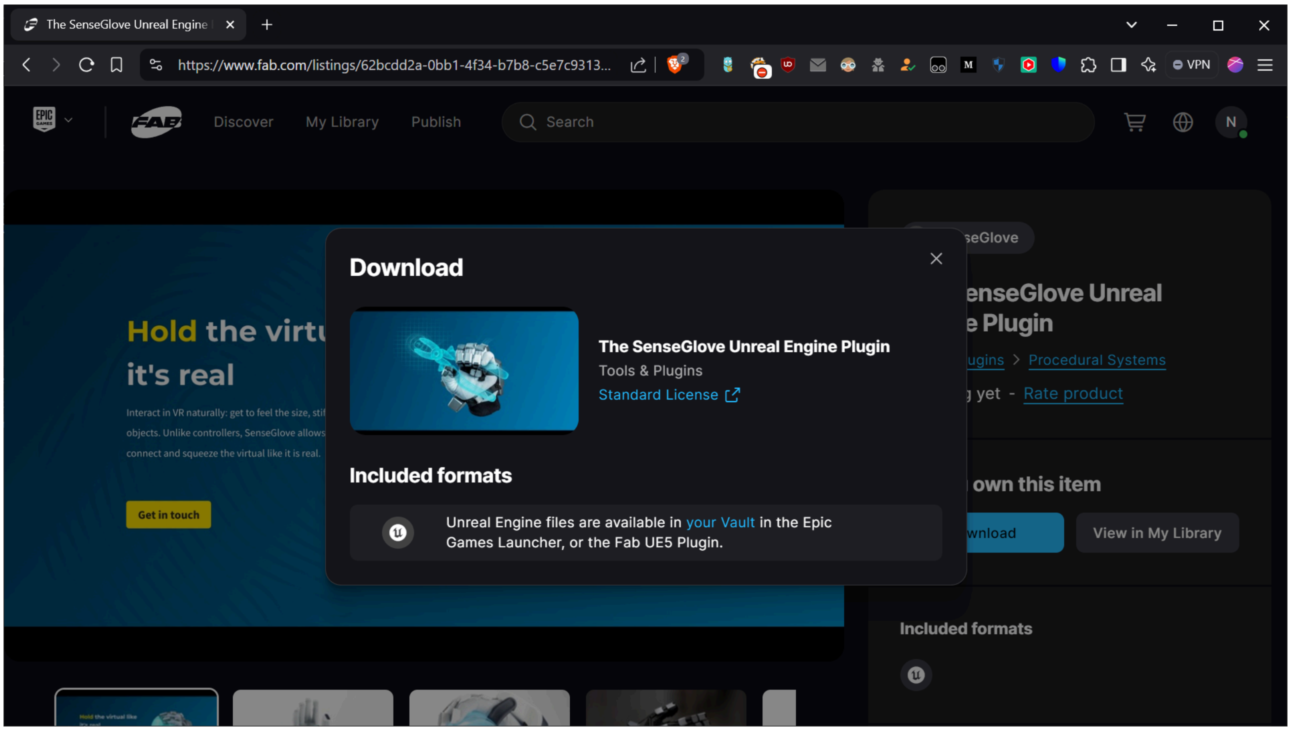
☐ I have read and agree to the Fab End User License Agreement*

Cancel Accept

Discover
My Library
Publish
Hold the v
it's real
Get in touch
SenseGlove
e SenseGlove Unreal
gine Plugin
s & Plugins > Procedural Systems
rating yet
Download
Add to My Library
uded formats
ails
lish date October 19, 2024
nse terms Standard License
rating Not Mature
ws usage with AI Yes
erated with AI No
Share
Report
Description
The SenseGlove Unreal Engine Plugin integrates the
C++ SDK functionality to Unreal Engine via C++ and
Get in Touch
News & Updates | Twitter | Facebook | LinkedIn | Contact us

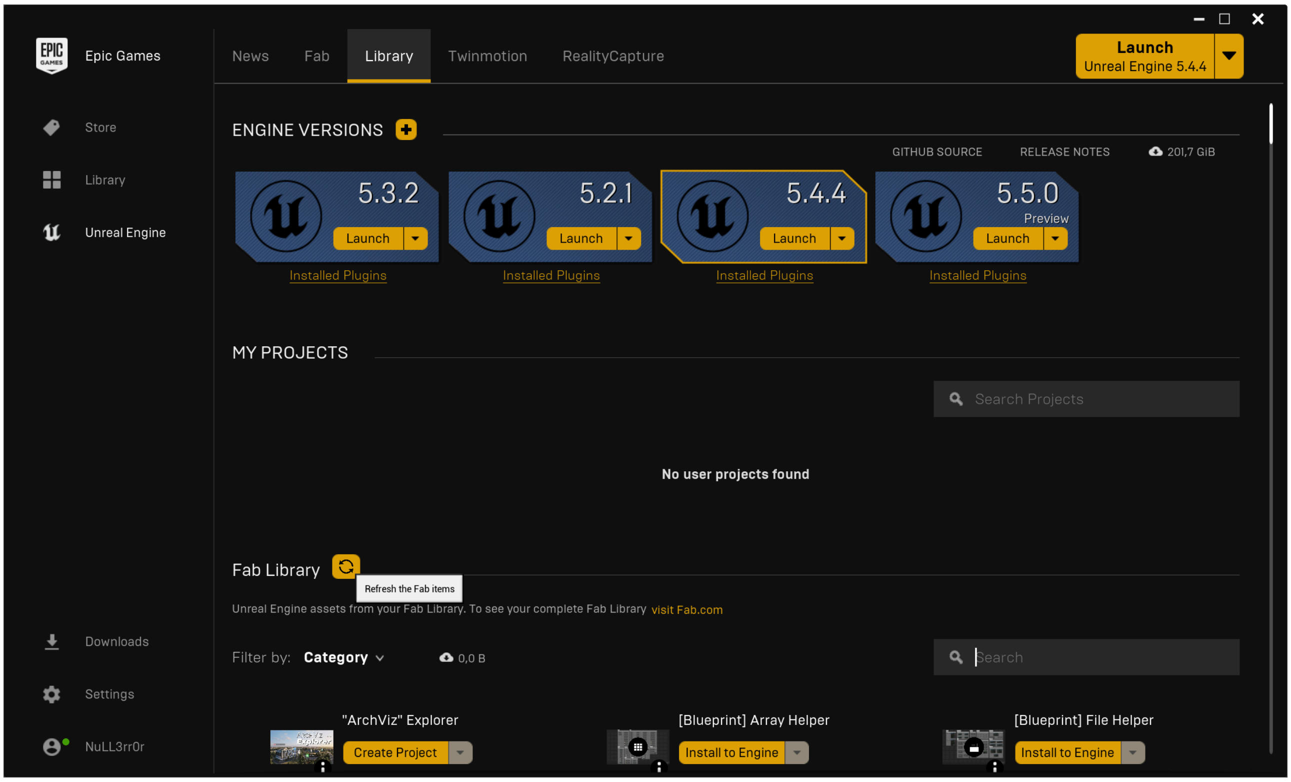
7. After clicking `Download`, a pop-up will notify you that the plugin is available in your Vault in the Epic Games Launcher, or the Fab UE5 Plugin.

> **Note**
>
> According to the Fab launch announcement:
>
> The Fab integration in UEFN is undergoing maintenance and will be back online shortly, and the Fab integration in the Unreal Engine 5 Editor is coming soon.

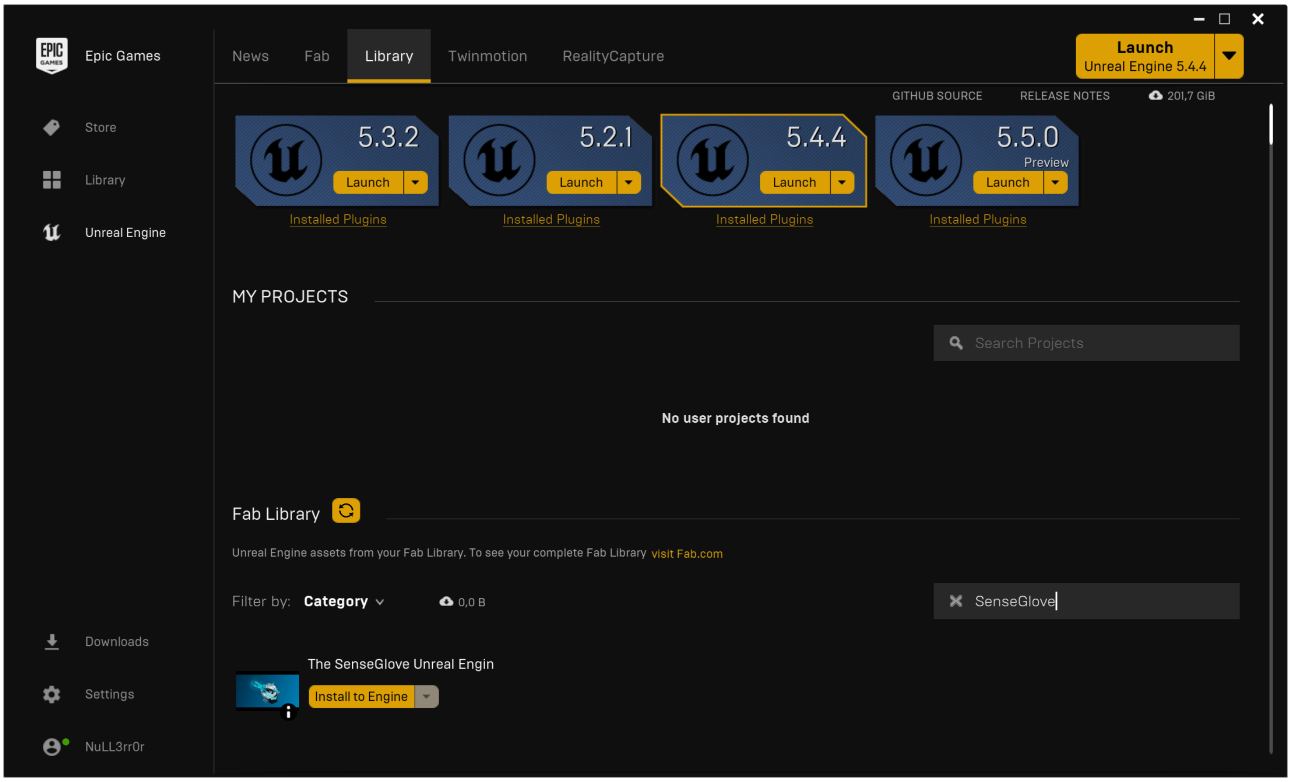


Unreal Engine files are available in your Vault in the Epic Games Launcher, or the Fab UE5 Plugin.

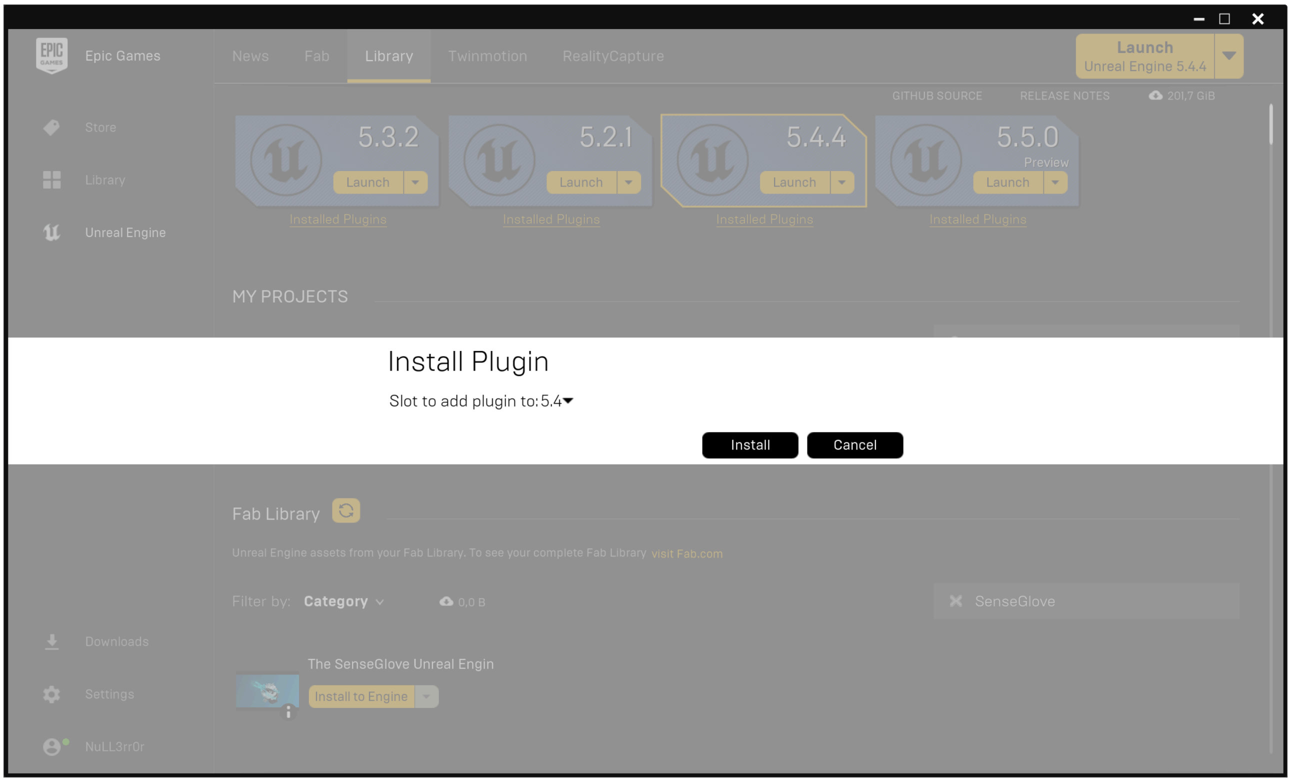
8. Go back to the Epic Games Launcher, navigate to the `Library` tab, and in the `Fab Library` section, click the `Refresh Fab items` button.



9. Once the Fab library is refreshed and synchronized, use the Vault search box to find the `SenseGlove Unreal Engine Plugin`. Click the `Install to Engine` button.



10. You'll be prompted to choose a compatible engine version. Select your desired engine version from the list, then click `Install`.

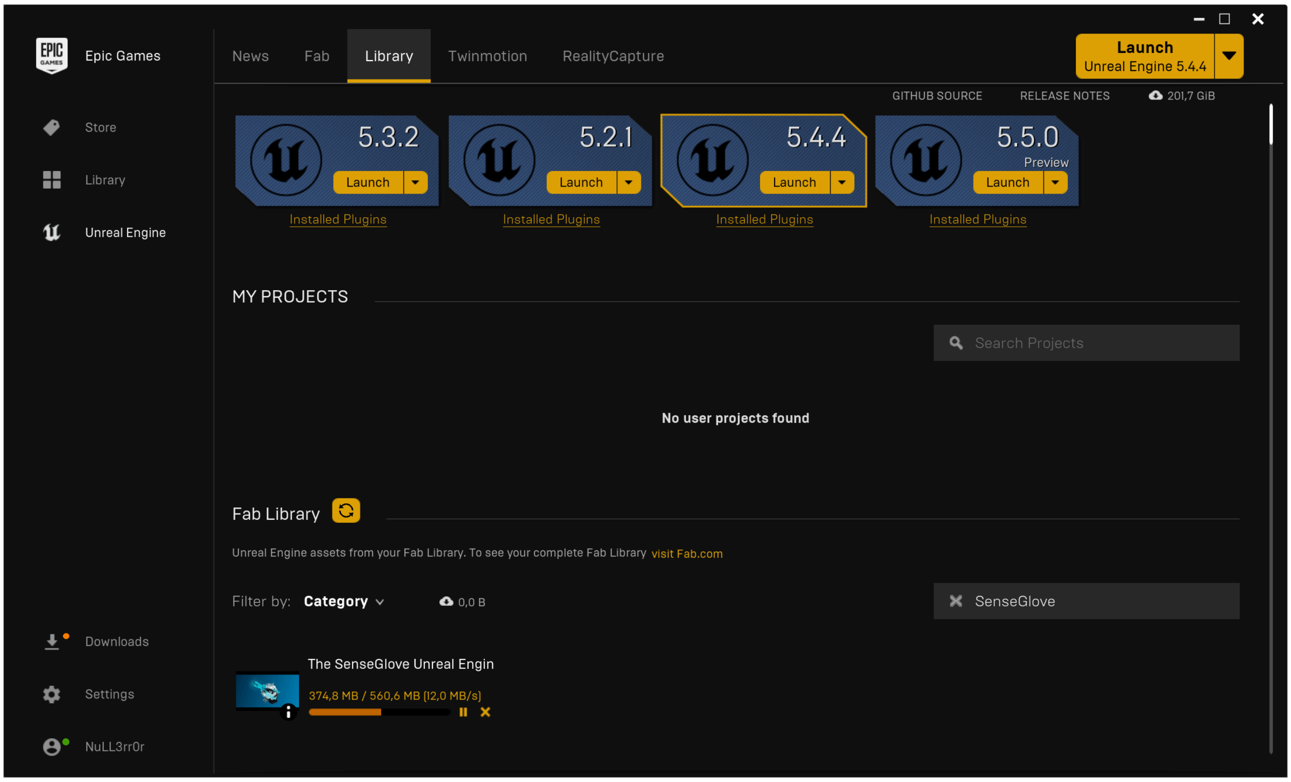


Install Plugin

Slot to add plugin to: 5.4

Unreal Engine assets from your Fab Library. To see your complete Fab Library visit Fab.com

11. The Epic Games Launcher will show the plugin's download and installation progress. Please wait for it to complete.



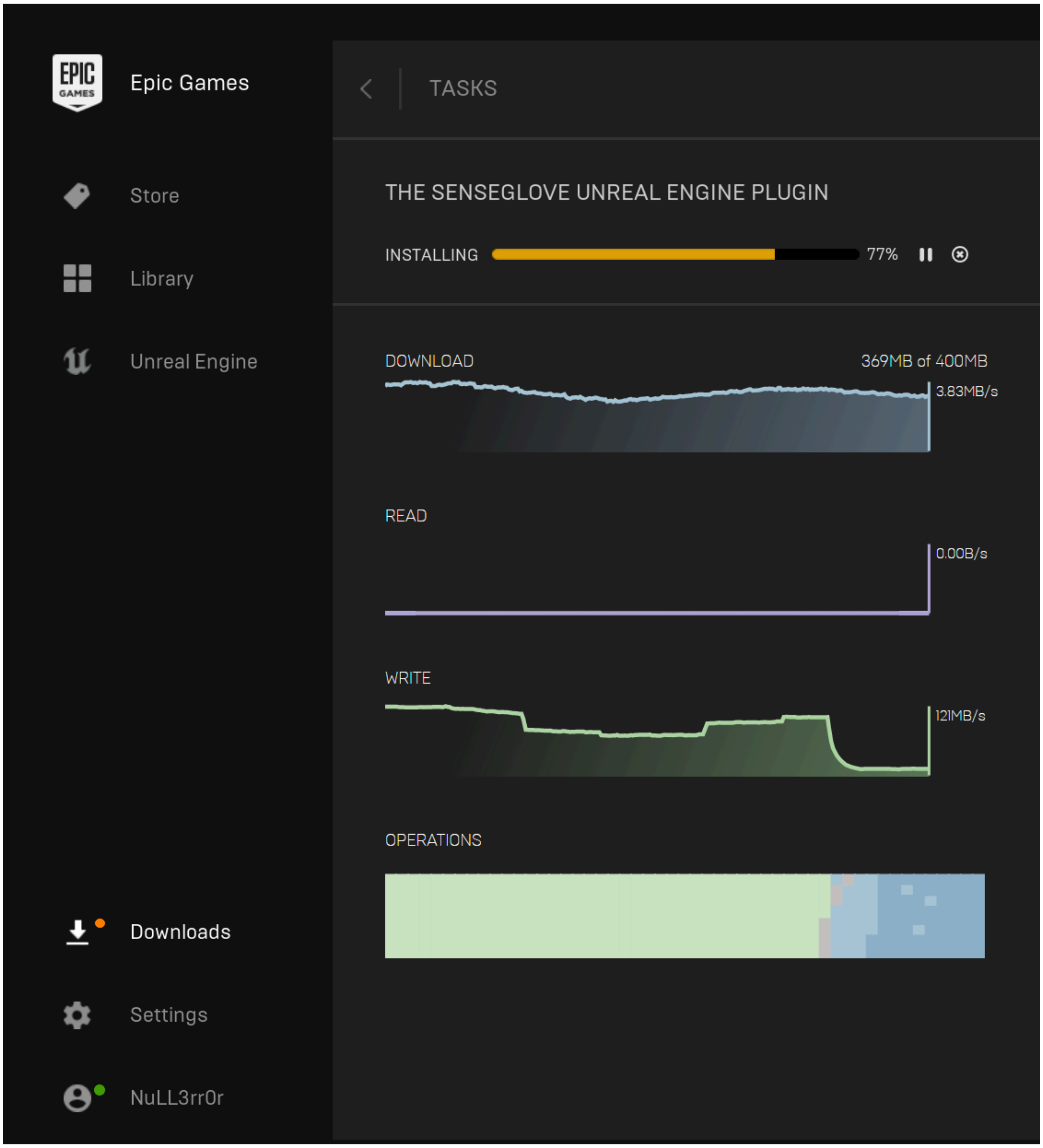
No user projects found

Unreal Engine assets from your Fab Library. To see your complete Fab Library visit Fab.com

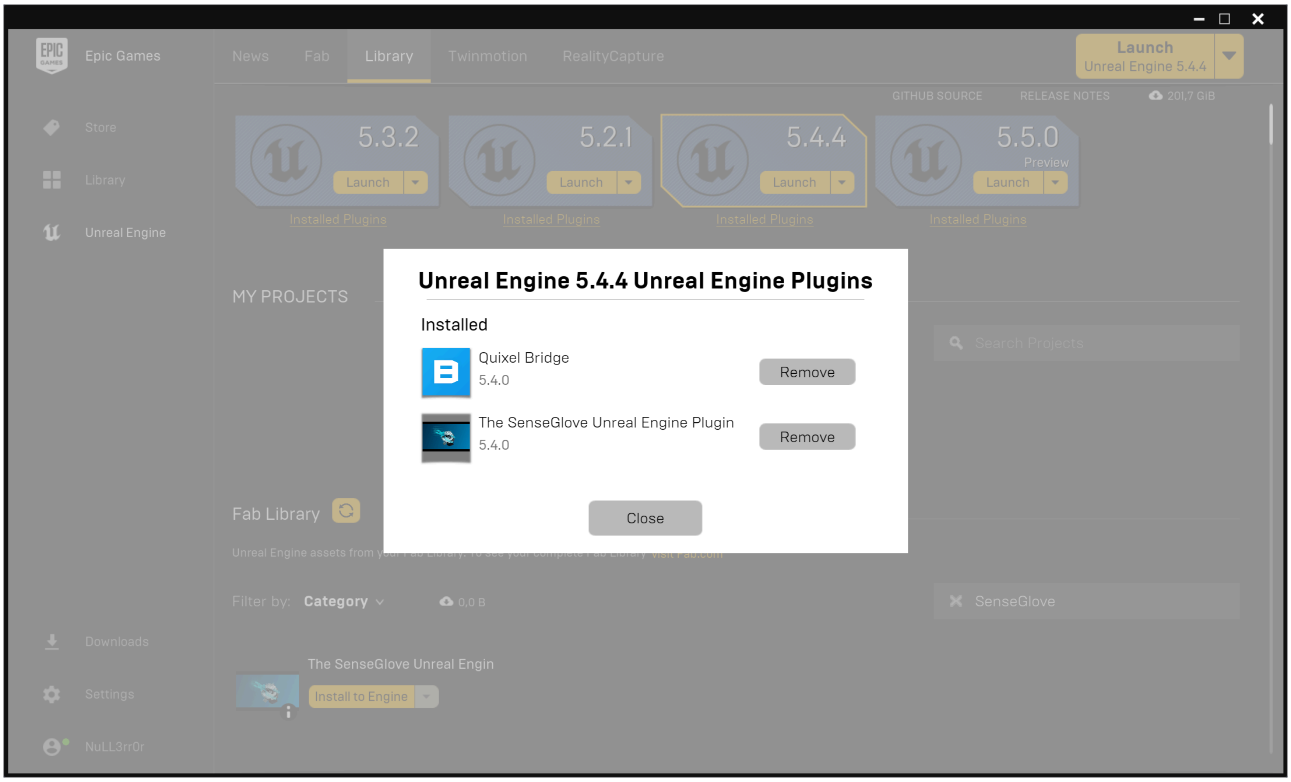
The SenseGlove Unreal Engin

374,8 MB / 560,6 MB (12,0 MB/s)

12. While the download and installation are in progress, you can see the progress in more details by clicking on the `Downloads` section on the sidebar.



13. Once the download and installation are complete, verify its installation by clicking `Installed Plugins` under the engine you've just installed it to. The SenseGlove plugin should appear as installed among other currently installed plugins.

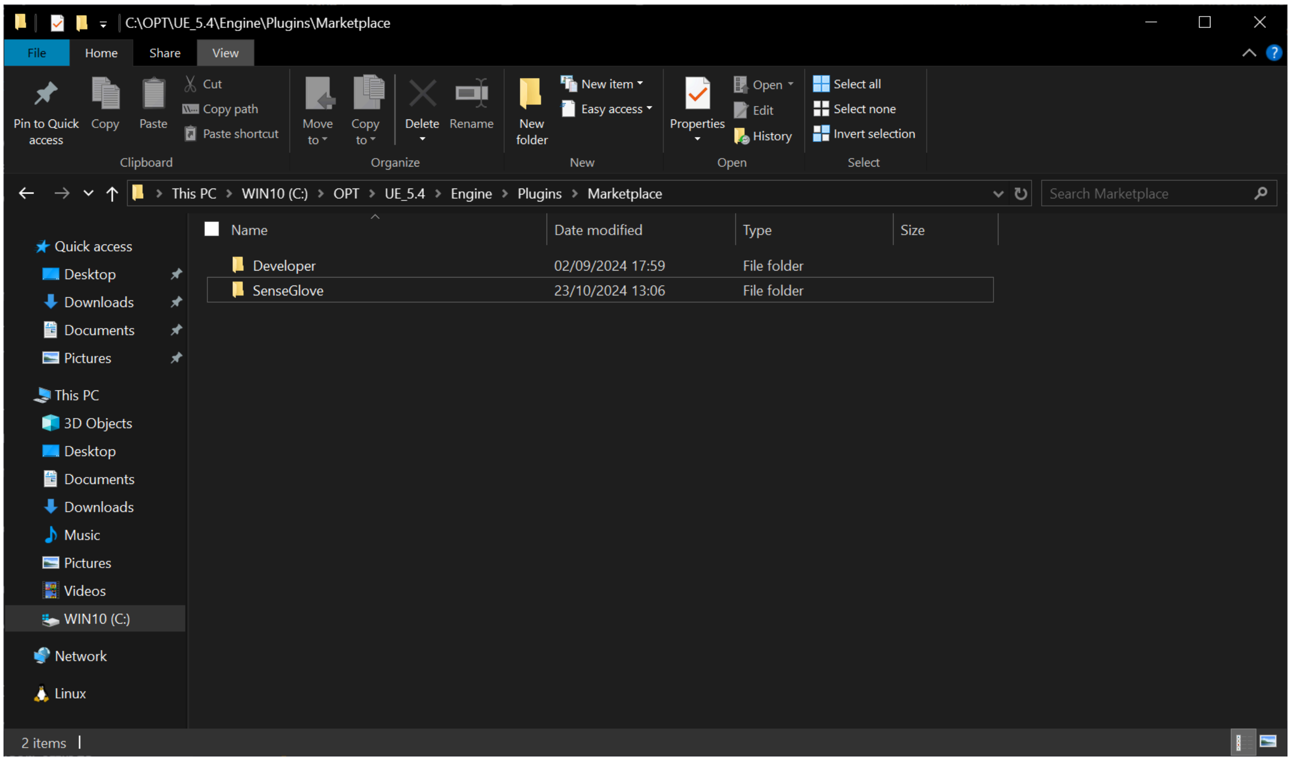

Unreal Engine 5.4.4 Unreal Engine Plugins

Installed

Quixel Bridge
5.4.0

The SenseGlove Unreal Engine Plugin
5.4.0

14. One last confirmation could be navigating to `YourEngineInstallationPath/Engine/Plugins/Marketplace` directory. The SenseGlove Unreal Engine Plugin source and binaries can be found inside this directory. This is especially useful in case one desires to copy the plugin for example to their own project's source code to run it at the project level instead of running it at the engine level.



Name	Date modified	Type	Size
Developer	02/09/2024 17:59	File folder	
SenseGlove	23/10/2024 13:06	File folder	

⚠ **Warning**

Please note that it is best practice to install the plugin either at the project level or the engine level, but not both. Having the plugin installed in both locations, at the same time, can lead to various issues, especially if the version of the plugin installed at the engine level differs from the one installed at the plugin level. A guide on verifying the plugin version is also available as well.

Plugin Installation via Microsoft Azure DevOps Repositories

While plugin installation via the Epic Games Launcher is the most convenient method for most users to obtain and install the latest version of the SenseGlove Unreal Engine Plugin via Fab, there might be valid reasons to instead download and install the plugin directly from the SenseGlove Unreal Engine Plugin Microsoft Azure DevOps Repository. These reasons may include:

- Downloading an older version that is no longer available on Fab.
- Downloading a recent version that has been submitted to Fab, but is still awaiting approval and publication by the Fab Team.
- Downloading an under-development, unstable release of the plugin for testing purposes.
- Or, any other specific needs that require direct access to the repository.

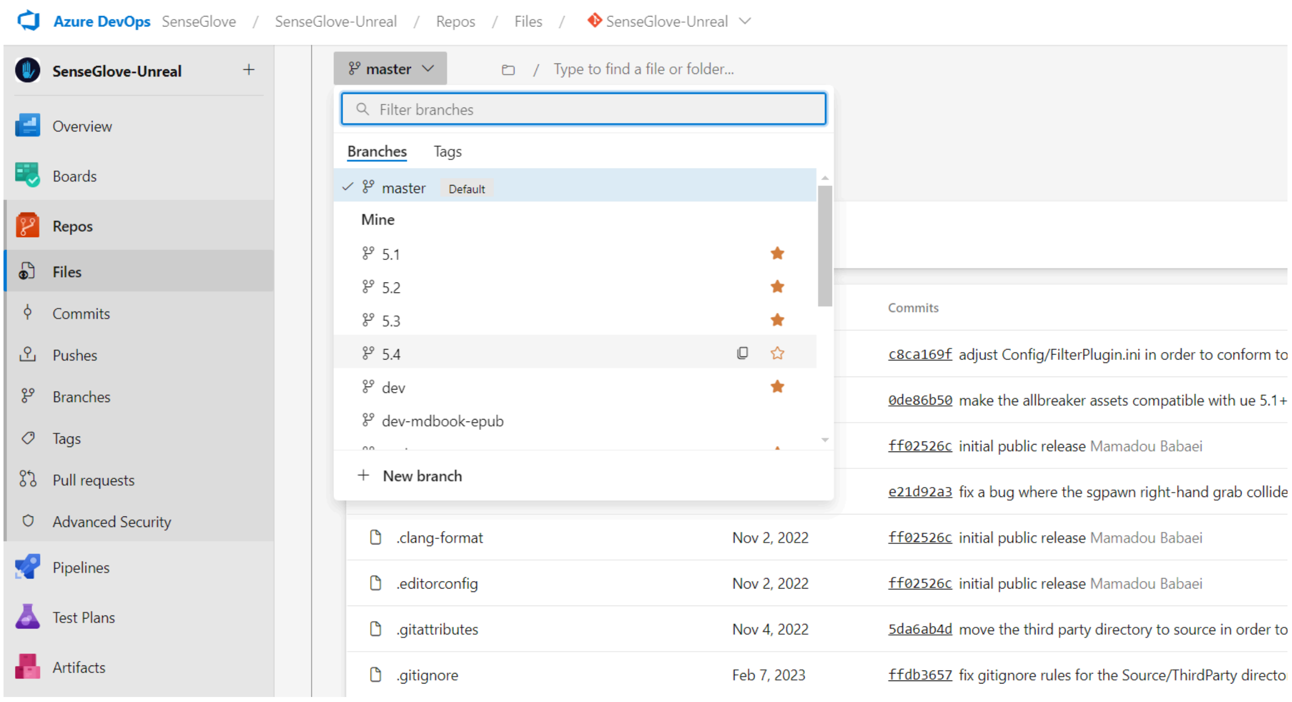
Nonetheless, here is a step-by-step guide to downloading and installing the plugin from the Microsoft Azure DevOps Repositories.

Download a Specific Version

To download a specific version of the plugin, follow these steps:

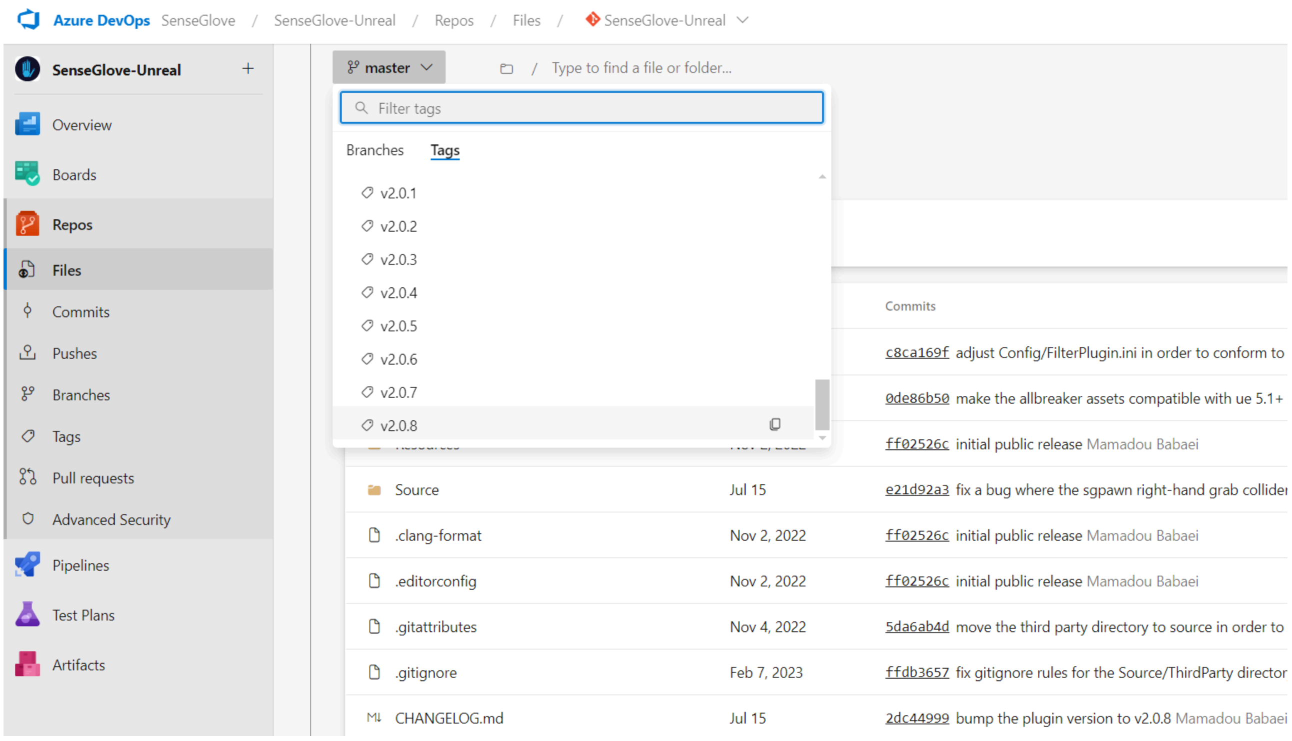
1. Navigate to the the SenseGlove Unreal Engine Plugin Microsoft Azure DevOps Repository.

2. Locate the branch dropdown menu at the top of the page, just below the navigation bar, and next to the `Copy to clipboard` icon. There you'll find a dropdown menu. By default, it usually selects the master branch.



c8ca169f adjust Config/FilterPlugin.ini in order to conform to
0de86b50 make the allbreaker assets compatible with ue 5.1+
ff02526c initial public release Mamadou Babaei
e21d92a3 fix a bug where the sgpawn right-hand grab collide
.clang-format Nov 2, 2022 ff02526c initial public release Mamadou Babaei
.editorconfig Nov 2, 2022 ff02526c initial public release Mamadou Babaei
.gitattributes Nov 4, 2022 5da6ab4d move the third party directory to source in order to
.gitignore Feb 7, 2023 ffdb3657 fix gitignore rules for the Source/ThirdParty directo

3. Use the dropdown menu to choose a desired branch containing the source code for a specific version of Unreal Engine or a specific release of the plugin marked with a release tag.



c8ca169f adjust Config/FilterPlugin.ini in order to conform to
0de86b50 make the allbreaker assets compatible with ue 5.1+
ff02526c initial public release Mamadou Babaei
Source Jul 15 e21d92a3 fix a bug where the sgpawn right-hand grab collider
.clang-format Nov 2, 2022 ff02526c initial public release Mamadou Babaei
.editorconfig Nov 2, 2022 ff02526c initial public release Mamadou Babaei
.gitattributes Nov 4, 2022 5da6ab4d move the third party directory to source in order to
.gitignore Feb 7, 2023 ffdb3657 fix gitignore rules for the Source/ThirdParty director
CHANGELOG.md Jul 15 2dc44999 bump the plugin version to v2.0.8 Mamadou Babaei

Note

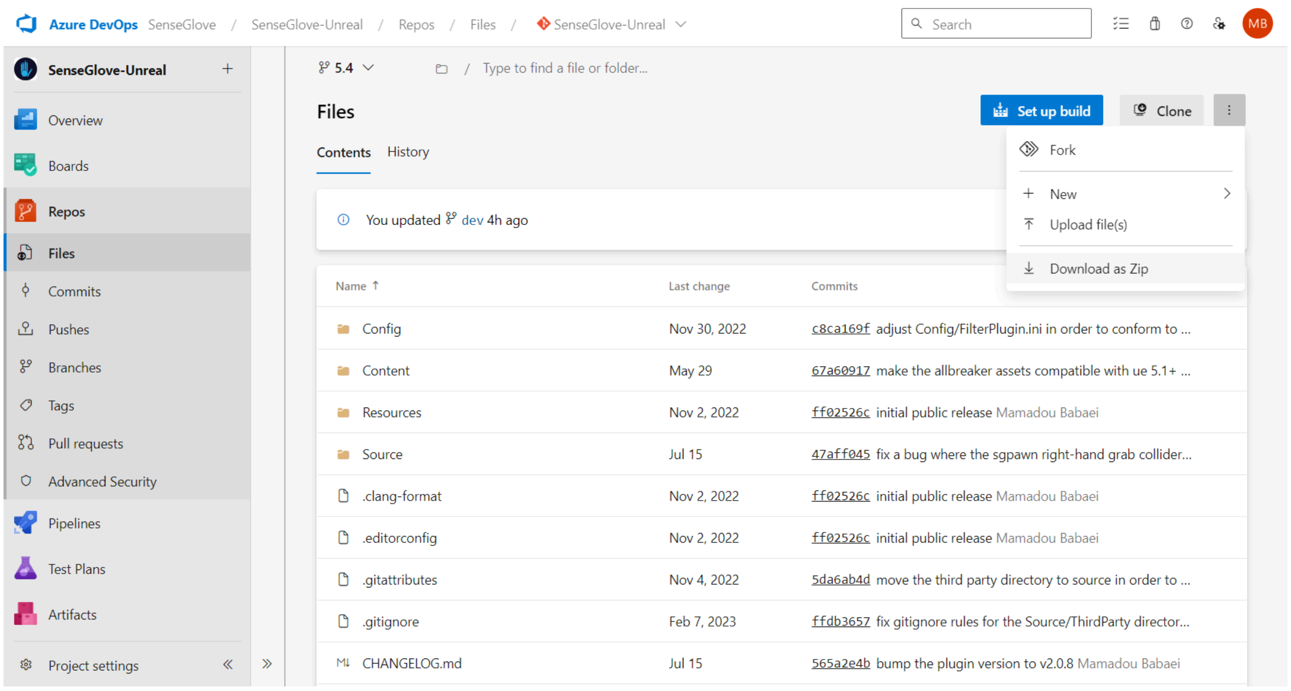
A branch named with engine version numbers, such as `5.4`, `5.3`, etc., ususally contains the source code for the latest stable version of the plugin compatible with that specific Unreal Engine version, provided that version is still supported. For a comprehensive list of supported engine versions please refer to the Platform Support Matrix.

As a general rule of thumb, the `master` branch should work with any supported Unreal Engine version. This is because it does not specify any `EngineVersion` inside the main `.uplugin` file. However, there may be rare exceptions where it does not work due to breaking changes between engine versions that the plugin cannot accommodate. One such a instance occurred with version `2.0.x` of the plugin, where some breaking changes prevented UE `5.1` from sharing similar code with versions `5.2+`. For this reason, it is generally recommended to select a branch specific to the version of the Unreal Engine you intend to use with the plugin.

The same principles that apply to the `master` branch also apply to the `dev` branch, which will discuss later.

We will also cover how to obtain a working version from a tag for scenarios like the one mentioned above.

4. After selecting your desired branch or tag, click on the kebab menu (three vertical dots) located at the top right of the screen and choose `Download as Zip` to obtain the source code for that branch or tag.

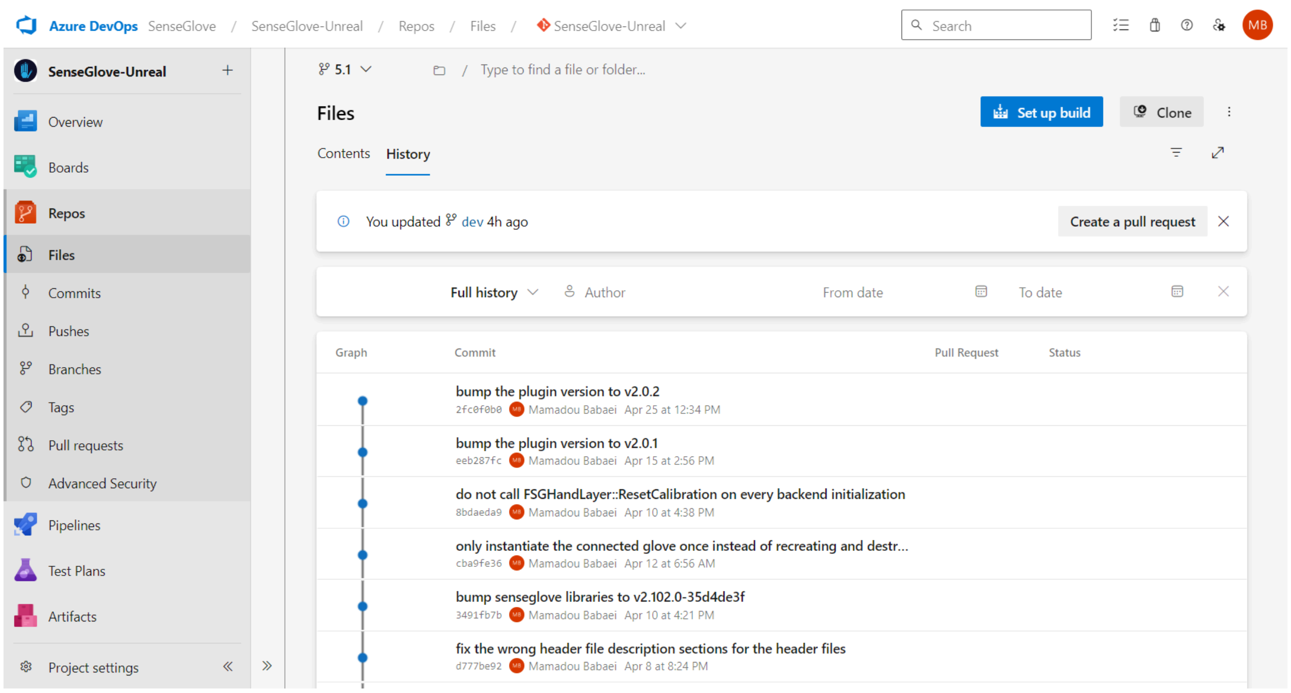


Name ↑	Last change	Commits
Config	Nov 30, 2022	c8ca169f adjust Config/FilterPlugin.ini in order to conform to ...
Content	May 29	67a60917 make the allbreaker assets compatible with ue 5.1+ ...
Resources	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
Source	Jul 15	47aff045 fix a bug where the sgpawn right-hand grab collider...
.clang-format	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
.editorconfig	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
.gitattributes	Nov 4, 2022	5da6ab4d move the third party directory to source in order to ...
.gitignore	Feb 7, 2023	ffdb3657 fix gitignore rules for the Source/ThirdParty director...
CHANGELOG.md	Jul 15	565a2e4b bump the plugin version to v2.0.8 Mamadou Babaei

Download a Specific Version for a Specifc Unreal Engine Version

As mentioned earlier, due to breaking changes between Unreal Engine versions, it might not be feasible to share the same source code across different Unreal Engine versions. Since release tags are created from the `master` branch, they contain code compatible only with the latest version of Unreal Engine. Therefore, the instructions for downloading a specific version from a release tag might not work with some Unreal Engine versions. In such cases, you can use an alternative approach:

1. First, choose the appropriate branch for your desired Unreal Engine version from the branch dropdown menu, as discussed earlier. Then navigate to the `History` tab.

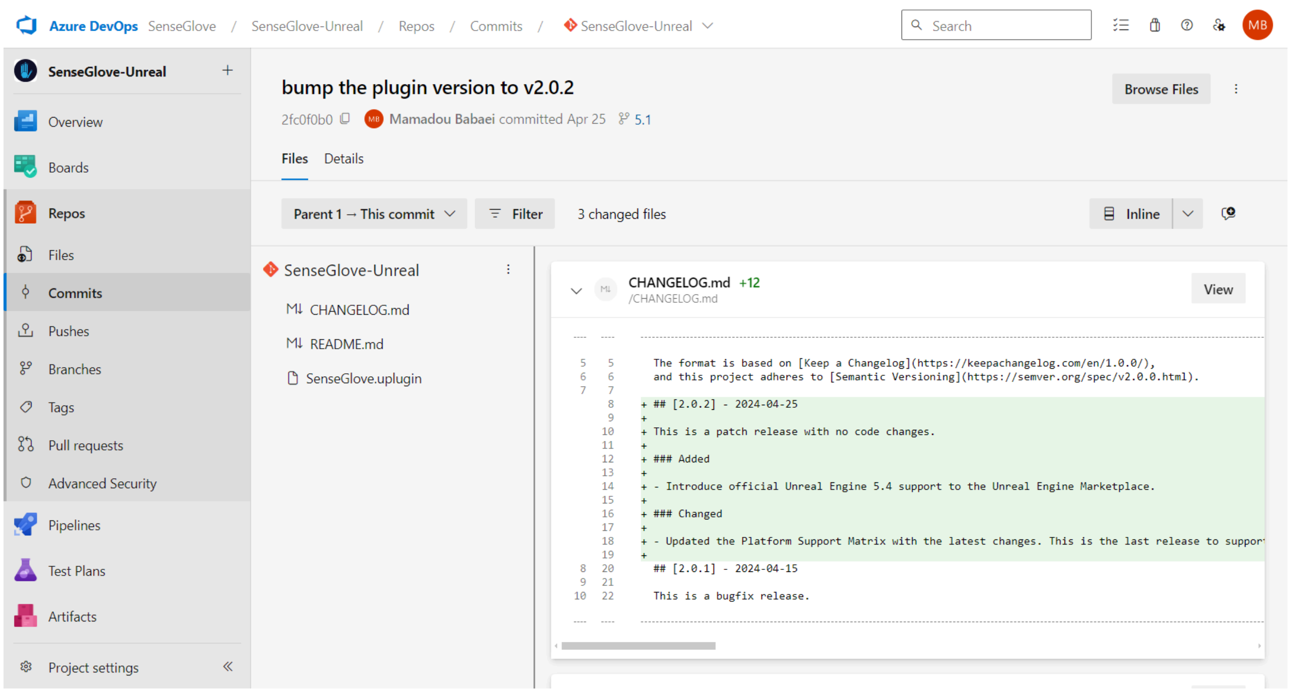


You updated dev 4h ago

Graph	Commit	Pull Request	Status
	bump the plugin version to v2.0.2 2fc0f0b0 Mamadou Babaei Apr 25 at 12:34 PM		
	bump the plugin version to v2.0.1 eeb287fc Mamadou Babaei Apr 15 at 2:56 PM		
	do not call FSGHandLayer::ResetCalibration on every backend initialization 8bdaeda9 Mamadou Babaei Apr 10 at 4:38 PM		
	only instantiate the connected glove once instead of recreating and destr... cba9fe36 Mamadou Babaei Apr 12 at 6:56 AM		
	bump senseglove libraries to v2.102.0-35d4de3f 3491fb7b Mamadou Babaei Apr 10 at 4:21 PM		
	fix the wrong header file description sections for the header files d777be92 Mamadou Babaei Apr 8 at 8:24 PM		

2. Look via the commit history for a commit message that says `bump the plugin version to vX.X.X` as all releases are finalized with this exact commit message and the plugin version. Next, click on the commit message for the version you are looking for.

3. Once you've selected the correct commit, click on the `Browse Files` button next to the kebab menu (three vertical dots) at the top right of the screen.



Azure DevOps SenseGlove / SenseGlove-Unreal / Repos / Commits / SenseGlove-Unreal

bump the plugin version to v2.0.2

2fc0f0b0 · Mamadou Babaei committed Apr 25 · 5.1

Files | Details

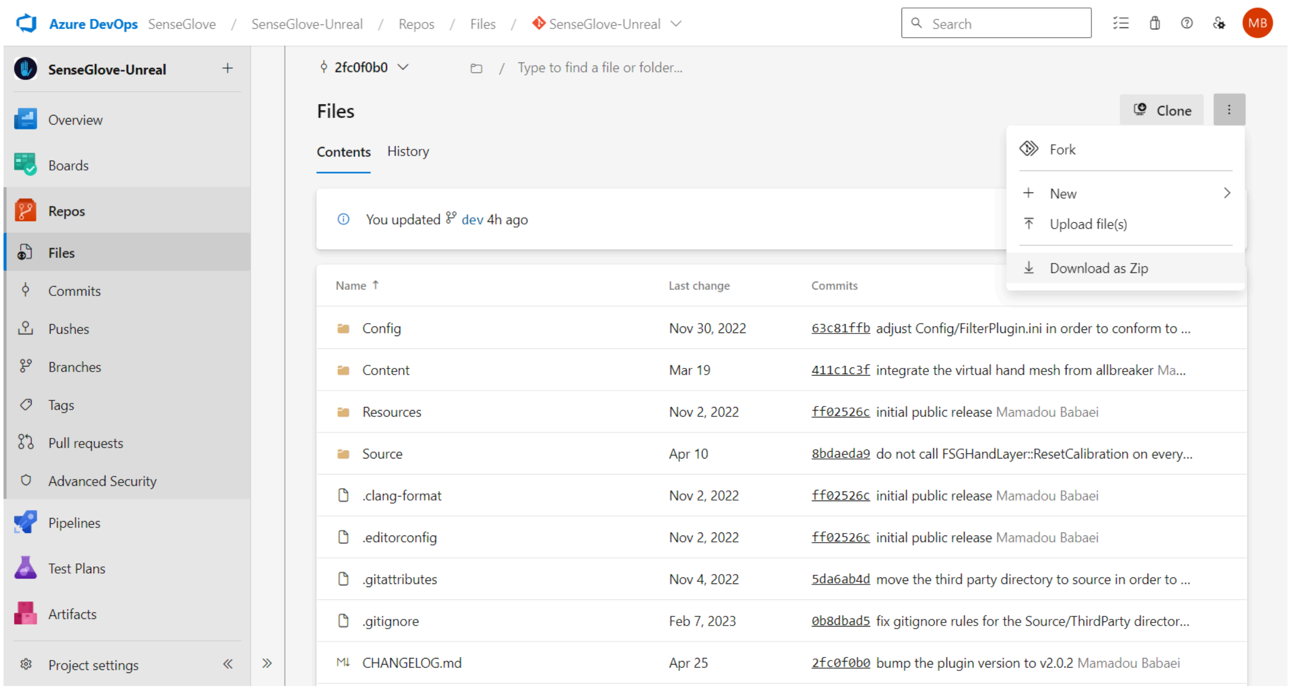
Parent 1 → This commit · Filter · 3 changed files

SenseGlove-Unreal
- CHANGELOG.md
- README.md
- SenseGlove.uplugin

CHANGELOG.md +12
/CHANGELOG.md

```
  The format is based on [Keep a Changelog](https://keepachangelog.com/en/1.0.0/),
  and this project adheres to [Semantic Versioning](https://semver.org/spec/v2.0.0.html).

+ ## [2.0.2] - 2024-04-25
+
+ This is a patch release with no code changes.
+
+ ### Added
+
+ - Introduce official Unreal Engine 5.4 support to the Unreal Engine Marketplace.
+
+ ### Changed
+
+ - Updated the Platform Support Matrix with the latest changes. This is the last release to suppor
+
  ## [2.0.1] - 2024-04-15

  This is a bugfix release.
```

4. You should now be in the `Content` tab, with the branch dropdown menu displaying the commit hash instead of a branch name or tag. Click on the kebab menu (three vertical dots) again, and select `Download as Zip`. This will give you a zip file containing the exact release you need, compatible with your chosen Unreal Engine version.



Azure DevOps SenseGlove / SenseGlove-Unreal / Repos / Files / SenseGlove-Unreal

2fc0f0b0 · Type to find a file or folder...

Files

Contents | History

Clone · Fork · New · Upload file(s) · Download as Zip

You updated dev 4h ago

Name ↑	Last change	Commits
Config	Nov 30, 2022	63c81ffb adjust Config/FilterPlugin.ini in order to conform to ...
Content	Mar 19	411c1c3f integrate the virtual hand mesh from allbreaker Ma...
Resources	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
Source	Apr 10	8bdaeda9 do not call FSGHandLayer::ResetCalibration on every...
.clang-format	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
.editorconfig	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
.gitattributes	Nov 4, 2022	5da6ab4d move the third party directory to source in order to ...
.gitignore	Feb 7, 2023	0b8dbad5 fix gitignore rules for the Source/ThirdParty director...
CHANGELOG.md	Apr 25	2fc0f0b0 bump the plugin version to v2.0.2 Mamadou Babaei

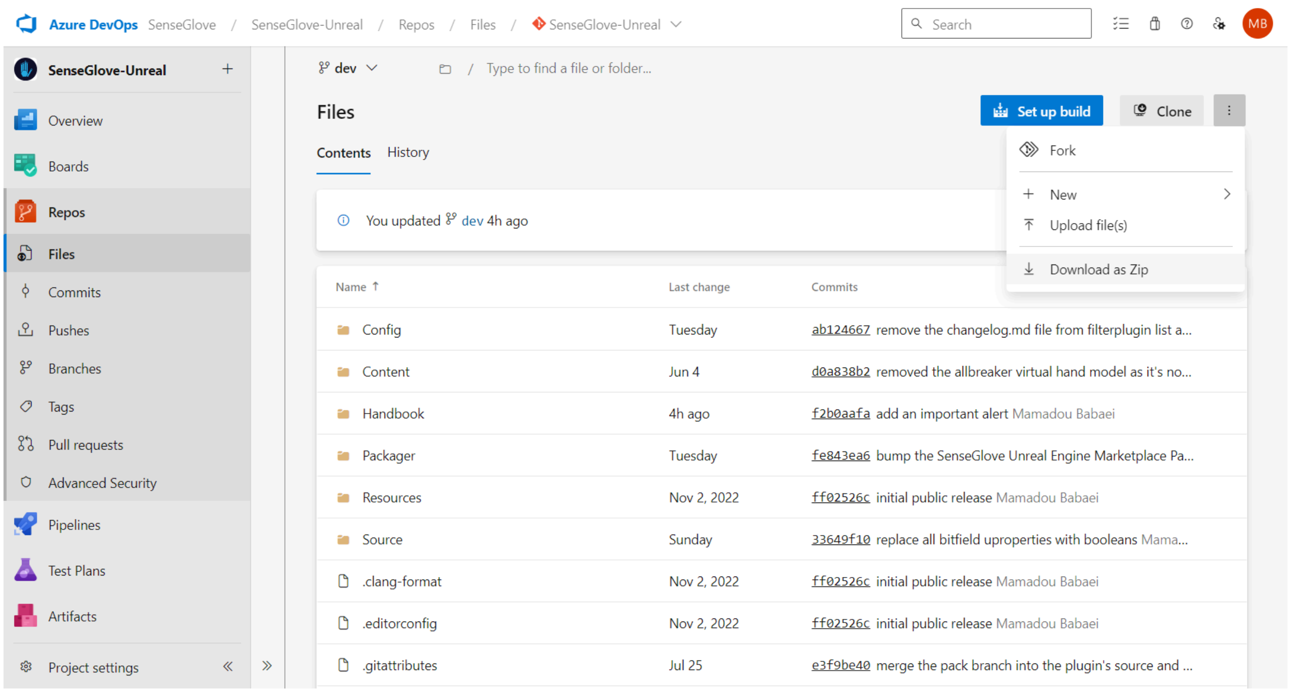
Download the Bleeding-edge Development Branch

Caution

The `dev` branch is an active development branch that is constant and ongoing changes. As a result, the code on this branch is primarily untested and therefore not production-ready. It may not even compile successfully or may lack comprehensive documentation. For any serious development, it is generally recommended to use a stable release of the plugin. The `dev` branch is publicly accessible to give you a preview of upcoming features and for trial purposes only.

The most up-to-date documentation for the dev branch can usually be found at: at: https://unreal.dev.senseglove.com/next.

Downloading the `dev` branch is as easy as choosing the `dev` branch from the branch dropdown menu (as discussed earlier) and then choosing `Download as Zip` from the kebab menu (three vertical dots).



You updated dev 4h ago

Name ↑	Last change	Commits
Config	Tuesday	ab124667 remove the changelog.md file from filterplugin list a...
Content	Jun 4	d0a838b2 removed the allbreaker virtual hand model as it's no...
Handbook	4h ago	f2b0aafa add an important alert Mamadou Babaei
Packager	Tuesday	fe843ea6 bump the SenseGlove Unreal Engine Marketplace Pa...
Resources	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
Source	Sunday	33649f10 replace all bitfield uproperties with booleans Mama...
.clang-format	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
.editorconfig	Nov 2, 2022	ff02526c initial public release Mamadou Babaei
.gitattributes	Jul 25	e3f9be40 merge the pack branch into the plugin's source and ...

Installation

Once you have obtained the desired plugin version compatible with the Unreal Engine version you have in mind using any of the methods mentioned above, it's time to build and install the plugin. There are two ways to install the SenseGlove Unreal Engine Plugin, one is at the engine level, and the other is per project.

- **Engine-level installation**: this method makes the plugin accessible to any project within that Unreal Engine version.
- **Per-project installation**: this method makes the plugin accessible only to a specific project.

> ⚠ Warning
>
> Please note that it is best practice to install the plugin either at the project level or the engine level, but not both. Having the plugin installed in both locations, at the same time, can lead to various issues, especially if the version of the plugin installed at the engine level differs from the one installed at the plugin level. A guide on verifying the plugin version is also available as well.

Engine-level installation

Per-project installation

1. Locate your existing C++ or Blueprint project, or create a new project from scratch.

> Important
>
> Before proceeding, make sure your project's Unreal Editor is closed, and you do not have your project open in your C++ IDE to avoid any issues.

2. Inside your project's root directory create a new `Plugins` directory if you don't have one already.

3. Inside the `Plugins` directory create a new directory named `SenseGlove`.
4. Extract the content of your downloaded zip file into the `SenseGlove` directory.
5. Remove any directories or files that are only meant for use by the SenseGlove Unreal Engine Plugin maintainers. These are not part of the distributed plugin package and are not required by either Unreal Engine or the SenseGlove Unreal Engine Plugin to function correctly.

The mandatory files and folders to stay are as follows:

```
Config
Content
Resources
Source
SenseGlove.uplugin
```

Anything else can be safely removed. For example, these files and folders can be safely deleted:

```
Handbook
Packager
.clang-format
.editorconfig
.gitattributes
.gitignore
README.md
```

6. Ensure your project has the correct structure.

For a Blueprint-only project, it should look something like this:

```
MyBlueprintProject
│
├── Config
│
├── Content
│
├── Plugins
│    │
│    └── SenseGlove
│         │
│         ├── Config
│         │
│         ├── Content
│         │
│         ├── Resources
│         │
│         ├── SenseGlove.uplugin
│         │
│         └── Source
│
└── MyBlueprintProject.uproject
```

For a C++ project, the structure should look like this:

```
MyCppProject
   │
   ├── Config
   │
   ├── Content
   │
   ├── Plugins
   │      │
   │      └── SenseGlove
   │             │
   │             ├── Config
   │             │
   │             ├── Content
   │             │
   │             ├── Resources
   │             │
   │             ├── SenseGlove.uplugin
   │             │
   │             └── Source
   │
   ├── Source
   │
   └── MyCppProject.uproject
```

Tip

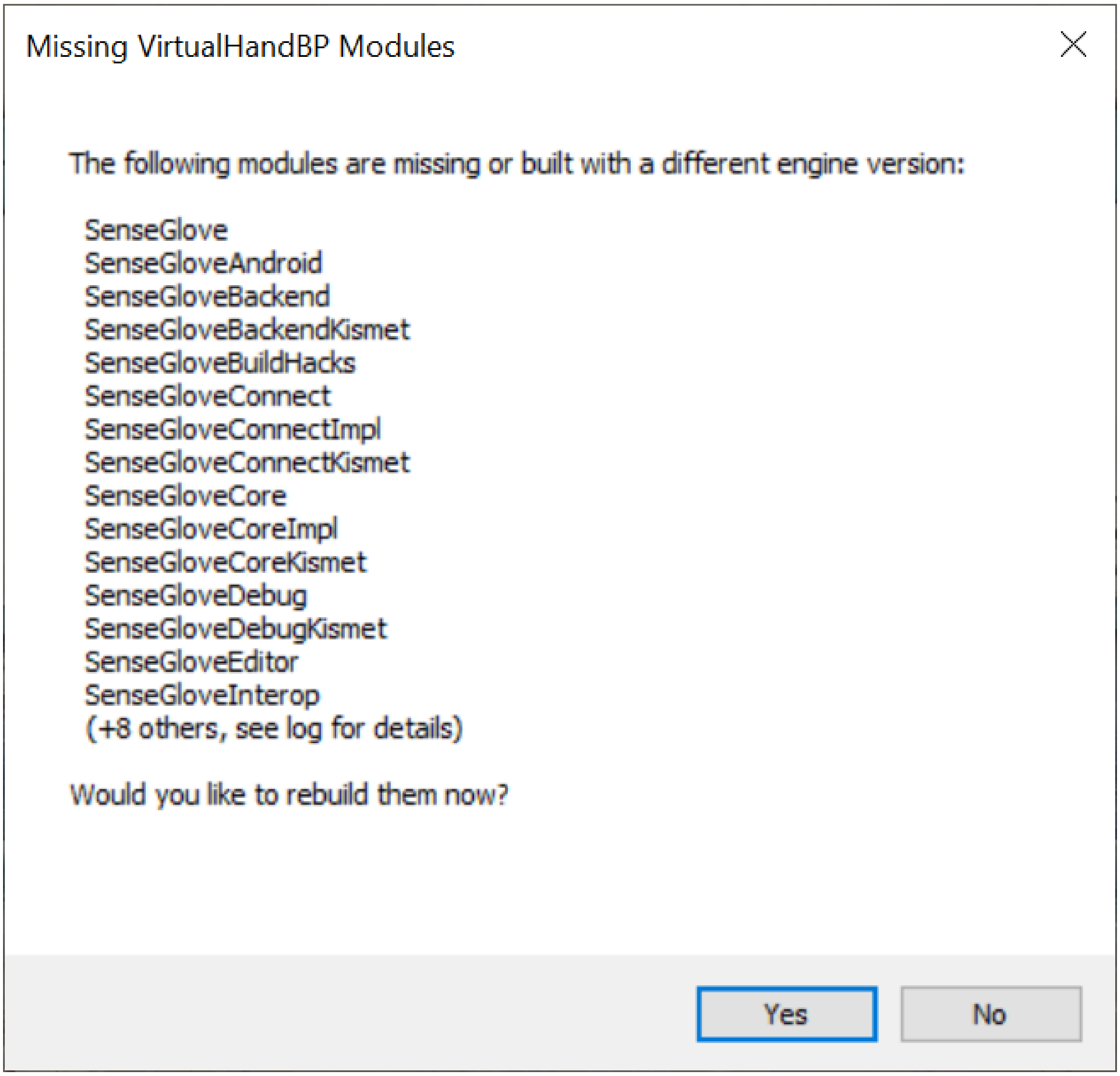
If you are keeping your project under Git and Git LFS, consider keeping the `.gitignore` and `.gitattributes` as they help keep irrelevant files out of the remote repository, or manage binary blobs efficiently.

7. OK, now it's time to build the plugin.

Note

For Linux build instructions see the Linux Build Instructions section.

For a Blueprint-only project, on Microsoft Windows simply double-clicking the project's `.uproject` file should present you with a pop-up informing you that some binary modules are missing.

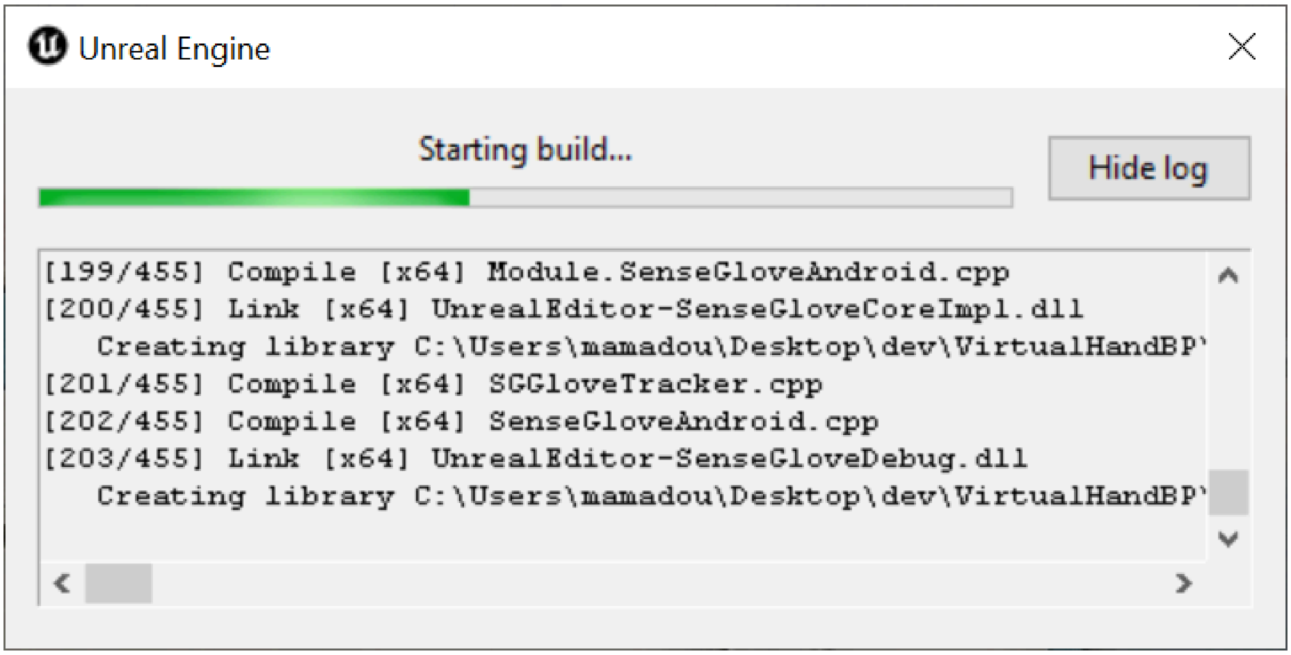

Missing VirtualHandBP Modules

The following modules are missing or built with a different engine version:

SenseGlove
SenseGloveAndroid
SenseGloveBackend
SenseGloveBackendKismet
SenseGloveBuildHacks
SenseGloveConnect
SenseGloveConnectImpl
SenseGloveConnectKismet
SenseGloveCore
SenseGloveCoreImpl
SenseGloveCoreKismet
SenseGloveDebug
SenseGloveDebugKismet
SenseGloveEditor
SenseGloveInterop
(+8 others, see log for details)

Would you like to rebuild them now?

After confirming, the build process will start automatically, and a dialog indicating the build progress will be shown:



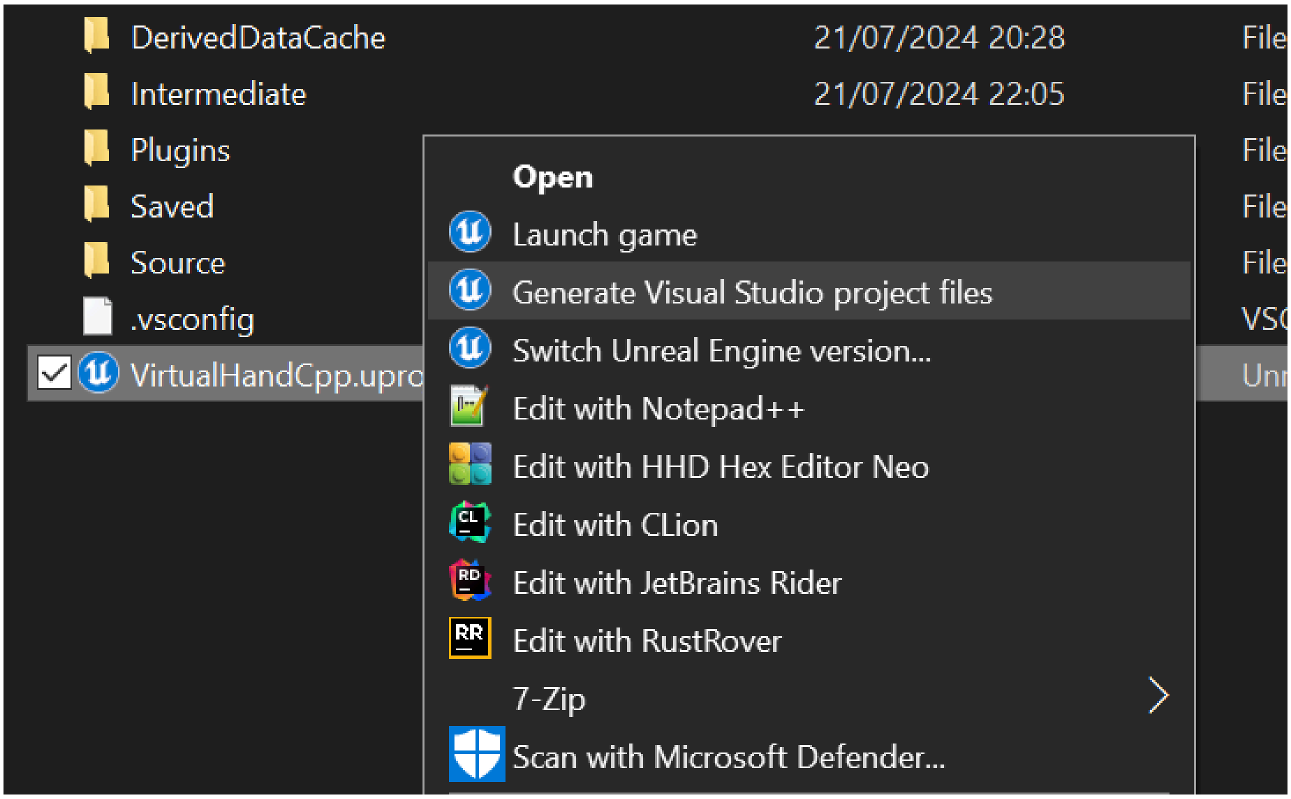
```
[199/455] Compile [x64] Module.SenseGloveAndroid.cpp
[200/455] Link [x64] UnrealEditor-SenseGloveCoreImpl.dll
   Creating library C:\Users\mamadou\Desktop\dev\VirtualHandBP
[201/455] Compile [x64] SGGloveTracker.cpp
[202/455] Compile [x64] SenseGloveAndroid.cpp
[203/455] Link [x64] UnrealEditor-SenseGloveDebug.dll
   Creating library C:\Users\mamadou\Desktop\dev\VirtualHandBP
```

Once finished successfully, the project will be loaded.

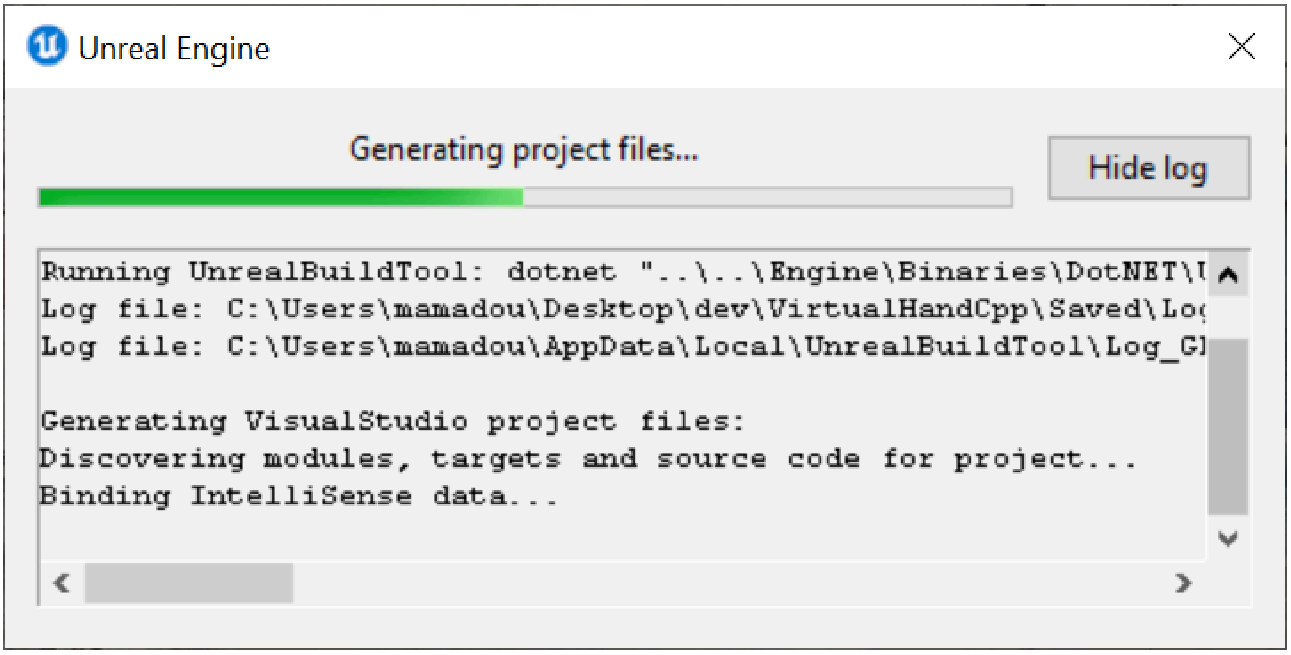
Note

Sometimes, due to an esoteric bug in some versions of Unreal Engine, the build process for Blueprint-only projects may immediately fail after choosing `Yes` in the `Missing Modules` dialog. If this happens, one workaround would be to try to build the plugin inside a temporary C++ project, then copy the `Plugins/SenseGlove` folder containing the binaries, from the C++ project to your Blueprint project and then try to reopen the project again.

For C++ projects, on Microsoft Windows, right-click on your C++ `.uproject` file and choose `Generate Visual Studio project files`:



A dialog will pop up shows you the progress of generating the Visual Studio project files:



```
Running UnrealBuildTool: dotnet "..\..\Engine\Binaries\DotNET\U
Log file: C:\Users\mamadou\Desktop\dev\VirtualHandCpp\Saved\Log
Log file: C:\Users\mamadou\AppData\Local\UnrealBuildTool\Log_Gl

Generating VisualStudio project files:
Discovering modules, targets and source code for project...
Binding IntelliSense data...
```

Once the project files are generated, open up the C++ project in your preferred C++ IDE and build the project. After this, the project can be loaded in the Unreal Editor.

8. Once the plugin has been built successfully, ensure the SenseGlove Unreal Engine is enabled and verify the plugin version matches the expected version.

Linux Build Instructions

When building the SenseGlove Unreal Engine Plugin on Linux, you won't encounter the `Missing Modules` dialog that appears on Microsoft Windows. Instead, examining the Unreal Editor logs reveals that the Unreal Editor automatically chooses `No` in response to the `Would you like to rebuild them now?` question as the `No is implied` states.

```
$ /path/to/UnrealEngine/Engine/Binaries/Linux/UnrealEditor \
    /path/to/MyBlueprintProject/MyBlueprintProject.uproject

LogLinux: Warning: MessageBox: The following modules are missing or built
with a different engine version:

    SenseGlove
    SenseGloveAndroid
    SenseGloveBackend
    SenseGloveBackendKismet
    SenseGloveBuildHacks
    SenseGloveConnect
    SenseGloveConnectImpl
    SenseGloveConnectKismet
    SenseGloveCore
    SenseGloveCoreImpl
    SenseGloveCoreKismet
    SenseGloveDebug
    SenseGloveDebugKismet
    SenseGloveEditor
    SenseGloveInterop
    (+8 others, see log for details)

Would you like to rebuild them now?: Missing MyBlueprintProject Modules: No
is implied.
LogCore: Engine exit requested (reason: EngineExit() was called)
LogExit: Preparing to exit.
LogPakFile: Destroying PakPlatformFile
LogExit: Exiting.
LogInit: Tearing down SDL.
Exiting abnormally (error code: 1)
```

If your Unreal Engine installation on Linux was obtained from the GitHub Sources y you can generate the project files using the following command:

```
$ /path/to/UnrealEngine/GenerateProjectFiles.sh \
    /path/to/MyProject/MyProject.uproject \
    -editor -game -makefile
```

However, if you are using a prebuilt Linux version of Unreal Engine, the main `GenerateProjectFiles.sh` script at the engine root does not exists. Instead, we have to invoke the underlying `GenerateProjectFiles.sh` script located elsewhere. This is a different script which shares the same name and is also present in the GitHub sources. The main `GenerateProjectFiles.sh` script at the engine root is actually a wrapper around this script.

```
$ /path/to/UnrealEngine/Engine/Build/BatchFiles/Linux/GenerateProjectFiles.sh
\
    /path/to/MyProject/MyProject.uproject \
    -editor -game -makefile
```

Still, running the any of the above commands on a Blueprint project results in the following error:

```
$
/path/to/UnrealEngine/Engine/Build/BatchFiles/Linux/GenerateProjectFiles.sh \
    /path/to/MyBlueprintProject/MyBlueprintProject.uproject \
    -editor -game -makefile

Setting up Unreal Engine project files...

Setting up bundled DotNet SDK
Log file: /home/mamadou/.config/Epic/UnrealBuildTool/Log_GPF.txt
Project file formats specified via the command line will be ignored when
generating
project files from the editor and other engine tools.

Consider setting your desired IDE from the editor preferences window, or
modify your
BuildConfiguration.xml file with:

<?xml version="1.0" encoding="utf-8" ?>
<Configuration xmlns="https://www.unrealengine.com/BuildConfiguration">
  <ProjectFileGenerator>
    <Format>Make</Format>
  </ProjectFileGenerator>
</Configuration>


Generating Make project files:
Discovering modules, targets and source code for project...
Total execution time: 0.35 seconds
Directory '/path/to/MyBlueprintProject/MyBlueprintProject' is missing
'Source' folder.
```

For a C++ project, however, the project files will generate without any issues:

```
$ /path/to/UnrealEngine/Engine/Build/BatchFiles/Linux/GenerateProjectFiles.sh
\
    /path/to/MyCppProject/MyCppProject.uproject \
    -editor -game -makefile

Setting up Unreal Engine project files...

Setting up bundled DotNet SDK
Log file: /home/mamadou/.config/Epic/UnrealBuildTool/Log_GPF.txt
Project file formats specified via the command line will be ignored when
generating
project files from the editor and other engine tools.

Consider setting your desired IDE from the editor preferences window, or
modify your
BuildConfiguration.xml file with:

<?xml version="1.0" encoding="utf-8" ?>
<Configuration xmlns="https://www.unrealengine.com/BuildConfiguration">
  <ProjectFileGenerator>
    <Format>Make</Format>
  </ProjectFileGenerator>
</Configuration>


Generating Make project files:
Discovering modules, targets and source code for project...
Generating data for project indexing... 100%

Generating QueryTargets data for editor...
Total execution time: 2.98 seconds
```

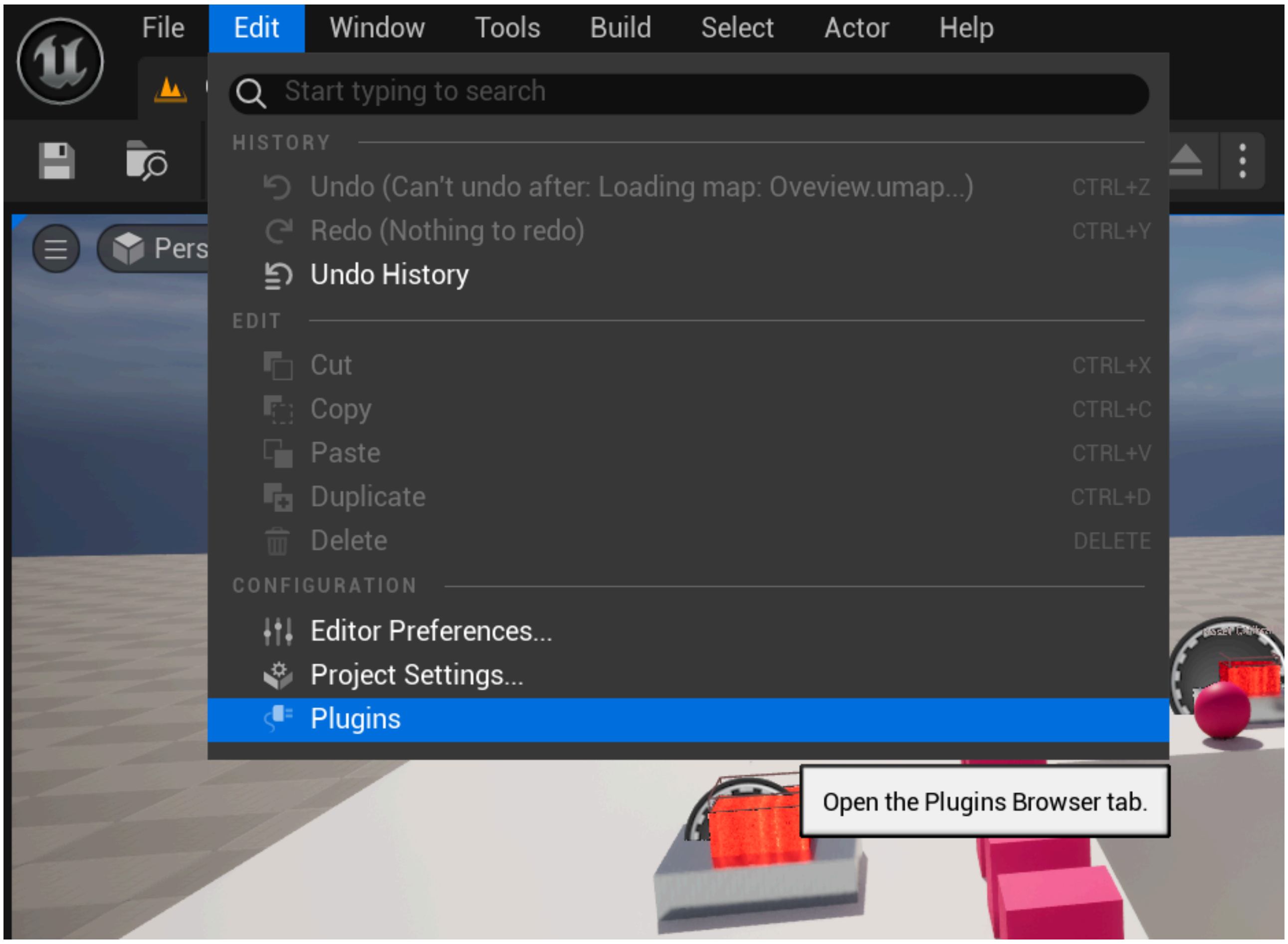
So, the workaround for Blueprint projects is to build the plugin inside a C++ project and then copy the `Plugin/SenseGlove` directory, which contains the built binary modules, to the corresponding directory in your Blueprint project.

```
$ /path/to/UnrealEngine/Engine/Build/BatchFiles/Linux/GenerateProjectFiles.sh
\
    /path/to/MyCppProject/MyCppProject.uproject \
    -editor -game -makefile
$ make MyCppProjectEditor -C /path/to/MyCppProject/
$ cp -vr \
    /path/to/MyCppProject/Plugins/SenseGlove \
    /path/to/MyBlueprintProject/Plugins/
$ /path/to/UnrealEngine/Engine/Binaries/Linux/UnrealEditor \
    /path/to/MyBlueprintProject/MyBlueprintProject.uproject
```

Enabling The SenseGlove Unreal Engine Plugin and Veirfying the Plugin Version

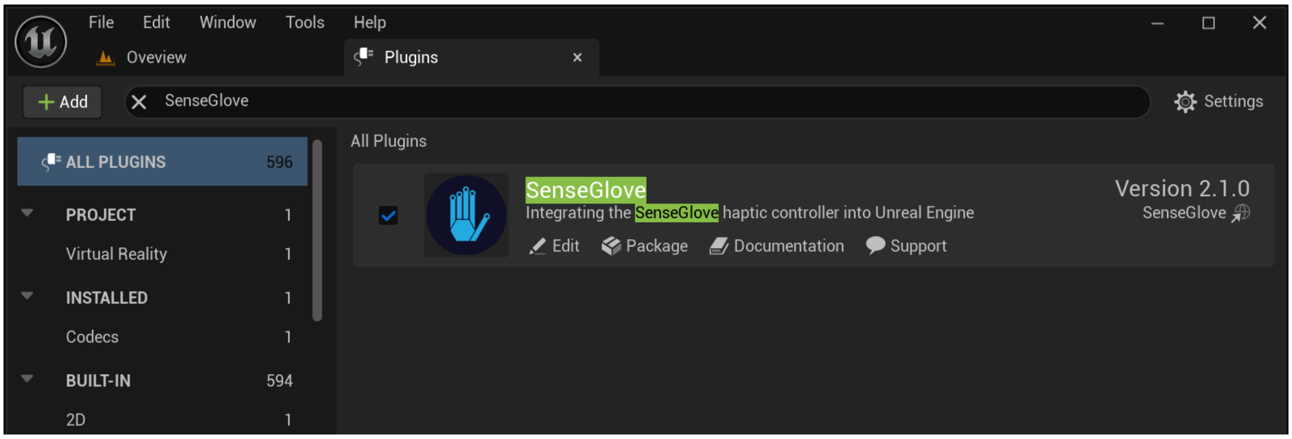
Enabling the SenseGlove Unreal Engine Plugin is a very simple and straightforward procedure. Furthermore, checking which version of the plugin your project is using may sometimes come in handy, especially if you have multiple versions of the plugin installed on different engine versions or various projects.

1. Inside the Unreal Editor for your project, select the `Plugins` from the `Edit` menu.



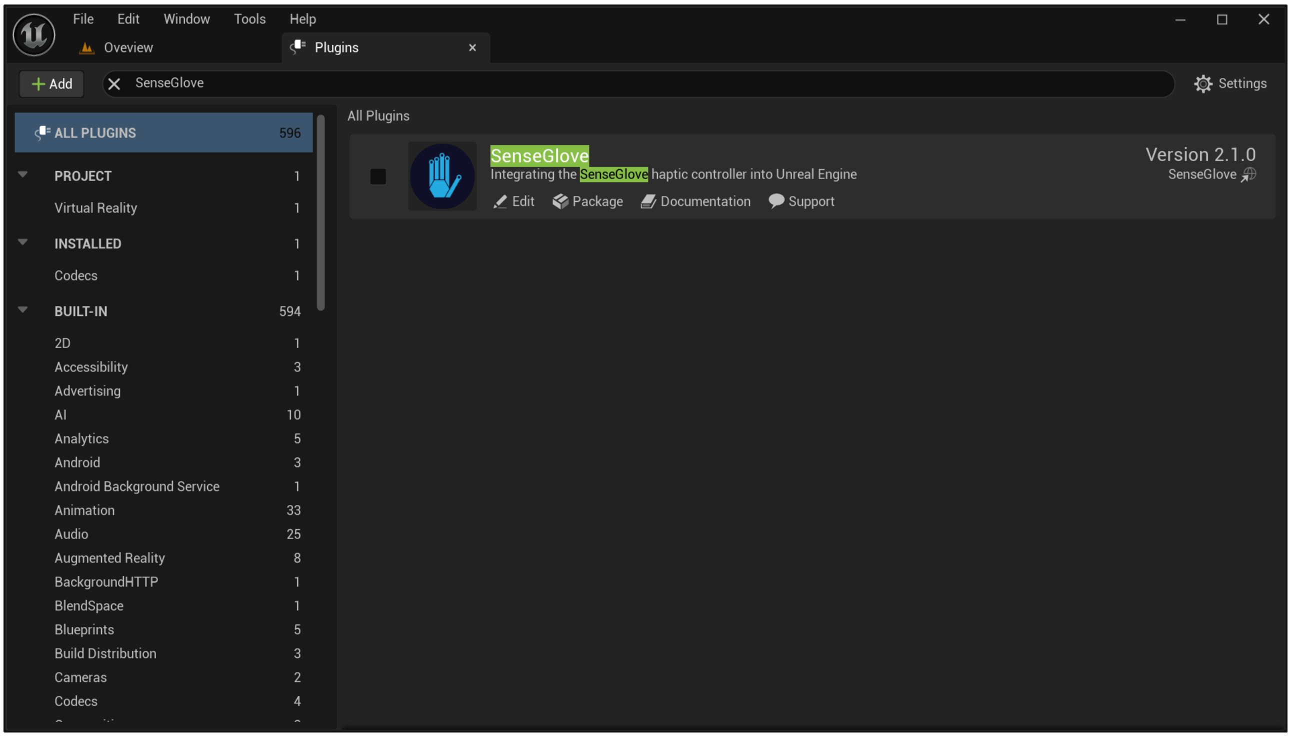
2. Once the plugin window/tab is open, start typing `SenseGlove` until you're able to spot the SenseGlove Unreal Engine Plugin. There you could find the plugin

version, and other useful resources, such as the documentation website or support contact.



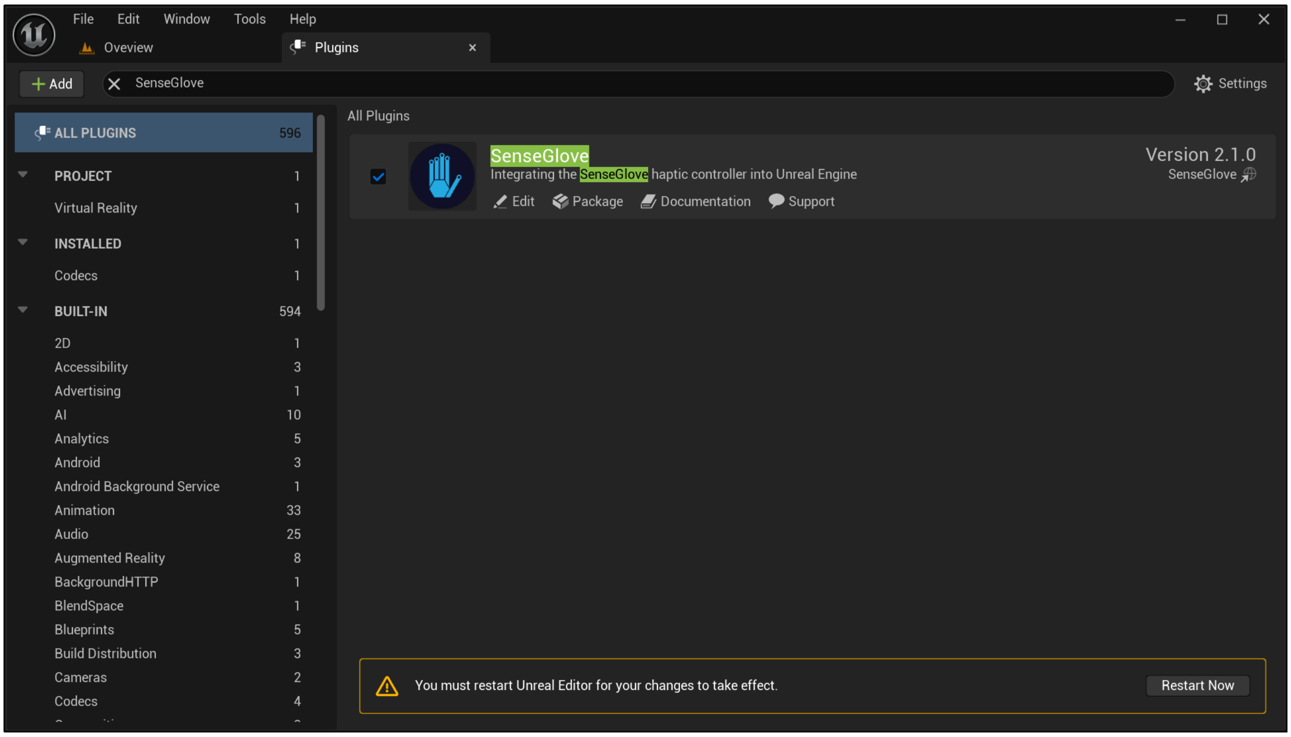
Integrating the SenseGlove haptic controller into Unreal Engine

3. If the plugin is not enabled, it does not have the checkmark next to it.



Integrating the SenseGlove haptic controller into Unreal Engine

4. It should be easy to click the checkmark and enable the plugin if that's not the case. Once the plugin is enabled, the Unreal Editor asks to be restarted. Click on the `Restart Now` button as this is mandatory to activate the plugin inside your project.



SenseGlove
Integrating the SenseGlove haptic controller into Unreal Engine
Version 2.1.0
SenseGlove

You must restart Unreal Editor for your changes to take effect.

5. The source code for the plugin might be required to be rebuilt depending on how you have obtained and installed the plugin, usually the Unreal Editor lets you know and does this automatically. If it's required to build the plugin source, and it fails to do so, it usually suggests an alternative approach such as opening your regenerating the project files and rebuilding the project inside a C++ IDE. Once this is done the Editor for your projects re-opens and you can follow steps `1` and `2` in order to verify the plugin's version and availability inside your project.

Video Tutorial

A video demonstrating the same instructions in more detail is also available on the SenseGlove YouTube channel.

SenseCom

SenseCom (short for SenseGlove Communications) is a background program that runs alongside your Unreal Engine application. Its primary function is to discover, and connect to SenseGlove devices on your system, exchanging data with them, much like a "SteamVR for Haptic Gloves." The SenseGlove Unreal Engine Plugin relies on SenseCom to communicate with any SenseGlove hardware.

Communication between your application and the physical gloves are possible via either Bluetooth Low Energy (a.k.a. Bluetooth LE, colloquially BLE, formerly marketed as Bluetooth Smart), or Bluetooth Serial (a.k.a. BT Serial) depending on the type and model of your glove, or the firmware version.

> **Important**
>
> Some glove models support firmware upgrades from a Bluetooth Serial firmware to a BLE-compatible firmware version. For more information, refer to the relevant documentation here, as this topic is beyond the scope of this handbook.

> **Note**
>
> SenseCom is required only for communication on Windows or Linux. For standalone Android devices, the communication functionality is embedded directly into your application.

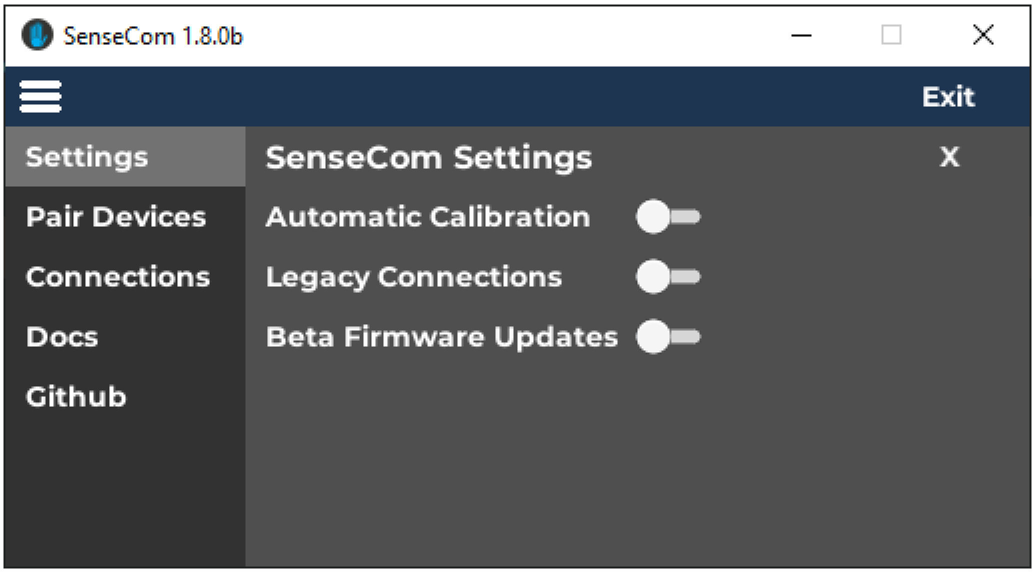
> **Note**
>
> For more detailed information and troubleshooting, consult the SenseCom documentation page on SGDocs, please.

SenseCom (Bluetooth Low Energy)

Up to SenseCom `v1.7.x`, the only supported Bluetooth protocol for communication was Bluetooth Serial. However, starting with the `v1.8.x` series, SenseCom introduced support for Bluetooth Low Energy, which is now the preferred method of communication.

Important

If SenseCom fails to recognize your gloves with Bluetooth Low Energy firmware, it may be because the `Legacy Connections` option is enabled. In that case SenseCom is only able to discover gloves with a Bluetooth Serial firmware. Enabling this option should allow SenseCom to discover and connect to your glove.



Automatic Calibration
Legacy Connections
Beta Firmware Updates

Important

Some glove models support firmware upgrades from a Bluetooth Serial firmware to a BLE-compatible firmware version. For more information, refer to the relevant documentation here, as this topic is beyond the scope of this handbook.

SenseCom on Android (Bluetooth Low Energy)

Unlike PCVR-mode on Windows or Linux, there's no separate SenseCom application available for Standalone-mode on Android; instead, the communication functionality is integrated into your application.

As a result, in Standalone-mode, unlike PCVR-mode where BLE gloves do not require pairing at the operating system level and connections are managed by SenseCom, you need to pair your desired gloves through your operating system's Bluetooth settings before launching any applications that rely on the SenseGlove Unreal Engine Plugin. These instructions vary depending on the vendor and model of your Head-Mounted Display device. Please refer to the official documentation for detailed instructions:

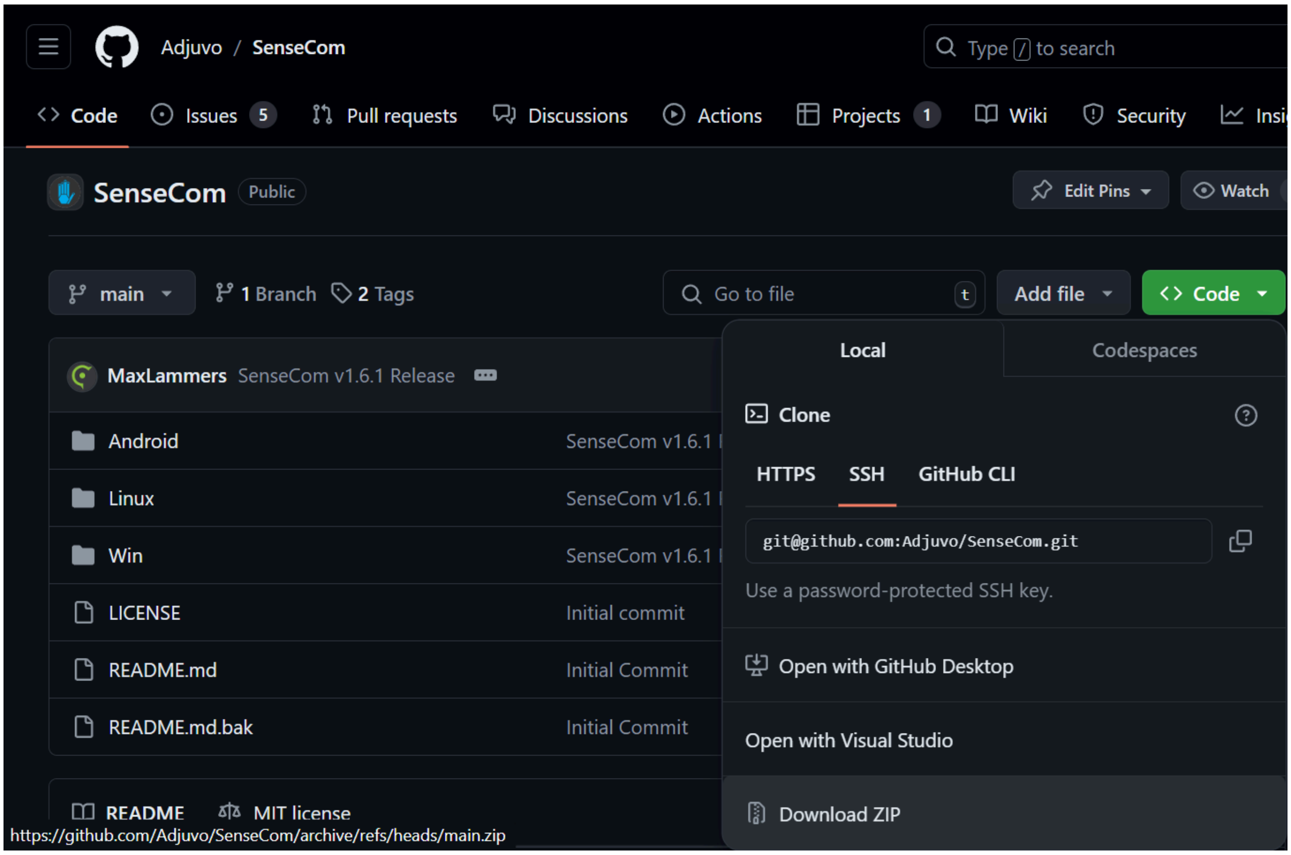
- Meta Quest: Connect a compatible Bluetooth device to Meta Quest headsets
- HTC VIVE: Pairing Bluetooth devices

If you are using a different kind of HMD, ensure you consult the vendor-specific instructions to properly pair your gloves with your HMD of choice in Standalone-mode.

SenseCom on GNU/Linux (Bluetooth Low Energy)

Follow these steps to quickly set up and run SenseCom on GNU/Linux:

1. First, obtain the SenseCom binaries from its GitHub repository.



```
git@github.com:Adjuvo/SenseCom.git
```

https://github.com/Adjuvo/SenseCom/archive/refs/heads/main.zip

2. Extract the SenseCom `.zip` file to a location on your computer.

```
$ unzip SenseCom-main.zip -d /some/path/
```

3. Navigate to the `SenseCom_Linux_Latest` folder containing the SenseCom binaries for GNU/Linux:

```
$ cd /some/path/SenseCom-main/Linux/SenseCom_Linux_Latest/
```

4. List the files and check the executable permissions for the main SenseCom binary, `SenseCom.x86_64`:

```
$ ls -ahl

total 20M
drwxr-xr-x 3 mamadou mamadou   5 Apr 10 11:24 .
drwxr-xr-x 3 mamadou mamadou   5 Apr 10 11:24 ..
drwxr-xr-x 7 mamadou mamadou  34 Apr 10 11:24 SenseCom_Data
-rw-r--r-- 1 mamadou mamadou 15K Apr 10 11:24 SenseCom.x86_64
-rw-r--r-- 1 mamadou mamadou 33M Apr 10 11:24 UnityPlayer.so
```

5. As seen above the `SenseCom.x86_64` binary does not have the executable permission. Run the following command to set the executable permission for all users:

```
$ chmod a+x SenseCom.x86_64
```

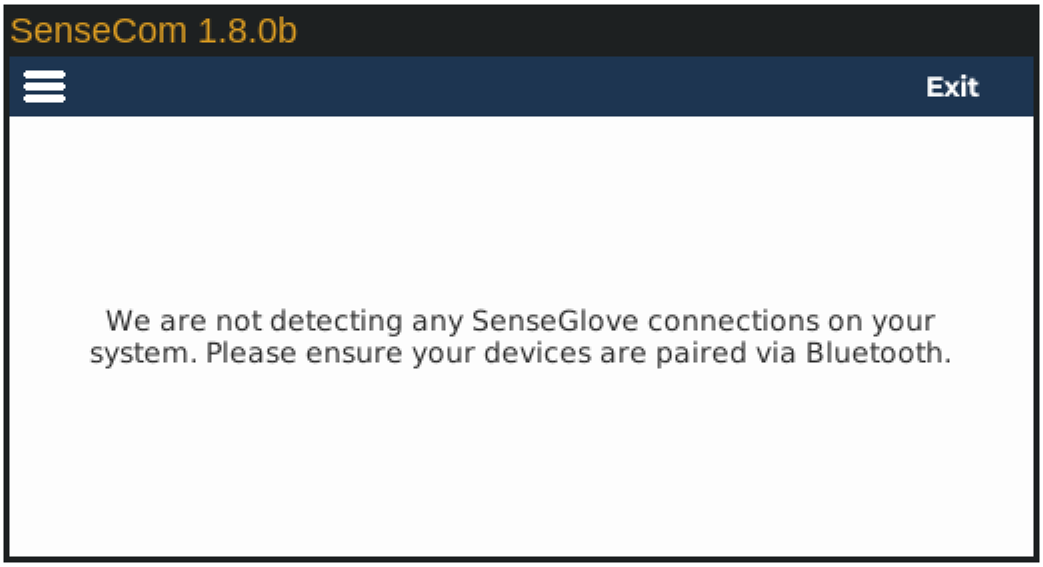
6. Veirfy the executable permission has been set on `SenseCom.x86_64`:

```
$ ls -l SenseCom.x86_64

-rwxr-xr-x 1 mamadou mamadou 14720 Apr 10 11:24 SenseCom.x86_64
```

7. Make sure the glove is powered on.

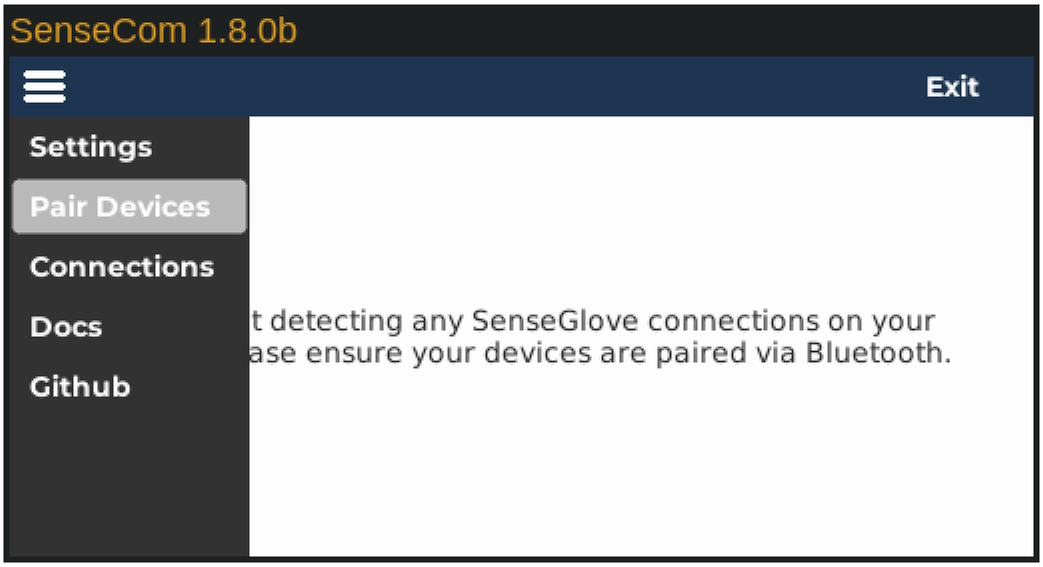
8. Time to run the SenseCom executable:

```
$ ./SenseCom.x86_64

[UnityMemory] Configuration Parameters - Can be set up in boot.config
    "memorysetup-bucket-allocator-granularity=16"
    "memorysetup-bucket-allocator-bucket-count=8"
    "memorysetup-bucket-allocator-block-size=4194304"
    "memorysetup-bucket-allocator-block-count=1"
    "memorysetup-main-allocator-block-size=16777216"
    "memorysetup-thread-allocator-block-size=16777216"
    "memorysetup-gfx-main-allocator-block-size=16777216"
    "memorysetup-gfx-thread-allocator-block-size=16777216"
    "memorysetup-cache-allocator-block-size=4194304"
    "memorysetup-typetree-allocator-block-size=2097152"
    "memorysetup-profiler-bucket-allocator-granularity=16"
    "memorysetup-profiler-bucket-allocator-bucket-count=8"
    "memorysetup-profiler-bucket-allocator-block-size=4194304"
    "memorysetup-profiler-bucket-allocator-block-count=1"
    "memorysetup-profiler-allocator-block-size=16777216"
    "memorysetup-profiler-editor-allocator-block-size=1048576"
    "memorysetup-temp-allocator-size-main=4194304"
    "memorysetup-job-temp-allocator-block-size=2097152"
    "memorysetup-job-temp-allocator-block-size-background=1048576"
    "memorysetup-job-temp-allocator-reduction-small-platforms=262144"
    "memorysetup-temp-allocator-size-background-worker=32768"
    "memorysetup-temp-allocator-size-job-worker=262144"
    "memorysetup-temp-allocator-size-preload-manager=262144"
    "memorysetup-temp-allocator-size-nav-mesh-worker=65536"
    "memorysetup-temp-allocator-size-audio-worker=65536"
    "memorysetup-temp-allocator-size-cloud-worker=32768"
    "memorysetup-temp-allocator-size-gfx=262144"
Loading in SingleInstance mode
```



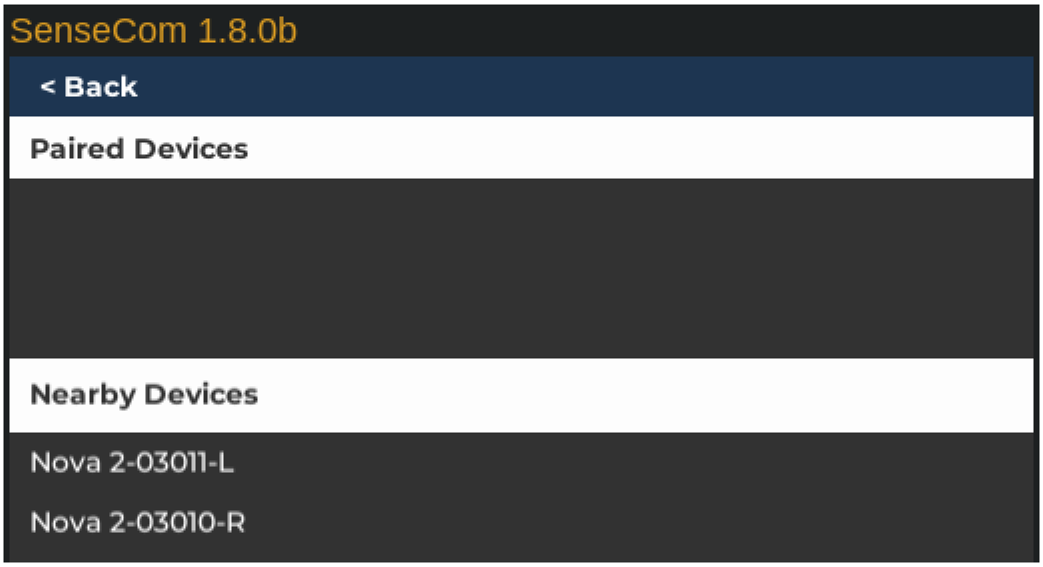
We are not detecting any SenseGlove connections on your system. Please ensure your devices are paired via Bluetooth.

9. After running SenseCom, it will not automatically connect to your gloves unless you have already paired them. To pair your devices, navigate to the hamburger menu and select `Pair Devices` .



t detecting any SenseGlove connections on your
ase ensure your devices are paired via Bluetooth.

10. Once inside the `Pair Devices` section, in case your gloves are already turned on, you should be able to spot them inside the `Nearby Devices` list.


SenseCom 1.8.0b

< Back

Paired Devices

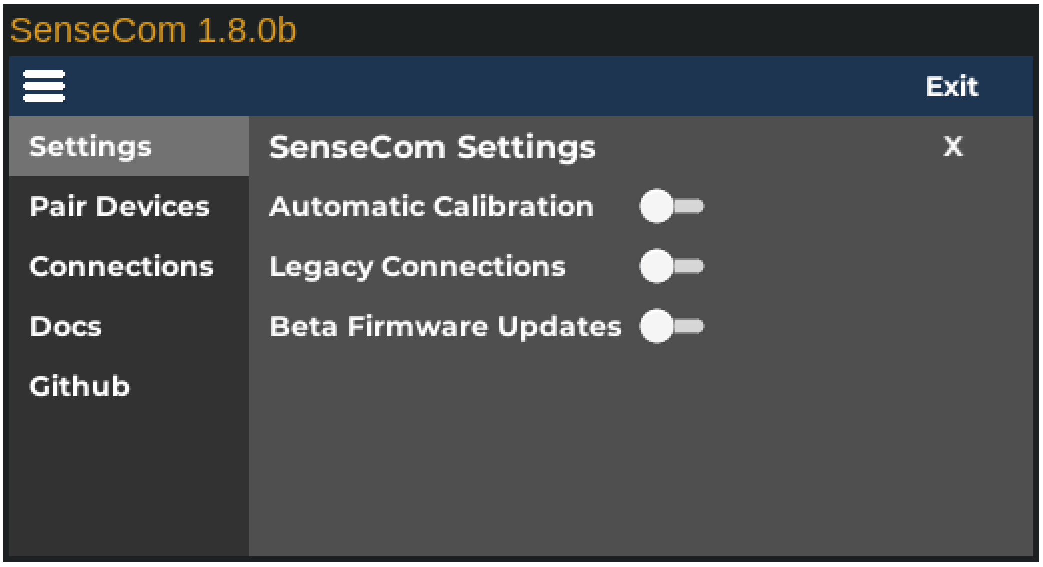
Nearby Devices

Nova 2-03011-L

Nova 2-03010-R

Important

If SenseCom fails to recognize your gloves with Bluetooth Low Energy firmware, it may be because the `Legacy Connections` option is enabled. In that case SenseCom is only able to discover gloves with a Bluetooth Serial firmware. Enabling this option should allow SenseCom to discover and connect to your glove.


SenseCom 1.8.0b

Exit

Settings

Pair Devices

Connections

Docs

Github

SenseCom Settings

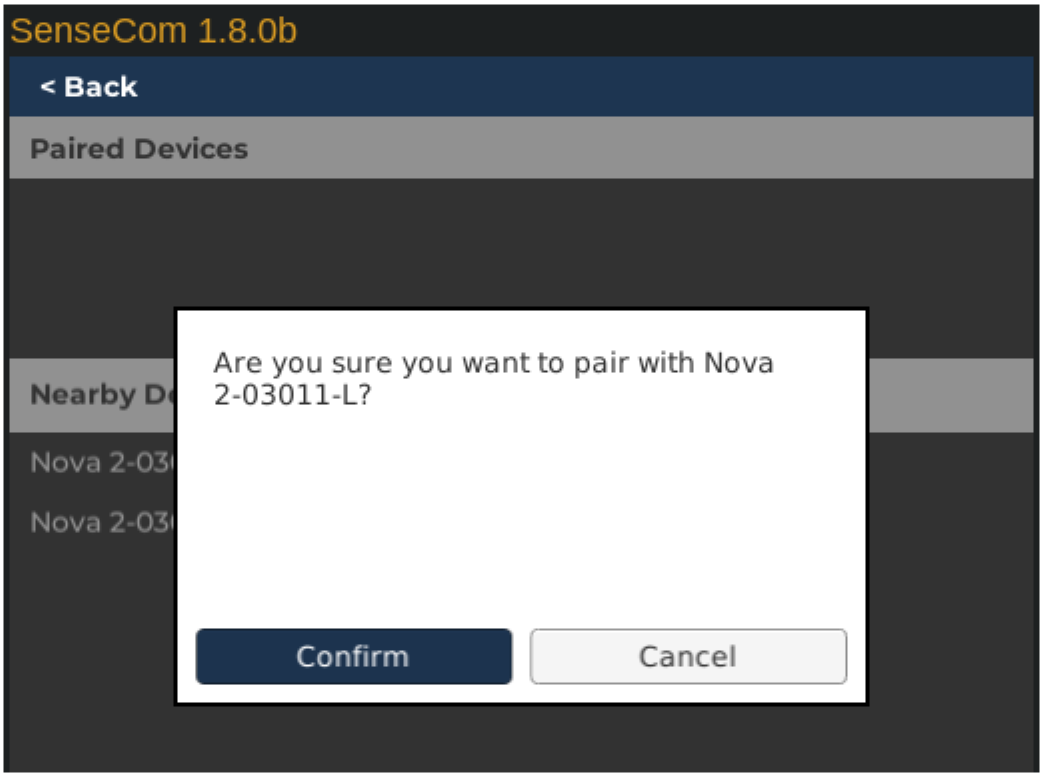
X

Automatic Calibration

Legacy Connections

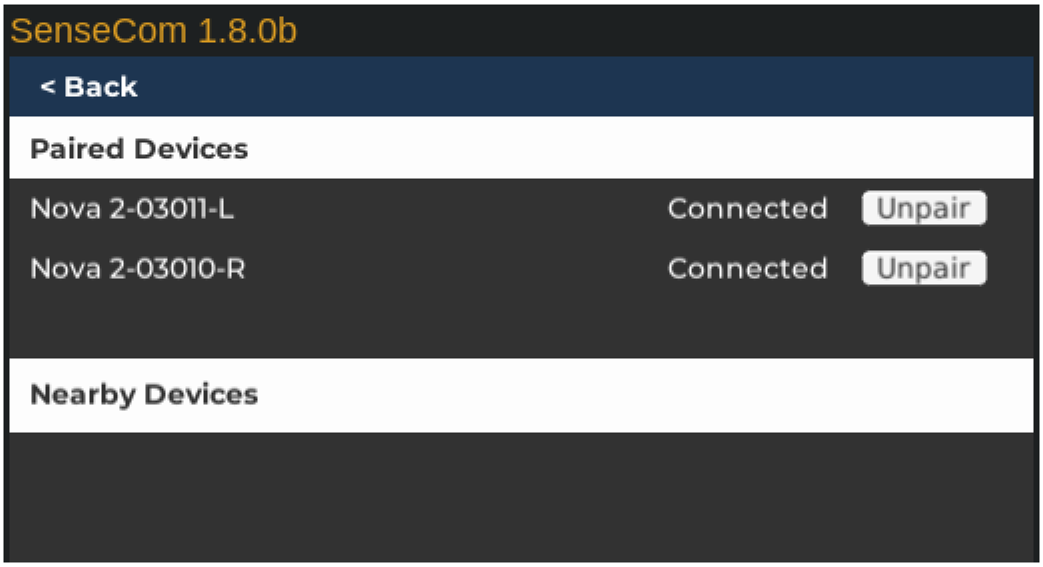
Beta Firmware Updates

11. Clicking on any glove within the `Nearby Devices` list will prompt a pairing confirmation. If this is the desired glove you wish to pair, proceed by clicking the `Confirm` button.



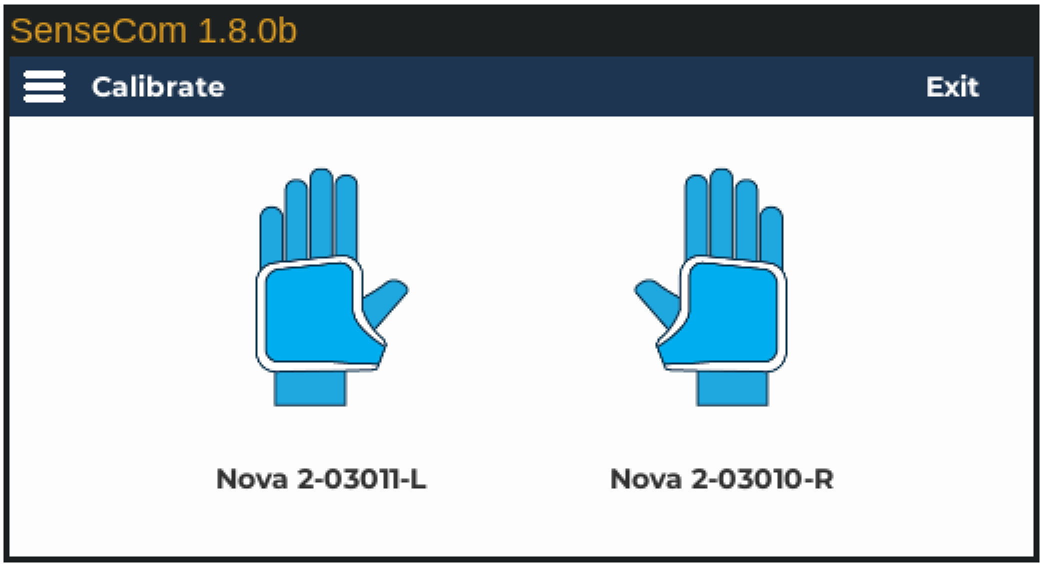
Are you sure you want to pair with Nova 2-03011-L?

12. After pairing all gloves, you can return to the main SenseCom window by pressing the `< Back` button. If needed, you can always revisit the `Paired Devices` list to unpair any gloves.



Nova 2-03011-L Connected
Nova 2-03010-R Connected

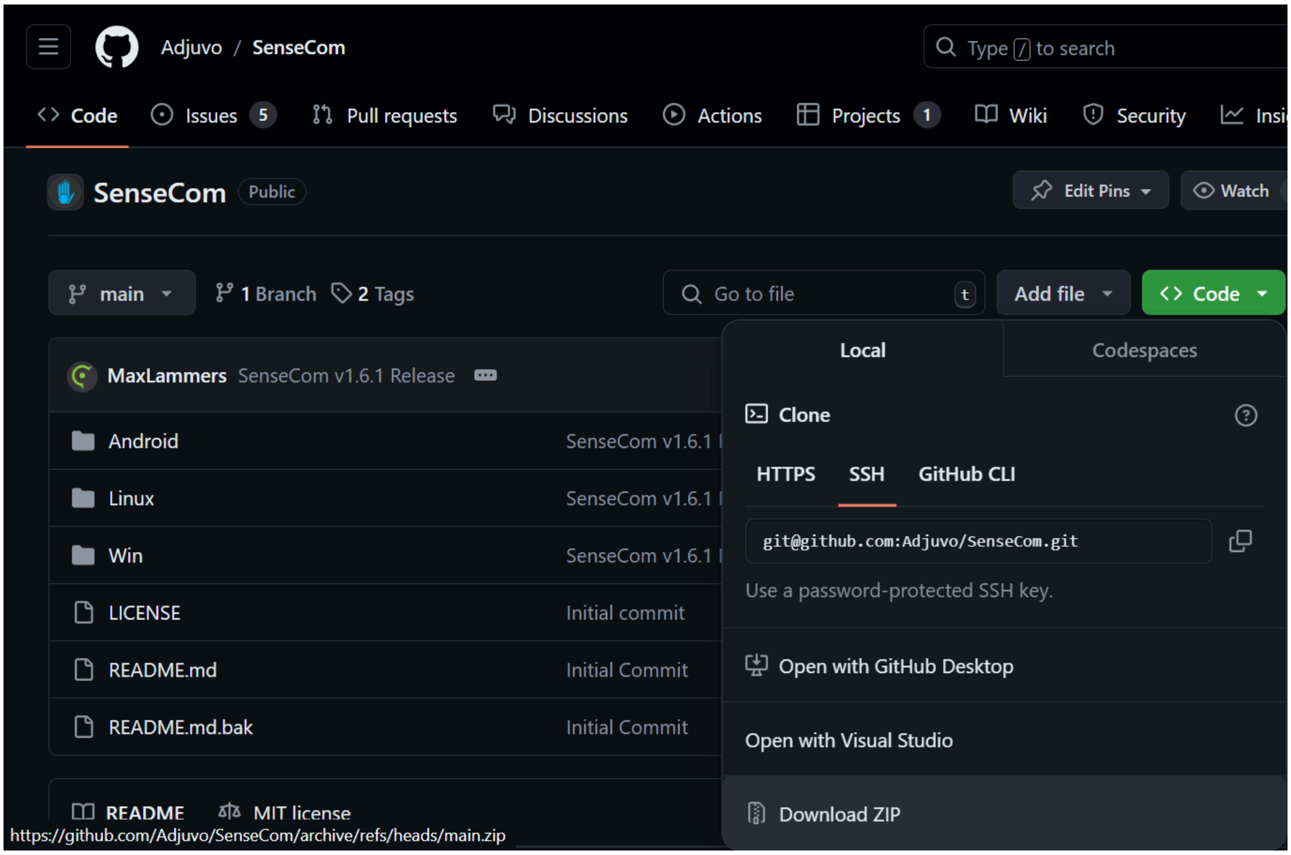
13. If you have followed all the steps correctly, upon returning to the main SenseCom window, you should see that your gloves are connected


SenseCom 1.8.0b
Calibrate
Exit
Nova 2-03011-L
Nova 2-03010-R

SenseCom on Microsoft Windows (Bluetooth Low Energy)

Follow these steps to quickly set up and run SenseCom on Microsoft Windows:

1. First, obtain the SenseCom binaries from its GitHub repository.



MaxLammers SenseCom v1.6.1 Release

Android	SenseCom v1.6.1
Linux	SenseCom v1.6.1
Win	SenseCom v1.6.1
LICENSE	Initial commit
README.md	Initial Commit
README.md.bak	Initial Commit

```
git@github.com:Adjuvo/SenseCom.git
```

Use a password-protected SSH key.

https://github.com/Adjuvo/SenseCom/archive/refs/heads/main.zip

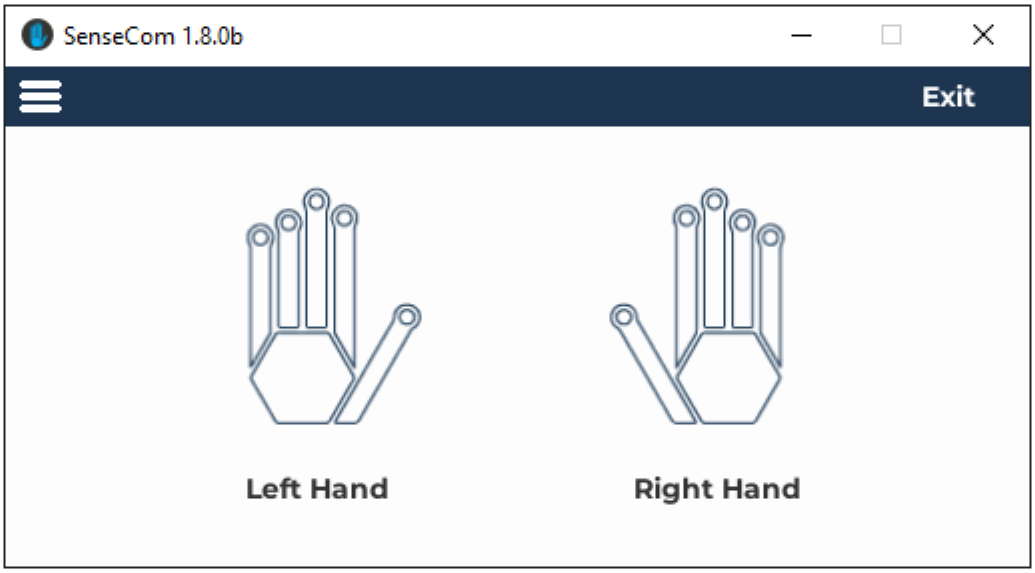
2. Extract the SenseCom `.zip` file to a location on your computer after downloading it.

3. Ensure any glove you would like to pair with and connect to your system is not paired, or connected to any other device, such as another PC or VR headset.

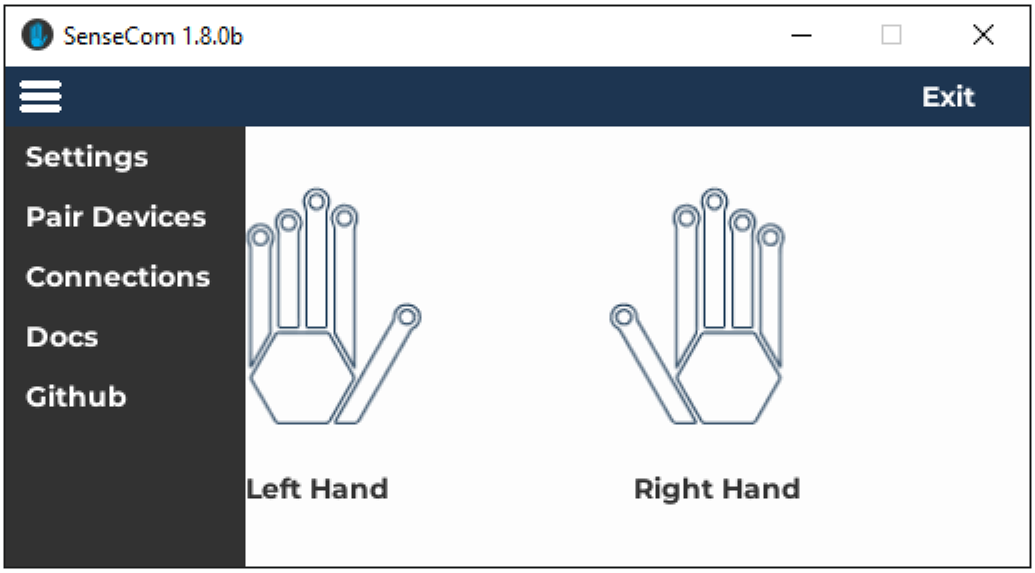
4. Make sure the glove is powered on.

5. Now, it's time to run SenseCom. Navigate to the folder where you extracted SenseCom and go to

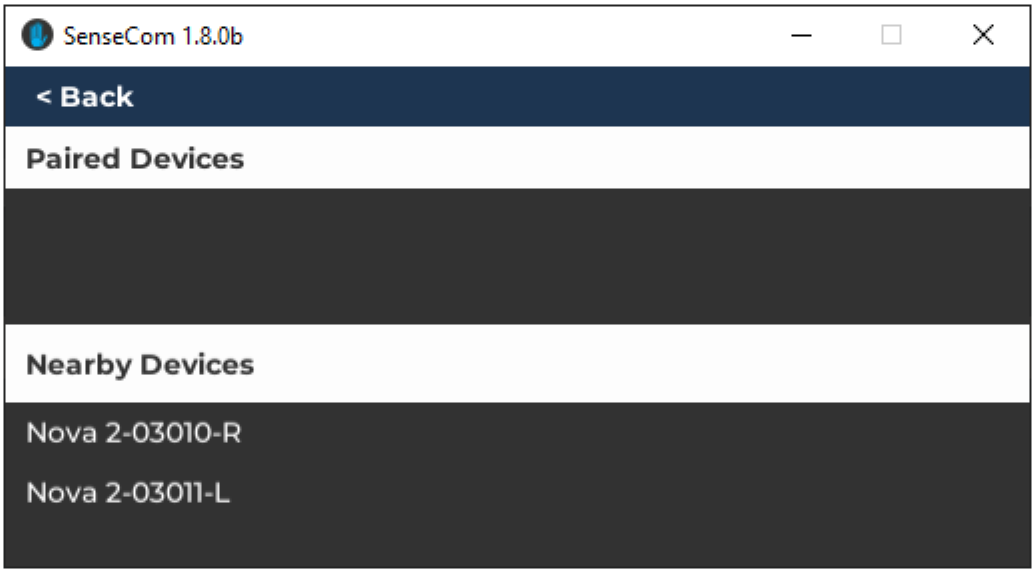
`/path/to/extracted/SenseCom/directory/Win/SenseCom_Win_Latest`, and then run the executable `SenseCom.exe`.



6. After running SenseCom, it will not automatically connect to your gloves unless you have already paired them. To pair your devices, navigate to the hamburger menu and select `Pair Devices`.



7. Once inside the `Pair Devices` section, in case your gloves are already turned on, you should be able to spot them inside the `Nearby Devices` list.



Paired Devices

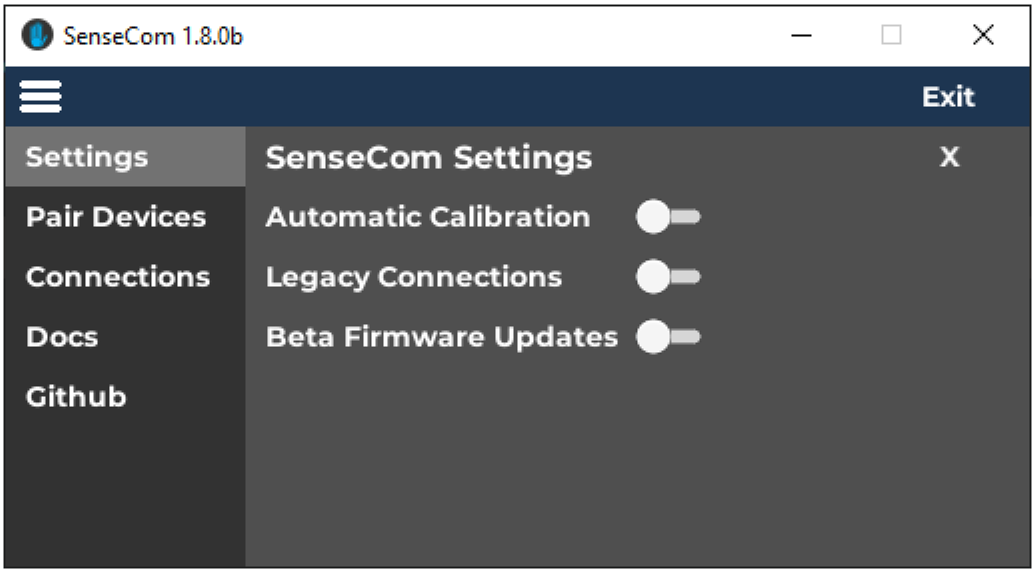
Nearby Devices

Nova 2-03010-R

Nova 2-03011-L

Important

If SenseCom fails to recognize your gloves with Bluetooth Low Energy firmware, it may be because the `Legacy Connections` option is enabled. In that case SenseCom is only able to discover gloves with a Bluetooth Serial firmware. Enabling this option should allow SenseCom to discover and connect to your glove.



SenseCom Settings

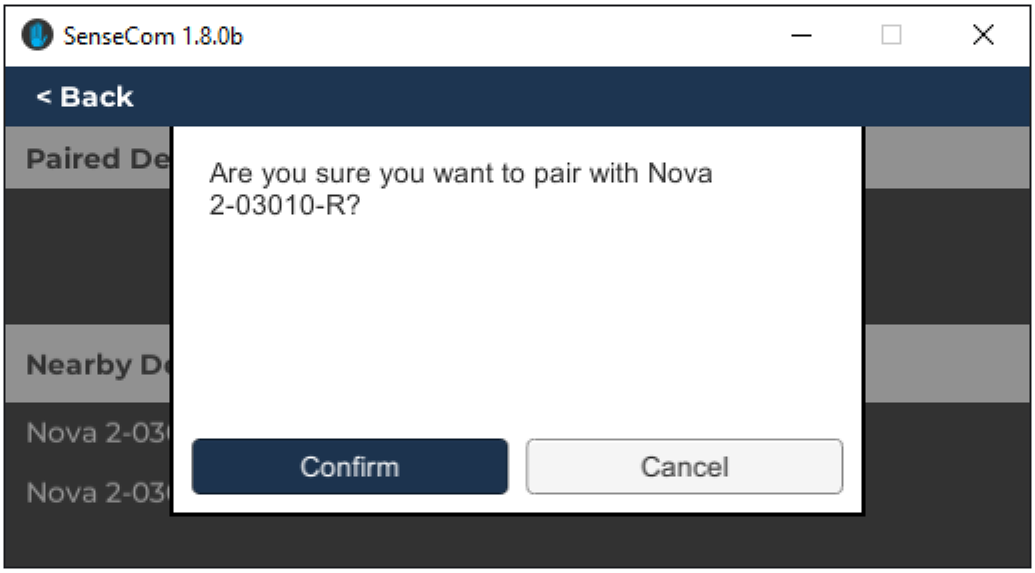
Automatic Calibration

Legacy Connections

Beta Firmware Updates

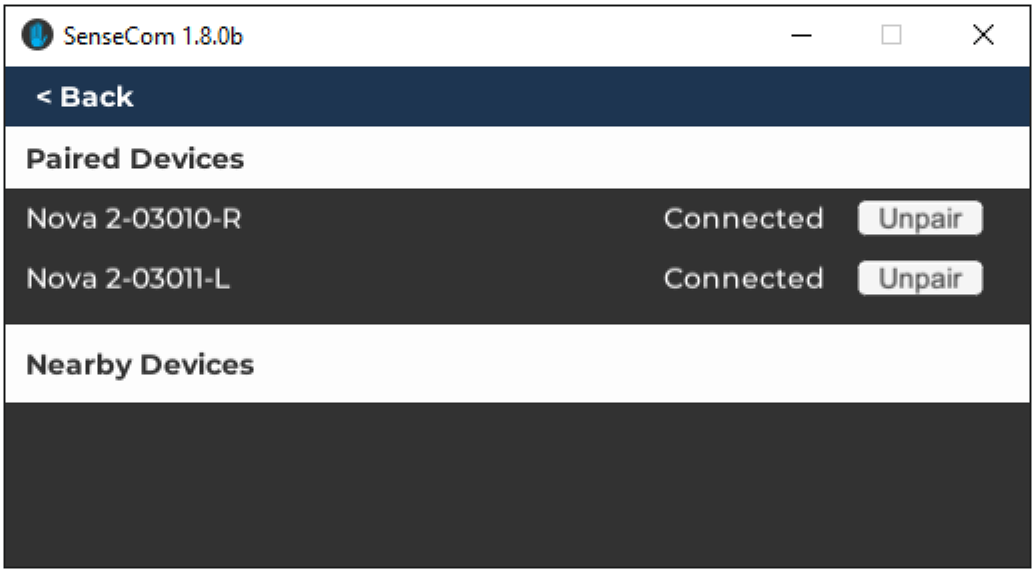
8. Clicking on any glove within the `Nearby Devices` list will prompt a pairing confirmation. If this is the desired glove you wish to pair, proceed by clicking the

`Confirm` button.



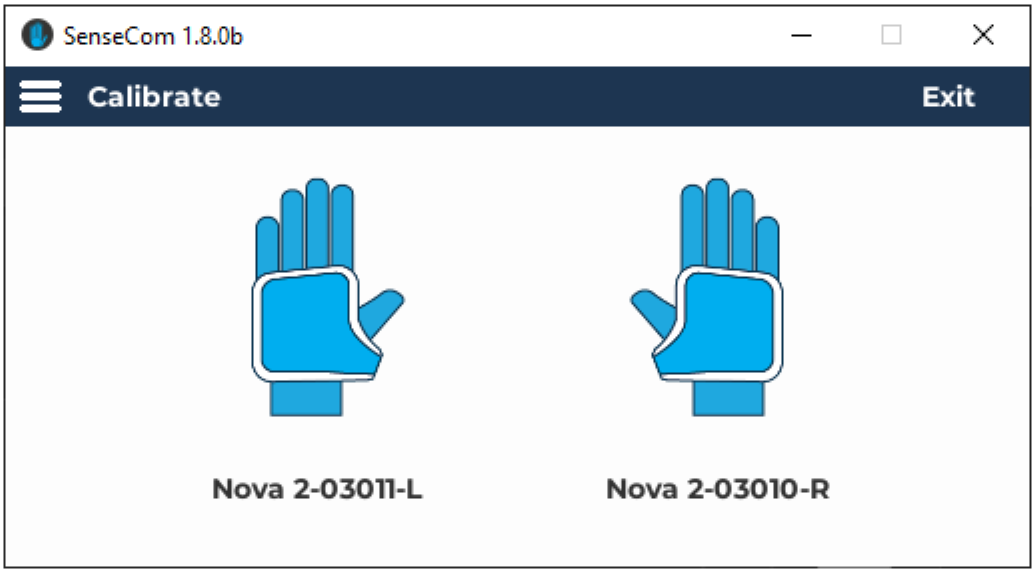
Are you sure you want to pair with Nova 2-03010-R?

9. After pairing all gloves, you can return to the main SenseCom window by pressing the `< Back` button. If needed, you can always revisit the `Paired Devices` list to unpair any gloves.



Nova 2-03010-R Connected
Nova 2-03011-L Connected

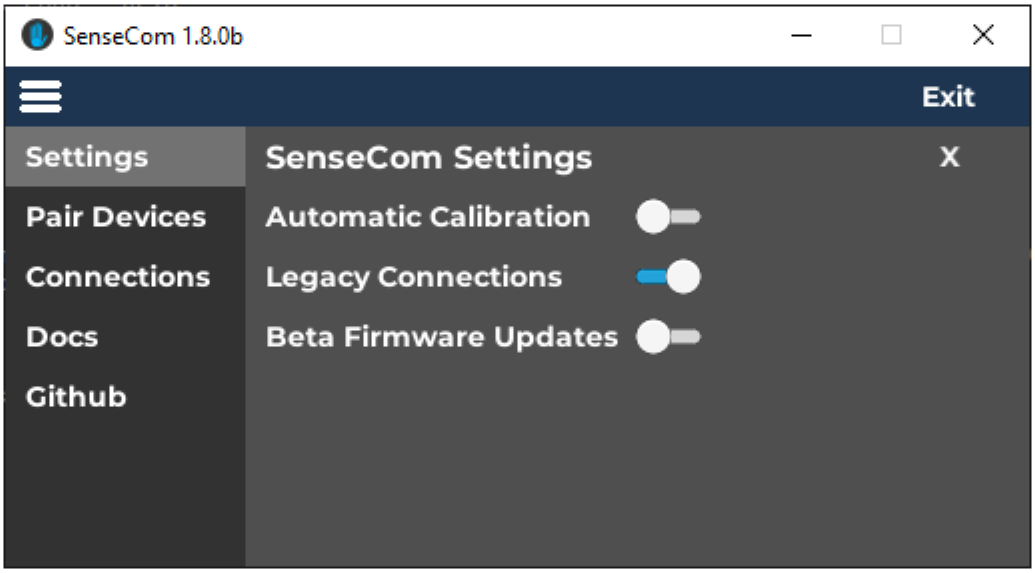
10. If you have followed all the steps correctly, upon returning to the main SenseCom window, you should see that your gloves are connected


SenseCom 1.8.0b
Calibrate
Exit
Nova 2-03011-L
Nova 2-03010-R

SenseCom (Bluetooth Serial)

Up to SenseCom `v1.7.x`, the only supported Bluetooth protocol for communication was Bluetooth Serial. However, starting with the `v1.8.x` series, SenseCom introduced support for Bluetooth Low Energy, which is now the preferred method of communication.

> **Important**
>
> If you are using SenseCom `v1.8.x+` and it fails to recognize your gloves with a Bluetooth Serial firmware, it may be because the `Legacy Connections` option is disabled, which is the default. Enabling this option should allow SenseCom to discover and connect to your glove.



Automatic Calibration [off]
Legacy Connections [✓ on]
Beta Firmware Updates [off]

> **Important**
>
> Some glove models support firmware upgrades from a Bluetooth Serial firmware to a BLE-compatible firmware version. For more information, refer to the relevant documentation here, as this topic is beyond the scope of this handbook.

SenseCom on Android (Bluetooth Serial)

Unlike PCVR-mode on Windows or Linux, there's no separate SenseCom application available for Standalone-mode on Android; instead, the communication functionality is integrated into your application.

As a result, you need to pair your desired gloves through your operating system's Bluetooth settings before launching any applications that rely on the SenseGlove Unreal Engine Plugin. These instructions vary depending on the vendor and model of your Head-Mounted Display device. Please refer to the official documentation for detailed instructions:

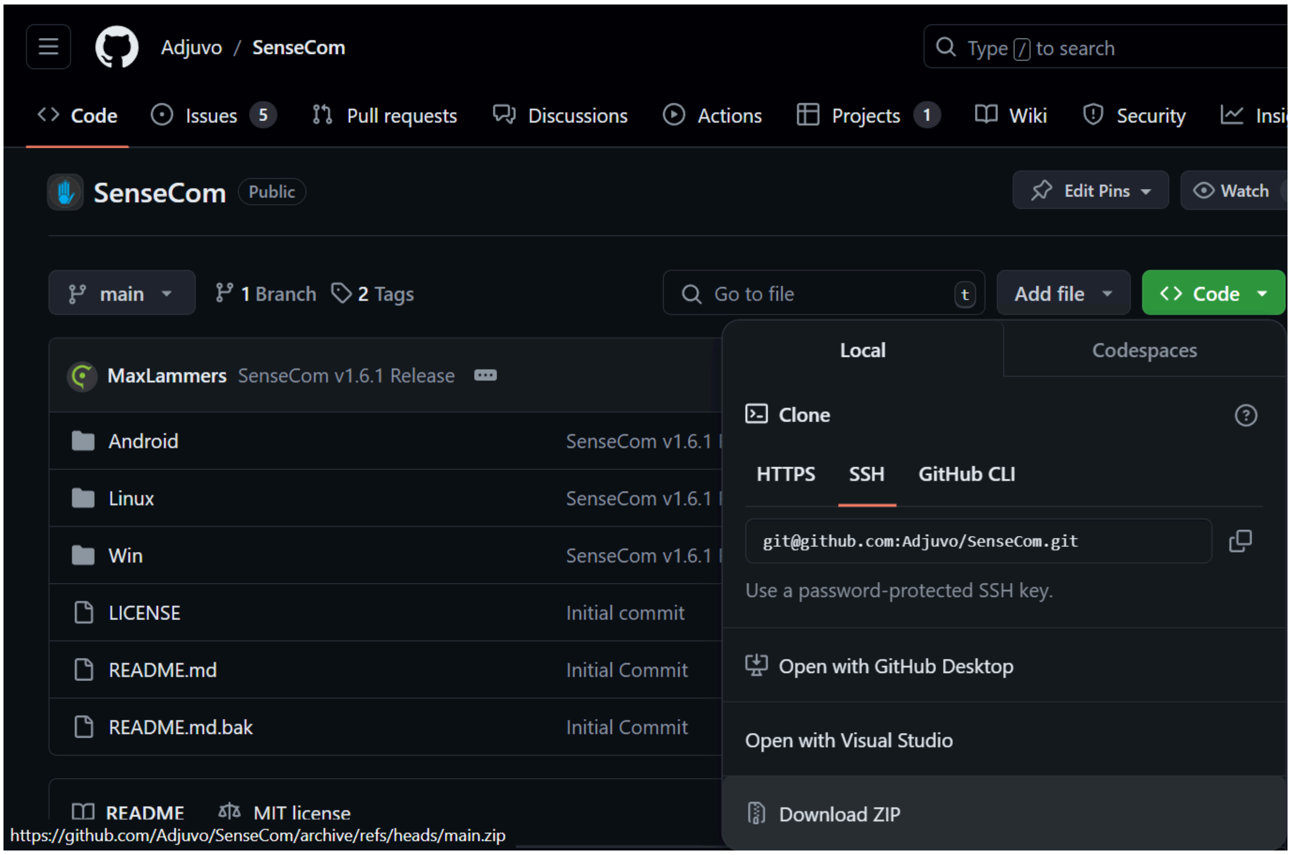
- Meta Quest: Connect a compatible Bluetooth device to Meta Quest headsets
- HTC VIVE: Pairing Bluetooth devices

If you are using a different kind of HMD, ensure you consult the vendor-specific instructions to properly pair your gloves with your HMD of choice in Standalone-mode.

SenseCom on GNU/Linux (Bluetooth Serial)

Follow these steps to quickly set up and run SenseCom on GNU/Linux:

1. First, obtain the SenseCom binaries from its GitHub repository.



MaxLammers SenseCom v1.6.1 Release

Android	SenseCom v1.6.1
Linux	SenseCom v1.6.1
Win	SenseCom v1.6.1
LICENSE	Initial commit
README.md	Initial Commit
README.md.bak	Initial Commit

```
git@github.com:Adjuvo/SenseCom.git
```

Use a password-protected SSH key.

https://github.com/Adjuvo/SenseCom/archive/refs/heads/main.zip

2. Extract the SenseCom `.zip` file to a location on your computer.

```
$ unzip SenseCom-main.zip -d /some/path/
```

3. Navigate to the `SenseCom_Linux_Latest` folder containing the SenseCom binaries for GNU/Linux:

```
$ cd /some/path/SenseCom-main/Linux/SenseCom_Linux_Latest/
```

4. List the files and check the executable permissions for the main SenseCom binary, `SenseCom.x86_64`:

```
$ ls -ahl

total 20M
drwxr-xr-x 3 mamadou mamadou   5 Apr 10 11:24 .
drwxr-xr-x 3 mamadou mamadou   5 Apr 10 11:24 ..
drwxr-xr-x 7 mamadou mamadou  34 Apr 10 11:24 SenseCom_Data
-rw-r--r-- 1 mamadou mamadou 15K Apr 10 11:24 SenseCom.x86_64
-rw-r--r-- 1 mamadou mamadou 33M Apr 10 11:24 UnityPlayer.so
```

5. As seen above the `SenseCom.x86_64` binary does not have the executable permission. Run the following command to set the executable permission for all users:

```
$ chmod a+x SenseCom.x86_64
```

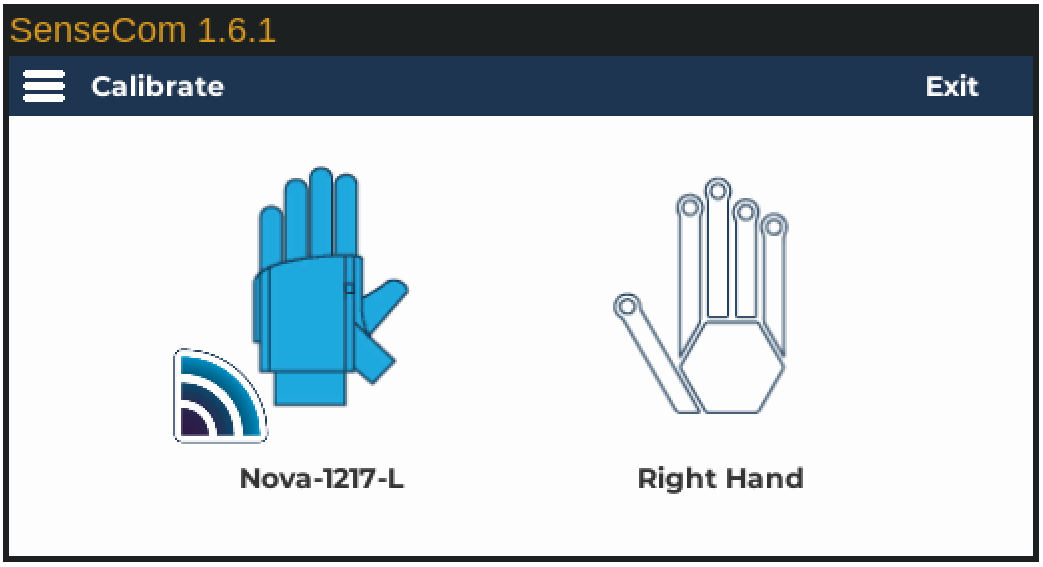
6. Veirfy the executable permission has been set on `SenseCom.x86_64`:

```
$ ls -l SenseCom.x86_64

-rwxr-xr-x 1 mamadou mamadou 14720 Apr 10 11:24 SenseCom.x86_64
```

7. Time to run the SenseCom executable:

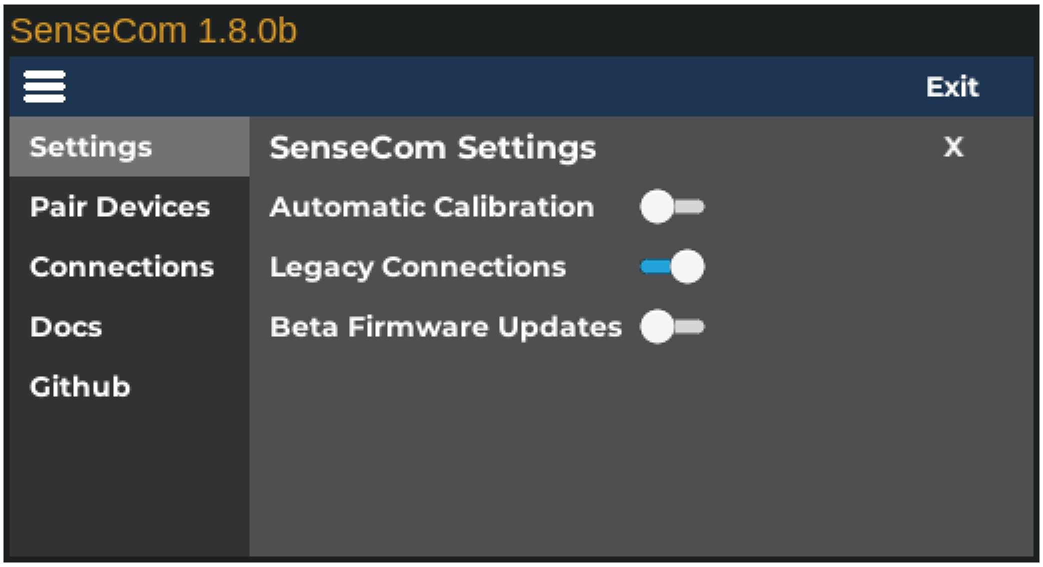
```
$ ./SenseCom.x86_64

[UnityMemory] Configuration Parameters - Can be set up in boot.config
    "memorysetup-bucket-allocator-granularity=16"
    "memorysetup-bucket-allocator-bucket-count=8"
    "memorysetup-bucket-allocator-block-size=4194304"
    "memorysetup-bucket-allocator-block-count=1"
    "memorysetup-main-allocator-block-size=16777216"
    "memorysetup-thread-allocator-block-size=16777216"
    "memorysetup-gfx-main-allocator-block-size=16777216"
    "memorysetup-gfx-thread-allocator-block-size=16777216"
    "memorysetup-cache-allocator-block-size=4194304"
    "memorysetup-typetree-allocator-block-size=2097152"
    "memorysetup-profiler-bucket-allocator-granularity=16"
    "memorysetup-profiler-bucket-allocator-bucket-count=8"
    "memorysetup-profiler-bucket-allocator-block-size=4194304"
    "memorysetup-profiler-bucket-allocator-block-count=1"
    "memorysetup-profiler-allocator-block-size=16777216"
    "memorysetup-profiler-editor-allocator-block-size=1048576"
    "memorysetup-temp-allocator-size-main=4194304"
    "memorysetup-job-temp-allocator-block-size=2097152"
    "memorysetup-job-temp-allocator-block-size-background=1048576"
    "memorysetup-job-temp-allocator-reduction-small-platforms=262144"
    "memorysetup-temp-allocator-size-background-worker=32768"
    "memorysetup-temp-allocator-size-job-worker=262144"
    "memorysetup-temp-allocator-size-preload-manager=262144"
    "memorysetup-temp-allocator-size-nav-mesh-worker=65536"
    "memorysetup-temp-allocator-size-audio-worker=65536"
    "memorysetup-temp-allocator-size-cloud-worker=32768"
    "memorysetup-temp-allocator-size-gfx=262144"
Loading in SingleInstance mode
```

8. If you have already paired any glove with your system, SenseCom should recognize and connect to your glove(s) shortly. If not, please follow the instructions on How to connect to Nova gloves using Blueman Bluetooth Manager or How to connect to Nova gloves using Command-line.



Important

If you are using SenseCom `v1.8.x+` and it fails to recognize your gloves with a Bluetooth Serial firmware, it may be because the `Legacy Connections` option is disabled, which is the default. Enabling this option should allow SenseCom to discover and connect to your glove.



Automatic Calibration: off
Legacy Connections: on
Beta Firmware Updates: off

Note

For more detailed information and troubleshooting, consult the SenseCom documentation page on SGDocs, please.

Connect to Nova gloves using Blueman Bluetooth Manager (Bluetooth Serial)

Follow these steps to pair a Nova glove with your PC on GNU/Linux usng the Blueman Bluetooth Manager:

1. Install Blueman Bluetooth Manager on your Linux distribution using the appropriate package manager:

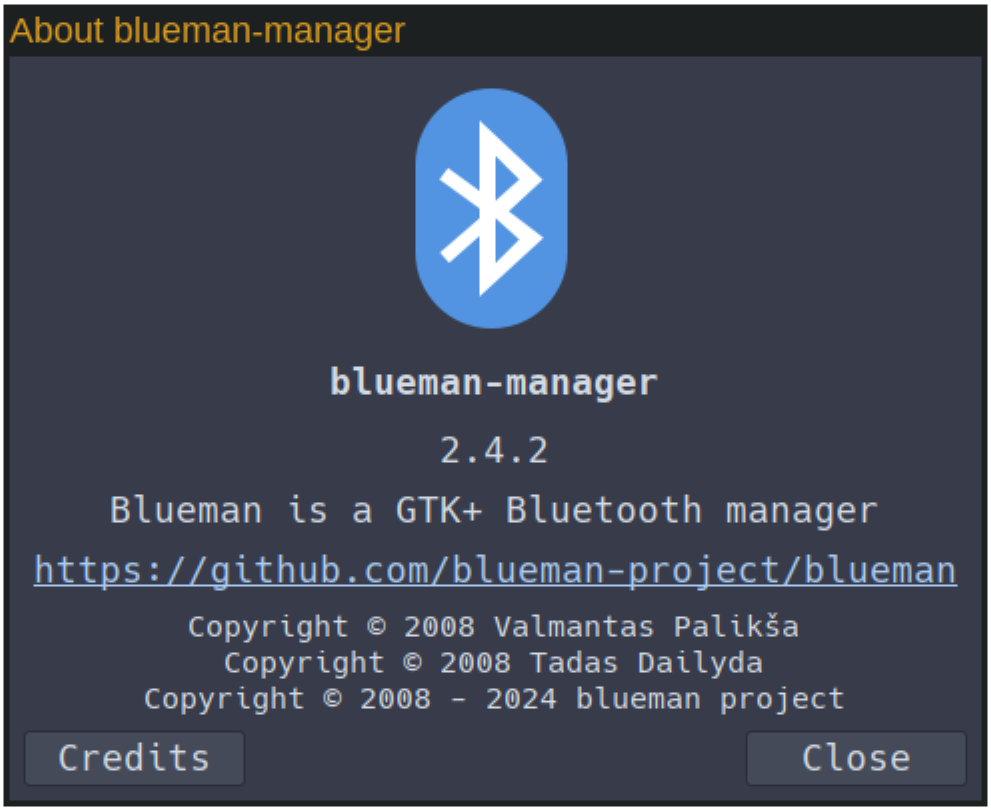
```
# Gentoo
$ emerge -atuv net-wireless/blueman

# Arch, Manjaro
$ sudo pacman -S blueman

# CentOS, Fedora, AlmaLinux, Rocky Linux
$ sudo dnf install blueman

# CentOS/RHEL
$ sudo yum install epel-release
$ sudo yum install blueman

# Debian, Ubuntu
$ sudo apt install blueman

# openSUSE
sudo zypper install blueman

# Solus
$ sudo eopkg install blueman

# Void Linux
$ sudo xbps-install -S blueman
```

> **Important**
>
> To properly set up the Bluetooth stack on your Linux distribution, additional steps may be required. For example, on Gentoo and Arch consult each distribution's official guide.

2. Ensure any glove you would like to pair with and connect to your system is not paired, or connected to any other device, such as another PC or VR headset.

3. Make sure the glove is turned on.

4. Start the Blueman Bluetooth Manager and verify you have a recent version installed by selecting `Help > About` from the application's menu.



About blueman-manager

blueman-manager

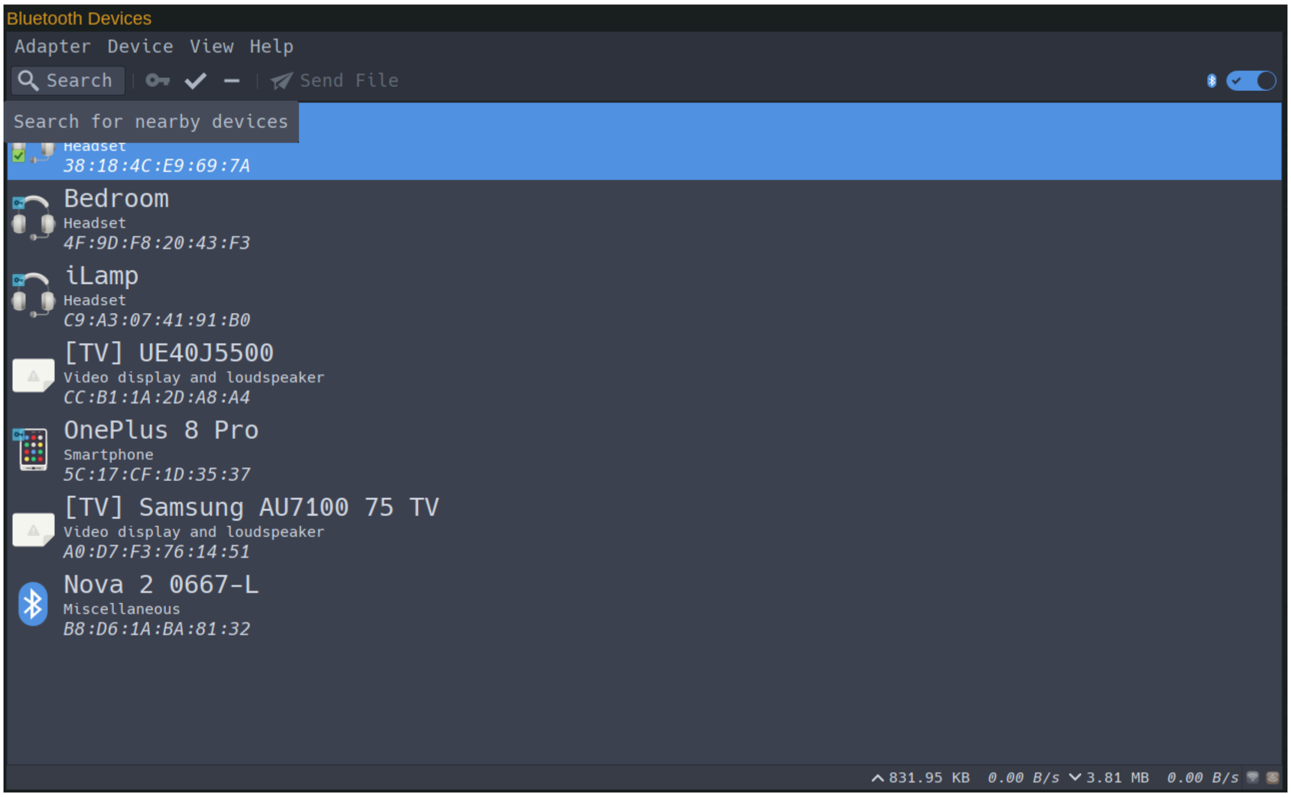
2.4.2

Blueman is a GTK+ Bluetooth manager

https://github.com/blueman-project/blueman

Copyright © 2008 Valmantas Palikša
Copyright © 2008 Tadas Dailyda
Copyright © 2008 - 2024 blueman project

5. If you don't see your glove, click the `Search` button on the toolbar or select `Adapter > Search` from the application's menu to look for new Bluetooth devices.

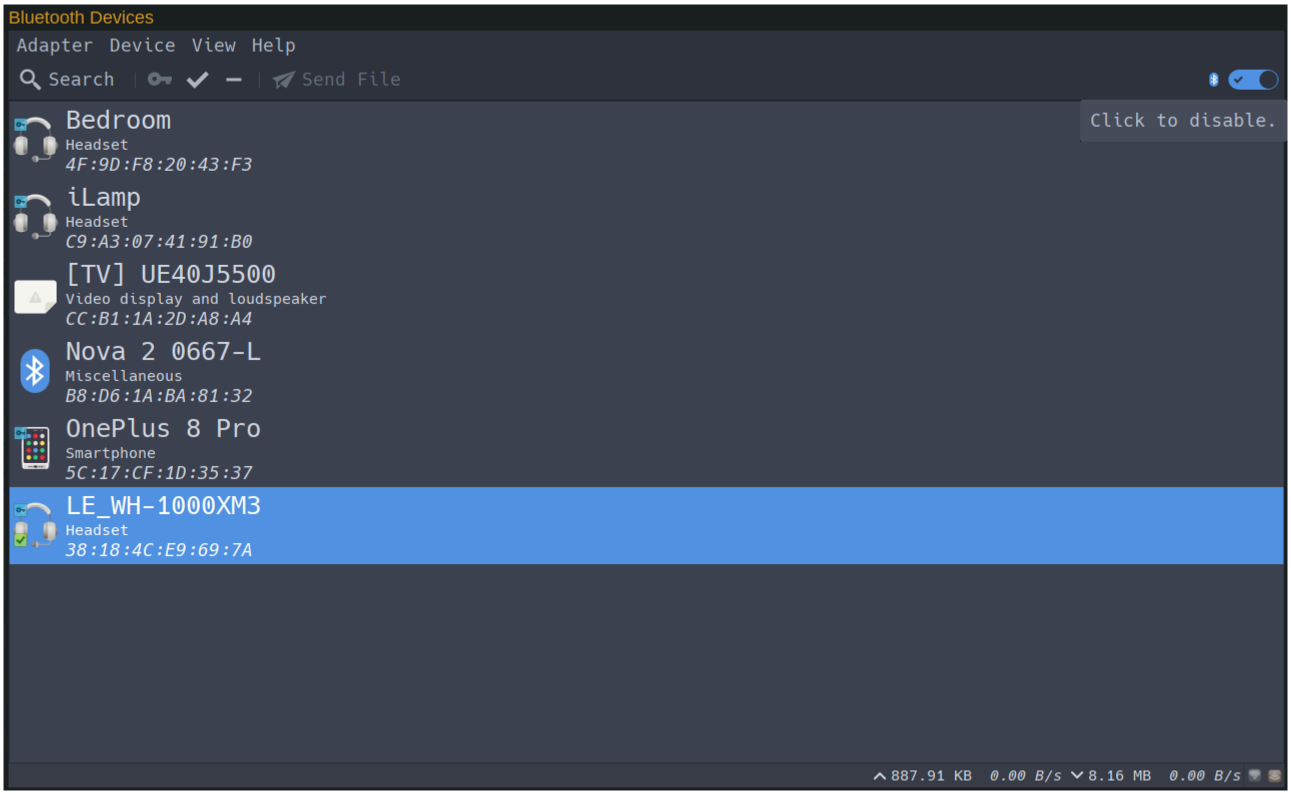


```
Headset
38:18:4C:E9:69:7A
Bedroom
Headset
4F:9D:F8:20:43:F3
iLamp
Headset
C9:A3:07:41:91:B0
[TV] UE40J5500
Video display and loudspeaker
CC:B1:1A:2D:A8:A4
OnePlus 8 Pro
Smartphone
5C:17:CF:1D:35:37
[TV] Samsung AU7100 75 TV
Video display and loudspeaker
A0:D7:F3:76:14:51
Nova 2 0667-L
Miscellaneous
B8:D6:1A:BA:81:32
```

```
831.95 KB  0.00 B/s  3.81 MB  0.00 B/s
```

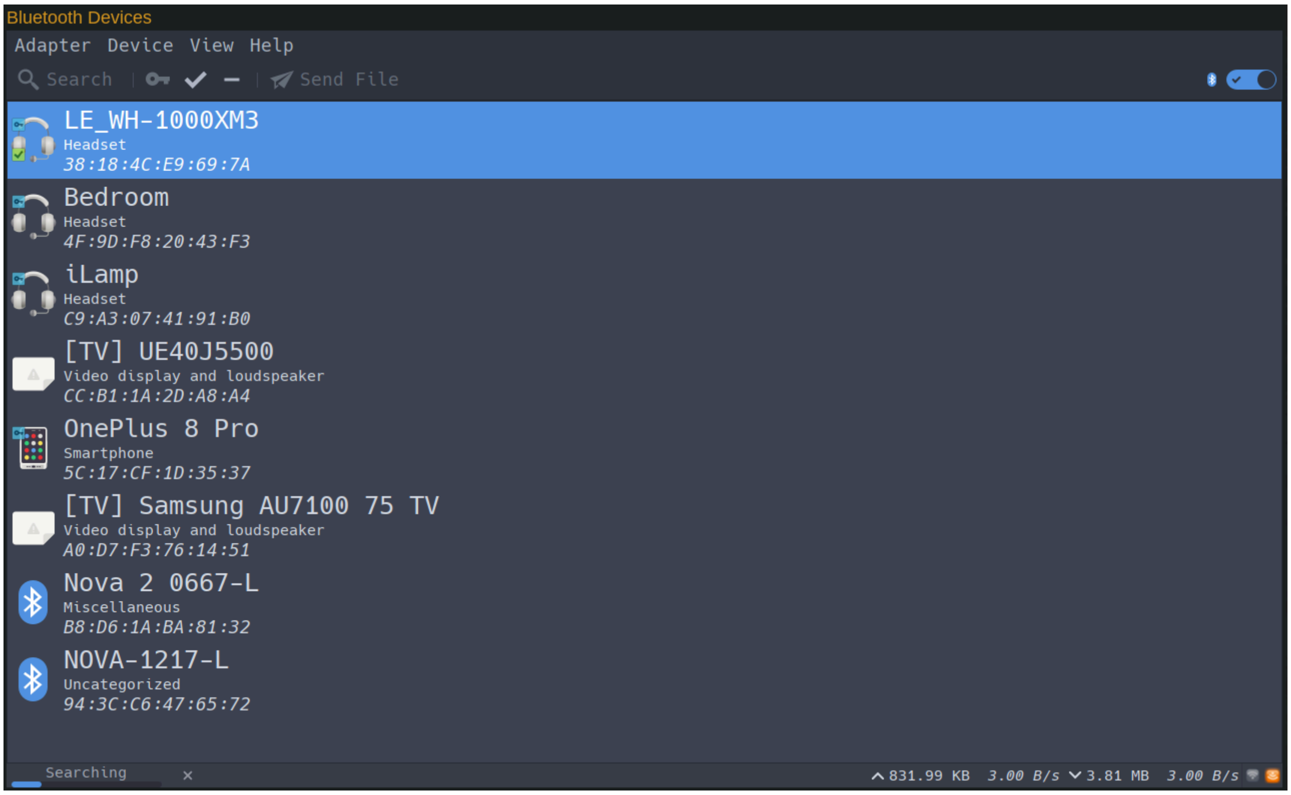
Important

Before starting the search operation, ensure that your PC's Bluetooth controller is turned on by verifying its status on the right side of the toolbar next to the Bluetooth logo. If disabled, the Search button will be grayed out.



```
Bedroom
Headset
4F:9D:F8:20:43:F3
iLamp
Headset
C9:A3:07:41:91:B0
[TV] UE40J5500
Video display and loudspeaker
CC:B1:1A:2D:A8:A4
Nova 2 0667-L
Miscellaneous
B8:D6:1A:BA:81:32
OnePlus 8 Pro
Smartphone
5C:17:CF:1D:35:37
LE_WH-1000XM3
Headset
38:18:4C:E9:69:7A

887.91 KB  0.00 B/s  8.16 MB  0.00 B/s
```

6. A progress bar will appear on the application's status bar. If a new device is found, it will be listed in the main device list area.

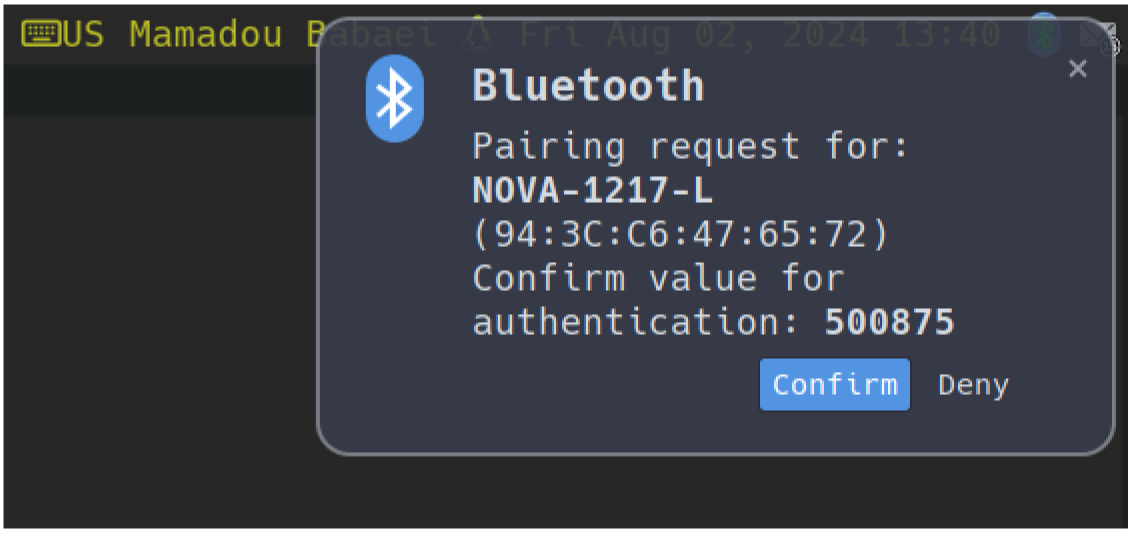


```
LE_WH-1000XM3
Headset
38:18:4C:E9:69:7A
Bedroom
Headset
4F:9D:F8:20:43:F3
iLamp
Headset
C9:A3:07:41:91:B0
[TV] UE40J5500
Video display and loudspeaker
CC:B1:1A:2D:A8:A4
OnePlus 8 Pro
Smartphone
5C:17:CF:1D:35:37
[TV] Samsung AU7100 75 TV
Video display and loudspeaker
A0:D7:F3:76:14:51
Nova 2 0667-L
Miscellaneous
B8:D6:1A:BA:81:32
NOVA-1217-L
Uncategorized
94:3C:C6:47:65:72
```

7. Once the glove is found, click on it to select it.

8. Either right-click on the device, or go to the `Device` menu, then choose `Pair`.

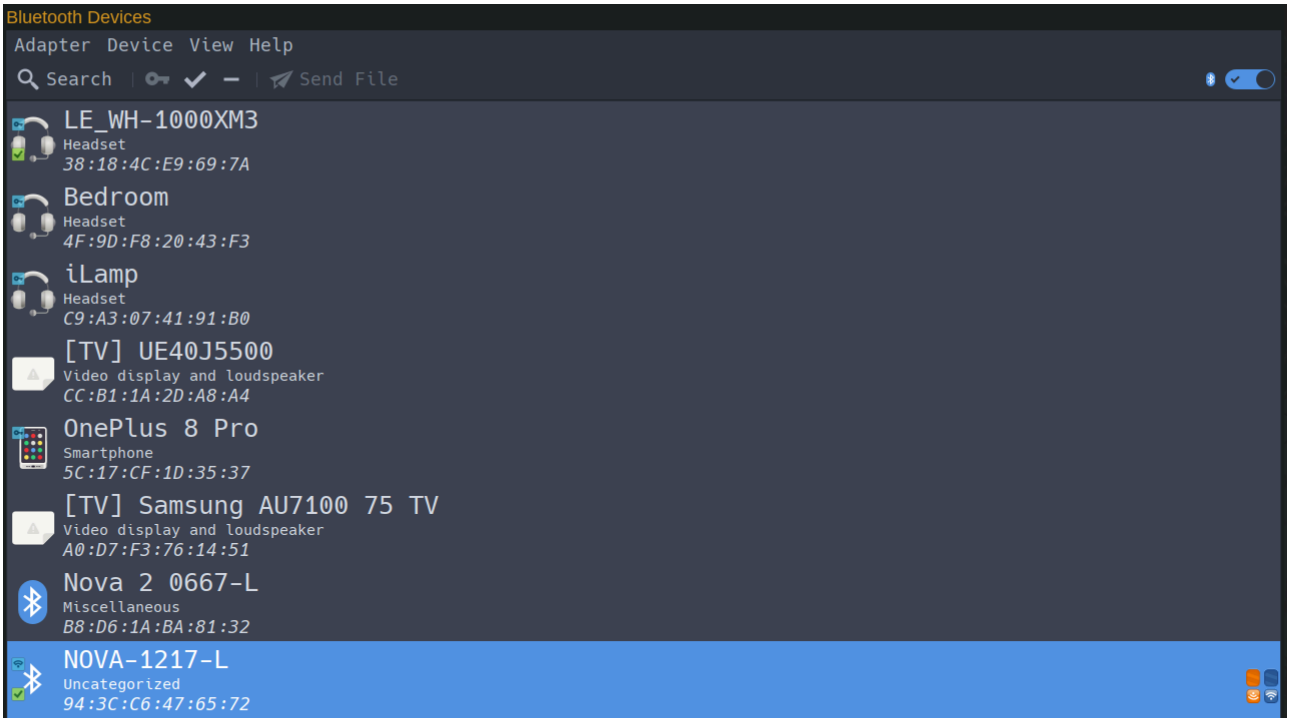
9. Blueman will prompt you to pair the glove with a notification. Click `Confirm` to proceed.



```
Bluetooth
Pairing request for:
NOVA-1217-L
(94:3C:C6:47:65:72)
Confirm value for
authentication: 500875
```

9. After pairing, either right-click on the device again, or go to the `Device` menu, then choose `Trust`.

10. If everything has been successful, the key icon indicates successful pairing, and the checkmark confirms the device is trusted.



```
LE_WH-1000XM3
Headset
38:18:4C:E9:69:7A
Bedroom
Headset
4F:9D:F8:20:43:F3
iLamp
Headset
C9:A3:07:41:91:B0
[TV] UE40J5500
Video display and loudspeaker
CC:B1:1A:2D:A8:A4
OnePlus 8 Pro
Smartphone
5C:17:CF:1D:35:37
[TV] Samsung AU7100 75 TV
Video display and loudspeaker
A0:D7:F3:76:14:51
Nova 2 0667-L
Miscellaneous
B8:D6:1A:BA:81:32
NOVA-1217-L
Uncategorized
94:3C:C6:47:65:72
```

11. Follow the SenseCom on GNU/Linux instructions and you should be able to successfully connect to the newly paired glove from SenseCom.

Video Tutorial

There is also a video tutorial demonstrating how to connect to Nova gloves on GNU/Linux using Blueman Bluetooth Manager.

Connect to Nova gloves using Command-line (Bluetooth Serial)

Follow these steps to pair a Nova glove to your PC on GNU/Linux usng command-line and Bluez:

1. Some Linux distributions include BlueZ in their default installation. If yours doesn't, install it using the appropriate package manager:

```
# Gentoo
$ emerge -atuv net-wireless/bluez

# Arch, Manjaro
$ sudo pacman -S bluez

# CentOS, Fedora, AlmaLinux, Rocky Linux
$ sudo dnf install bluez

# CentOS/RHEL
$ sudo yum install bluez

# Debian, Ubuntu
$ sudo apt install bluez

# openSUSE
sudo zypper install bluez

# Solus
$ sudo eopkg install bluez

# Void Linux
$ sudo xbps-install -S bluez
```

Important

To properly set up the Bluetooth stack on your Linux distribution, additional steps may be required. For example, on Gentoo and Arch consult each distribution's official guide.

2. Run the following command to ensure that BlueZ is installed and check your `bluetoothctl` version:

```
bluetoothctl version
Version 5.77
```

3. Ensure that the `bluetooth` service is started and running. For example, on Gentoo Linux:

```
$  rc-service bluetooth start
```

You might see one of these outputs based on whether it's already running or not:

```
 * Starting bluetooth ...
# or
 * WARNING: bluetooth has already been started
```

4. Ensure any glove you would like to pair with and connect to your system is not paired, or connected to any other device, such as another PC or VR headset.

5. Make sure the glove is turned on.

6. Use `bluetoothctl list` or `bluetoothctl show` command to extract your PC's Bluetooth Controller MAC Address which is useful for later on:

```
$ bluetoothctl list

Controller CC:15:31:90:69:87 BlueZ 5.77 [default]

$ bluetoothctl show
Controller CC:15:31:90:69:87 (public)
    Manufacturer: 0x0002 (2)
    Version: 0x0b (11)
    Name: BlueZ 5.77
    Alias: BlueZ 5.77
    Class: 0x007c010c (8126732)
    Powered: yes
    PowerState: on
    Discoverable: no
    DiscoverableTimeout: 0x0000003c (60)
    Pairable: no
    UUID: Message Notification Se.. (00001133-0000-1000-8000-00805f9b34fb)
    UUID: A/V Remote Control        (0000110e-0000-1000-8000-00805f9b34fb)
    UUID: OBEX Object Push          (00001105-0000-1000-8000-00805f9b34fb)
    UUID: Message Access Server     (00001132-0000-1000-8000-00805f9b34fb)
    UUID: PnP Information           (00001200-0000-1000-8000-00805f9b34fb)
    UUID: IrMC Sync                 (00001104-0000-1000-8000-00805f9b34fb)
    UUID: Headset                   (00001108-0000-1000-8000-00805f9b34fb)
    UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
    UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
    UUID: Phonebook Access Server   (0000112f-0000-1000-8000-00805f9b34fb)
    UUID: Audio Sink                (0000110b-0000-1000-8000-00805f9b34fb)
    UUID: Device Information        (0000180a-0000-1000-8000-00805f9b34fb)
    UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
    UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-00805f9b34fb)
    UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
    UUID: OBEX File Transfer        (00001106-0000-1000-8000-00805f9b34fb)
    Modalias: usb:v1D6Bp0246d054D
    Discovering: no
    Roles: central
    Roles: peripheral
Advertising Features:
    ActiveInstances: 0x00 (0)
    SupportedInstances: 0x0c (12)
    SupportedIncludes: tx-power
    SupportedIncludes: appearance
    SupportedIncludes: local-name
    SupportedSecondaryChannels: 1M
    SupportedSecondaryChannels: 2M
    SupportedCapabilities.MinTxPower: 0xffffffde (-34)
    SupportedCapabilities.MaxTxPower: 0x0007 (7)
    SupportedCapabilities.MaxAdvLen: 0xfb (251)
    SupportedCapabilities.MaxScnRspLen: 0xfb (251)
    SupportedFeatures: CanSetTxPower
    SupportedFeatures: HardwareOffload
```

7. Ensure the controller is powered on:

```
$ bluetoothctl power on
```

```
Changing power on succeeded
```

8. Enable the agent to listen for Bluetooth events that require user interaction, such as pairing requests and managing device authorizations:

```
$ bluetoothctl agent on
```

9. Set the current agent as the default agent:

```
$ bluetoothctl default-agent
```

```
No agent is registered
```

10. Set the controller to be discoverable for `180` seconds:

```
$ bluetoothctl discoverable on
```

```
bluetoothctl discoverable on
hci0 new_settings: powered connectable ssp br/edr le secure-conn wide-band-
speech
hci0 new_settings: powered connectable discoverable ssp br/edr le secure-conn
wide-band-speech
Changing discoverable on succeeded
```

> **Note**
>
> To change the default discoverable timeout, you can set it manually using the `bluetoothctl discoverable-timeout` command.

```
$ bluetoothctl discoverable-timeout 300
```

```
Changing discoverable-timeout 300 succeeded
```

11. Then, make the controller pairable as well:

```
$ bluetoothctl pairable on
```

```
hci0 new_settings: powered connectable discoverable bondable ssp br/edr le
secure-conn wide-band-speech
Changing pairable on succeeded
```

12. Begin scanning for devices:

```
$ bluetoothctl scan on
```

```
SetDiscoveryFilter success
```

13. After a few seconds, list the discovered devices:

bluetoothctl devices

```
Device 78:D2:52:42:33:2F 78-D2-52-42-33-2F
Device 94:3C:C6:47:65:72 NOVA-1217-L
Device AC:F1:08:37:9F:93 LG DSN7CY(93)
Device 70:D6:10:9D:73:8F 70-D6-10-9D-73-8F
Device 7F:2C:8C:8D:09:9F 7F-2C-8C-8D-09-9F
Device F9:56:4B:86:1E:13 F9-56-4B-86-1E-13
Device C9:A3:07:41:91:B0 iLamp
Device 4F:9D:F8:20:43:F3 Bedroom
Device CC:B1:1A:2D:A8:A4 [TV] UE40J5500
Device A0:D7:F3:76:14:51 [TV] Samsung AU7100 75 TV
Device 5C:17:CF:1D:35:37 OnePlus 8 Pro
Device E2:F8:03:F6:D8:CB E2-F8-03-F6-D8-CB
Device 38:18:4C:E9:69:7A LE_WH-1000XM3
Device B8:D6:1A:BA:81:32 Nova 2 0667-L
```

Note

If your device is not listed yet, you can run this command multiple times as `bluetoothctl` continues the device discovery in the background.

14. Use the following command to pair with the discoved glove:

```
$ bluetoothctl pair GLOVE_MAC_ADDRESS
```

For example:

```
$ bluetoothctl pair 94:3C:C6:47:65:72

Attempting to pair with 94:3C:C6:47:65:72
[CHG] Device 94:3C:C6:47:65:72 Connected: yes
[CHG] Device 94:3C:C6:47:65:72 Bonded: yes
[CHG] Device 94:3C:C6:47:65:72 UUIDs: 00001101-0000-1000-8000-00805f9b34fb
[CHG] Device 94:3C:C6:47:65:72 ServicesResolved: yes
[CHG] Device 94:3C:C6:47:65:72 Paired: yes
Pairing successful
```

> **Note**
>
> If you encounter the `Failed to pair: org.bluez.Error.AuthenticationFailed` error message, it might be misleading. Check if there is a line with the glove's MAC address followed by `Connected: yes` , which indicates that the connection was actually successful.

```
Attempting to pair with 94:3C:C6:47:65:72
[CHG] Device 94:3C:C6:47:65:72 Connected: yes
Failed to pair: org.bluez.Error.AuthenticationFailed
```

15. Mark the device as trusted by issuing the following command:

```
$ bluetoothctl trust GLOVE_MAC_ADDRESS
```

For example:

```
$ bluetoothctl trust 94:3C:C6:47:65:72

[CHG] Device 94:3C:C6:47:65:72 Trusted: yes
Changing 94:3C:C6:47:65:72 trust succeeded
```

16. Attempt to connect to the glove again:

```
$ bluetoothctl connect GLOVE_MAC_ADDRESS
```

For example:

```
$ bluetoothctl connect 94:3C:C6:47:65:72

Attempting to connect to 94:3C:C6:47:65:72
[CHG] Device 38:18:4C:E9:69:7A RSSI: 0xffffffd0 (-48)
[CHG] Device 94:3C:C6:47:65:72 Connected: yes
[CHG] Device 94:3C:C6:47:65:72 UUIDs: 00001101-0000-1000-8000-00805f9b34fb
[CHG] Device 94:3C:C6:47:65:72 ServicesResolved: yes
Failed to connect: org.bluez.Error.NotAvailable br-connection-profile-
unavailable
```

Note

Again, the error message may be misleading. The connection is often successful despite the error.

17. If desired, you can extract some information from the glove using:

```
$ bluetoothctl info GLOVE_MAC_ADDRESS
```

For example:

```
bluetoothctl info 94:3C:C6:47:65:72
Device 94:3C:C6:47:65:72 (public)
    Name: NOVA-1217-L
    Alias: NOVA-1217-L
    Class: 0x00001f00 (7936)
    Paired: yes
    Bonded: yes
    Trusted: yes
    Blocked: no
    Connected: yes
    LegacyPairing: no
    UUID: Serial Port               (00001101-0000-1000-8000-00805f9b34fb)
```

18. Create an RFCOMM device:

```
$ sudo rfcomm connect /dev/rfcommX GLOVE_MAC_ADDRESS CHANNEL_NUMBER
```

For example:

```
$ sudo rfcomm connect /dev/rfcomm0 94:3C:C6:47:65:72 1

Connected /dev/rfcomm0 to 94:3C:C6:47:65:72 on channel 1
Press CTRL-C for hangup
```

> **Note**
>
> The `rfcomm` command requires root permision, so it must be run with `sudo`.

> **Tip**
>
> To determine the channel number, run the following command:

```
$ sdptool browse GLOVE_MAC_ADDRESS
```

```
$ sdptool browse 94:3C:C6:47:65:72
Browsing 94:3C:C6:47:65:72 ...
Service Name: SPP_SERVER
Service RecHandle: 0x10000
Service Class ID List:
  "Serial Port" (0x1101)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Profile Descriptor List:
  "Serial Port" (0x1101)
    Version: 0x010
```

> **Note**
>
> If you have more than one glove or in general multiple serial Bluetooth devices connected to your device connected to your PC, then `/dev/rfcomm0` may already be allocated to another device. In that case, increment the number until finding a

> free rfcomm device. You can query the existing rfcomm devices using the command: `ls /dev/rfcomm*`.

19. Follow the SenseCom on GNU/Linux instructions and you should be able to successfully connect to the newly paired glove from SenseCom.
20. Once the SenseCom is closed and we are done with the gloves, we can disconnect the gloves using:

```
$ bluetoothctl disconnect ${SG_DEVICE}
$ sudo rfcomm release ${SG_RFCOMM}
```

For example:

```
$ bluetoothctl disconnect 94:3C:C6:47:65:72
$ sudo rfcomm release /dev/rfcomm0
```

> **Note**
>
> Again, the `rfcomm` command requires elevated permissions, so it must be run with the `sudo` command.

Scripts to Easily Connect and Disconnect from a Glove

You can automate the above tedious process using scripts for connecting and disconnecting gloves.

`sg-connect.sh`:

```
#!/usr/bin/env sh

CTRL_DEVICE="YOUR_BLUETOOTH_CONTROLLER_MAC_ADDRESS"
SG_DEVICE="YOUR_SENSEGLOVE_MAC_ADDRESS"
SG_RFCOMM="/dev/rfcomm0"

bluetoothctl pairable on
bluetoothctl discoverable on
bluetoothctl pair ${SG_DEVICE}
bluetoothctl trust ${SG_DEVICE}
bluetoothctl connect ${SG_DEVICE}
rfcomm connect ${SG_RFCOMM} ${SG_DEVICE} 1 &
```

sg-disconnect.sh:

```
#!/usr/bin/env sh

SG_DEVICE="YOUR_SENSEGLOVE_MAC_ADDRESS"
SG_RFCOMM="/dev/rfcomm0"

bluetoothctl disconnect ${SG_DEVICE}
rfcomm release ${SG_RFCOMM}
```

Example Scripts for a Left-Handed Glove

```
$ cat sg-connect-left.sh

#!/usr/bin/env sh

CTRL_DEVICE="CC:15:31:90:69:87"
SG_DEVICE="94:3C:C6:47:65:72"
SG_RFCOMM="/dev/rfcomm0"

bluetoothctl pairable on
bluetoothctl discoverable on
bluetoothctl pair ${SG_DEVICE}
bluetoothctl trust ${SG_DEVICE}
bluetoothctl connect ${SG_DEVICE}
rfcomm connect ${SG_RFCOMM} ${SG_DEVICE} 1 &

$ cat sg-disconnect-left.sh

#!/usr/bin/env sh

SG_DEVICE="94:3C:C6:47:65:72"
SG_RFCOMM="/dev/rfcomm0"

bluetoothctl disconnect ${SG_DEVICE}
rfcomm release ${SG_RFCOMM}

# Set the executable permissions for all users:
$ chmod a+x sg-connect-left.sh
$ chmod a+x sg-disconnect-left.sh

# Before running SenseCom:

$ sudo ./sg-connect-left.sh

Password:

Changing pairable on succeeded
hci0 new_settings: powered connectable bondable ssp br/edr le secure-conn
wide-band-speech
hci0 new_settings: powered connectable discoverable bondable ssp br/edr le
secure-conn wide-band-speech
Changing discoverable on succeeded
Attempting to pair with 94:3C:C6:47:65:72
Failed to pair: org.bluez.Error.AlreadyExists
Changing 94:3C:C6:47:65:72 trust succeeded
Attempting to connect to 94:3C:C6:47:65:72
```

```
hci0 94:3C:C6:47:65:72 type BR/EDR connected eir_len 18
[CHG] Device 94:3C:C6:47:65:72 Connected: yes
[CHG] Device 94:3C:C6:47:65:72 ServicesResolved: yes
Failed to connect: org.bluez.Error.NotAvailable br-connection-profile-
unavailable

# Run SenseCom in between!

# Once SenseCom is closed:

$ sudo ./sg-disconnect-left.sh

sudo ./sg-disconnect-left.sh

Password:

Attempting to disconnect from 94:3C:C6:47:65:72
hci0 94:3C:C6:47:65:72 type BR/EDR disconnected with reason 2
[CHG] Device 94:3C:C6:47:65:72 ServicesResolved: no
Successful disconnected
Can't release device: No such device
```

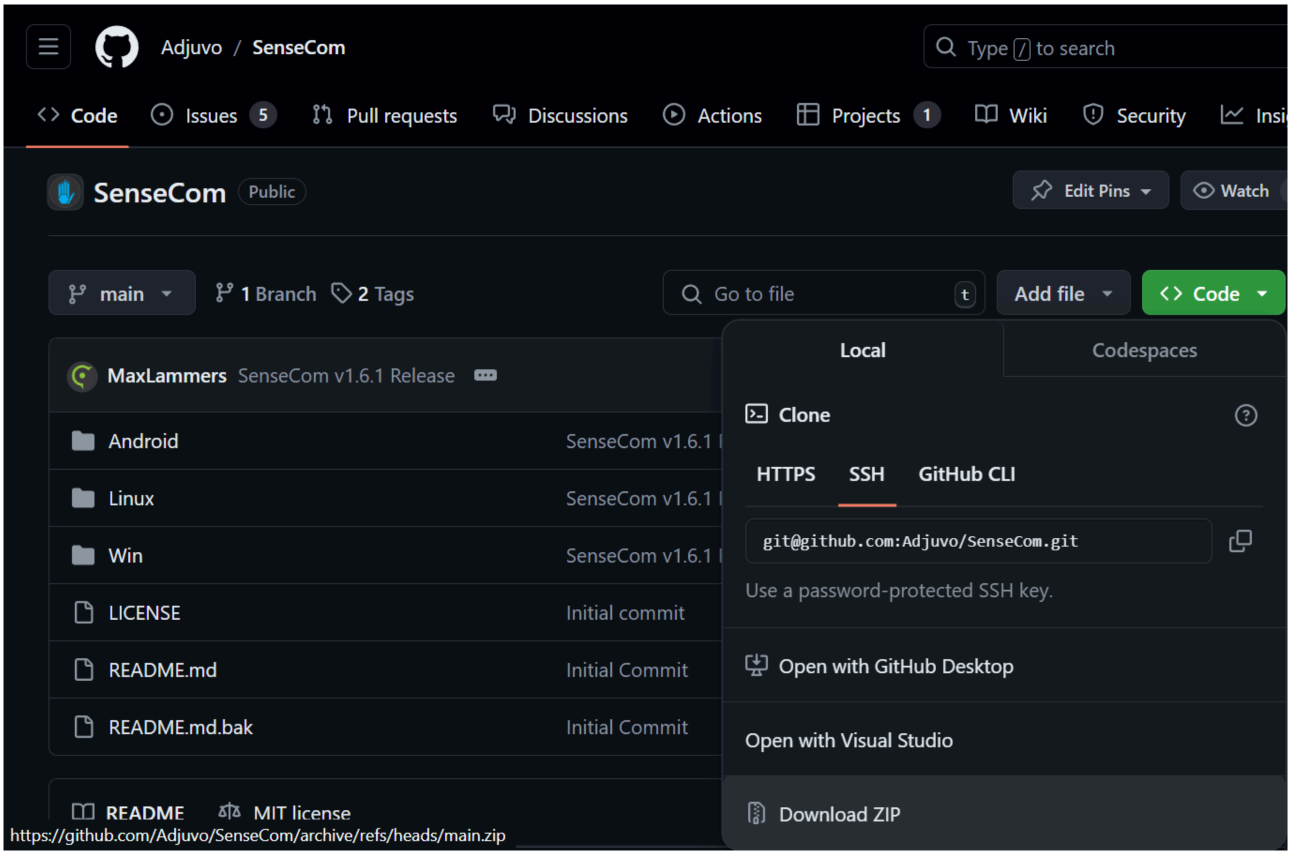
Video Tutorial

There is also a video tutorial demonstrating how to connect to Nova gloves on GNU/Linux using the command line.

SenseCom on Microsoft Windows (Bluetooth Serial)

Follow these steps to quickly set up and run SenseCom on Microsoft Windows:

1. First, obtain the SenseCom binaries from its GitHub repository.



MaxLammers	SenseCom v1.6.1 Release
Android	SenseCom v1.6.1
Linux	SenseCom v1.6.1
Win	SenseCom v1.6.1
LICENSE	Initial commit
README.md	Initial Commit
README.md.bak	Initial Commit

```
git@github.com:Adjuvo/SenseCom.git
```

Use a password-protected SSH key.

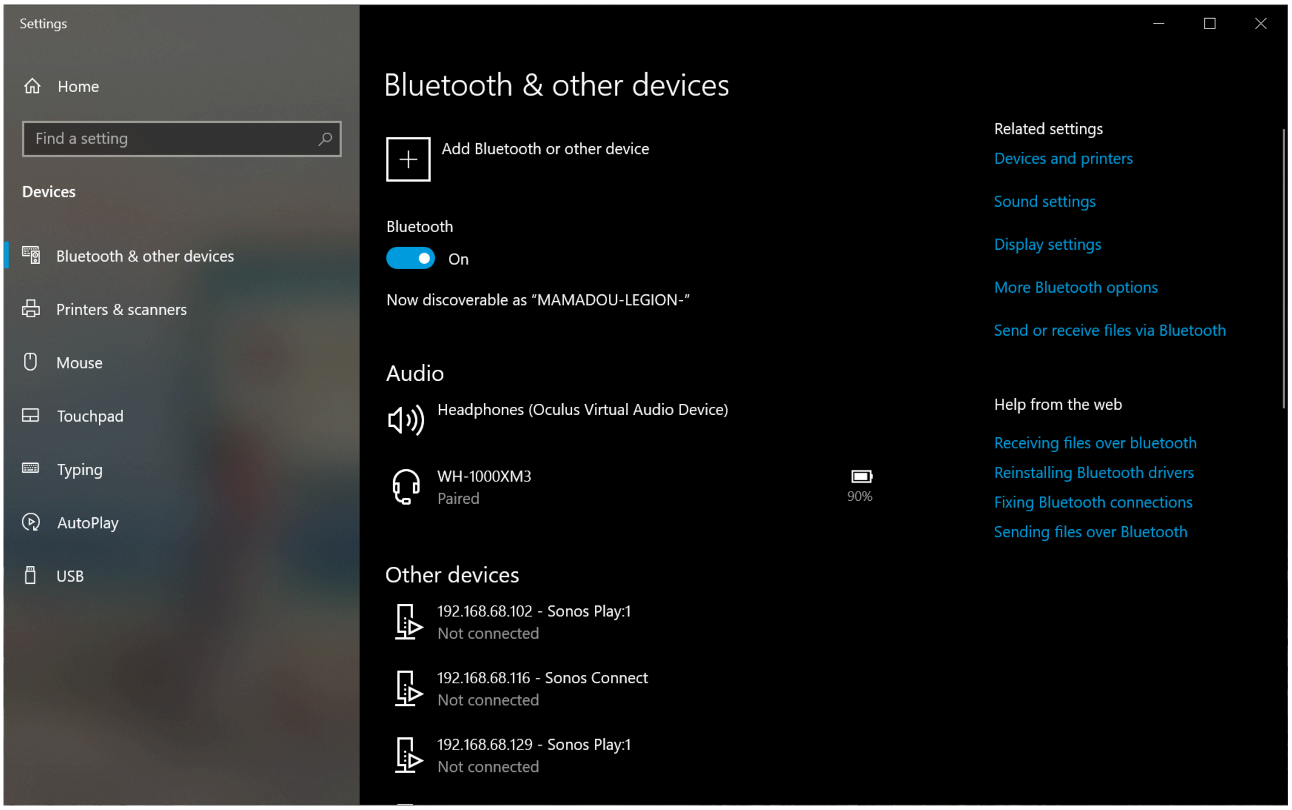
https://github.com/Adjuvo/SenseCom/archive/refs/heads/main.zip

2. Extract the SenseCom `.zip` file to a location on your computer after downloading it.

3. Ensure any glove you would like to pair with and connect to your system is not paired, or connected to any other device, such as another PC or VR headset.

4. Make sure the glove is powered on.

5. Access Windows Bluetooth Settings by navigating to `Settings > Devices > Bluetooth & other devices`.


Settings

Home

Find a setting

Devices

Bluetooth & other devices
Printers & scanners
Mouse
Touchpad
Typing
AutoPlay
USB

Bluetooth & other devices

Add Bluetooth or other device

Bluetooth
On

Now discoverable as "MAMADOU-LEGION-"

Audio

Headphones (Oculus Virtual Audio Device)

WH-1000XM3
Paired
90%

Other devices

192.168.68.102 - Sonos Play:1
Not connected

192.168.68.116 - Sonos Connect
Not connected

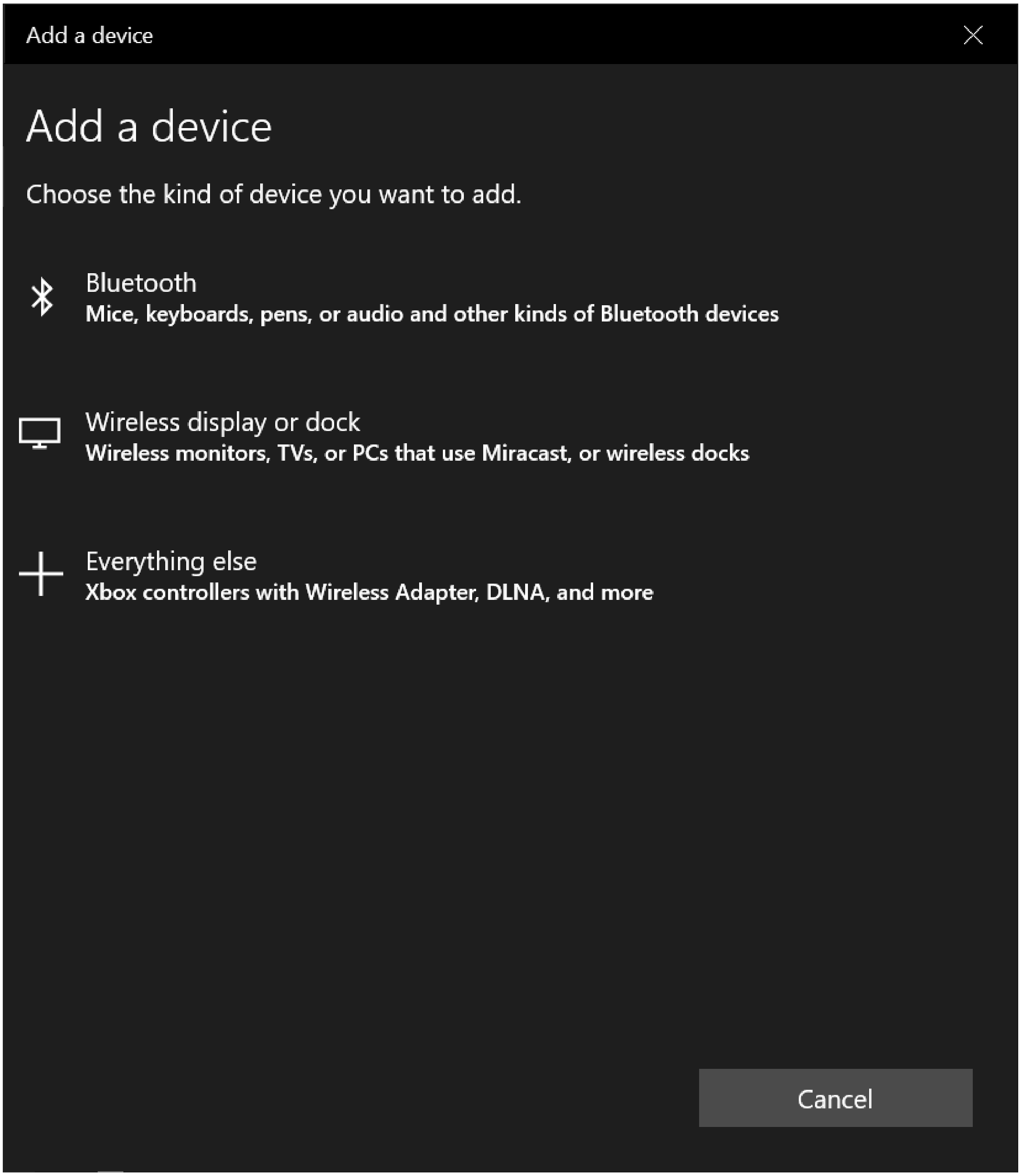
192.168.68.129 - Sonos Play:1
Not connected

Related settings
Devices and printers
Sound settings
Display settings
More Bluetooth options
Send or receive files via Bluetooth

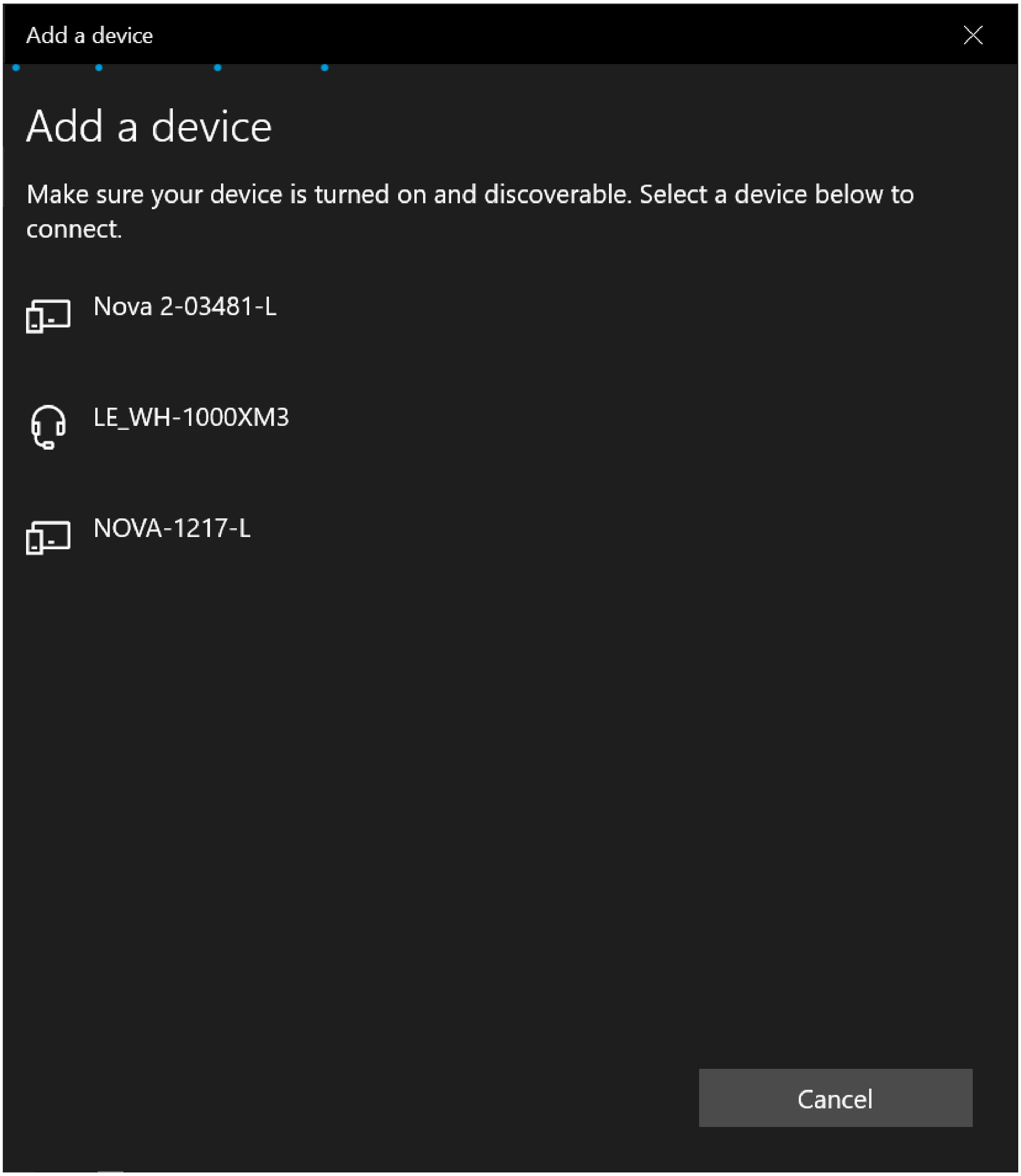
Help from the web
Receiving files over bluetooth
Reinstalling Bluetooth drivers
Fixing Bluetooth connections
Sending files over Bluetooth

6. Click on `Add Bluetooth or other devices`.

7. In the new window click on `Bluetooth`.


Add a device
Add a device
Choose the kind of device you want to add.
Bluetooth
Mice, keyboards, pens, or audio and other kinds of Bluetooth devices
Wireless display or dock
Wireless monitors, TVs, or PCs that use Miracast, or wireless docks
Everything else
Xbox controllers with Wireless Adapter, DLNA, and more
Cancel

8. Wait for the glove to be discovered, then click on it.



Add a device

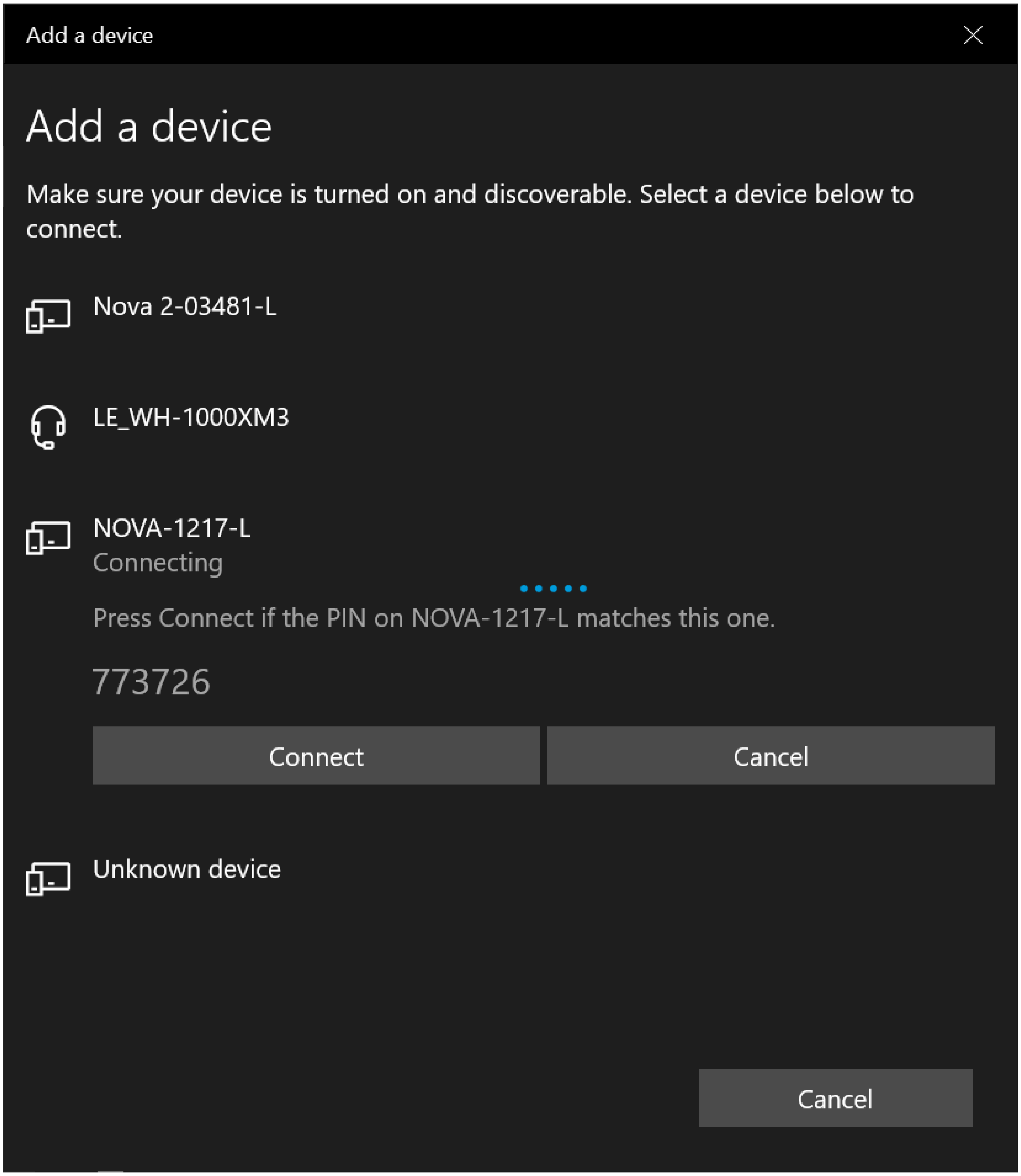
Make sure your device is turned on and discoverable. Select a device below to connect.

Nova 2-03481-L

LE_WH-1000XM3

NOVA-1217-L

9. Click `Connect` to connect and pair the glove.



Add a device

Add a device

Make sure your device is turned on and discoverable. Select a device below to connect.

Nova 2-03481-L

LE_WH-1000XM3

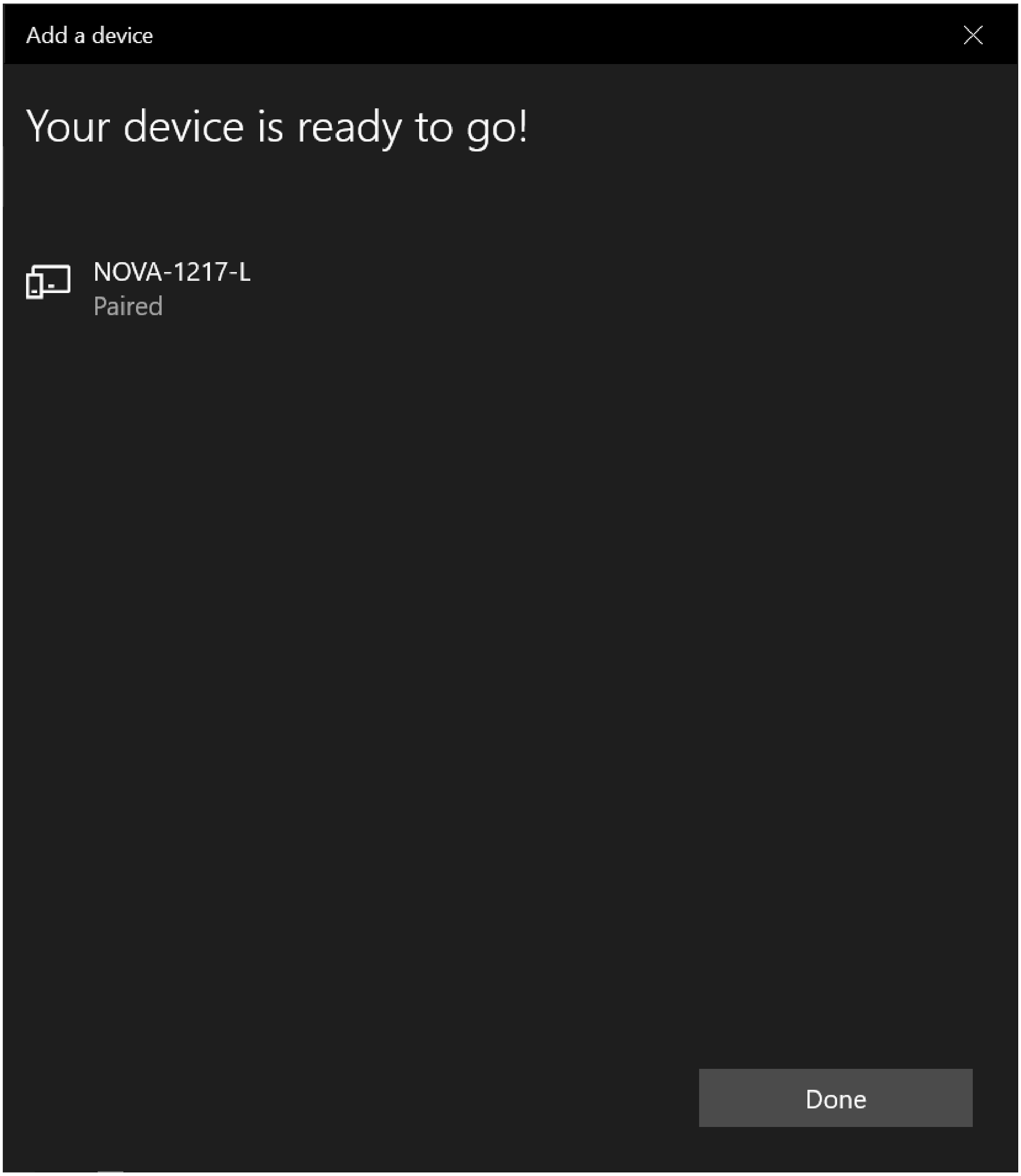
NOVA-1217-L
Connecting

Press Connect if the PIN on NOVA-1217-L matches this one.

773726

Unknown device

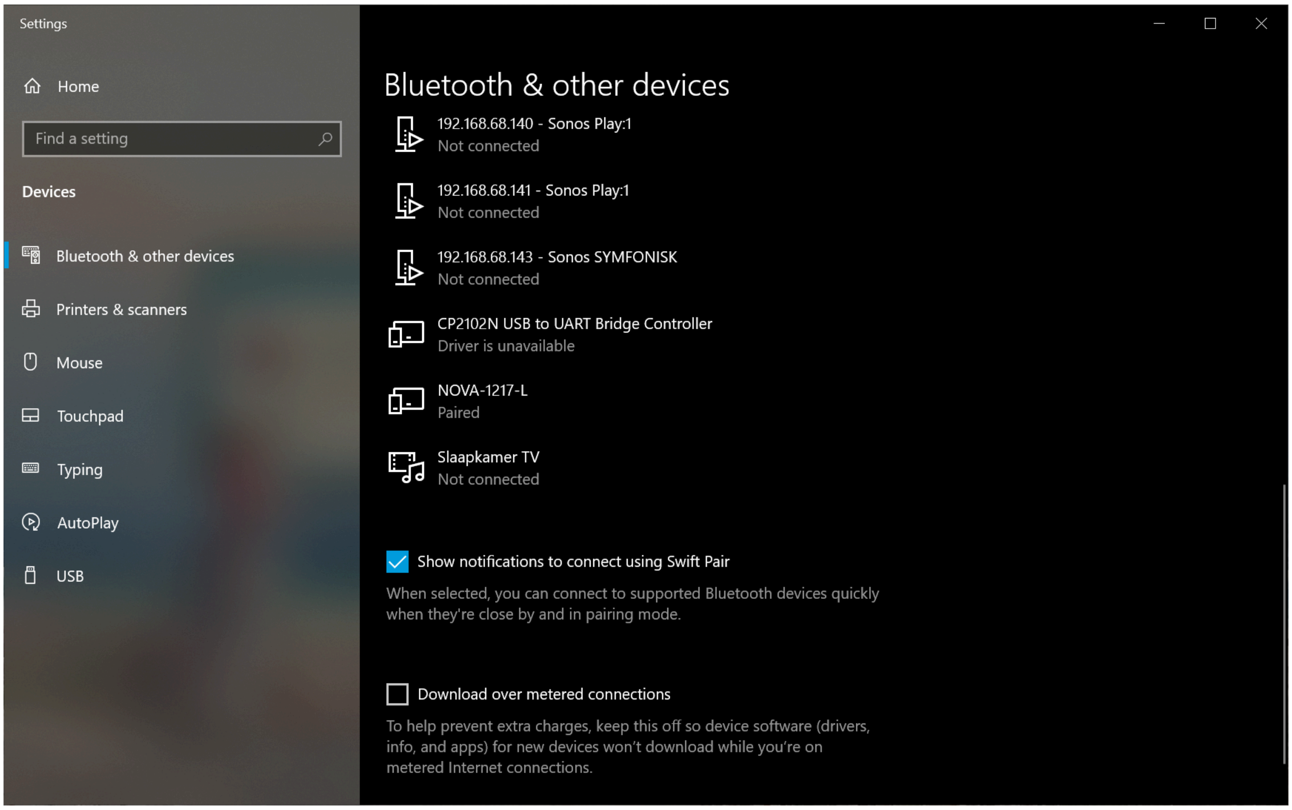
10. Once the glove is paired, you're good to go. Click on `Done` .


Add a device

Your device is ready to go!

NOVA-1217-L
Paired

11. Once you are back to Windows Bluetooth settings, verify that the glove is listed as a paired device.



Bluetooth & other devices

192.168.68.140 - Sonos Play:1
Not connected

192.168.68.141 - Sonos Play:1
Not connected

192.168.68.143 - Sonos SYMFONISK
Not connected

CP2102N USB to UART Bridge Controller
Driver is unavailable

NOVA-1217-L
Paired

Slaapkamer TV
Not connected

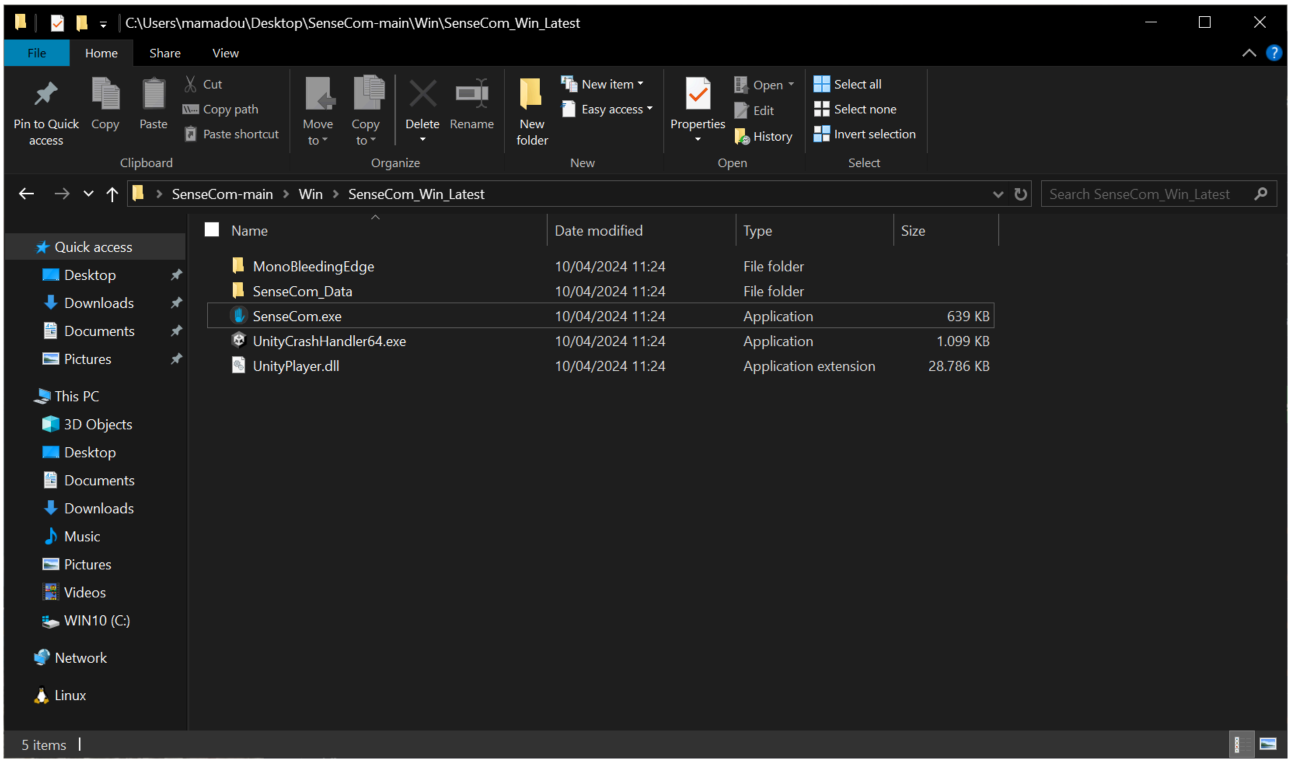
[✓] Show notifications to connect using Swift Pair

When selected, you can connect to supported Bluetooth devices quickly when they're close by and in pairing mode.

[ ] Download over metered connections

To help prevent extra charges, keep this off so device software (drivers, info, and apps) for new devices won't download while you're on metered Internet connections.

12. After successfully paring your glove, it's time to run SenseCom. Navigate to the folder where you extracted SenseCom and go to `/path/to/extracted/SenseCom/directory/Win/SenseCom_Win_Latest`, and then run the executable `SenseCom.exe`.



C:\Users\mamadou\Desktop\SenseCom-main\Win\SenseCom_Win_Latest

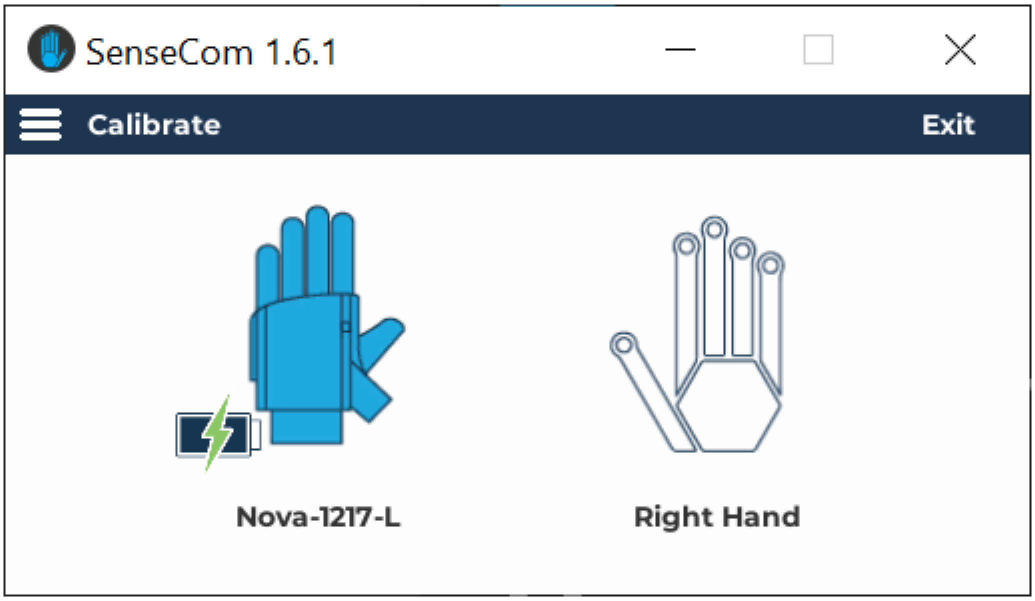
SenseCom-main > Win > SenseCom_Win_Latest

Name	Date modified	Type	Size
MonoBleedingEdge	10/04/2024 11:24	File folder	
SenseCom_Data	10/04/2024 11:24	File folder	
SenseCom.exe	10/04/2024 11:24	Application	639 KB
UnityCrashHandler64.exe	10/04/2024 11:24	Application	1.099 KB
UnityPlayer.dll	10/04/2024 11:24	Application extension	28.786 KB

Note

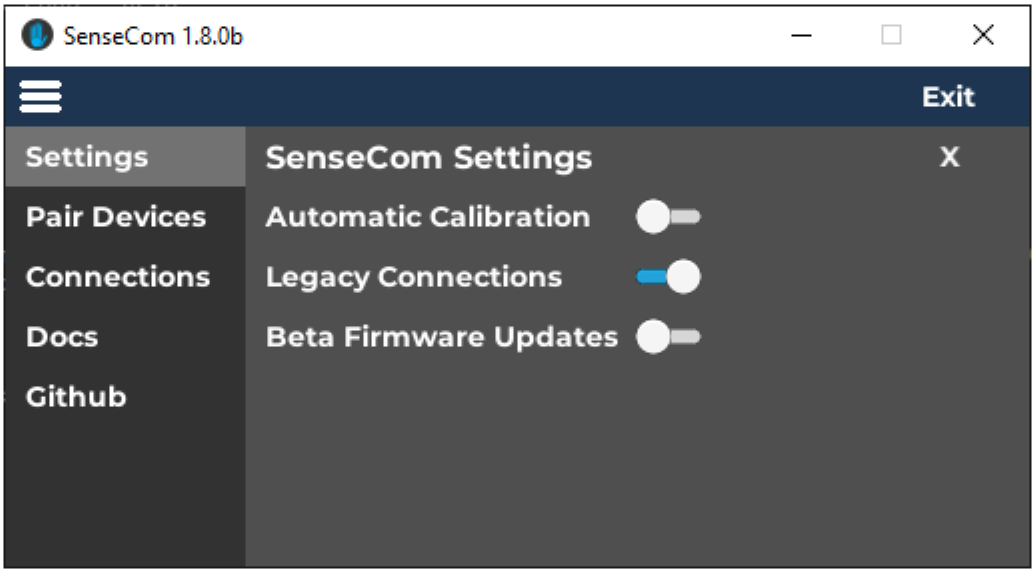
Inside the `/path/to/extracted/SenseCom/directory/Win/` folder, a SenseCom installer is available if you wish to permanently install it on your operating system.

13. In a moment, SenseCom should recognize and connect to your glove(s):



Important

If you are using SenseCom `v1.8.x+` and it fails to recognize your gloves with a Bluetooth Serial firmware, it may be because the `Legacy Connections` option is disabled, which is the default. Enabling this option should allow SenseCom to discover and connect to your glove.



Automatic Calibration: off
Legacy Connections: on
Beta Firmware Updates: off

Note

For more detailed information and troubleshooting, consult the SenseCom documentation page on SGDocs, please.

14. At this stage, SenseCom is ready and you should be able to connect to and communicate with SenseGlove devices from inside your Unreal Engine applications.

Enabling XR_EXT_hand_tracking OpenXR Extension on VR Headsets

> **Important**
>
> Starting from version `v2.1.0`, the SenseGlove Unreal Engine Plugin requires the `XR_EXT_hand_tracking` OpenXR extension to function. Without this OpenXR extension the plugin won't output any glove data.

Since version `v2.1.0`, the SenseGlove Unreal Engine Plugin requires the `XR_EXT_hand_tracking` OpenXR extension to function. Whether you are streaming your immersive 3D VR application from your PC to your VR headset, or deploying it to your VR headset in standalone mode, enabling `XR_EXT_hand_tracking` support, might require additional plugins or settings depending on the HMD's vendor or model.

PCVR Mode

For instructions on how to setup `XR_EXT_hand_tracking` support in PCVR mode please refer to the relevant section.

Standalone Mode

For instructions on how to setup `XR_EXT_hand_tracking` support in standalone mode on Android please refer to the relevant section.

Third-Party Tutorials

As a part of this OpenXR comprehensive tutorial series, you will learn how to enable the developer runtime features, set up the OpenXR runtime, and the `XR_EXT_hand_tracking` support in PCVR mode. Furthermore, it will show you how to enable hand-tracking on Android (standalone mode) using the Meta XR and VIVE OpenXR plugins.

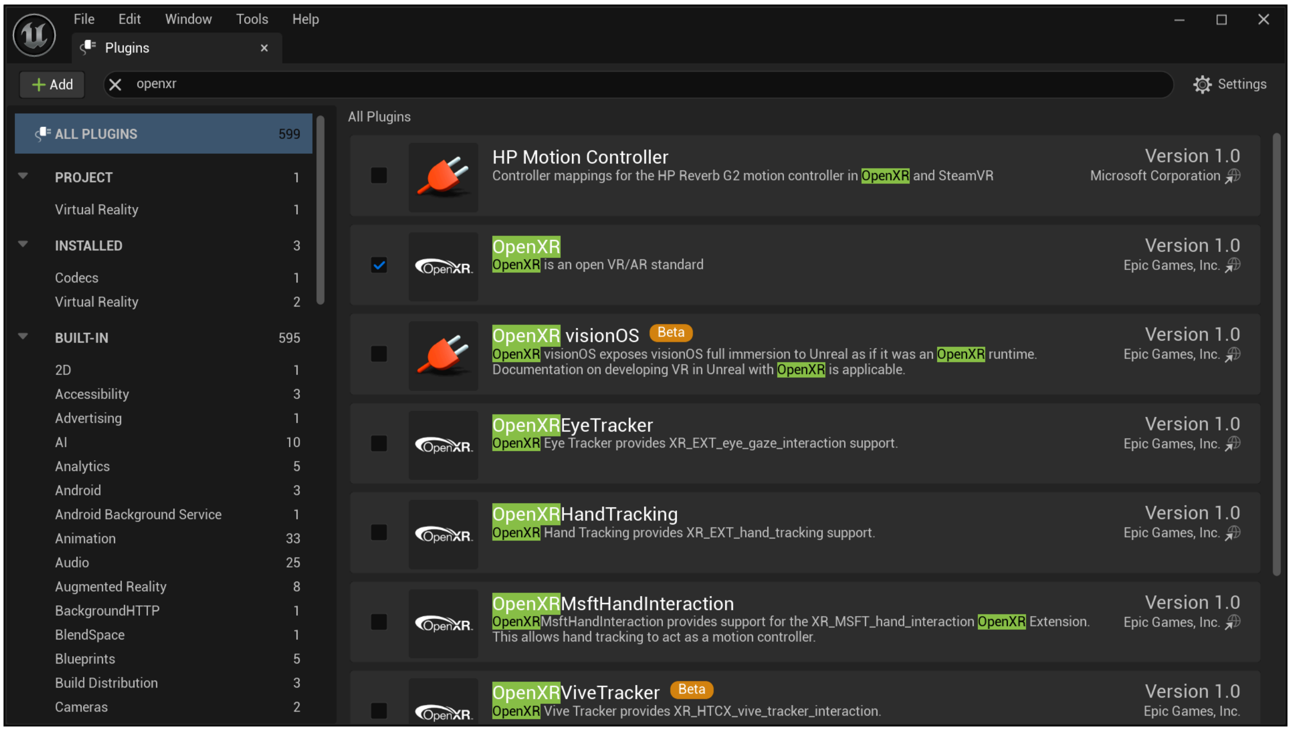
Enabling XR_EXT_hand_tracking OpenXR Extension on VR Headsets in PCVR Mode

Starting from version `v2.1.0`, the SenseGlove Unreal Engine Plugin requires the `XR_EXT_hand_tracking` OpenXR extension to function. If you are streaming your immersive 3D VR application from your PC to your VR headset, enabling `XR_EXT_hand_tracking` support, requires additional plugins and settings depending on the HMD's vendor or model.

Enabling OpenXR Plugin and Disabling OpenXRHandTracking Plugin

Regardless of the type or vendor of the HMD you have in mind for development or deployment purposes, the `OpenXR` plugin is required as a prerequisite. Also, ensure the `OpenXRHandTracking` is disabled as it conflicts with the SenseGlove Unreal Engine Plugin since both implement the same `XR_EXT_hand_tracking` OpenXR extension.

Though enabling the SenseGlove Unreal Engine Plugin should enable the `OpenXR` plugin automatically, it is recommended to ensure this plugin is enabled, and most importantly `OpenXRHandTracking` is disabled, by navigating to `Edit > Plugins` in the Unreal Editor menus.



HP Motion Controller — Controller mappings for the HP Reverb G2 motion controller in OpenXR and SteamVR — Version 1.0, Microsoft Corporation

OpenXR (enabled) — OpenXR is an open VR/AR standard — Version 1.0, Epic Games, Inc.

OpenXR visionOS (Beta) — OpenXR visionOS exposes visionOS full immersion to Unreal as if it was an OpenXR runtime. Documentation on developing VR in Unreal with OpenXR is applicable. — Version 1.0, Epic Games, Inc.

OpenXREyeTracker — OpenXR Eye Tracker provides XR_EXT_eye_gaze_interaction support. — Version 1.0, Epic Games, Inc.

OpenXRHandTracking — OpenXR Hand Tracking provides XR_EXT_hand_tracking support. — Version 1.0, Epic Games, Inc.

OpenXRMsftHandInteraction — OpenXRMsftHandInteraction provides support for the XR_MSFT_hand_interaction OpenXR Extension. This allows hand tracking to act as a motion controller. — Version 1.0, Epic Games, Inc.

OpenXRViveTracker (Beta) — OpenXR Vive Tracker provides XR_HTCX_vive_tracker_interaction. — Version 1.0, Epic Games, Inc.

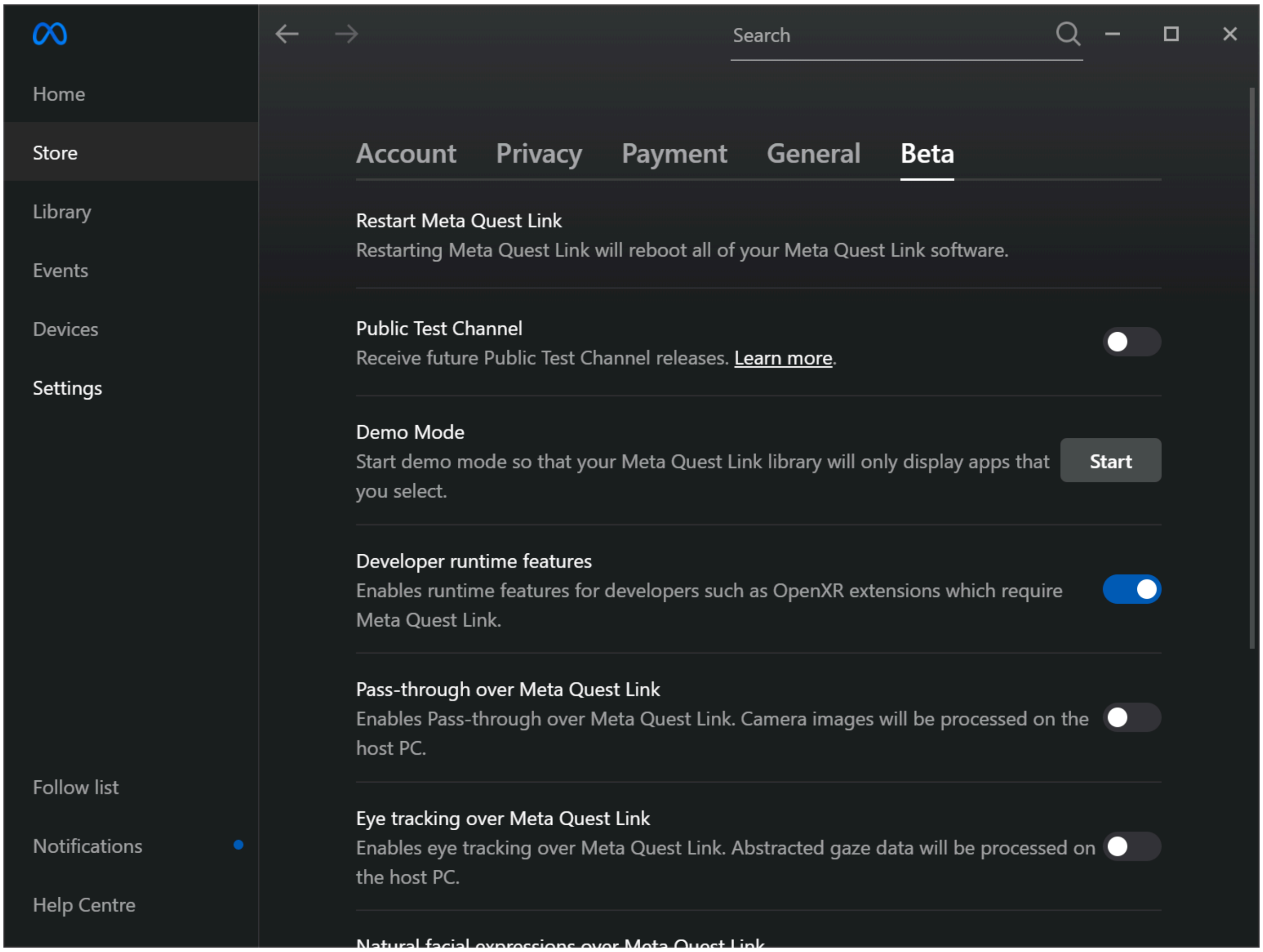
The `OpenXRHandTracking` plugin implements the `XR_EXT_hand_tracking` OpenXR extension.

Meta Quest

To set up `XR_EXT_hand_tracking` support on Meta Quest HMDs in PCVR mode, depending on your project requirements (e.g. whether you rely on the `Meta XR` plugin or not), additional setup steps are required.

Meta Quest Link App

For Meta Quest headsets, enable the `Developer runtime features` under the `Settings > Beta` section inside the Meta Quest Link app:



Restart Meta Quest Link
Restarting Meta Quest Link will reboot all of your Meta Quest Link software.

Public Test Channel
Receive future Public Test Channel releases. Learn more.

Demo Mode
Start demo mode so that your Meta Quest Link library will only display apps that you select.

Developer runtime features
Enables runtime features for developers such as OpenXR extensions which require Meta Quest Link.

Pass-through over Meta Quest Link
Enables Pass-through over Meta Quest Link. Camera images will be processed on the host PC.

Eye tracking over Meta Quest Link
Enables eye tracking over Meta Quest Link. Abstracted gaze data will be processed on the host PC.

Natural facial expressions over Meta Quest Link

Important

Enabling `Developer runtime features` in the Meta Quest Link requires a Meta Developer Account. If you are not signed in using a Meta Developer Account, this option won't be shown to you inside Meta Quest Link.

Caution

Streaming to Meta Quest headsets from SteamVR is no longer supported because the migration to OpenXR has caused controller offsets for Meta Quest HMDs to break on SteamVR. One possible reason is that SteamVR lists `XR_FB_hand_tracking` as an unsupported feature. Further investigation is needed to identify the exact underlying cause.

Meta XR Plugin

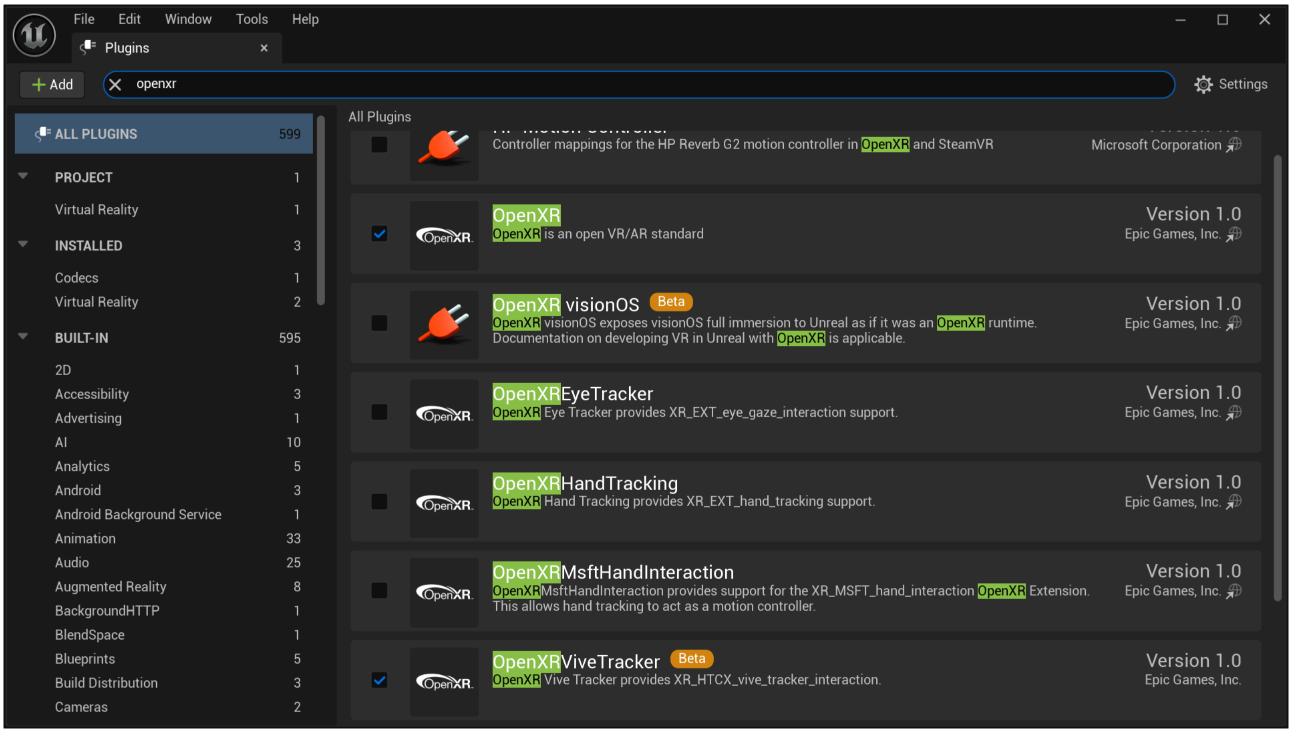
> **Caution**
>
> Please note that enabling the `Meta XR` plugin alongside the SenseGlove plugin will result in crashes or unexpected behavior. `Meta XR` plugin compatibility is being worked on and might be supported in the future.

HTC VIVE

To set up `XR_EXT_hand_tracking` support on HTC VIVE HMDs in PCVR mode, additional plugins or configuration steps are required.

OpenXRViveTracker Plugin

To enable VIVE Trackers support ensure the `OpenXRViveTracker` plugin is enabled by navigating to `Edit > Plugins` in the Unreal Editor menus. This plugin should be enabled, or wrist tracking won't function on VIVE devices at all.


File Edit Window Tools Help

Plugins

Add openxr Settings

ALL PLUGINS	599
PROJECT	1
Virtual Reality	1
INSTALLED	3
Codecs	1
Virtual Reality	2
BUILT-IN	595
2D	1
Accessibility	3
Advertising	1
AI	10
Analytics	5
Android	3
Android Background Service	1
Animation	33
Audio	25
Augmented Reality	8
BackgroundHTTP	1
BlendSpace	1
Blueprints	5
Build Distribution	3
Cameras	2

All Plugins

Controller mappings for the HP Reverb G2 motion controller in OpenXR and SteamVR — Microsoft Corporation

[✓] OpenXR — Version 1.0 — Epic Games, Inc.
OpenXR is an open VR/AR standard

[ ] OpenXR visionOS Beta — Version 1.0 — Epic Games, Inc.
OpenXR visionOS exposes visionOS full immersion to Unreal as if it was an OpenXR runtime. Documentation on developing VR in Unreal with OpenXR is applicable.

[ ] OpenXREyeTracker — Version 1.0 — Epic Games, Inc.
OpenXR Eye Tracker provides XR_EXT_eye_gaze_interaction support.

[ ] OpenXRHandTracking — Version 1.0 — Epic Games, Inc.
OpenXR Hand Tracking provides XR_EXT_hand_tracking support.

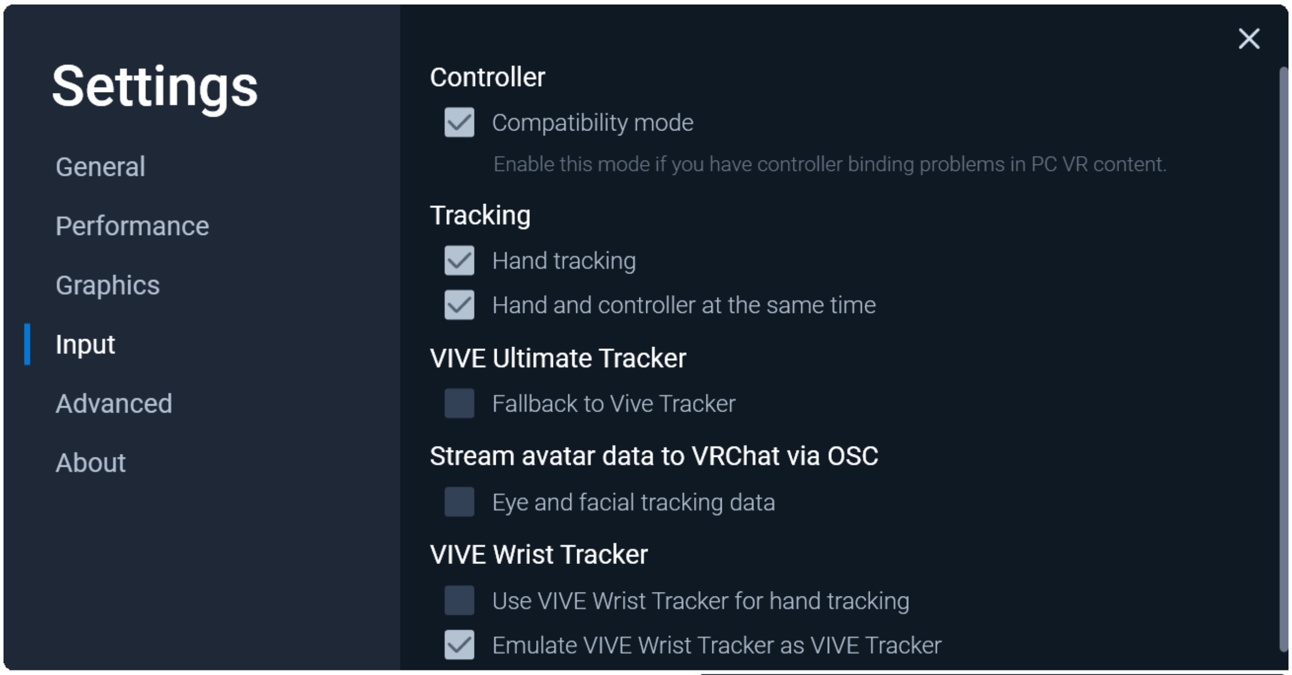
[ ] OpenXRMsftHandInteraction — Version 1.0 — Epic Games, Inc.
OpenXRMsftHandInteraction provides support for the XR_MSFT_hand_interaction OpenXR Extension. This allows hand tracking to act as a motion controller.

[✓] OpenXRViveTracker Beta — Version 1.0 — Epic Games, Inc.
OpenXR Vive Tracker provides XR_HTCX_vive_tracker_interaction.

The `OpenXRViveTracker` plugin implements the `XR_HTCX_vive_tracker_interaction` OpenXR extension which is necessary to use VIVE Trackers or to emulate the VIVE Wrist Trackers as VIVE Trackers on Windows.

VIVE Business Streaming App

For VIVE headsets relying on the VIVE Business Streaming application, ensure the Hand Tracking settings under `Input` are enabled and `XR_HTCX_vive_tracker_interaction` is enabled for VIVE Wrist Trackers by enabling `Emulate VIVE Wrist Tracker as VIVE Tracker`:



Controller

- ✓ Compatibility mode

  Enable this mode if you have controller binding problems in PC VR content.

Tracking

- ✓ Hand tracking
- ✓ Hand and controller at the same time

VIVE Ultimate Tracker

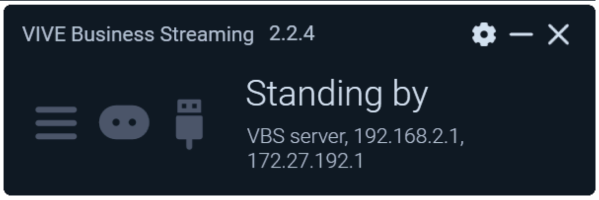
- ☐ Fallback to Vive Tracker

Stream avatar data to VRChat via OSC

- ☐ Eye and facial tracking data

VIVE Wrist Tracker

- ☐ Use VIVE Wrist Tracker for hand tracking
- ✓ Emulate VIVE Wrist Tracker as VIVE Tracker


VIVE Business Streaming 2.2.4

Standing by

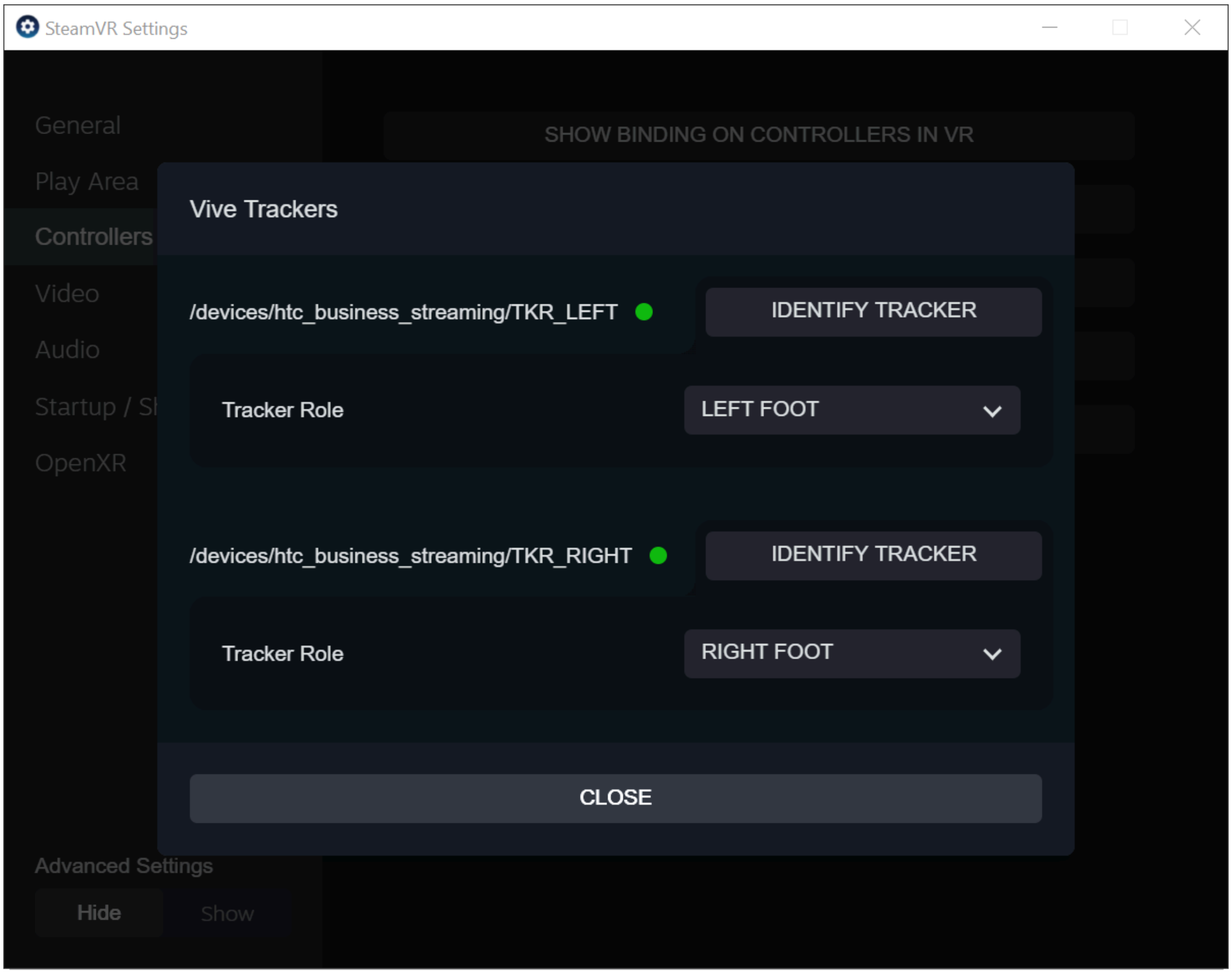
VBS server, 192.168.2.1, 172.27.192.1

Note

Tracking and accessing `FXRHandTrackingState` output from SenseGlove devices do not require Hand and Body Tracking to be enabled on the HMD device. Enabling this feature is only necessary if you wish to use hand-tracking as a fallback option when no glove is connected to your PC.

SteamVR App

After enabling the OpenXRViveTracker plugin and enabling `Emulate VIVE Wrist Tracker as VIVE Tracker` for VIVE HMDs relying on the VIVE Business Streaming, you need to perform one final setup in the SteamVR app for the SenseGlove Unreal Engine Plugin to be able to retrieve the correct wrist-tracking offsets. Once you have paired your VIVE Trackers or VIVE Wrist Trackers, navigate to `SteamVR Settings > Controllers > MANAGE TRACKERS` and make sure your left tracker is set to `LEFT FOOT` and the right tracker is set to `RIGHT FOOT`:



SteamVR Settings

General
Play Area
Controllers
Video
Audio
Startup / Sh
OpenXR
Advanced Settings
Hide Show

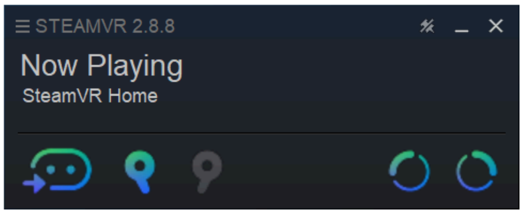
SHOW BINDING ON CONTROLLERS IN VR

Vive Trackers

/devices/htc_business_streaming/TKR_LEFT ● IDENTIFY TRACKER
Tracker Role: LEFT FOOT

/devices/htc_business_streaming/TKR_RIGHT ● IDENTIFY TRACKER
Tracker Role: RIGHT FOOT

CLOSE



STEAMVR 2.8.8
Now Playing
SteamVR Home

SenseGlove Wrist Tracking Settings

Once you have set up everything, it's time to adjust the SenseGlove Wrist Tracking Settings inside the project-wide plugin's settings. For detailed information, please visit the Wrist Tracking Hardware and HMD auto-detection configuration section as well.

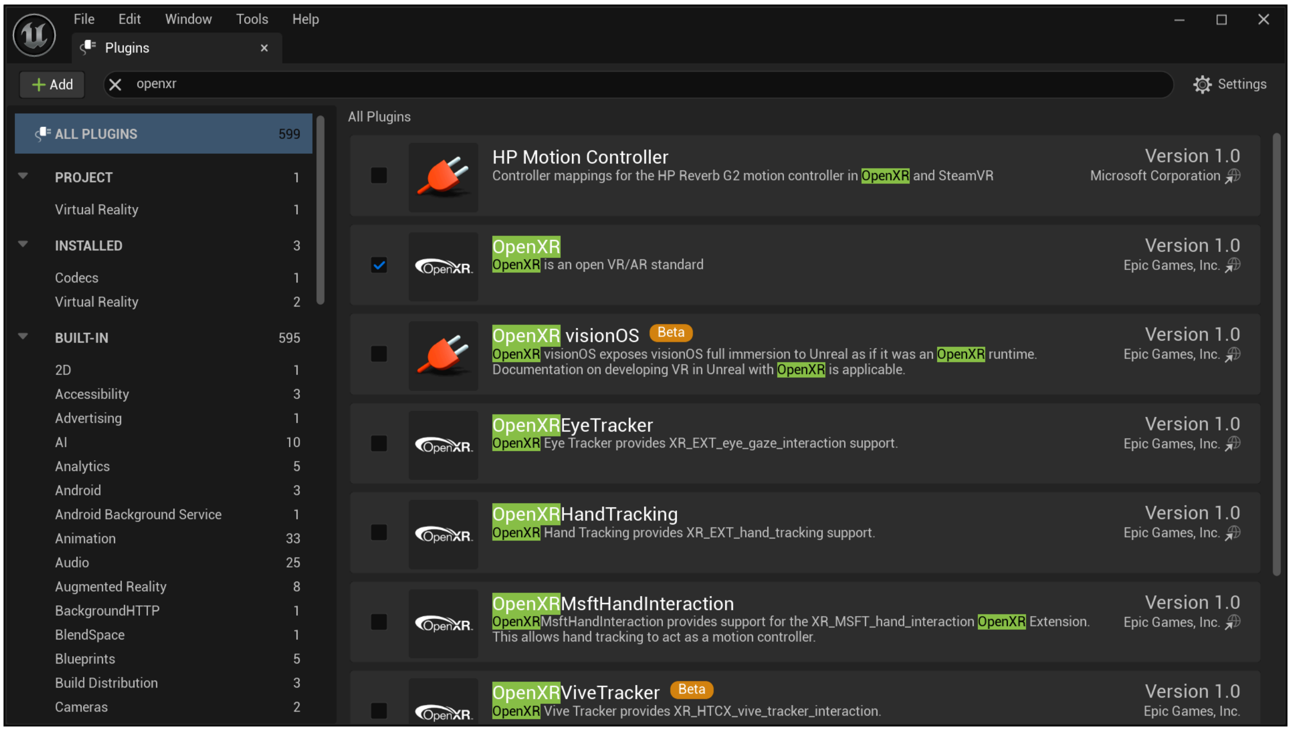
Enabling XR_EXT_hand_tracking OpenXR Extension on VR Headsets in Standalone Mode

Starting from version `v2.1.0`, the SenseGlove Unreal Engine Plugin requires the `XR_EXT_hand_tracking` OpenXR extension to function. If you are deploying your immersive 3D VR application to your VR headset in standalone mode, enabling `XR_EXT_hand_tracking` support, requires additional plugins and settings depending on the HMD's vendor or model.

Enabling OpenXR Plugin and Disabling OpenXRHandTracking Plugin

Regardless of the type or vendor of the HMD you have in mind for development or deployment purposes, the `OpenXR` plugin is required as a prerequisite. Also, ensure the `OpenXRHandTracking` is disabled as it conflicts with the SenseGlove Unreal Engine Plugin since both implement the same `XR_EXT_hand_tracking` OpenXR extension.

Though enabling the SenseGlove Unreal Engine Plugin should enable the `OpenXR` plugin automatically, it is recommended to ensure this plugin is enabled, and most importantly `OpenXRHandTracking` is disabled, by navigating to `Edit > Plugins` in the Unreal Editor menus.



ALL PLUGINS	599
PROJECT	1
Virtual Reality	1
INSTALLED	3
Codecs	1
Virtual Reality	2
BUILT-IN	595
2D	1
Accessibility	3
Advertising	1
AI	10
Analytics	5
Android	3
Android Background Service	1
Animation	33
Audio	25
Augmented Reality	8
BackgroundHTTP	1
BlendSpace	1
Blueprints	5
Build Distribution	3
Cameras	2

All Plugins

Enabled	Plugin	Version
☐	**HP Motion Controller** Controller mappings for the HP Reverb G2 motion controller in OpenXR and SteamVR	Version 1.0 Microsoft Corporation
✓	**OpenXR** OpenXR is an open VR/AR standard	Version 1.0 Epic Games, Inc.
☐	**OpenXR visionOS** Beta OpenXR visionOS exposes visionOS full immersion to Unreal as if it was an OpenXR runtime. Documentation on developing VR in Unreal with OpenXR is applicable.	Version 1.0 Epic Games, Inc.
☐	**OpenXREyeTracker** OpenXR Eye Tracker provides XR_EXT_eye_gaze_interaction support.	Version 1.0 Epic Games, Inc.
☐	**OpenXRHandTracking** OpenXR Hand Tracking provides XR_EXT_hand_tracking support.	Version 1.0 Epic Games, Inc.
☐	**OpenXRMsftHandInteraction** OpenXRMsftHandInteraction provides support for the XR_MSFT_hand_interaction OpenXR Extension. This allows hand tracking to act as a motion controller.	Version 1.0 Epic Games, Inc.
☐	**OpenXRViveTracker** Beta OpenXR Vive Tracker provides XR_HTCX_vive_tracker_interaction.	Version 1.0 Epic Games, Inc.

The `OpenXRHandTracking` plugin implements the `XR_EXT_hand_tracking` OpenXR extension.

Meta Quest

To set up `XR_EXT_hand_tracking` support on Meta Quest HMDs in Standalone mode, depending on your project requirements (e.g. whether you rely on the `Meta XR` plugin or not), additional setup steps are required.

> **Important**
>
> Although, the SenseGlove plugin does not require the `Meta XR` plugin to function, and relying solely on the `OpenXR` and `OpenXRHandTracking` plugins would suffice for functional glove data retrieval using OpenXR, hand-tracking as a fallback mechanism won't work on Android without the `Meta XR` plugin availability.

Meta XR Plugin

> **Caution**
>
> Please note that enabling the `Meta XR` plugin alongside the SenseGlove plugin will result in crashes or unexpected behavior. `Meta XR` plugin compatibility is being worked on and might be supported in the future.

HTC VIVE

To set up `XR_EXT_hand_tracking` support on HTC VIVE HMDs in Standalone mode, additional plugins or configuration steps are required.

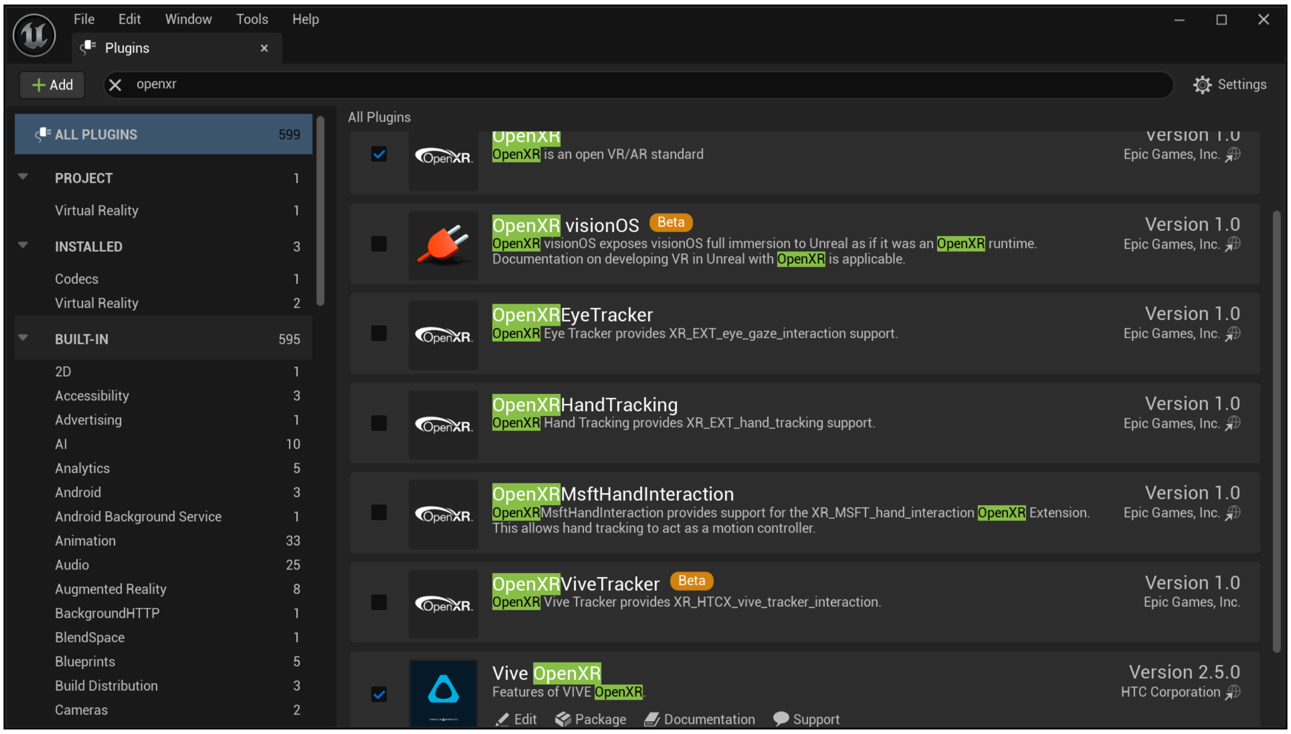
> **Caution**
>
> The SenseGlove Unreal Engine Plugin `v2.7.x` is the last release series to support Unreal Engine `5.4`, and its support will be removed in future minor or major releases. This is important to keep in mind if your target development and deployment platform is HTC VIVE in Standalone Mode. Unfortunately, HTC has not released any updates to their HTC ViveOpenXR plugin since December 6, 2024. Their latest release [1] [2], ViveOpenXR Plugin `v2.5.0`, supports only Unreal Engine `5.3` and `5.4`. HTC VIVE PCVR Mode is unaffected and will remain fully functional because, on Microsoft Windows, it is supported via the OpenXRViveTracker Plugin, which is bundled with Unreal Engine and officially maintained by Epic Games. If you still intend to target HTC in Standalone Mode, you are welcome to continue using the latest SenseGlove Unreal Engine Plugin `v2.7.x`, which will retain HTC Standalone Mode support. However, please keep in mind that once newer versions of the SenseGlove Unreal Engine Plugin are released and UE `5.4` is no longer supported, the latest release of the plugin supporting UE `5.4` will not receive new features, hardware support, or bug fixes. If at any point in the future HTC releases a new version of their ViveOpenXR plugin that supports any Unreal Engine version we actively support,+ in accordance with our support policy and Platform Support Matrix, we will make every reasonable effort to reintroduce HTC Standalone Mode support.

OpenXRViveTracker Plugin

Unlike the PCVR-mode on Windows, the `OpenXRViveTracker` plugin is not required on Android since it only provides an implementation of the `XR_HTCX_vive_tracker_interaction` OpenXR extension which is necessary when we use VIVE Trackers on Windows or we emulate the VIVE Wrist Trackers as VIVE Trackers on Windows. Instead, we require the `XR_HTCX_vive_wrist_tracker_interaction` OpenXR extension to be able to use VIVE Wrist Trackers on Android, which is provided by the `ViveOpenXR` plugin. So, you can safely ignore enabling this plugin for Android standalone deployments.

ViveOpenXR Plugin

To enable VIVE Wrist Trackers support ensure the `ViveOpenXR` plugin is enabled by navigating to `Edit > Plugins` in the Unreal Editor menus. This plugin should be enabled, or wrist tracking won't function on VIVE devices at all.



All Plugins

ALL PLUGINS	599
PROJECT	1
Virtual Reality	1
INSTALLED	3
Codecs	1
Virtual Reality	2
BUILT-IN	595
2D	1
Accessibility	3
Advertising	1
AI	10
Analytics	5
Android	3
Android Background Service	1
Animation	33
Audio	25
Augmented Reality	8
BackgroundHTTP	1
BlendSpace	1
Blueprints	5
Build Distribution	3
Cameras	2

✓ OpenXR — OpenXR is an open VR/AR standard — Version 1.0, Epic Games, Inc.

OpenXR visionOS (Beta) — OpenXR visionOS exposes visionOS full immersion to Unreal as if it was an OpenXR runtime. Documentation on developing VR in Unreal with OpenXR is applicable. — Version 1.0, Epic Games, Inc.

OpenXREyeTracker — OpenXR Eye Tracker provides XR_EXT_eye_gaze_interaction support. — Version 1.0, Epic Games, Inc.

OpenXRHandTracking — OpenXR Hand Tracking provides XR_EXT_hand_tracking support. — Version 1.0, Epic Games, Inc.

OpenXRMsftHandInteraction — OpenXRMsftHandInteraction provides support for the XR_MSFT_hand_interaction OpenXR Extension. This allows hand tracking to act as a motion controller. — Version 1.0, Epic Games, Inc.

OpenXRViveTracker (Beta) — OpenXR Vive Tracker provides XR_HTCX_vive_tracker_interaction. — Version 1.0, Epic Games, Inc.

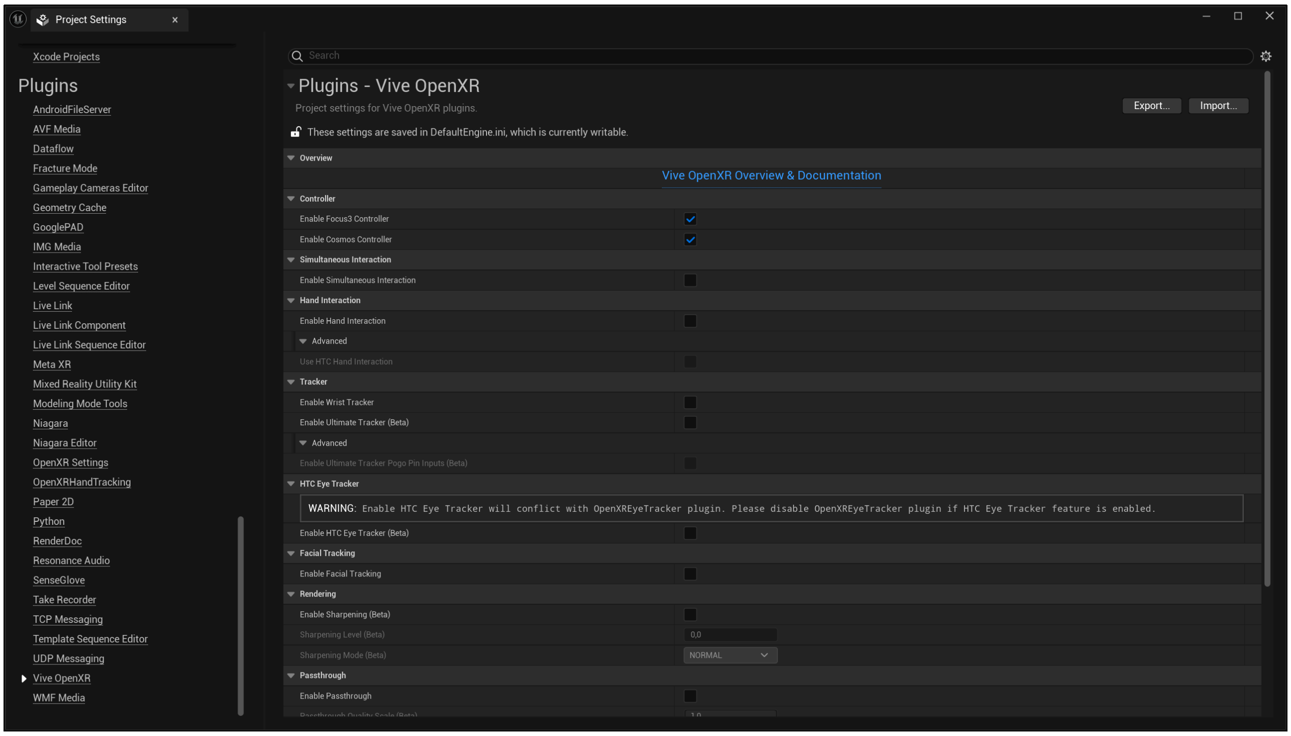
✓ Vive OpenXR — Features of VIVE OpenXR. — Version 2.5.0, HTC Corporation — Edit, Package, Documentation, Support

The `ViveOpenXR` plugin implements the `XR_HTCX_vive_wrist_tracker_interaction` OpenXR extension.

Important

Without the `ViveOpenXR` plugin deploying an immersive 3D VR application to your HTC VIVE in Standalone mode won't be possible, whether you enable `bStartInVR`, or not. Without it your app will be deployed and recognized as a 2D app.

By default, the `ViveOpenXR` plugin settings located in `Edit > Project Settings > Plugins > Vive OpenXR` look something like these:

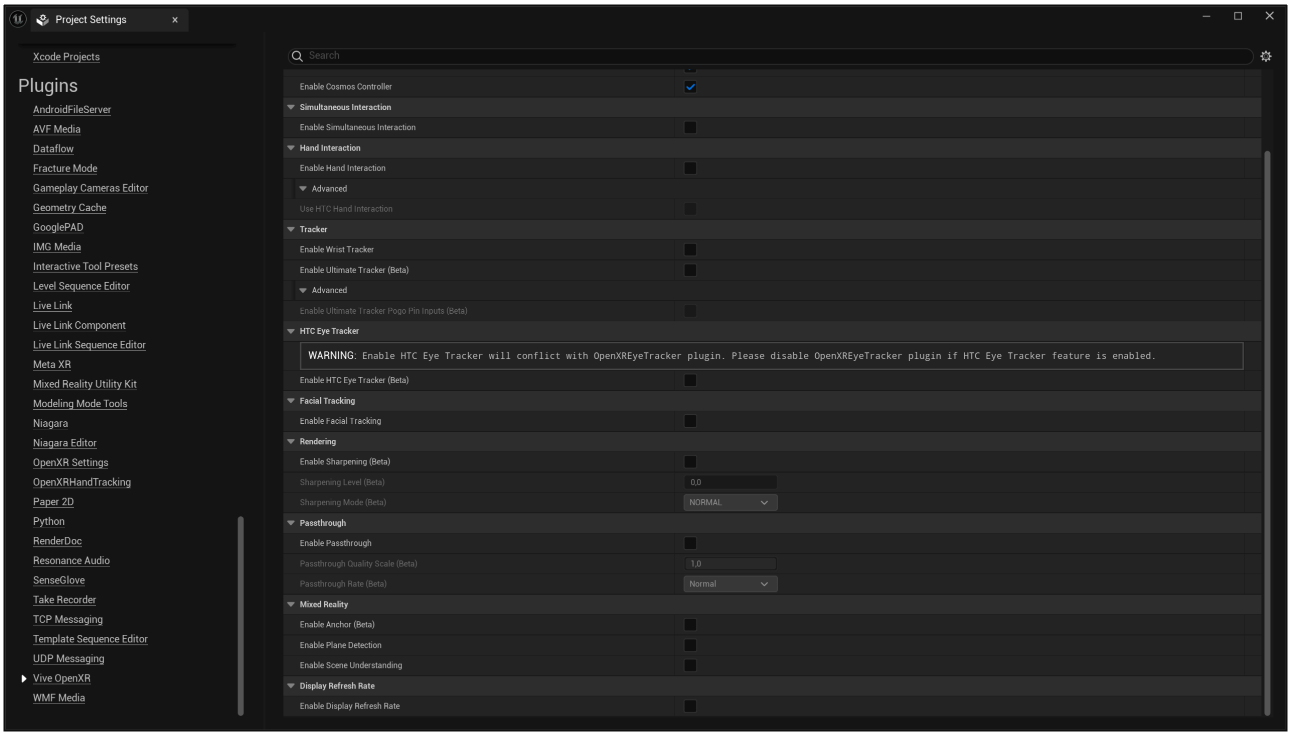


Plugins - Vive OpenXR

Project settings for Vive OpenXR plugins.

These settings are saved in DefaultEngine.ini, which is currently writable.

Overview	
Vive OpenXR Overview & Documentation	
Controller	
Enable Focus3 Controller	✓
Enable Cosmos Controller	✓
Simultaneous Interaction	
Enable Simultaneous Interaction	☐
Hand Interaction	
Enable Hand Interaction	☐
Advanced	
Use HTC Hand Interaction	☐
Tracker	
Enable Wrist Tracker	☐
Enable Ultimate Tracker (Beta)	☐
Advanced	
Enable Ultimate Tracker Pogo Pin Inputs (Beta)	☐
HTC Eye Tracker	
WARNING: `Enable HTC Eye Tracker will conflict with OpenXREyeTracker plugin. Please disable OpenXREyeTracker plugin if HTC Eye Tracker feature is enabled.`	
Enable HTC Eye Tracker (Beta)	☐
Facial Tracking	
Enable Facial Tracking	☐
Rendering	
Enable Sharpening (Beta)	☐
Sharpening Level (Beta)	0,0
Sharpening Mode (Beta)	NORMAL
Passthrough	
Enable Passthrough	☐
Passthrough Quality Scale (Beta)	1.0



Project Settings

Xcode Projects

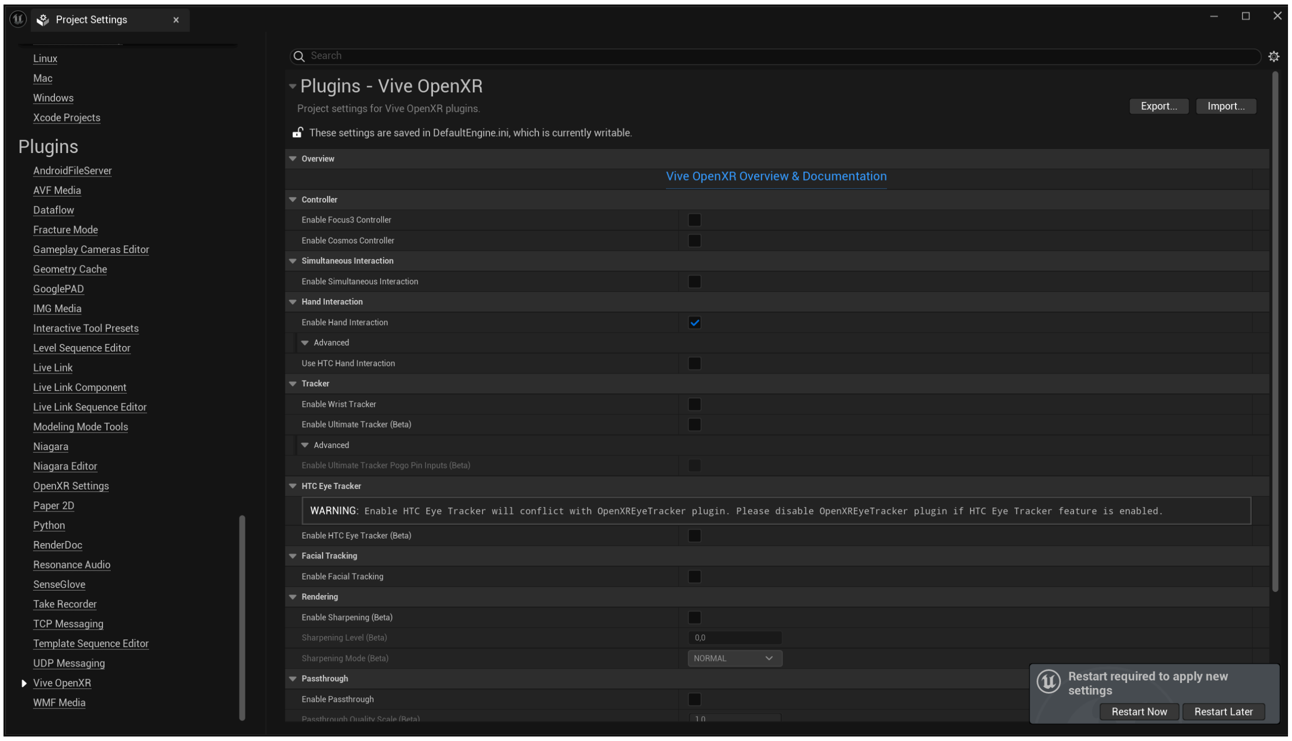
Plugins

AndroidFileServer
AVF Media
Dataflow
Fracture Mode
Gameplay Cameras Editor
Geometry Cache
GooglePAD
IMG Media
Interactive Tool Presets
Level Sequence Editor
Live Link
Live Link Component
Live Link Sequence Editor
Meta XR
Mixed Reality Utility Kit
Modeling Mode Tools
Niagara
Niagara Editor
OpenXR Settings
OpenXRHandTracking
Paper 2D
Python
RenderDoc
Resonance Audio
SenseGlove
Take Recorder
TCP Messaging
Template Sequence Editor
UDP Messaging
Vive OpenXR
WMF Media

Enable Cosmos Controller	✓
Simultaneous Interaction	
Enable Simultaneous Interaction	☐
Hand Interaction	
Enable Hand Interaction	☐
Advanced	
Use HTC Hand Interaction	☐
Tracker	
Enable Wrist Tracker	☐
Enable Ultimate Tracker (Beta)	☐
Advanced	
Enable Ultimate Tracker Pogo Pin Inputs (Beta)	☐
HTC Eye Tracker	
WARNING: Enable HTC Eye Tracker will conflict with OpenXREyeTracker plugin. Please disable OpenXREyeTracker plugin if HTC Eye Tracker feature is enabled.	
Enable HTC Eye Tracker (Beta)	☐
Facial Tracking	
Enable Facial Tracking	☐
Rendering	
Enable Sharpening (Beta)	☐
Sharpening Level (Beta)	0,0
Sharpening Mode (Beta)	NORMAL
Passthrough	
Enable Passthrough	☐
Passthrough Quality Scale (Beta)	1,0
Passthrough Rate (Beta)	Normal
Mixed Reality	
Enable Anchor (Beta)	☐
Enable Plane Detection	☐
Enable Scene Understanding	☐
Display Refresh Rate	
Enable Display Refresh Rate	☐

The following settings control the availability of hand-tracking when the `ViveOpenXR` plugin is enabled:

- **Enable Hand Interaction**: This enables the hand interactions with the OpenXR hand interaction extension `XR_HTC_hand_interaction`. Changing this setting will prompt you to restart the engine to apply the new settings. This setting should be enabled.



Plugins - Vive OpenXR

Project settings for Vive OpenXR plugins.

These settings are saved in DefaultEngine.ini, which is currently writable.

Overview

Vive OpenXR Overview & Documentation

Controller

Enable Focus3 Controller	☐
Enable Cosmos Controller	☐

Simultaneous Interaction

Enable Simultaneous Interaction	☐

Hand Interaction

Enable Hand Interaction	✓

Advanced

Use HTC Hand Interaction	☐

Tracker

Enable Wrist Tracker	☐
Enable Ultimate Tracker (Beta)	☐

Advanced

Enable Ultimate Tracker Pogo Pin Inputs (Beta)	☐

HTC Eye Tracker

WARNING: Enable HTC Eye Tracker will conflict with OpenXREyeTracker plugin. Please disable OpenXREyeTracker plugin if HTC Eye Tracker feature is enabled.

Enable HTC Eye Tracker (Beta)	☐

Facial Tracking

Enable Facial Tracking	☐

Rendering

Enable Sharpening (Beta)	☐
Sharpening Level (Beta)	0,0
Sharpening Mode (Beta)	NORMAL

Passthrough

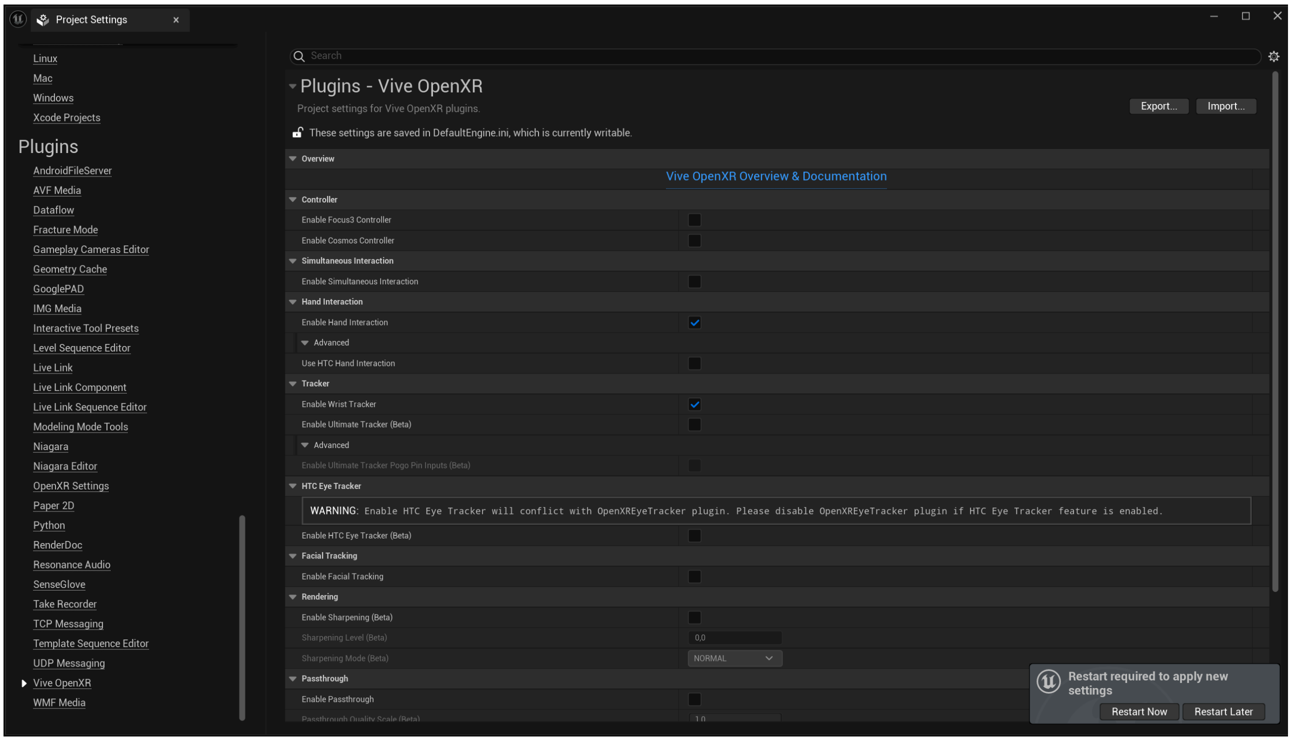
Enable Passthrough	☐
Passthrough Quality Scale (Beta)	1,0

Restart required to apply new settings

- **Use HTC Hand Interaction**: This selects which OpenXR hand interaction extension to use. If enabled, `XR_HTC_hand_interaction` will be used, effectively breaking SenseGlove glove and hand-tracking data output. If disabled, `XR_EXT_hand_interaction` will be used, which is compatible with the SenseGlove Unreal Engine Plugin. Changing this setting will prompt you to restart the engine to apply the new settings. This setting should be disabled.

The following settings control the availability of wrist-tracking when the `ViveOpenXR` plugin is enabled:

- **Enable Wrist Tracker**: This option controls the `XR_HTC_vive_wrist_tracker_interaction` OpenXR extension. If enabled, in turn, it enables the use of HTC Wrist Tracker interaction profiles in OpenXR. Changing this setting will prompt you to restart the engine to apply the new settings.



Plugins - Vive OpenXR

Project settings for Vive OpenXR plugins.

These settings are saved in DefaultEngine.ini, which is currently writable.

Overview

Controller
- Enable Focus3 Controller ☐
- Enable Cosmos Controller ☐

Simultaneous Interaction
- Enable Simultaneous Interaction ☐

Hand Interaction
- Enable Hand Interaction ✓
- Advanced
  - Use HTC Hand Interaction ☐

Tracker
- Enable Wrist Tracker ✓
- Enable Ultimate Tracker (Beta) ☐
- Advanced
  - Enable Ultimate Tracker Pogo Pin Inputs (Beta) ☐

HTC Eye Tracker

WARNING: `Enable HTC Eye Tracker will conflict with OpenXREyeTracker plugin. Please disable OpenXREyeTracker plugin if HTC Eye Tracker feature is enabled.`

- Enable HTC Eye Tracker (Beta) ☐

Facial Tracking
- Enable Facial Tracking ☐

Rendering
- Enable Sharpening (Beta) ☐
- Sharpening Level (Beta) 0,0
- Sharpening Mode (Beta) NORMAL

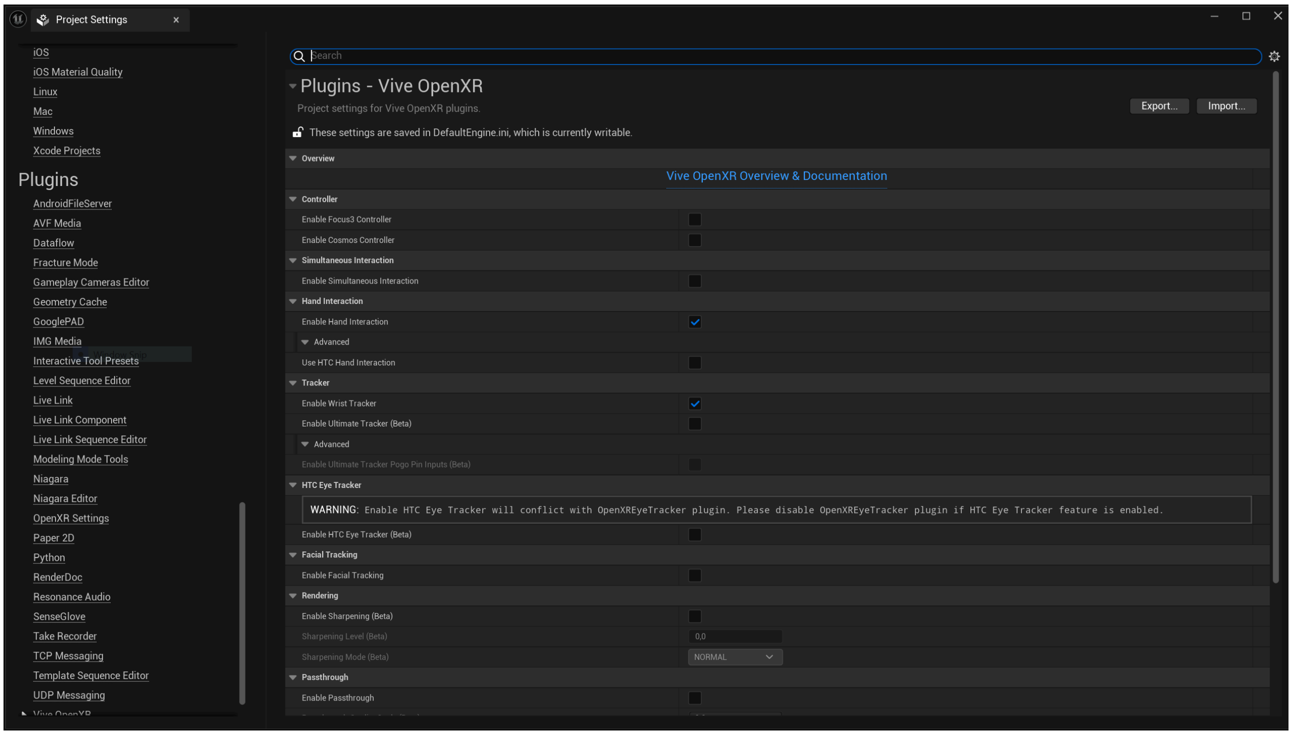
Passthrough
- Enable Passthrough ☐
- Passthrough Quality Scale (Beta) 1.0

Restart required to apply new settings

- **Enable Ultimate Tracker (Beta)**: This option controls the `XR_HTC_path_enumeration` and `XR_HTC_vive_xr_tracker_interaction` OpenXR extensions. If enabled, in turn, they enable the use of HTC Xr Tracker interaction profiles in OpenXR. Changing this setting will prompt you to restart the engine to apply the new settings.

- **Enable Ultimate Tracker Pogo Pin Inputs (Beta)**: Enables or disables the input options for Unreal's Enhanced Input System. Changing this setting will prompt you to restart the engine to apply the new settings.

The following setup demonstrates a functional immersive 3D VR application with the minimum `ViveOpenXR` required features enabled to make it compatible with the SenseGlove Unreal Engine Plugin:



Plugins - Vive OpenXR

Project settings for Vive OpenXR plugins.

These settings are saved in DefaultEngine.ini, which is currently writable.

Overview

Vive OpenXR Overview & Documentation

Controller
- Enable Focus3 Controller ☐
- Enable Cosmos Controller ☐

Simultaneous Interaction
- Enable Simultaneous Interaction ☐

Hand Interaction
- Enable Hand Interaction ✓
- Advanced
- Use HTC Hand Interaction ☐

Tracker
- Enable Wrist Tracker ✓
- Enable Ultimate Tracker (Beta) ☐
- Advanced
- Enable Ultimate Tracker Pogo Pin Inputs (Beta) ☐

HTC Eye Tracker

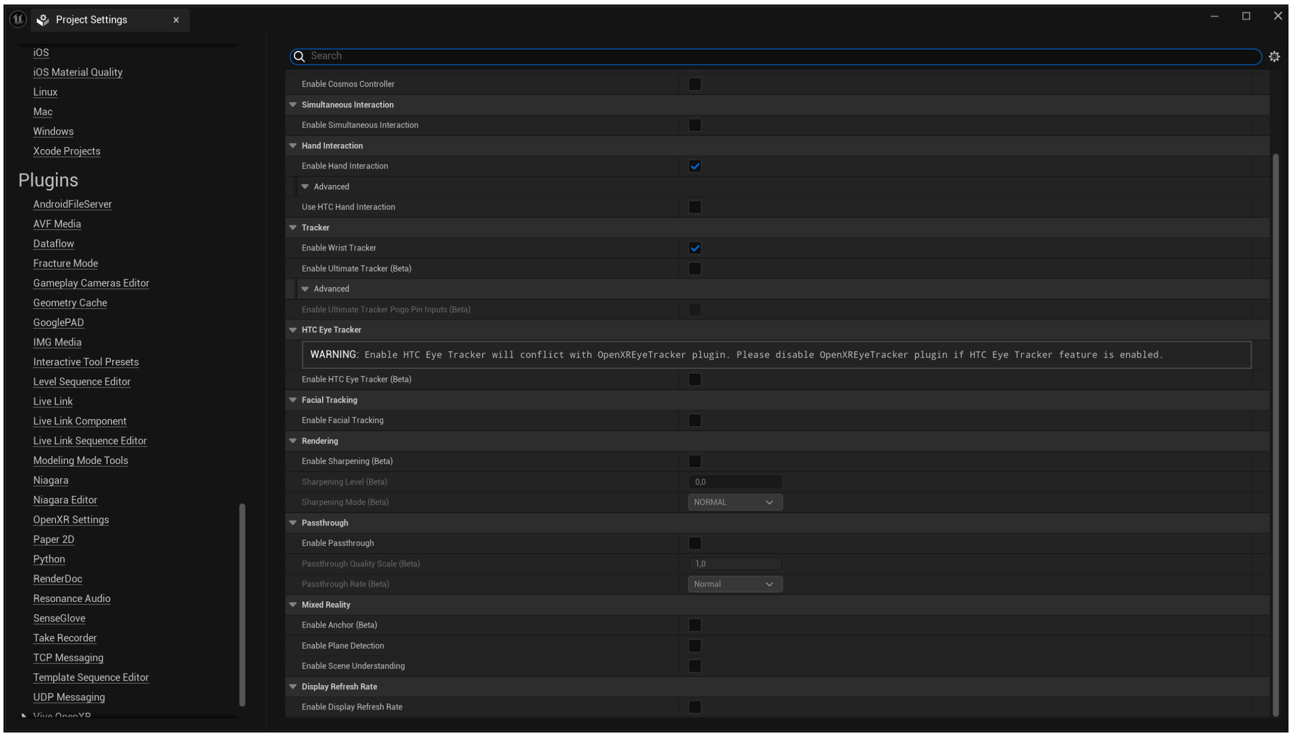
```
WARNING: Enable HTC Eye Tracker will conflict with OpenXREyeTracker plugin. Please disable OpenXREyeTracker plugin if HTC Eye Tracker feature is enabled.
```

- Enable HTC Eye Tracker (Beta) ☐

Facial Tracking
- Enable Facial Tracking ☐

Rendering
- Enable Sharpening (Beta) ☐
- Sharpening Level (Beta) 0,0
- Sharpening Mode (Beta) NORMAL

Passthrough
- Enable Passthrough ☐



- Enable Cosmos Controller ☐

Simultaneous Interaction
- Enable Simultaneous Interaction ☐

Hand Interaction
- Enable Hand Interaction ✓
- Advanced
- Use HTC Hand Interaction ☐

Tracker
- Enable Wrist Tracker ✓
- Enable Ultimate Tracker (Beta) ☐
- Advanced
- Enable Ultimate Tracker Pogo Pin Inputs (Beta) ☐

HTC Eye Tracker

```
WARNING: Enable HTC Eye Tracker will conflict with OpenXREyeTracker plugin. Please disable OpenXREyeTracker plugin if HTC Eye Tracker feature is enabled.
```

- Enable HTC Eye Tracker (Beta) ☐

Facial Tracking
- Enable Facial Tracking ☐

Rendering
- Enable Sharpening (Beta) ☐
- Sharpening Level (Beta) 0,0
- Sharpening Mode (Beta) NORMAL

Passthrough
- Enable Passthrough ☐
- Passthrough Quality Scale (Beta) 1,0
- Passthrough Rate (Beta) Normal

Mixed Reality
- Enable Anchor (Beta) ☐
- Enable Plane Detection ☐
- Enable Scene Understanding ☐

Display Refresh Rate
- Enable Display Refresh Rate ☐

SenseGlove Wrist Tracking Settings

Once you have set up everything, it's time to adjust the SenseGlove Wrist Tracking Settings inside the project-wide plugin's settings. For detailed information, please visit the Wrist Tracking Hardware and HMD auto-detection configuration section as well.

Third-Party Tutorials: XR_EXT_hand_tracking Setup

Introduction to Virtual Reality, OpenXR Hand-Tracking, and Gesture Detection in Unreal Engine

A part of this comprehensive tutorial will guide you through setting up the OpenXR runtime and enabling developer runtime features:

Unreal Engine OpenXR Hand-Tracking on Android with Meta XR (Quest 3S/3/Pro/2) and HTC VIVE OpenXR (Focus Vision/XR Elite/Focus 3) Plugins

A part of this comprehensive tutorial will guide you through setting up the Meta XR and VIVE OpenXR plugins for Android standalone-mode deployment:

Setting Up the SenseGlove Default Classes

Setting up the default SenseGlove classes is recommended if you want to take full advantage of the quality-of-life features provided by the SenseGlove Unreal Engine Plugin. These features are designed to streamline the development process within the Unreal Engine environment. For instance, if you need a quick setup with a virtual hand mesh already integrated into a pawn, enabling you to get started with your project in just a few minutes, it is essential to configure the default classes and familiarize yourself with these classes.

If you wish to extend the functionality of these classes, you can do so by subclassing them. The default SenseGlove classes, which are prefixed with `SG`, include:

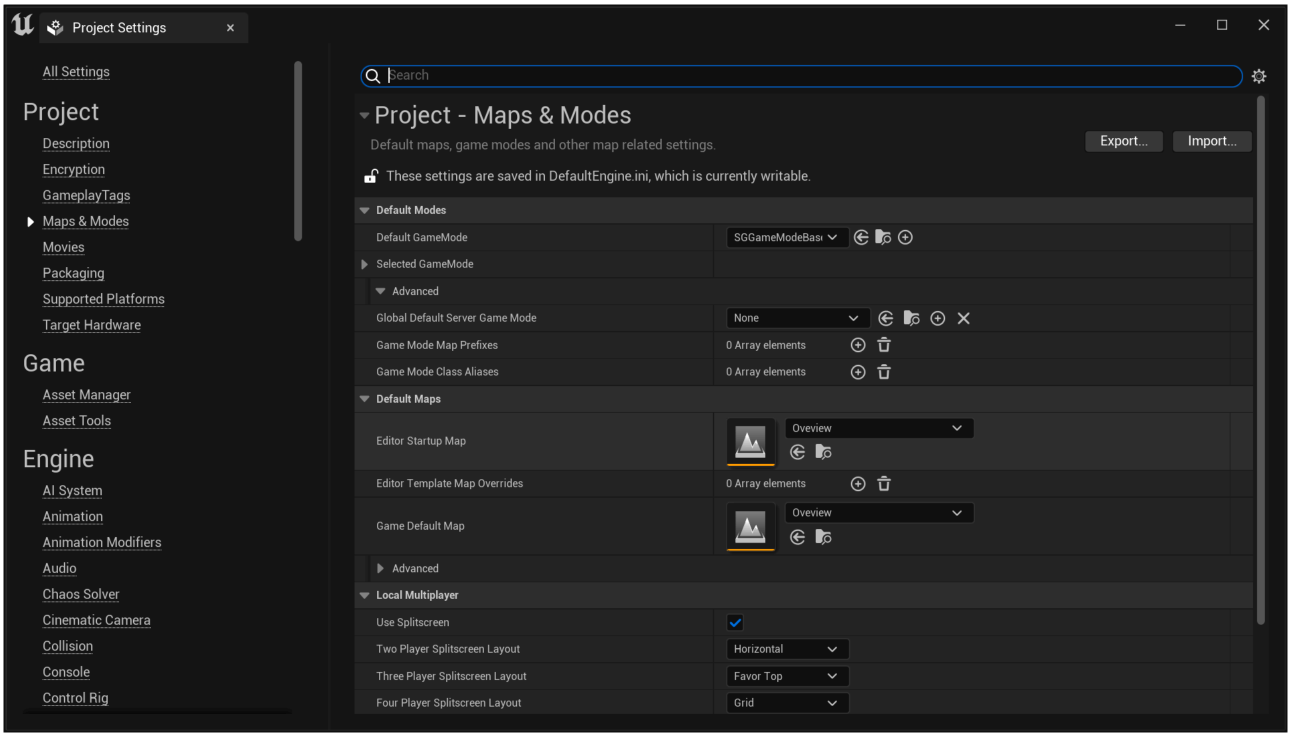
- SGGameModeBase
- SGPawn
- SGPlayerController
- SGGameInstance
- SGGameUserSettings

However, if you prefer a different approach or do not require the functionality provided by the default SenseGlove classes, you can opt to utilize individual components like `SGVirtualHandComponent`, `SGWristTrackerComponent`, etc., directly within your own actors. Alternatively, you can develop a completely custom system from scratch, leveraging the low-level SenseGlove C++ or Blueprint APIs.

Additionally, you can enforce setting the default SenseGlove classes during initialization via the plugin settings, if desired.

Setting Up SGGameModeBase

After installing and enabling the SenseGlove Unreal Engine Plugin, the easiest and most straightforward approach to get started is to just set the default GameMode to `SGGameModeBase` from `Edit > Project Settings... > Maps & Modes > Default Mode > Default GameMode`. By doing this, the `Default Pawn Class` is automatically set to `SGPawn`, and the `Player Controller Class` is set to `SGPlayerController`. This setup ensures that a SenseGlove pawn will automatically spawn when you hit the play button in the editor.



Project - Maps & Modes

Default maps, game modes and other map related settings.

These settings are saved in DefaultEngine.ini, which is currently writable.

Default Modes	
Default GameMode	SGGameModeBas
Selected GameMode	
Advanced	
Global Default Server Game Mode	None
Game Mode Map Prefixes	0 Array elements
Game Mode Class Aliases	0 Array elements
Default Maps	
Editor Startup Map	Overview
Editor Template Map Overrides	0 Array elements
Game Default Map	Overview
Advanced	
Local Multiplayer	
Use Splitscreen	✓
Two Player Splitscreen Layout	Horizontal
Three Player Splitscreen Layout	Favor Top
Four Player Splitscreen Layout	Grid

Tip

For greater control and customization, consider extending the SGGameModeBase.

Note

Currently, setting `SGGameModeBase` or a subclass of it as the `Default GameMode` is not a strict requirement. Its primary function is to ensure that a default `SGPawn`

> and `SGPlayerController` are set. However, this might change in the future, and it could become a mandatory setting.

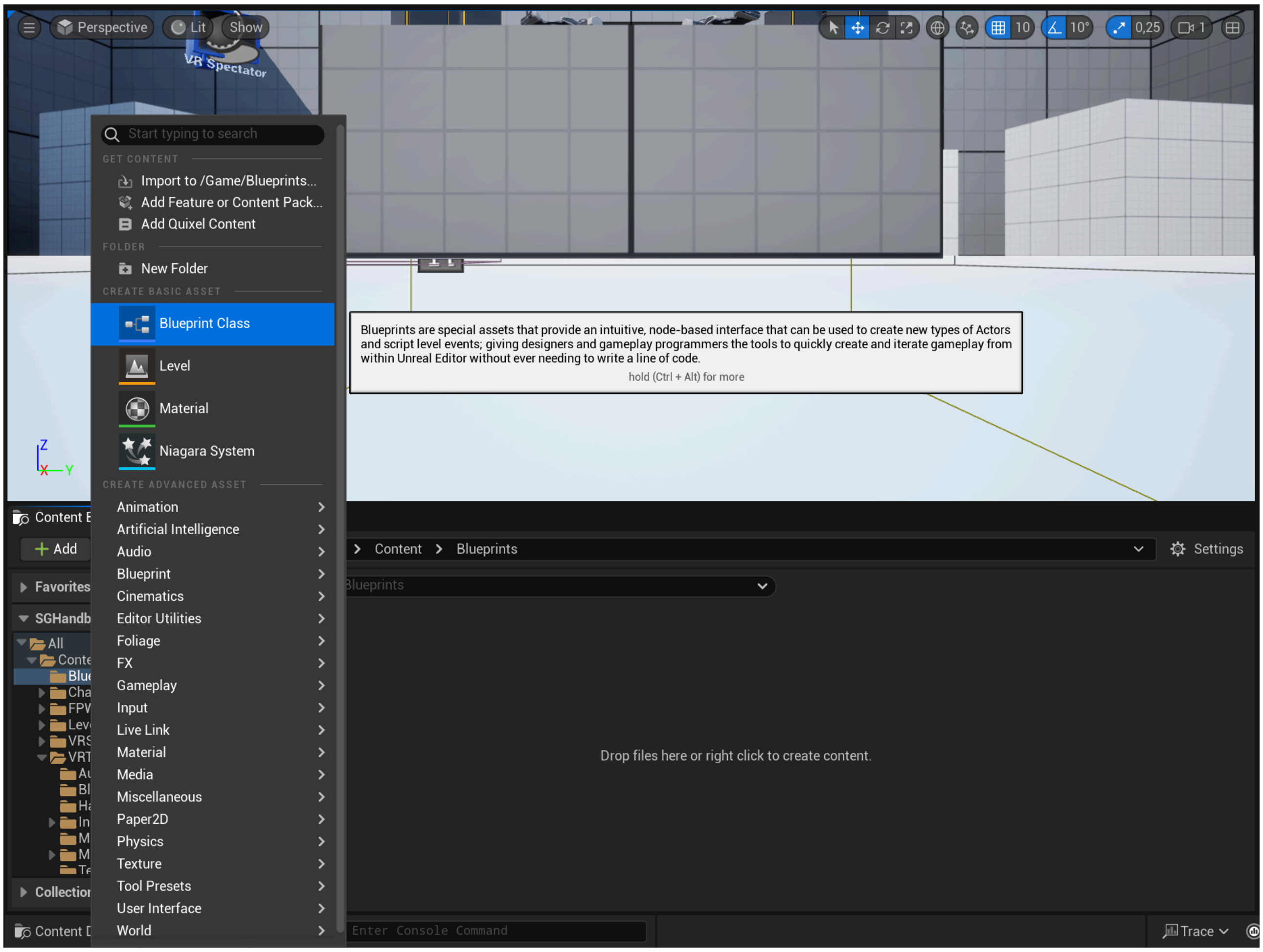
> **Important**
>
> While setting `SGGameModeBase` as the `Default GameMode` will automatically spawn the default SGPawn at BeginPlay and initiate communication with the SenseGlove devices, it will not display any virtual hands in your simulation by default. You might still need to configure the Virtual Hand Meshes and the Wrist Tracking Hardware separately.

> **Important**
>
> Before starting the simulation in the editor, make sure that SenseCom is running and XR_EXT_hand_tracking is enabled. Without these, your simulation will not receive hand pose data from the SenseGlove devices.

Extending SGGameModeBase

Follow these steps to extend and set up your own version of `SGGameModeBase` :

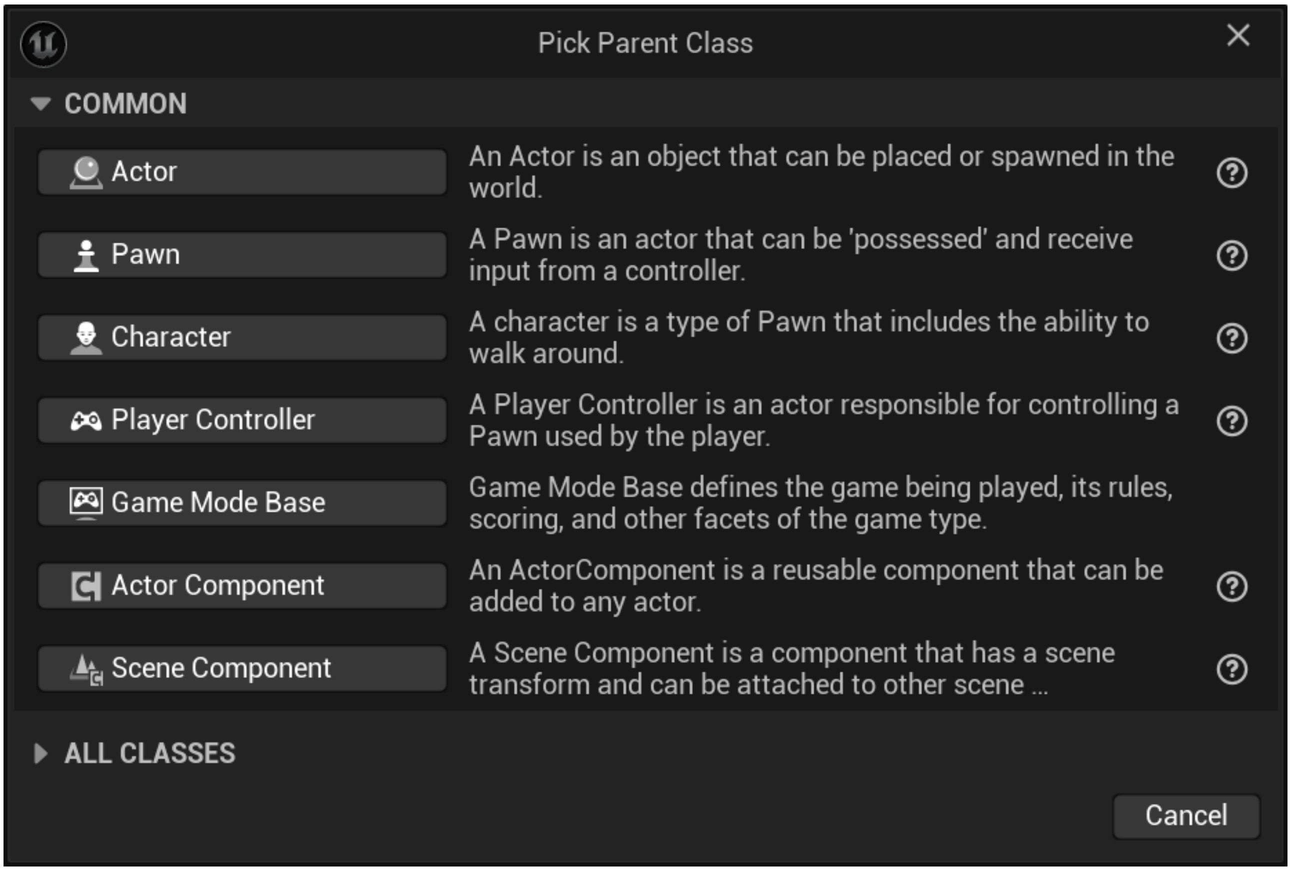
1. In the Content Browser, click the `+ Add` button, then select `Blueprint Class from the menu` . Alternatively, right-click inside the Content Browser and choose `Blueprint Class` from the context menu.



Blueprints are special assets that provide an intuitive, node-based interface that can be used to create new types of Actors and script level events; giving designers and gameplay programmers the tools to quickly create and iterate gameplay from within Unreal Editor without ever needing to write a line of code.
hold (Ctrl + Alt) for more

Drop files here or right click to create content.

2. A dialog will appear asking you to choose a parent class. Click on the ALL CLASSES section to expand the list of available classes.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

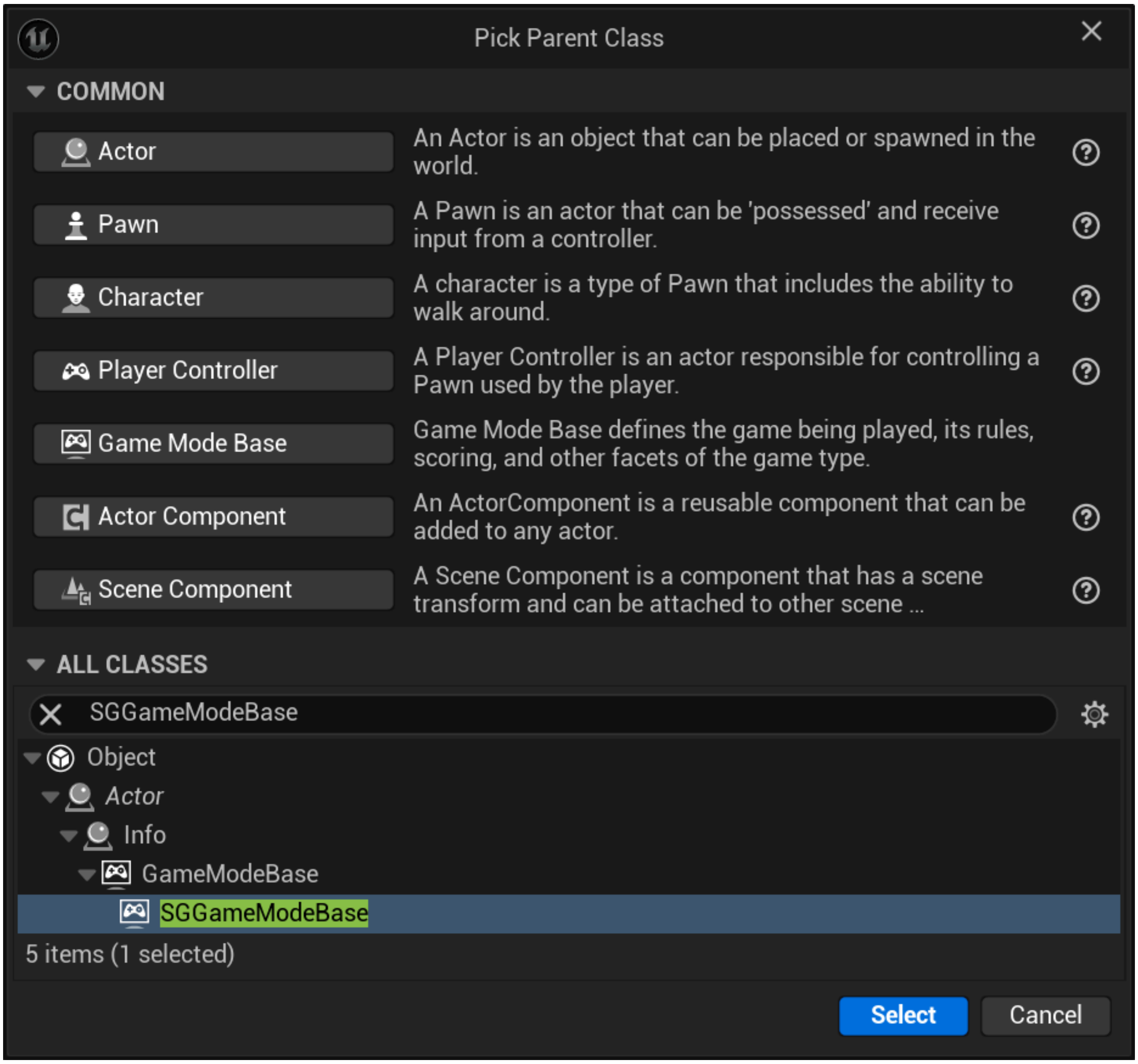
A Player Controller is an actor responsible for controlling a Pawn used by the player.

Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

An ActorComponent is a reusable component that can be added to any actor.

A Scene Component is a component that has a scene transform and can be attached to other scene ...

3. In the expanded `ALL CLASSES` section, start typing `SGGameModeBase` in the Search box. When `SGGameModeBase` appears, select it and click the `Select` button to create your new Blueprint class based on it.


Pick Parent Class

COMMON

Actor	An Actor is an object that can be placed or spawned in the world.
Pawn	A Pawn is an actor that can be 'possessed' and receive input from a controller.
Character	A character is a type of Pawn that includes the ability to walk around.
Player Controller	A Player Controller is an actor responsible for controlling a Pawn used by the player.
Game Mode Base	Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.
Actor Component	An ActorComponent is a reusable component that can be added to any actor.
Scene Component	A Scene Component is a component that has a scene transform and can be attached to other scene ...

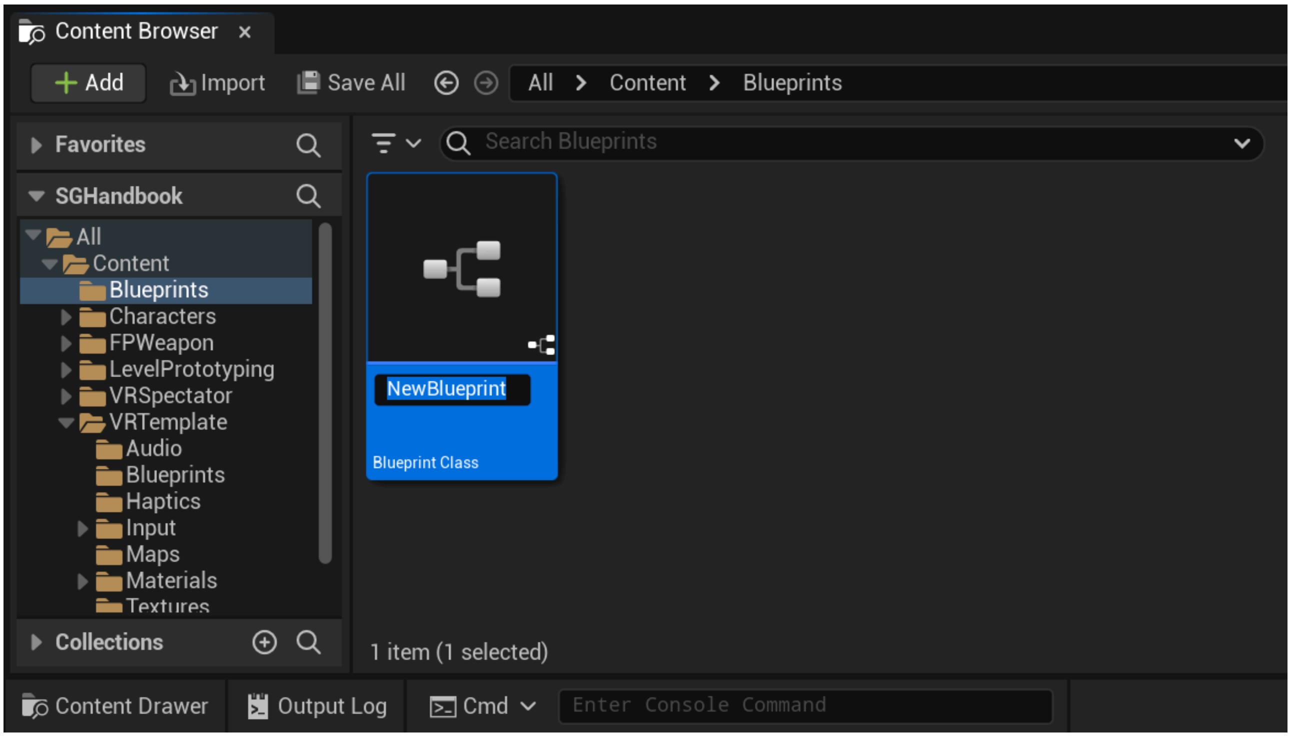
ALL CLASSES

SGGameModeBase

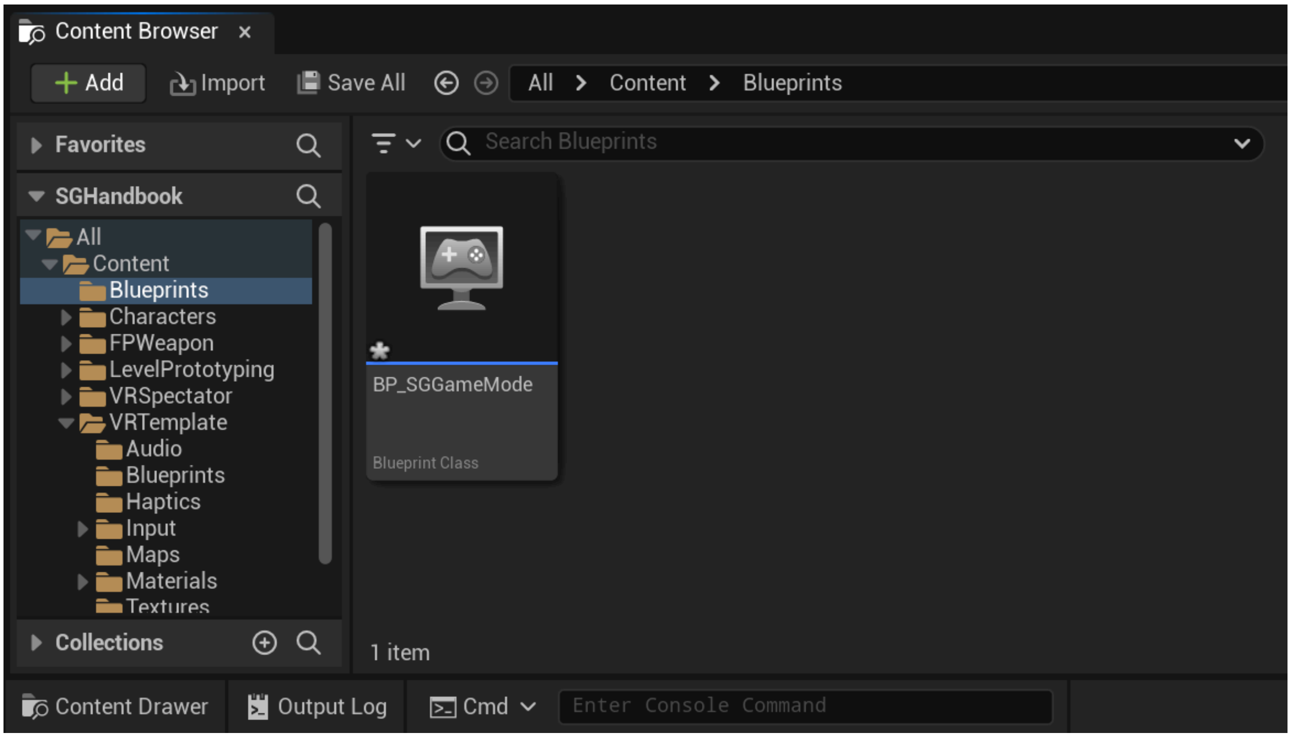
```
Object
  Actor
    Info
      GameModeBase
        SGGameModeBase
```

5 items (1 selected)

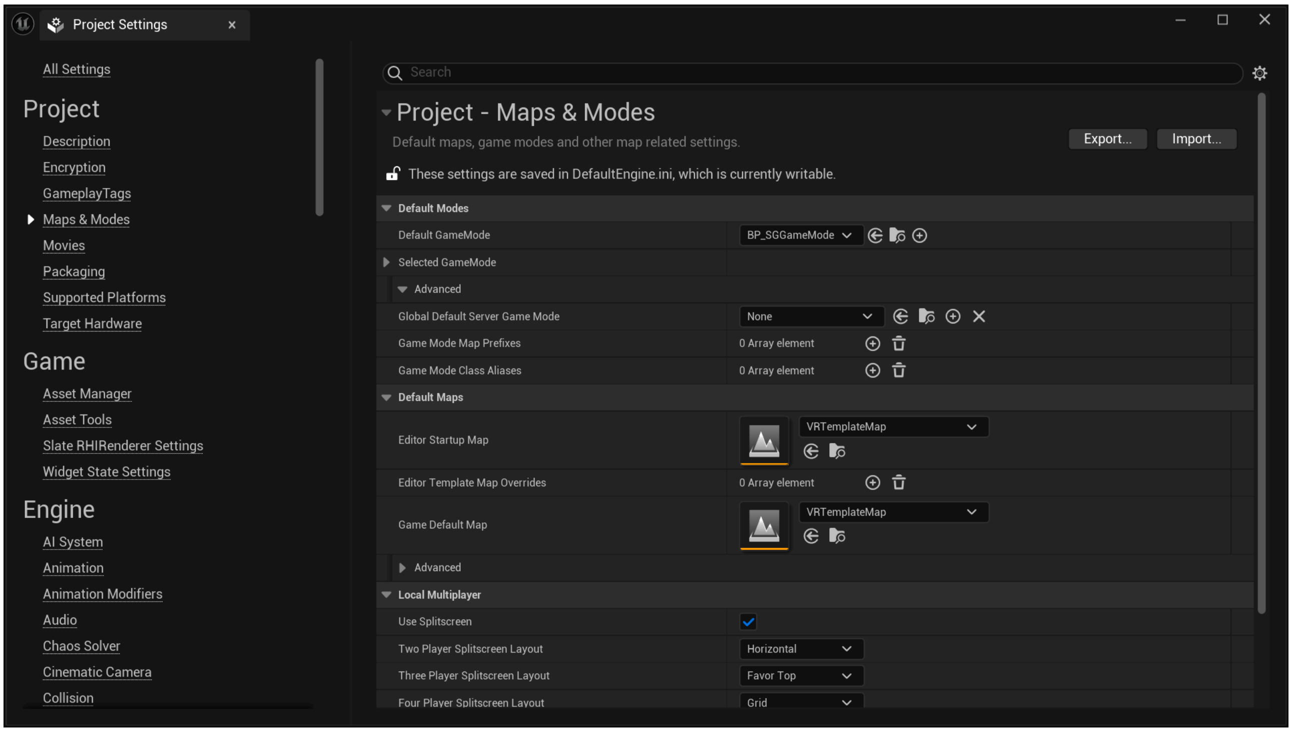
4. After returning to the Content Browser, the Unreal Editor will prompt you to rename `NewBlueprint` to your desired class name. You can rename the class at any time by pressing `F2` or by right-clicking on it and selecting `Rename` from the context menu.



5. Once you have renamed the `NewBlueprint` class to your desired name, click on `Save All` to save the new class to disk.



6. Finally, set your newly created subclass of `SGGameModeBase` as the `Default GameMode`. You can do this by navigating to `Project Settings > Project > Maps & Modes > Default Modes > Default GameMode`.



Project - Maps & Modes

Default maps, game modes and other map related settings.

These settings are saved in DefaultEngine.ini, which is currently writable.

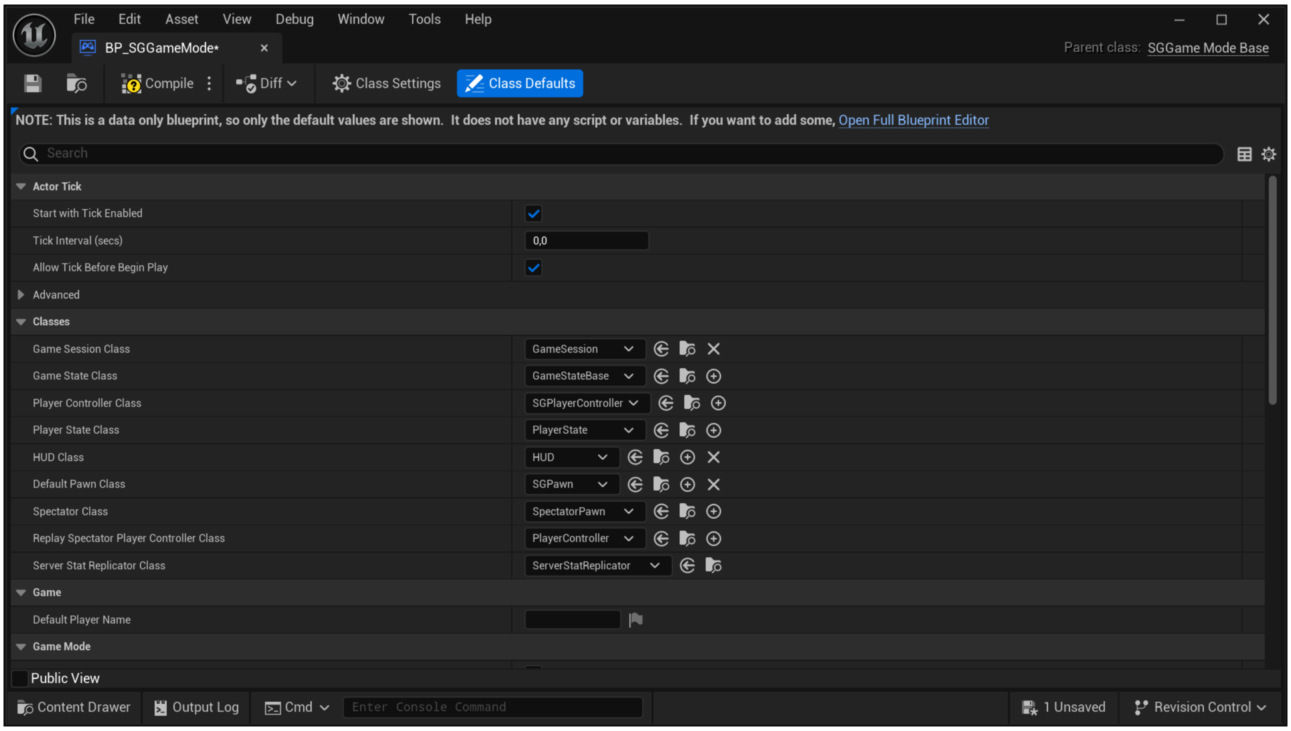
Default Modes
- Default GameMode: BP_SGGameMode
- Selected GameMode
- Advanced
  - Global Default Server Game Mode: None
  - Game Mode Map Prefixes: 0 Array element
  - Game Mode Class Aliases: 0 Array element

Default Maps
- Editor Startup Map: VRTemplateMap
- Editor Template Map Overrides: 0 Array element
- Game Default Map: VRTemplateMap
- Advanced

Local Multiplayer
- Use Splitscreen: ✓
- Two Player Splitscreen Layout: Horizontal
- Three Player Splitscreen Layout: Favor Top
- Four Player Splitscreen Layout: Grid

Setting Up SGPawn

Depening on the Unreal Engine version and your project's type and configuration, you might be able to set `SGPawn` as the `Default Pawn Class` by navigating to `Project Settings > Project > Maps & Modes > Default Modes > Selected GameMode > Default Pawn Class`. However, regardless of the engine version or project type and configuration, you can always configure this by opening your `Default GameMode` and setting the `Default Pawn Class` directly from there. Once set, click on the `Compile` button and save your game mode Blueprint asset.


BP_SGGameMode*

Parent class: SGGame Mode Base

NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some, Open Full Blueprint Editor

Actor Tick	
Start with Tick Enabled	✓
Tick Interval (secs)	0,0
Allow Tick Before Begin Play	✓
Advanced	
Classes	
Game Session Class	GameSession
Game State Class	GameStateBase
Player Controller Class	SGPlayerController
Player State Class	PlayerState
HUD Class	HUD
Default Pawn Class	SGPawn
Spectator Class	SpectatorPawn
Replay Spectator Player Controller Class	PlayerController
Server Stat Replicator Class	ServerStatReplicator
Game	
Default Player Name	
Game Mode	

💡 Tip

For greater control and customization, consider extending the SGPawn.

Caution

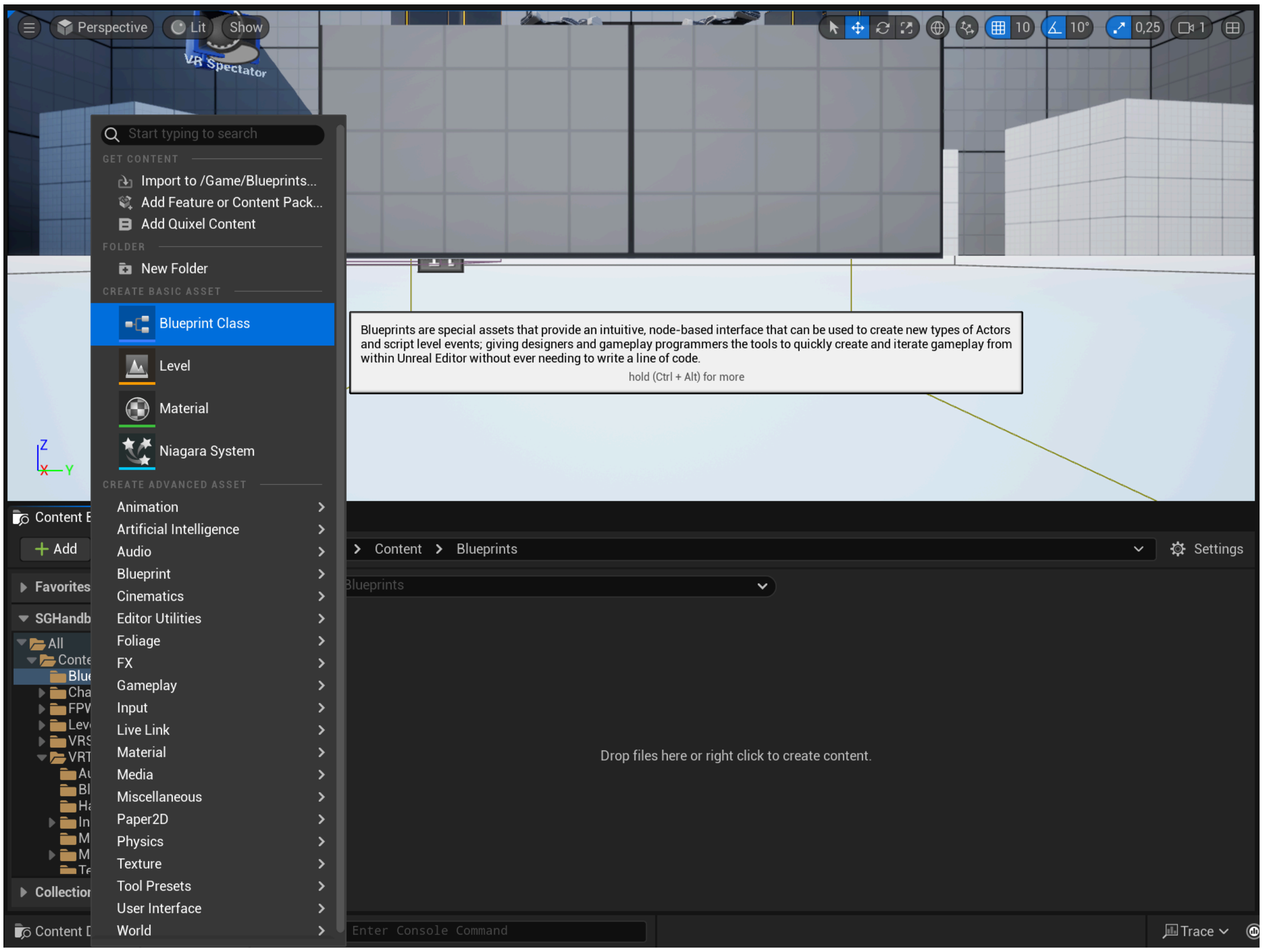
> Setting `SGPawn` or a subclass of it as the `Default Pawn Class` without setting `SGPlayerController` or a subclass of it as the default `Player Controller Class` will cause the `SGPawn` to not function properly. So, it's a strict requirement.

> **Important**
>
> To have a fully functional `SGPawn`, simply setting it up is not enough. You still need to setup the Virtual Hand Meshes and setup the Wrist Tracking Hardware.

Extending SGPawn

Follow these steps to extend and set up your own version of `SGPawn`:

1. In the Content Browser, click the `+ Add` button, then select `Blueprint Class from the menu`. Alternatively, right-click inside the Content Browser and choose `Blueprint Class` from the context menu.

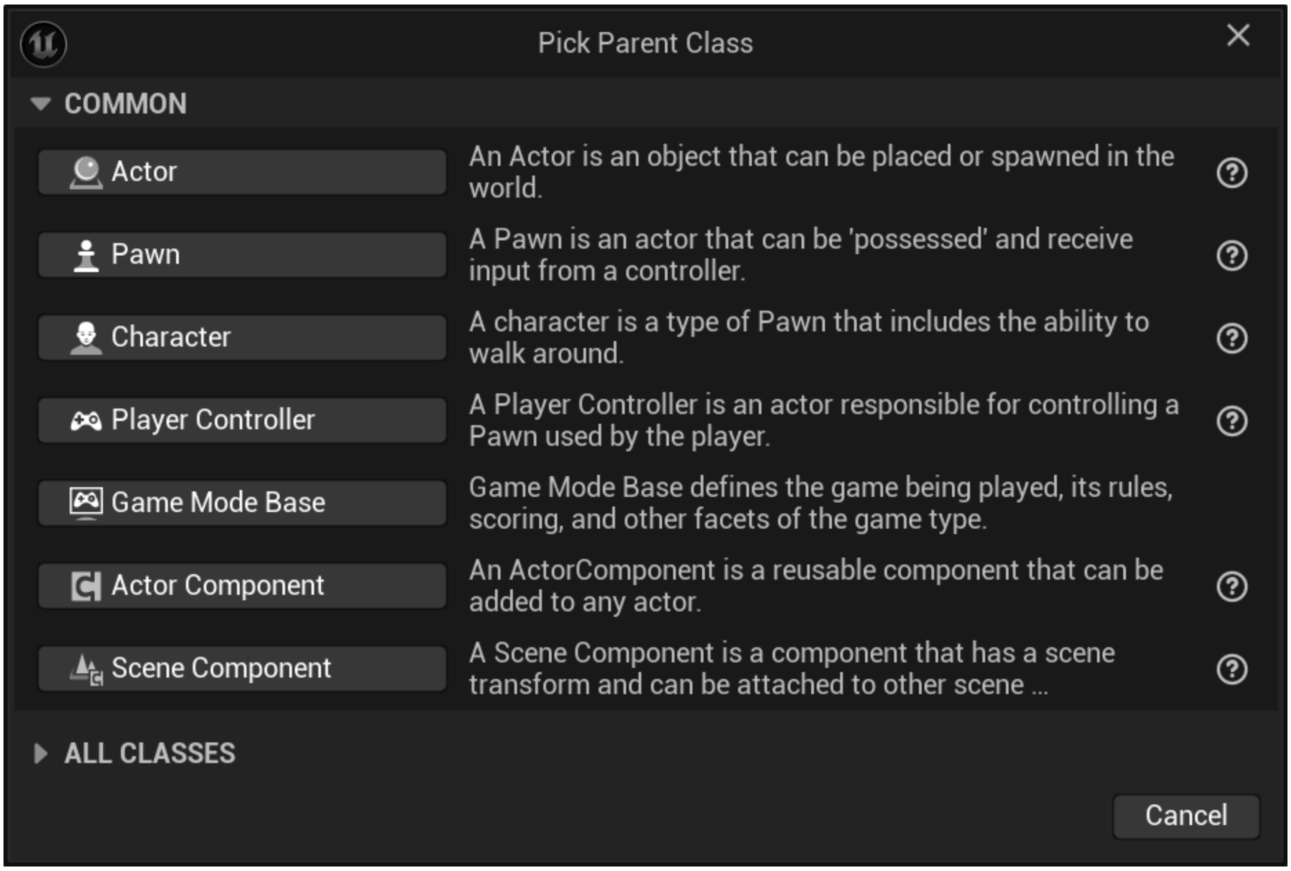


Blueprints are special assets that provide an intuitive, node-based interface that can be used to create new types of Actors and script level events; giving designers and gameplay programmers the tools to quickly create and iterate gameplay from within Unreal Editor without ever needing to write a line of code.
hold (Ctrl + Alt) for more

Drop files here or right click to create content.

Enter Console Command

2. A dialog will appear asking you to choose a parent class. Click on the ALL CLASSES section to expand the list of available classes.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

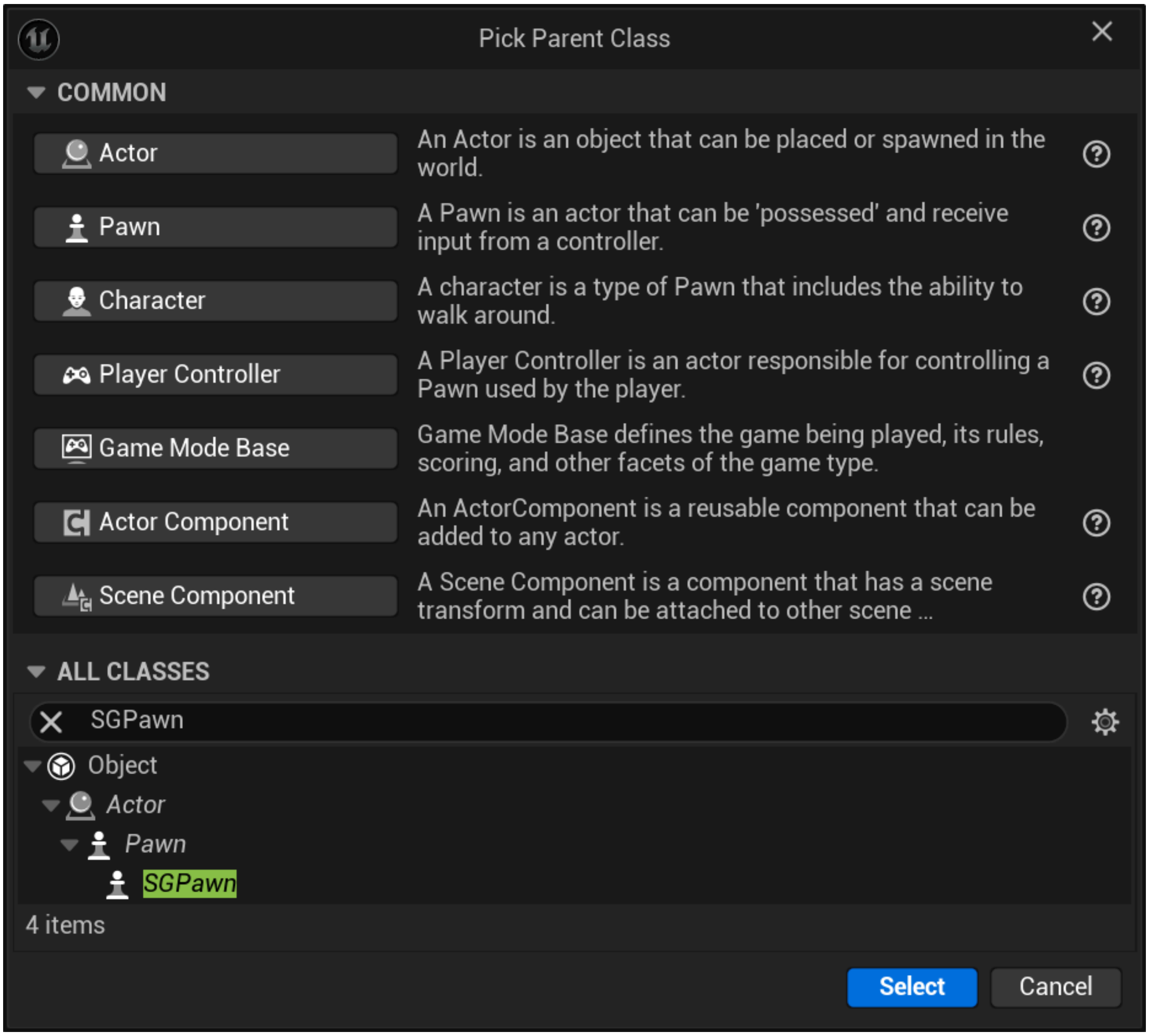
A Player Controller is an actor responsible for controlling a Pawn used by the player.

Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

An ActorComponent is a reusable component that can be added to any actor.

A Scene Component is a component that has a scene transform and can be attached to other scene ...

3. In the expanded `ALL CLASSES` section, start typing `SGPawn` in the Search box. When `SGPawn` appears, select it and click the `Select` button to create your new Blueprint class based on it.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

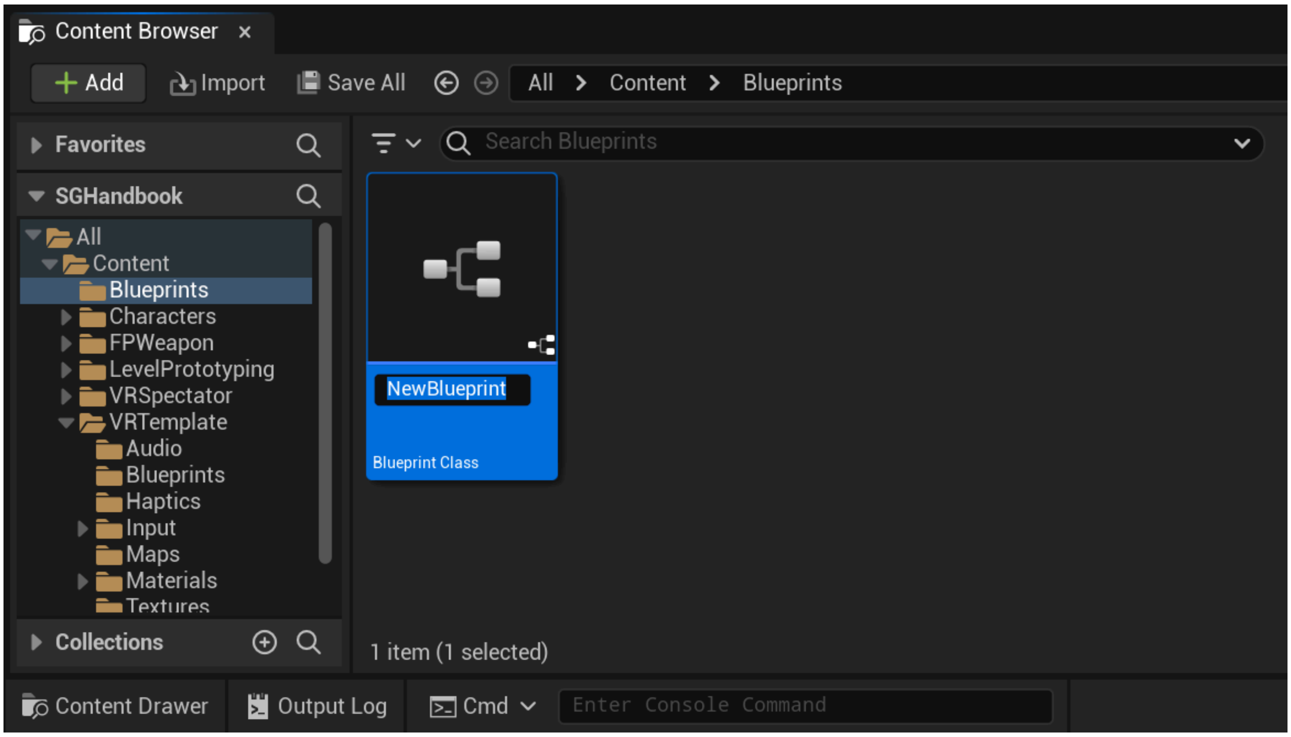
A Player Controller is an actor responsible for controlling a Pawn used by the player.

Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

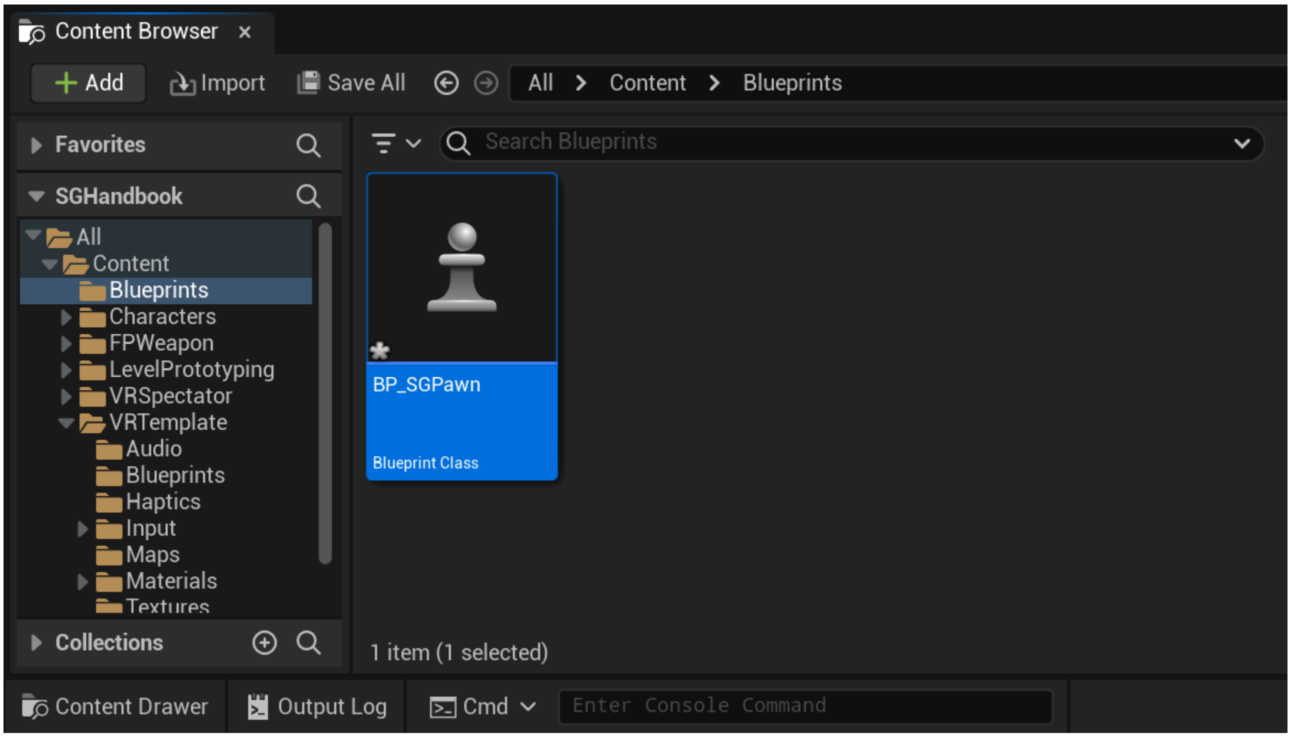
An ActorComponent is a reusable component that can be added to any actor.

A Scene Component is a component that has a scene transform and can be attached to other scene ...

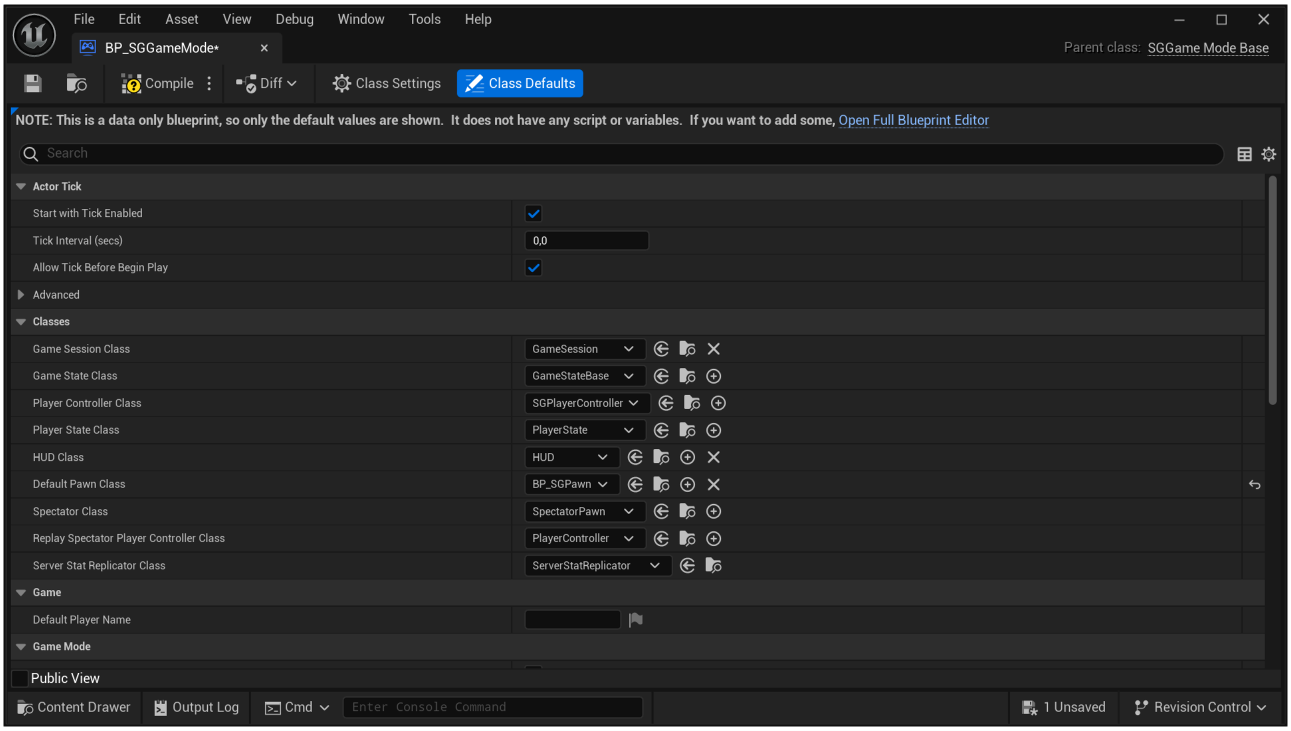
4. After returning to the Content Browser, the Unreal Editor will prompt you to rename `NewBlueprint` to your desired class name. You can rename the class at any time by pressing `F2` or by right-clicking on it and selecting `Rename` from the context menu.



5. Once you have renamed the `NewBlueprint` class to your desired name, click on `Save All` to save the new class to disk.



6. Finally, set your newly created subclass of `SGPawn` as the `Default Pawn Class`. Depening on the Unreal Engine version and your project's type and configuration, you might be able do this by navigating to `Project Settings > Project > Maps & Modes > Default Modes > Selected GameMode > Default Pawn Class`. However, regardless of the engine version or project type and configuration, you can always configure this by opening your `Default GameMode` and setting the `Default Pawn Class` directly from there. Once set, click on the `Compile` button and save your game mode Blueprint asset.



NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some, Open Full Blueprint Editor

Actor Tick
- Start with Tick Enabled: ✓
- Tick Interval (secs): 0,0
- Allow Tick Before Begin Play: ✓
- Advanced

Classes
- Game Session Class: GameSession
- Game State Class: GameStateBase
- Player Controller Class: SGPlayerController
- Player State Class: PlayerState
- HUD Class: HUD
- Default Pawn Class: BP_SGPawn
- Spectator Class: SpectatorPawn
- Replay Spectator Player Controller Class: PlayerController
- Server Stat Replicator Class: ServerStatReplicator

Game
- Default Player Name:

Game Mode

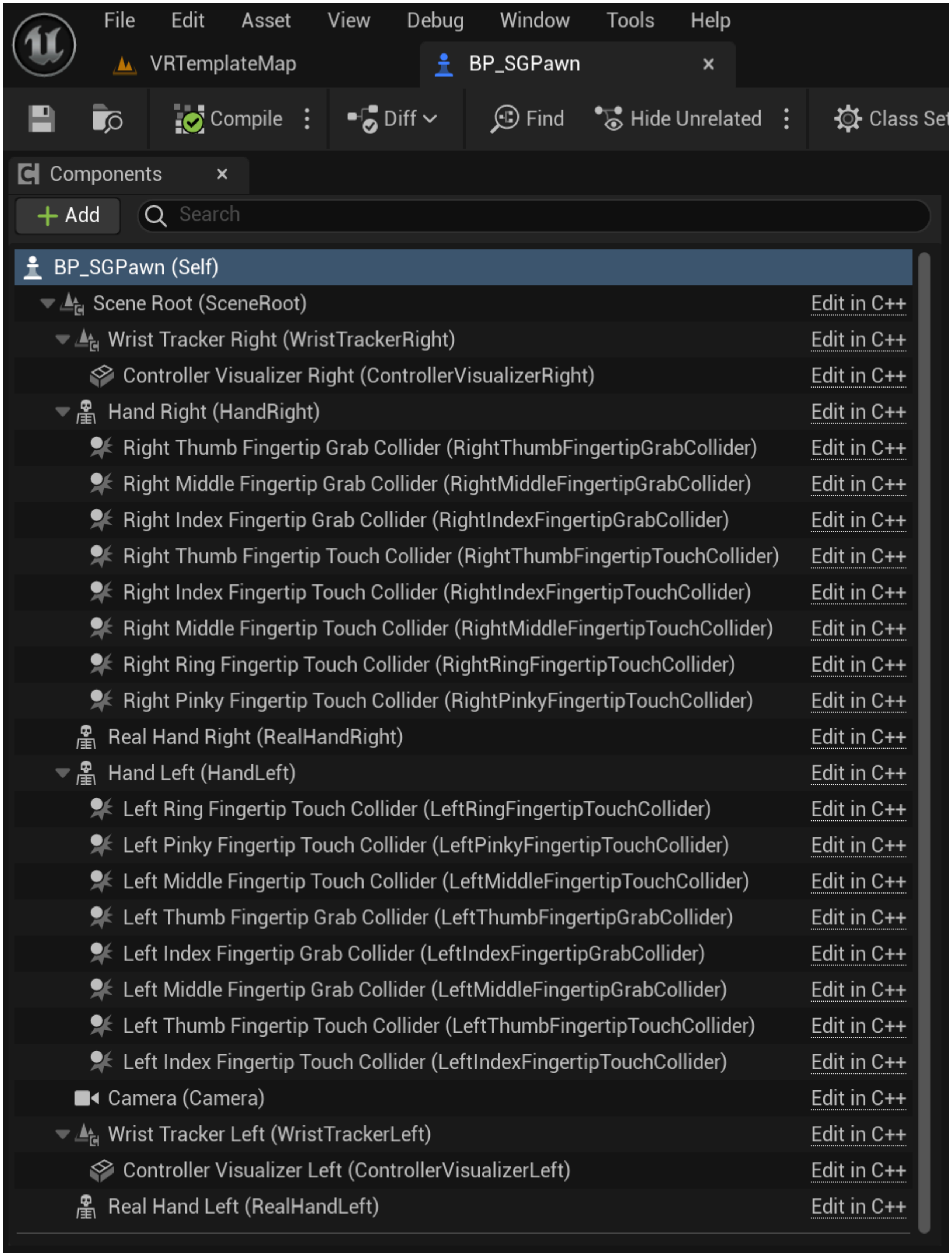
> **Important**
>
> To have a fully functional `SGPawn`, simply setting it up is not enough. You still need to setup the Virtual Hand Meshes and setup the Wrist Tracking Hardware.

Customizing SGPawn

Customizing the `SGPawn` after subclassing is straightforward and flexible.

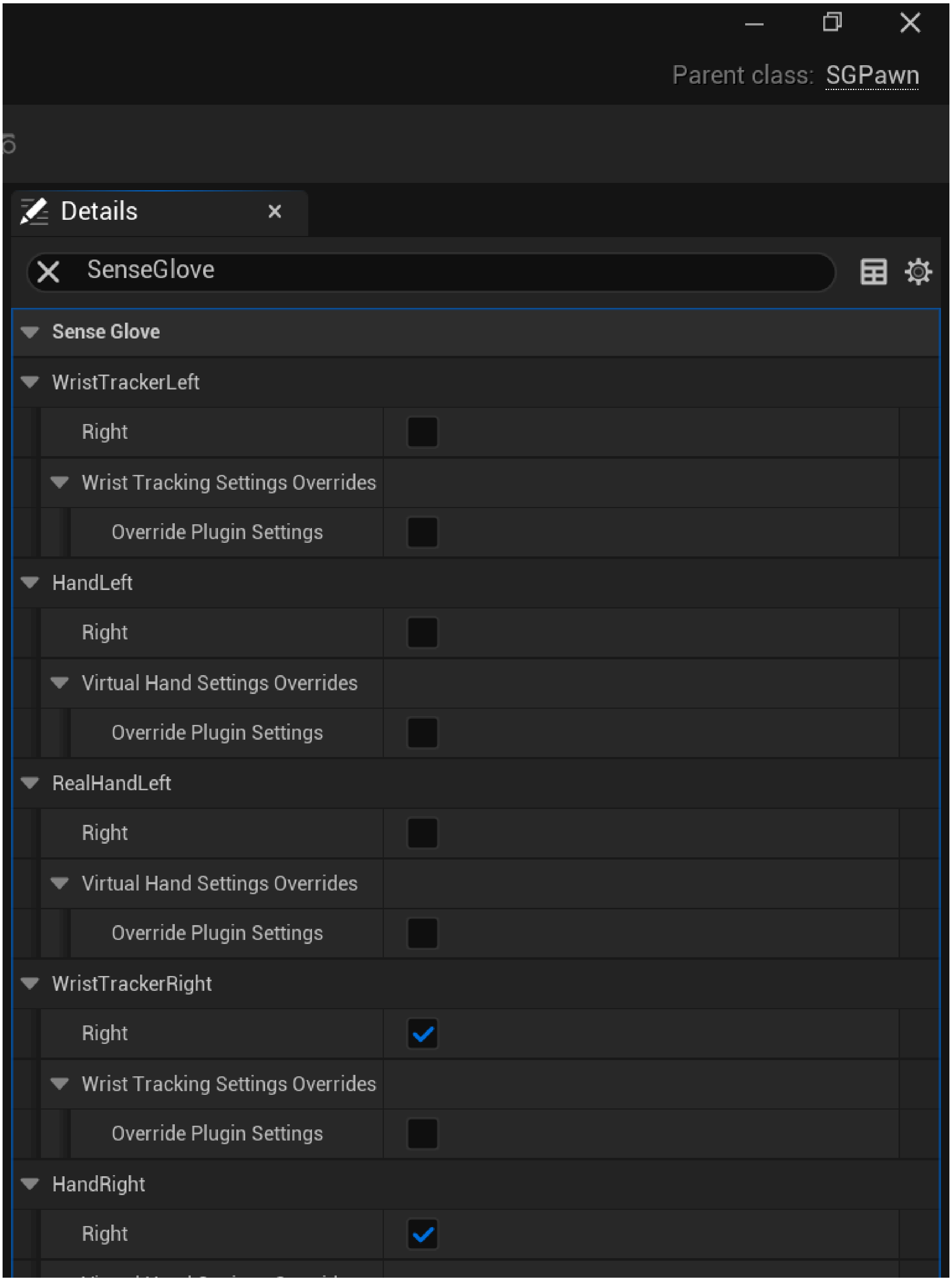
The `SGPawn` class includes several key subcomponents:

- `Wrist Tracker Left` and `Wrist Tracker Right` of type `SGWristTrackerComponent`.
- `HandLeft` and `HandRight` of type `SGVirtualHandComponent` and represent the virtual hand models visible to the user in the simulation.
- `RealHandLeft` and `RealHandRight` of type `SGVirtualHandComponent`. By default, these are hidden and represent the real hands within the simulation. These components are useful if you need to separate the rendering of the virtual hands from the real hands. For instance, the virtual hands typically have collisions and cannot pass through objects, while the real hands are not constrained in this way.



BP_SGPawn (Self)
- Scene Root (SceneRoot) — Edit in C++
  - Wrist Tracker Right (WristTrackerRight) — Edit in C++
    - Controller Visualizer Right (ControllerVisualizerRight) — Edit in C++
  - Hand Right (HandRight) — Edit in C++
    - Right Thumb Fingertip Grab Collider (RightThumbFingertipGrabCollider) — Edit in C++
    - Right Middle Fingertip Grab Collider (RightMiddleFingertipGrabCollider) — Edit in C++
    - Right Index Fingertip Grab Collider (RightIndexFingertipGrabCollider) — Edit in C++
    - Right Thumb Fingertip Touch Collider (RightThumbFingertipTouchCollider) — Edit in C++
    - Right Index Fingertip Touch Collider (RightIndexFingertipTouchCollider) — Edit in C++
    - Right Middle Fingertip Touch Collider (RightMiddleFingertipTouchCollider) — Edit in C++
    - Right Ring Fingertip Touch Collider (RightRingFingertipTouchCollider) — Edit in C++
    - Right Pinky Fingertip Touch Collider (RightPinkyFingertipTouchCollider) — Edit in C++
  - Real Hand Right (RealHandRight) — Edit in C++
  - Hand Left (HandLeft) — Edit in C++
    - Left Ring Fingertip Touch Collider (LeftRingFingertipTouchCollider) — Edit in C++
    - Left Pinky Fingertip Touch Collider (LeftPinkyFingertipTouchCollider) — Edit in C++
    - Left Middle Fingertip Touch Collider (LeftMiddleFingertipTouchCollider) — Edit in C++
    - Left Thumb Fingertip Grab Collider (LeftThumbFingertipGrabCollider) — Edit in C++
    - Left Index Fingertip Grab Collider (LeftIndexFingertipGrabCollider) — Edit in C++
    - Left Middle Fingertip Grab Collider (LeftMiddleFingertipGrabCollider) — Edit in C++
    - Left Thumb Fingertip Touch Collider (LeftThumbFingertipTouchCollider) — Edit in C++
    - Left Index Fingertip Touch Collider (LeftIndexFingertipTouchCollider) — Edit in C++
  - Camera (Camera) — Edit in C++
  - Wrist Tracker Left (WristTrackerLeft) — Edit in C++
    - Controller Visualizer Left (ControllerVisualizerLeft) — Edit in C++
  - Real Hand Left (RealHandLeft) — Edit in C++

Also, it's possible to filter the properties for these SenseGlove components inside the `Details` panel inside the `SGPawn` Blueprint Editor by typing the word `SenseGlove` inside `Search` box of the `Details` panel.

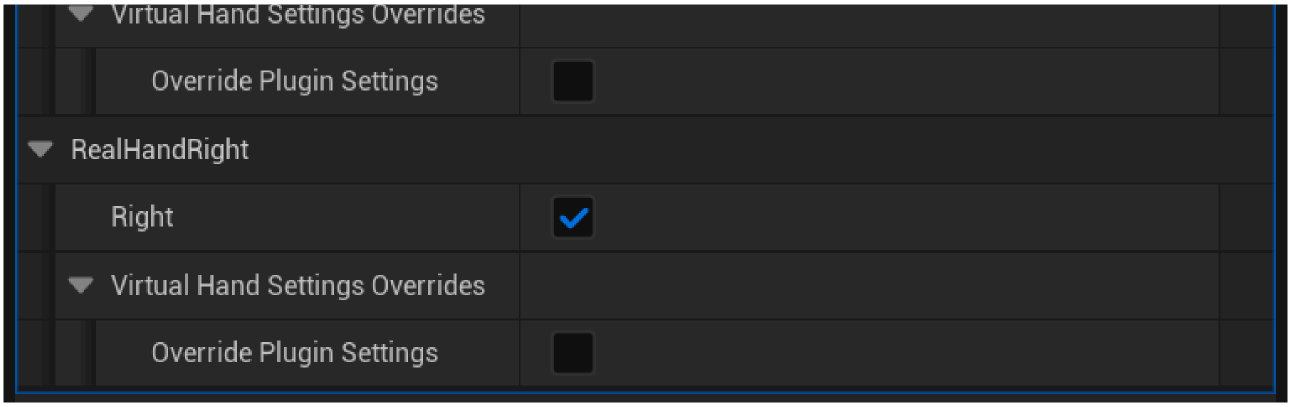

Parent class: SGPawn

Details

SenseGlove

Sense Glove

WristTrackerLeft	
Right	☐
Wrist Tracking Settings Overrides	
Override Plugin Settings	☐
HandLeft	
Right	☐
Virtual Hand Settings Overrides	
Override Plugin Settings	☐
RealHandLeft	
Right	☐
Virtual Hand Settings Overrides	
Override Plugin Settings	☐
WristTrackerRight	
Right	✓
Wrist Tracking Settings Overrides	
Override Plugin Settings	☐
HandRight	
Right	✓

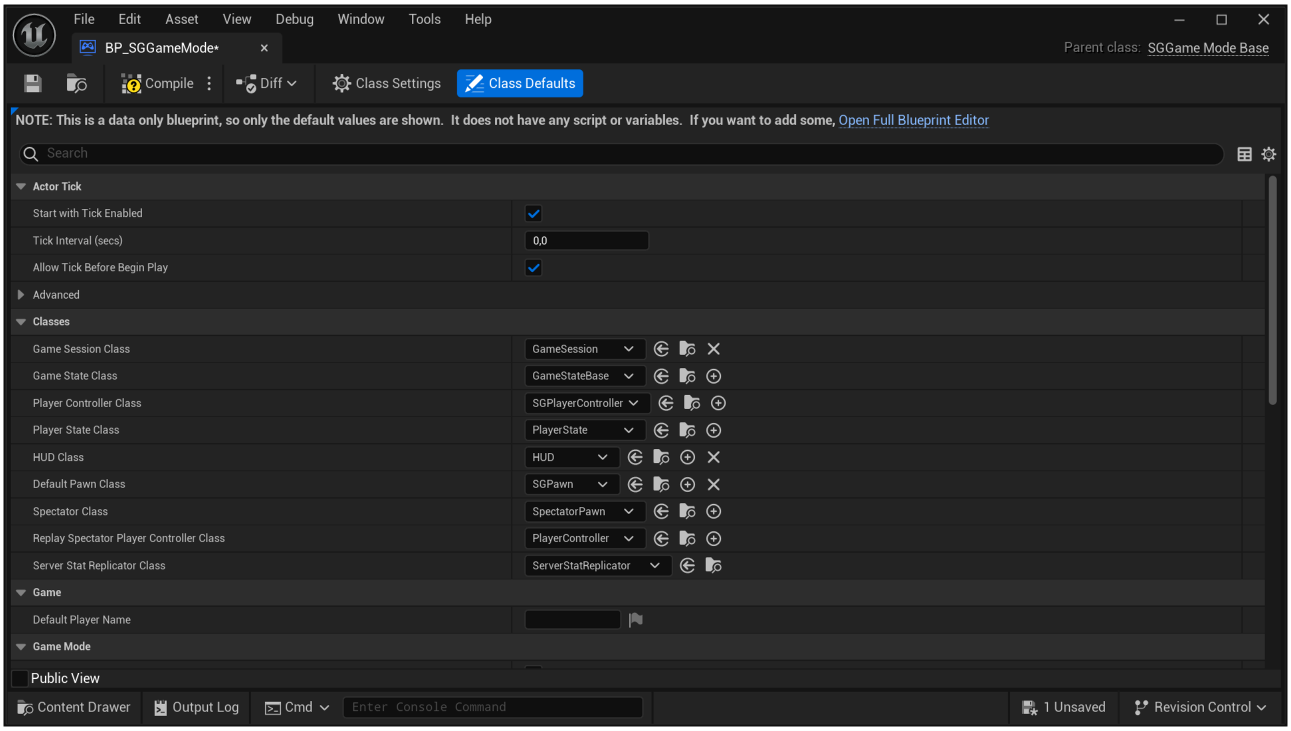


Virtual Hand Settings Overrides	
Override Plugin Settings	☐
RealHandRight	
Right	✓
Virtual Hand Settings Overrides	
Override Plugin Settings	☐

Please visit how to setup the Virtual Hand Meshes, The Virtual Hand Mesh Settings, and how to setup the Wrist Tracking Hardware sections for more information.

Setting Up SGPlayerController

Depening on the Unreal Engine version and your project's type and configuration, you might be able to set `SGPlayerController` as the default `Player Controller Class` by navigating to `Project Settings > Project > Maps & Modes > Default Modes > Selected GameMode > Player Controller Class`. However, regardless of the engine version or project type and configuration, you can always configure this by opening your `Default GameMode` and setting the default `Player Controller Class` directly from there. Once set, click on the `Compile` button and save your game mode Blueprint asset.



NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some, Open Full Blueprint Editor

Actor Tick	
Start with Tick Enabled	✓
Tick Interval (secs)	0,0
Allow Tick Before Begin Play	✓
Advanced	
Classes	
Game Session Class	GameSession
Game State Class	GameStateBase
Player Controller Class	SGPlayerController
Player State Class	PlayerState
HUD Class	HUD
Default Pawn Class	SGPawn
Spectator Class	SpectatorPawn
Replay Spectator Player Controller Class	PlayerController
Server Stat Replicator Class	ServerStatReplicator
Game	
Default Player Name	
Game Mode	

Tip

For greater control and customization, consider extending the SGPlayerController.

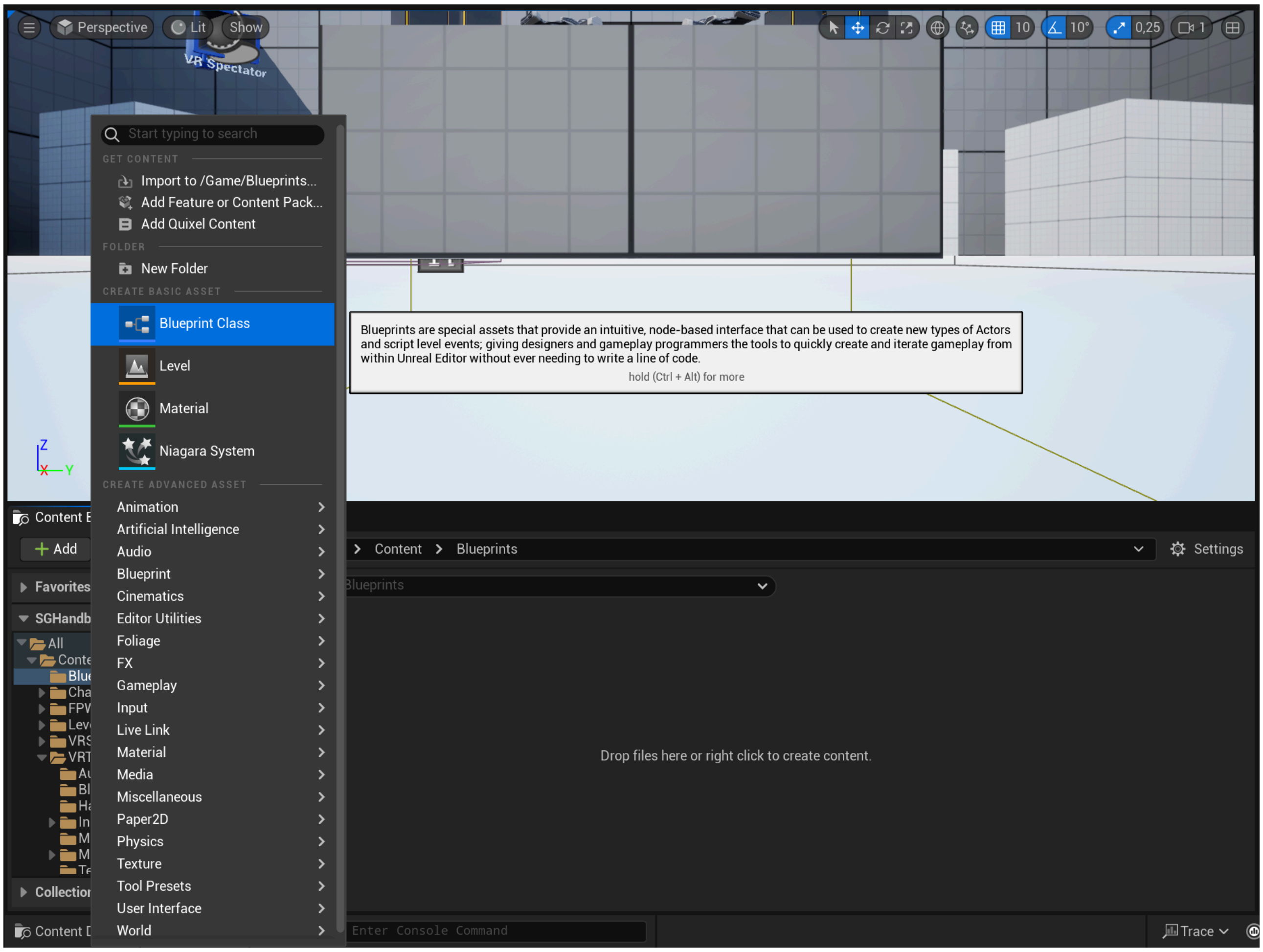
Caution

> Setting `SGPlayerController` or a subclass of it as the default `Player Controller Class` without setting `SGPawn` or a subclass of it as the `Default Pawn Class` will cause your simulation or editor to crash. So, it's a strict requirement.

Extending SGPlayerController

Follow these steps to extend and set up your own version of `SGPlayerController`:

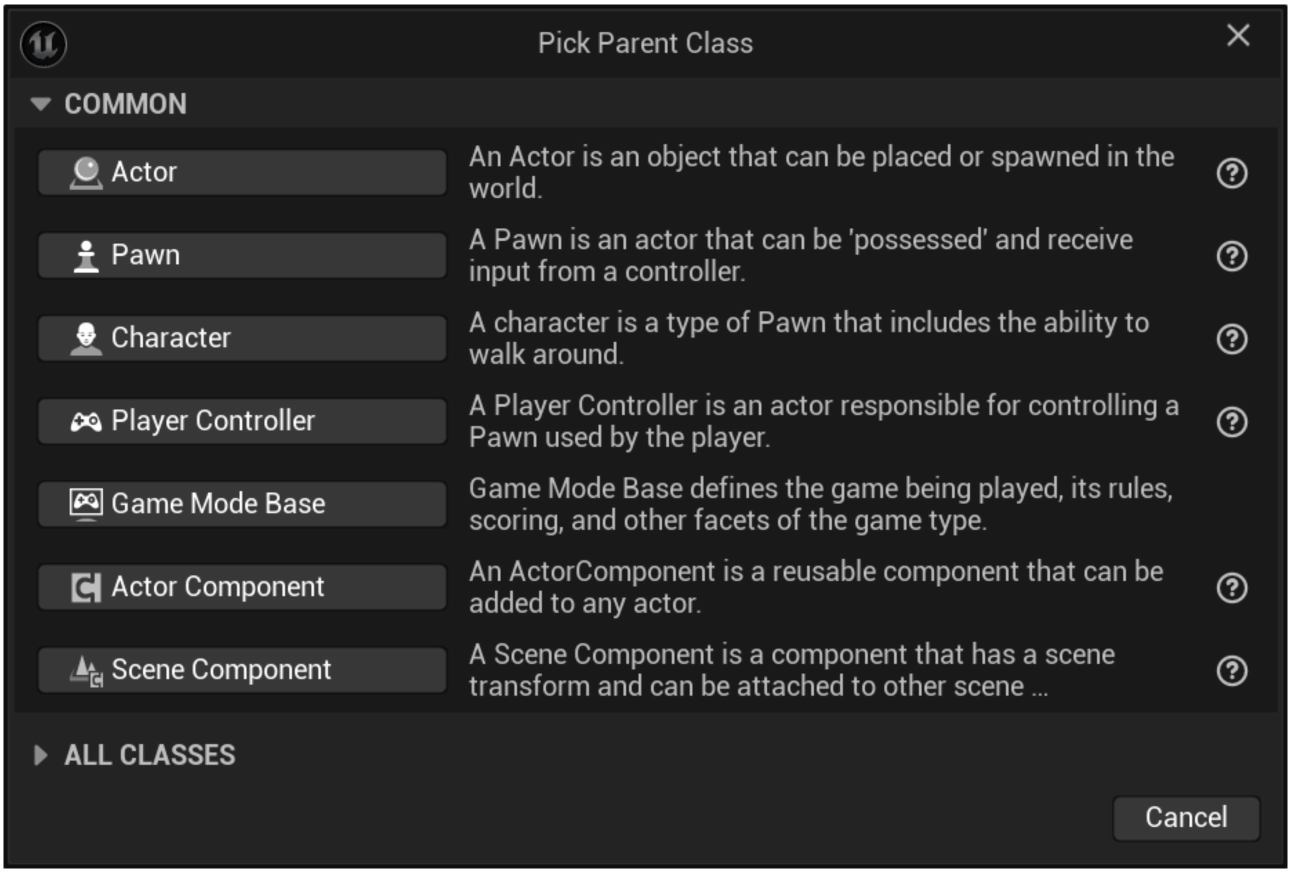
1. In the Content Browser, click the `+ Add` button, then select `Blueprint Class from the menu`. Alternatively, right-click inside the Content Browser and choose `Blueprint Class` from the context menu.



Blueprints are special assets that provide an intuitive, node-based interface that can be used to create new types of Actors and script level events; giving designers and gameplay programmers the tools to quickly create and iterate gameplay from within Unreal Editor without ever needing to write a line of code.
hold (Ctrl + Alt) for more

Drop files here or right click to create content.

2. A dialog will appear asking you to choose a parent class. Click on the `ALL CLASSES` section to expand the list of available classes.



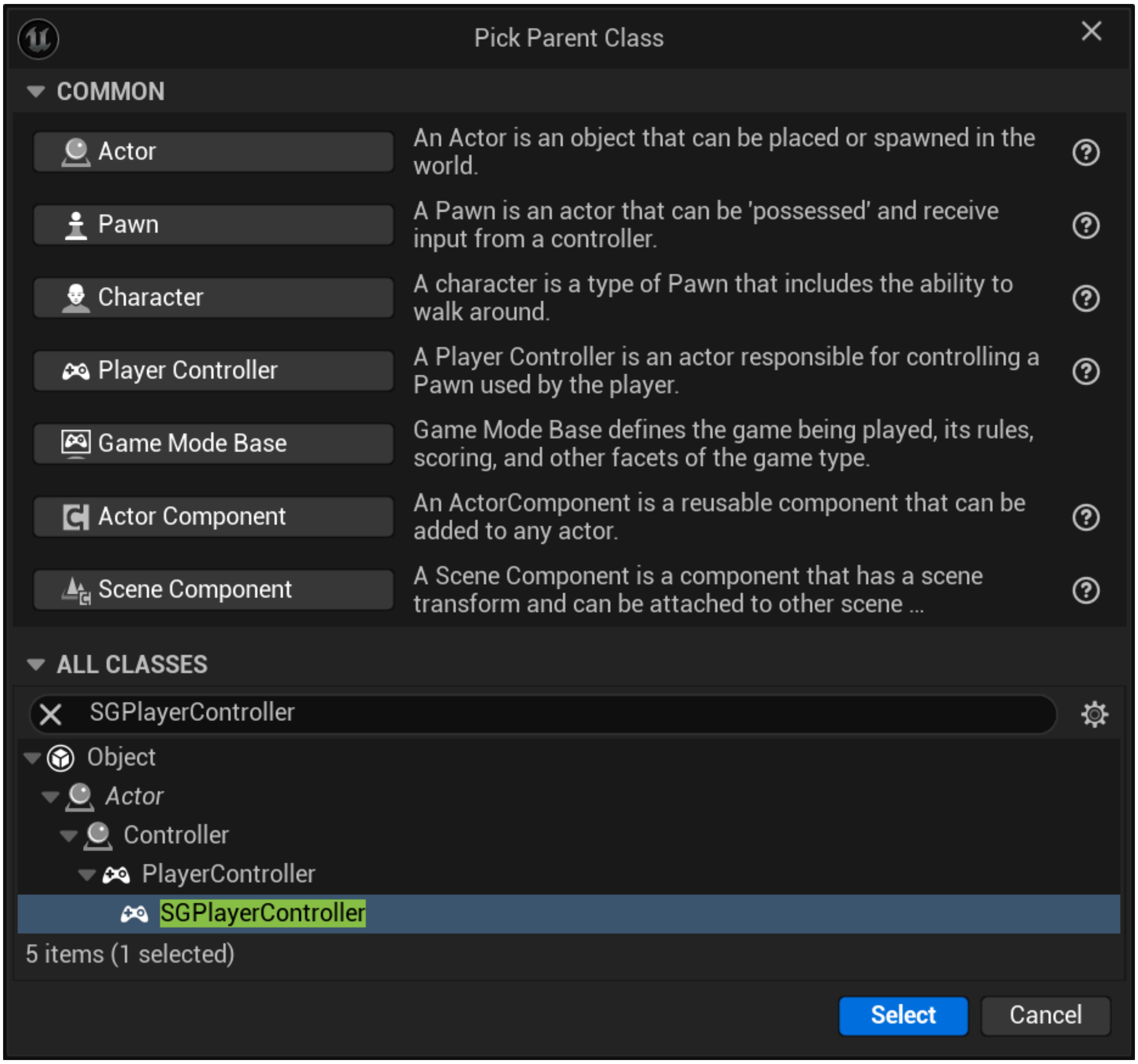
Pick Parent Class

COMMON

Actor	An Actor is an object that can be placed or spawned in the world.
Pawn	A Pawn is an actor that can be 'possessed' and receive input from a controller.
Character	A character is a type of Pawn that includes the ability to walk around.
Player Controller	A Player Controller is an actor responsible for controlling a Pawn used by the player.
Game Mode Base	Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.
Actor Component	An ActorComponent is a reusable component that can be added to any actor.
Scene Component	A Scene Component is a component that has a scene transform and can be attached to other scene ...

ALL CLASSES

3. In the expanded `ALL CLASSES` section, start typing `SGPlayerController` in the Search box. When `SGPlayerController` appears, select it and click the `Select` button to create your new Blueprint class based on it.


Pick Parent Class

COMMON

Actor	An Actor is an object that can be placed or spawned in the world.
Pawn	A Pawn is an actor that can be 'possessed' and receive input from a controller.
Character	A character is a type of Pawn that includes the ability to walk around.
Player Controller	A Player Controller is an actor responsible for controlling a Pawn used by the player.
Game Mode Base	Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.
Actor Component	An ActorComponent is a reusable component that can be added to any actor.
Scene Component	A Scene Component is a component that has a scene transform and can be attached to other scene ...

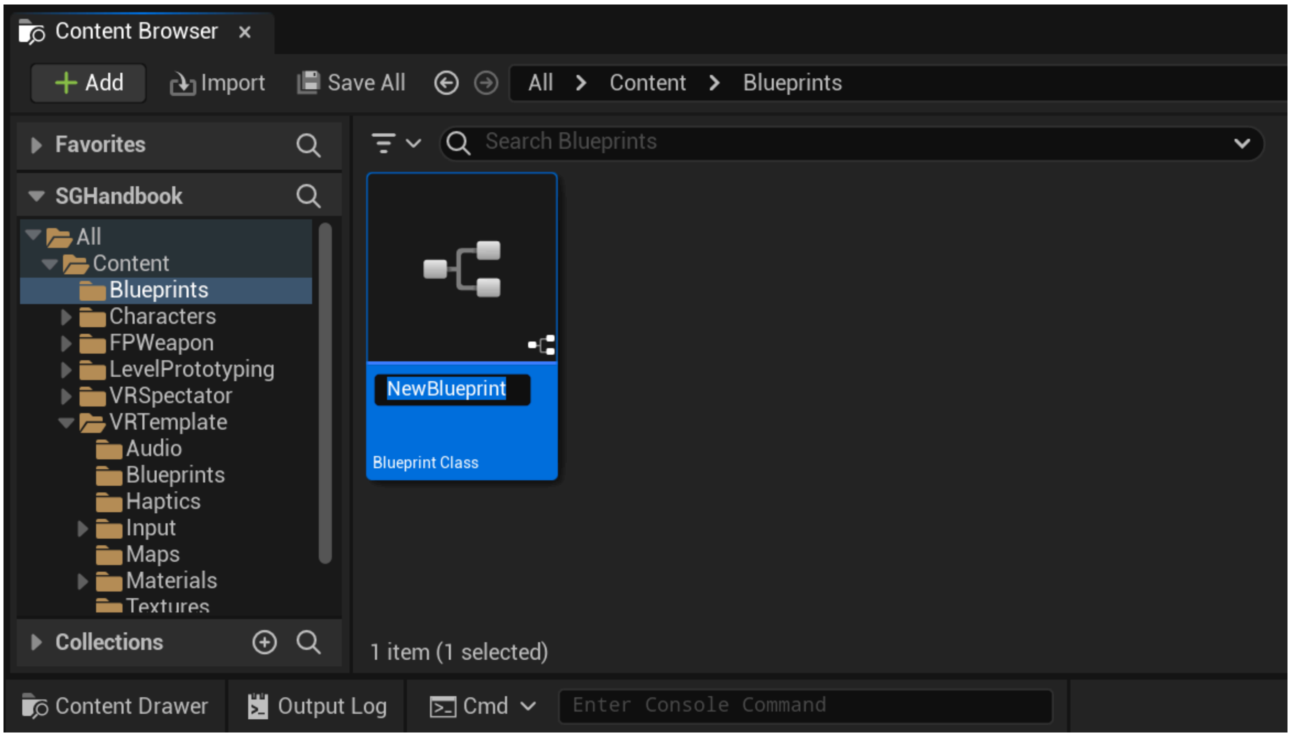
ALL CLASSES

SGPlayerController

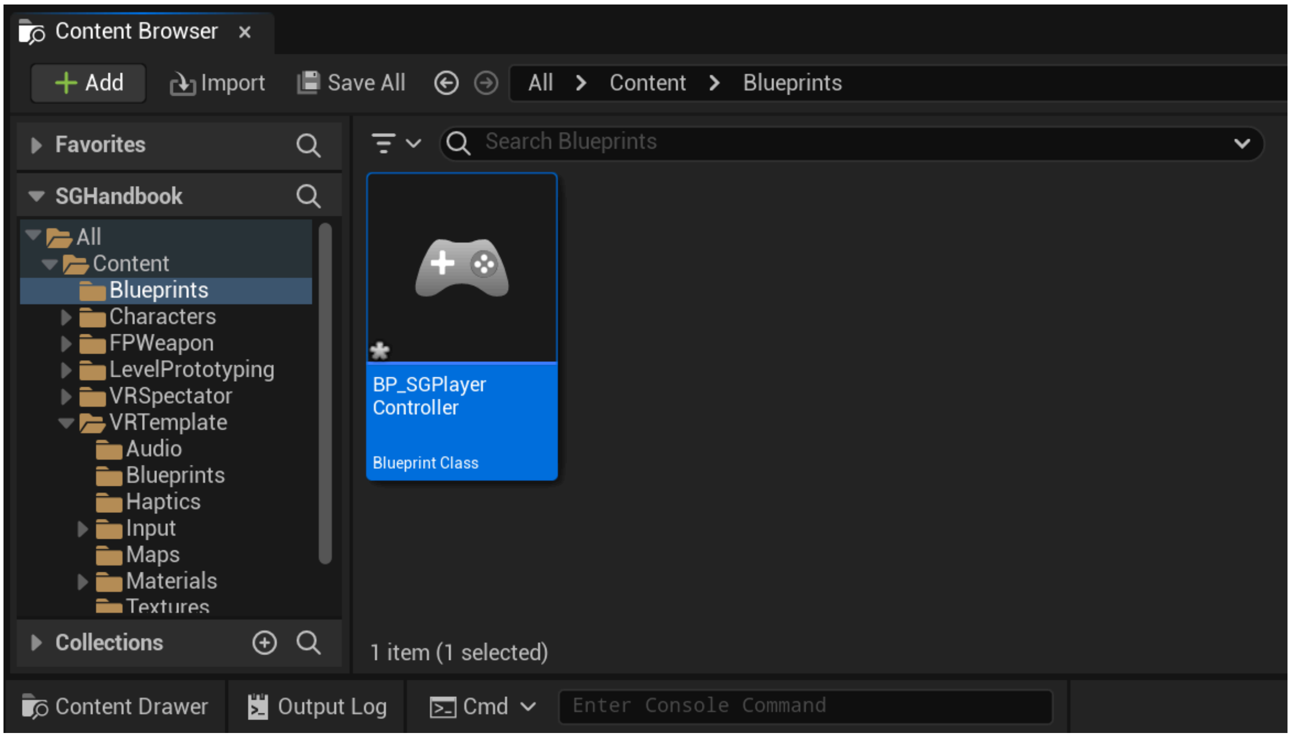
```
Object
  Actor
    Controller
      PlayerController
        SGPlayerController
```

5 items (1 selected)

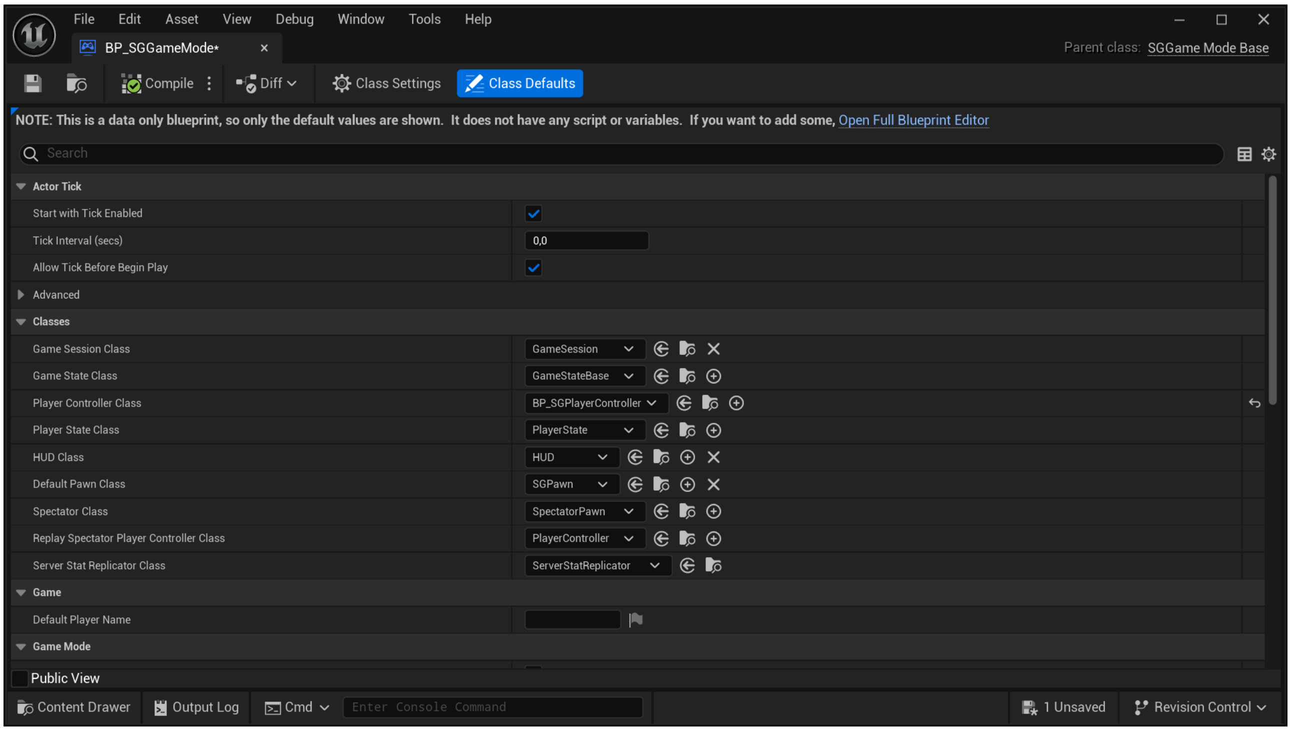
4. After returning to the Content Browser, the Unreal Editor will prompt you to rename `NewBlueprint` to your desired class name. You can rename the class at any time by pressing `F2` or by right-clicking on it and selecting `Rename` from the context menu.



5. Once you have renamed the `NewBlueprint` class to your desired name, click on `Save All` to save the new class to disk.



6. Finally, set your newly created subclass of `SGPlayerController` as the default `Player Controller Class`. Depening on the Unreal Engine version and your project's type and configuration, you might be able do this by navigating to `Project Settings > Project > Maps & Modes > Default Modes > Selected GameMode > Player Controller Class`. However, regardless of the engine version or project type and configuration, you can always configure this by opening your `Default GameMode` and setting the default `Player Controller Class` directly from there. Once set, click on the `Compile` button and save your game mode Blueprint asset.

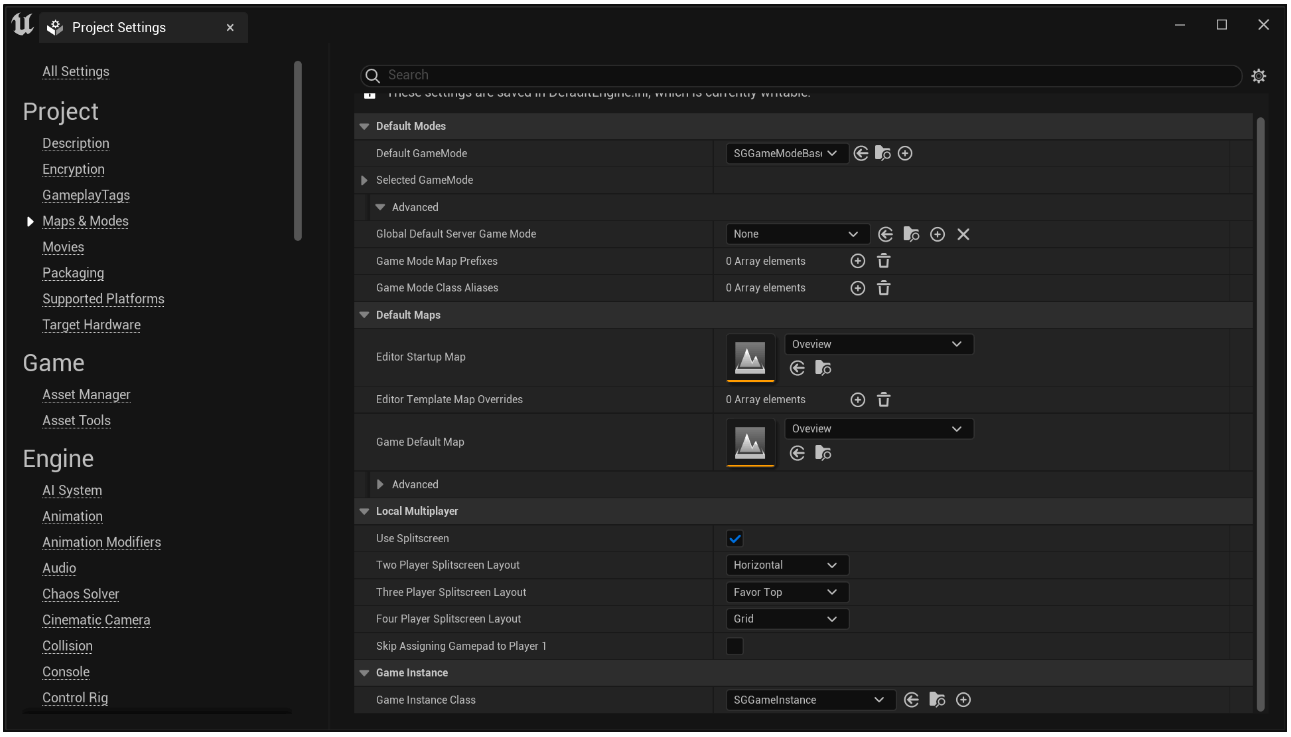


NOTE: This is a data only blueprint, so only the default values are shown. It does not have any script or variables. If you want to add some, Open Full Blueprint Editor

Actor Tick	
Start with Tick Enabled	✓
Tick Interval (secs)	0,0
Allow Tick Before Begin Play	✓
Advanced	
Classes	
Game Session Class	GameSession
Game State Class	GameStateBase
Player Controller Class	BP_SGPlayerController
Player State Class	PlayerState
HUD Class	HUD
Default Pawn Class	SGPawn
Spectator Class	SpectatorPawn
Replay Spectator Player Controller Class	PlayerController
Server Stat Replicator Class	ServerStatReplicator
Game	
Default Player Name	
Game Mode	

Setting Up SGGameInstance

Setting `SGGameInstance` as the default `Game Instance Class` is very straightforward. You can do this by navigating to `Project Settings > Project > Maps & Modes > Game Instance > Game Instance Class`.



These settings are saved in DefaultEngine.ini, which is currently writable.

Default Modes

Default GameMode	SGGameModeBase
Selected GameMode	
Advanced	
Global Default Server Game Mode	None
Game Mode Map Prefixes	0 Array elements
Game Mode Class Aliases	0 Array elements

Default Maps

Editor Startup Map	Oveview
Editor Template Map Overrides	0 Array elements
Game Default Map	Oveview
Advanced	

Local Multiplayer

Use Splitscreen	✓
Two Player Splitscreen Layout	Horizontal
Three Player Splitscreen Layout	Favor Top
Four Player Splitscreen Layout	Grid
Skip Assigning Gamepad to Player 1	

Game Instance

Game Instance Class	SGGameInstance

Tip

For greater control and customization, consider extending the SGGameInstance.

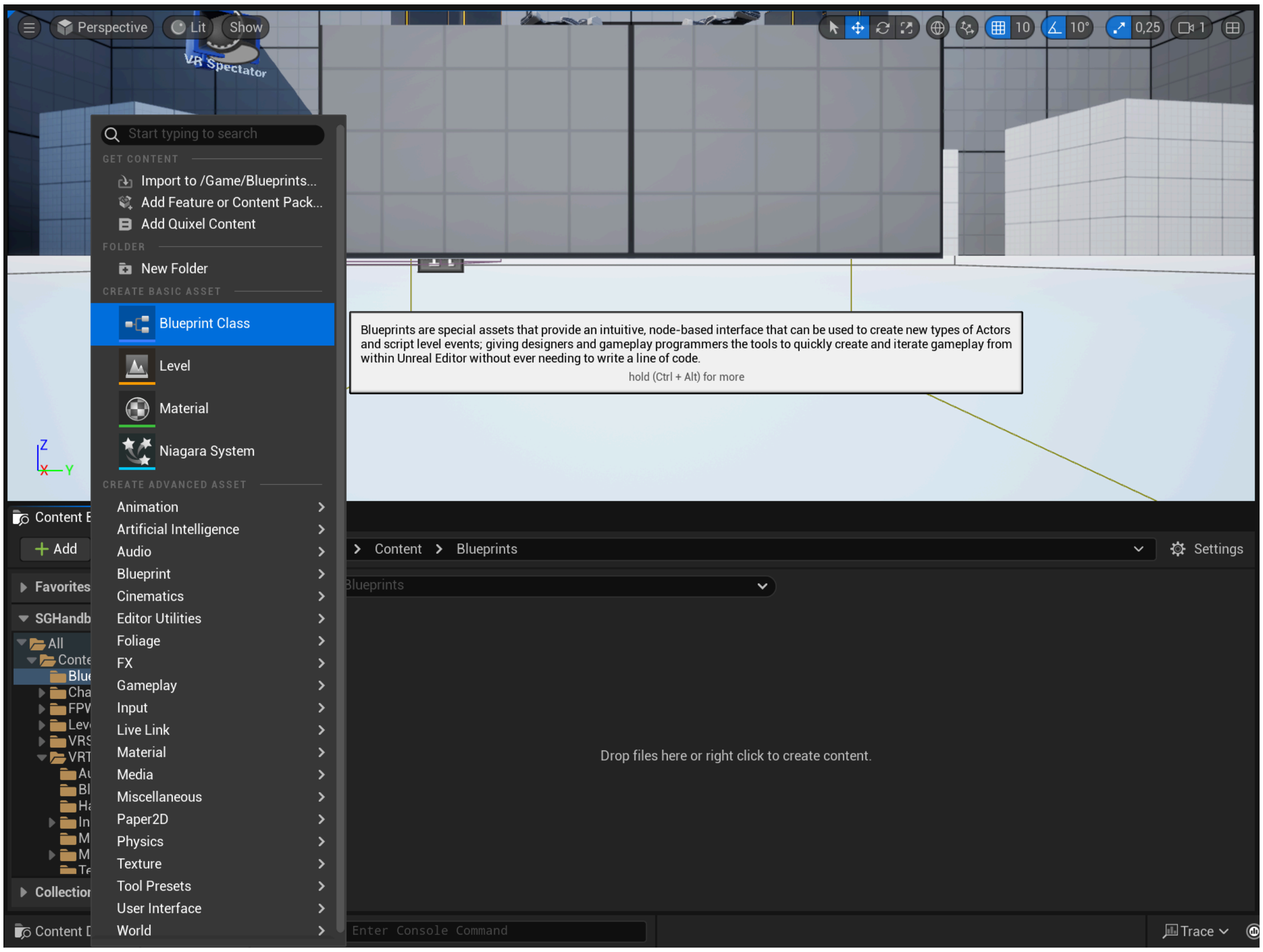
Important

Currently, setting `SGGameModeBase` or a subclass of it as the default `Game Instance Class` is not a strict requirement. However, if you intend to use any SenseGlove console command it becomes mandatory. If not set, SenseGlove console commands will not be recognized by Unreal Engine.

Extending SGGameInstance

Follow these steps to extend and set up your own version of `SGGameInstance`:

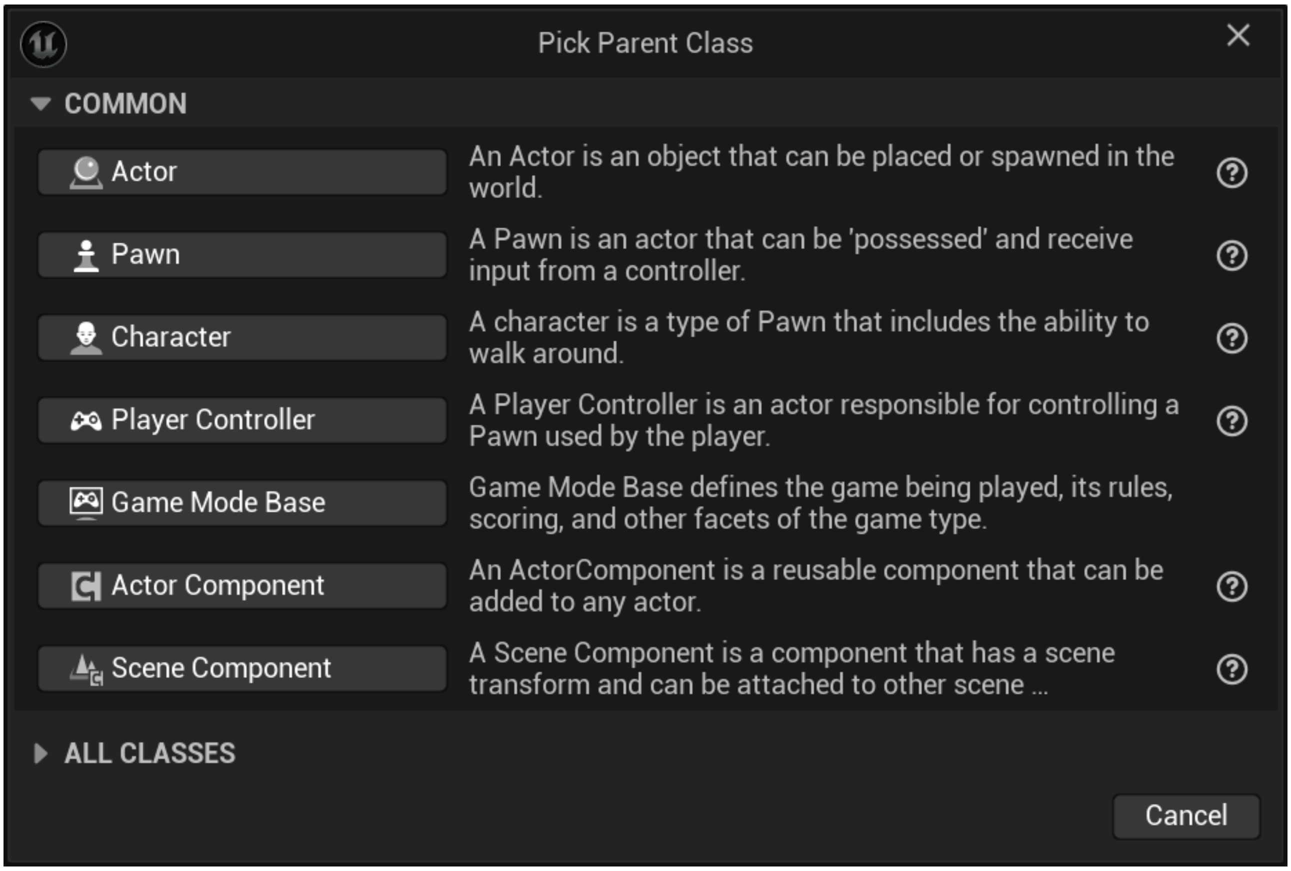
1. In the Content Browser, click the `+ Add` button, then select `Blueprint Class from the menu`. Alternatively, right-click inside the Content Browser and choose `Blueprint Class` from the context menu.



Blueprints are special assets that provide an intuitive, node-based interface that can be used to create new types of Actors and script level events; giving designers and gameplay programmers the tools to quickly create and iterate gameplay from within Unreal Editor without ever needing to write a line of code.
hold (Ctrl + Alt) for more

Drop files here or right click to create content.

2. A dialog will appear asking you to choose a parent class. Click on the `ALL CLASSES` section to expand the list of available classes.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

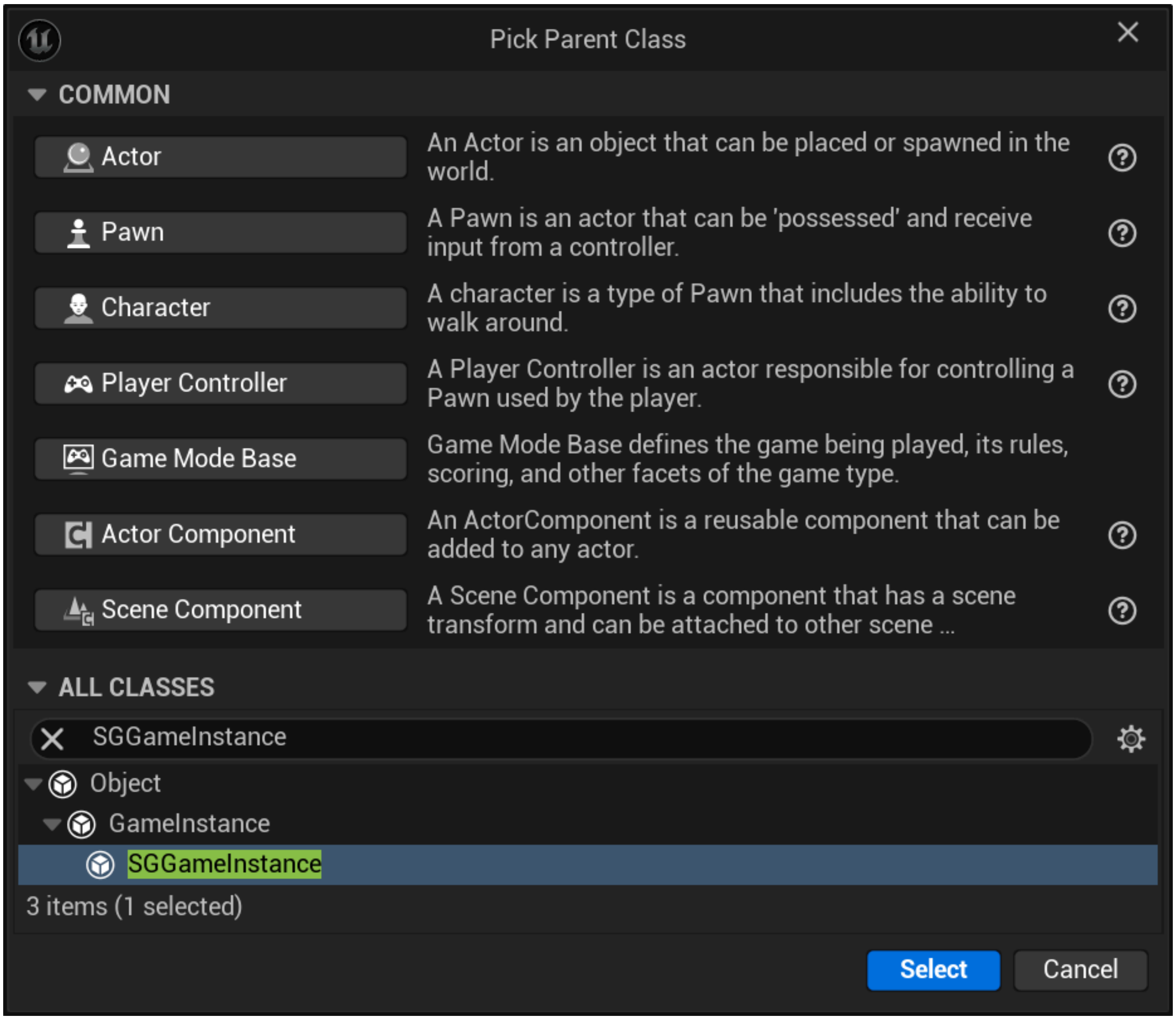
A Player Controller is an actor responsible for controlling a Pawn used by the player.

Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

An ActorComponent is a reusable component that can be added to any actor.

A Scene Component is a component that has a scene transform and can be attached to other scene ...

3. In the expanded `ALL CLASSES` section, start typing `SGGameInstance` in the Search box. When `SGGameInstance` appears, select it and click the `Select` button to create your new Blueprint class based on it.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

A Player Controller is an actor responsible for controlling a Pawn used by the player.

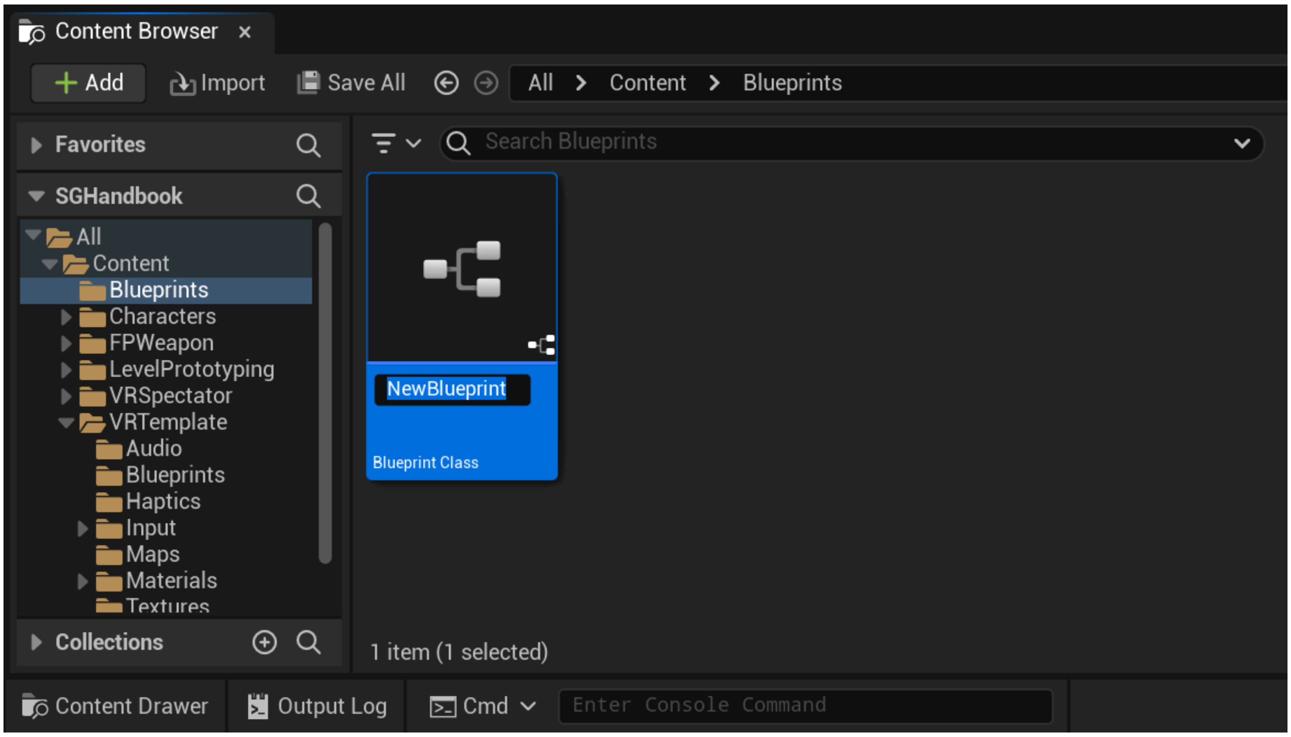
Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

An ActorComponent is a reusable component that can be added to any actor.

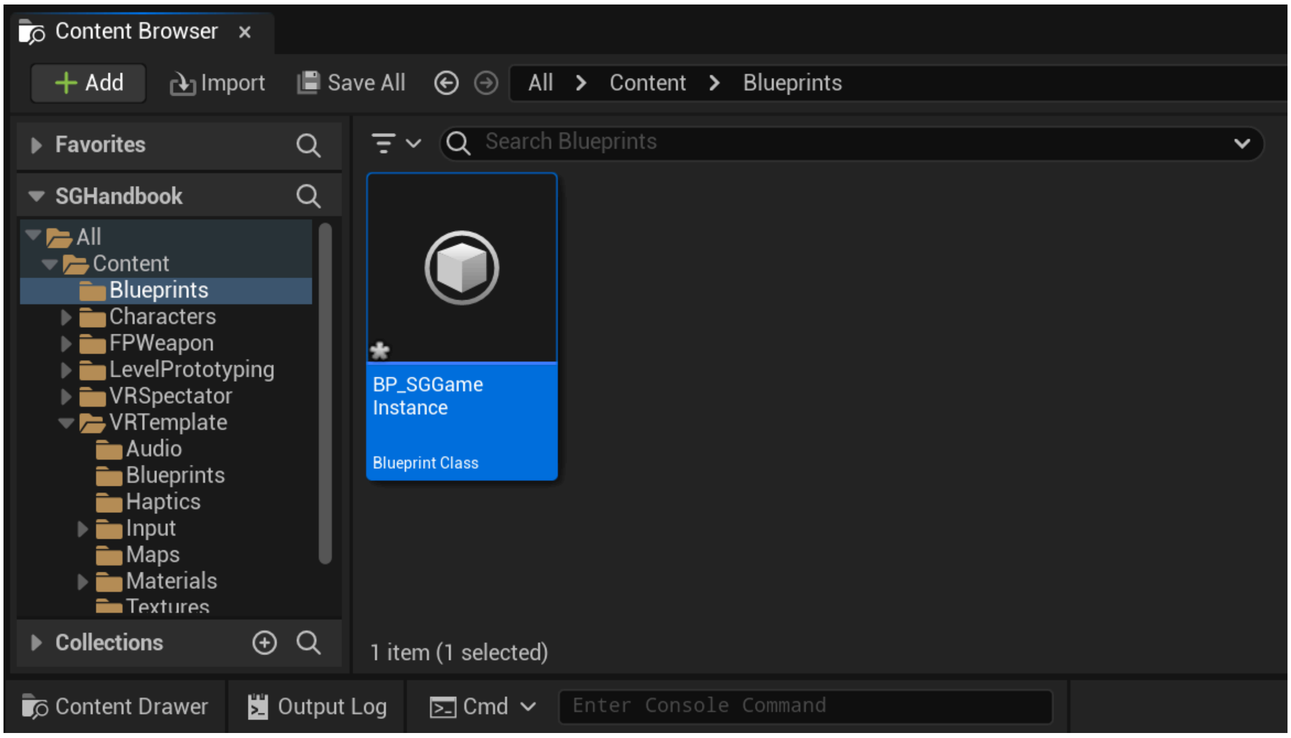
A Scene Component is a component that has a scene transform and can be attached to other scene ...

3 items (1 selected)

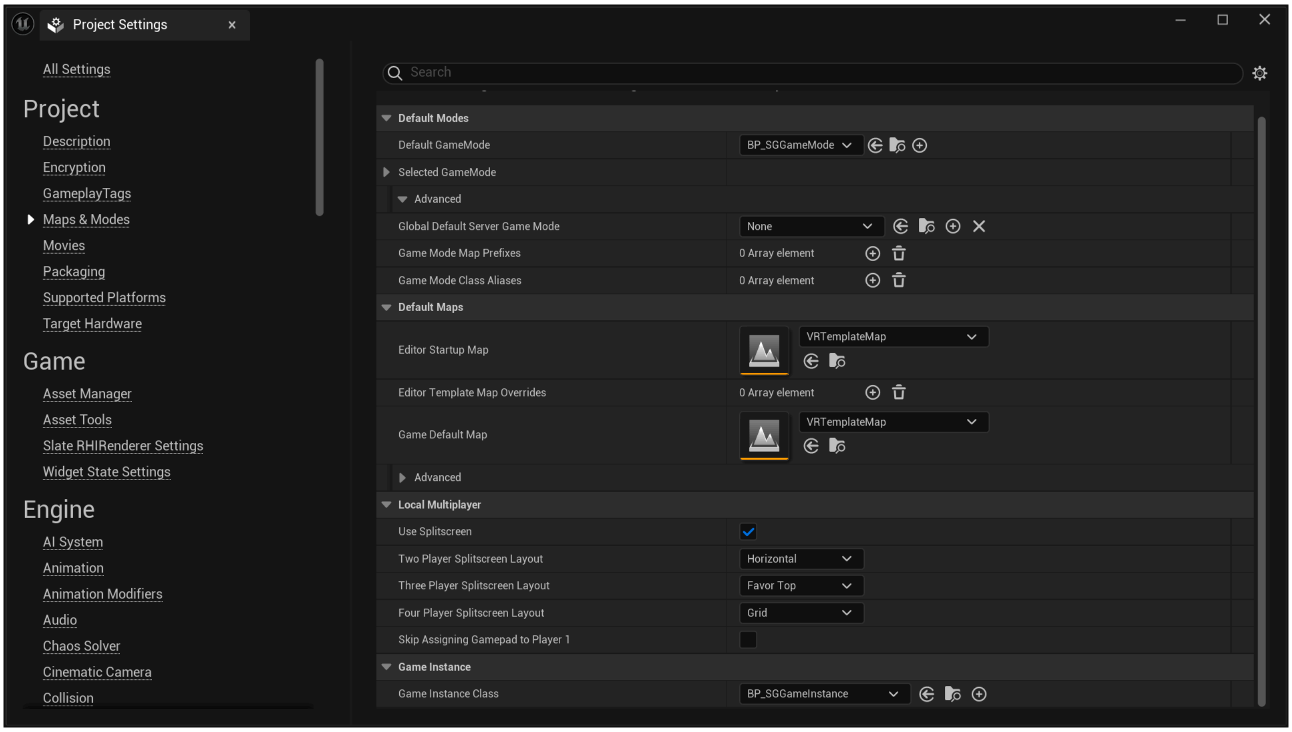
4. After returning to the Content Browser, the Unreal Editor will prompt you to rename `NewBlueprint` to your desired class name. You can rename the class at any time by pressing `F2` or by right-clicking on it and selecting `Rename` from the context menu.



5. Once you have renamed the `NewBlueprint` class to your desired name, click on `Save All` to save the new class to disk.



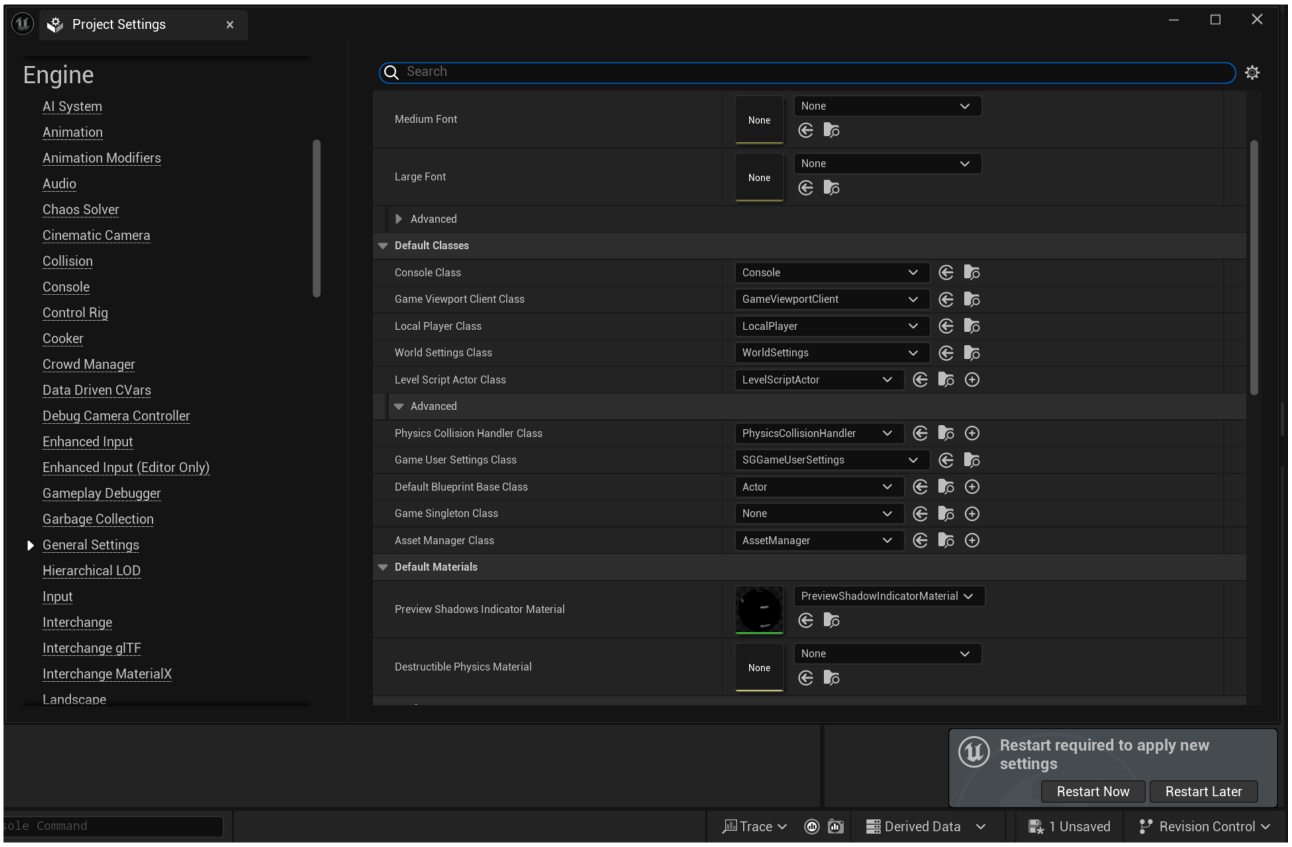
6. Finally, set your newly created subclass of `SGGameInstance` as the default `Game Instance Class`. You can do this by navigating to `Project Settings > Project > Maps & Modes > Game Instance > Game Instance Class`.



Default Modes
Default GameMode: BP_SGGameMode
Selected GameMode
Advanced
Global Default Server Game Mode: None
Game Mode Map Prefixes: 0 Array element
Game Mode Class Aliases: 0 Array element
Default Maps
Editor Startup Map: VRTemplateMap
Editor Template Map Overrides: 0 Array element
Game Default Map: VRTemplateMap
Advanced
Local Multiplayer
Use Splitscreen: ✓
Two Player Splitscreen Layout: Horizontal
Three Player Splitscreen Layout: Favor Top
Four Player Splitscreen Layout: Grid
Skip Assigning Gamepad to Player 1: ☐
Game Instance
Game Instance Class: BP_SGGameInstance

Setting Up SGGameUserSettings

Setting `SGGameUserSettings` as the default `Game User Settings Class` is very straightforward. You can do this by navigating to `Project Settings > Engine > General Settings > Default Classes > Advanced > Game User Settings Class`. Once you change the default `Game User Settings Class` the Unreal Editor will prompt you with `Restart required to apply new settings`. For the changes to take effect, click on the `Restart Now` button and wait for the editor to reopen.



Medium Font	None
Large Font	None
▸ Advanced	
Default Classes	
Console Class	Console
Game Viewport Client Class	GameViewportClient
Local Player Class	LocalPlayer
World Settings Class	WorldSettings
Level Script Actor Class	LevelScriptActor
▾ Advanced	
Physics Collision Handler Class	PhysicsCollisionHandler
Game User Settings Class	SGGameUserSettings
Default Blueprint Base Class	Actor
Game Singleton Class	None
Asset Manager Class	AssetManager
Default Materials	
Preview Shadows Indicator Material	PreviewShadowIndicatorMaterial
Destructible Physics Material	None

Restart required to apply new settings

Restart Now | Restart Later

Tip

For greater control and customization, consider extending the SGGameUserSettings.

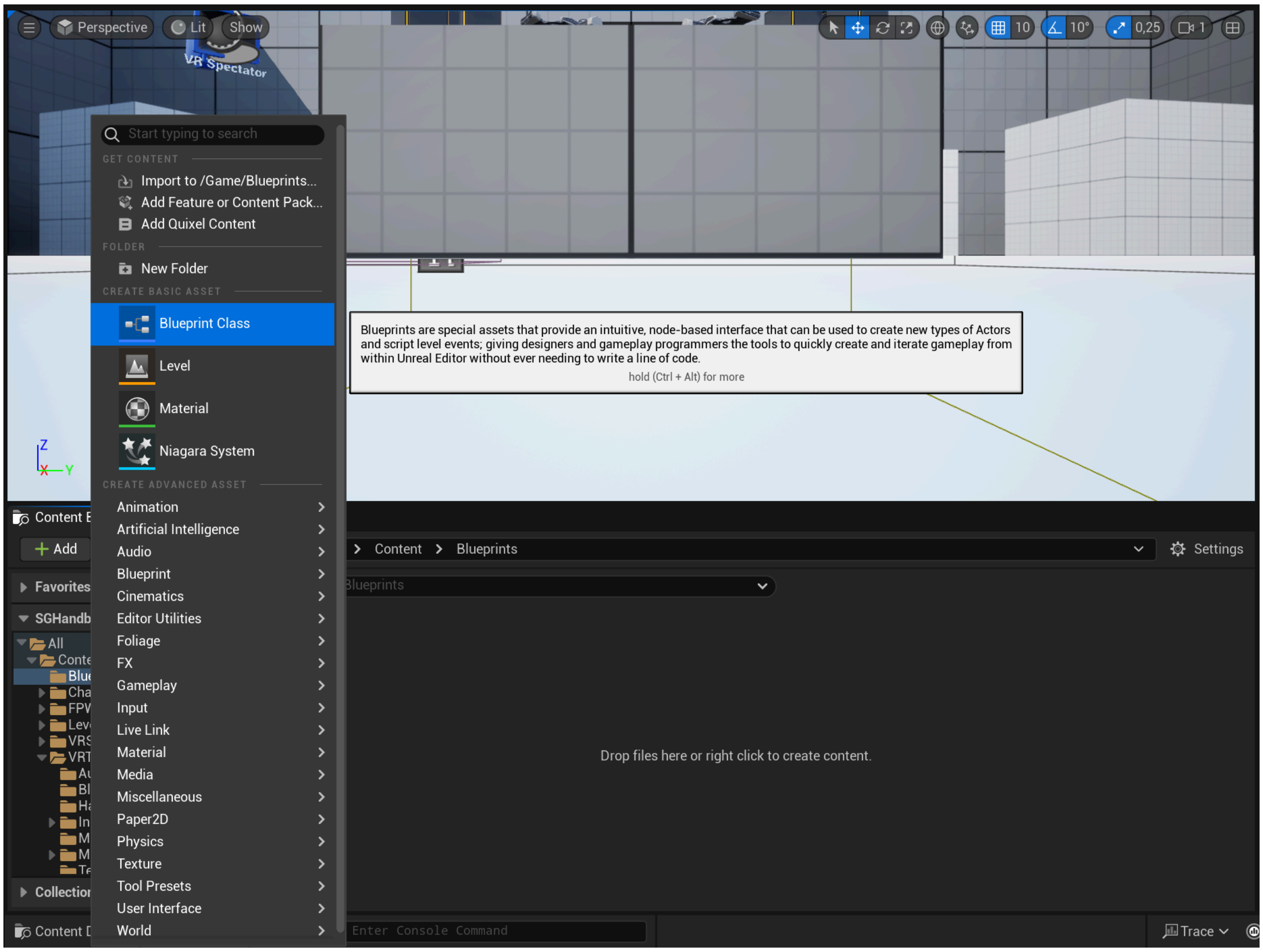
Important

> Currently, setting `SGGameUserSettings` or a subclass of it as the default `Game User Settings Class` is not a strict requirement. However, if you intend to use any SGGameUserSettings-related SenseGlove console command it becomes mandatory. If not set, calling any SGGameUserSettings-related SenseGlove console command will cause your simulation or editor to crash.

Extending SGGameUserSettings

Follow these steps to extend and set up your own version of `SGGameUserSettings`:

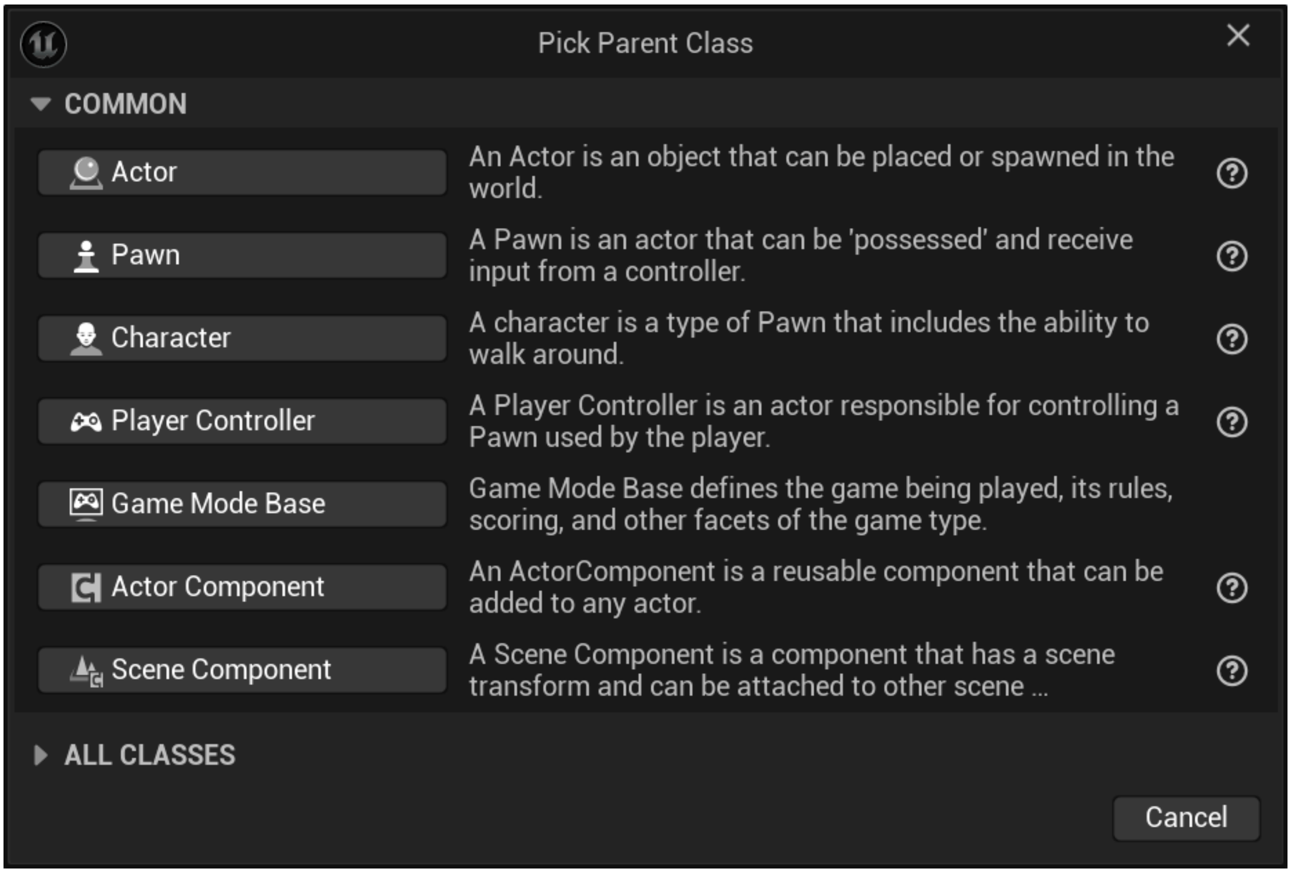
1. In the Content Browser, click the `+ Add` button, then select `Blueprint Class from the menu`. Alternatively, right-click inside the Content Browser and choose `Blueprint Class` from the context menu.



Blueprints are special assets that provide an intuitive, node-based interface that can be used to create new types of Actors and script level events; giving designers and gameplay programmers the tools to quickly create and iterate gameplay from within Unreal Editor without ever needing to write a line of code.
hold (Ctrl + Alt) for more

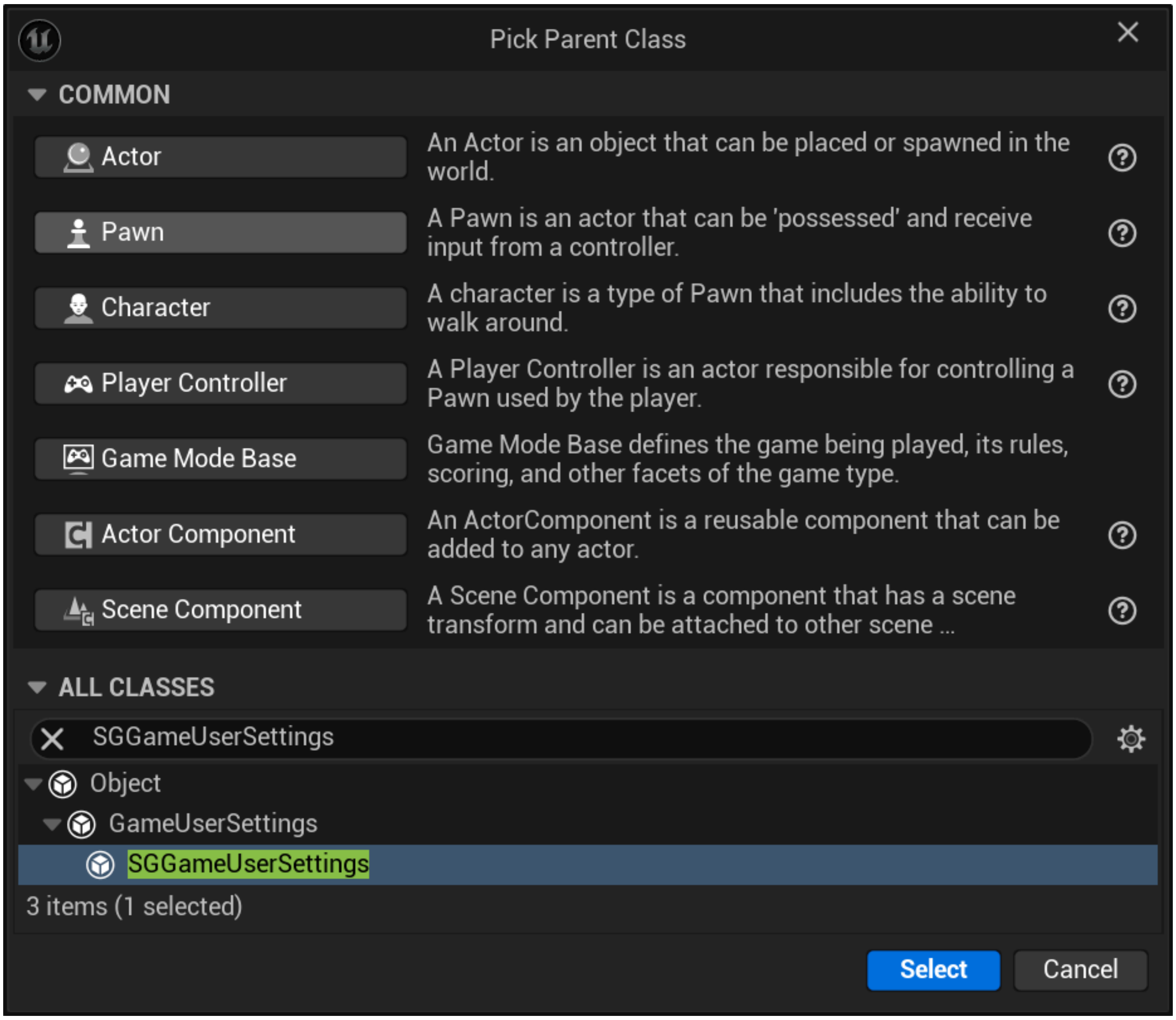
Drop files here or right click to create content.

2. A dialog will appear asking you to choose a parent class. Click on the `ALL CLASSES` section to expand the list of available classes.



Actor	An Actor is an object that can be placed or spawned in the world.
Pawn	A Pawn is an actor that can be 'possessed' and receive input from a controller.
Character	A character is a type of Pawn that includes the ability to walk around.
Player Controller	A Player Controller is an actor responsible for controlling a Pawn used by the player.
Game Mode Base	Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.
Actor Component	An ActorComponent is a reusable component that can be added to any actor.
Scene Component	A Scene Component is a component that has a scene transform and can be attached to other scene ...

3. In the expanded `ALL CLASSES` section, start typing `SGGameUserSettings` in the Search box. When `SGGameUserSettings` appears, select it and click the `Select` button to create your new Blueprint class based on it.



An Actor is an object that can be placed or spawned in the world.

A Pawn is an actor that can be 'possessed' and receive input from a controller.

A character is a type of Pawn that includes the ability to walk around.

A Player Controller is an actor responsible for controlling a Pawn used by the player.

Game Mode Base defines the game being played, its rules, scoring, and other facets of the game type.

An ActorComponent is a reusable component that can be added to any actor.

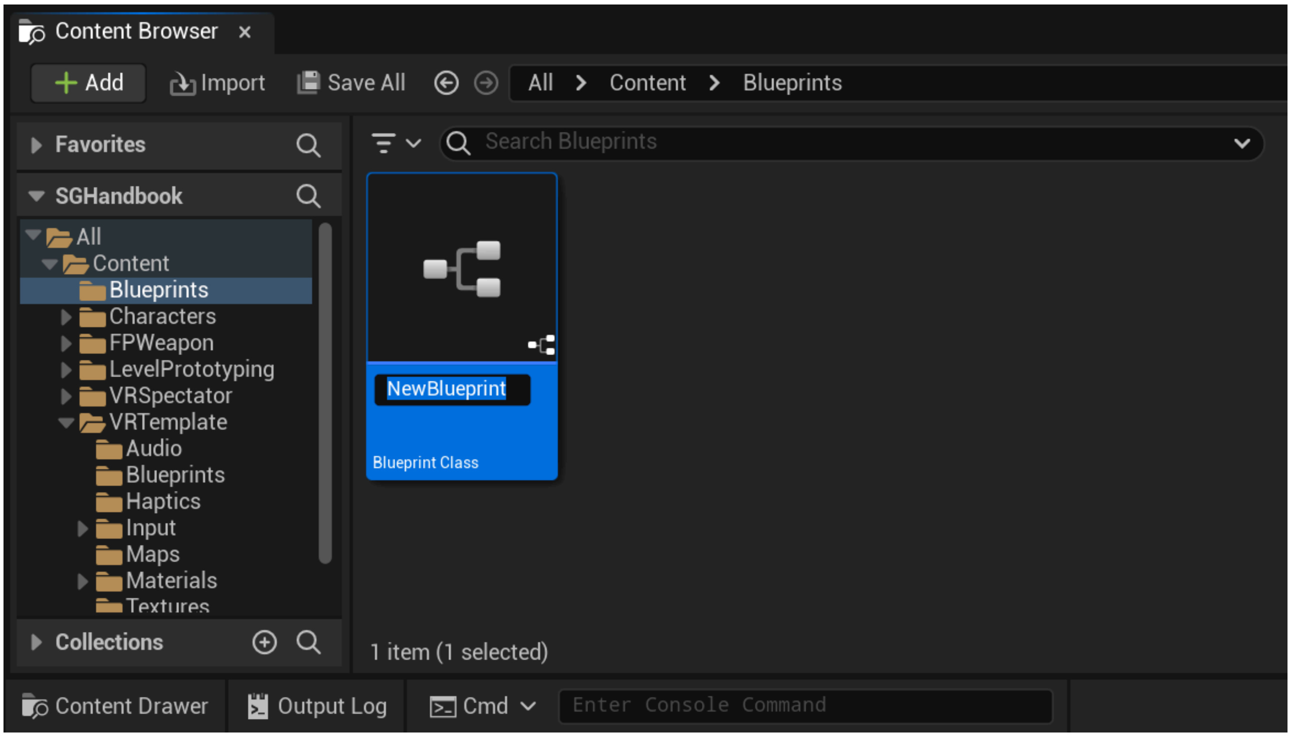
A Scene Component is a component that has a scene transform and can be attached to other scene ...

SGGameUserSettings

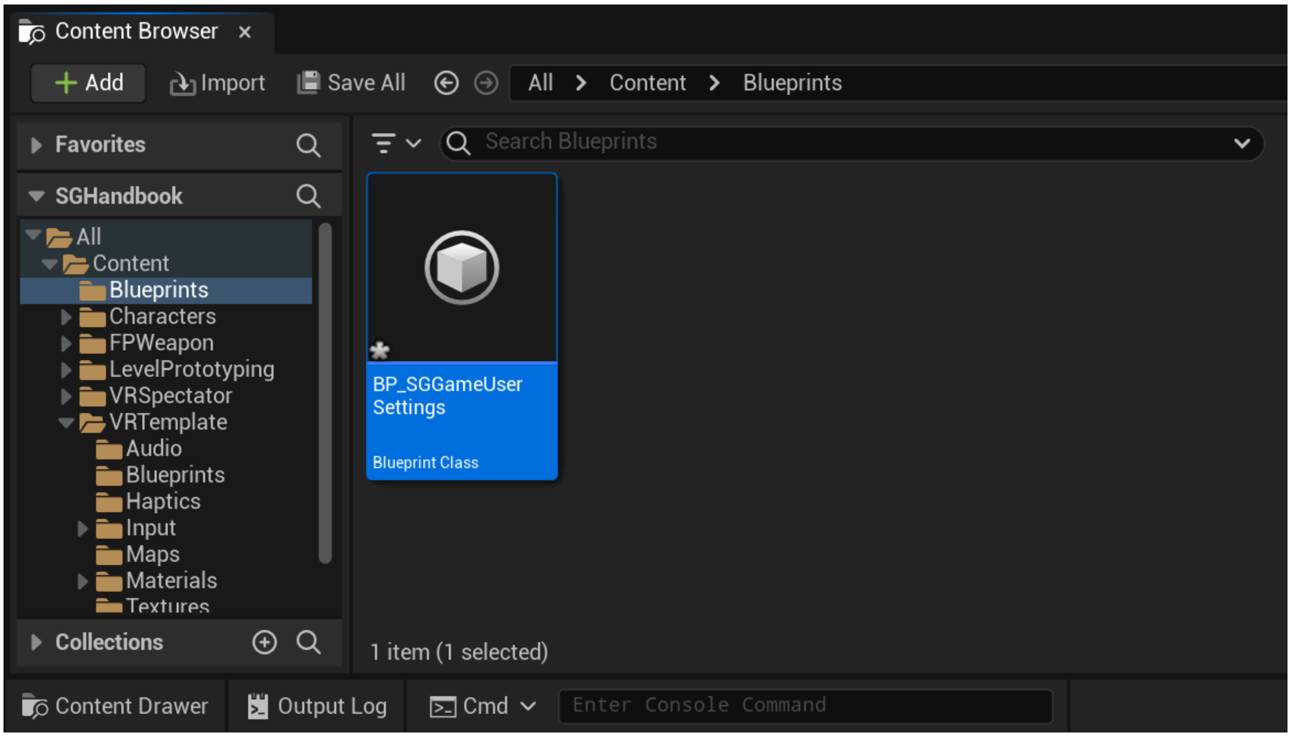
- Object
  - GameUserSettings
    - SGGameUserSettings

3 items (1 selected)

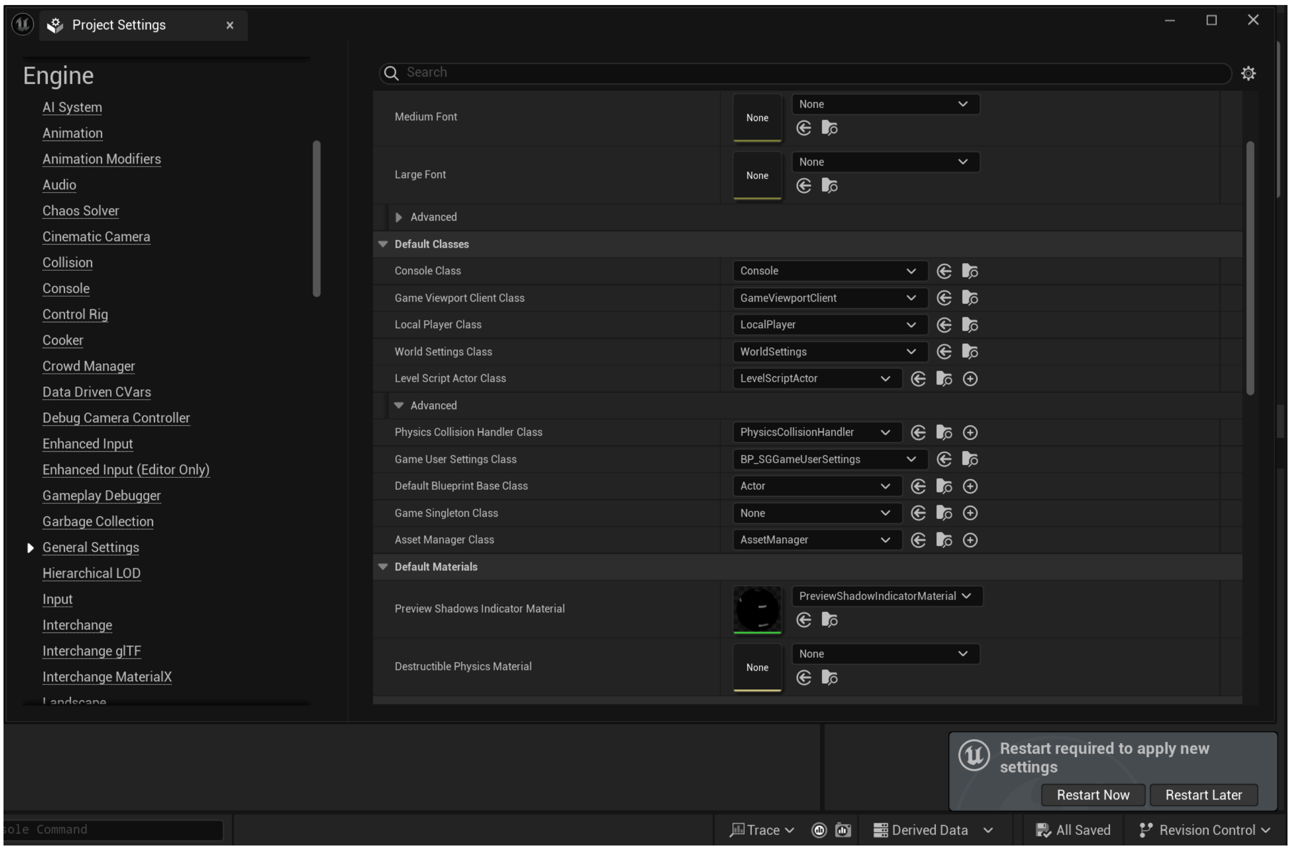
4. After returning to the Content Browser, the Unreal Editor will prompt you to rename `NewBlueprint` to your desired class name. You can rename the class at any time by pressing `F2` or by right-clicking on it and selecting `Rename` from the context menu.



5. Once you have renamed the `NewBlueprint` class to your desired name, click on `Save All` to save the new class to disk.



6. Finally, set your newly created subclass of `SGGameUserSettings` as the default `Game User Settings Class`. You can do this by navigating to `Project Settings > Engine > General Settings > Default Classes > Advanced > Game User Settings Class`. Once you change the default `Game User Settings Class` the Unreal Editor will prompt you with `Restart required to apply new settings`. For the changes to take effect, click on the `Restart Now` button and wait for the editor to reopen.


Project Settings

Engine
- AI System
- Animation
- Animation Modifiers
- Audio
- Chaos Solver
- Cinematic Camera
- Collision
- Console
- Control Rig
- Cooker
- Crowd Manager
- Data Driven CVars
- Debug Camera Controller
- Enhanced Input
- Enhanced Input (Editor Only)
- Gameplay Debugger
- Garbage Collection
- General Settings
- Hierarchical LOD
- Input
- Interchange
- Interchange glTF
- Interchange MaterialX
- Landscape

Medium Font	None
Large Font	None
Advanced	
Default Classes	
Console Class	Console
Game Viewport Client Class	GameViewportClient
Local Player Class	LocalPlayer
World Settings Class	WorldSettings
Level Script Actor Class	LevelScriptActor
Advanced	
Physics Collision Handler Class	PhysicsCollisionHandler
Game User Settings Class	BP_SGGameUserSettings
Default Blueprint Base Class	Actor
Game Singleton Class	None
Asset Manager Class	AssetManager
Default Materials	
Preview Shadows Indicator Material	PreviewShadowIndicatorMaterial
Destructible Physics Material	None

Restart required to apply new settings

Restart Now | Restart Later

Setting Up the Virtual Hand Meshes

Setting up Virtual Hand Meshes involves two key steps:

1. Importing the virtual hand meshes into your project.
2. Configuring the virtual hand settings.

In this section we focus on the first part. For detailed information on step two, please visit the Virtual Hand configuration section.

Compatible Virtual Hand Meshes

The SenseGlove Unreal Engine Plugin is compatible with any virtual hand mesh that adheres to the Epic rig and bone structure. Additionally, the virtual hand meshes must be exported with specific settings to meet all requirements. If you're planning to model and rig your own virtual hand meshes, the Epic FBX Skeletal Mesh Pipeline is a useful starting point.

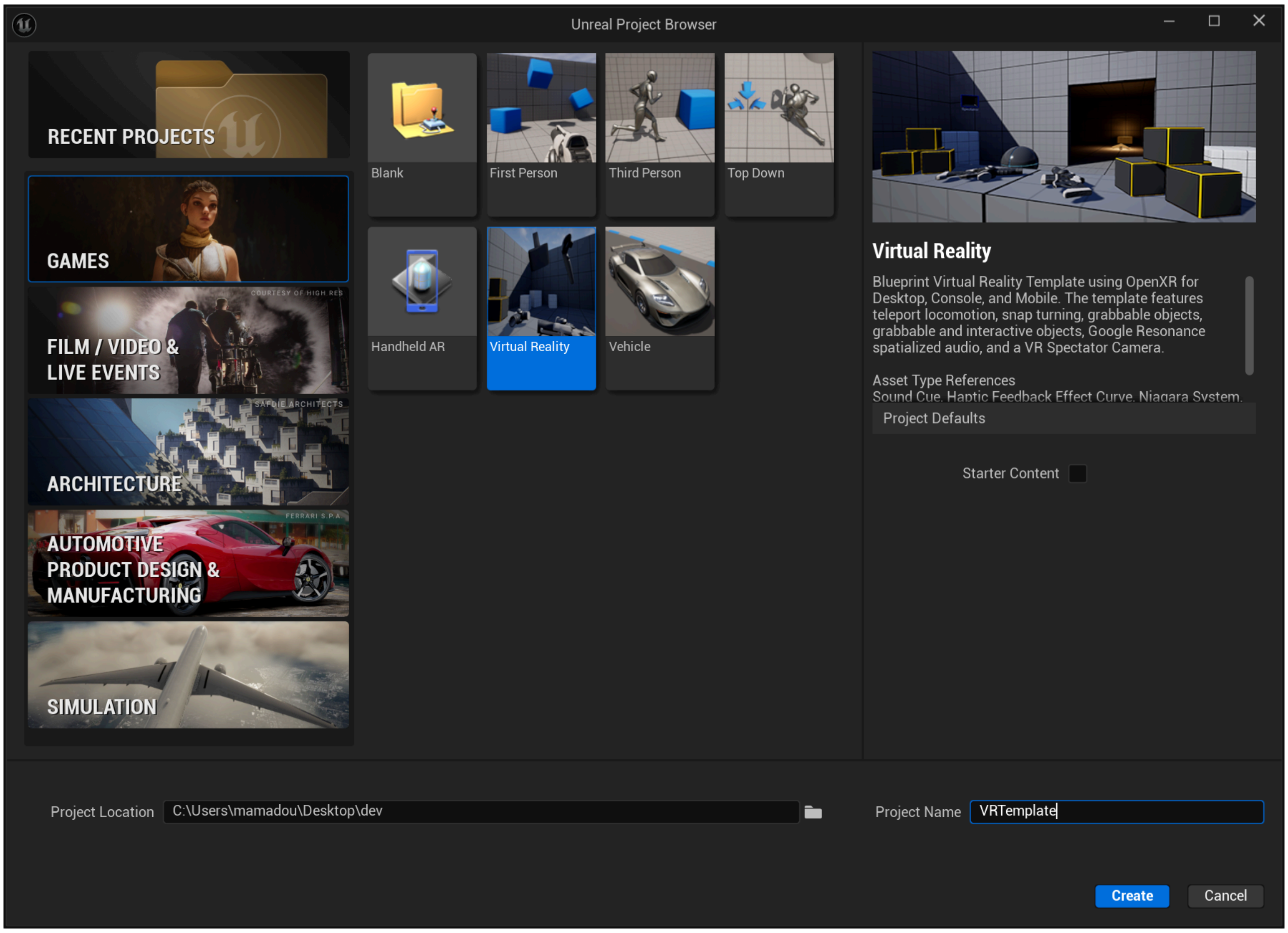
However, if you're looking to get up and running with the SenseGlove Unreal Engine Plugin quickly, the process is much simpler. Unreal Engine has included two sets of compatible virtual hand models with the Unreal Engine VR Template since version `5.1`. This guide will walk you through how to export these virtual hand models from the VR Template and import them into your VR simulation.

> **Caution**
>
> While it is possible to migrate the virtual hand meshes directly from the Content Browser of the VR Template, this approach is not recommended. As part of the setup process, it is necessary to configure the SenseGlove Grab and Touch sockets. Although it's possible to set up these sockets manually, as demonstrated in one of our older tutorials, we no longer recommend doing so. Since version `v2.1.0` of the SenseGlove Unreal Engine Plugin, we've included a tool that automates the socket setup with a single click, eliminating the need for the tedious manual process.

> Unfortunately, the SenseGlove Sockets Editor tool does not support skeletal meshes that share their skeleton. This is the case with the hand models included in the VR Template. Because of this limitation, we will be reimporting the virtual hand meshes with separate skeletons to ensure full compatibility with the SenseGlove Sockets Editor.

Exporting the Virtual Hand Meshes from the VRTemplate

1. Start by creating a new Unreal Engine project using the VR Template. In the Unreal Project Browser, select `GAMES > Virtual Reality`.



Virtual Reality

Blueprint Virtual Reality Template using OpenXR for Desktop, Console, and Mobile. The template features teleport locomotion, snap turning, grabbable objects, grabbable and interactive objects, Google Resonance spatialized audio, and a VR Spectator Camera.

Asset Type References
Sound Cue, Haptic Feedback Effect Curve, Niagara System,

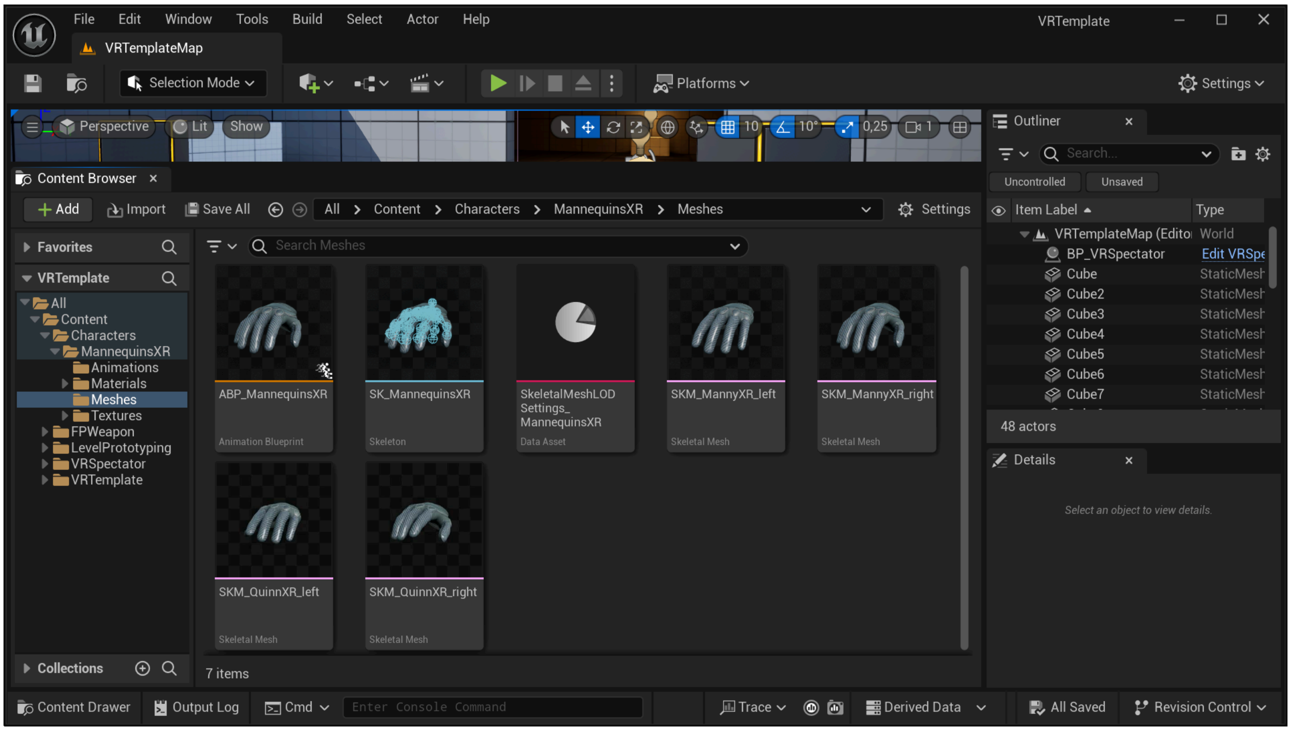
Starter Content

Project Location C:\Users\mamadou\Desktop\dev

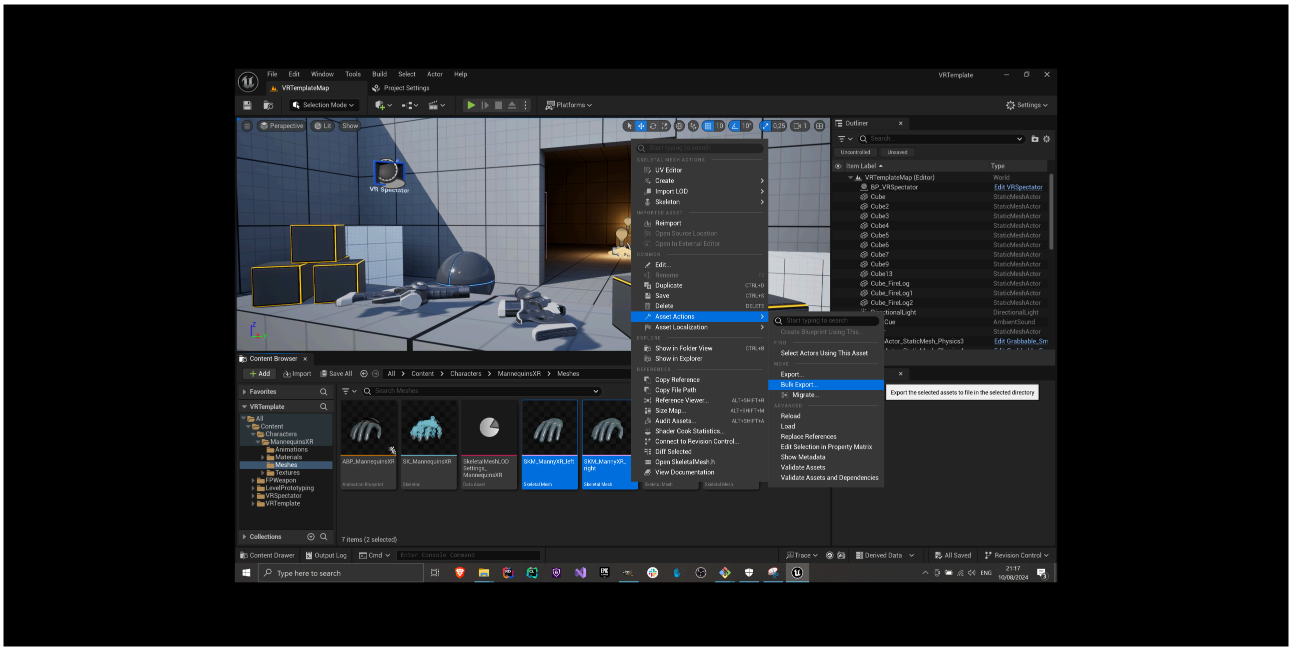
Project Name VRTemplate

2. Once the Unreal Editor opens with your new project, navigate to the Content Browser. Go to `All > Content > Characters > MannequinsXR > Meshes`. Here,

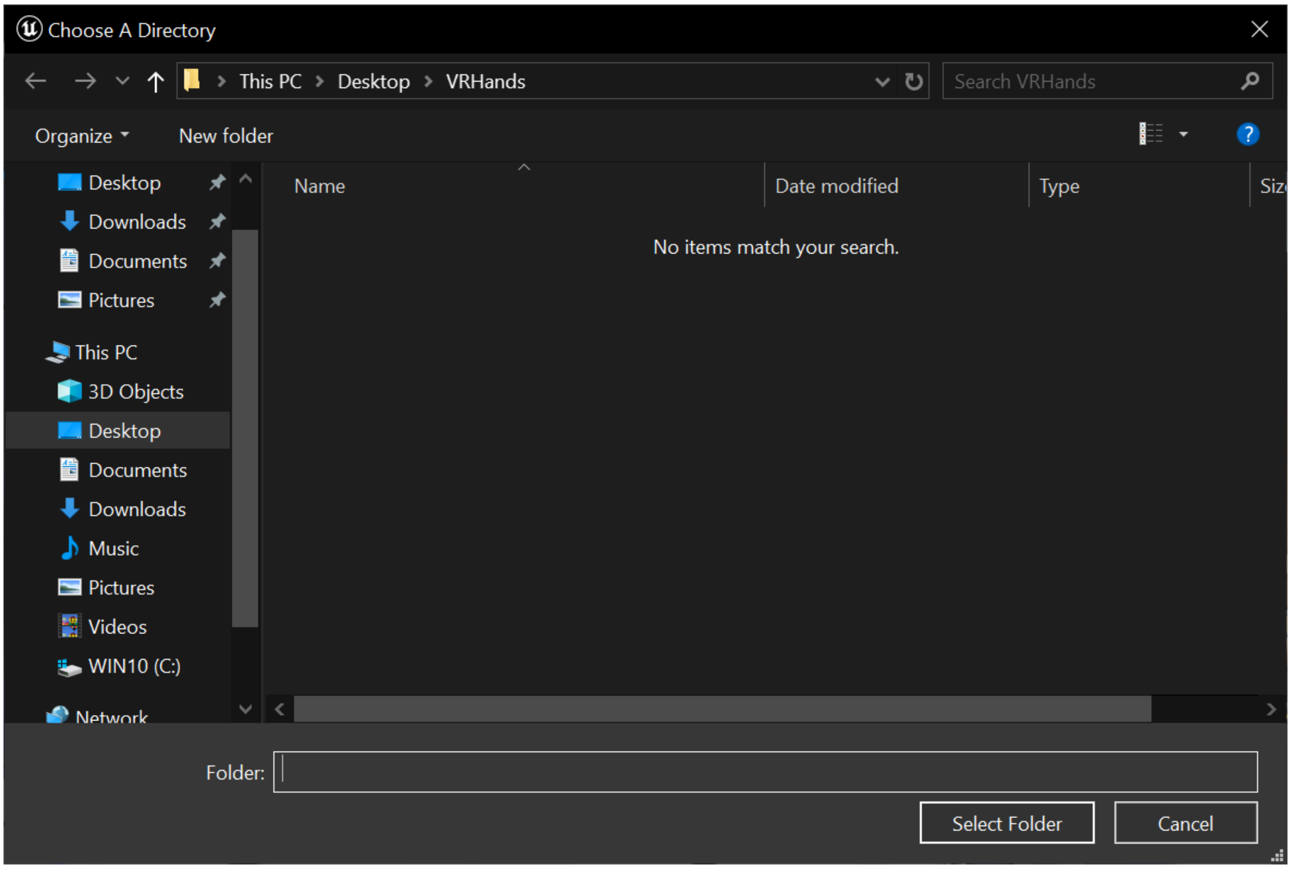
you'll find two sets of virtual hand meshes: `SDKM_MannyXR_left` and `SDKM_MannyXR_right` (male hands), and `SDKM_QuinnXR_left` and `SDKM_QuinnXR_right` (female hands).



3. Choose the pair of hand meshes you want to export. Right-click on them, then select `Asset Actions` followed by `Bulk Export...` from the context menu.

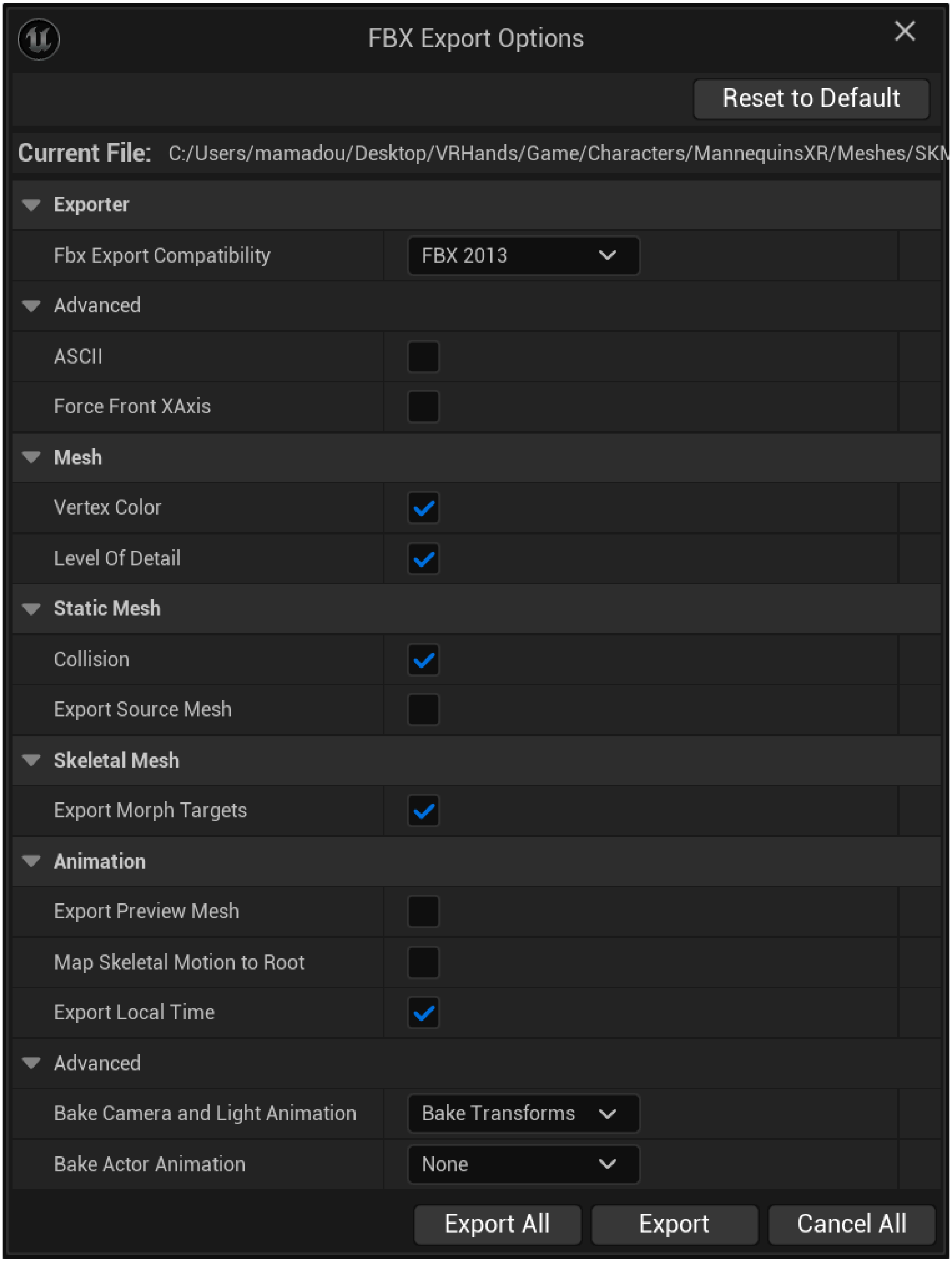


4. In the file dialog that appears, choose a folder to save the exported hands, and click the `Select Folder` button to export the meshes in FBX format.



No items match your search.

5. The Unreal Editor will then display the `FBX Export Options` dialog. Leave the default settings unchanged and click `Export All` to proceed.



FBX Export Options

Reset to Default

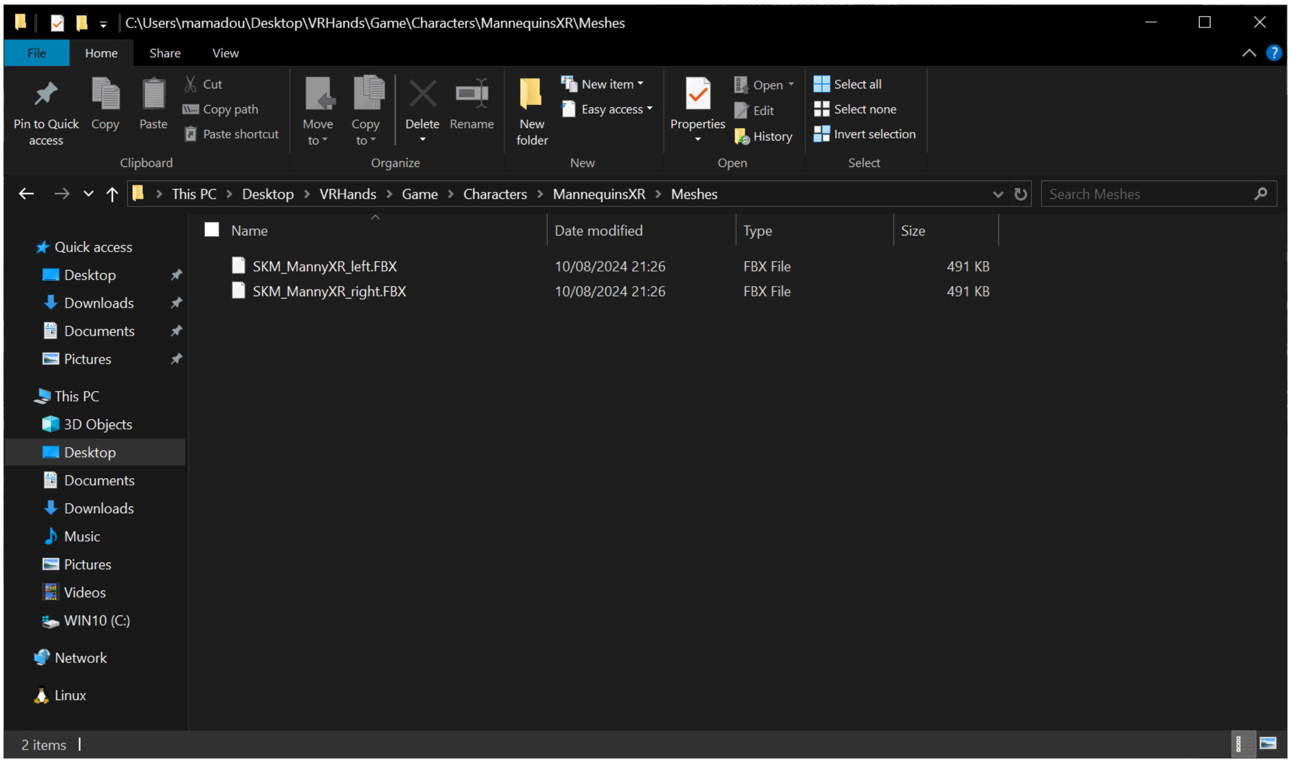
Current File: C:/Users/mamadou/Desktop/VRHands/Game/Characters/MannequinsXR/Meshes/SKM

Exporter	
Fbx Export Compatibility	FBX 2013
Advanced	
ASCII	☐
Force Front XAxis	☐
Mesh	
Vertex Color	✓
Level Of Detail	✓
Static Mesh	
Collision	✓
Export Source Mesh	☐
Skeletal Mesh	
Export Morph Targets	✓
Animation	
Export Preview Mesh	☐
Map Skeletal Motion to Root	☐
Export Local Time	✓
Advanced	
Bake Camera and Light Animation	Bake Transforms
Bake Actor Animation	None

Tip

If you're unsure whether the options are set to their defaults, you can click the `Reset to Default` button in the top-right corner of the dialog to restore the default settings.

6. After exporting, you can find the FBX files for both hands in the directory you selected: `/path/you/chose/for/bulk/export/Game/Characters/MannequinsXR/Meshes/` .



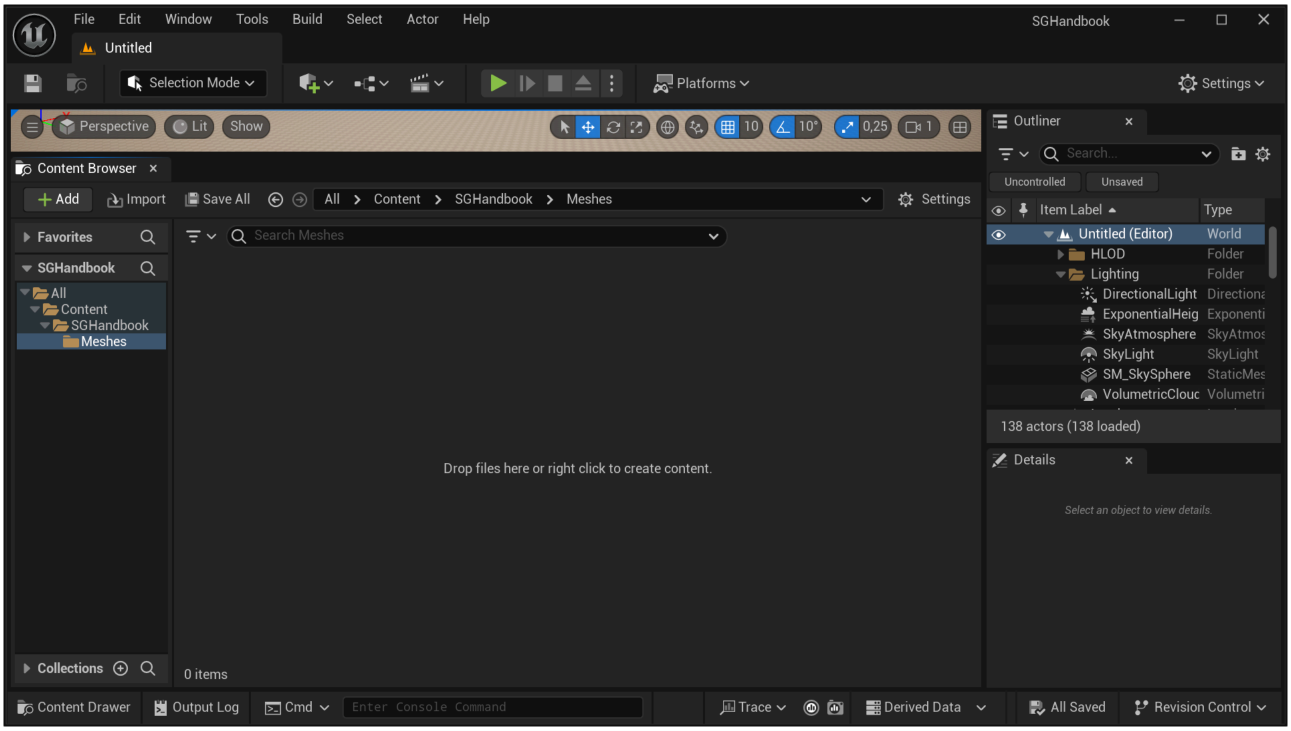
C:\Users\mamadou\Desktop\VRHands\Game\Characters\MannequinsXR\Meshes

Name	Date modified	Type	Size
SKM_MannyXR_left.FBX	10/08/2024 21:26	FBX File	491 KB
SKM_MannyXR_right.FBX	10/08/2024 21:26	FBX File	491 KB

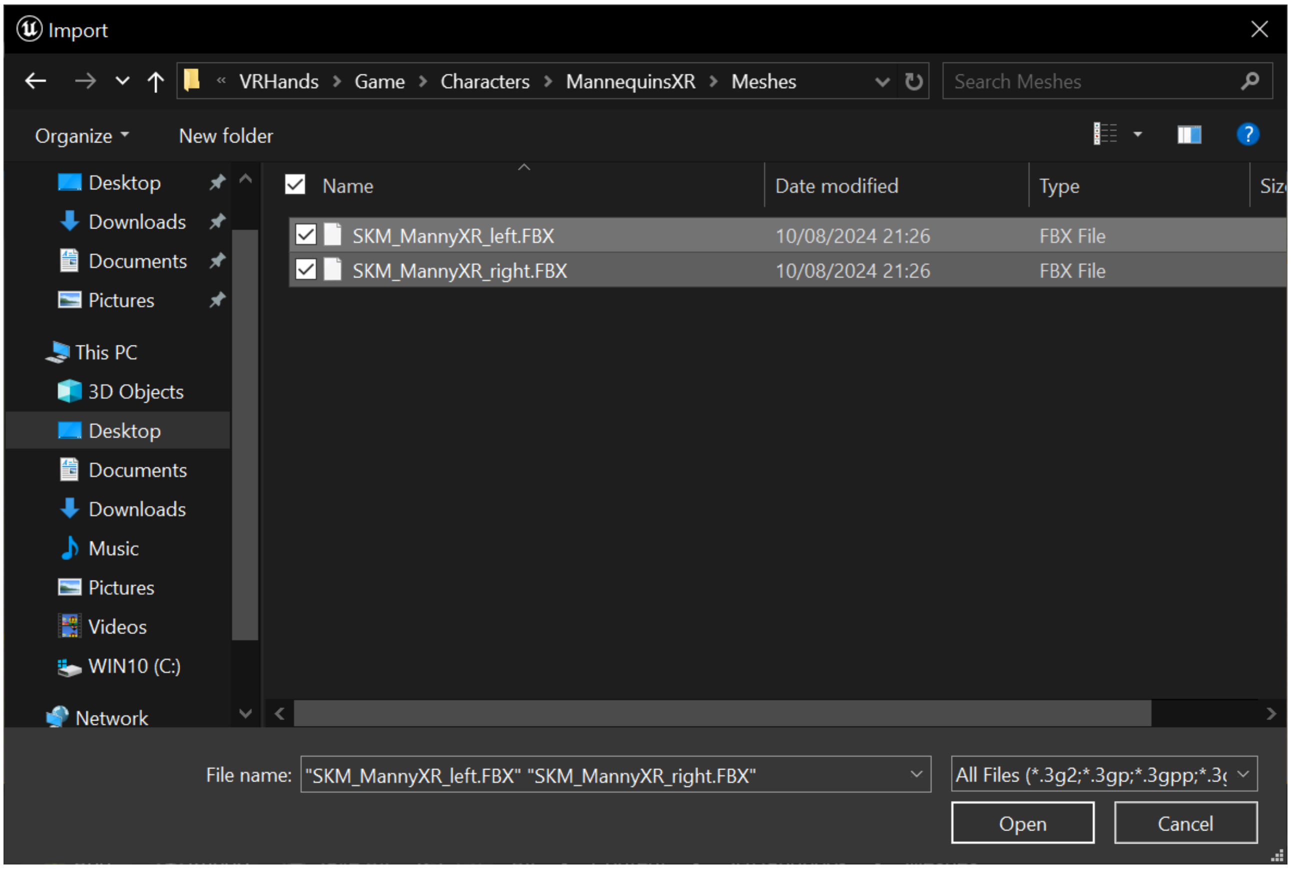
2 items

Importing the Virtual Hand Meshes into Your Own Project

1. Start by creating a new folder inside your project's Content Browser. Navigate to that folder, then press the `Import` button next to the `+ Add` button at the top of the Content Browser.



2. In the `Import` dialog that appears, navigate to the folder containing the virtual hand meshes. Select both FBX files and click the `Open` button.

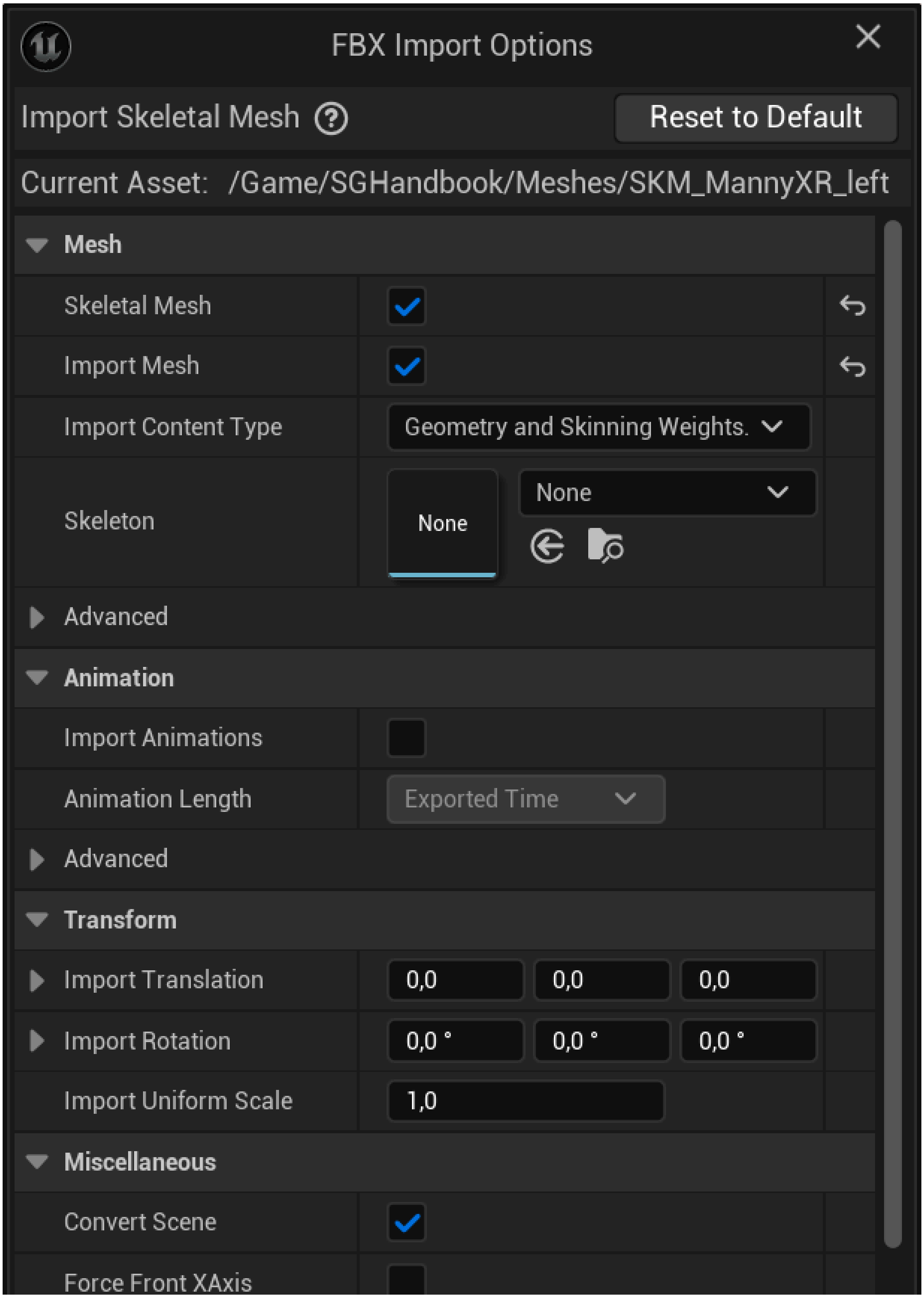


Name	Date modified	Type
SKM_MannyXR_left.FBX	10/08/2024 21:26	FBX File
SKM_MannyXR_right.FBX	10/08/2024 21:26	FBX File

File name: "SKM_MannyXR_left.FBX" "SKM_MannyXR_right.FBX"

All Files (*.3g2;*.3gp;*.3gpp;*.3(

3. The Unreal Editor will display the `FBX Import Options` dialog. Leave the default settings unchanged and click `Import All` to proceed.


FBX Import Options

Import Skeletal Mesh

Reset to Default

Current Asset: /Game/SGHandbook/Meshes/SKM_MannyXR_left

Mesh

Skeletal Mesh	✓
Import Mesh	✓
Import Content Type	Geometry and Skinning Weights.
Skeleton	None
Advanced	

Animation

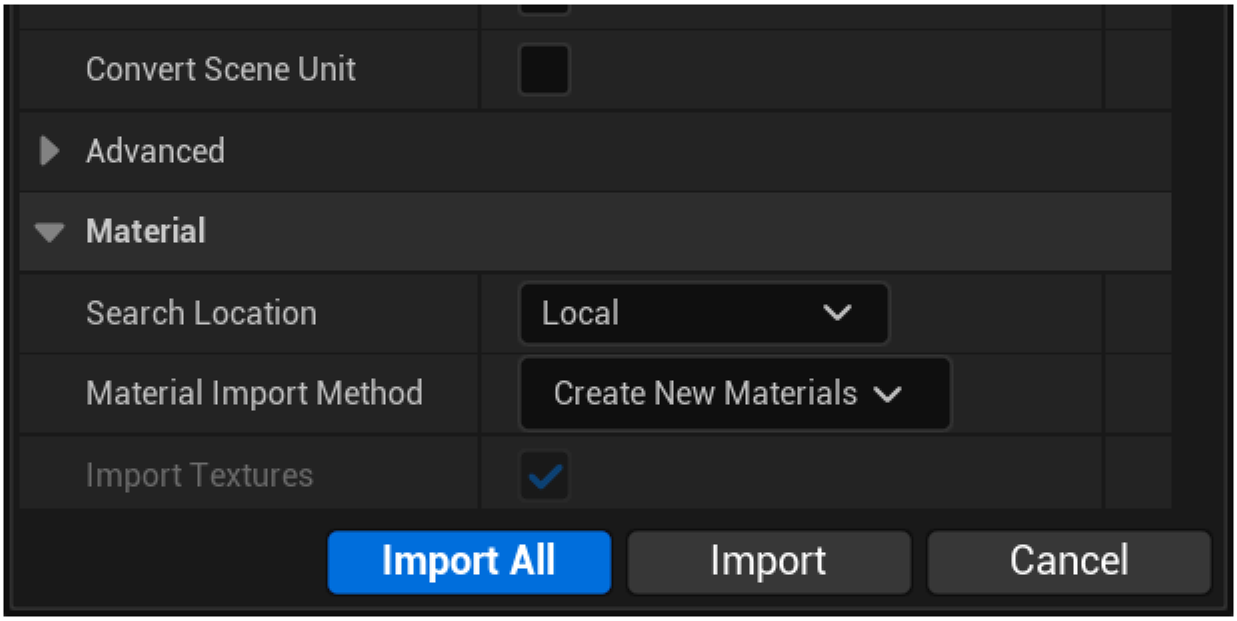
Import Animations	
Animation Length	Exported Time
Advanced	

Transform

Import Translation	0,0	0,0	0,0
Import Rotation	0,0 °	0,0 °	0,0 °
Import Uniform Scale	1,0		

Miscellaneous

Convert Scene	✓
Force Front XAxis	


Convert Scene Unit

Advanced

Material

Search Location: Local

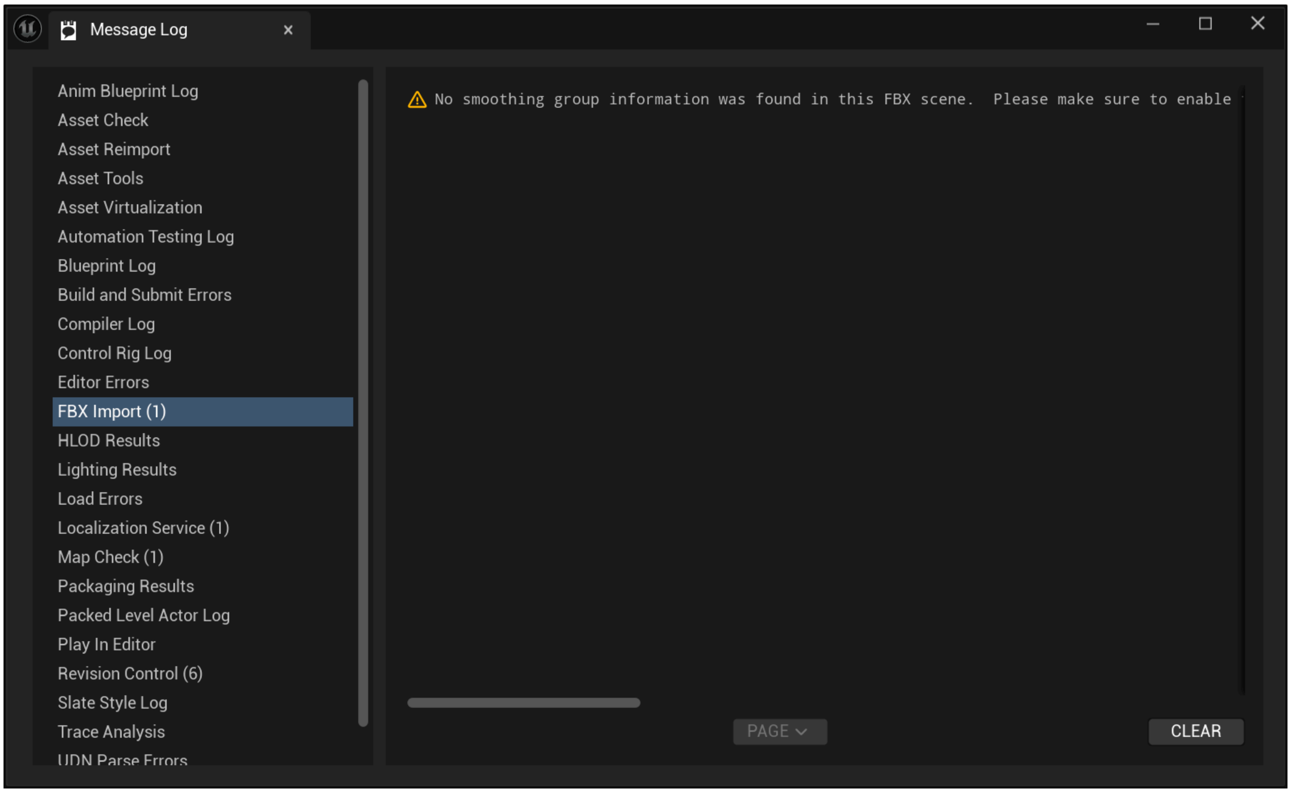
Material Import Method: Create New Materials

Import Textures: ✓

Tip

If you're unsure whether the options are set to their defaults, you can click the `Reset to Default` button in the top-right corner of the dialog to restore the default settings.

4. After the import process is done, a dialog will display the import logs. Any errors or warnings encountered during the import process will be shown here.



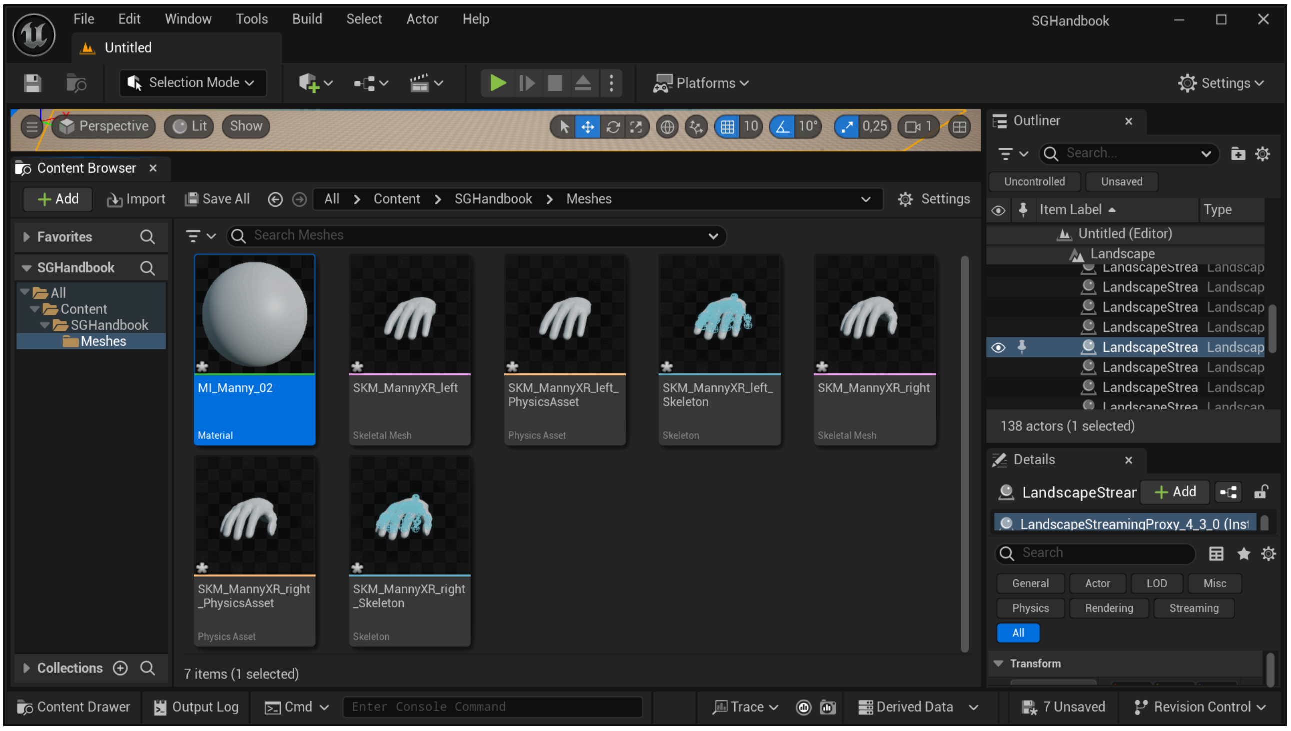
```
⚠ No smoothing group information was found in this FBX scene.  Please make sure to enable
```

Note

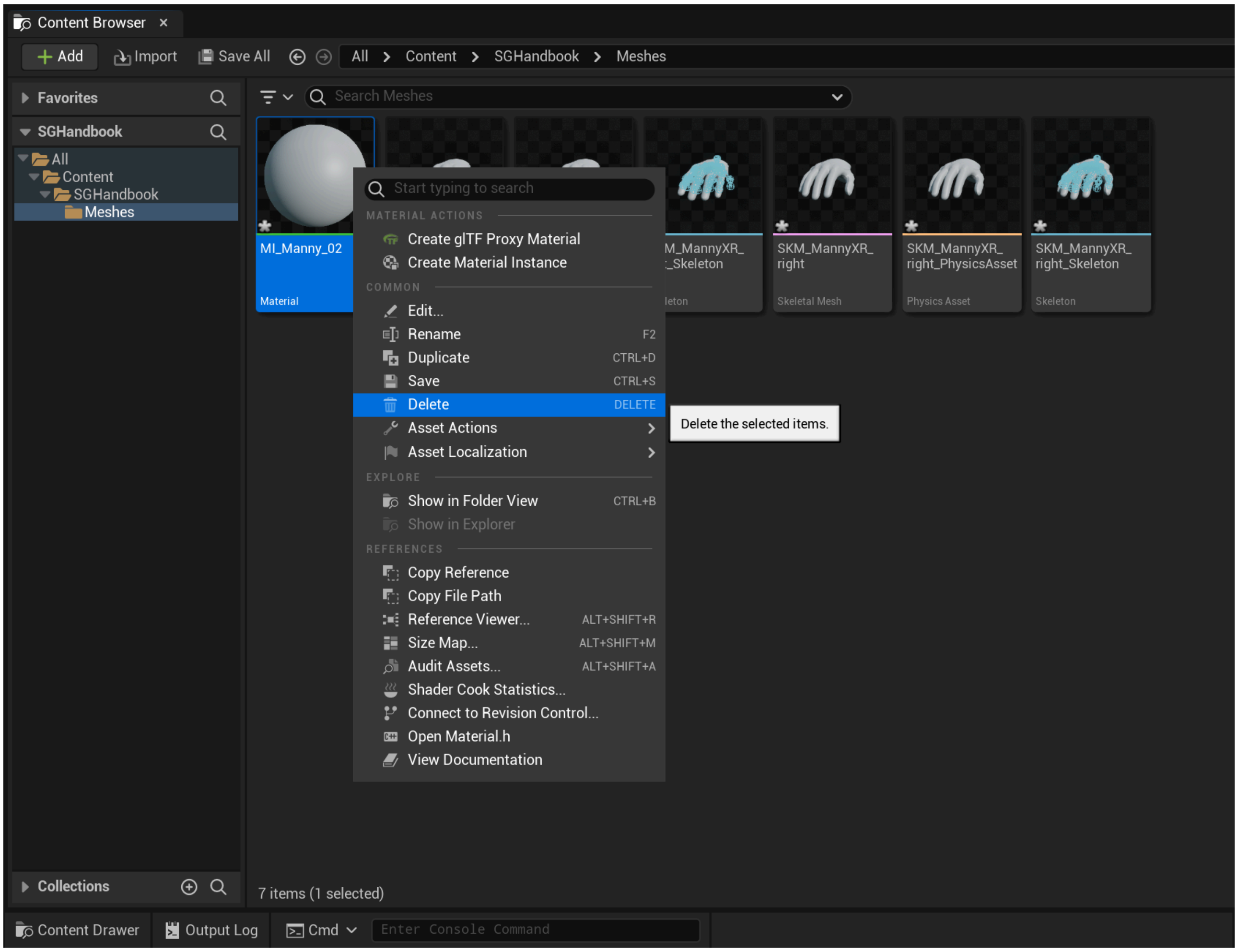
The following warning can be safely ignored:

```
FBXImport: Warning: No smoothing group information was found in this FBX scene.  Please make sure to enable the 'Export Smoothing Groups' option in the FBX Exporter plug-in before exporting the file.  Even for tools that don't support smoothing groups, the FBX Exporter will generate appropriate smoothing data at export-time so that correct vertex normals can be inferred while importing.
```

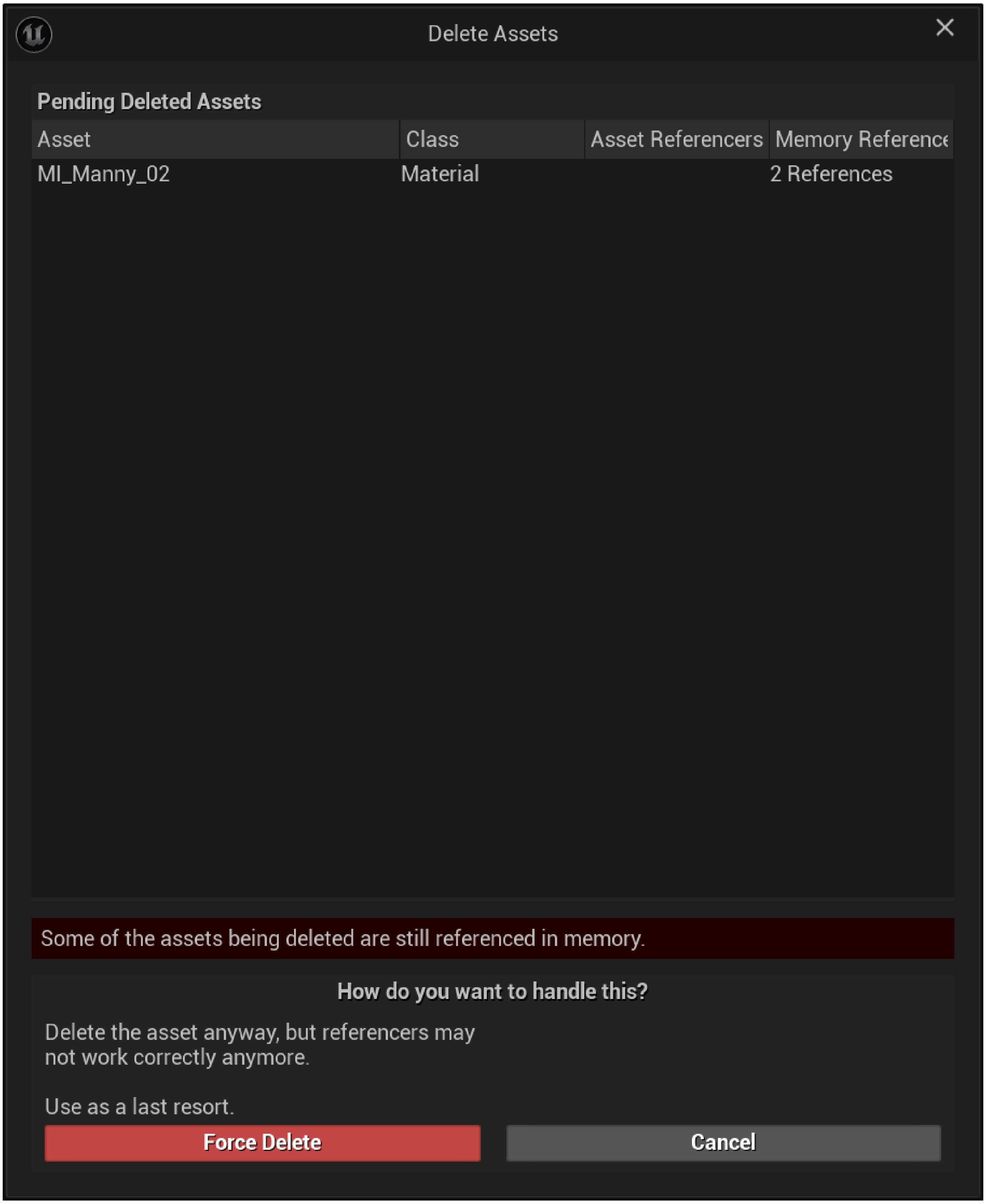
5. The imported virtual hand meshes should now appear in the folder you selected in the Content Browser. Unreal Engine will create a Skeletal Mesh, a Skeleton, and a Physics Asset for each imported mesh, along with a default Material asset shared between both virtual hand meshes.



6. You can choose to keep or modify the default material. However, since the SenseGlove Unreal Engine Plugin provides a default material, we choose to delete the default material created by Unreal Engine during the import process. We'll assign the SenseGlove default material to the imported virtual hand meshes in the next steps. Right-click on the default material and select `Delete`.



7. In the `Delete Assets` dialog, click `Force Delete` to confirm the deletion of the default material.



Pending Deleted Assets

Asset	Class	Asset Referencers	Memory Reference
MI_Manny_02	Material		2 References

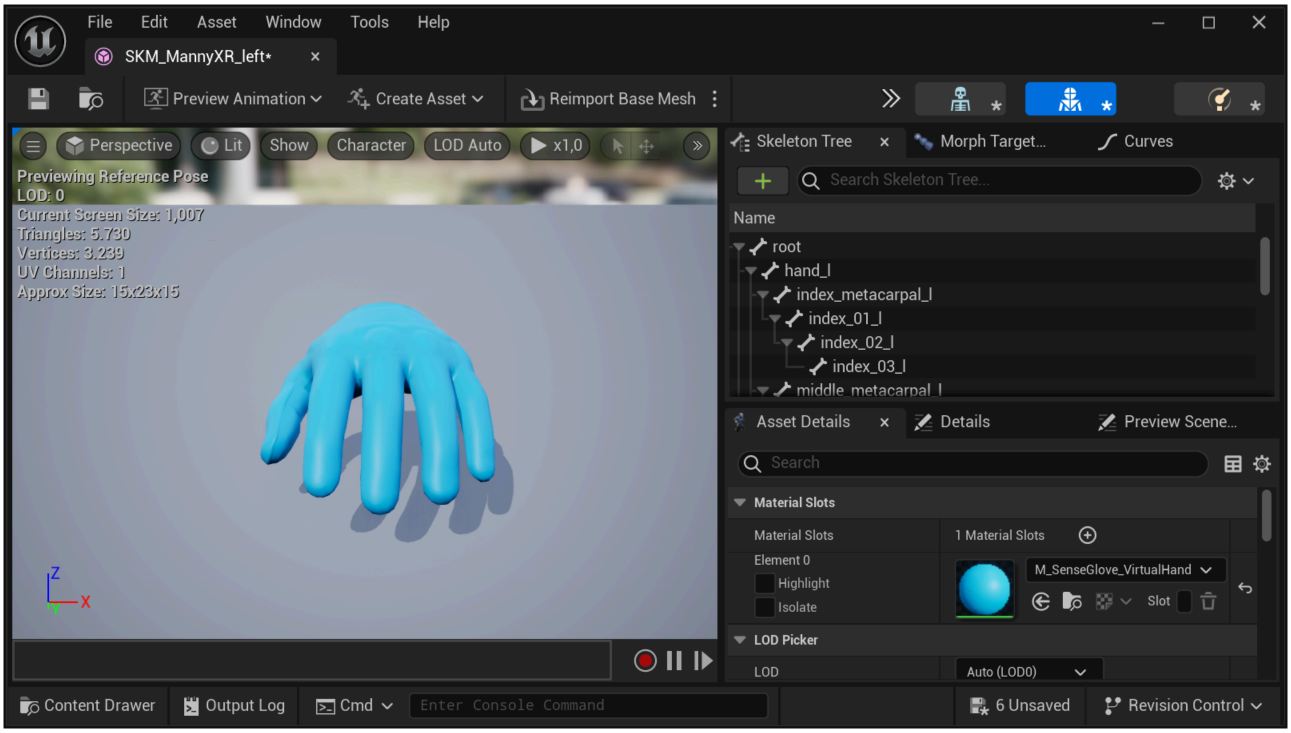
Some of the assets being deleted are still referenced in memory.

How do you want to handle this?

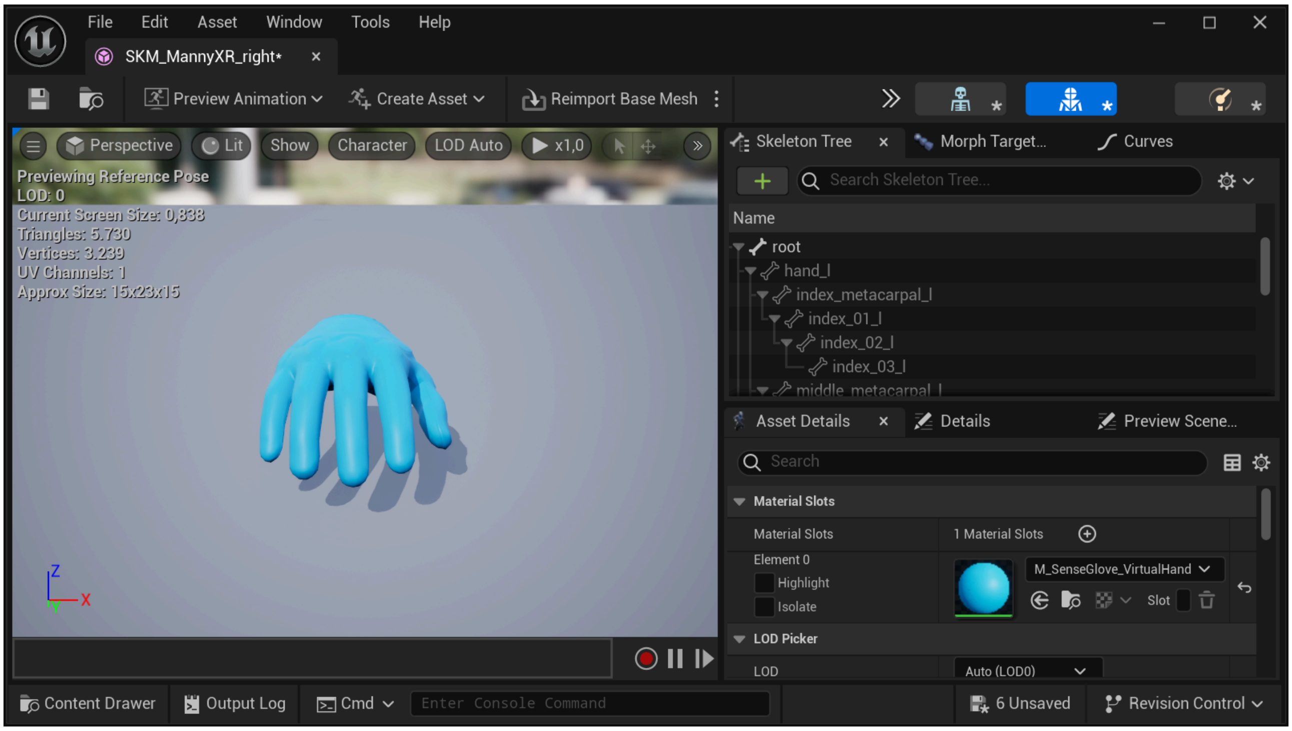
Delete the asset anyway, but referencers may not work correctly anymore.

Use as a last resort.

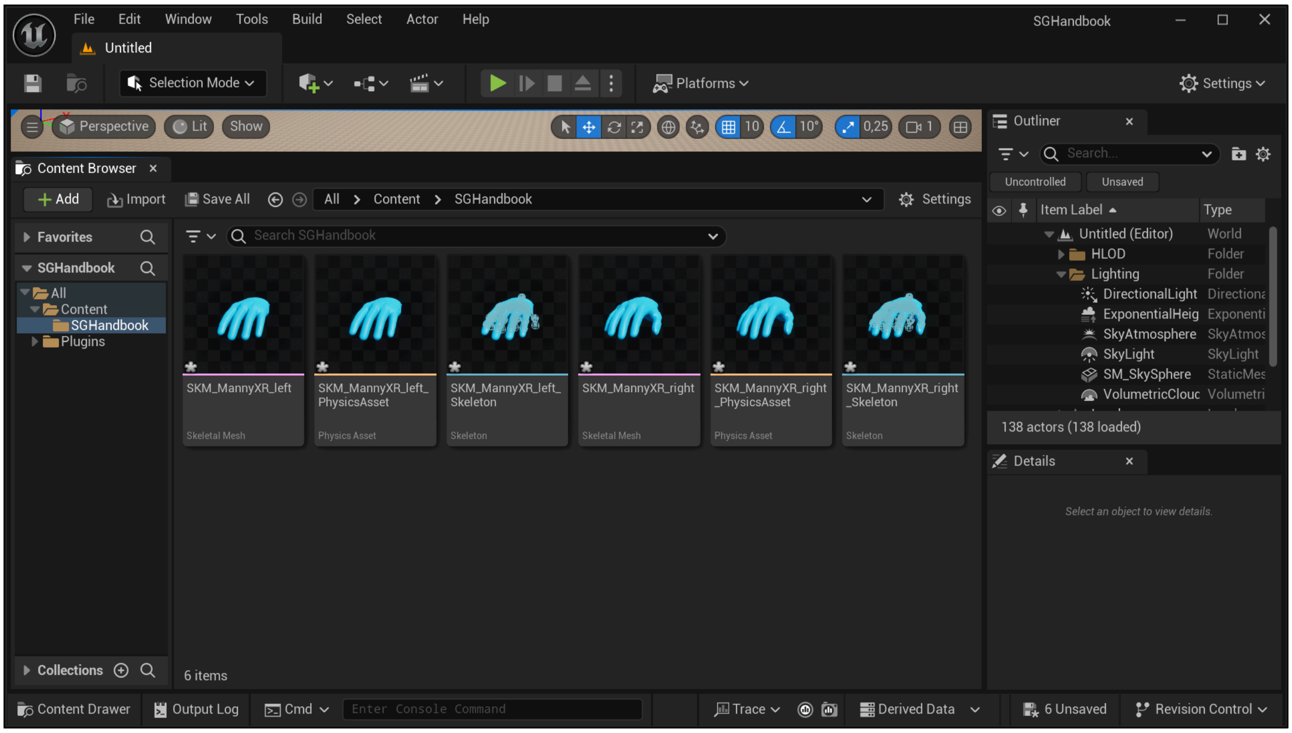
8. Open the Skeletal Mesh asset for the left hand and assign the `M_SenseGlove_VirtualHand` material from the `Asset Details` panel.



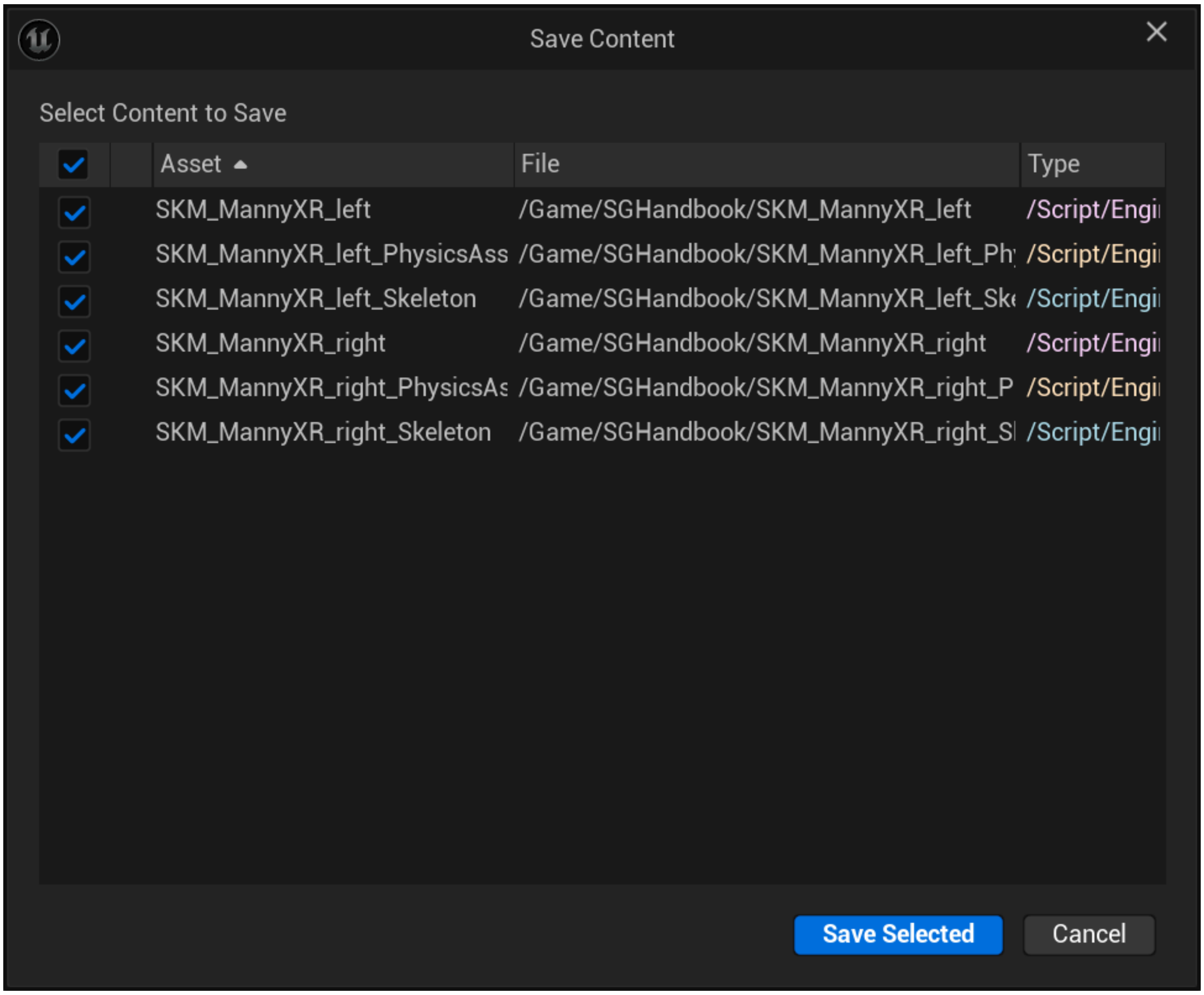
9. Repeat the process for the Skeletal Mesh asset of the right hand, and assign the `M_SenseGlove_VirtualHand` material in the `Asset Details` panel.



10. Return to the Content Browser by closing all asset windows and click the `Save All` button to save all imported virtual hand mesh assets to disk.



11. In the `Save Content` dialog, choose `Save Selected` to confirm the saving all action.

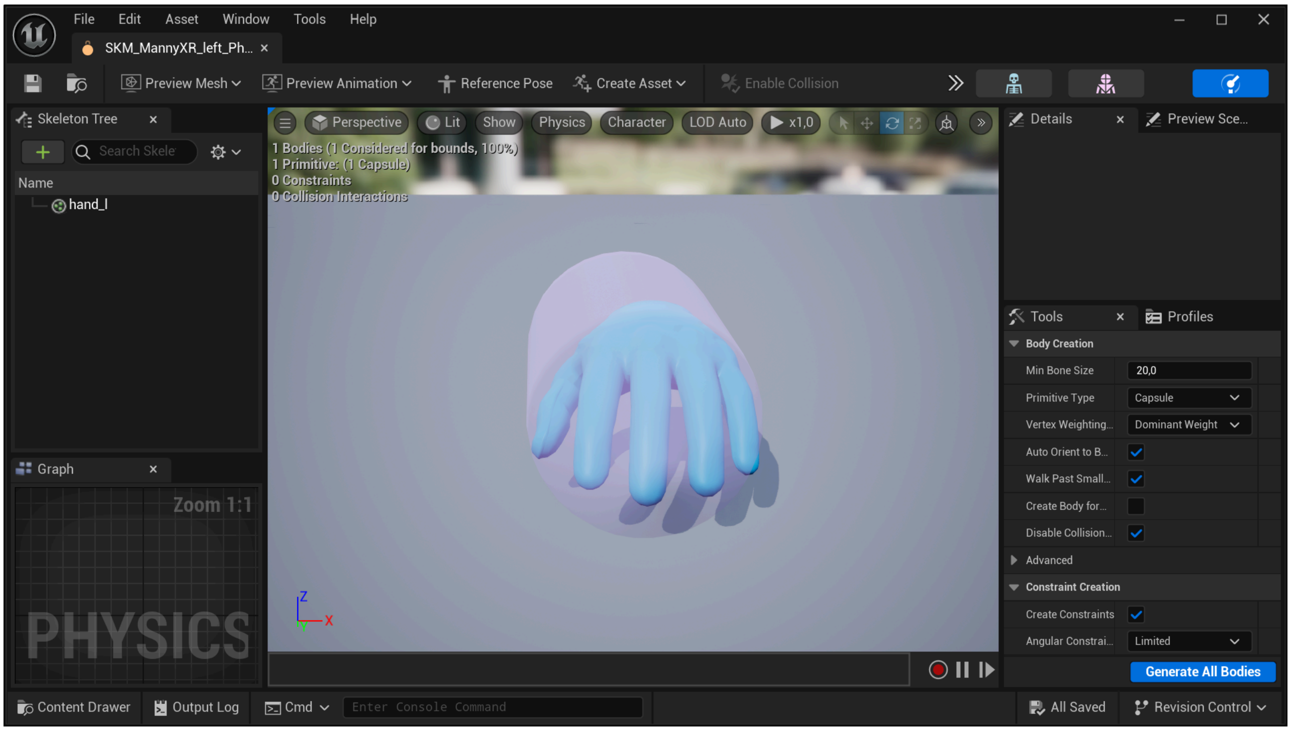

Save Content

Select Content to Save

✓	Asset	File	Type
✓	SKM_MannyXR_left	/Game/SGHandbook/SKM_MannyXR_left	/Script/Engi
✓	SKM_MannyXR_left_PhysicsAss	/Game/SGHandbook/SKM_MannyXR_left_Ph	/Script/Engi
✓	SKM_MannyXR_left_Skeleton	/Game/SGHandbook/SKM_MannyXR_left_Ske	/Script/Engi
✓	SKM_MannyXR_right	/Game/SGHandbook/SKM_MannyXR_right	/Script/Engi
✓	SKM_MannyXR_right_PhysicsAs	/Game/SGHandbook/SKM_MannyXR_right_P	/Script/Engi
✓	SKM_MannyXR_right_Skeleton	/Game/SGHandbook/SKM_MannyXR_right_Sl	/Script/Engi

Setting up the Rigid Bodies

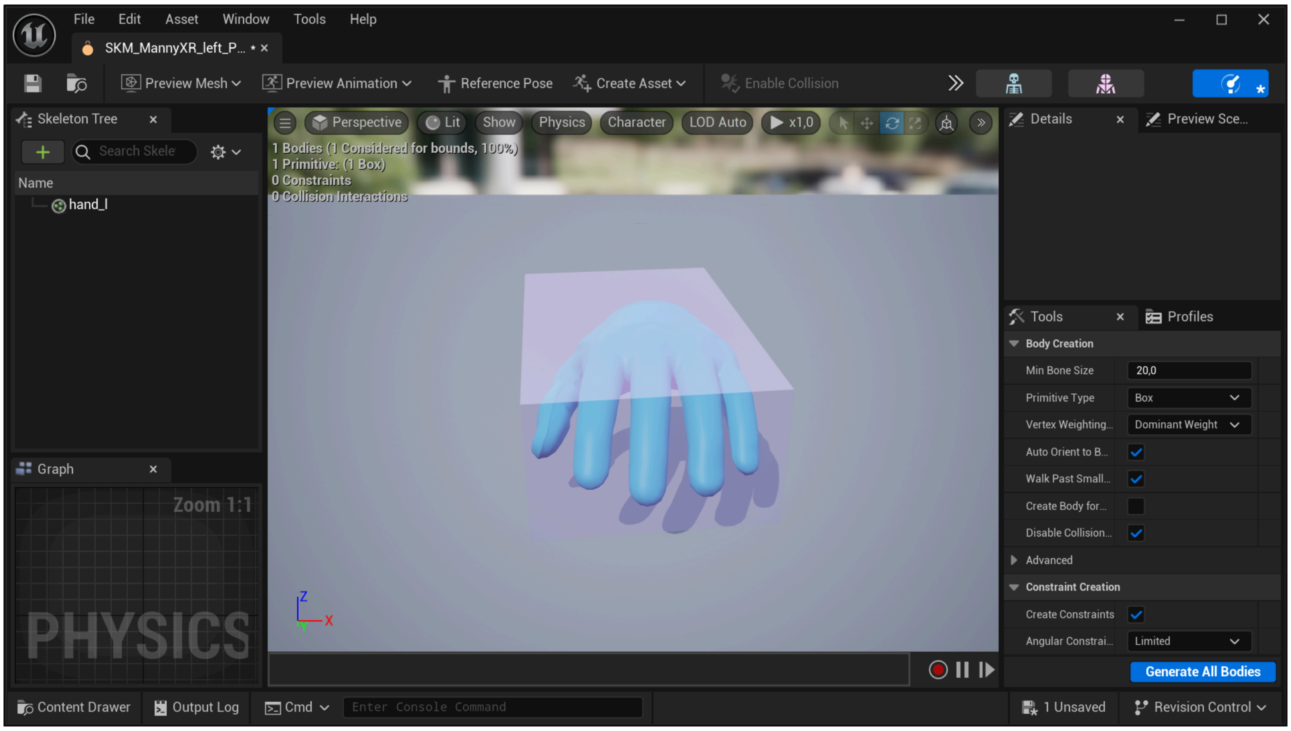
1. Open the Physics Asset for the left virtual hand mesh by double-clicking it in the Content Browser. This will open the PhAT (Physics Asset Tool) editor, where the virtual hand mesh for the left hand will appear with a default physics body, usually shaped as a capsule.



1 Bodies (1 Considered for bounds, 100%)
1 Primitive: (1 Capsule)
0 Constraints
0 Collision Interactions

Body Creation	
Min Bone Size	20,0
Primitive Type	Capsule
Vertex Weighting...	Dominant Weight
Auto Orient to B...	✓
Walk Past Small...	✓
Create Body for...	
Disable Collision...	✓
Advanced	
Constraint Creation	
Create Constraints	✓
Angular Constrai...	Limited

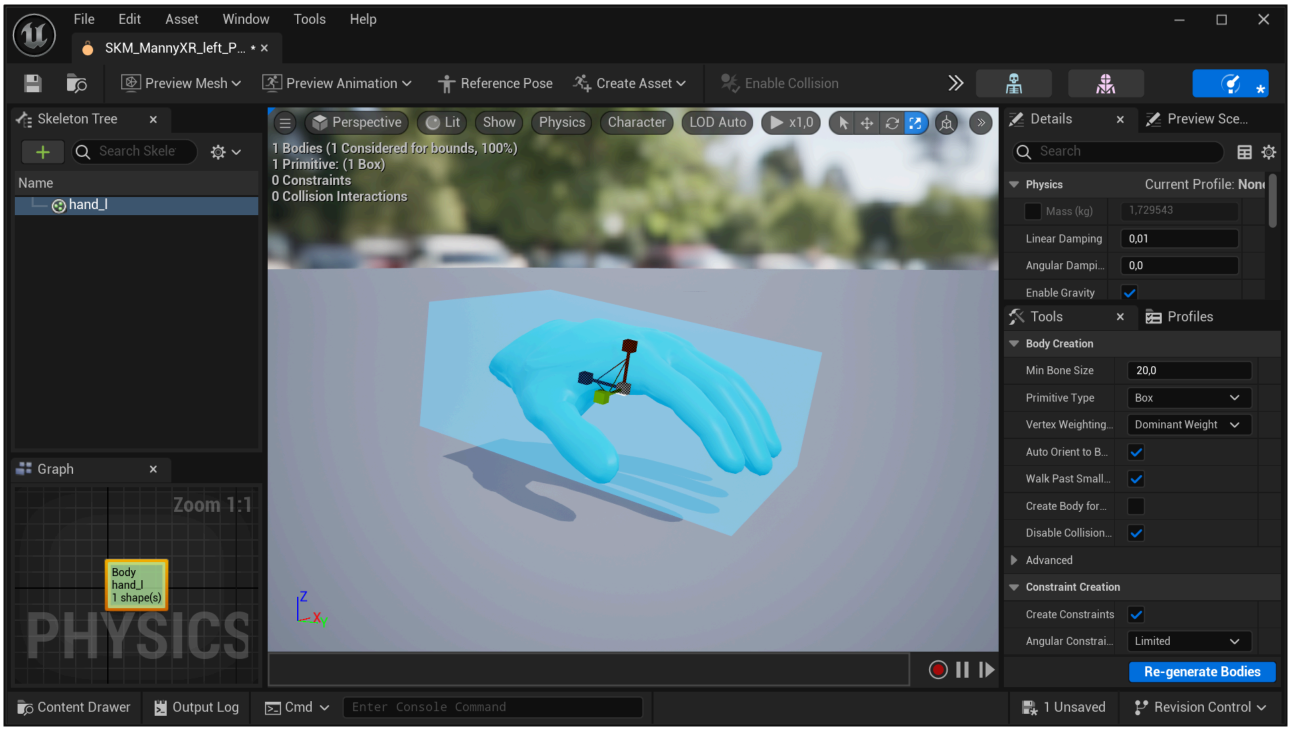
2. In the `Tools` panel, under the `Body Creation` section, locate the `Primitive Type` dropdown and select `Box` instead of the default `Capsule` shape. Then, click the `Generate All Bodies` button at the bottom of the `Tools` panel to create a new physics body.



1 Bodies (1 Considered for bounds, 100%)
1 Primitive: (1 Box)
0 Constraints
0 Collision Interactions

Body Creation	
Min Bone Size	20,0
Primitive Type	Box
Vertex Weighting...	Dominant Weight
Auto Orient to B...	✓
Walk Past Small...	✓
Create Body for...	
Disable Collision...	✓
Advanced	
Constraint Creation	
Create Constraints	✓
Angular Constrai...	Limited

3. After generating the new body, some adjustments are required for optimal interactions inside your VR simulations. Press the `r` key on your keyboard to enter scaling mode and use the arrows to resize the physics body. To reposition the body, press the `w` key to switch to translation mode. For adjusting the rotation, press the `e` key. Toggle between these modes as needed to fine-tune the physics body to your requirements.



1 Bodies (1 Considered for bounds, 100%)
1 Primitive: (1 Box)
0 Constraints
0 Collision Interactions

Physics — Current Profile: None

Mass (kg)	1,729543
Linear Damping	0,01
Angular Dampi...	0,0
Enable Gravity	✓

Body Creation

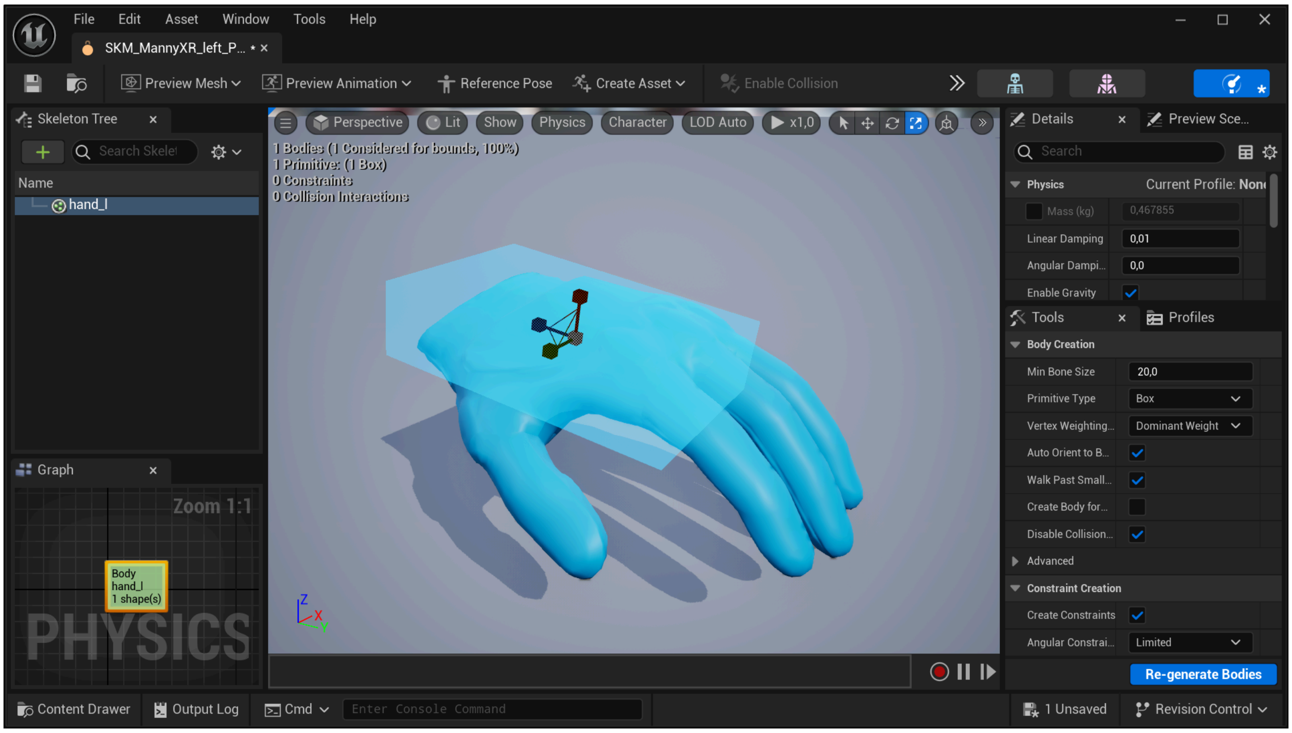
Min Bone Size	20,0
Primitive Type	Box
Vertex Weighting...	Dominant Weight
Auto Orient to B...	✓
Walk Past Small...	✓
Create Body for...	
Disable Collision...	✓

Advanced

Constraint Creation

Create Constraints	✓
Angular Constrai...	Limited

4. You can always revisit and adjust the rigid body later after testing its impact in your VR simulations. For now, save the asset and close the PhAT editor.



1 Bodies (1 Considered for bounds, 100%)
1 Primitive: (1 Box)
0 Constraints
0 Collision Interactions

Physics — Current Profile: None
Mass (kg): 0,467855
Linear Damping: 0,01
Angular Dampi...: 0,0
Enable Gravity: ✓

Body Creation
Min Bone Size: 20,0
Primitive Type: Box
Vertex Weighting...: Dominant Weight
Auto Orient to B...: ✓
Walk Past Small...: ✓
Create Body for...:
Disable Collision...: ✓
Advanced

Constraint Creation
Create Constraints: ✓
Angular Constrai...: Limited

5. Repeat the same procedure for the right virtual hand mesh.

Note

An older yet still relevant video tutorial demonstrating a similar procedure is also available.

Setting up the SenseGlove Grab and Touch Sockets

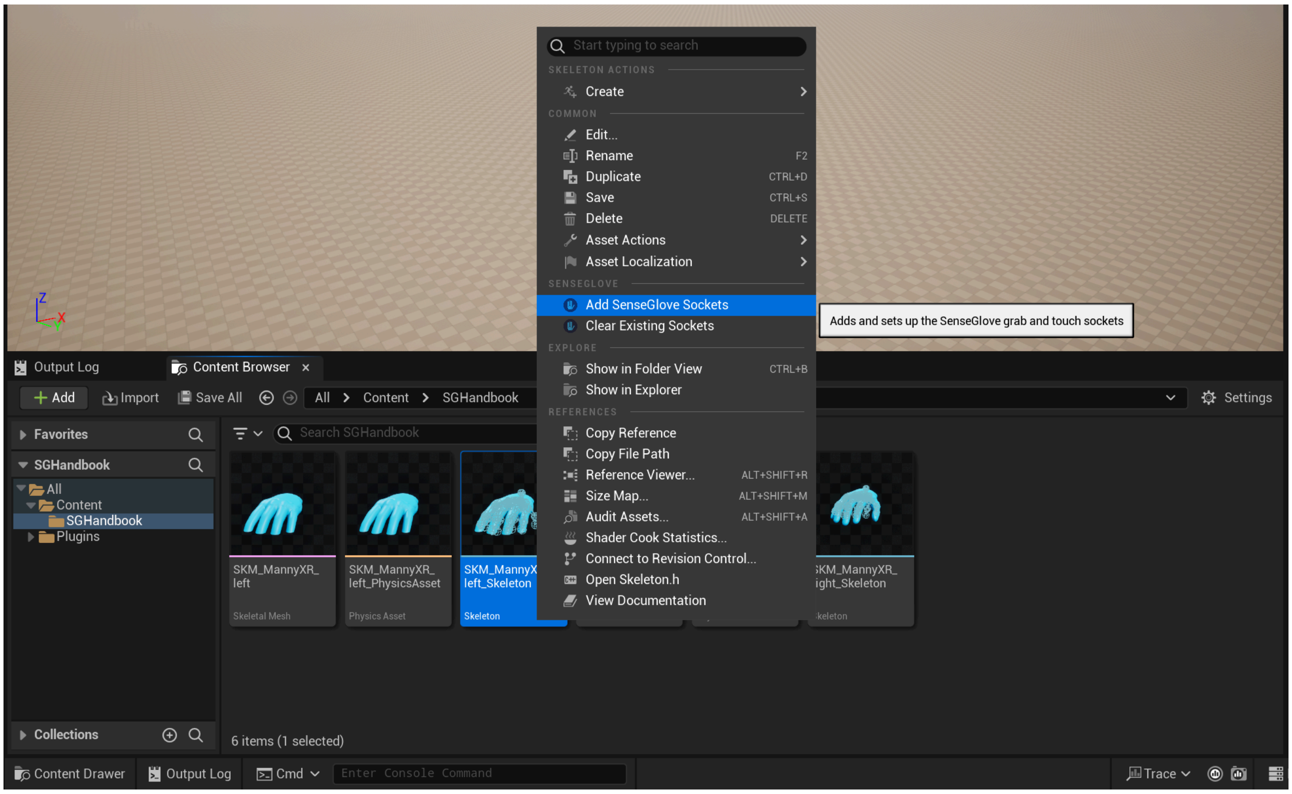
To ensure the Grab/Release and Touch systems function correctly, multiple sockets must be set up on each virtual hand mesh with precise locations and rotations. Before version `v2.1.0` of the SenseGlove Unreal Engine Plugin, this was a manual and time-consuming process. However, with the `v2.1.0` release, the plugin now includes the SenseGlove Sockets Editor, a built-in tool specifically designed for this task.

> **Note**
>
> If for any reason you still prefer to manually set up the sockets, a detailed video tutorial is available.

Accessing the SenseGlove Sockets Editor

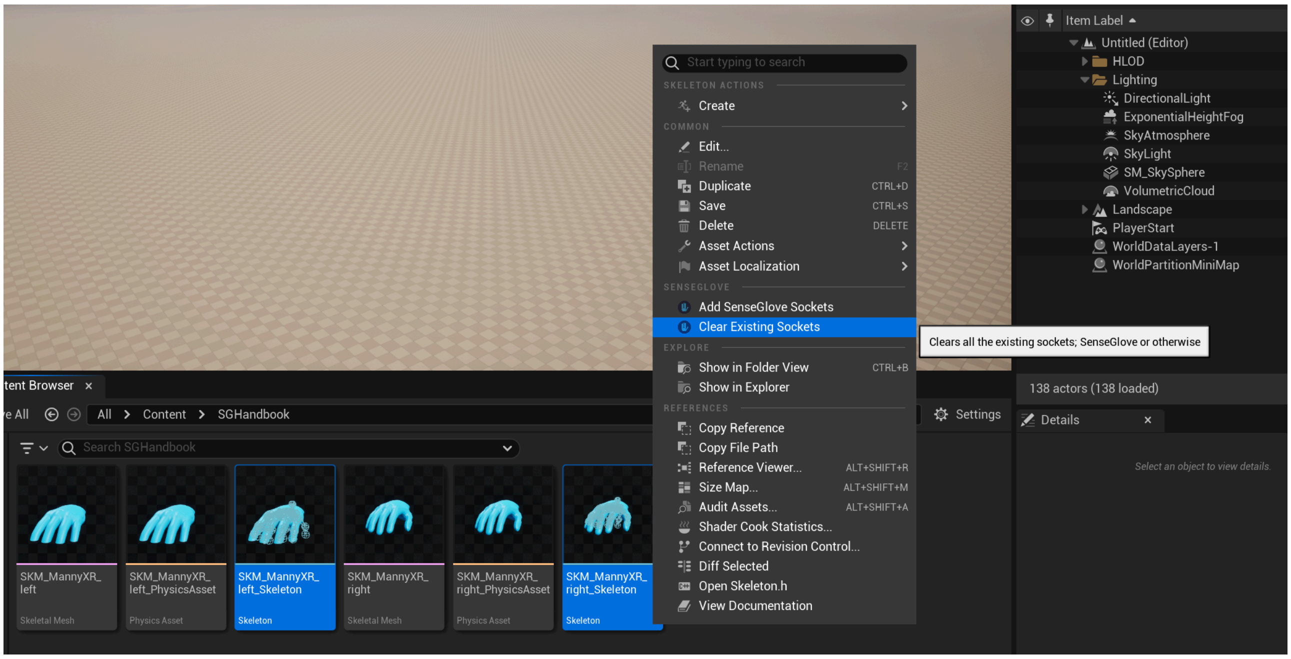
The SenseGlove Sockets Editor can be utilized in three ways:

1. By right-clicking on any Skeleton or Skeletal Mesh asset inside the Unreal Content Browser.

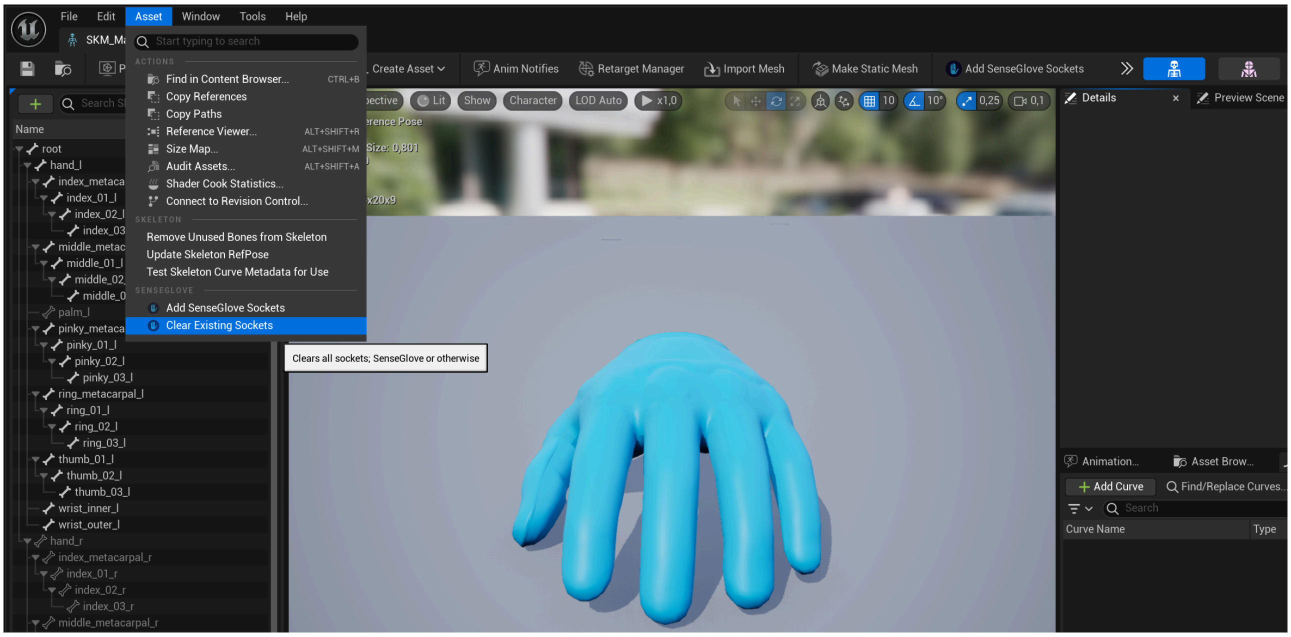


Tip

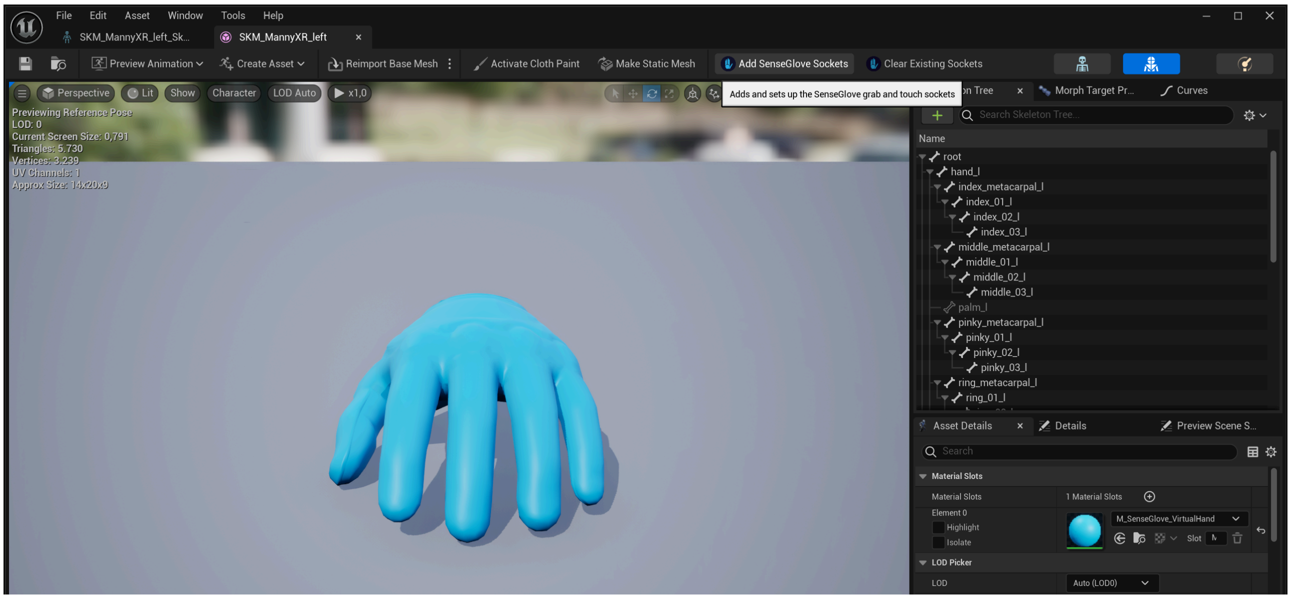
You can also perform Sockets Editor actions in bulk by selecting multiple assets of the same type and right-clicking on one of them. Note that if the selected assets are not all of the same type, Sockets Editor actions will not appear (e.g. selecting assets of type Skeletons and Skeletal Meshes together).



2. From the `Asset` menu in the Skeleton Editor or Skeletal Mesh Editor for any open Skeleton or Skeletal Mesh asset.



3. From the Skeleton Editor or Skeletal Mesh Editor toolbar for any open Skeleton or Skeletal Mesh asset.



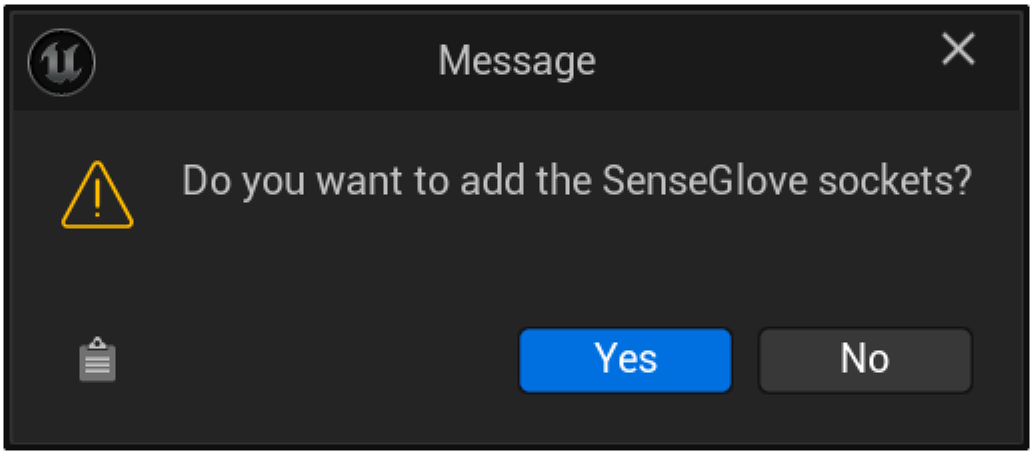
The SenseGlove Sockets Editor currently offers two actions:

1. `Add SenseGlove Sockets` : which adds and sets up the SenseGlove grab and touch sockets to any virtual hand mesh that adheres to the Epic rig and bone structure.
2. `Clear Existing Sockets` : which destructively clears all existing sockets; SenseGlove or otherwise, from any mesh.

> **Important**
>
> Simply performing any of these actions won't permanently modify your assets. In fact, if you close the Unreal Editor without saving your assets first, all changes performed by the SenseGlove Sockets Editor will be lost forever. This is by design and the plugin will leave this final choice to the user. So, in order to apply the changes permanently, you must save the assets manually.

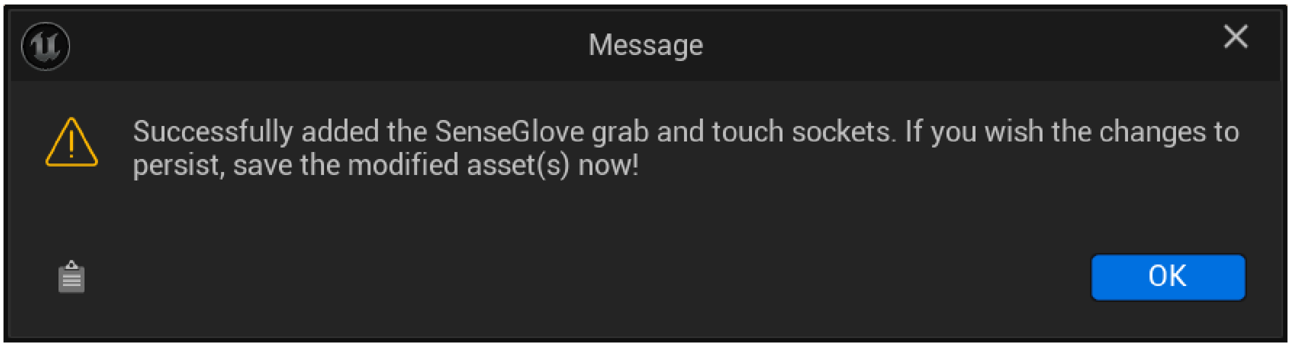
Adding the SenseGlove Sockets

When you invoke the `Add SenseGlove Sockets` action, the Sockets Editor will prompt you for confirmation:


Message

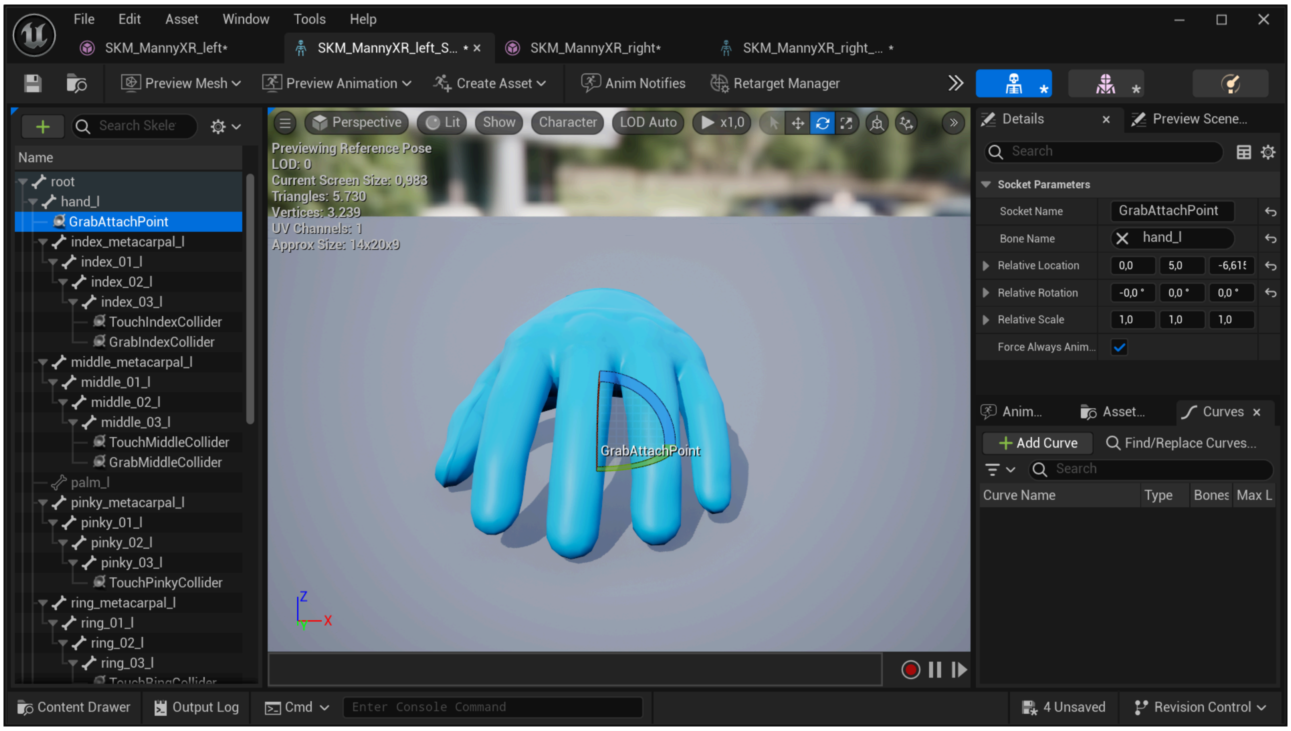
Do you want to add the SenseGlove sockets?

If it succeeds at adding the standard SenseGlove sockets, you will receive a confirmation message:


Message

Successfully added the SenseGlove grab and touch sockets. If you wish the changes to persist, save the modified asset(s) now!

After closing the dialog, the editors for the affected Skeleton and Skeletal Mesh assets will open, displaying the newly added sockets:



Name
- root
  - hand_l
    - GrabAttachPoint
    - index_metacarpal_l
      - index_01_l
        - index_02_l
          - index_03_l
            - TouchIndexCollider
            - GrabIndexCollider
    - middle_metacarpal_l
      - middle_01_l
        - middle_02_l
          - middle_03_l
            - TouchMiddleCollider
            - GrabMiddleCollider
    - palm_l
    - pinky_metacarpal_l
      - pinky_01_l
        - pinky_02_l
          - pinky_03_l
            - TouchPinkyCollider
    - ring_metacarpal_l
      - ring_01_l
        - ring_02_l
          - ring_03_l

```
Previewing Reference Pose
LOD: 0
Current Screen Size: 0,983
Triangles: 5.730
Vertices: 3.239
UV Channels: 1
Approx Size: 14x20x9
```

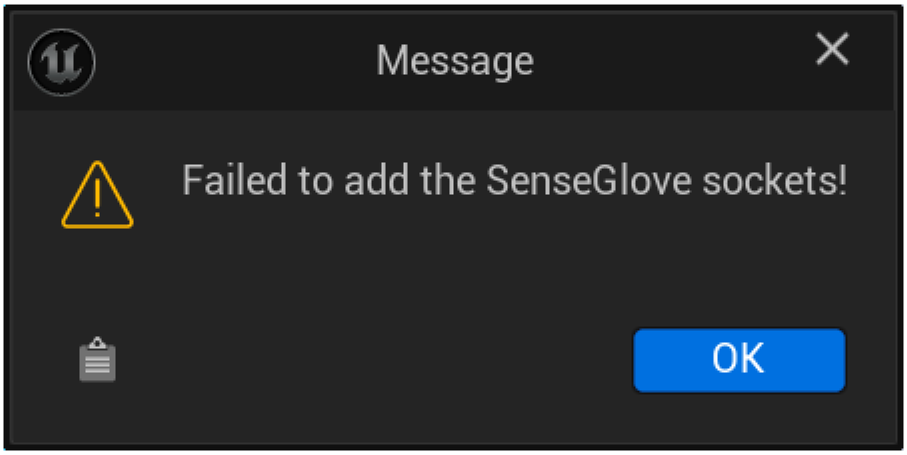
Socket Parameters

Socket Name	GrabAttachPoint		
Bone Name	hand_l		
Relative Location	0,0	5,0	-6,61
Relative Rotation	-0,0 °	0,0 °	0,0 °
Relative Scale	1,0	1,0	1,0
Force Always Anim...	✓		

To ensure the changes persist, save the assets to disk.

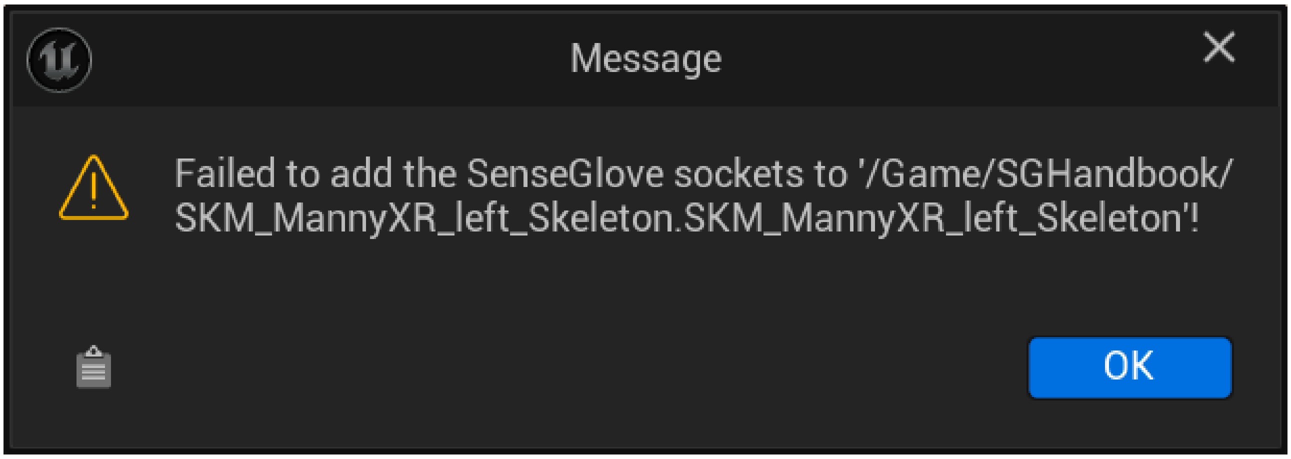
Note

The `Add SenseGlove Sockets` action can fail for various reasons, so it's important to investigate and identify the cause if an issue arises.



Message

Failed to add the SenseGlove sockets!



```
Failed to add the SenseGlove sockets to '/Game/SGHandbook/
SKM_MannyXR_left_Skeleton.SKM_MannyXR_left_Skeleton'!
```

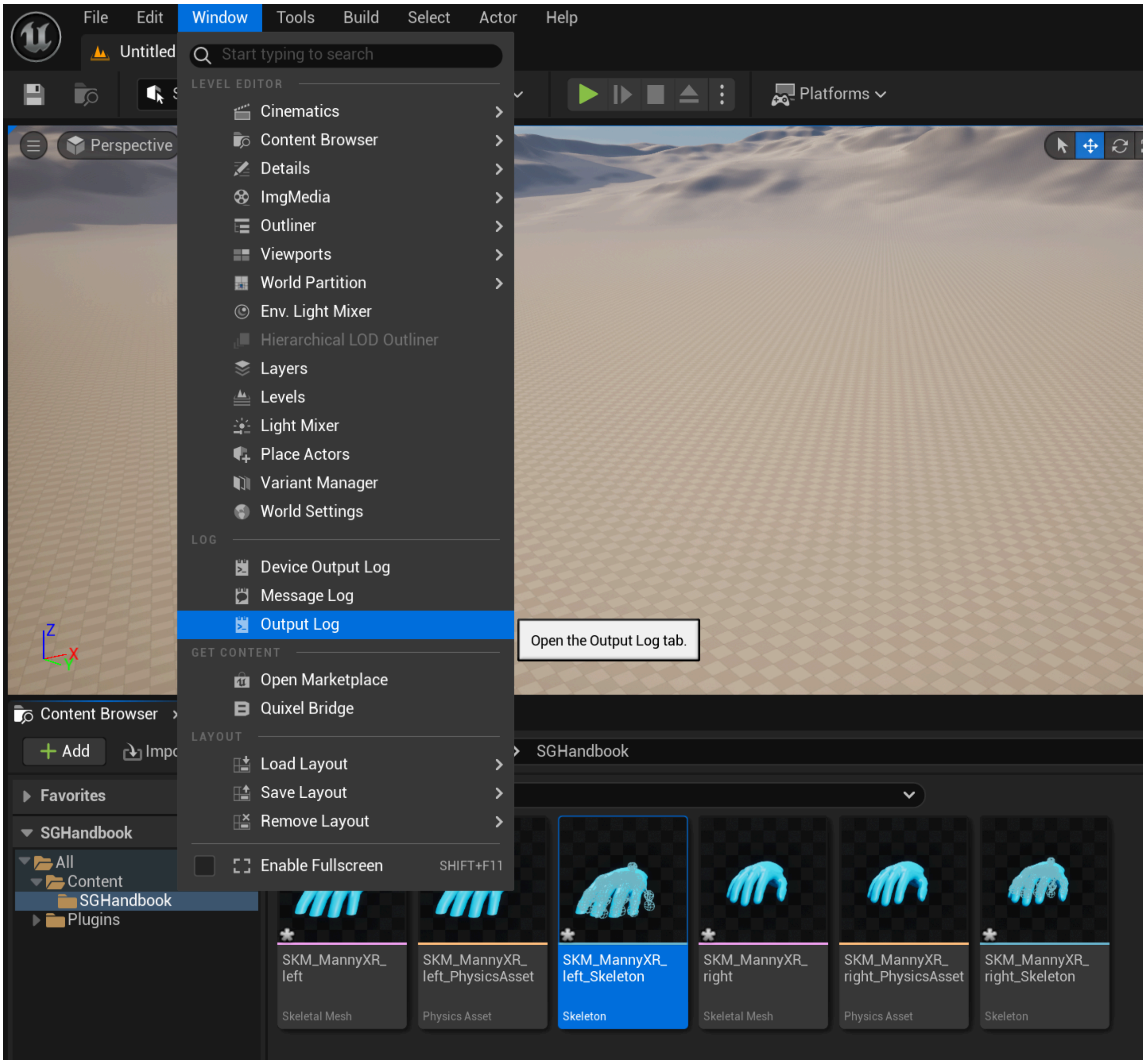
Important

A common cause of failure is that the SenseGlove sockets have already been set up, or the meshes you're using already have the necessary sockets. In this case, consider using the `Clear Existing Sockets` action first.

Caution

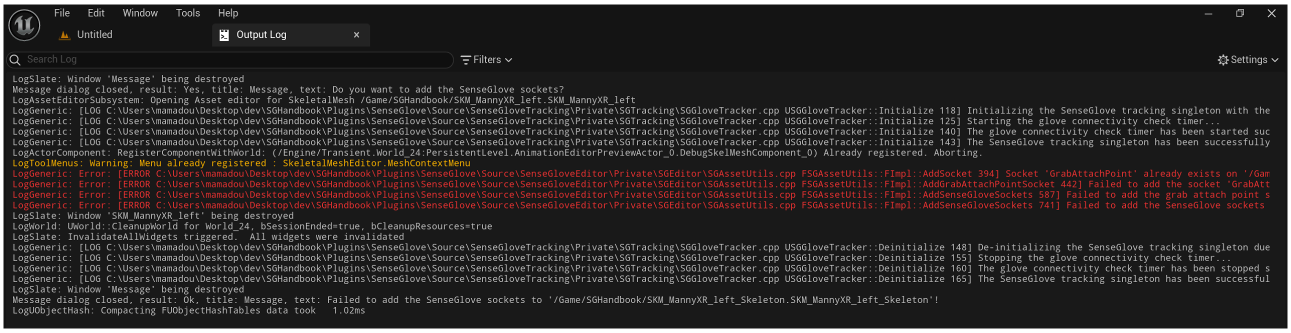
Another common cause of failure is if your virtual hand meshes share a skeleton. As noted in the Compatible Virtual Hand Meshes section, the SenseGlove Sockets Editor does not support skeletal meshes that share their skeleton. You may need to export and re-import the virtual hand meshes in in a compatible manner first.

In any case, the SenseGlove Sockets Editor reports all failures in the Unreal Editor logs. To view and investigate the logs, simply head to the `Window` menu and click on `Output Log`:



For example, in the following screenshots the following errors are stated: `Socket 'GrabAttachPoint' already exists on '/Game/SGHandbook/SKM_MannyXR_left.SKM_MannyXR_left'; refuse to add a duplicate!` .

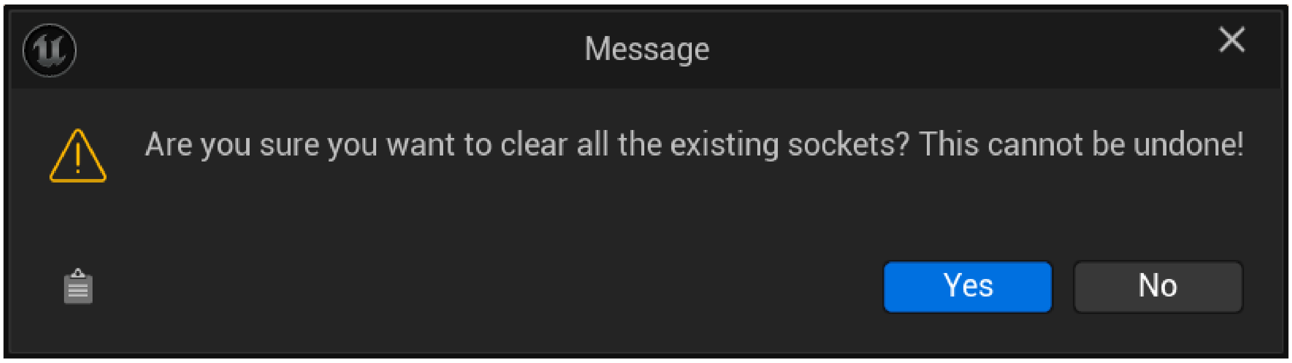
```
LogGeneric: Error: [ERROR
C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveE
ditor\Private\SGEditor\SGAssetUtils.cpp FSGAssetUtils::FImpl::AddSocket 394]
Socket 'GrabAttachPoint' already exists on
'/Game/SGHandbook/SKM_MannyXR_left.SKM_MannyXR_left'; refuse to add a
duplicate!
LogGeneric: Error: [ERROR
C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveE
ditor\Private\SGEditor\SGAssetUtils.cpp
FSGAssetUtils::FImpl::AddGrabAttachPointSocket 442] Failed to add the socket
'GrabAttachPoint' to '/Game/SGHandbook/SKM_MannyXR_left.SKM_MannyXR_left'!
LogGeneric: Error: [ERROR
C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveE
ditor\Private\SGEditor\SGAssetUtils.cpp
FSGAssetUtils::FImpl::AddSenseGloveSockets 587] Failed to add the grab attach
point socket to asset '/Game/SGHandbook/SKM_MannyXR_left.SKM_MannyXR_left'!
LogGeneric: Error: [ERROR
C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveE
ditor\Private\SGEditor\SGAssetUtils.cpp
FSGAssetUtils::FImpl::AddSenseGloveSockets 741] Failed to add the SenseGlove
sockets to the asset '/Game/SGHandbook/SKM_MannyXR_left.SKM_MannyXR_left'!
```



```
LogSlate: Window 'Message' being destroyed
Message dialog closed, result: Yes, title: Message, text: Do you want to add the SenseGlove sockets?
LogAssetEditorSubsystem: Opening Asset editor for SkeletalMesh /Game/SGHandbook/SKM_MannyXR_left.SKM_MannyXR_left
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Initialize 118] Initializing the SenseGlove tracking singleton with the
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Initialize 125] Starting the glove connectivity check timer...
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Initialize 140] The glove connectivity check timer has been started suc
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Initialize 143] The SenseGlove tracking singleton has been successfully
LogActorComponent: RegisterComponentWithWorld: (/Engine/Transient.World_24:PersistentLevel.AnimationEditorPreviewActor_0.DebugSkelMeshComponent_0) Already registered. Aborting.
LogToolMenus: Warning: Menu already registered : SkeletalMeshEditor.MeshContextMenu
LogGeneric: Error: [ERROR C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveEditor\Private\SGEditor\SGAssetUtils.cpp FSGAssetUtils::FImpl::AddSocket 394] Socket 'GrabAttachPoint' already exists on '/Gam
LogGeneric: Error: [ERROR C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveEditor\Private\SGEditor\SGAssetUtils.cpp FSGAssetUtils::FImpl::AddGrabAttachPointSocket 442] Failed to add the socket 'GrabAtt
LogGeneric: Error: [ERROR C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveEditor\Private\SGEditor\SGAssetUtils.cpp FSGAssetUtils::FImpl::AddSenseGloveSockets 587] Failed to add the grab attach point s
LogGeneric: Error: [ERROR C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveEditor\Private\SGEditor\SGAssetUtils.cpp FSGAssetUtils::FImpl::AddSenseGloveSockets 741] Failed to add the SenseGlove sockets
LogSlate: Window 'SKM_MannyXR_left' being destroyed
LogWorld: UWorld::CleanupWorld for World_24, bSessionEnded=true, bCleanupResources=true
LogSlate: InvalidateAllWidgets triggered.  All widgets were invalidated
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Deinitialize 148] De-initializing the SenseGlove tracking singleton due
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Deinitialize 155] Stopping the glove connectivity check timer...
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Deinitialize 160] The glove connectivity check timer has been stopped s
LogGeneric: [LOG C:\Users\mamadou\Desktop\dev\SGHandbook\Plugins\SenseGlove\Source\SenseGloveTracking\Private\SGTracking\SGGloveTracker.cpp USGGloveTracker::Deinitialize 165] The SenseGlove tracking singleton has been successful
LogSlate: Window 'Message' being destroyed
Message dialog closed, result: Ok, title: Message, text: Failed to add the SenseGlove sockets to '/Game/SGHandbook/SKM_MannyXR_left_Skeleton.SKM_MannyXR_left_Skeleton'!
LogUObjectHash: Compacting FUObjectHashTables data took   1.02ms
```

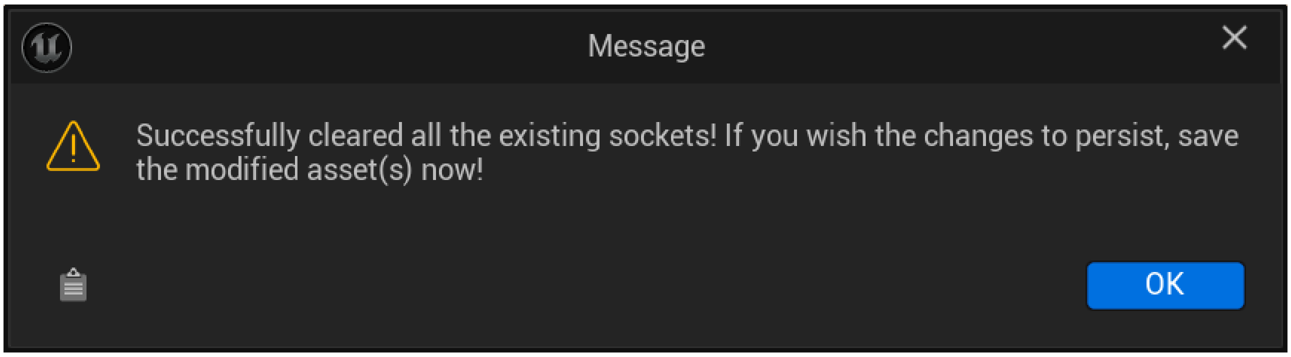
Clearing All Existing Sockets

When you invoke the `Clear Existing Sockets` action, the Sockets Editor will ask for your confirmation:


Message

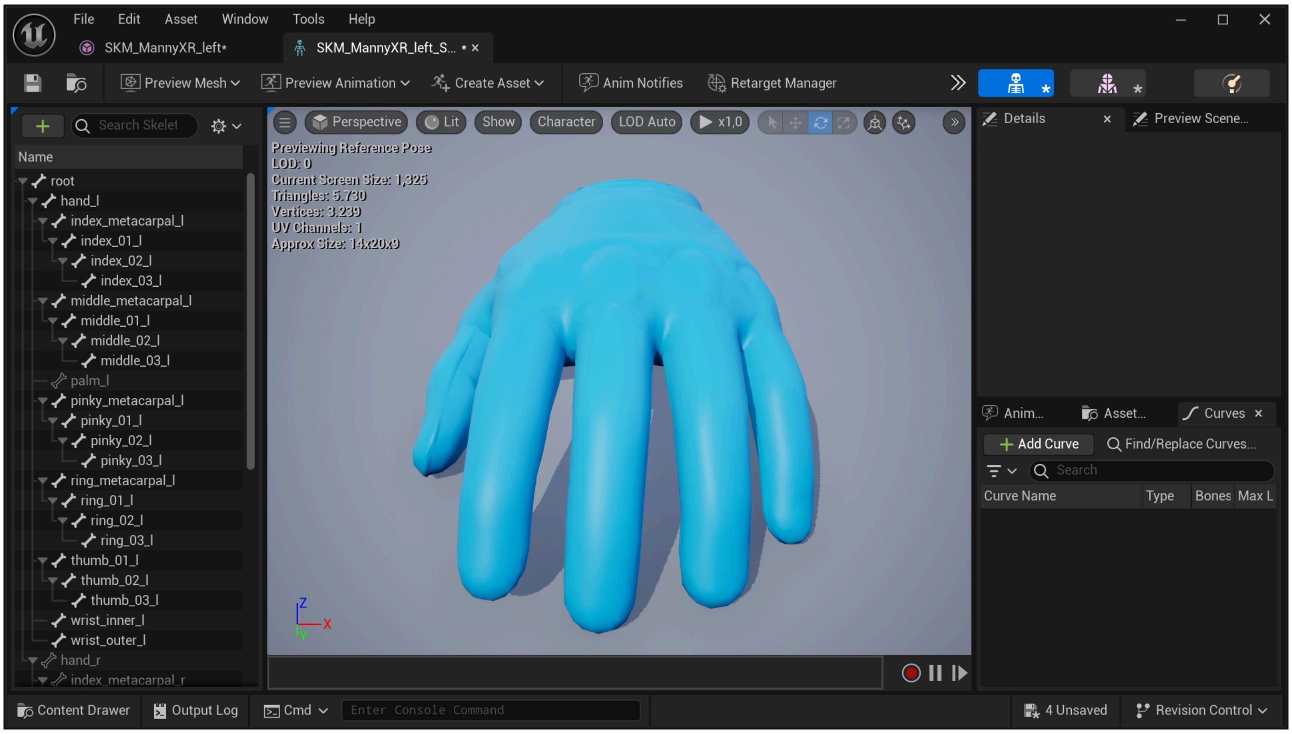
Are you sure you want to clear all the existing sockets? This cannot be undone!

If successful, you will receive a message indicating all the existing sockets have been cleared:


Message

Successfully cleared all the existing sockets! If you wish the changes to persist, save the modified asset(s) now!

After closing the dialog, the editors for the affected Skeleton and Skeletal Mesh assets will open, displaying the affected assets with all sockets cleared:



Configuring the SGPawn and Plugin Virtual Hand Mesh Settings

The final step in setting up the virtual hand meshes is to configure the `SGPawn` and Plugin `Virtual Hand Mesh Settings` to ensure they utilize the newly created virtual hand meshes.

Please visit Setting Up SGPawn, The Virtual Hand Mesh Settings, and how to setup the Wrist Tracking Hardware sections for more information.

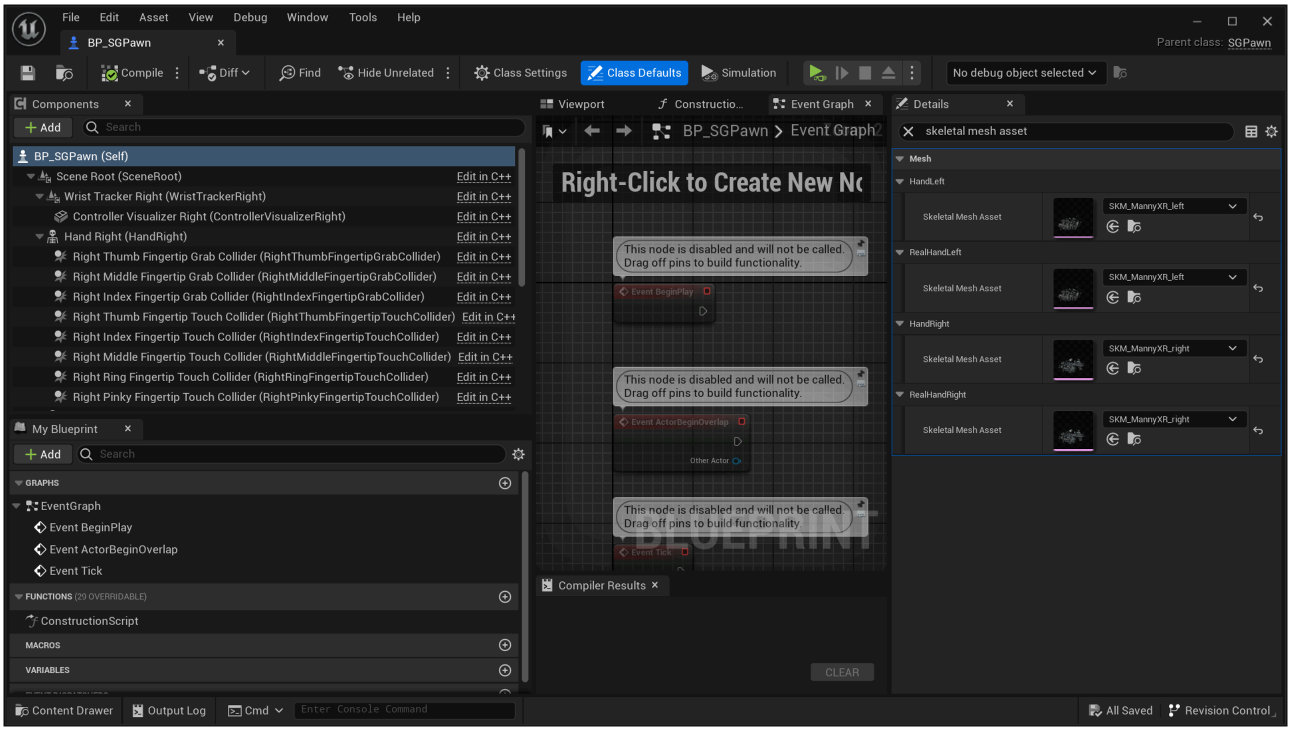
SGPawn Configuration

In the `SGPawn` Blueprint class, make sure to assign the appropriate `Skeletal Mesh Asset` to the following components:

- `HandLeft`
- `HandRight`

- `RealHandLeft`
- `RealHandRight`

This ensures that the correct hand meshes are used for both virtual and real hands.



BP_SGPawn (Self)
- Scene Root (SceneRoot) — Edit in C++
  - Wrist Tracker Right (WristTrackerRight) — Edit in C++
    - Controller Visualizer Right (ControllerVisualizerRight) — Edit in C++
  - Hand Right (HandRight) — Edit in C++
    - Right Thumb Fingertip Grab Collider (RightThumbFingertipGrabCollider) — Edit in C++
    - Right Middle Fingertip Grab Collider (RightMiddleFingertipGrabCollider) — Edit in C++
    - Right Index Fingertip Grab Collider (RightIndexFingertipGrabCollider) — Edit in C++
    - Right Thumb Fingertip Touch Collider (RightThumbFingertipTouchCollider) — Edit in C++
    - Right Index Fingertip Touch Collider (RightIndexFingertipTouchCollider) — Edit in C++
    - Right Middle Fingertip Touch Collider (RightMiddleFingertipTouchCollider) — Edit in C++
    - Right Ring Fingertip Touch Collider (RightRingFingertipTouchCollider) — Edit in C++
    - Right Pinky Fingertip Touch Collider (RightPinkyFingertipTouchCollider) — Edit in C++

EventGraph
- Event BeginPlay
- Event ActorBeginOverlap
- Event Tick

ConstructionScript

Details — search: skeletal mesh asset

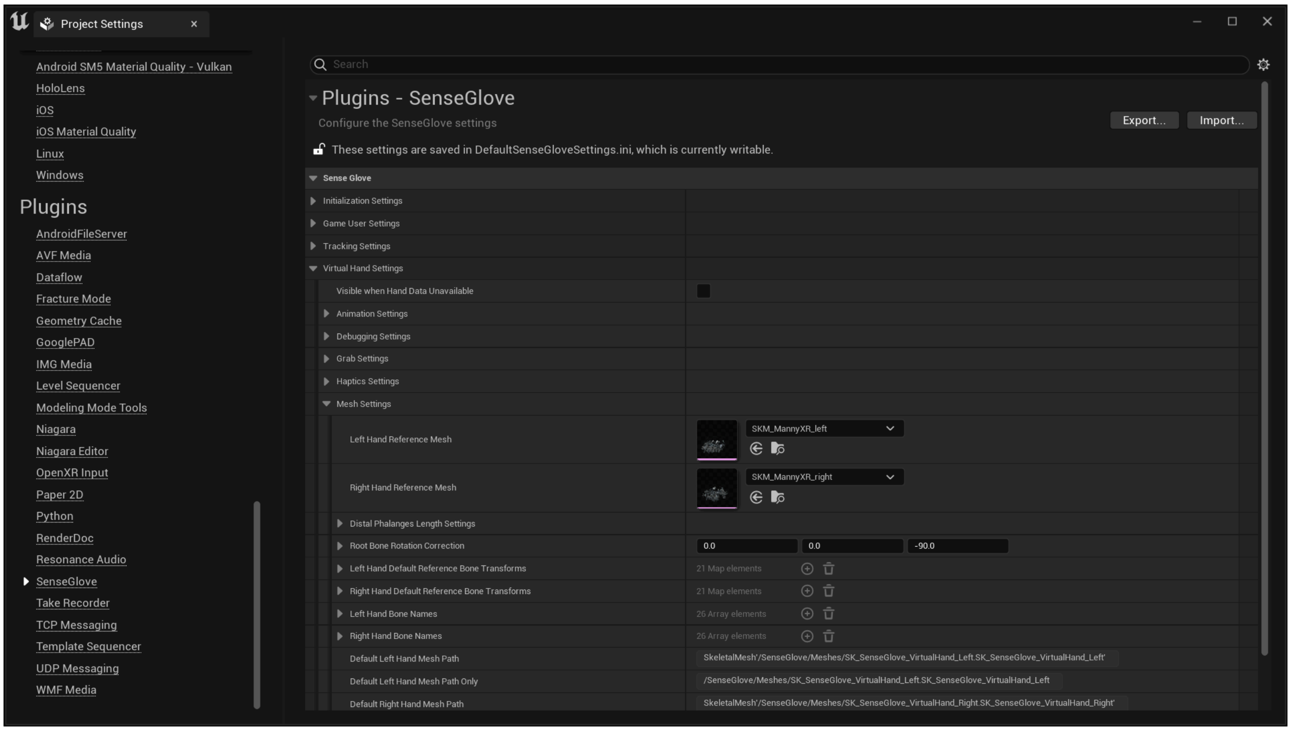
Mesh
- HandLeft — Skeletal Mesh Asset: SKM_MannyXR_left
- RealHandLeft — Skeletal Mesh Asset: SKM_MannyXR_left
- HandRight — Skeletal Mesh Asset: SKM_MannyXR_right
- RealHandRight — Skeletal Mesh Asset: SKM_MannyXR_right

Plugin Virtual Hand Mesh Settings

Next, navigate to `Project Settings > Plugins > SenseGlove > Virtual Hand Settings > Mesh Settings` and specify the correct left and right-hand meshes for:

- `Left Hand Reference Mesh`
- `Right Hand Reference Mesh`

This configuration guarantees that the tracking system correctly interprets the bone transforms of the virtual hand meshes when generating `FXRHandTrackingState`. Additionally, it allows the animation system to accurately use these bone transforms when processing `FXRHandTrackingState` and animating the virtual hand meshes.



Project Settings

Android SM5 Material Quality - Vulkan
HoloLens
iOS
iOS Material Quality
Linux
Windows

Plugins

AndroidFileServer
AVF Media
Dataflow
Fracture Mode
Geometry Cache
GooglePAD
IMG Media
Level Sequencer
Modeling Mode Tools
Niagara
Niagara Editor
OpenXR Input
Paper 2D
Python
RenderDoc
Resonance Audio
SenseGlove
Take Recorder
TCP Messaging
Template Sequencer
UDP Messaging
WMF Media

Plugins - SenseGlove

Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

Sense Glove	
Initialization Settings	
Game User Settings	
Tracking Settings	
Virtual Hand Settings	
Visible when Hand Data Unavailable	☐
Animation Settings	
Debugging Settings	
Grab Settings	
Haptics Settings	
Mesh Settings	
Left Hand Reference Mesh	SKM_MannyXR_left
Right Hand Reference Mesh	SKM_MannyXR_right
Distal Phalanges Length Settings	
Root Bone Rotation Correction	0.0 / 0.0 / -90.0
Left Hand Default Reference Bone Transforms	21 Map elements
Right Hand Default Reference Bone Transforms	21 Map elements
Left Hand Bone Names	26 Array elements
Right Hand Bone Names	26 Array elements
Default Left Hand Mesh Path	SkeletalMesh'/SenseGlove/Meshes/SK_SenseGlove_VirtualHand_Left.SK_SenseGlove_VirtualHand_Left'
Default Left Hand Mesh Path Only	/SenseGlove/Meshes/SK_SenseGlove_VirtualHand_Left.SK_SenseGlove_VirtualHand_Left
Default Right Hand Mesh Path	SkeletalMesh'/SenseGlove/Meshes/SK_SenseGlove_VirtualHand_Right.SK_SenseGlove_VirtualHand_Right'

Setting Up the Wrist Tracking Hardware

To enable the SenseGlove Unreal Engine Plugin to track the gloves position and rotation in the world, you need to specify a positional tracking hardware, referred to as Wrist Tracking Hardware within the plugin. By default, if the Wrist Tracking Hardware is not explicitly set, the plugin will attempt to automatically detect it by identifying your Head-mounted display (HMD) hardware. However, this auto-detection feature may not be entirely reliable, as it is still experimental, and it may occasionally fail.

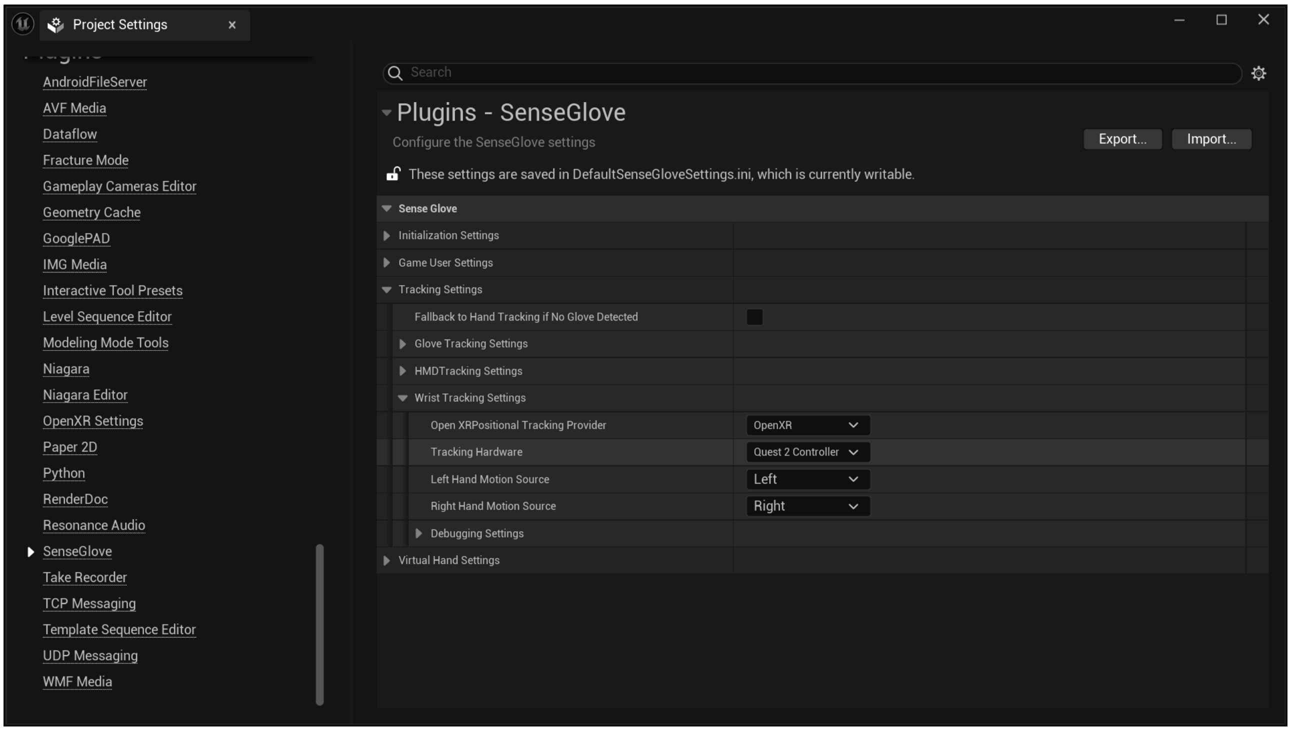
For detailed information, please visit the Wrist Tracking Hardware and HMD auto-detection configuration section.

Prerequisites

Before you even consider setting up the Wrist Tracking Settings for the SenseGlove Unreal Engine Plugin, please make sure that you have already taken all the necessary steps in the Enabling XR_EXT_hand_tracking OpenXR Extension on VR Headsets section for PCVR or Standalone modes. Otherwise, there's no guarantee that the plugin can access the location data from the wrist-tracking hardware of your choice. So, as the first troubleshooting measure, we always recommend double-checking the relevant prerequisite guides above.

Meta Quest 2 Controller

For the Meta Quest 2, whether in PCVR or Standalone mode, the functional wrist-tracking settings looks something like this:



Plugins - SenseGlove

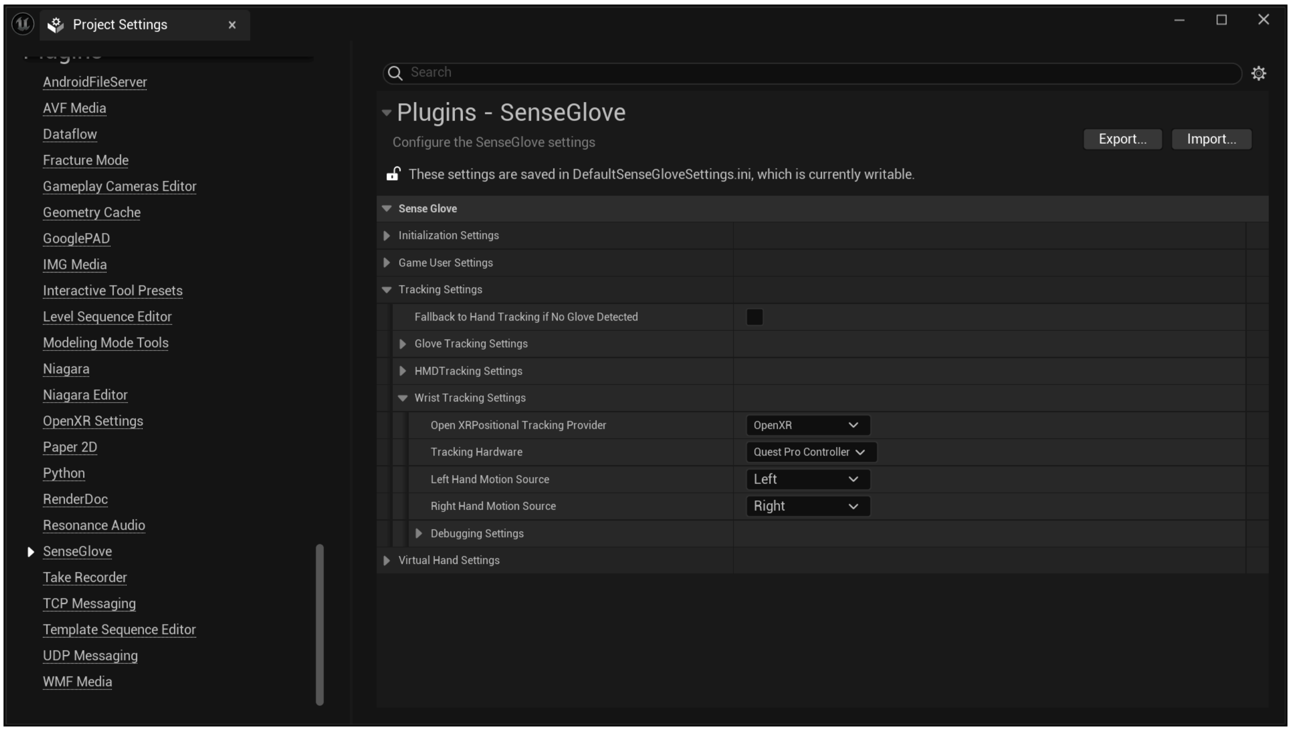
Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

- Sense Glove
  - Initialization Settings
  - Game User Settings
  - Tracking Settings
    - Fallback to Hand Tracking if No Glove Detected: ☐
    - Glove Tracking Settings
    - HMDTracking Settings
    - Wrist Tracking Settings
      - Open XRPositional Tracking Provider: OpenXR
      - Tracking Hardware: Quest 2 Controller
      - Left Hand Motion Source: Left
      - Right Hand Motion Source: Right
      - Debugging Settings
  - Virtual Hand Settings

Meta Quest Pro Controller

For the Meta Quest 3, whether in PCVR or Standalone mode, the functional wrist-tracking settings looks something like this:



Project Settings

AndroidFileServer
AVF Media
Dataflow
Fracture Mode
Gameplay Cameras Editor
Geometry Cache
GooglePAD
IMG Media
Interactive Tool Presets
Level Sequence Editor
Modeling Mode Tools
Niagara
Niagara Editor
OpenXR Settings
Paper 2D
Python
RenderDoc
Resonance Audio
SenseGlove
Take Recorder
TCP Messaging
Template Sequence Editor
UDP Messaging
WMF Media

Plugins - SenseGlove

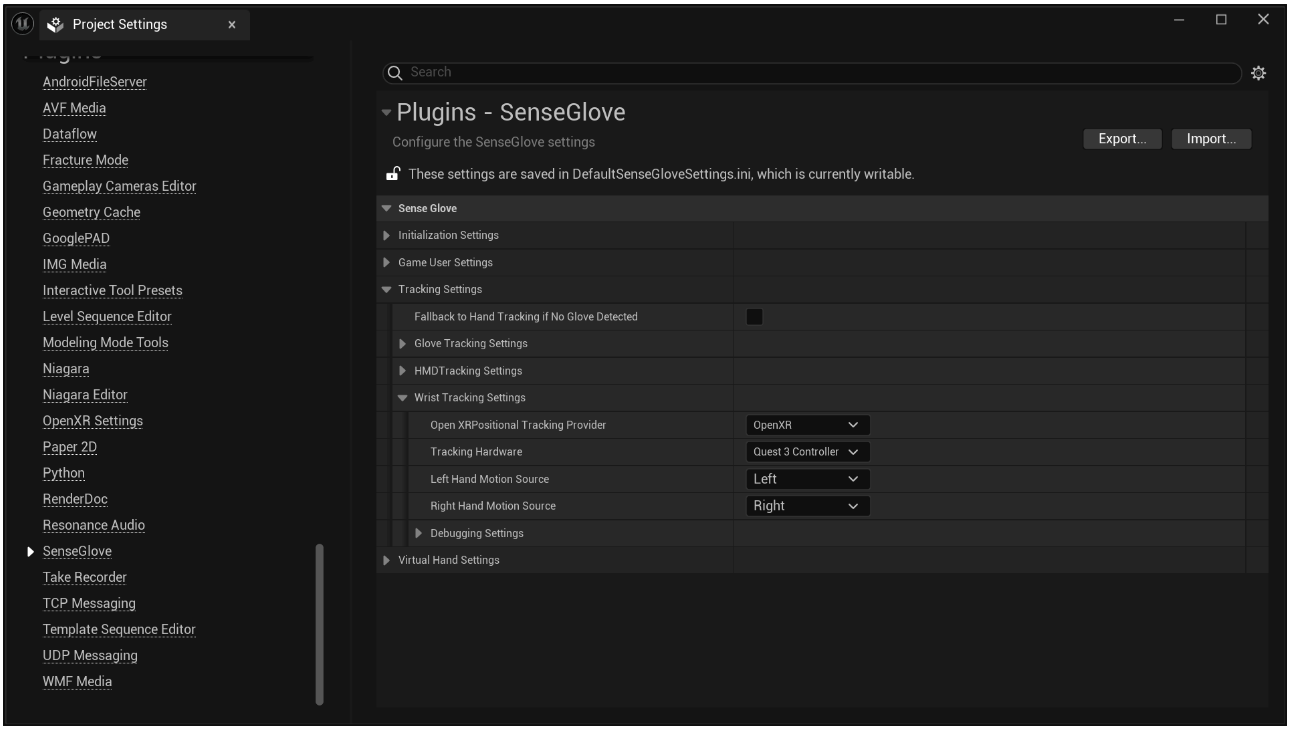
Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

Sense Glove
- Initialization Settings
- Game User Settings
- Tracking Settings
  - Fallback to Hand Tracking if No Glove Detected: ☐
  - Glove Tracking Settings
  - HMDTracking Settings
  - Wrist Tracking Settings
    - Open XRPositional Tracking Provider: OpenXR
    - Tracking Hardware: Quest Pro Controller
    - Left Hand Motion Source: Left
    - Right Hand Motion Source: Right
    - Debugging Settings
- Virtual Hand Settings

Meta Quest 3 Controller

For the Meta Quest 3, whether in PCVR or Standalone mode, the functional wrist-tracking settings looks something like this:


Project Settings

Plugins - SenseGlove

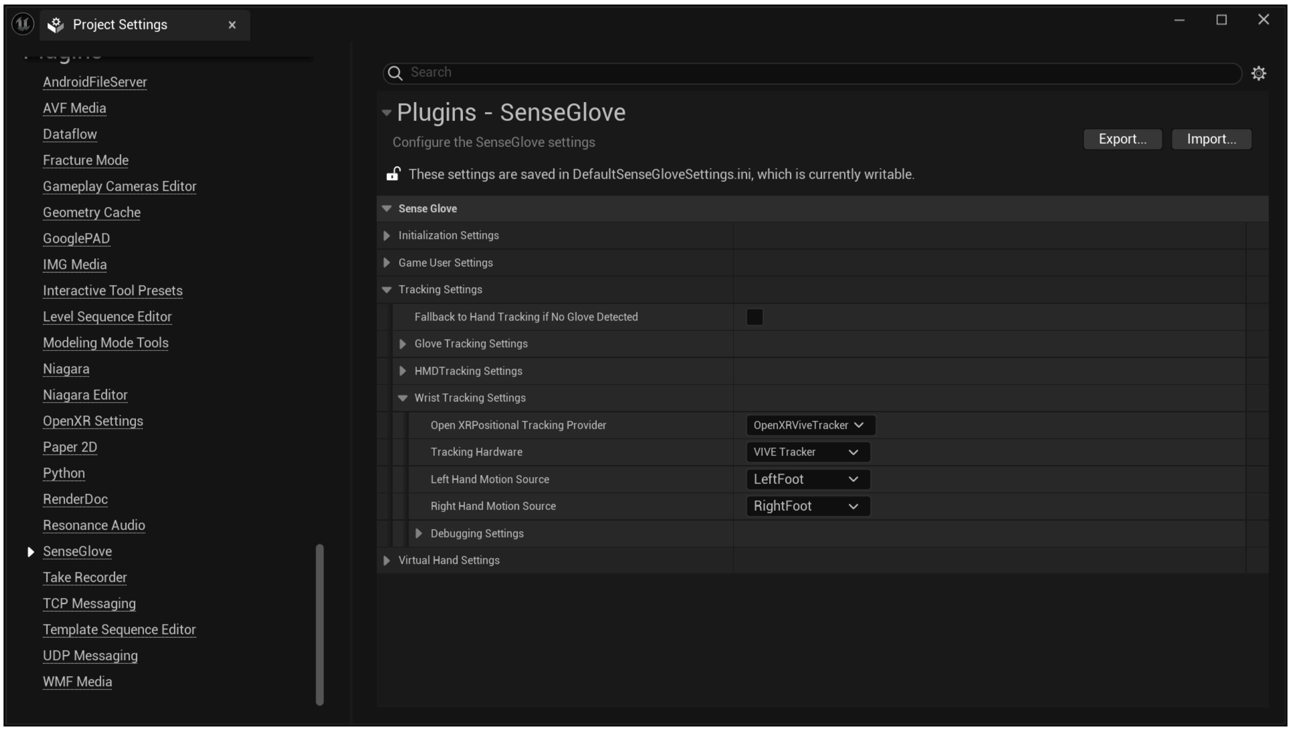
Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

- Sense Glove
  - Initialization Settings
  - Game User Settings
  - Tracking Settings
    - Fallback to Hand Tracking if No Glove Detected: ☐
    - Glove Tracking Settings
    - HMDTracking Settings
    - Wrist Tracking Settings
      - Open XRPositional Tracking Provider: OpenXR
      - Tracking Hardware: Quest 3 Controller
      - Left Hand Motion Source: Left
      - Right Hand Motion Source: Right
      - Debugging Settings
  - Virtual Hand Settings

HTC VIVE Tracker

For the HTC VIVE Pro using VIVE Trackers, which only supports the PCVR mode, the functional wrist-tracking settings looks something like this:



Plugins - SenseGlove

Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

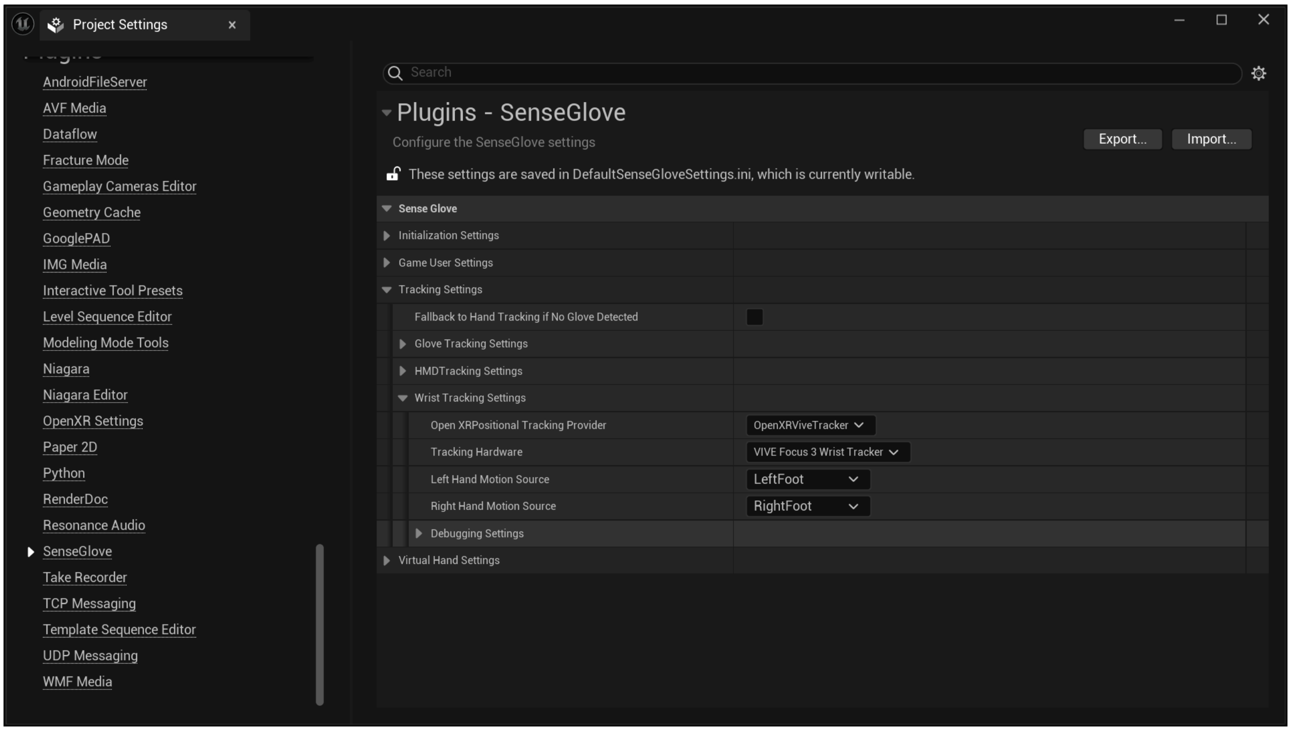
Sense Glove
- Initialization Settings
- Game User Settings
- Tracking Settings
  - Fallback to Hand Tracking if No Glove Detected
  - Glove Tracking Settings
  - HMDTracking Settings
  - Wrist Tracking Settings
    - Open XRPositional Tracking Provider: OpenXRViveTracker
    - Tracking Hardware: VIVE Tracker
    - Left Hand Motion Source: LeftFoot
    - Right Hand Motion Source: RightFoot
    - Debugging Settings
- Virtual Hand Settings

HTC VIVE Focus 3 Wrist Tracker

The wrist-tracking settings for the HTC VIVE Focus Vision, VIVE XR Elite, and VIVE Focus 3, when using VIVE Wrist Trackers, will vary depending on the platform and configuration in use.

PCVR Mode

When running in PCVR mode, the functional wrist-tracking settings looks something like this:



Plugins - SenseGlove

Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

- Sense Glove
  - Initialization Settings
  - Game User Settings
  - Tracking Settings
    - Fallback to Hand Tracking if No Glove Detected: ☐
    - Glove Tracking Settings
    - HMDTracking Settings
    - Wrist Tracking Settings
      - Open XRPositional Tracking Provider: OpenXRViveTracker
      - Tracking Hardware: VIVE Focus 3 Wrist Tracker
      - Left Hand Motion Source: LeftFoot
      - Right Hand Motion Source: RightFoot
      - Debugging Settings
  - Virtual Hand Settings

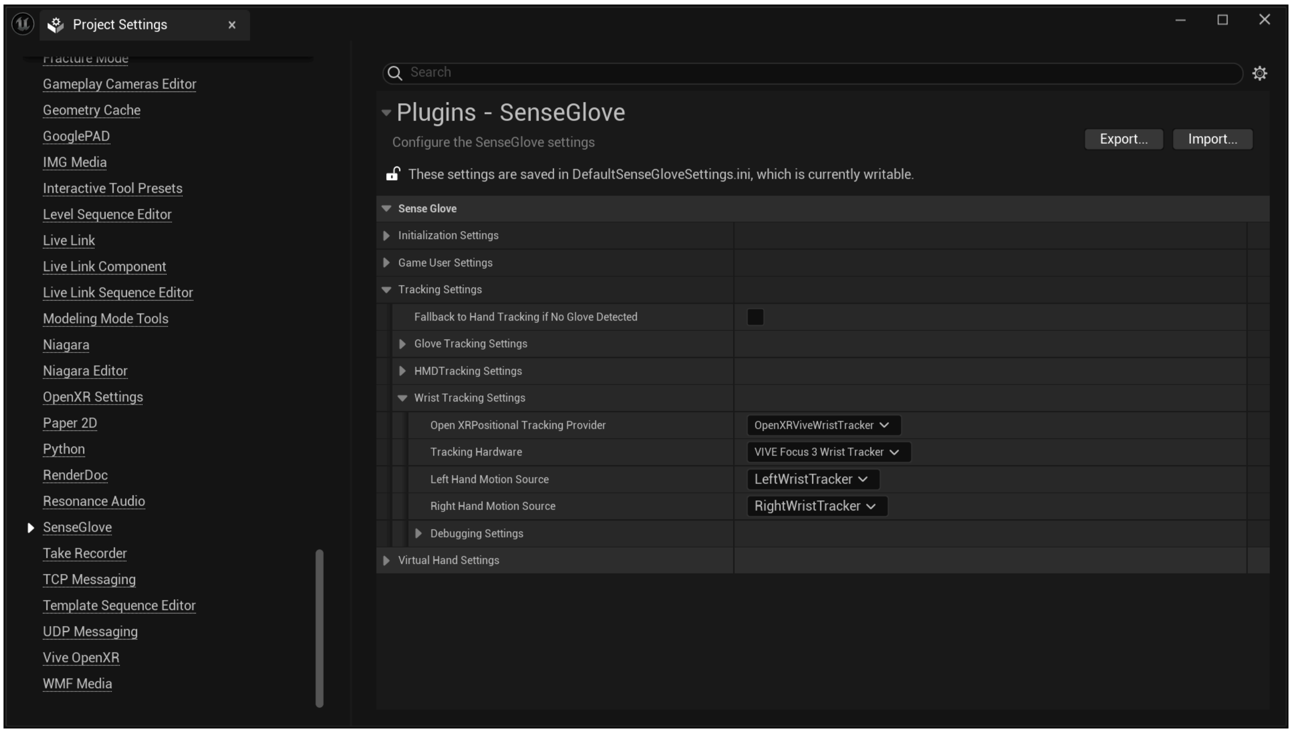
Standalone Mode

Caution

The SenseGlove Unreal Engine Plugin `v2.7.x` is the last release series to support Unreal Engine `5.4`, and its support will be removed in future minor or major releases. This is important to keep in mind if your target development and deployment platform is HTC VIVE in Standalone Mode. Unfortunately, HTC has not released any updates to their HTC ViveOpenXR plugin since December 6, 2024. Their latest release [1] [2], ViveOpenXR Plugin `v2.5.0`, supports only Unreal Engine `5.3` and `5.4`. HTC VIVE PCVR Mode is unaffected and will remain fully functional because, on Microsoft Windows, it is supported via the OpenXRViveTracker Plugin, which is bundled with Unreal Engine and officially maintained by Epic Games. If you still intend to target HTC in Standalone Mode, you are welcome to continue using the latest SenseGlove Unreal Engine Plugin `v2.7.x`, which will retain HTC Standalone Mode support. However, please keep in mind that once newer versions of the SenseGlove Unreal Engine Plugin are released and UE `5.4` is no longer supported, the latest release of the plugin supporting UE `5.4` will not receive new features, hardware support, or bug fixes.

> If at any point in the future HTC releases a new version of their ViveOpenXR plugin that supports any Unreal Engine version we actively support,+ in accordance with our support policy and Platform Support Matrix, we will make every reasonable effort to reintroduce HTC Standalone Mode support.

When running in Standalone mode, the functional wrist-tracking settings looks something like this:



Plugins - SenseGlove

Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

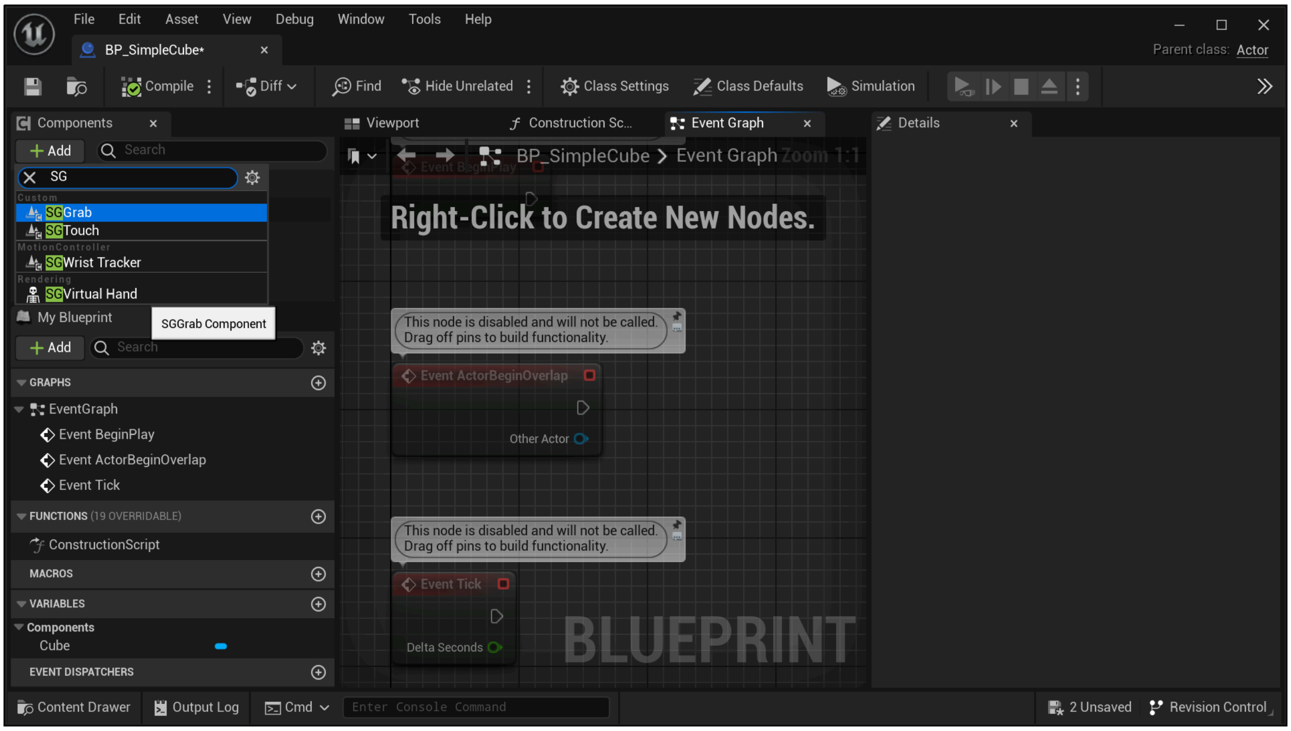
- Sense Glove
  - Initialization Settings
  - Game User Settings
  - Tracking Settings
    - Fallback to Hand Tracking if No Glove Detected: ☐
    - Glove Tracking Settings
    - HMDTracking Settings
    - Wrist Tracking Settings
      - Open XRPositional Tracking Provider: OpenXRViveWristTracker
      - Tracking Hardware: VIVE Focus 3 Wrist Tracker
      - Left Hand Motion Source: LeftWristTracker
      - Right Hand Motion Source: RightWristTracker
      - Debugging Settings
  - Virtual Hand Settings

Setting up the Grab/Release System

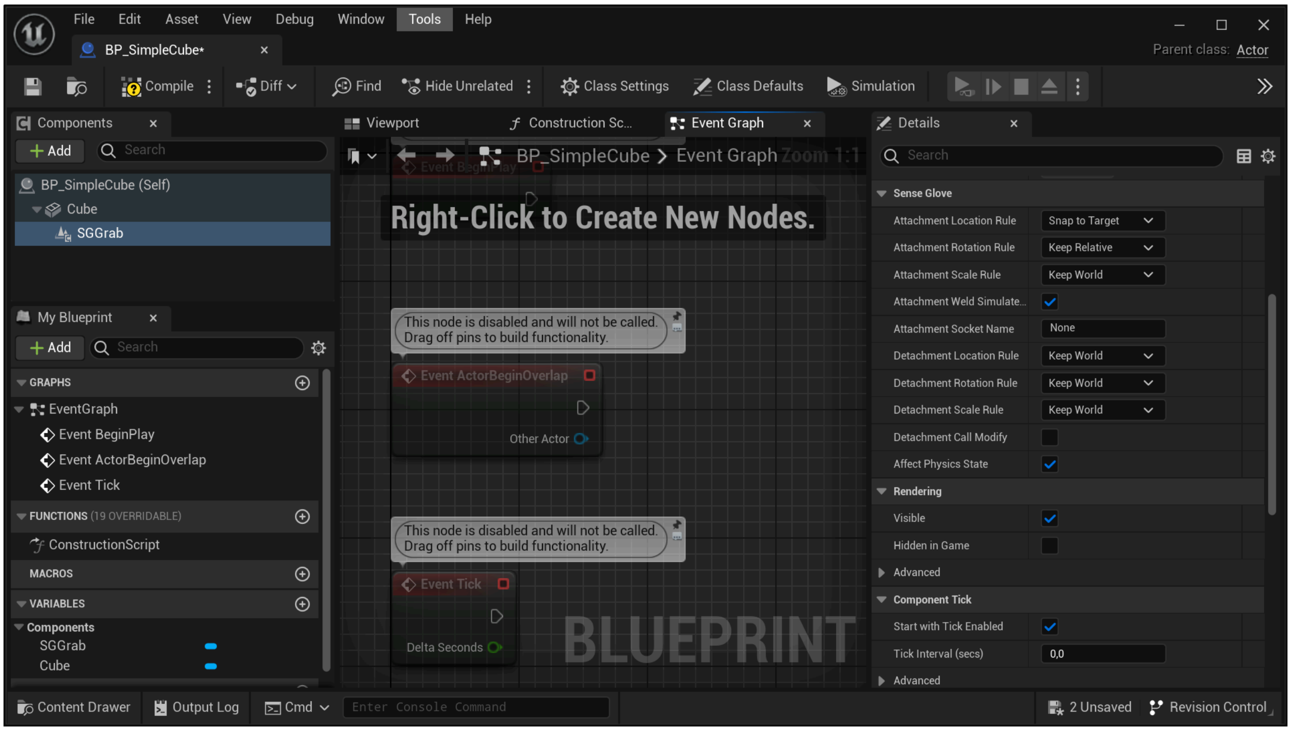
Setting up the SenseGlove Grab/Release System involves two main steps. The first step, configuring the virtual hand meshes for both real and virtual hands, is handled automatically by the plugin. The second step, which is also straightforward, involves setting up any existing actor in the Unreal Blueprint Editor that you want to respond to with haptic feedback when your SenseGlove device comes into contact with it:

1. Open any existing actor in the Unreal Blueprint Editor that you would like to respond to with haptic feedback when your SenseGlove device comes into contact with it.

2. In the `Components` panel, click the `+ Add` button, then type `SGGrab` into the `Search Components` input field. Once found, click on `SGGrab` to add it to the current actor. You can rename the `SGGrab` component to your desired name.



3. With the `SGGrab` component selected in the `Components` panel, navigate to the `Details` panel. Under the `SenseGlove` section, adjust the settings for the grab/release system to suit your needs.



This node is disabled and will not be called.
Drag off pins to build functionality.

This node is disabled and will not be called.
Drag off pins to build functionality.

Sense Glove	
Attachment Location Rule	Snap to Target
Attachment Rotation Rule	Keep Relative
Attachment Scale Rule	Keep World
Attachment Weld Simulate...	✓
Attachment Socket Name	None
Detachment Location Rule	Keep World
Detachment Rotation Rule	Keep World
Detachment Scale Rule	Keep World
Detachment Call Modify	☐
Affect Physics State	✓
Rendering	
Visible	✓
Hidden in Game	☐
Advanced	
Component Tick	
Start with Tick Enabled	✓
Tick Interval (secs)	0,0
Advanced	

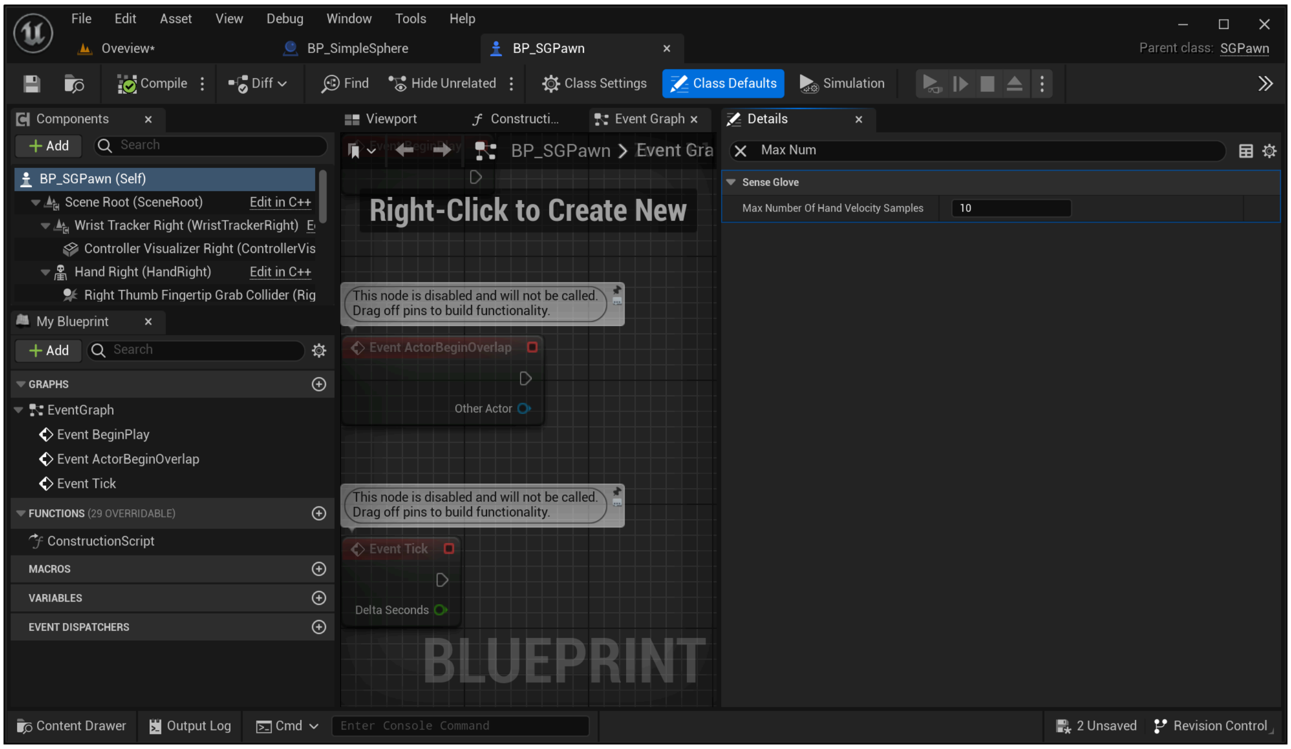
Note

Any property prefixed with `Attachment` is a parameter directly passed to Unreal's `FAttachmentTransformRules` during the grab process, while any property prefixed with `Detachment` is a parameter directly passed to Unreal's `FDetachmentTransformRules` during the release process.

Caution

If `AttachmentSocketName` is unspecified, or incorrect the grabbable object will be attached to the root bone of the virtual hand mesh, which probably is not ideal.

4. A key setting for the release system is located within your `SGPawn` instance. In the `Details` panel for your `SGPawn`, find the `Max Number of Hand Velocity Samples` setting and adjust it according to your needs. This setting determines the velocity of objects released from the hands by averaging the specified number of frames. Optimizing this value depends on the framerate of your simulation at runtime.

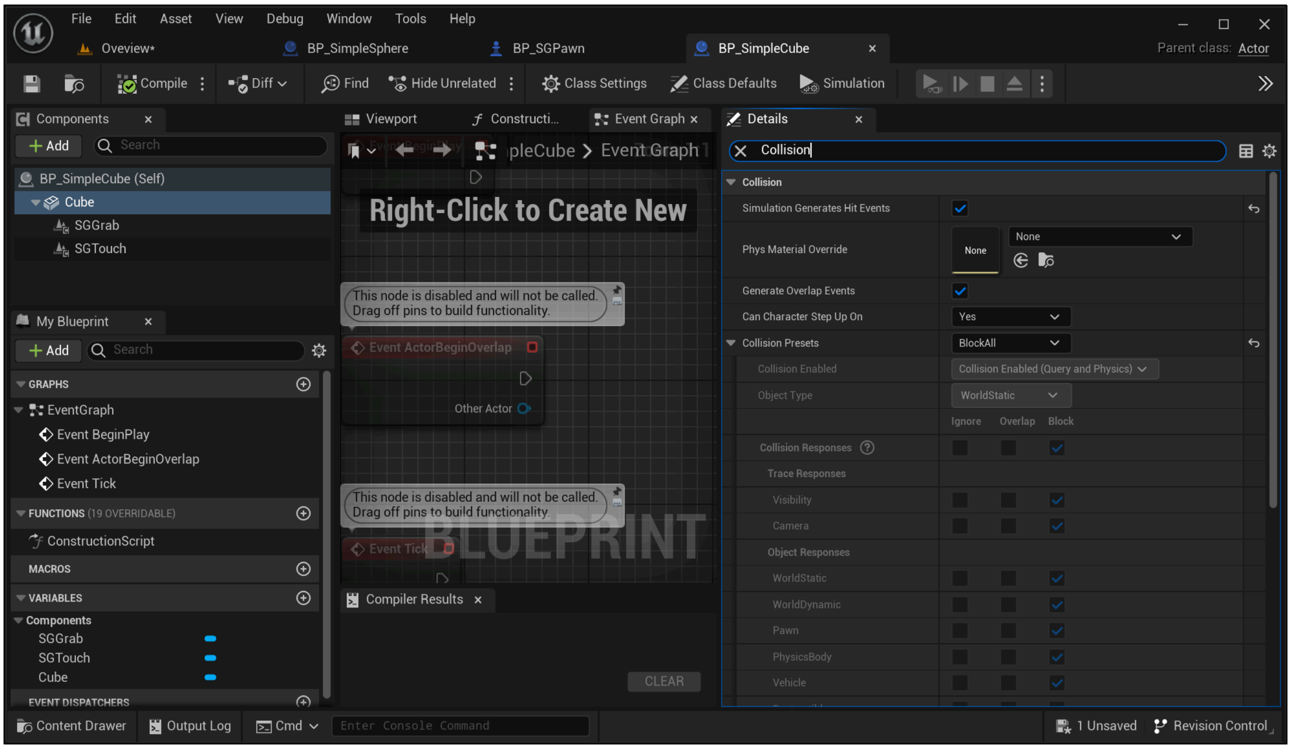


Max Num

Sense Glove

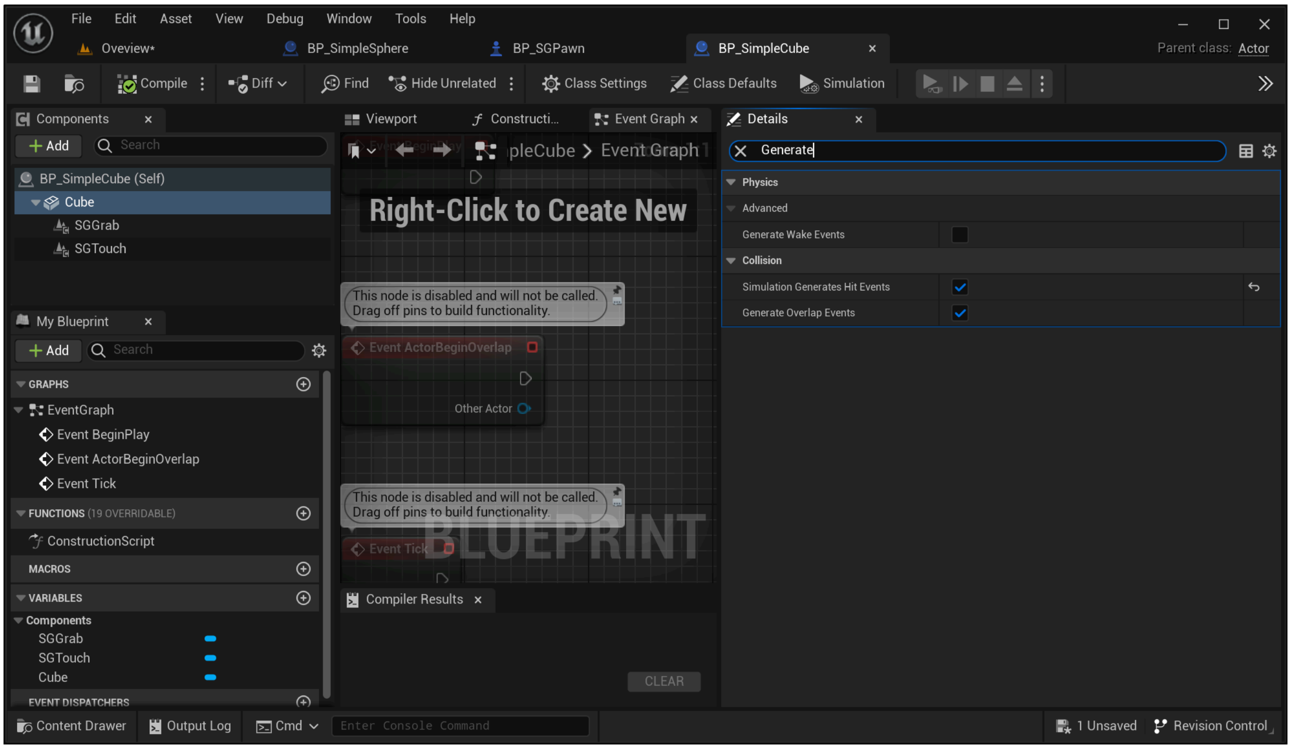
Max Number Of Hand Velocity Samples 10

5. One last aspect of the grabbable actors to take into account for the grab system to function properly is the collision settings of their mesh components. If you'd like to prevent the virtual hand meshes from passing through a grabbable actor, it's necessary to set the `Collision Presets` to `Block All` inside the `Details` panel for the actor's mesh components.



Collision			
Simulation Generates Hit Events	✓		
Phys Material Override	None		
Generate Overlap Events	✓		
Can Character Step Up On	Yes		
Collision Presets	BlockAll		
Collision Enabled	Collision Enabled (Query and Physics)		
Object Type	WorldStatic		
	Ignore	Overlap	Block
Collision Responses			✓
Trace Responses			
Visibility			✓
Camera			✓
Object Responses			
WorldStatic			✓
WorldDynamic			✓
Pawn			✓
PhysicsBody			✓
Vehicle			✓

6. Additionally, enabling `Simulation Generates Hit Events` and `Generate Overlap Events` on the actors mesh components is mandatory. These settings are crucial for notifying the grab system when the virtual hand meshes come into contact with the actor.



Details

Generate

Physics
Advanced
Generate Wake Events ☐
Collision
Simulation Generates Hit Events ✓
Generate Overlap Events ✓

Video Tutorials

The following tutorials, though for much older releases of the plugin, still provide in-depth guidance on the same process:

- Setting up Grabbing and Haptic Feedback functionalities (SGBasicDemo)

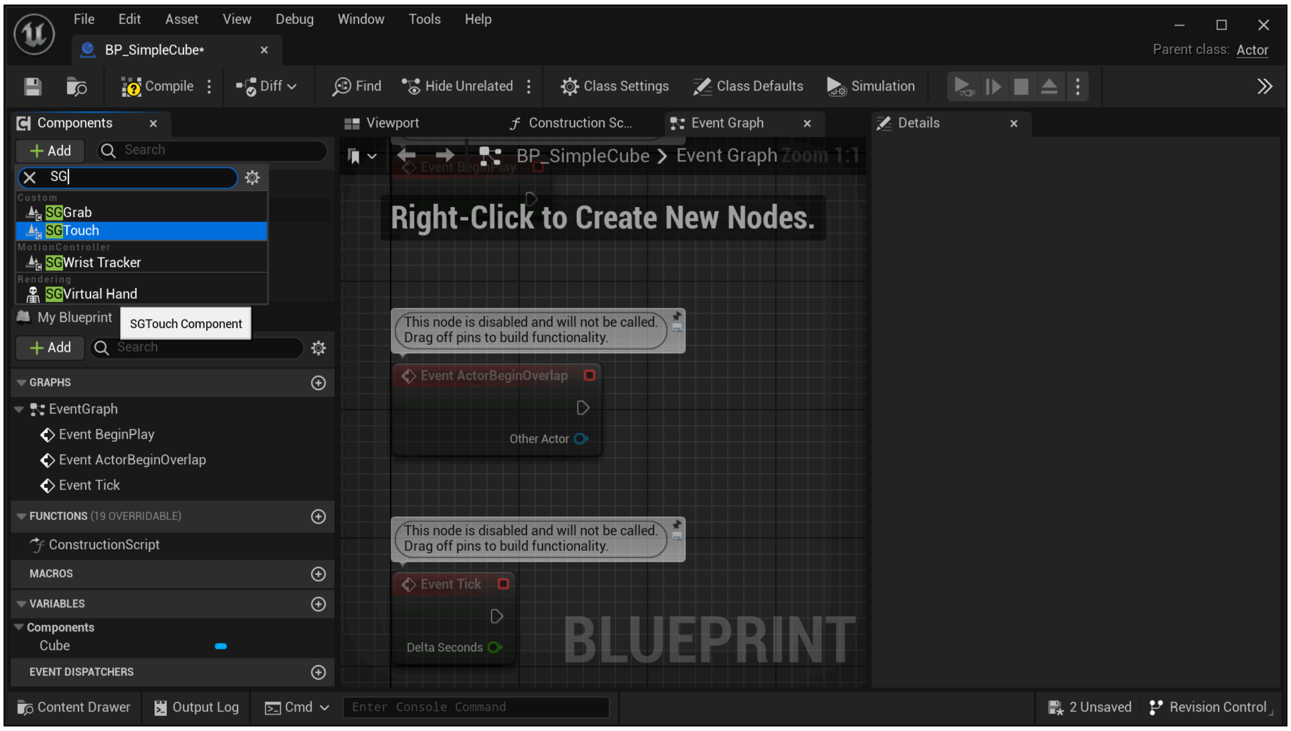
- SGBasicDemo: setup throwing objects and physics settings for the real and virtual hands

Setting up the Touch System

Configuring the SenseGlove Touch System involves two key steps. The first step, which is automatically handled by the plugin, is configuring the virtual hand meshes for both real and virtual hands. The second step, which is also straightforward, involves setting up any existing actor in the Unreal Blueprint Editor that you want to respond to with haptic feedback when your SenseGlove device comes into contact with it:

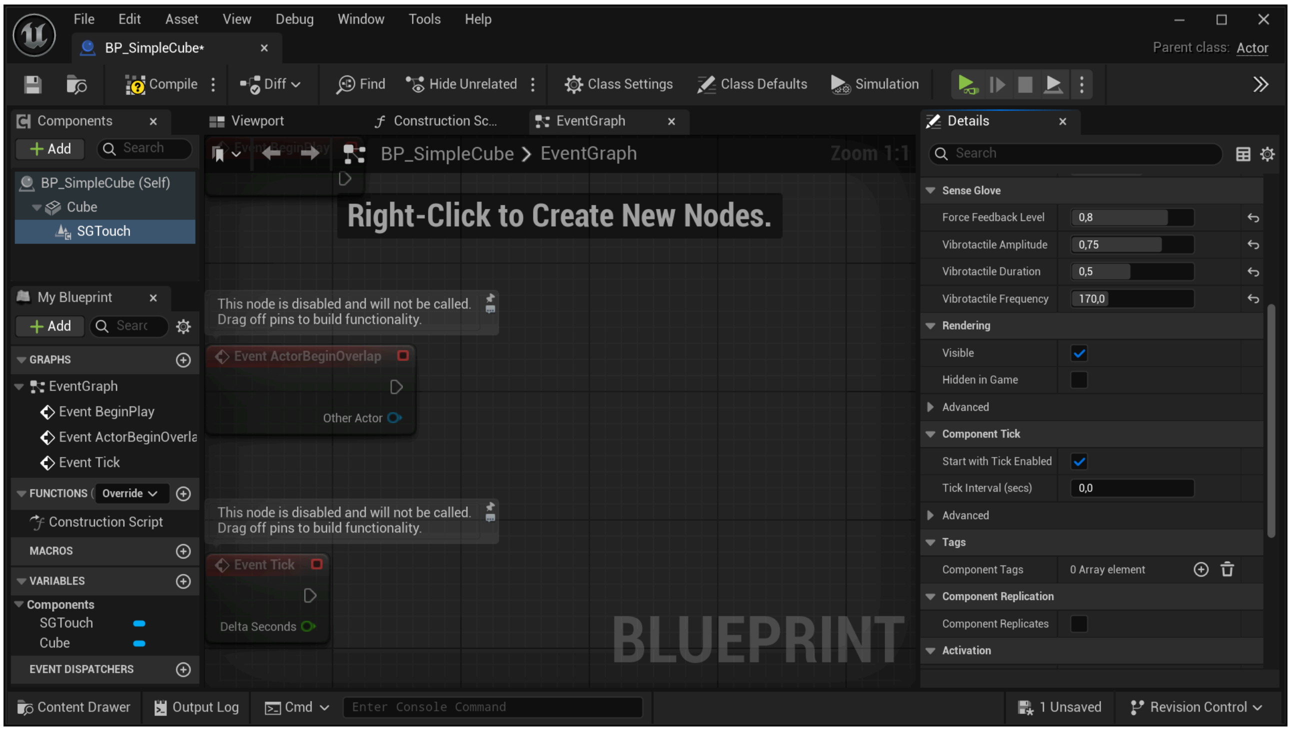
1. Open any existing actor in the Unreal Blueprint Editor that you would like to respond to with haptic feedback when your SenseGlove device comes into contact with it.

2. In the `Components` panel, click the `+ Add` button, then type `SGTouch` into the `Search Components` input field. Once found, click on `SGTouch` to add it to the current actor. You can rename the `SGTouch` component to your desired name.



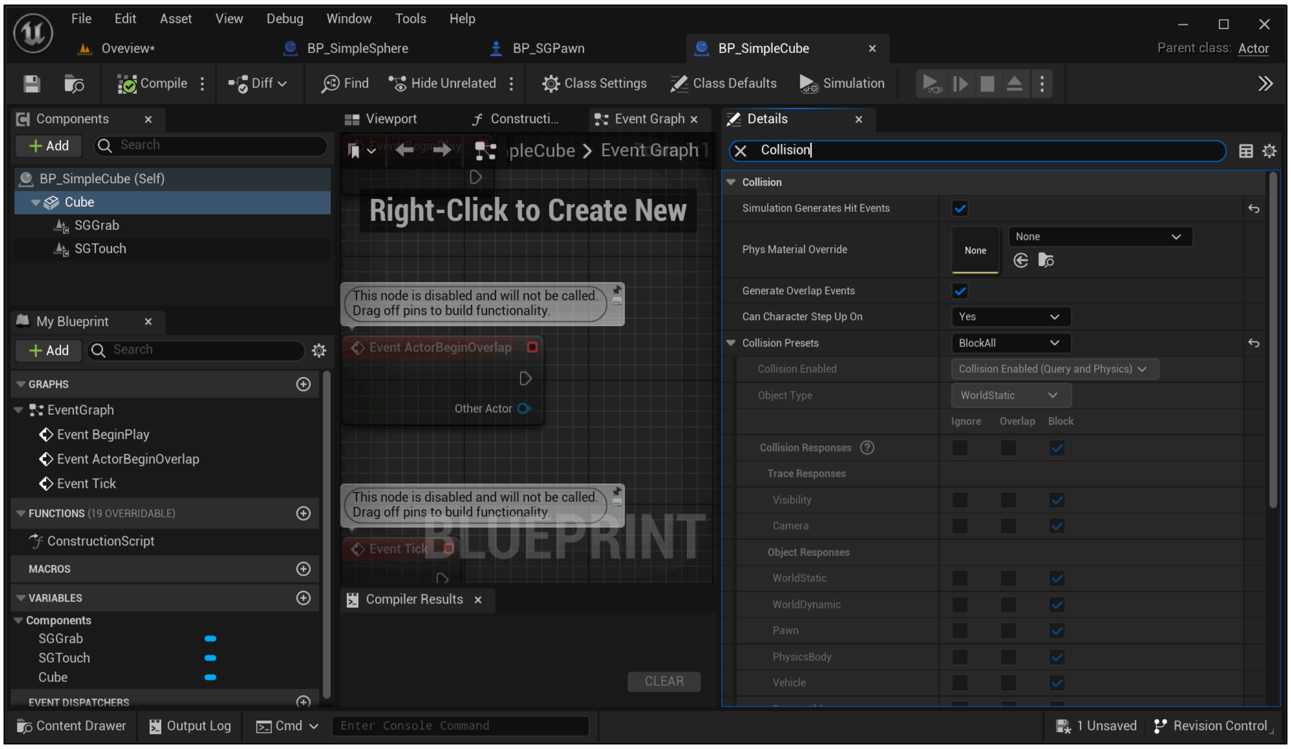
3. With the `SGTouch` component selected in the `Components` panel, navigate to the `Details` panel. Under the `SenseGlove` section, adjust the settings for the touch

system to suit your needs.



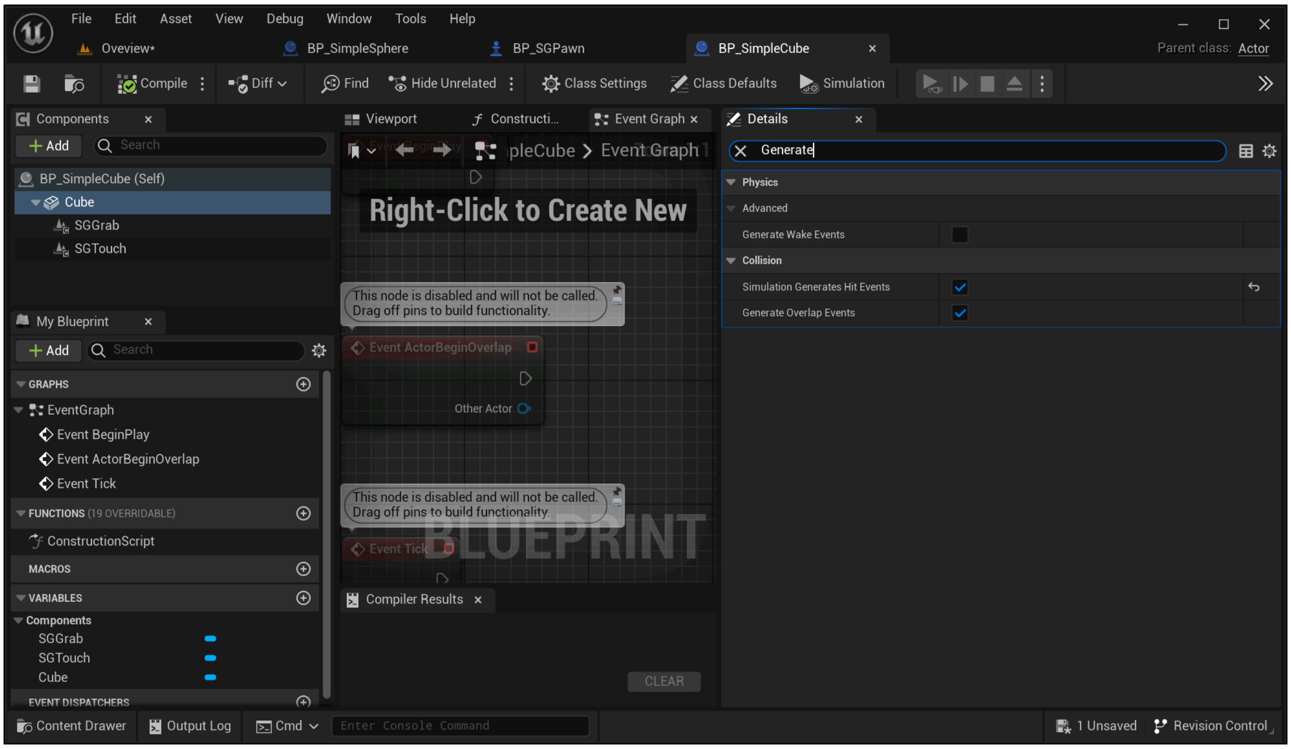
Sense Glove
Force Feedback Level 0,8
Vibrotactile Amplitude 0,75
Vibrotactile Duration 0,5
Vibrotactile Frequency 170,0
Rendering
Visible ✓
Hidden in Game
Advanced
Component Tick
Start with Tick Enabled ✓
Tick Interval (secs) 0,0
Advanced
Tags
Component Tags 0 Array element
Component Replication
Component Replicates
Activation

4. One last aspect of the touchable actors to take into account for the touch system to function properly is the collision settings of their mesh components. If you'd like to prevent the virtual hand meshes from passing through a touchable actor, it's necessary to set the `Collision Presets` to `Block All` inside the `Details` panel for the actor's mesh components.



Collision			
Simulation Generates Hit Events	✓		
Phys Material Override	None		
Generate Overlap Events	✓		
Can Character Step Up On	Yes		
Collision Presets	BlockAll		
Collision Enabled	Collision Enabled (Query and Physics)		
Object Type	WorldStatic		
	Ignore	Overlap	Block
Collision Responses			✓
Trace Responses			
Visibility			✓
Camera			✓
Object Responses			
WorldStatic			✓
WorldDynamic			✓
Pawn			✓
PhysicsBody			✓
Vehicle			✓

5. Additionally, enabling `Simulation Generates Hit Events` and `Generate Overlap Events` on the actors mesh components is mandatory. These settings are crucial for notifying the touch system when the virtual hand meshes come into contact with the actor.



Details — search: Generate

Physics
- Advanced
  - Generate Wake Events: ☐

Collision
- Simulation Generates Hit Events: ✓
- Generate Overlap Events: ✓

Video Tutorials

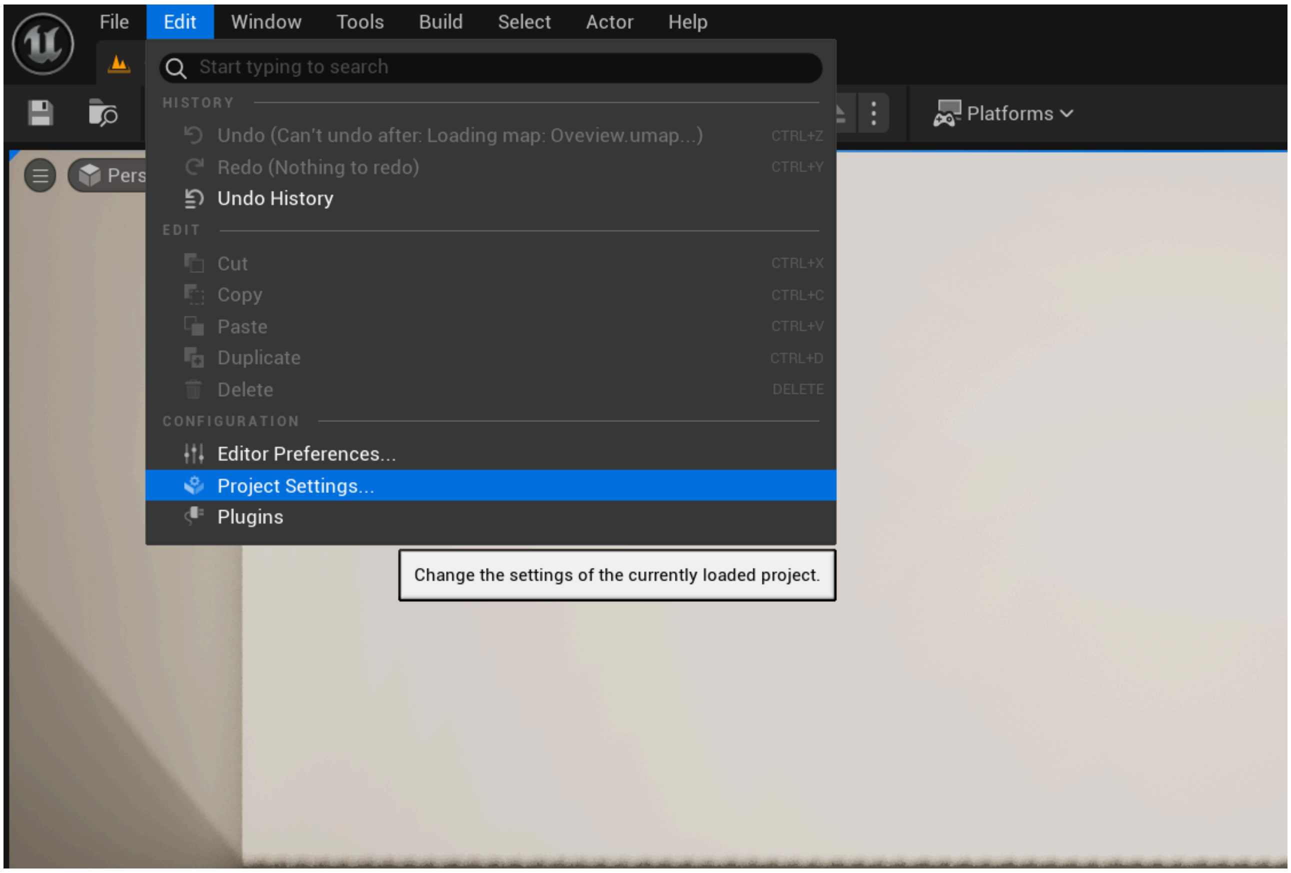
The following tutorials, though for much older releases of the plugin, still provide in-depth guidance on the same process:

- Setting up Grabbing and Haptic Feedback functionalities (SGBasicDemo)

- SGBasicDemo: setup throwing objects and physics settings for the real and virtual hands

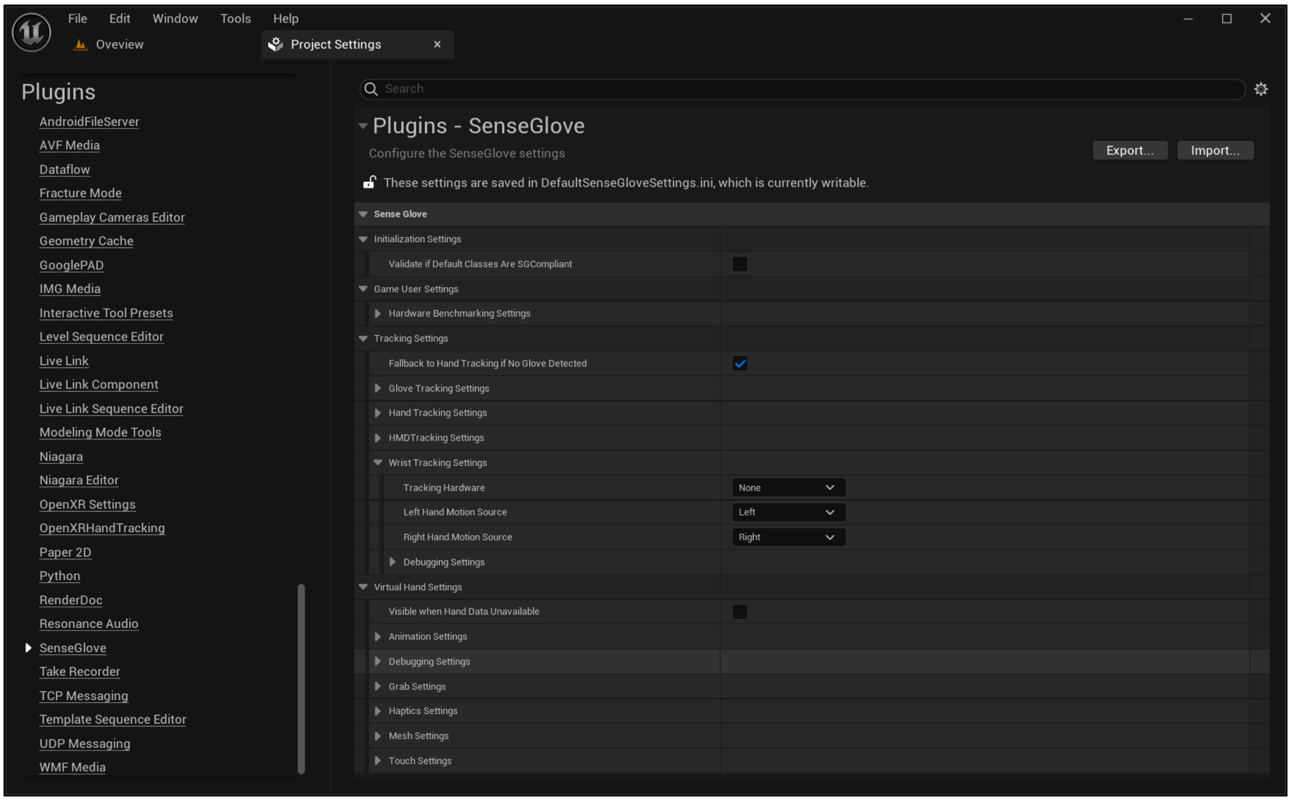
The Plugin Settings

Once the SenseGlove Unreal Engine Plugin is enabled the plugin settings can be accessed through `Edit > Project Setting...` inside your project's Unreal Editor.



Change the settings of the currently loaded project.

The SenseGlove Unreal Engine Plugin offers fine-grained control over various aspects of its functionality through its settings system. It also allows you to override specific settings from subcomponents when possible. In the following sections, we will explore the settings and the override system in detail.



Plugins

- AndroidFileServer
- AVF Media
- Dataflow
- Fracture Mode
- Gameplay Cameras Editor
- Geometry Cache
- GooglePAD
- IMG Media
- Interactive Tool Presets
- Level Sequence Editor
- Live Link
- Live Link Component
- Live Link Sequence Editor
- Modeling Mode Tools
- Niagara
- Niagara Editor
- OpenXR Settings
- OpenXRHandTracking
- Paper 2D
- Python
- RenderDoc
- Resonance Audio
- SenseGlove
- Take Recorder
- TCP Messaging
- Template Sequence Editor
- UDP Messaging
- WMF Media

Plugins - SenseGlove

Configure the SenseGlove settings

These settings are saved in DefaultSenseGloveSettings.ini, which is currently writable.

- Sense Glove
  - Initialization Settings
    - Validate if Default Classes Are SGCompliant: ☐
  - Game User Settings
    - Hardware Benchmarking Settings
  - Tracking Settings
    - Fallback to Hand Tracking if No Glove Detected: ✓
    - Glove Tracking Settings
    - Hand Tracking Settings
    - HMDTracking Settings
    - Wrist Tracking Settings
      - Tracking Hardware: None
      - Left Hand Motion Source: Left
      - Right Hand Motion Source: Right
      - Debugging Settings
  - Virtual Hand Settings
    - Visible when Hand Data Unavailable: ☐
    - Animation Settings
    - Debugging Settings
    - Grab Settings
    - Haptics Settings
    - Mesh Settings
    - Touch Settings

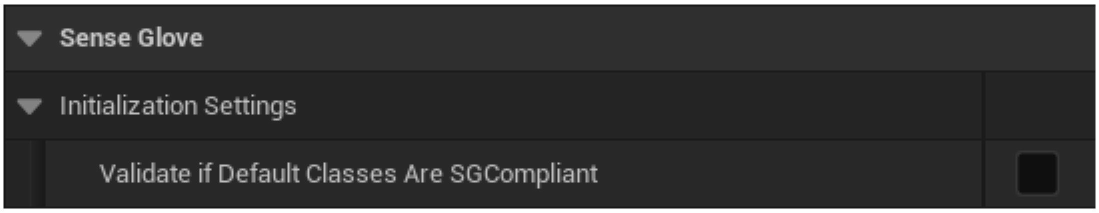
Settings Categories

The plugin settings are organized into four main categories, and each of those might contain its own sub-categories. These main categories are as follows:

- The Initialization Settings
- The Game User Settings
- The Tracking Settings
- The Virtual Hand Settings

The Plugin Initialization Settings

The Initialization Settings section is designed to control how the SenseGlove Unreal Engine Plugin is initialized, allowing you to customize its behavior to suit your project's needs.



Validate if Default Classes Are SGCompliant ☐

bValidateIfDefaultClassesAreSGCompliant

If enabled, the plugin tries to check and validate whether the default for classes such as GameMode, GameInstance, etc. are indeed SenseGlove classes or SenseGlove-derived classes. If not, it attempts to set them. If you don't like this behavior for whatever reason, consider disabling this option.

By default, this option is disabled.

Caution

Due to the current initialization mechanism, setting the default classes might occasionally fail. Therefore, it's essential to verify that the default classes have been correctly set. You can do this by checking the following sections in the project settings:

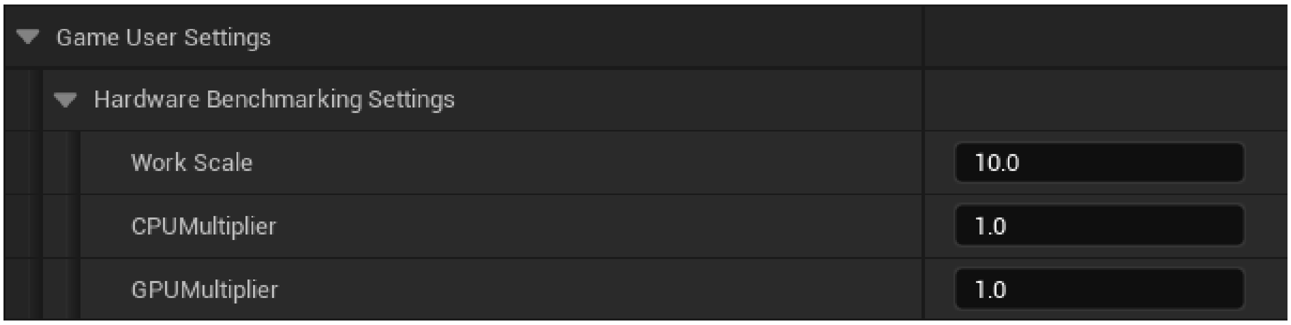
- `Project Settings > Project > Maps & Modes > Default Modes > Default GameMode`
- `Project Settings > Project > Maps & Modes > Default Modes > Selected GameMode > Default Pawn Class`
- `Project Settings > Project > Maps & Modes > Default Modes > Selected GameMode > Player Controller Class`

> - `Project Settings > Project > Maps & Modes > Game Instance > Game Instance Class`
> - `Project Settings > Engine > General Settings > Default Classes > Advanced > Game User Settings Class`

For more information visit the SenseGlove default classes.

The Game User Settings

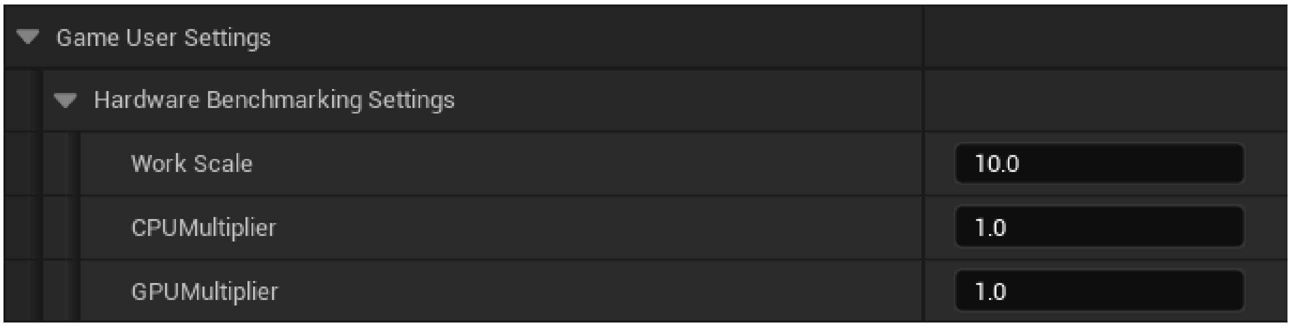
The Game User Settings control the behavior of the SenseGlove instance of `UGameUserSettings`. The `USGGameUserSettings` class extends the functionality of `UGameUserSettings` to provide enhanced customization options specifically for applications that utilize the SenseGlove Unreal Engine Plugin.



Game User Settings	
Hardware Benchmarking Settings	
Work Scale	10.0
CPUMultiplier	1.0
GPUMultiplier	1.0

The Hardware-benchmarking Settings

The settings in this section are utilized by the `USGGameUserSettings::SetEngineScalabilitySettings()` method when the `Scalability` parameter is set to `ESGEngineScalabilitySettings::Auto`. When the engine scalability settings set to auto the graphics settings are determined by running a hardware benchmark by calling the `UGameUserSettings::RunHardwareBenchmark()`. The settings listed here are basically the parameters passed to `UGameUserSettings::RunHardwareBenchmark()`.



Game User Settings	
Hardware Benchmarking Settings	
Work Scale	10.0
CPUMultiplier	1.0
GPUMultiplier	1.0

WorkScale

The `WorkScale` parameter determines the intensity of the benchmark test. Higher values result in more intensive testing, which can help achieve more accurate scalability settings.

The default value is `10`.

CPUMultiplier

The `CPUMultiplier` parameter allows you to adjust the impact of CPU performance on the benchmark results. Increasing this value will emphasize CPU performance more heavily in determining scalability settings.

The default value is `1.0f` .

GPUMultiplier

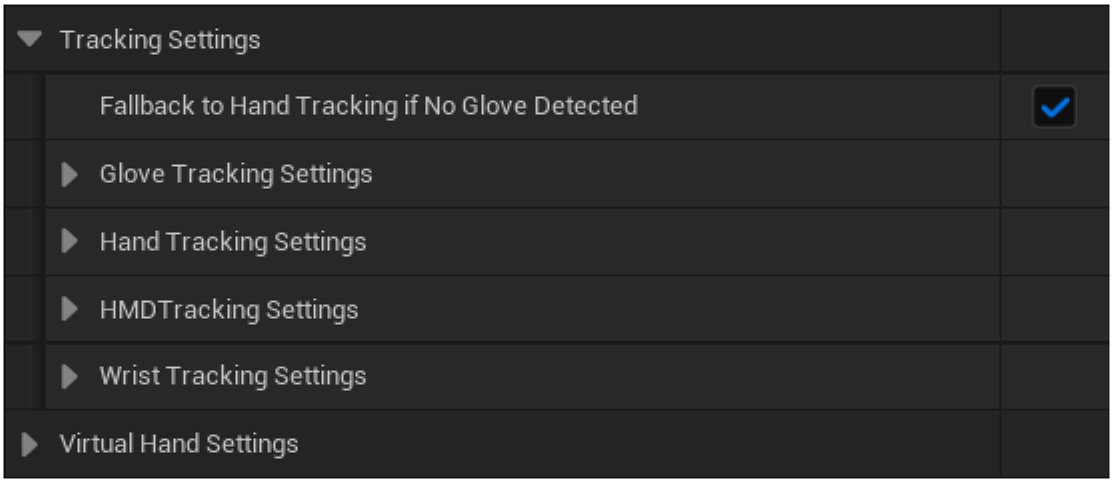
The `GPUMultiplier` parameter lets you modify the influence of GPU performance on the benchmark outcomes. A higher value will increase the weight of GPU performance in setting scalability.

The default value is `1.0f` .

The Tracking Settings

The tracking settings are primarily used by the SenseGlove `Tracking` module and are divided into various subsections, each focusing on a specific aspect of tracking. These subsections, along with the other settings directly provided by this section, provide comprehensive control over the tracking functionalities. The subsections are as follows:

- The Glove-tracking Settings
- The Hand-tracking Settings
- The HMD-tracking Settings
- The Wrist-tracking Settings



Tracking Settings
- Fallback to Hand Tracking if No Glove Detected: ✓
- Glove Tracking Settings
- Hand Tracking Settings
- HMDTracking Settings
- Wrist Tracking Settings

Virtual Hand Settings

bFallbackToHandTrackingIfNoGloveDetected

Determines whether to fallback to hand-tracking, or not, when no SenseGlove device is detected:

- If disabled, only a real glove will be tracked.
- If enabled, the plugin will fall back to hand-tracking when it's available and supported by the HMD device.

Note

> Disabling this option hides the hand-tracking settings section, while enabling it makes the hand-tracking settings visible.

Glove Tracking Settings

Provides the tracking settings related to SenseGlove devices.

Hand Tracking Settings

The settings in this section only affects the hand-tracking functionality when it's enabled and available. When enabled the bare hands can be used instead of SenseGlove devices to interact within the VR simulation, of course without the haptics feedback provided by the SenseGlove devices.

> Important
>
> If you don't see the hand-tracking settings, ensure that the option `bFallbackToHandTrackingIfNoGloveDetected` is checked.

HMD Tracking Settings

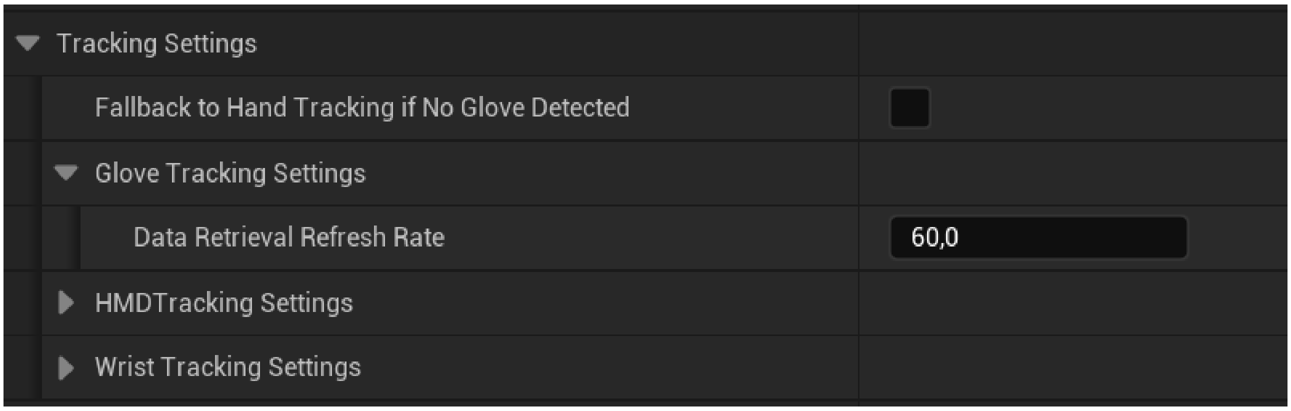
Provides the tracking settings related to head-mounted displays (HDMs) and their auto-detection functionality.

Wrist Tracking Settings

Provides the tracking settings applicable to wrist-tracking hardware.

The Glove-tracking Settings

Provides the tracking settings related to SenseGlove devices.



Tracking Settings

Fallback to Hand Tracking if No Glove Detected

Glove Tracking Settings

Data Retrieval Refresh Rate 60,0

HMDTracking Settings

Wrist Tracking Settings

DataRetrievalRefreshRate

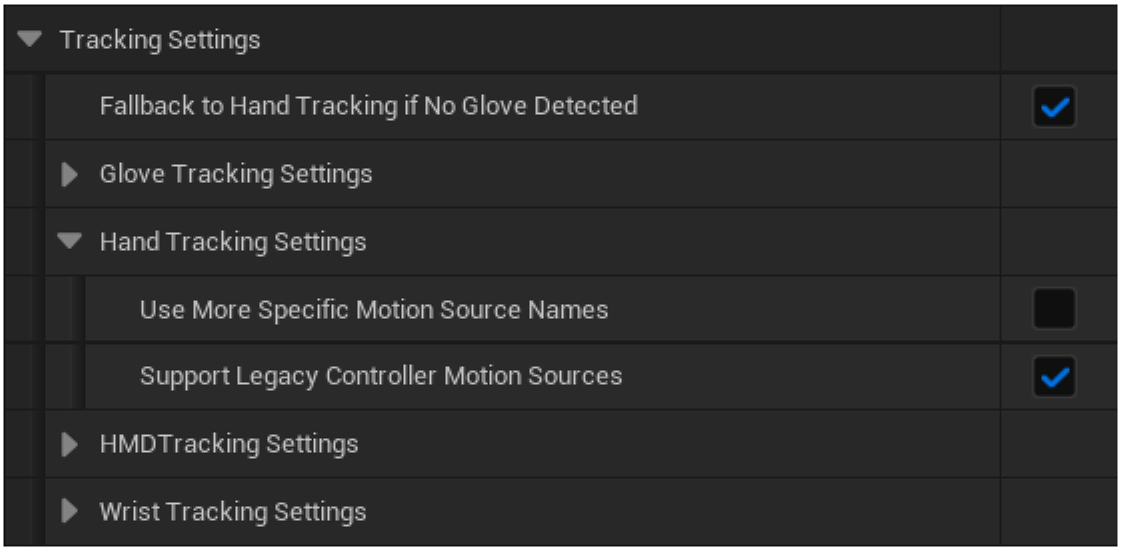
The glove data retrieval refresh rate. This affects the interval in which the tracking module checks for glove connectivity and data retrieval.

The default is s `60Hz` (data retrieval operations per second).

The Hand-tracking Settings

The settings in this section only affects the hand-tracking functionality when it's enabled and available. When enabled the bare hands can be used instead of SenseGlove devices to interact within the VR simulation, of course without the haptics feedback provided by the SenseGlove devices.

> **Important**
>
> If you don't see the hand-tracking settings, ensure that the option `bFallbackToHandTrackingIfNoGloveDetected` is checked.



Tracking Settings	
Fallback to Hand Tracking if No Glove Detected	✓
Glove Tracking Settings	
Hand Tracking Settings	
Use More Specific Motion Source Names	☐
Support Legacy Controller Motion Sources	✓
HMDTracking Settings	
Wrist Tracking Settings	

bUseMoreSpecificMotionSourceNames

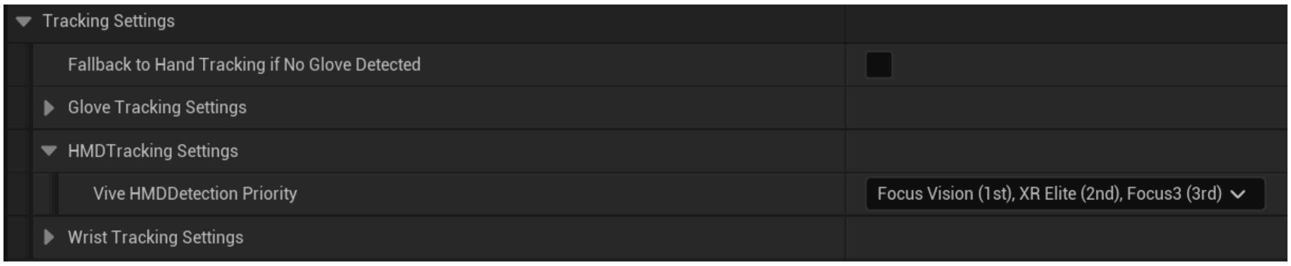
If disabled, (the default) the motion sources for hand tracking will be of the form `[Left|Right][Keypoint]`. If enabled, they will be of the form `HandTracking[Left|Right][Keypoint]`. It is recommended to be enabled to avoid collisions between motion sources from different device types.

bSupportLegacyControllerMotionSources

If enabled, hand tracking supports the `Left` and `Right` legacy motion sources. If disabled, it does not. It is recommended to be disabled unless you need legacy compatibility in older unreal projects.

The HMD-tracking Settings

Provides the tracking settings related to head-mounted displays (HDMs) and their auto-detection functionality.



Tracking Settings

Fallback to Hand Tracking if No Glove Detected

Glove Tracking Settings

HMDTracking Settings

Vive HMDDetection Priority: Focus Vision (1st), XR Elite (2nd), Focus3 (3rd)

Wrist Tracking Settings

ViveHMDDetectionPriority

Determines which VIVE HMD to prioritize for detection, as the current detection mechanism cannot differentiate between HTC VIVE Focus Vision, HTC VIVE XR Elite, and HTC VIVE Focus 3.

The following values are possible:

```
/*
 * The HTC VIVE HMD detection priority for HTC devices that we cannot
distinguish.
 */
UENUM(BlueprintType)
enum class ESGViveHMDDetectionPriority : uint8
{
    /* First try to detect HTC VIVE Focus Vision, then XR Elite, and then
Focus 3. */
    FocusVision_XRElite_Focus3 UMETA(DisplayName = "Focus Vision (1st), XR
Elite (2nd), Focus3 (3rd)"),

    /* First try to detect HTC VIVE Focus Vision, then Focus 3, and then XR
Elite. */
    FocusVision_Focus3_XRElite UMETA(DisplayName = "Focus Vision (1st),
Focus3 (2nd), XR Elite (3rd)"),

    /* First try to detect HTC VIVE XR Elite, then Focus Vision, and then
Focus 3. */
    XRElite_FocusVision_Focus3 UMETA(DisplayName = "XR Elite (1st), Focus
Vision (2nd), Focus3 (3rd)"),

    /* First try to detect HTC VIVE XR Elite, then Focus 3, and then Focus
Vision. */
    XRElite_Focus3_FocusVision UMETA(DisplayName = "XR Elite (1st), Focus3
(2nd), Focus Vision (3rd)"),

    /* First try to detect HTC VIVE Focus 3, then Focus Vision, and then XR
Elite. */
    Focus3_FocusVision_XRElite UMETA(DisplayName = "Focus3 (1st), Focus
Vision (2nd), XR Elite (3rd)"),

    /* First try to detect HTC VIVE Focus 3, then XR Elite, and then Focus
Vision. */
    Focus3_XRElite_FocusVision UMETA(DisplayName = "Focus3 (1st), XR Elite
(2nd), Focus Vision (3rd)"),
};
```

The Wrist-tracking Settings

Provides the tracking settings applicable to wrist-tracking hardware.

OpenXRPositionalTrackingProvider

Specifies the type of OpenXR provider to use in order to extract the positional tracking data from the underlying XR system. If set to None, the plugin attempts to set this automatically by considering a combination of approaches including, the current value of `TrackingHardware` specified below, the platform it's targeted at, the available OpenXR plugins, along with HMD auto-detection mechanism to specify a compatible OpenXR tracking provider. Please note that the OpenXR provider depends on the combination of plugins, platform, and the settings you use. For example, it is possible to use a Vive Focus 3 Wrist Tracker on Microsoft Windows along with the Epic `OpenXRViveTracker` plugin and check the option `Emulate VIVE Wrist Tracker as VIVE Tracker` inside the VIVE Business Streaming application's Input settings. In that case, the correct OpenXR positional tracking provider to use would be `OpenXRViveTracker`. However, using the official ViveOpenXR plugin on Android, the correct OpenXR provider would be `OpenXRViveWristTracker`.

> **Caution**
>
> HMD auto-detection is currently an experimental feature and may fail because HMD vendors occasionally change the properties utilized by the plugin for HMD detection. If you encounter issues, such as incorrect tracker offsets, it is recommended to explicitly specify the tracking hardware.

> **Caution**
>
> Due to highly experimental nature of the HMD auto-detection feature, the HTC VIVE Focus Vision, HTC VIVE Focus 3, and HTC VIVE XR Elite cannot be distinguished from each other in the current iteration. However, since the tracker devices and offsets for all these headsets are the same, this should not affect the

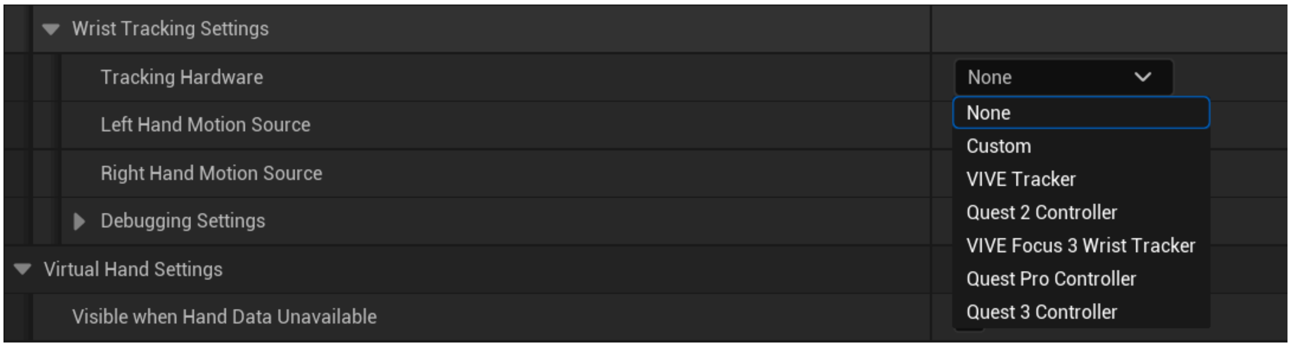
> performance or any functionality. The order in which the HMD is detected can be specified through the HMD-tracker setting `ViveHMDDetectionPriority`.

TrackingHardware

Specifies the type of tracking hardware to use. If set to `None`, the plugin attempts at HMD auto-detection to automatically specify a compatible tracking hardware. If set to `Custom`, any desired location and rotation can be specified.

At the moment the following hardware are supported:

- Quest 2 Controllers
- Quest 3 Controllers
- Quest Pro Controllers
- VIVE Focus 3 Wrist Trackers
- VIVE Trackers


Wrist Tracking Settings
Tracking Hardware: None
Left Hand Motion Source
Right Hand Motion Source
Debugging Settings
Virtual Hand Settings
Visible when Hand Data Unavailable

> **Caution**
>
> HMD auto-detection is currently an experimental feature and may fail because HMD vendors occasionally change the properties utilized by the plugin for HMD detection. If you encounter issues, such as incorrect tracker offsets, it is recommended to explicitly specify the tracking hardware.

> **Caution**

> Due to highly experimental nature of the HMD auto-detection feature, the HTC VIVE Focus Vision, HTC VIVE Focus 3, and HTC VIVE XR Elite cannot be distinguished from each other in the current iteration. However, since the tracker devices and offsets for all these headsets are the same, this should not affect the performance or any functionality. The order in which the HMD is detected can be specified through the HMD-tracker setting `ViveHMDDetectionPriority`.

TrackingHardwareLocationOffsetLeftHand

Sets a custom location offset for left hand's wrist-tracking hardware.

> **Note**
>
> This setting is visible and valid only if TrackingHardware is set to `Custom`.

TrackingHardwareLocationOffsetRightHand

Sets a custom location offset for right hand's wrist-tracking hardware.

> **Note**
>
> This setting is visible and valid only if TrackingHardware is set to `Custom`.

TrackingHardwareRotationOffsetLeftHand

Sets a custom rotation offset for left hand's wrist-tracking hardware.

> **Note**
>
> This setting is visible and valid only if TrackingHardware is set to `Custom`.

TrackingHardwareRotationOffsetRightHand

Sets a custom rotation offset for right hand's wrist-tracking hardware.

> ⓘ Note
>
> This setting is visible and valid only if TrackingHardware is set to `Custom`.

LeftHandMotionSource

Determines the motion source for the left hand. For Oculus HMDs, this is usually `Left`, and for VIVE HMDs using VIVE Wrist Trackers, VIVE Business Streaming, and SteamVR, it's typically `LeftFoot`. For the `OpenXRVive` plugin on Android Standalone Mode using the VIVE Wrist Trackers, this typically is `LeftWristTracker`.

> ⓘ Note
>
> For VIVE devices using SteamVR, the motion source hardware for the left hand can be specified by the user through the SteamVR app.

RightHandMotionSource

Determines the motion source for the right hand. For Oculus HMDs, this is usually `Right`, and for VIVE HMDs using VIVE Wrist Trackers, VIVE Business Streaming, and SteamVR, it's typically `RightFoot`. For the `OpenXRVive` plugin on Android Standalone Mode using the VIVE Wrist Trackers, this typically is `RightWristTracker`.

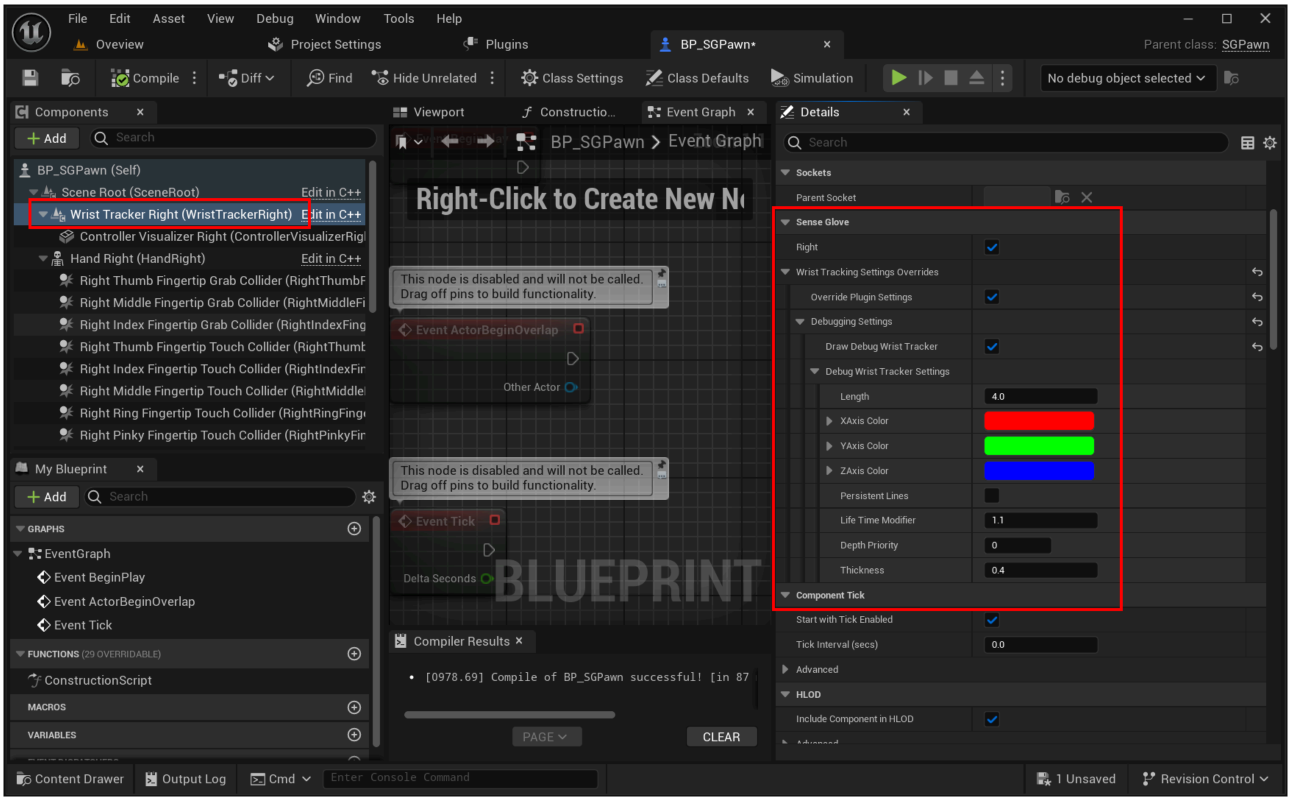
> ⓘ Note
>
> For VIVE devices using SteamVR, the motion source hardware for the right hand can be specified by the user through the SteamVR app.

DebuggingSettings

Provides debugging options for visually debugging the wrist tracker.

Overriding the Wrist-tracking Settings from the Wrist Tracker Component

It's possible to override some of the wrist tracker settings through the details panel of any specific Wrist Tracker Component. When overriden by enabling the `SenseGlove > Wrist Tracking Settings Override > Override Plugin Settings` option inside the details panel, these settings take precedence over the plugin's global settings.



File Edit Asset View Debug Window Tools Help
Overview
Project Settings
Plugins
BP_SGPawn*
Parent class: SGPawn
Compile
Diff
Find
Hide Unrelated
Class Settings
Class Defaults
Simulation
No debug object selected
Components
Add
Search
BP_SGPawn (Self)
Scene Root (SceneRoot) Edit in C++
Wrist Tracker Right (WristTrackerRight) Edit in C++
Controller Visualizer Right (ControllerVisualizerRig
Hand Right (HandRight) Edit in C++
Right Thumb Fingertip Grab Collider (RightThumbF
Right Middle Fingertip Grab Collider (RightMiddleFi
Right Index Fingertip Grab Collider (RightIndexFing
Right Thumb Fingertip Touch Collider (RightThumt
Right Index Fingertip Touch Collider (RightIndexFir
Right Middle Fingertip Touch Collider (RightMiddle
Right Ring Fingertip Touch Collider (RightRingFing
Right Pinky Fingertip Touch Collider (RightPinkyFir
My Blueprint
GRAPHS
EventGraph
Event BeginPlay
Event ActorBeginOverlap
Event Tick
FUNCTIONS (29 OVERRIDABLE)
ConstructionScript
MACROS
VARIABLES
Viewport
Constructio...
Event Graph
BP_SGPawn > Event Graph
Right-Click to Create New N
This node is disabled and will not be called. Drag off pins to build functionality.
Event ActorBeginOverlap
Other Actor
This node is disabled and will not be called. Drag off pins to build functionality.
Event Tick
Delta Seconds
BLUEPRINT
Compiler Results
PAGE
CLEAR
Details
Search
Content Drawer
Output Log
Cmd
Enter Console Command
1 Unsaved
Revision Control

Sockets
Parent Socket

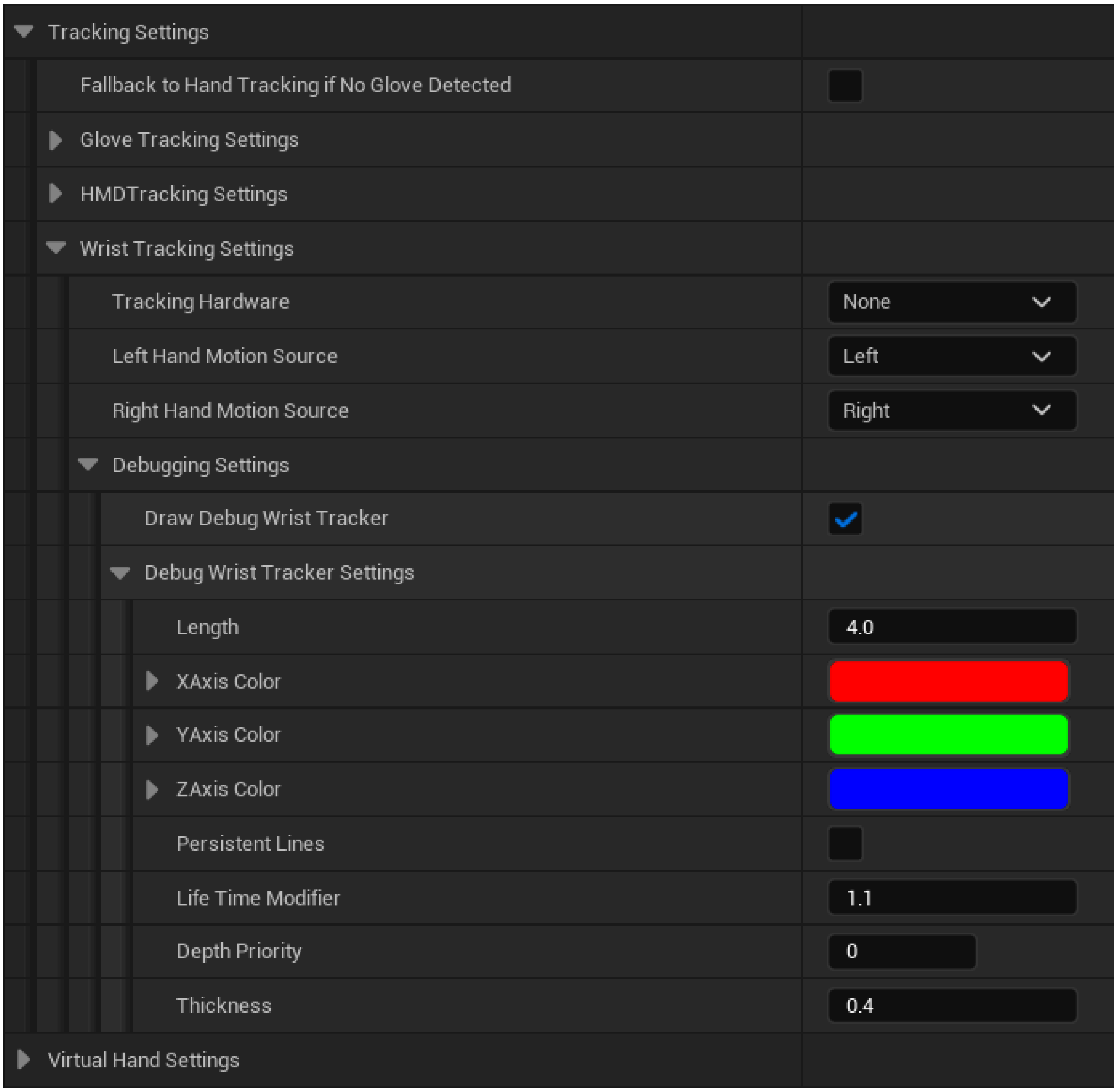
Sense Glove
Right ✓
Wrist Tracking Settings Overrides
Override Plugin Settings ✓
Debugging Settings
Draw Debug Wrist Tracker ✓
Debug Wrist Tracker Settings
Length 4.0
XAxis Color 🟥
YAxis Color 🟩
ZAxis Color 🟦
Persistent Lines
Life Time Modifier 1.1
Depth Priority 0
Thickness 0.4

Component Tick
Start with Tick Enabled ✓
Tick Interval (secs) 0.0
Advanced
HLOD
Include Component in HLOD ✓

```
• [0978.69] Compile of BP_SGPawn successful! [in 87
```

The Wrist-tracking Debugging Settings

Provides debugging options for visually debugging the wrist tracker.



Tracking Settings	
Fallback to Hand Tracking if No Glove Detected	☐
Glove Tracking Settings	
HMDTracking Settings	
Wrist Tracking Settings	
Tracking Hardware	None
Left Hand Motion Source	Left
Right Hand Motion Source	Right
Debugging Settings	
Draw Debug Wrist Tracker	✓
Debug Wrist Tracker Settings	
Length	4.0
XAxis Color	🟥
YAxis Color	🟩
ZAxis Color	🟦
Persistent Lines	☐
Life Time Modifier	1.1
Depth Priority	0
Thickness	0.4
Virtual Hand Settings	

bDrawDebugWristTracker

If enabled, visualizes the debug wrist trackers where possible.

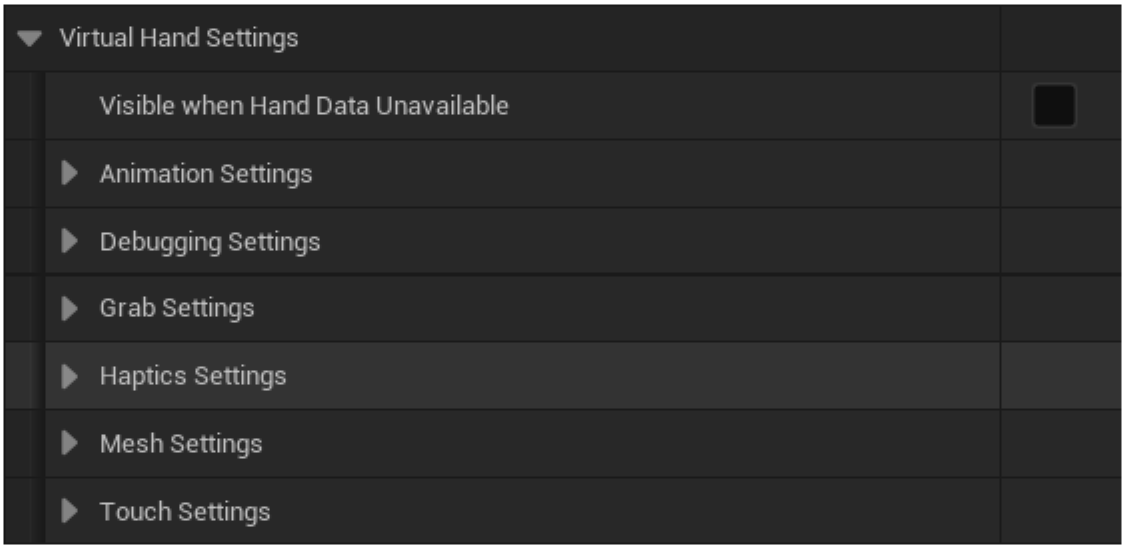
DebugWristTrackerSettings

Visible and valid only if `bDrawDebugGizmo` is enabled.

The Virtual Hand Settings

The Virtual Hand Settings are utilized by various SenseGlove modules such as `Debug`, `Editor`, `Tracking`, and the main module. These settings are divided into several subsections, each focusing on a specific aspect of the virtual hand functionality. Together with the settings provided directly in this section, they offer comprehensive control over any system or component that utilizes the virtual hand. The subsections are as follows:

- The Animation Settings
- The Debugging Settings
- The Grab Settings
- The Haptics Settings
- The Mesh Settings
- The Touch Settings


Virtual Hand Settings
Visible when Hand Data Unavailable ☐
Animation Settings
Debugging Settings
Grab Settings
Haptics Settings
Mesh Settings
Touch Settings

bVisibleWhenHandDataUnavailable

Used by the Virtual Hand Component to determine its visibility when no hand data, either from a SenseGlove or hand-tracking, is available. If enabled, the virtual hand mesh remains visible even when no data is available. By default, this setting is disabled, providing users of the simulation with a clear indicator that no hand data is currently available.

Animation Settings

Controls how the virtual hand model is animated by the animation system.

Debugging Settings

Primarily used for visually debugging low-level hand data. When enabled, the Virtual Hand Component visualizes a debug virtual hand by drawing all individual hand joints.

Grab Settings

Utilized by the SenseGlove Sockets Editor to automatically generate the hand sockets required by the Grab system to function.

The `SGPawn` also utilizes these settings to set up the grab colliders on the virtual hand components.

Haptics Settings

Utilized by the haptics system.

Mesh Settings

Utilized by the SenseGlove `Tracking` module to account for the current virtual hand mesh when generating hand pose data, resulting in more accurate glove or hand data representation and also smoother animations.

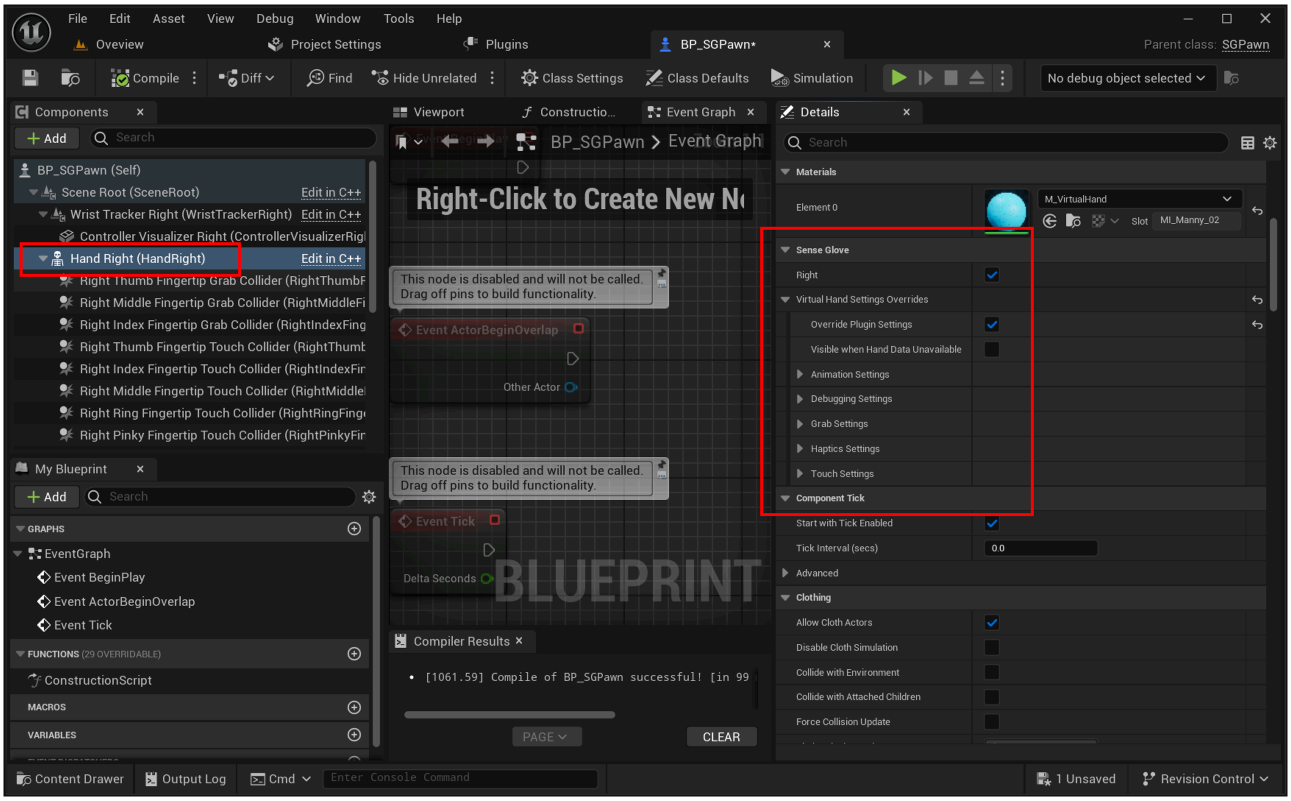
Touch Settings

Utilized by the SenseGlove Sockets Editor to automatically generate the hand sockets required by the Touch system to function.

The `SGPawn` also utilizes these settings to set up the touch colliders on the virtual hand components.

Overriding the Virtual Hand Settings from the Wrist Tracker Component

It's possible to override some of the virtual hand settings through the details panel of any specific Virtual Hand Component. When overriden by enabling the `SenseGlove > Virtual Hand Settings Override > Override Plugin Settings` option inside the details panel, these settings take precedence over the plugin's global settings.



BP_SGPawn (Self)
- Scene Root (SceneRoot) Edit in C++
  - Wrist Tracker Right (WristTrackerRight) Edit in C++
    - Controller Visualizer Right (ControllerVisualizerRight)
  - Hand Right (HandRight) Edit in C++
    - Right Thumb Fingertip Grab Collider (RightThumbF
    - Right Middle Fingertip Grab Collider (RightMiddleFi
    - Right Index Fingertip Grab Collider (RightIndexFing
    - Right Thumb Fingertip Touch Collider (RightThumb
    - Right Index Fingertip Touch Collider (RightIndexFir
    - Right Middle Fingertip Touch Collider (RightMiddle
    - Right Ring Fingertip Touch Collider (RightRingFing
    - Right Pinky Fingertip Touch Collider (RightPinkyFir

EventGraph
- Event BeginPlay
- Event ActorBeginOverlap
- Event Tick

ConstructionScript

This node is disabled and will not be called.
Drag off pins to build functionality.

This node is disabled and will not be called.
Drag off pins to build functionality.

```
• [1061.59] Compile of BP_SGPawn successful! [in 99
```

Materials

Element 0	M_VirtualHand; Slot MI_Manny_02

Sense Glove

Right	✓
Virtual Hand Settings Overrides	
Override Plugin Settings	✓
Visible when Hand Data Unavailable	☐
Animation Settings	
Debugging Settings	
Grab Settings	
Haptics Settings	
Touch Settings	

Component Tick

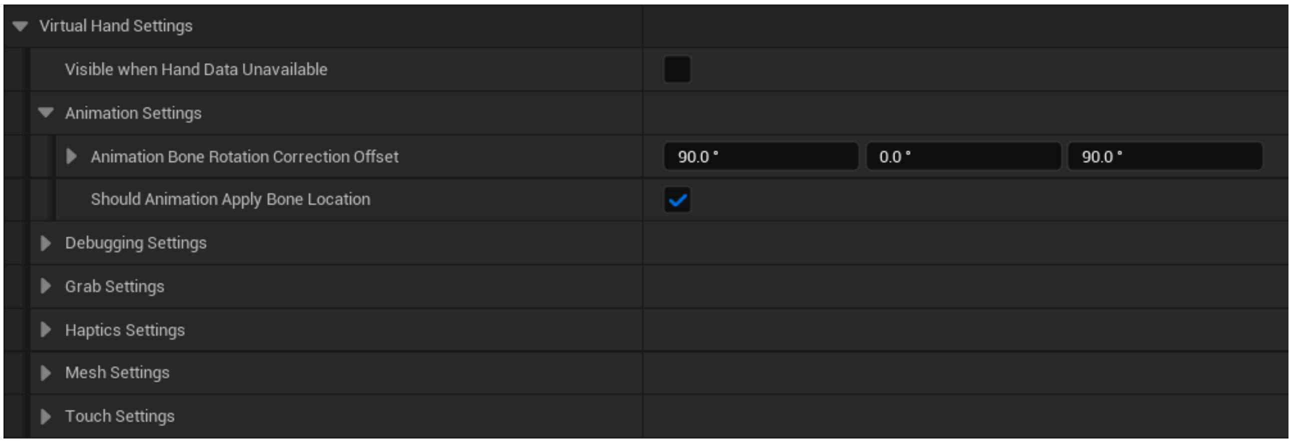
Start with Tick Enabled	✓
Tick Interval (secs)	0.0
Advanced	

Clothing

Allow Cloth Actors	✓
Disable Cloth Simulation	☐
Collide with Environment	☐
Collide with Attached Children	☐
Force Collision Update	☐

The Virtual Hand Animation Settings

Controls how the virtual hand model is animated by the animation system.



Virtual Hand Settings	
Visible when Hand Data Unavailable	☐
Animation Settings	
Animation Bone Rotation Correction Offset	90.0 ° / 0.0 ° / 90.0 °
Should Animation Apply Bone Location	✓
Debugging Settings	
Grab Settings	
Haptics Settings	
Mesh Settings	
Touch Settings	

AnimationBoneRotationCorrectionOffset

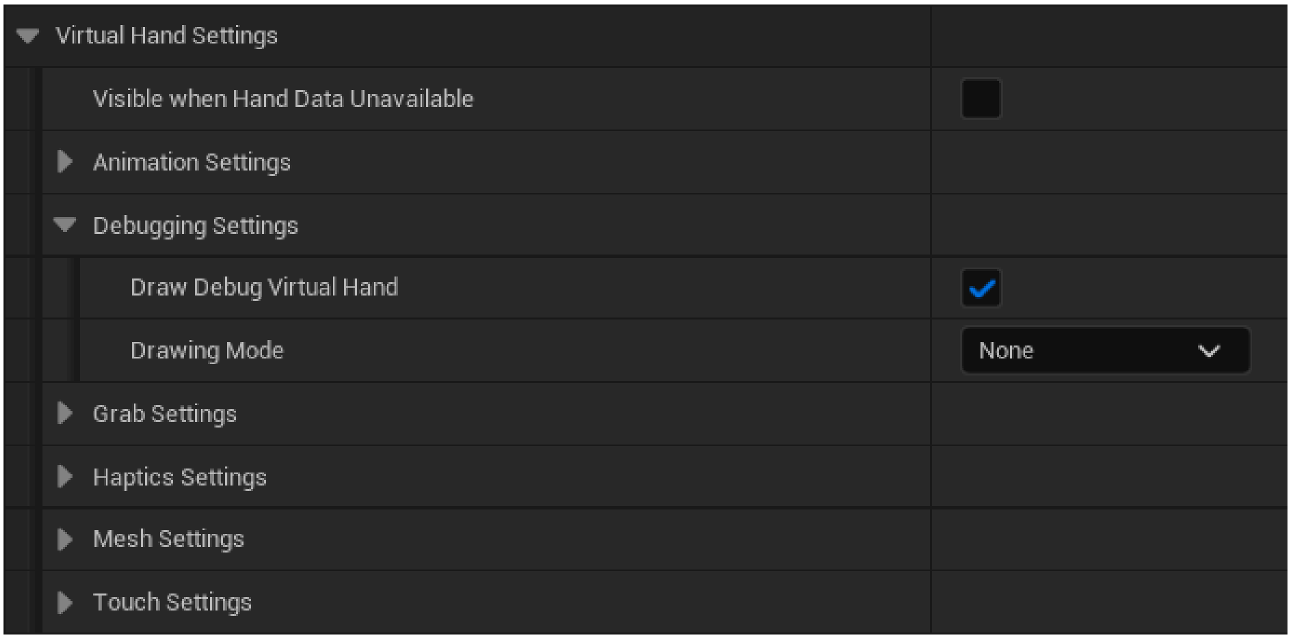
Specifies the offset to apply to each bone's rotation when translating hand pose data to the virtual hand bones. This is useful if the virtual hand mesh was imported with an initial rotation. For example, the virtual hand model shipped with Unreal Engine's VRTemplate typically has an initial `90.0f` degrees rotation on the `Yaw` axis. By default, this option has been set up with the Unreal Engine's VRTemplate virtual hand model in mind.

bShouldAnimationApplyBoneLocation

When enabled, the animation system applies the joint locations to the current virtual hand mesh bones in addition to the joint rotation. Otherwise, only the joint rotations are applied, and joint locations are ignored, leaving the bone locations untouched on the virtual hand mesh when animating it. Enabling this option typically improves the virtual hand animation. By default, this option is enabled.

The Virtual Hand Debugging Settings

Primarily used for visually debugging low-level hand data. When enabled, the Virtual Hand Component visualizes a debug virtual hand by drawing all individual hand joints.



Virtual Hand Settings	
Visible when Hand Data Unavailable	☐
Animation Settings	
Debugging Settings	
Draw Debug Virtual Hand	✓
Drawing Mode	None
Grab Settings	
Haptics Settings	
Mesh Settings	
Touch Settings	

bDrawDebugVirtualHand

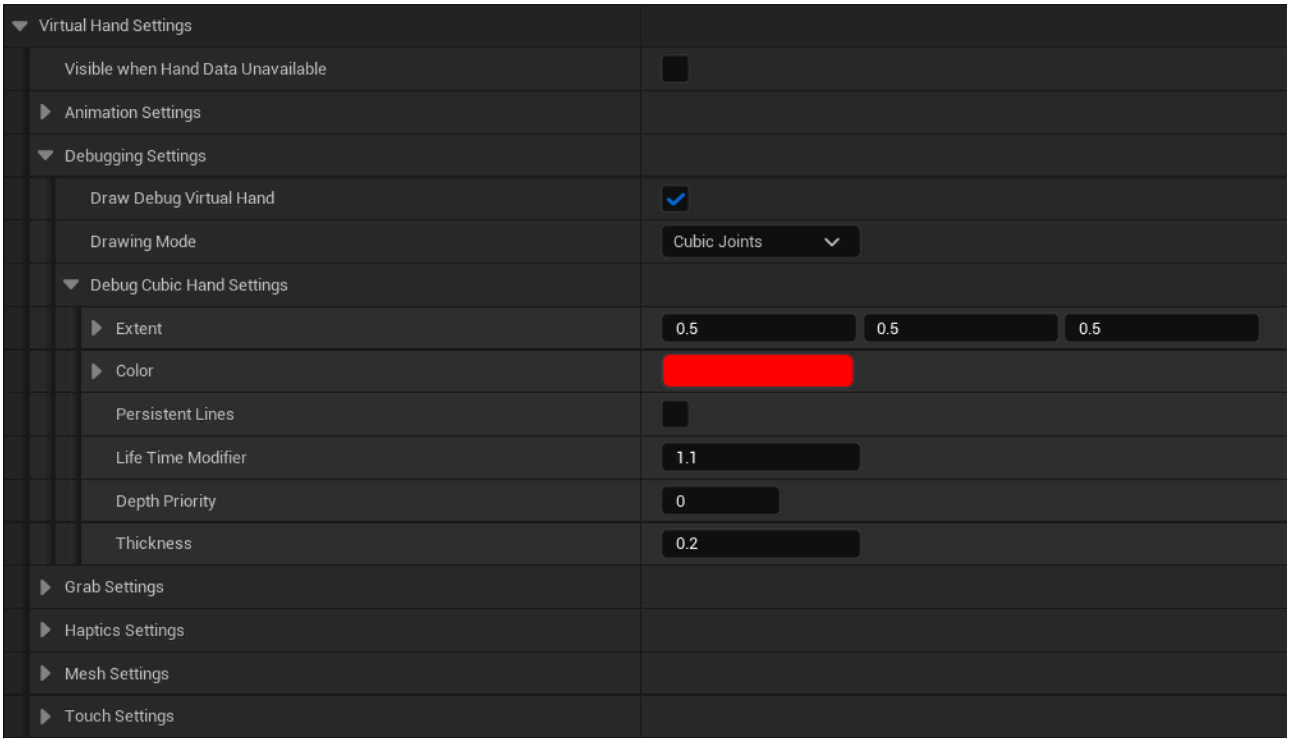
If enabled, visualizes the debug virtual hand where possible.

DrawingMode

Determines the virtual hand drawing mode. If set to `CubicJoints`, for every joint a debug cube will be drawn. If set to `GizmoJoints`, for every joint a debug gizmo will be drawn.

DebugCubicHandSettings

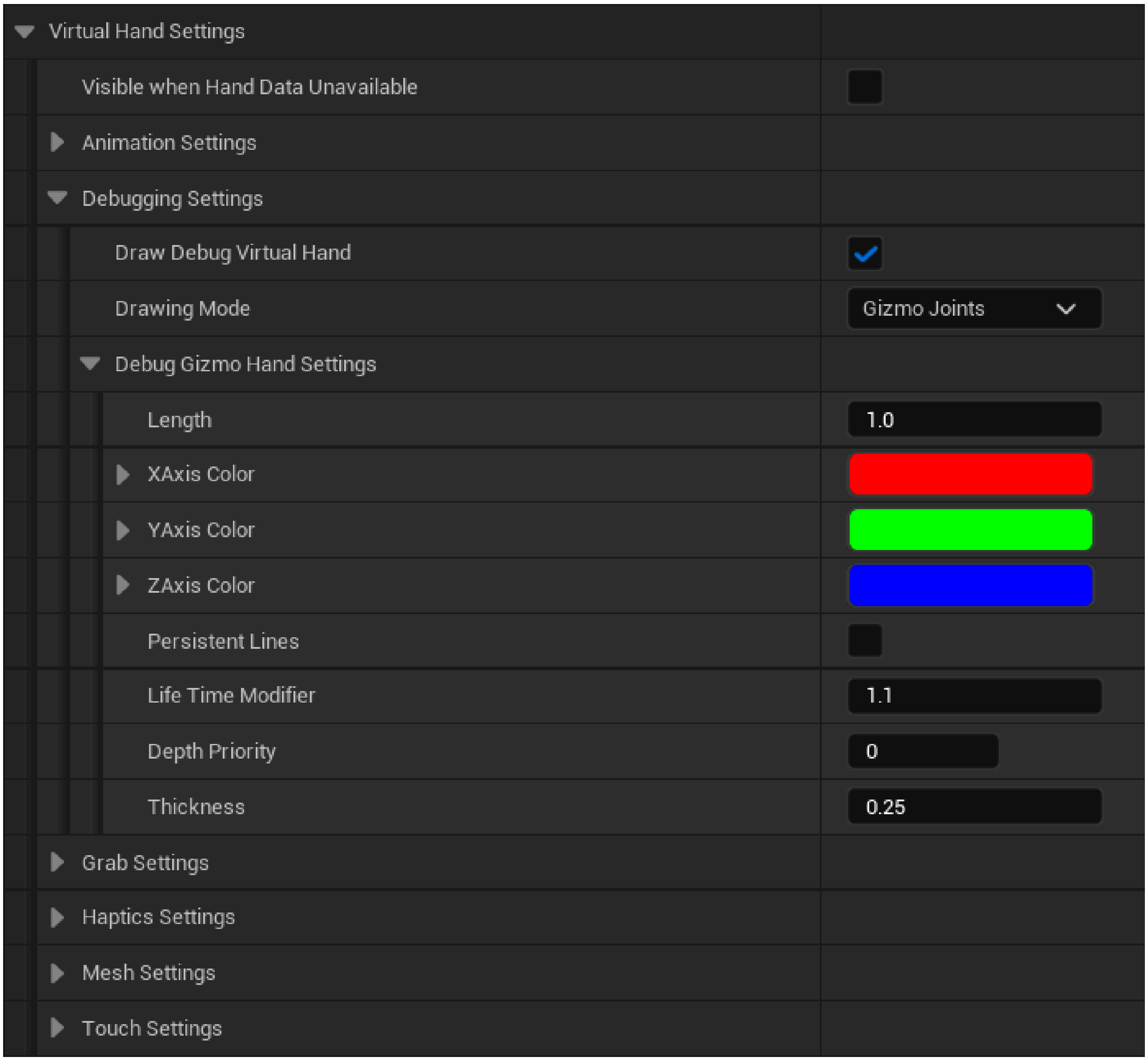
Visible and valid only if `bDrawDebugVirtualHand` is enabled and `DrawingMode` has been set to `ESGDebugVirtualHandDrawingMode::CubicJoints`.



Virtual Hand Settings	
Visible when Hand Data Unavailable	☐
Animation Settings	
Debugging Settings	
Draw Debug Virtual Hand	✓
Drawing Mode	Cubic Joints
Debug Cubic Hand Settings	
Extent	0.5 / 0.5 / 0.5
Color	🟥
Persistent Lines	☐
Life Time Modifier	1.1
Depth Priority	0
Thickness	0.2
Grab Settings	
Haptics Settings	
Mesh Settings	
Touch Settings	

DebugGizmoHandSettings

Visible and valid only if `bDrawDebugVirtualHand` is enabled and `DrawingMode` has been set to `ESGDebugVirtualHandDrawingMode::GizmoJoints`.

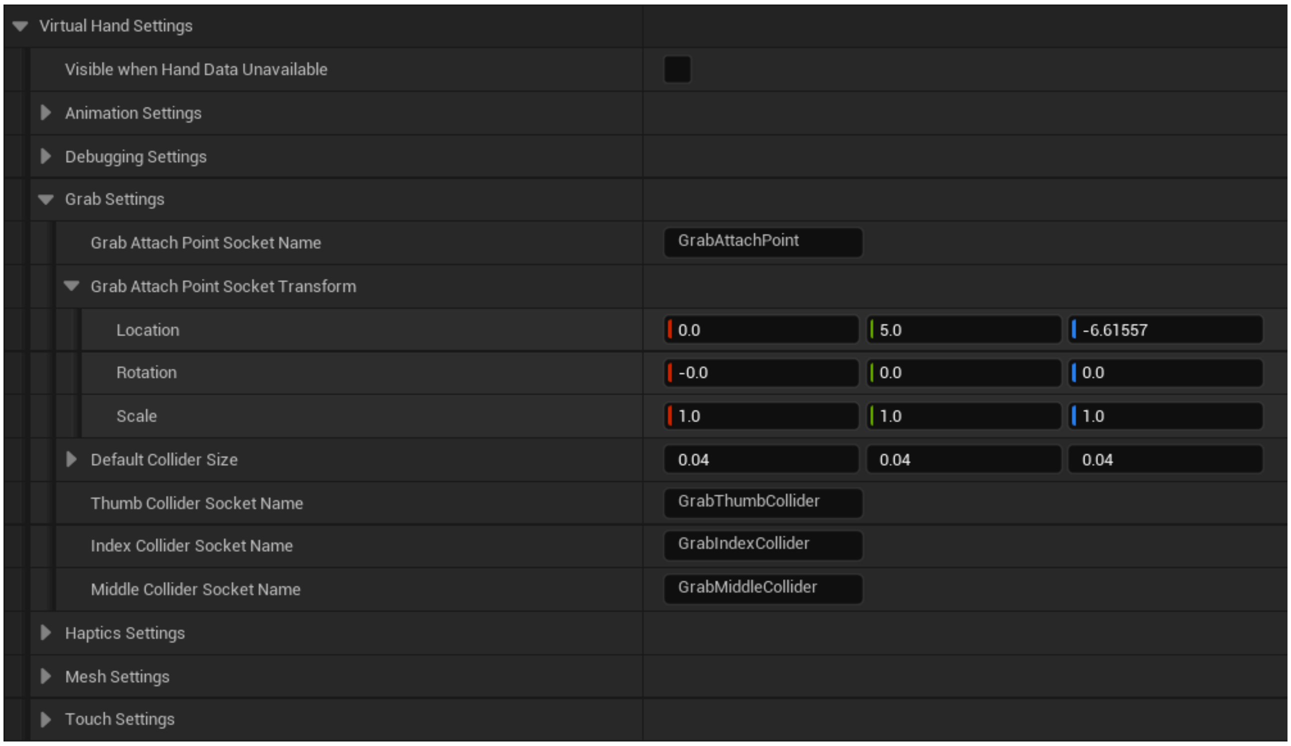


Virtual Hand Settings	
Visible when Hand Data Unavailable	☐
Animation Settings	
Debugging Settings	
Draw Debug Virtual Hand	✓
Drawing Mode	Gizmo Joints
Debug Gizmo Hand Settings	
Length	1.0
XAxis Color	🟥
YAxis Color	🟩
ZAxis Color	🟦
Persistent Lines	☐
Life Time Modifier	1.1
Depth Priority	0
Thickness	0.25
Grab Settings	
Haptics Settings	
Mesh Settings	
Touch Settings	

The Virtual Hand Grab Settings

Utilized by the SenseGlove Sockets Editor to automatically generate the hand sockets required by the Grab system to function.

The `SGPawn` also utilizes these settings to set up the grab colliders on the virtual hand components.



Virtual Hand Settings			
Visible when Hand Data Unavailable	☐		
Animation Settings			
Debugging Settings			
Grab Settings			
Grab Attach Point Socket Name	GrabAttachPoint		
Grab Attach Point Socket Transform			
Location	0.0	5.0	-6.61557
Rotation	-0.0	0.0	0.0
Scale	1.0	1.0	1.0
Default Collider Size	0.04	0.04	0.04
Thumb Collider Socket Name	GrabThumbCollider		
Index Collider Socket Name	GrabIndexCollider		
Middle Collider Socket Name	GrabMiddleCollider		
Haptics Settings			
Mesh Settings			
Touch Settings			

GrabAttachPointSocketName

The default socket name for the grab attach point, usually located at the palm of the hand.

GrabAttachPointSocketTransform

The default socket transform (location, rotation, scale) for the grab attach point, usually located at the palm of the hand.

DefaultColliderSize

The default collider size for the fingers' grab colliders.

ThumbColliderSocketName

The default socket name for the thumb finger's grab collider, usually located at the tip of the thumb finger.

IndexColliderSocketName

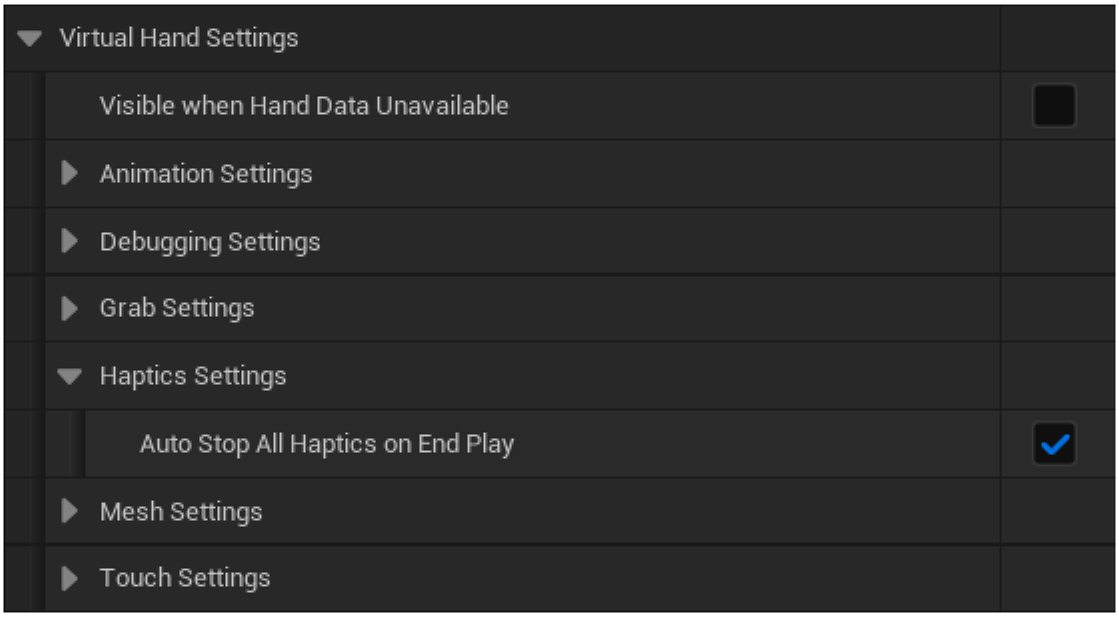
The default socket name for the index finger's grab collider, usually located at the tip of the index finger.

MiddleColliderSocketName

The default socket name for the middle finger's grab collider, usually located at the tip of the middle finger.

The Virtual Hand Haptics Settings

Utilized by the haptics system.



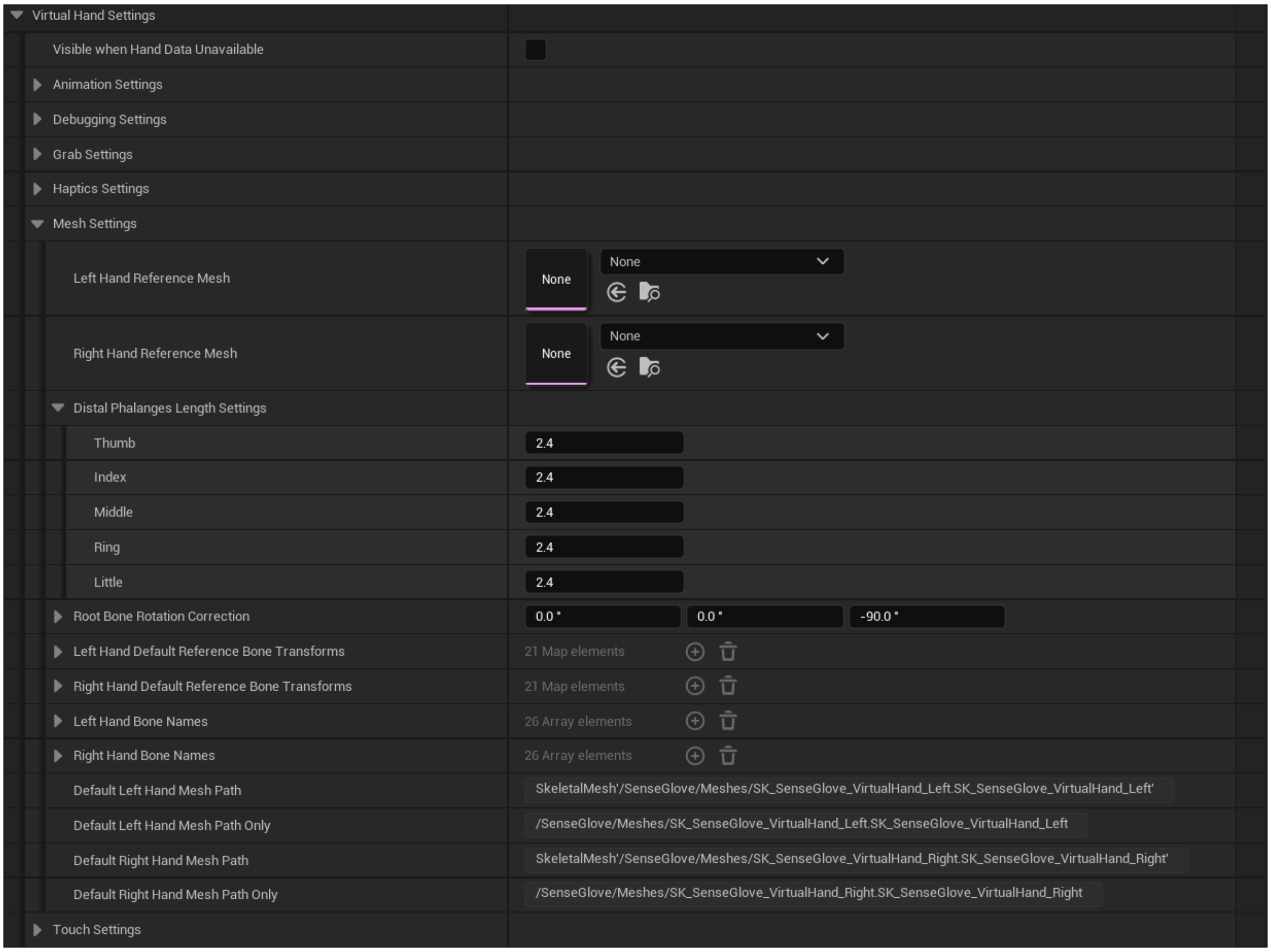
Virtual Hand Settings
- Visible when Hand Data Unavailable ☐
- Animation Settings
- Debugging Settings
- Grab Settings
- Haptics Settings
  - Auto Stop All Haptics on End Play ✓
- Mesh Settings
- Touch Settings

bAutoStopAllHapticsOnEndPlay

Forces all haptics to stop automatically on the `EndPlay` event. This is useful for situations where the simulation has ended, but ongoing haptic feedback might remain active on the glove indefinitely. By default, this setting is enabled.

The Virtual Hand Mesh Settings

Utilized by the SenseGlove `Tracking` module to account for the current virtual hand mesh when generating hand pose data, resulting in more accurate glove or hand data representation and also smoother animations.



Virtual Hand Settings	
Visible when Hand Data Unavailable	☐
Animation Settings	
Debugging Settings	
Grab Settings	
Haptics Settings	
Mesh Settings	
Left Hand Reference Mesh	None / None
Right Hand Reference Mesh	None / None
Distal Phalanges Length Settings	
Thumb	2.4
Index	2.4
Middle	2.4
Ring	2.4
Little	2.4
Root Bone Rotation Correction	0.0 ° 0.0 ° -90.0 °
Left Hand Default Reference Bone Transforms	21 Map elements
Right Hand Default Reference Bone Transforms	21 Map elements
Left Hand Bone Names	26 Array elements
Right Hand Bone Names	26 Array elements
Default Left Hand Mesh Path	SkeletalMesh'/SenseGlove/Meshes/SK_SenseGlove_VirtualHand_Left.SK_SenseGlove_VirtualHand_Left'
Default Left Hand Mesh Path Only	/SenseGlove/Meshes/SK_SenseGlove_VirtualHand_Left.SK_SenseGlove_VirtualHand_Left
Default Right Hand Mesh Path	SkeletalMesh'/SenseGlove/Meshes/SK_SenseGlove_VirtualHand_Right.SK_SenseGlove_VirtualHand_Right'
Default Right Hand Mesh Path Only	/SenseGlove/Meshes/SK_SenseGlove_VirtualHand_Right.SK_SenseGlove_VirtualHand_Right
Touch Settings	

LeftHandReferenceMesh

The virtual hand model for the left hand is to be used by the SenseGlove `Tracking` module to generate all the `26` joint data present in the `FXRHandTrackingState`. The main reason the `Tracking` module requires a virtual hand mesh as a reference is the SenseGlove Hand Pose format only provides `15` joints. So, the remaining joint data for `FXRHandTrackingState` are calculated from a virtual hand mesh compatible with the Epic rig and also the values specified by `DistalPhalangesLengthSettings`. Furthermore, when calculating the existing joints data, their current locations and rotations are taken into account in calculating the resulting `FXRHandTrackingState`.

By default, no virtual hand mesh is set.

> **Caution**
>
> If no virtual hand mesh is set, the `Tracking` module will fall back to hard-coded values extracted from the standard virtual hand model shipped by Unreal Engine VRTemplate. This may result in distorted hand mesh while animating a hand in case a different hand mesh other than the default Epic virtual hand mesh is being set on the virtual hand components.

RightHandReferenceMesh

The virtual hand model for the right hand is to be used by the SenseGlove `Tracking` module to generate all the `26` joint data present in the `FXRHandTrackingState`. The main reason the `Tracking` module requires a virtual hand mesh as a reference is the SenseGlove Hand Pose format only provides `15` joints. So, the remaining joint data for `FXRHandTrackingState` are calculated from a virtual hand mesh compatible with the Epic rig and also the values specified by `DistalPhalangesLengthSettings`. Furthermore, when calculating the existing joints data, their current locations and rotations are taken into account in calculating the resulting `FXRHandTrackingState`.

By default, no virtual hand mesh is set.

> **Caution**
>
> If no virtual hand mesh is set, the `Tracking` module will fall back to hard-coded values extracted from the standard virtual hand model shipped by Unreal Engine VRTemplate. This may result in distorted hand mesh while animating a hand in case a different hand mesh other than the default Epic virtual hand mesh is being set on the virtual hand components.

DistalPhalangesLengthSettings

The length of distal phalanges that cannot be retrieved from any virtual hand mesh compliant with the Epic standard rig. Also, the SenseGlove Hand Pose format does not provide these. This is used by SenseGlove `Tracking` module to calculate an `FXRHandTrackingState` the all `26` joints. The values you specify here depend on the shape of the virtual hand mesh and the defaults are approximated for the virtual hand model shipped with the Unreal Engine VRTemplate.

RootBoneRotationCorrection

Used mostly by the SenseGlove `Tracking` module and `SGPawn` to offset for any initial rotation during the virtual hand mesh import process. This is the case for example with the virtual hand model shipped with Unreal Engine's VRTemplate, which typically has an initial `-90.0f` degrees rotation on the `Yaw` axis. By default, this option has been set up with the Unreal Engine's VRTemplate virtual hand model in mind.

LeftHandDefaultReferenceBoneTransforms

Read-only and for internal use only.

RightHandDefaultReferenceBoneTransforms

Read-only and for internal use only.

LeftHandBoneNames

Read-only and for internal use only.

RightHandBoneNames

Read-only and for internal use only.

DefaultLeftHandMeshPath

Read-only and for internal use only.

DefaultLeftHandMeshPathOnly

Read-only and for internal use only.

DefaultRightHandMeshPath

Read-only and for internal use only.

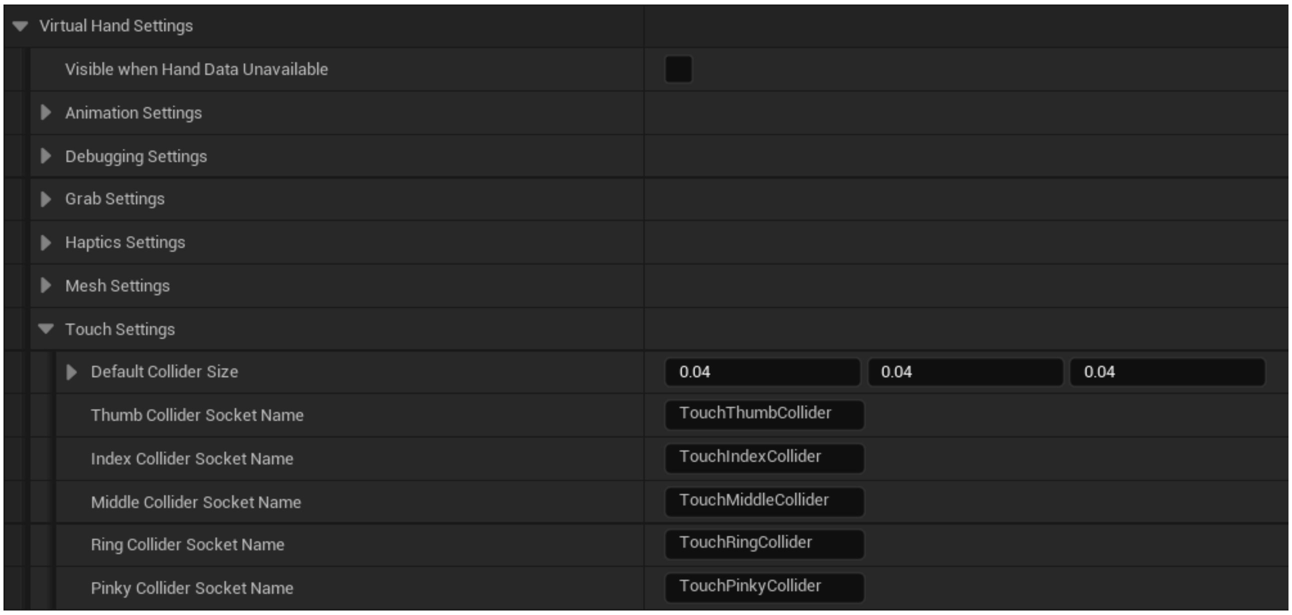
DefaultRightHandMeshPathOnly

Read-only and for internal use only.

The Virtual Hand Touch Settings

Utilized by the SenseGlove Sockets Editor to automatically generate the hand sockets required by the Touch system to function.

The `SGPawn` also utilizes these settings to set up the touch colliders on the virtual hand components.



Virtual Hand Settings	
Visible when Hand Data Unavailable	☐
Animation Settings	
Debugging Settings	
Grab Settings	
Haptics Settings	
Mesh Settings	
Touch Settings	
Default Collider Size	0.04 0.04 0.04
Thumb Collider Socket Name	TouchThumbCollider
Index Collider Socket Name	TouchIndexCollider
Middle Collider Socket Name	TouchMiddleCollider
Ring Collider Socket Name	TouchRingCollider
Pinky Collider Socket Name	TouchPinkyCollider

DefaultColliderSize

The default collider size for the fingers' touch colliders.

ThumbColliderSocketName

The default socket name for the thumb finger's touch collider, usually located at the tip of the thumb finger.

IndexColliderSocketName

The default socket name for the index finger's touch collider, usually located at the tip of the index finger.

MiddleColliderSocketName

The default socket name for the middle finger's touch collider, usually located at the tip of the middle finger.

RingColliderSocketName

The default socket name for the ring finger's touch collider, usually located at the tip of the ring finger.

PinkyColliderSocketName

The default socket name for the pinky finger's touch collider, usually located at the tip of the pinky finger.

Overriding The Plugin Settings

The override system allows you to customize and override the global settings for the SenseGlove Unreal Engine Plugin through specific subcomponents where applicable. This feature enables more precise control over the behavior of individual components within your project.

The SenseGlove Virtual Hand Component

The Virtual Hand Component provides the ability to override certain aspects of the global plugin settings, allowing for tailored interactions and behaviors specific to virtual hands. For more details, refer to the Virtual Hand Settings section.

The SenseGlove Wrist Tracker Component

The Wrist Tracker Component enables overriding of specific global plugin settings, providing flexibility in wrist tracking configurations. For additional information, see the Wrist-tracker Settings section.

The SenseGlove Console Commands

The SenseGlove Unreal Engine Plugin offers a variety of utility console commands to enhance your development experience.

> **Important**
>
> To ensure the SenseGlove console commands are registered and recognized by Unreal Engine, set the default Game Instance class to `SGGameInstance` or a subclass of it. This can be done through: `Project Settings > Project > Maps & Modes > Game Instance > Game Instance Class`. Failing to do so will result in the error: `Command not recognized: SG_*` in the logs. For more details, refer to `SGGameInstance`.

SGGameUserSettings Console Commands

> **Caution**
>
> Before running any of the following console commands, ensure that the default Game User Settings class is set to `SGGameUserSettings` or a subclass of it. This can be configured via: `Project Settings > Engine > General Settings > Default Classes > Advanced > Game User Settings Class`. Failure to set this correctly will cause your simulation or editor to crash upon calling any of the following console commands. For more information, refer to `SGGameUserSettings`.

SG_GetEngineScalabilitySettings

This console command prints the current Engine Scalability Settings to the logs.

SG_SetEngineScalabilitySettings

This console command sets the Engine Scalability Settings for both the current game and the editor. It accepts a `Scalability` parameter with the following valid values:

- `Low`
- `Medium`
- `High`
- `Epic`
- `Cinematic`
- `Auto`

> **Note**
>
> The `Auto` option is used for benchmarking purposes. It will adjust the engine scalability settings to one of the other levels based on the benchmarking results.

Deploying to Android (Standalone)

Epic Games provides official documentation for setting up Unreal projects targeting Android:

- Setting Up Android SDK and NDK for Unreal
- Android Quick Start

Here are a few important notes to consider:

- Since SenseGlove provides native libraries built for Android, it's crucial to consult the Platform Support Matrix before deciding to deploy your project to Android.
- Currently, all third-party native libraries are built against Android NDK API Level `29` .
- On Meta Quest devices, building against Android SDK API Level `29` or `32` has been tested and is supported.
- A video tutorial on deploying to Oculus Quest devices and Android is also available.

Caution

> As of the `v2.1.0` release of the SenseGlove Unreal Engine Plugin, the `XR_EXT_hand_tracking` OpenXR extension is required for the plugin to function. Without this OpenXR extension, the plugin won't output any glove data. Unreal Engine provides `XR_EXT_hand_tracking` support through the `OpenXR` and `OpenXRHandTracking` plugins. However, for this to function on Android in standalone mode in conjunction with other plugins such as `Meta XR` or `ViveOpenXR` plugins, or hand-tracking as a fallback mechanism when no glove data is available, extra configuration steps and considerations are required.

Third-Party Tutorials

This Android Standalone Mode Deployment tutorials series covers how to build and deploy Unreal Engine `5.5` Projects APK to Android and Meta Quest 3S/3/Pro/2 in standalone mode. Furthermore, it covers the how-to on enabling OpenXR hand-tracking on Android with Meta XR (Quest 3S/3/Pro/2) and HTC VIVE OpenXR (Focus Vision/XR Elite/Focus 3) plugins.

Third-Party Tutorials: Android Standalone Mode Deployment

On-Click Unreal Engine 5.7 Android Packaging & APK Build Tutorial | Meta Quest & HTC VIVE Standalone

This tutorial provides a complete, streamlined guide to packaging and deploying Unreal Engine 5.7 projects to Android devices in standalone mode, with full support for Meta Quest and HTC VIVE headsets. Viewers learn how to correctly install and configure every required component, including Android Studio, SDK, NDK, JDK, and Visual Studio, using either a manual setup, or an open-source one-click PowerShell installer featured in the video.

The tutorial walks through creating a fresh Unreal Engine VR Template project, configuring Android project settings, enabling ADB device debugging, and generating a working APK using the updated Project Launcher workflow in UE 5.7. It also covers how to integrate the Meta XR and HTC VIVE OpenXR plugins to ensure proper VR recognition and hand-tracking functionality on each platform.

By the end, developers understand how to go from a blank project to a fully packaged and deployable Android APK, with functioning VR, correct device targeting, and reliable deployment pipelines on both Meta Quest and HTC VIVE standalone headsets.

Build & Deploy Unreal Engine 5.5 Projects APK to Android & Meta Quest 3S/3/Pro/2 in Standalone Mode

This video will guide you through the process of building and deploying an Unreal Engine 5.5 project (or any version newer than `4.24`) to Android Standalone Mode for Meta Quest 2 and Quest 3 devices. The steps outlined here should also apply to other Android-based VR headsets.

It will show you where and how to download and install the necessary prerequisites, such as the Android SDK, NDK, Java Development Kit (JDK), and Microsoft Visual Studio. Next, it will configure both the development environment and Unreal Engine for a successful build. Finally, it will walk you through deploying your project to your VR headset and troubleshooting common errors to ensure a smooth experience.

Unreal Engine OpenXR Hand-Tracking on Android with Meta XR (Quest 3S/3/Pro/2) and HTC VIVE OpenXR (Focus Vision/XR Elite/Focus 3) Plugins

The OpenXR hand-tracking provided by Epic won't work out of the box on Android when projects are deployed in standalone mode to HMDs such as Meta Quest or HTC VIVE devices.

In this tutorial you'll learn how to deploy your Unreal Engine projects to Android with functional hand-tracking on the following head-mounted displays:

- Meta Quest 3S
- Meta Quest 3
- Meta Quest Pro
- Meta Quest 2
- HTC VIVE Focus Vision
- HTC VIVE XR Elite
- HTC VIVE Focus 3

Upgrade Guide

Upgrading from v2.7.x to v2.8.x

Summary of Breaking Changes

Starting with **SenseGlove Unreal Engine Plugin v2.8.x**, support for `FXRMotionControllerData` has been **removed**.

Although:

- Unreal Engine **5.5** and **5.6** still support the deprecated `FXRMotionControllerData`, and
- These engine versions are still supported by plugin v2.8.x,

the plugin now fully transitions to **`FXRHandTrackingState`**, which is the modern and recommended API introduced in **Unreal Engine 5.5+** for OpenXR hand tracking.

Additionally:

- Plugin v2.8.x no longer supports **Unreal Engine 5.4**.
- Because `FXRHandTrackingState` is the forward-compatible OpenXR API, continued support for `FXRMotionControllerData` is no longer maintained in the plugin.

We have supported `FXRHandTrackingState` since `v2.2.0` (released 2024-10-22), and it has been stable in production for several releases.

Who Is Affected?

This change only affects you if:

- You maintain a **custom hand manipulation or tracking system**, and
- Your implementation directly consumes `FXRMotionControllerData` provided by the SenseGlove plugin.

If you rely solely on the plugin's provided components and standard integration workflow, no action is required.

Required Migration

If you are directly consuming `FXRMotionControllerData` , for your project to build or function as expected, you must migrate to:

```
FXRHandTrackingState
```

The migration is straightforward because both structures represent similar hand tracking data concepts within Unreal's OpenXR framework.

A step-by-step explanation of how to work with `FXRHandTrackingState` is available in this third-party tutorial:

👉 Introduction to Virtual Reality, OpenXR Hand Tracking, and Gesture Detection in Unreal Engine

Why This Change?

- `FXRMotionControllerData` is deprecated.
- `FXRHandTrackingState` is the future-proof OpenXR hand tracking API.
- The existing Unreal Engine's OpenXR implementation is centered around `FXRHandTrackingState` and `FXRMotionControllerData` has been completely removed form UE `5.7` +.
- Removing legacy support simplifies the plugin architecture and aligns it with Unreal's forward direction.

If you are already using `FXRHandTrackingState` or not directly consuming the OpenXR data, no changes are required when upgrading to v2.8.x.

Upgrading from v2.0.x to v2.1.x

The transition from `v2.0.x` to `v2.1.x` introduces numerous changes, including several breaking changes. The effort required to upgrade your project will vary depending on its complexity and which features of the SenseGlove Unreal Engine Project you are using. However, if you are working with a simple Blueprint project like SGBasicDemo, the upgrade process is quite straightforward. We successfully upgraded SGBasicDemo to SGBasicDemo-OpenXR by following the procedure outlined below.

These are the notable changes that might affect your project:

- The SenseGlove Virtual Hand and Wrist Tracker components no longer rely on the SenseGlove Hand Pose data from the underlying SenseGlove API. Instead, they use `FXRMotionControllerData`.
- The virtual hand animation system has been revamped to use `FXRMotionControllerData` and no longer relies on SenseGlove Hand Angles. This means the virtual hand meshes are animated using world space transforms instead of parent bone space transforms.
- The Allbreaker virtual hand meshes have been removed and are no longer supported as they are incompatible with the new OpenXR tracking and animation system.

Caution

Please consult the changelog before upgrading your project to see if any change affects or breaks your current project.

Note

For upgrading older versions of the plugin to `v2.0.0`, a YouTube tutorial is available.

1. Remove the existing `Plugins/SenseGlove` folder from your project.

2. Obtain the latest `v2.1.x` version of the plugin either from the Epic Games Launcher or Microsoft Azure DevOps Repositories and place it in the `Plugins/SenseGlove` folder that you've just removed.

3. It might be best to clean up and remove the following folders from your project before generating the project files or attempting to open your project with the Unreal Editor. This might prevent a certain class of build issues:

```
- Binaries
- Intermediate
- Saved
```

4. Build your project using your favorite IDE if it's a C++ project, or open your project's `.uproject` file with the Unreal Editor and wait for the Editor to build the necessary binaries and open the project.

5. Remove the Allbreaker virtual hand meshes if you are using them, as they are no longer compatible with the new animation system.

6. Import and set up a set of compatible virtual hand meshes such as the VRTemplate virtual hand meshes, and configure the materials, rigid bodies, and

the SenseGlove Grab and Touch Sockets using the SenseGlove Sockets Editor.

7. Set up the SGPawn to use the new virtual hand meshes for the `HandLeft`, `HandRight`, `RealHandLeft`, and `RealHandRight` components.

8. Adjust the Virtual Hand Mesh Settings and ensure the `Left Hand Reference Mesh` and `Right Hand Reference Mesh` are set correctly.

9. Check and adjust the Virtual Hand Animation Settings as needed.

10. You might also want to set up the Wrist Tracking Hardware to use the new experimental HMD auto-detection feature. This allows the plugin to automatically configure the wrist tracking hardware at runtime, rather than limiting your builds to a specific HMD.

11. Set up the `SGGameInstance` and `SGGameUserSettings` if you want to use the new SenseGlove console commands or take advantage of the Engine Scalability Settings to achieve higher framerates in your project.

12. Additionally, the latest release introduces the ability to use hand-tracking as an alternative to SenseGlove hardware—albeit without haptic feedback—for rapid development and testing. It's also recommended to enable the Fallback to HandTracking if No Glove Detected feature to seamlessly switch to hand-tracking when a glove isn't connected.

13. If all steps have been followed correctly, your project should now be fully compatible with the new plugin release.

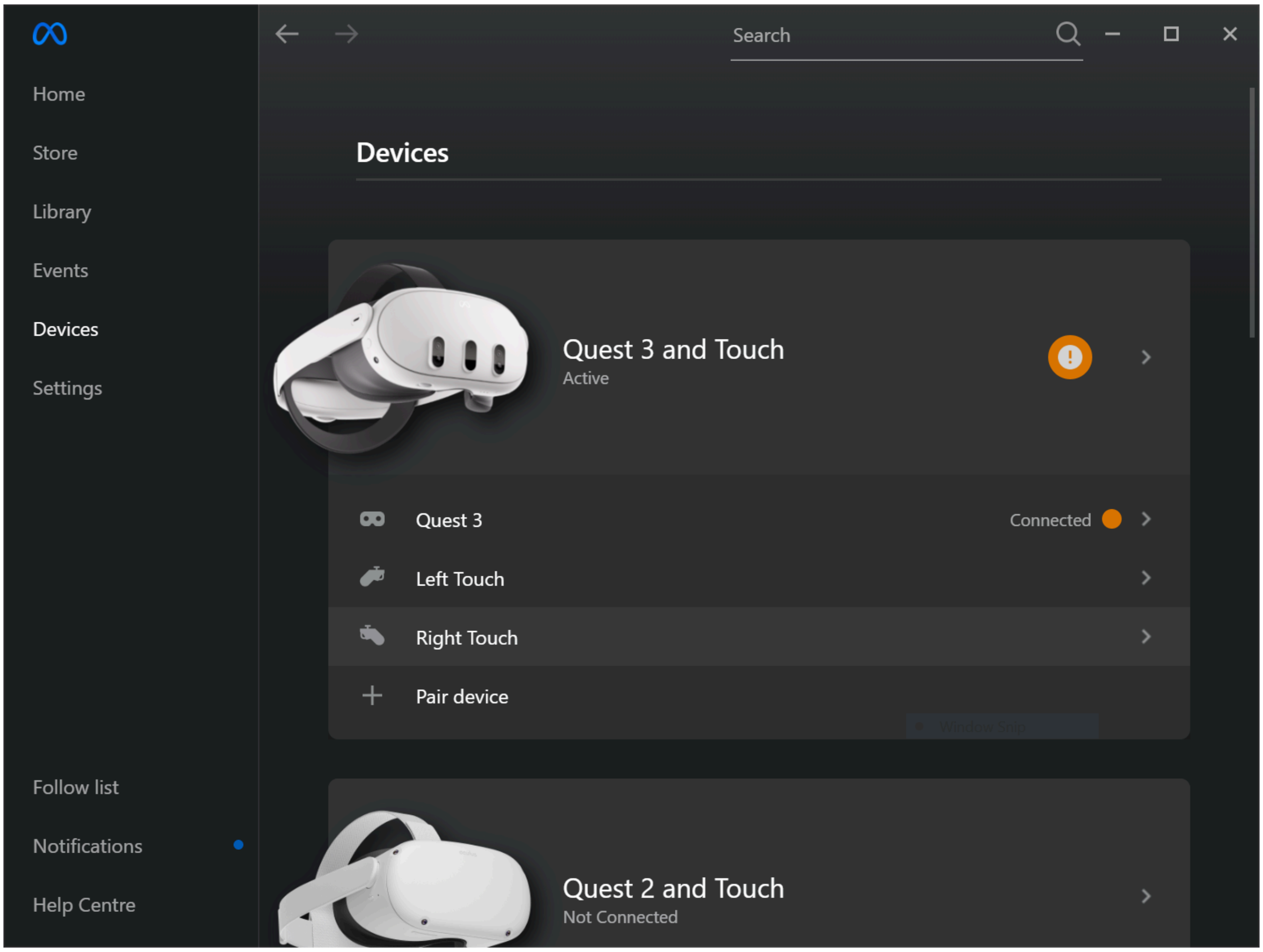
Optimizing Your Project for Higher FPS

Enhancing the performance and framerate of Unreal Engine VR applications, whether running standalone or streaming from a PC, can sometimes be challenging depending on the nature of your project. This guide will walk you through generic strategies that can significantly boost your project's performance and framerate with minimal effort.

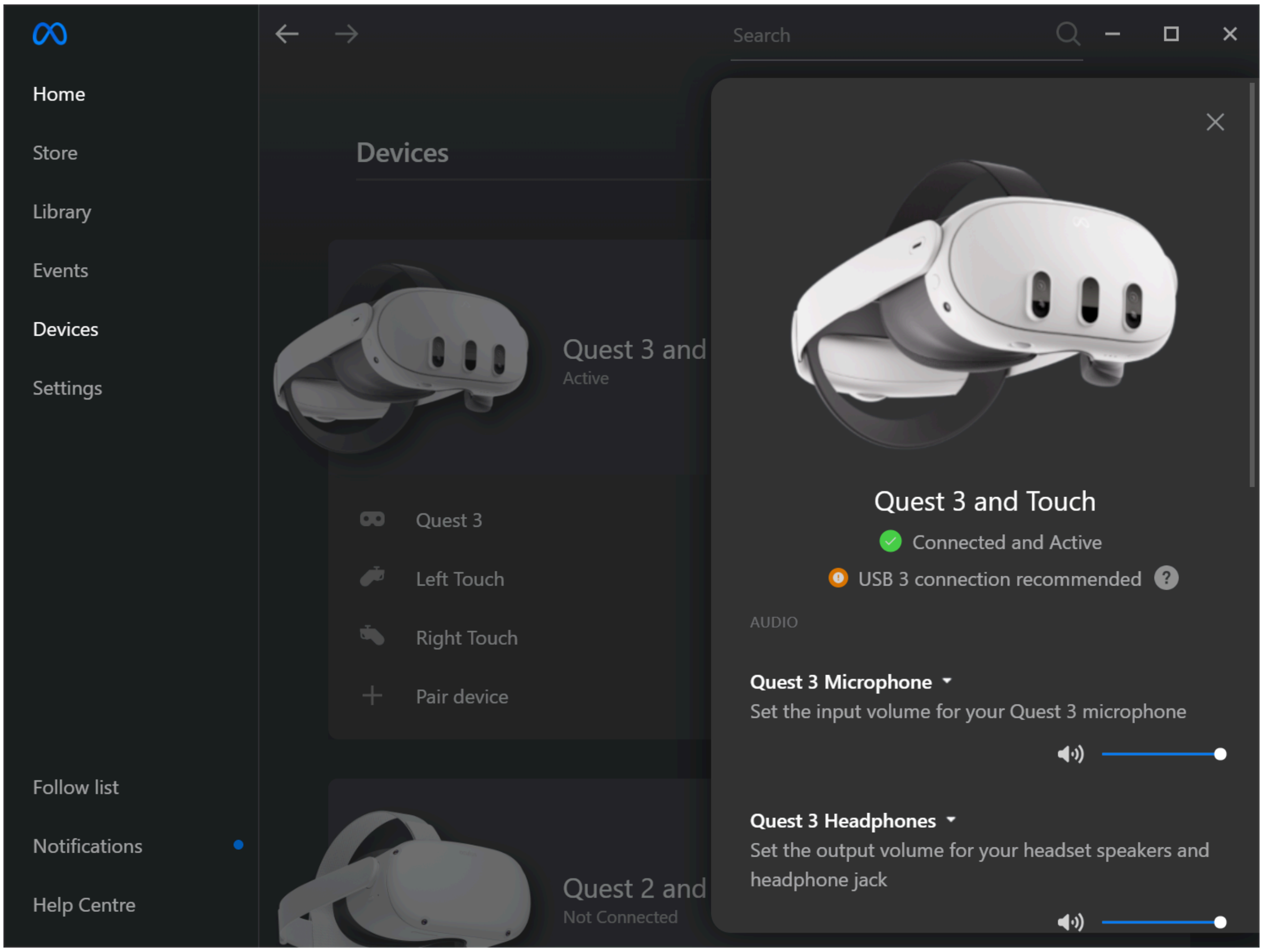
Meta Quest Link Advanced Graphics Preferences

When streaming from a PC to Meta Quest devices, the default refresh rate is set at `72 Hz`. However, you can increase this to `120 Hz`, which not only enhances the refresh rate but also reduces the rendering resolution, potentially improving performance. Follow these steps to make the adjustment:

1. Open the Meta Quest Link app and navigate to the `Devices` tab.



2. Choose the device for which you would like to tweak the refresh rate.



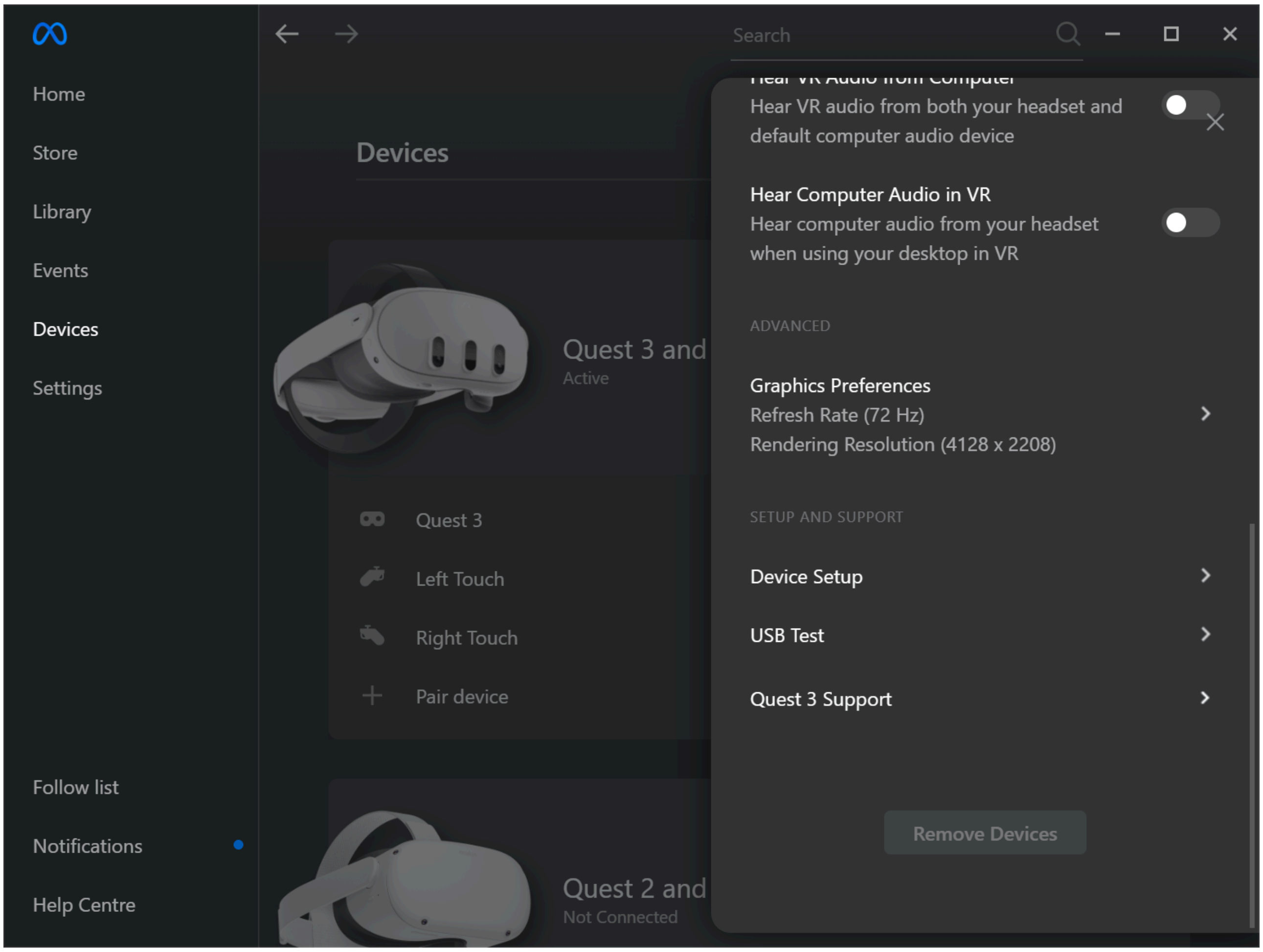
Quest 3 and Touch
Connected and Active
USB 3 connection recommended

AUDIO

Quest 3 Microphone
Set the input volume for your Quest 3 microphone

Quest 3 Headphones
Set the output volume for your headset speakers and headphone jack

3. In the device settings, scroll down to the `Advanced` section and select `Graphics Preferences`.



Hear VR audio from both your headset and default computer audio device

Hear Computer Audio in VR
Hear computer audio from your headset when using your desktop in VR

ADVANCED

Graphics Preferences
Refresh Rate (72 Hz)
Rendering Resolution (4128 x 2208)

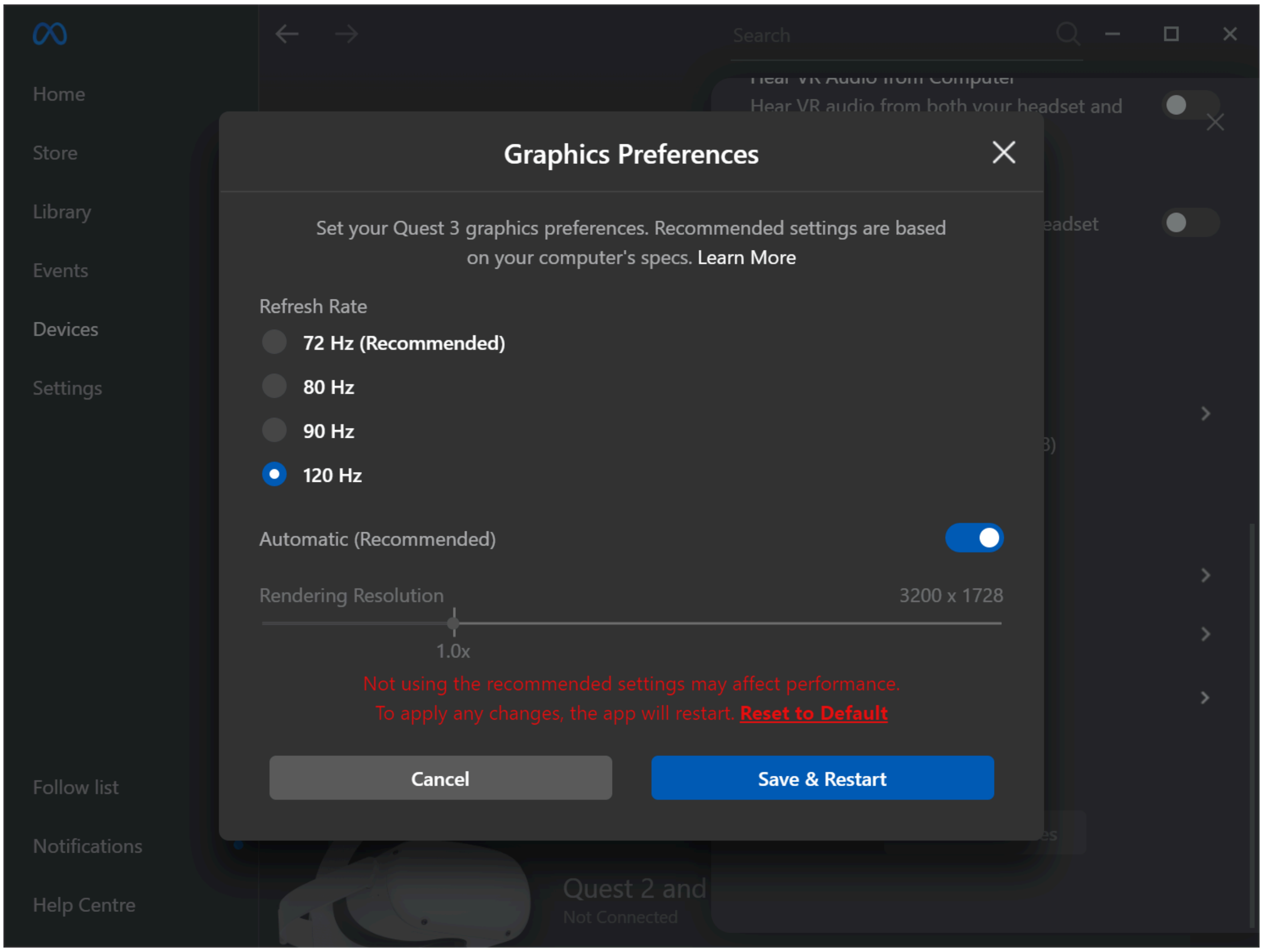
SETUP AND SUPPORT

Device Setup

USB Test

Quest 3 Support

4. Choose your desired refresh rate. In this case select a refresh rate of `120 Hz`. After making your selection, click `OK`, and the Meta Quest Link app will restart to apply the changes.



Graphics Preferences

Set your Quest 3 graphics preferences. Recommended settings are based on your computer's specs. Learn More

Refresh Rate

72 Hz (Recommended)
80 Hz
90 Hz
[✓ 120 Hz]

Automatic (Recommended) [✓ On]

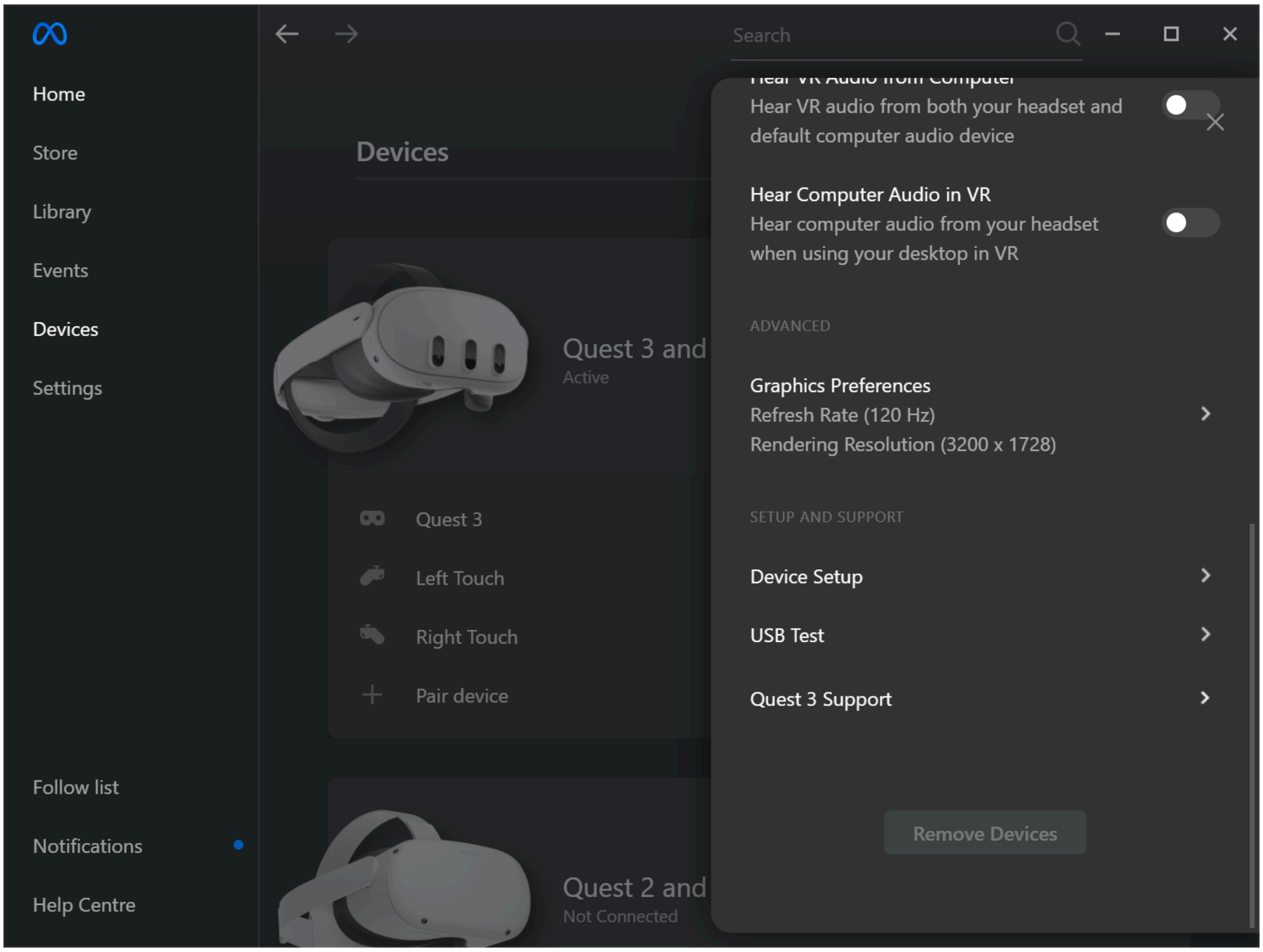
Rendering Resolution 3200 x 1728

1.0x

Not using the recommended settings may affect performance.
To apply any changes, the app will restart. Reset to Default

Cancel | Save & Restart

5. Once the Meta Quest Link app restarts, go back to the `Devices` tab, select your device, and confirm the refresh rate setting under `Advanced > Graphics Preferences`.



Hear VR Audio from Computer
Hear VR audio from both your headset and default computer audio device

Hear Computer Audio in VR
Hear computer audio from your headset when using your desktop in VR

ADVANCED

Graphics Preferences
Refresh Rate (120 Hz)
Rendering Resolution (3200 x 1728)

SETUP AND SUPPORT

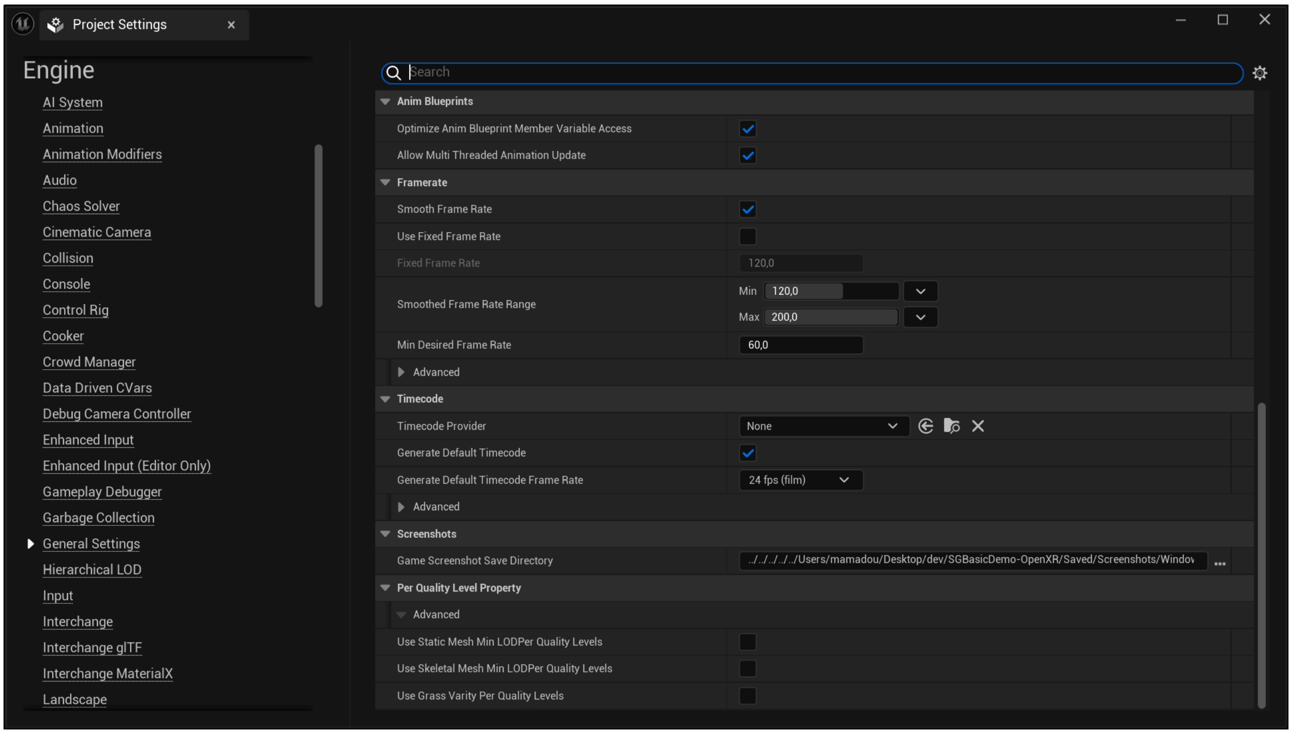
Device Setup

USB Test

Quest 3 Support

Remove Devices

6. Now, open your Unreal Engine project and navigate to `Project Settings`. Under `Engine > General Settings > Framerate`, you can fine-tune and experiment with the framerate settings to match your project's requirements.



Anim Blueprints
- Optimize Anim Blueprint Member Variable Access: ✓
- Allow Multi Threaded Animation Update: ✓

Framerate
- Smooth Frame Rate: ✓
- Use Fixed Frame Rate: ☐
- Fixed Frame Rate: 120,0
- Smoothed Frame Rate Range: Min 120,0 / Max 200,0
- Min Desired Frame Rate: 60,0
- Advanced

Timecode
- Timecode Provider: None
- Generate Default Timecode: ✓
- Generate Default Timecode Frame Rate: 24 fps (film)
- Advanced

Screenshots
- Game Screenshot Save Directory: ../../../../Users/mamadou/Desktop/dev/SGBasicDemo-OpenXR/Saved/Screenshots/Windov

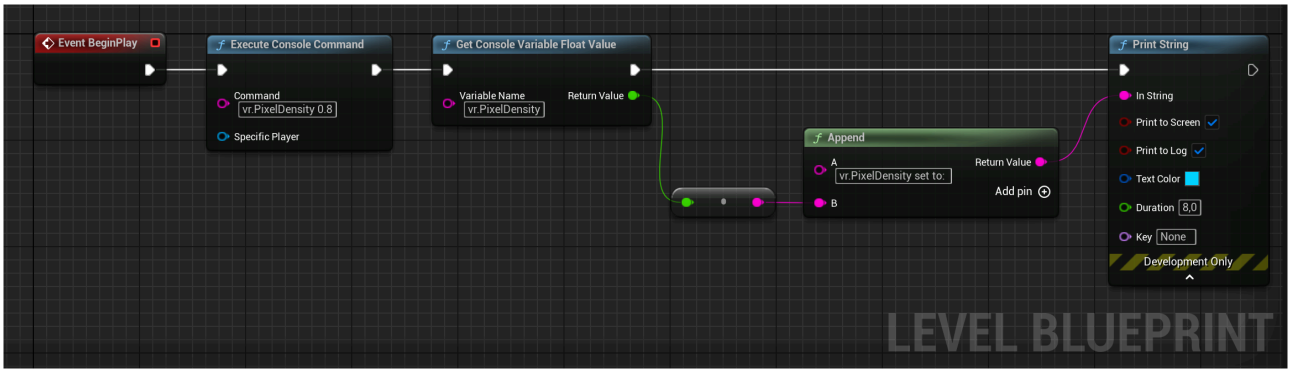
Per Quality Level Property
- Advanced
- Use Static Mesh Min LODPer Quality Levels: ☐
- Use Skeletal Mesh Min LODPer Quality Levels: ☐
- Use Grass Varity Per Quality Levels: ☐

HTC VIVE Specific Optimizations in Standalone Mode

Headsets like the HTC VIVE Focus 3 and Focus Vision have larger framebuffers, higher eye-buffer size, wider fields of view (FOV), and varying refresh rates compared to HMDs like the Meta Quest series. As a result, performance can differ when running the same content across these devices.

Setting a Custom Pixel Density

If performance on the Focus 3 or Focus Vision is lower than on the Quest, we recommend adjusting the **Pixel Density** to approximately `0.8` (or a value suited to your content). This reduces the eye buffer size, and matches the resolution of Quest 2, hence increasing FPS:



Command vr.PixelDensity 0.8

Variable Name vr.PixelDensity

A vr.PixelDensity set to:

B

Print to Screen ✓

Print to Log ✓

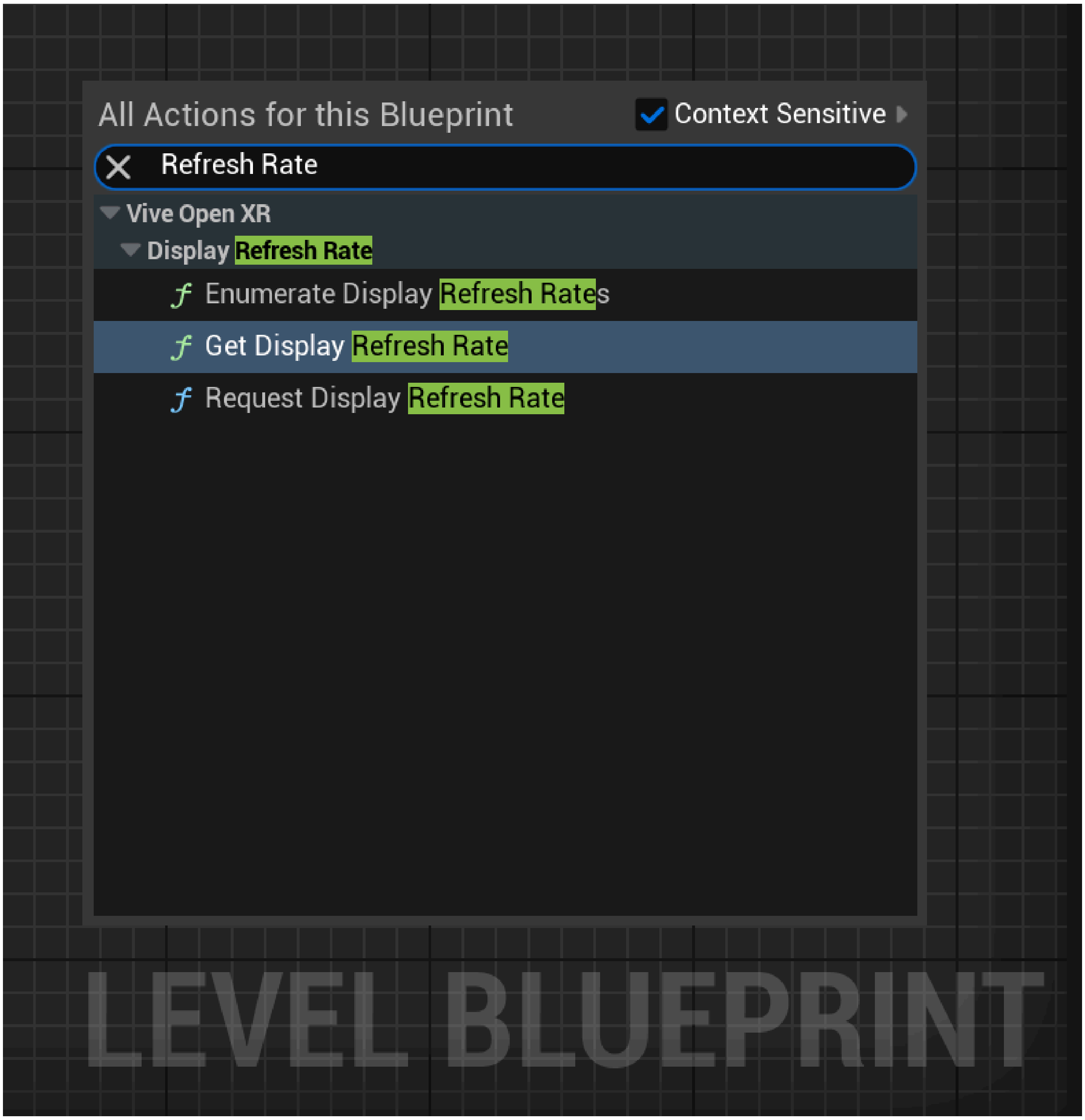
Text Color 🟦

Duration 8,0

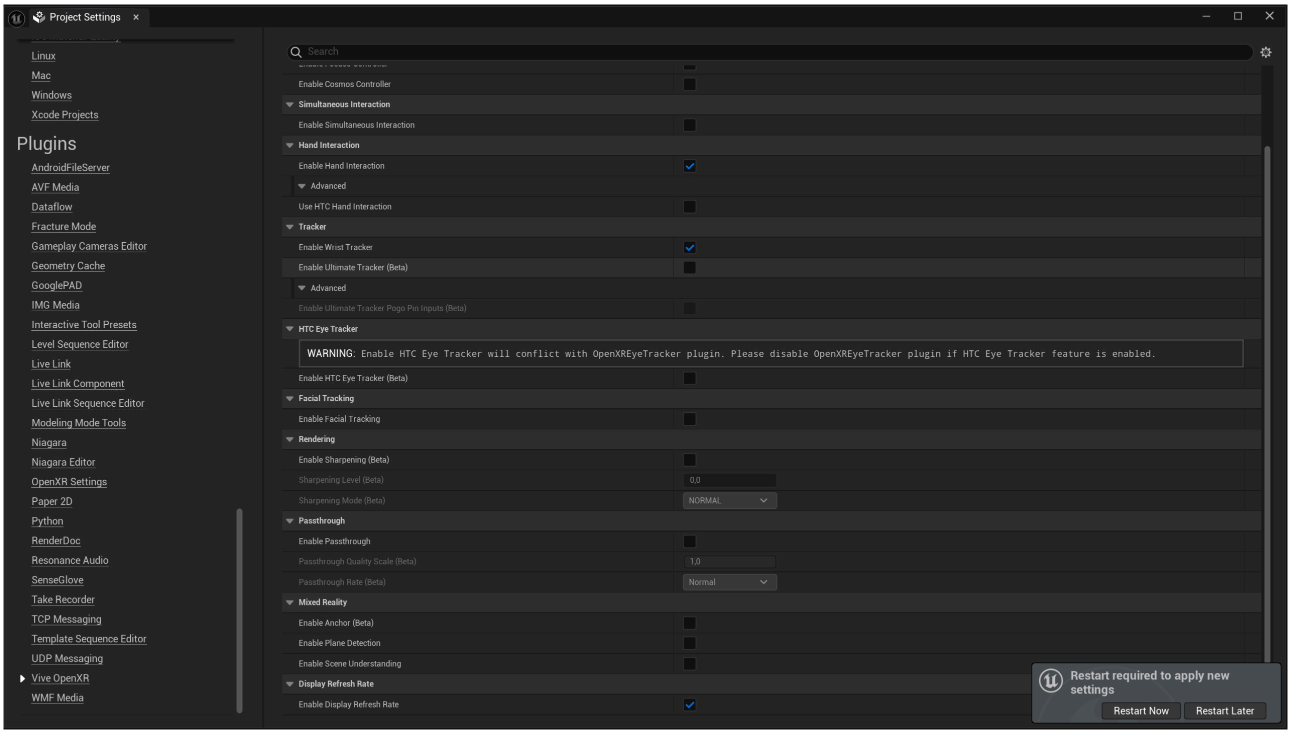
Key None

Setting a Lower Refresh Rate

The `ViveOpenXR` plugin provides various Blueprint functions to query the supported refresh rates and adjust the current refresh rate according to your needs:



But, before you'd be able to make these adjustments you have to enable the relevant settings by navigating to `Edit > Project Settings > Plugins > Vive OpenXR` and ensure the `Enable Display Refresh Rate` option is checked. This option enables the OpenXR extension `XR_FB_display_refresh_rate` support which allows your application to dynamically adjust the display refresh rate in order to improve the overall user experience. Please note that You need to restart the engine to apply new settings after changing this setting.



Linux
Mac
Windows
Xcode Projects

Plugins
AndroidFileServer
AVF Media
Dataflow
Fracture Mode
Gameplay Cameras Editor
Geometry Cache
GooglePAD
IMG Media
Interactive Tool Presets
Level Sequence Editor
Live Link
Live Link Component
Live Link Sequence Editor
Modeling Mode Tools
Niagara
Niagara Editor
OpenXR Settings
Paper 2D
Python
RenderDoc
Resonance Audio
SenseGlove
Take Recorder
TCP Messaging
Template Sequence Editor
UDP Messaging
Vive OpenXR
WMF Media

Enable Cosmos Controller ☐

Simultaneous Interaction
Enable Simultaneous Interaction ☐

Hand Interaction
Enable Hand Interaction ✓
Advanced
Use HTC Hand Interaction ☐

Tracker
Enable Wrist Tracker ✓
Enable Ultimate Tracker (Beta) ☐
Advanced
Enable Ultimate Tracker Pogo Pin Inputs (Beta) ☐

HTC Eye Tracker
```
WARNING: Enable HTC Eye Tracker will conflict with OpenXREyeTracker plugin. Please disable OpenXREyeTracker plugin if HTC Eye Tracker feature is enabled.
```
Enable HTC Eye Tracker (Beta) ☐

Facial Tracking
Enable Facial Tracking ☐

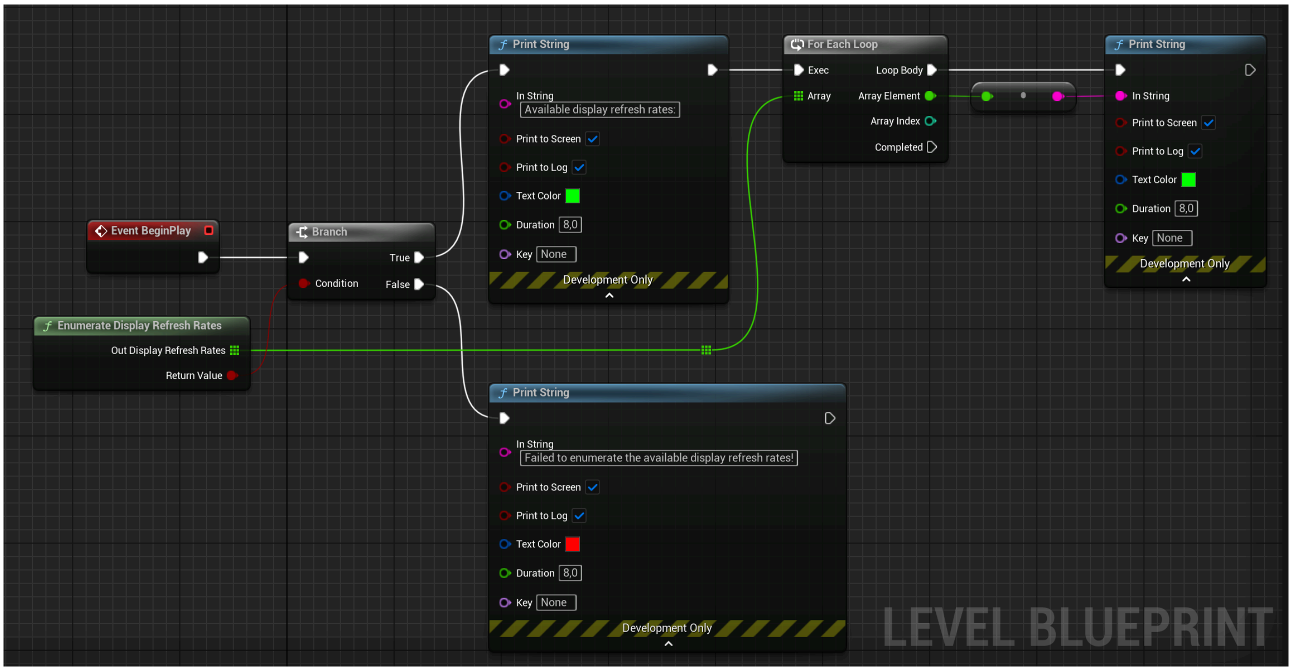
Rendering
Enable Sharpening (Beta) ☐
Sharpening Level (Beta) 0,0
Sharpening Mode (Beta) NORMAL

Passthrough
Enable Passthrough ☐
Passthrough Quality Scale (Beta) 1,0
Passthrough Rate (Beta) Normal

Mixed Reality
Enable Anchor (Beta) ☐
Enable Plane Detection ☐
Enable Scene Understanding ☐

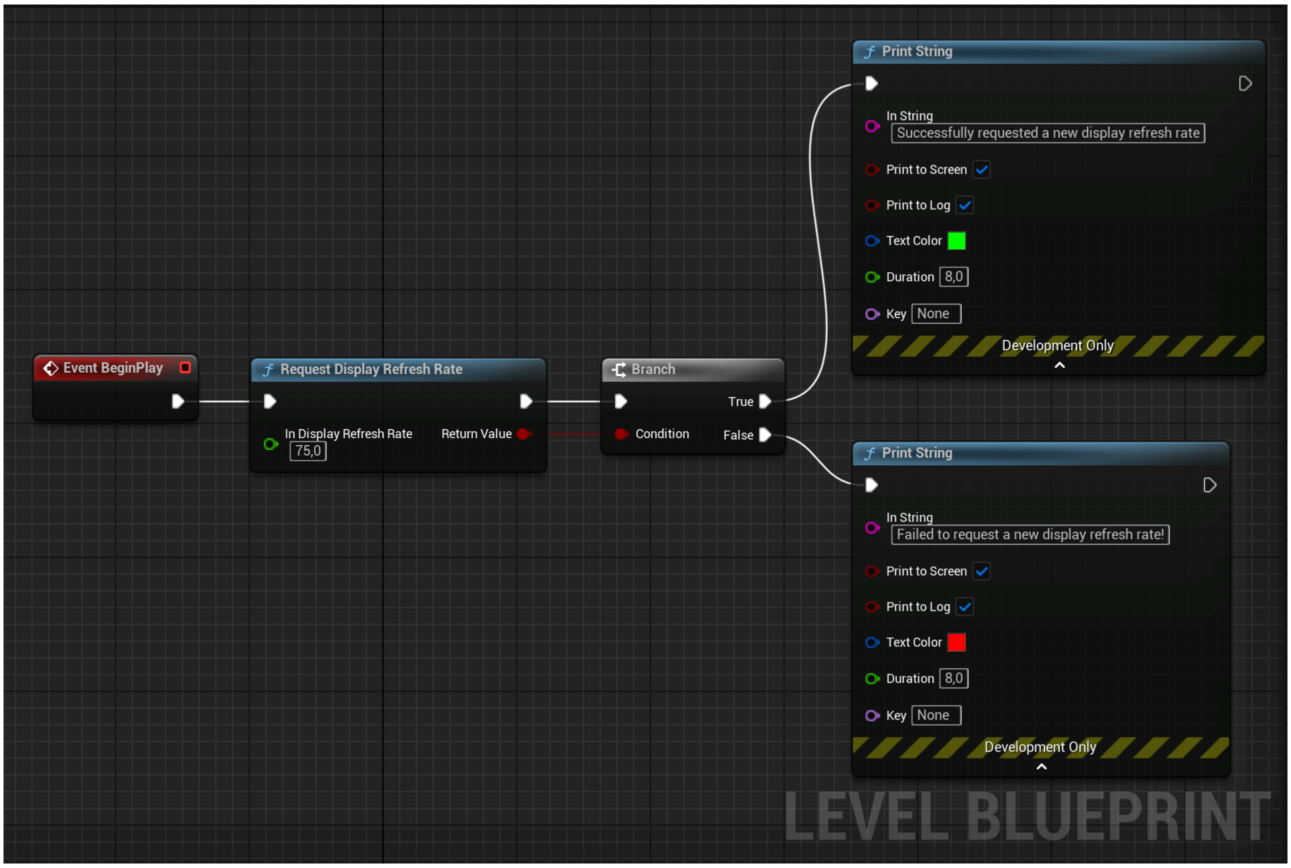
Display Refresh Rate
Enable Display Refresh Rate ✓

In order to query all the available display refresh rates use the Blueprint function `Enumerate Display Refresh Rates`:

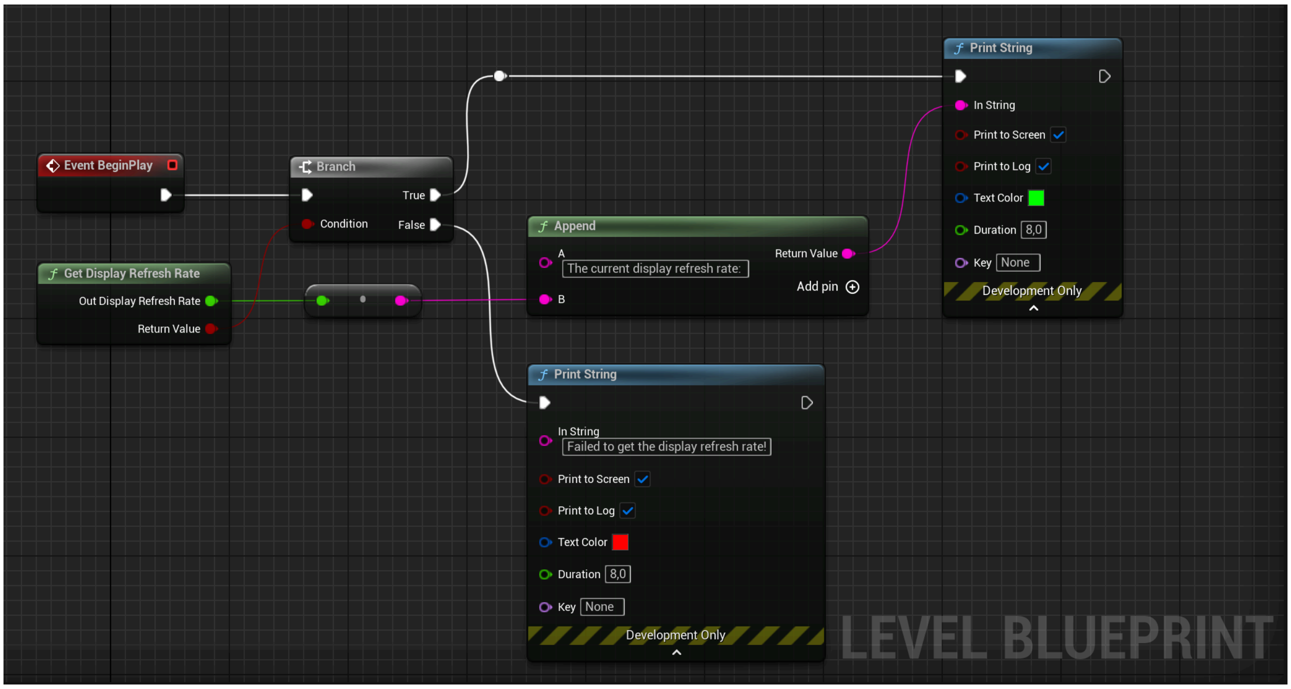


At the moment with most devices it returns `75.0` and `90.0` hz values.

To change the current display refresh rate use the Blueprint function `Request Display Refresh Rate`:



To obtain the current display refresh rate use the Blueprint function `Get Display Refresh Rate`:



If you're looking to squeeze more performance out of your HTC VIVE HMD in standalone mode, consider lowering the target refresh rate from `90hz` to `75hz`. This can help reduce GPU workload and improve overall stability while still maintaining a somewhat smooth experience.

HTC VIVE Optimal Rendering Settings

As a last resort consider adding the following rendering settings to your project's `DefaultEngine.ini` under the `/Script/Engine.RendererSettings` section and experiment with them until you obtain your desired results:

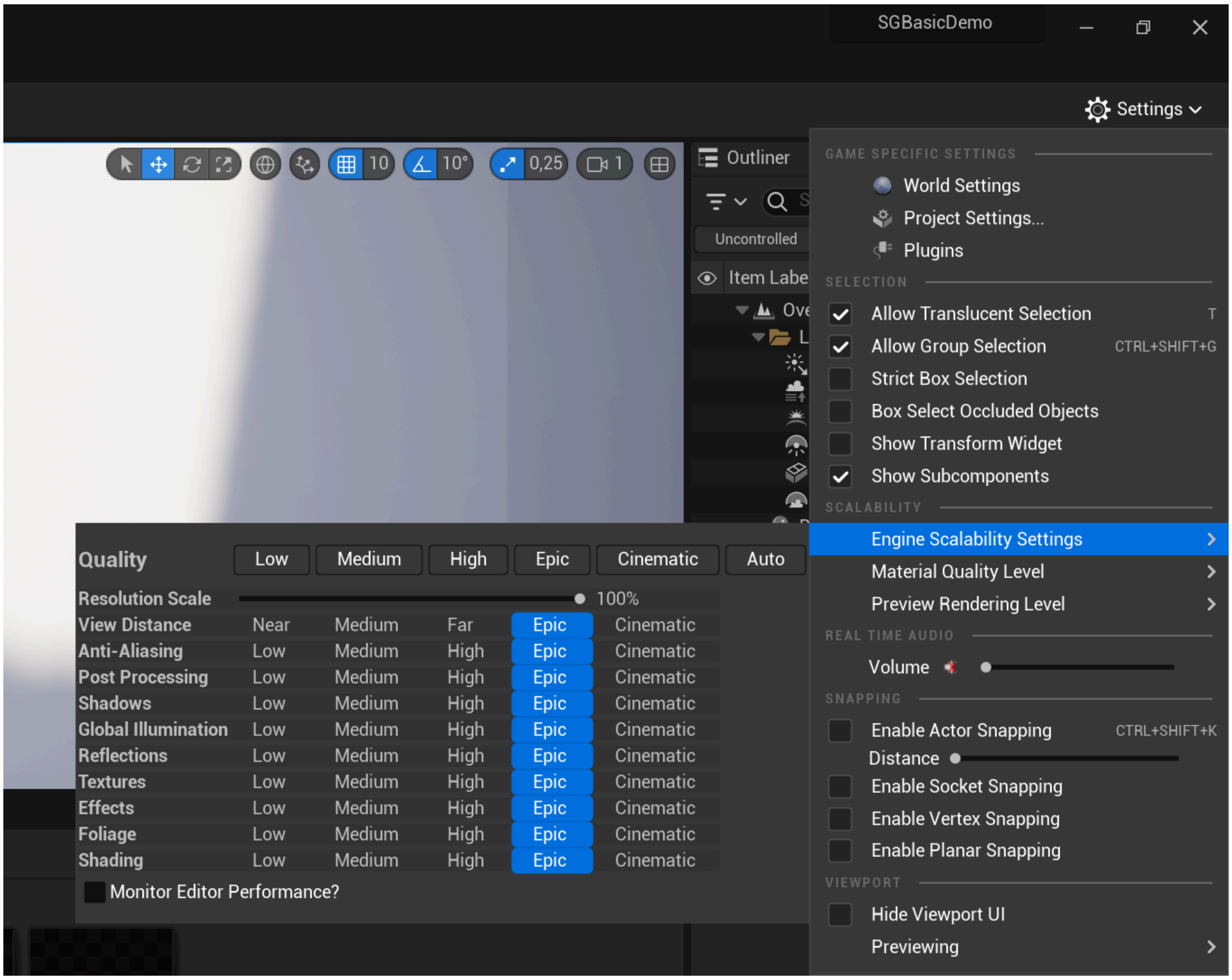
```
[/Script/Engine.RendererSettings]
r.Mobile.DisableVertexFog=True
r.Mobile.AllowDitheredLODTransition=False
r.Mobile.AllowSoftwareOcclusion=False
r.Mobile.VirtualTextures=False
r.DiscardUnusedQuality=False
r.AllowOcclusionQueries=True
r.MinScreenRadiusForLights=0.030000
r.MinScreenRadiusForDepthPrepass=0.030000
r.MinScreenRadiusForCSMDepth=0.010000
r.PrecomputedVisibilityWarning=False
r.TextureStreaming=True
Compat.UseDXT5NormalMaps=False
r.VirtualTextures=False
r.VirtualTexturedLightmaps=False
r.VT.TileSize=128
r.VT.TileBorderSize=4
r.vt.FeedbackFactor=16
r.VT.EnableCompressZlib=True
r.VT.EnableCompressCrunch=False
r.ClearCoatNormal=False
r.ReflectionCaptureResolution=128
r.ReflectionEnvironmentLightmapMixBasedOnRoughness=True
r.ForwardShading=True
r.VertexFoggingForOpaque=True
r.AllowStaticLighting=True
r.NormalMapsForStaticLighting=False
r.GenerateMeshDistanceFields=False
r.DistanceFieldBuild.EightBit=False
r.GenerateLandscapeGIData=False
r.DistanceFieldBuild.Compress=False
r.TessellationAdaptivePixelsPerTriangle=48.000000
r.SeparateTranslucency=True
r.TranslucentSortPolicy=0
TranslucentSortAxis=(X=0.000000,Y=-1.000000,Z=0.000000)
r.CustomDepth=1
r.CustomDepthTemporalAAJitter=True
r.PostProcessing.PropagateAlpha=2
r.DefaultFeature.Bloom=False
r.DefaultFeature.AmbientOcclusion=False
r.DefaultFeature.AmbientOcclusionStaticFraction=True
r.DefaultFeature.AutoExposure=False
r.DefaultFeature.AutoExposure.Method=0
r.DefaultFeature.AutoExposure.Bias=1.000000
r.DefaultFeature.AutoExposure.ExtendDefaultLuminanceRange=True
r.DefaultFeature.AutoExposure.ExtendDefaultLuminanceRange=True
r.EyeAdaptation.EditorOnly=False
```

```
r.DefaultFeature.MotionBlur=False
r.DefaultFeature.LensFlare=False
r.TemporalAA.Upsampling=False
r.SSGI.Enable=False
r.AntiAliasingMethod=3
r.DefaultFeature.LightUnits=1
r.DefaultBackBufferPixelFormat=4
r.Shadow.UnbuiltPreviewInGame=True
r.StencilForLODDither=False
r.EarlyZPass=3
r.EarlyZPassOnlyMaterialMasking=False
r.DBuffer=True
r.ClearSceneMethod=1
r.VelocityOutputPass=0
r.Velocity.EnableVertexDeformation=0
r.SelectiveBasePassOutputs=False
bDefaultParticleCutouts=False
fx.GPUSimulationTextureSizeX=1024
fx.GPUSimulationTextureSizeY=1024
r.AllowGlobalClipPlane=False
r.GBufferFormat=1
r.MorphTarget.Mode=True
r.GPUCrashDebugging=False
vr.InstancedStereo=True
r.MobileHDR=False
vr.MobileMultiView=True
r.Mobile.UseHWsRGBEncoding=True
vr.RoundRobinOcclusion=False
vr.ODSCapture=False
r.MeshStreaming=False
r.WireframeCullThreshold=5.000000
r.RayTracing=False
r.RayTracing.UseTextureLod=False
r.SupportStationarySkylight=True
r.SupportLowQualityLightmaps=True
r.SupportPointLightWholeSceneShadows=True
r.SupportAtmosphericFog=True
r.SupportSkyAtmosphere=True
r.SupportSkyAtmosphereAffectsHeightFog=False
r.SkinCache.CompileShaders=False
r.SkinCache.DefaultBehavior=1
r.SkinCache.SceneMemoryLimitInMB=128.000000
r.Mobile.EnableStaticAndCSMShadowReceivers=True
r.Mobile.EnableMovableLightCSMShaderCulling=True
r.Mobile.AllowDistanceFieldShadows=True
r.Mobile.AllowMovableDirectionalLights=True
r.MobileNumDynamicPointLights=4
r.MobileDynamicPointLightsUseStaticBranch=True
```

```
r.Mobile.EnableMovableSpotlights=False
r.Mobile.EnableMovableSpotlightsShadow=False
r.GPUSkin.Support16BitBoneIndex=False
r.GPUSkin.Limit2BoneInfluences=False
r.SupportDepthOnlyIndexBuffers=True
r.SupportReversedIndexBuffers=True
r.LightPropagationVolume=False
r.Mobile.AmbientOcclusion=False
r.GPUSkin.UnlimitedBoneInfluences=False
r.GPUSkin.UnlimitedBoneInfluencesThreshold=8
r.Mobile.PlanarReflectionMode=0
bStreamSkeletalMeshLODs=(Default=False,PerPlatform=())
bDiscardSkeletalMeshOptionalLODs=(Default=False,PerPlatform=())
VisualizeCalibrationColorMaterialPath=None
VisualizeCalibrationCustomMaterialPath=None
VisualizeCalibrationGrayscaleMaterialPath=None
r.Mobile.AntiAliasing=3
r.Mobile.FloatPrecisionMode=2
r.OpenGL.ForceDXC=0
r.DynamicGlobalIlluminationMethod=1
r.ReflectionMethod=1
r.Shadow.Virtual.Enable=0
r.Lumen.TranslucencyReflections.FrontLayer.EnableForProject=False
```

Game User Settings and Engine Scalability Settings

Unreal Engine offers predefined graphics quality profiles known as Engine Scalability Settings, which can be easily adjusted to optimize performance. These settings can be modified directly within the Unreal Editor through the Settings menu on the toolbar or dynamically at runtime using code. Importantly, these settings are universal, meaning changes made in the Unreal Editor will apply to the game when run in PIE (Play In Editor) mode, and settings adjusted via code will also affect the editor itself.

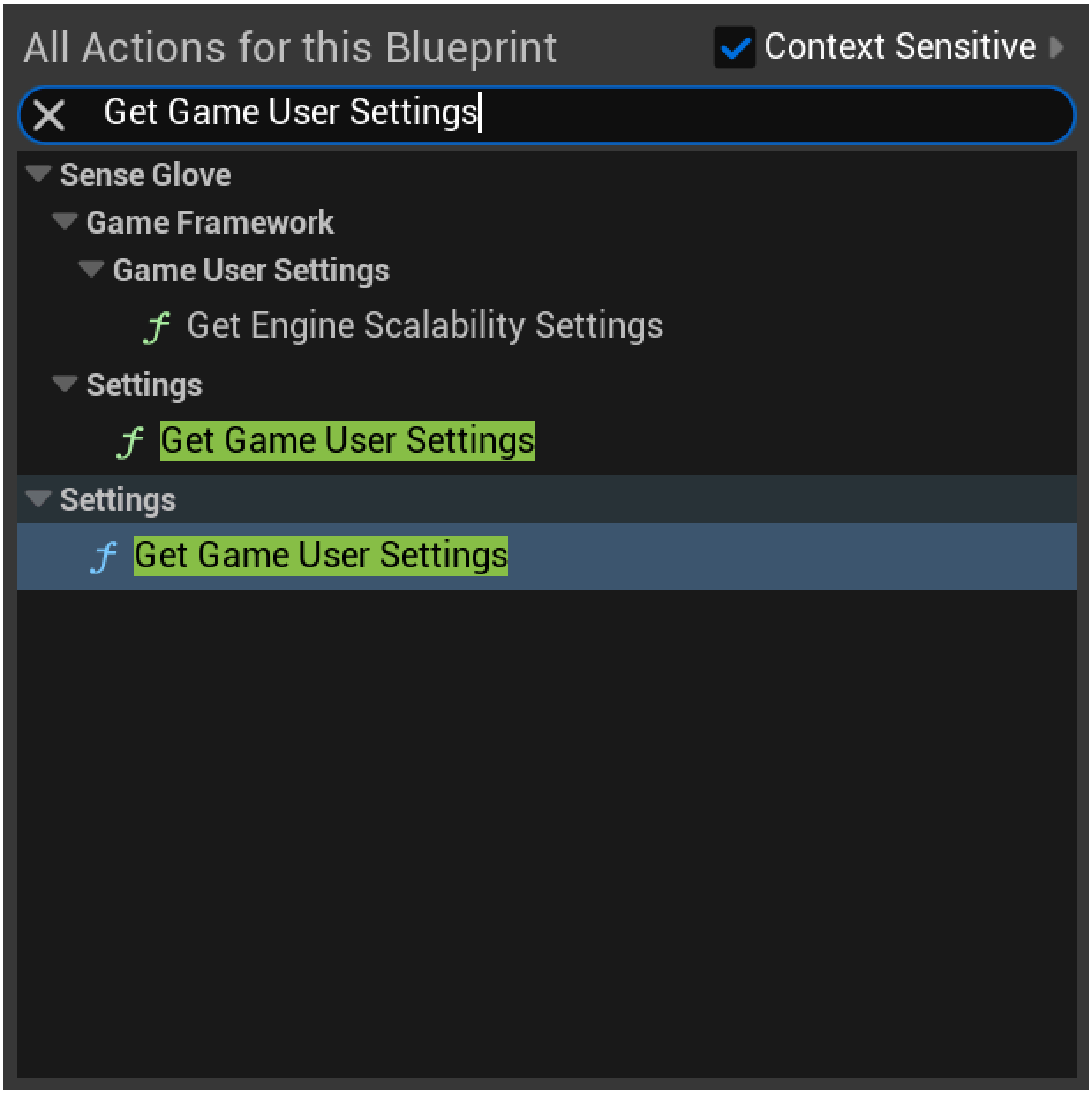


Quality	Low	Medium	High	Epic	Cinematic	Auto
Resolution Scale	100%					
View Distance	Near	Medium	Far	Epic	Cinematic	
Anti-Aliasing	Low	Medium	High	Epic	Cinematic	
Post Processing	Low	Medium	High	Epic	Cinematic	
Shadows	Low	Medium	High	Epic	Cinematic	
Global Illumination	Low	Medium	High	Epic	Cinematic	
Reflections	Low	Medium	High	Epic	Cinematic	
Textures	Low	Medium	High	Epic	Cinematic	
Effects	Low	Medium	High	Epic	Cinematic	
Foliage	Low	Medium	High	Epic	Cinematic	
Shading	Low	Medium	High	Epic	Cinematic	

Note

The SenseGlove Unreal Engine Plugin includes specialized console commands that allow you to switch between different Engine Scalability Settings on the fly. Please note that these commands require you to set up SGGameInstance and `SGGameUserSettings`.

In order to switch between various Engine Scalability Settings, you can use the `Get Game User Settings` Blueprint function and then cast it to `SGGameInstance`.

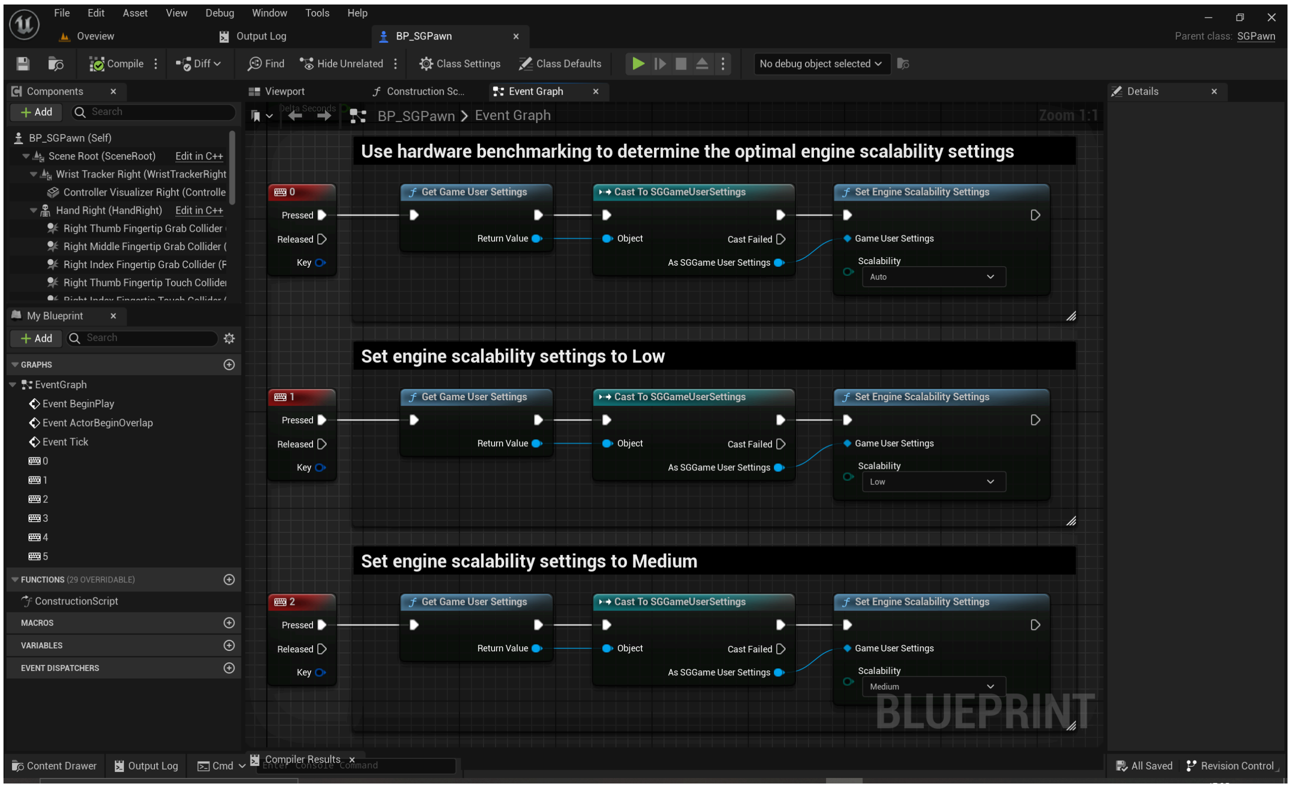


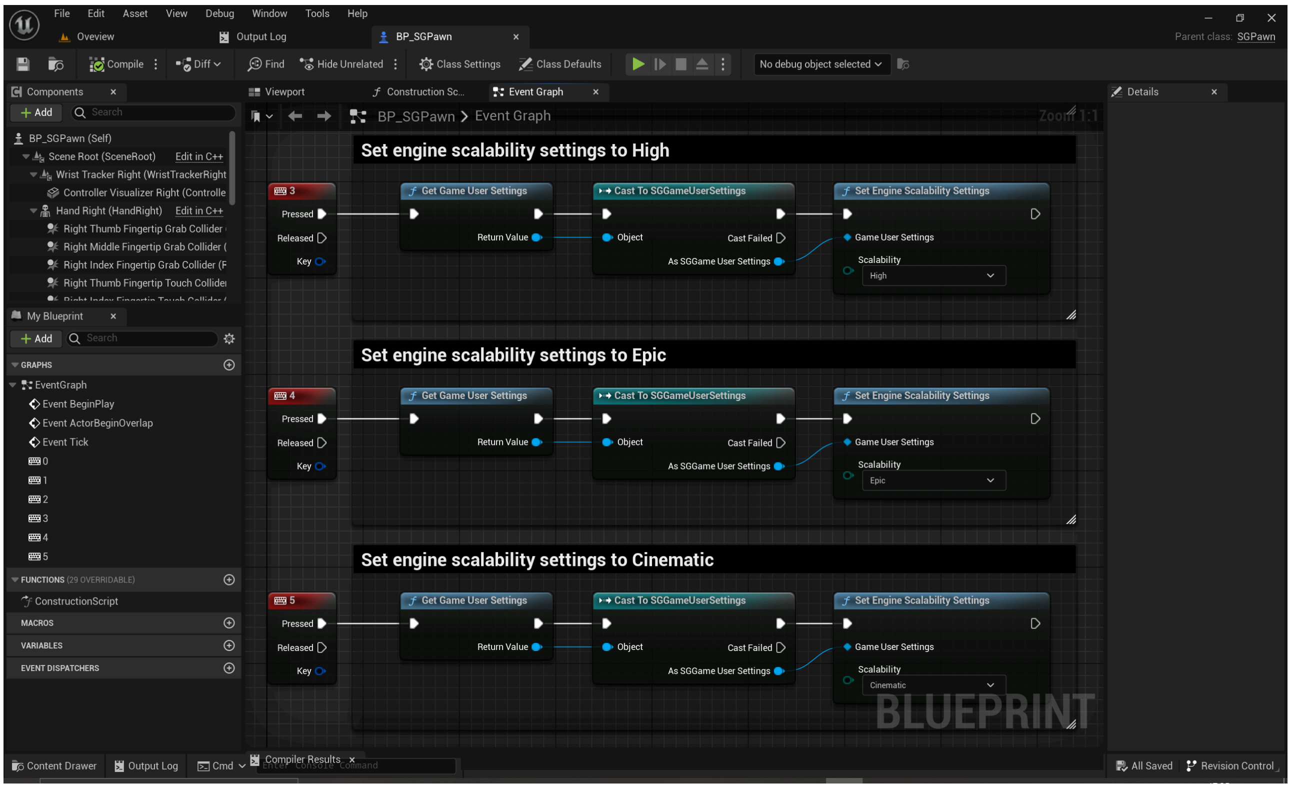
Important

Unreal Engine's default Blueprint functions only allow you to set Engine Scalability Settings to `Low` or `Epic`. To access the full range of settings, `SGGameUserSettings` extends Blueprint access to all Engine Scalability Settings and includes hardware benchmarking to detect the optimal settings. Therefore, it's essential to make `SGGameUserSettings` or a subclass of it the default Game User Settings class to utilize all these features.

The following Blueprint code from the SGBasicDemo-OpenXR example scene demonstrates how to bind numeric keys `1` to `5` to set various Engine Scalability Settings, and key `0` to utilize hardware benchmarking to determine the optimal Engine Scalability Settings:

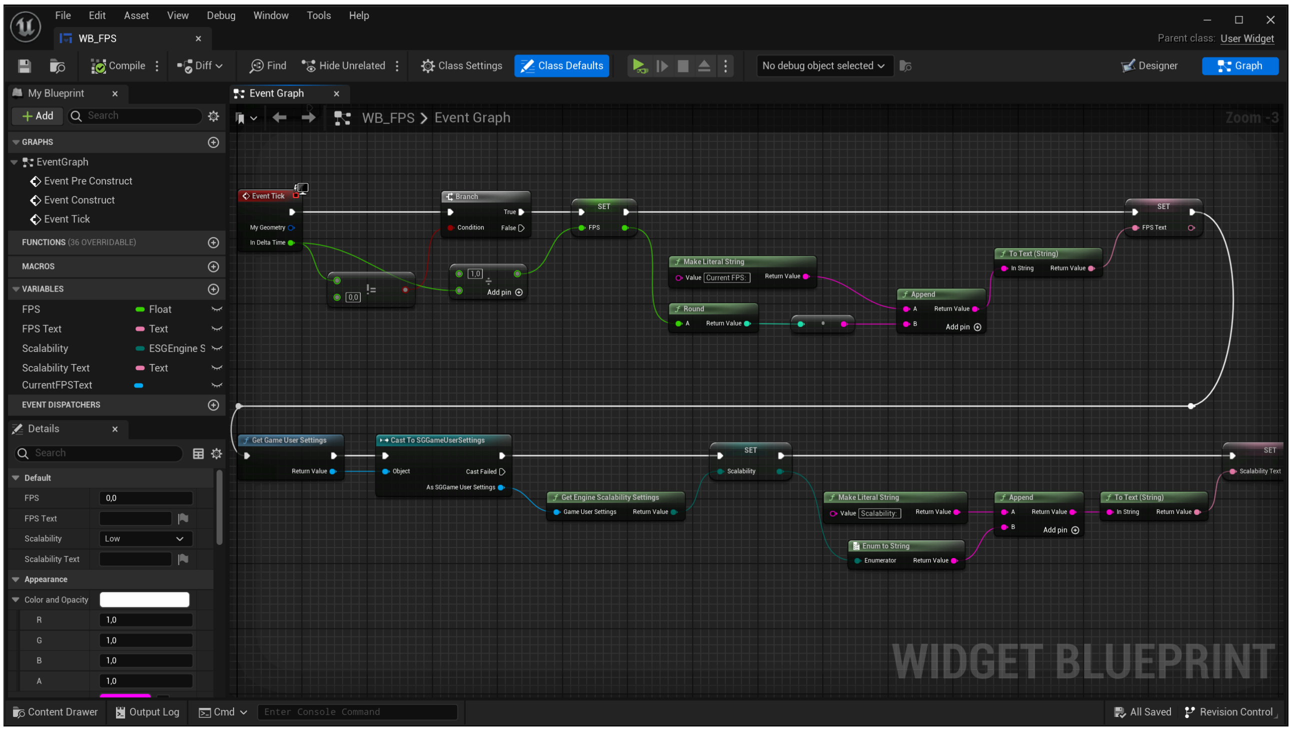
- `0`: Use hardware-benchmarking to determine the optimal Engine Scalability Settings.
- `1`: Set Engine Scalability Settings to `Low`.
- `2`: Set Engine Scalability Settings to `Medium`.
- `3`: Set Engine Scalability Settings to `High`.
- `4`: Set Engine Scalability Settings to `Epic`.
- `5`: Set Engine Scalability Settings to `Cinematic`.




BP_SGPawn > Event Graph
Set engine scalability settings to High
3
Pressed
Released
Key
Get Game User Settings
Return Value
Cast To SGGameUserSettings
Object
Cast Failed
As SGGame User Settings
Set Engine Scalability Settings
Game User Settings
Scalability
High
Set engine scalability settings to Epic
4
Epic
Set engine scalability settings to Cinematic
5
Cinematic
BLUEPRINT

💡 Tip

The SGBasicDemo-OpenXR includes an example 3D widget actor that displays the current FPS and Engine Scalability Settings. This widget can be placed within a VR scene and is located in `All > Content > Blueprints > UI > BP_FPS3DWidget`. The underlying UMG widget can be found at `All > Content > Blueprints > UI > WB_FPS` within the Content Browser for the SGBasicDemo-OpenXR example scene.

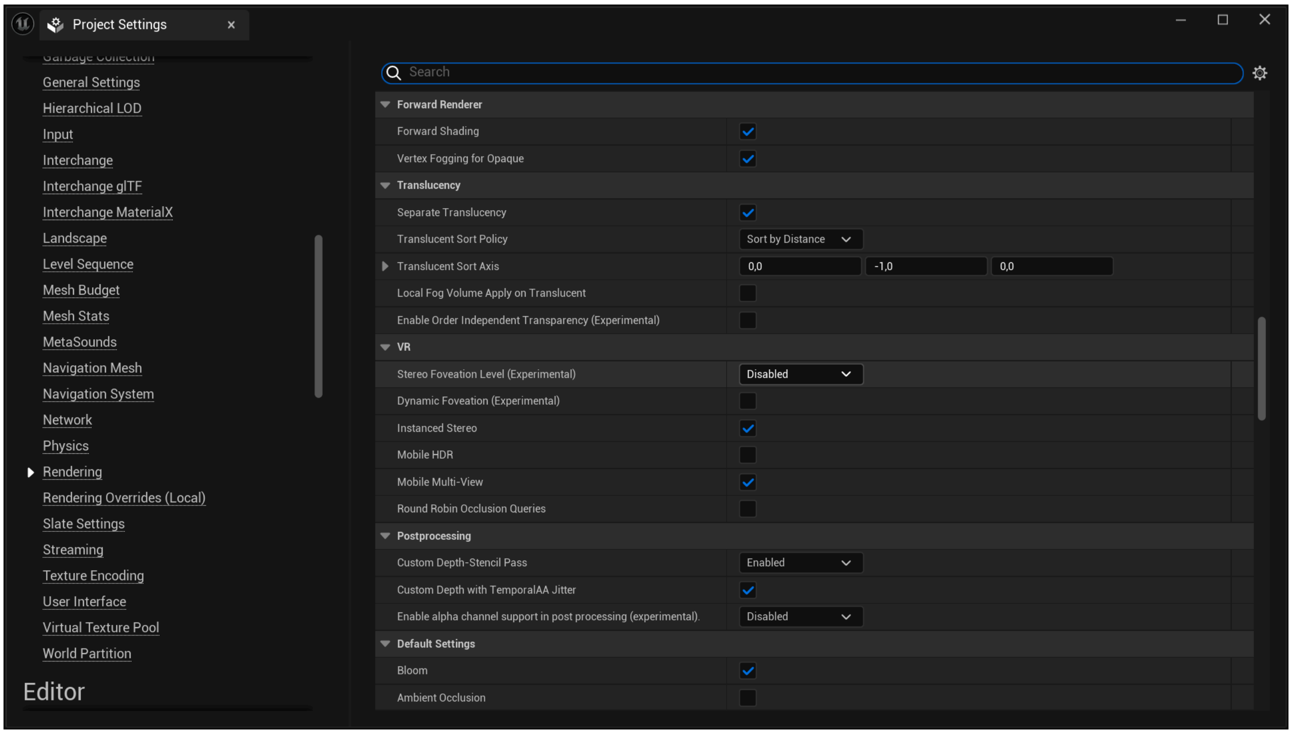


Optimizing Unreal Projects for Mobile

We have the SGBasicDemo-OpenXR project, which has been optimized for mobile. You can explore the project configuration by reviewing the settings inside the `Config` folder and compare them with your own project settings. In addition, here are some crucial guidelines and settings that you may want to adjust for further optimization:

General Rendering Settings

Forward Shading: Enable Forward Shading for better performance. It's more efficient on mobile platforms.



Forward Renderer

Forward Shading	✓
Vertex Fogging for Opaque	✓

Translucency

Separate Translucency	✓
Translucent Sort Policy	Sort by Distance
Translucent Sort Axis	0,0 / -1,0 / 0,0
Local Fog Volume Apply on Translucent	☐
Enable Order Independent Transparency (Experimental)	☐

VR

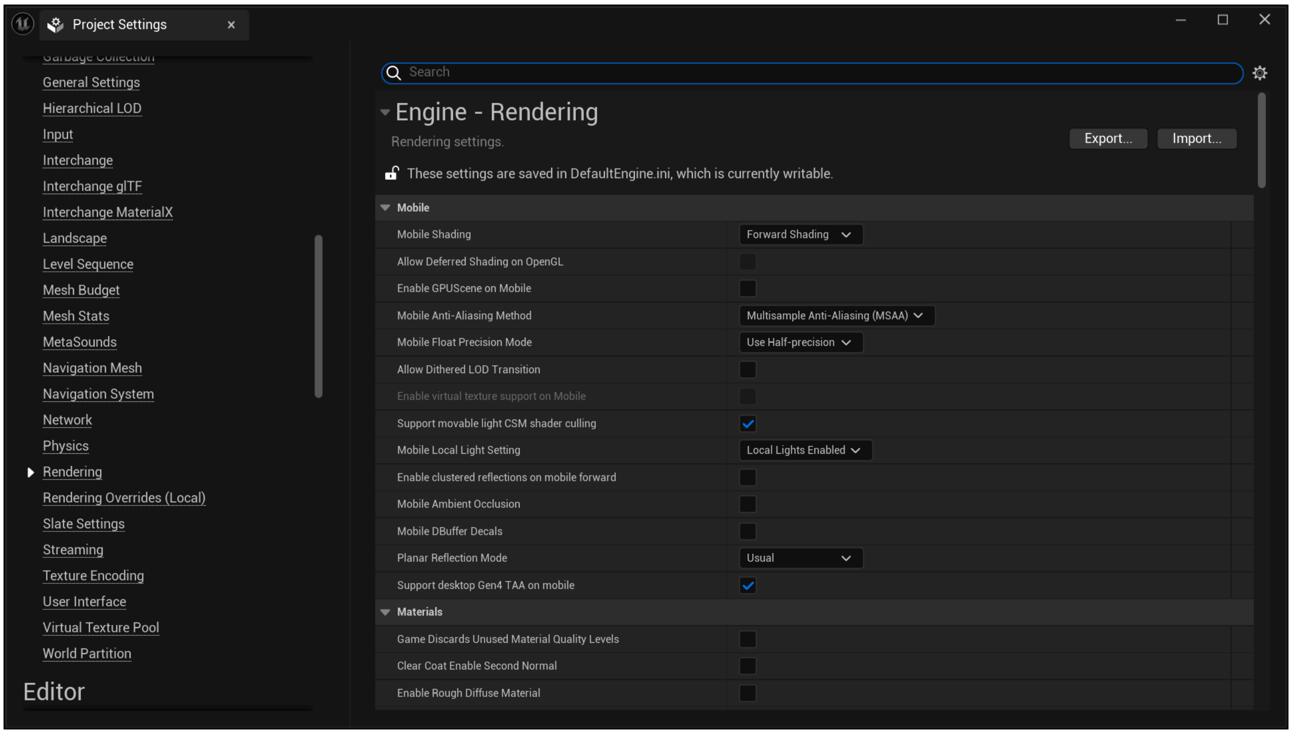
Stereo Foveation Level (Experimental)	Disabled
Dynamic Foveation (Experimental)	☐
Instanced Stereo	✓
Mobile HDR	☐
Mobile Multi-View	✓
Round Robin Occlusion Queries	☐

Postprocessing

Custom Depth-Stencil Pass	Enabled
Custom Depth with TemporalAA Jitter	✓
Enable alpha channel support in post processing (experimental).	Disabled

Default Settings

Bloom	✓
Ambient Occlusion	☐



Engine - Rendering

Rendering settings.

These settings are saved in DefaultEngine.ini, which is currently writable.

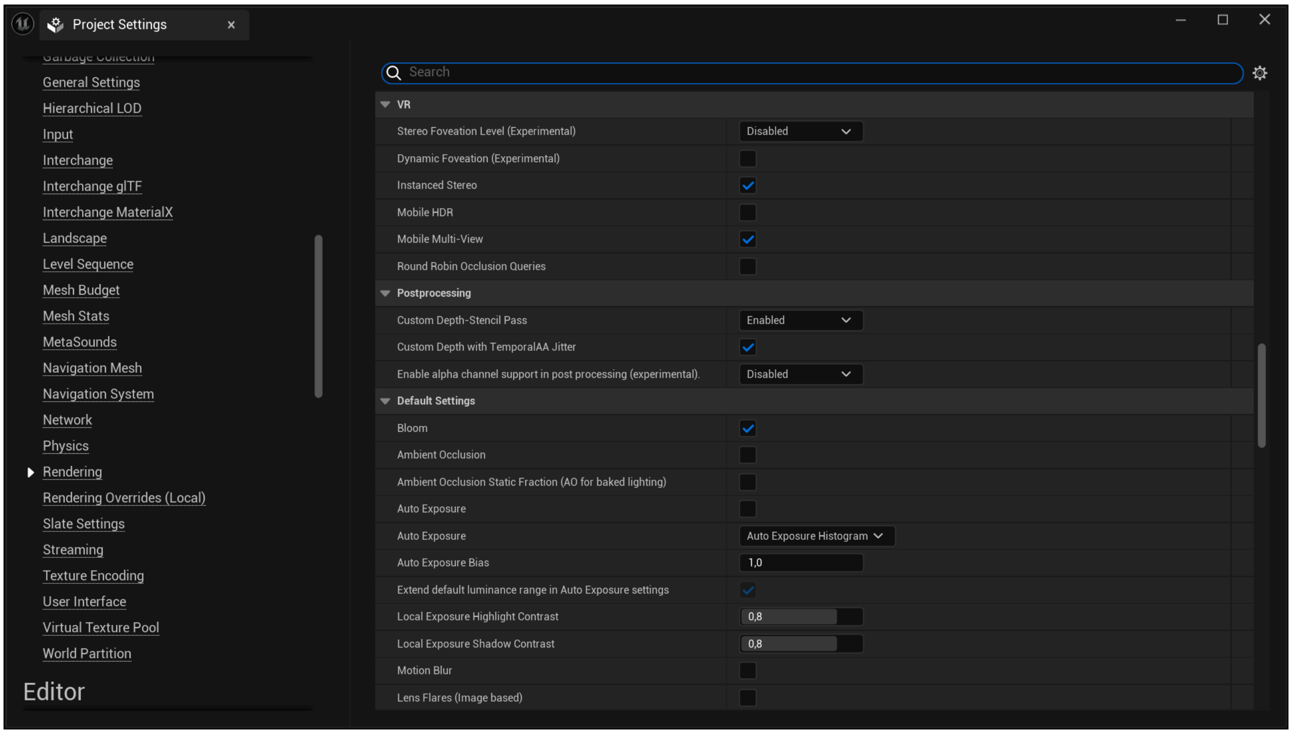
Mobile

Mobile Shading	Forward Shading
Allow Deferred Shading on OpenGL	☐
Enable GPUScene on Mobile	☐
Mobile Anti-Aliasing Method	Multisample Anti-Aliasing (MSAA)
Mobile Float Precision Mode	Use Half-precision
Allow Dithered LOD Transition	☐
Enable virtual texture support on Mobile	☐
Support movable light CSM shader culling	✓
Mobile Local Light Setting	Local Lights Enabled
Enable clustered reflections on mobile forward	☐
Mobile Ambient Occlusion	☐
Mobile DBuffer Decals	☐
Planar Reflection Mode	Usual
Support desktop Gen4 TAA on mobile	✓

Materials

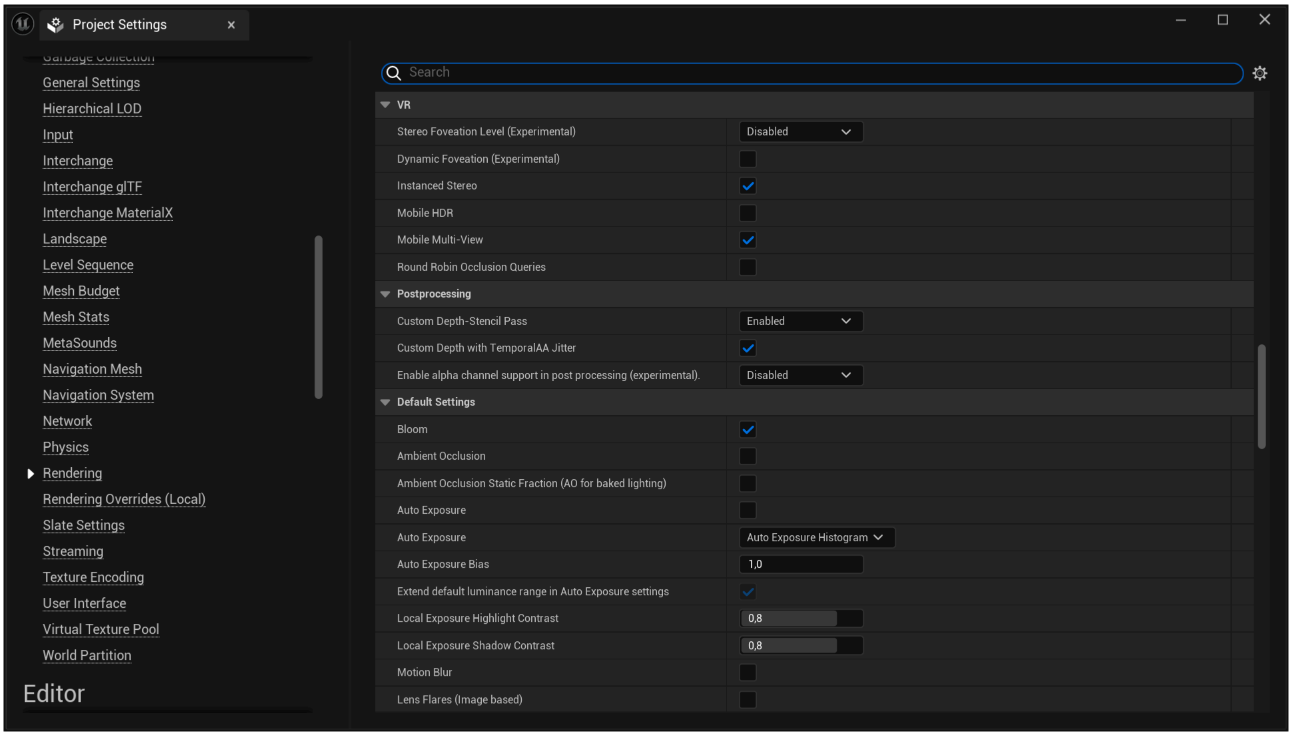
Game Discards Unused Material Quality Levels	☐
Clear Coat Enable Second Normal	☐
Enable Rough Diffuse Material	☐

Mobile HDR: Disable this setting. Mobile HDR can significantly affect performance, especially on lower-end devices.



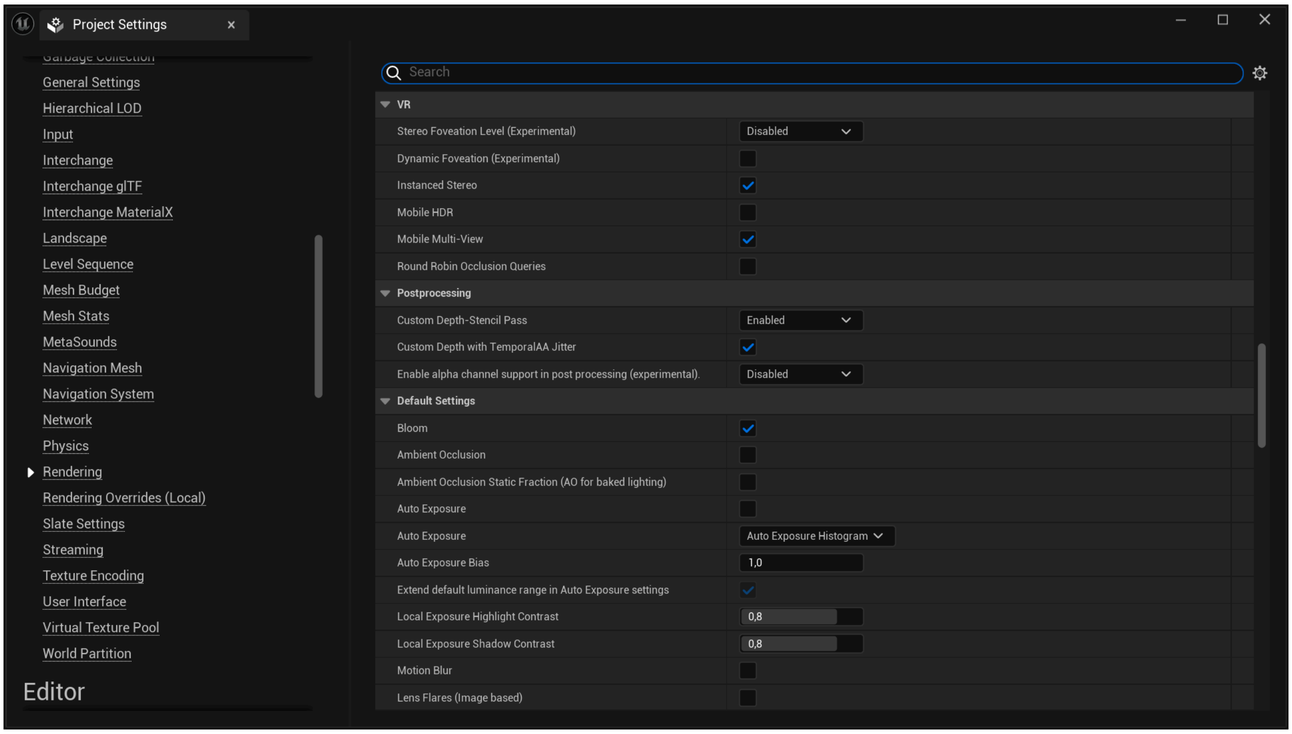
VR	
Stereo Foveation Level (Experimental)	Disabled
Dynamic Foveation (Experimental)	☐
Instanced Stereo	✓
Mobile HDR	☐
Mobile Multi-View	✓
Round Robin Occlusion Queries	☐
Postprocessing	
Custom Depth-Stencil Pass	Enabled
Custom Depth with TemporalAA Jitter	✓
Enable alpha channel support in post processing (experimental).	Disabled
Default Settings	
Bloom	✓
Ambient Occlusion	☐
Ambient Occlusion Static Fraction (AO for baked lighting)	☐
Auto Exposure	☐
Auto Exposure	Auto Exposure Histogram
Auto Exposure Bias	1,0
Extend default luminance range in Auto Exposure settings	✓
Local Exposure Highlight Contrast	0,8
Local Exposure Shadow Contrast	0,8
Motion Blur	☐
Lens Flares (Image based)	☐

Instanced Stereo: Enable this setting. It is a rendering technique used in Unreal Engine primarily for virtual reality (VR) applications. Its main purpose is to optimize the rendering process when creating VR experiences by reducing the workload associated with rendering two slightly different images for each eye.



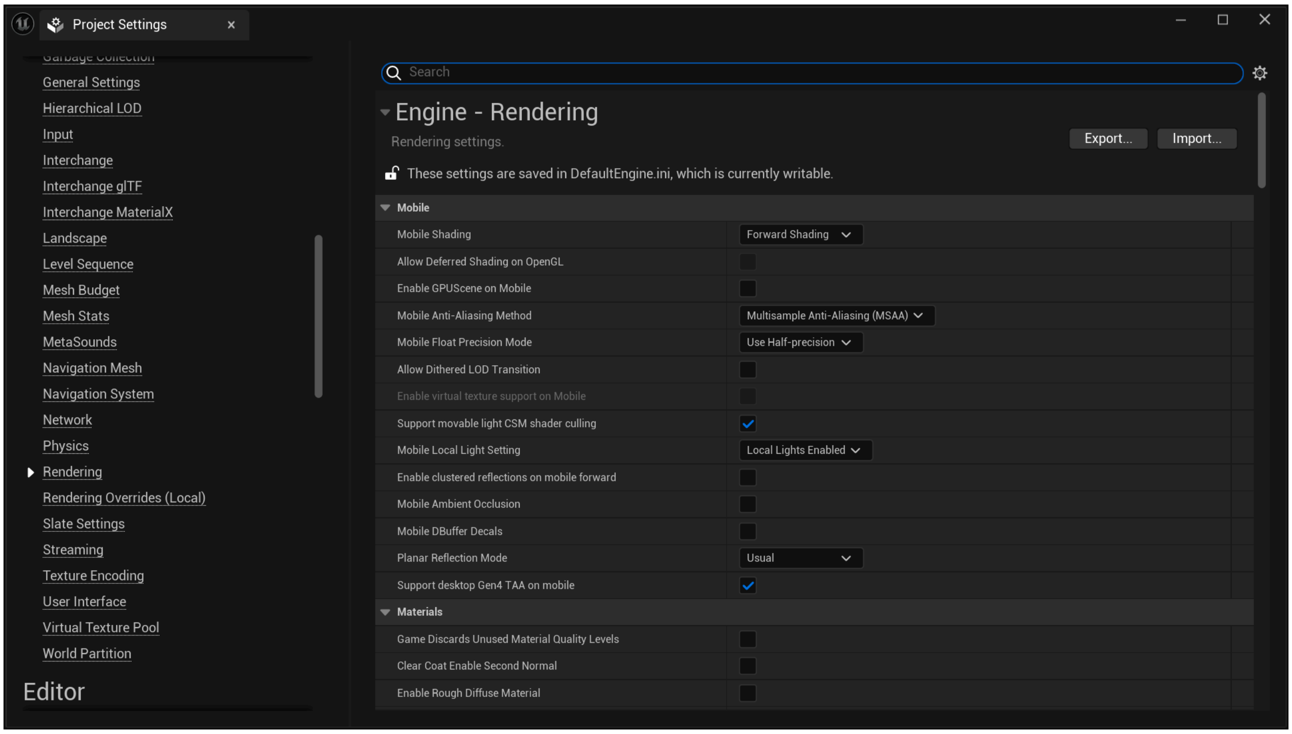
VR	
Stereo Foveation Level (Experimental)	Disabled
Dynamic Foveation (Experimental)	☐
Instanced Stereo	✓
Mobile HDR	☐
Mobile Multi-View	✓
Round Robin Occlusion Queries	☐
Postprocessing	
Custom Depth-Stencil Pass	Enabled
Custom Depth with TemporalAA Jitter	✓
Enable alpha channel support in post processing (experimental).	Disabled
Default Settings	
Bloom	✓
Ambient Occlusion	☐
Ambient Occlusion Static Fraction (AO for baked lighting)	☐
Auto Exposure	☐
Auto Exposure	Auto Exposure Histogram
Auto Exposure Bias	1,0
Extend default luminance range in Auto Exposure settings	✓
Local Exposure Highlight Contrast	0,8
Local Exposure Shadow Contrast	0,8
Motion Blur	☐
Lens Flares (Image based)	☐

Mobile Multi-View: Enable this setting. It is a rendering feature in Unreal Engine designed to optimize the performance of Virtual Reality (VR) applications on mobile devices, particularly when using VR platforms like Google Daydream or Samsung Gear VR. It is similar in concept to Instanced Stereo, but specifically optimized for mobile hardware.



VR	
Stereo Foveation Level (Experimental)	Disabled
Dynamic Foveation (Experimental)	
Instanced Stereo	✓
Mobile HDR	
Mobile Multi-View	✓
Round Robin Occlusion Queries	
Postprocessing	
Custom Depth-Stencil Pass	Enabled
Custom Depth with TemporalAA Jitter	✓
Enable alpha channel support in post processing (experimental).	Disabled
Default Settings	
Bloom	✓
Ambient Occlusion	
Ambient Occlusion Static Fraction (AO for baked lighting)	
Auto Exposure	
Auto Exposure	Auto Exposure Histogram
Auto Exposure Bias	1,0
Extend default luminance range in Auto Exposure settings	✓
Local Exposure Highlight Contrast	0,8
Local Exposure Shadow Contrast	0,8
Motion Blur	
Lens Flares (Image based)	

Mobile Anti-Aliasing Method: Use `FXAA (Fast Approximate Anti-Aliasing)` or `MSAA (Multisample Anti-Aliasing)`. `MSAA` is often preferred for mobile as it gives better visual quality without a huge performance hit.



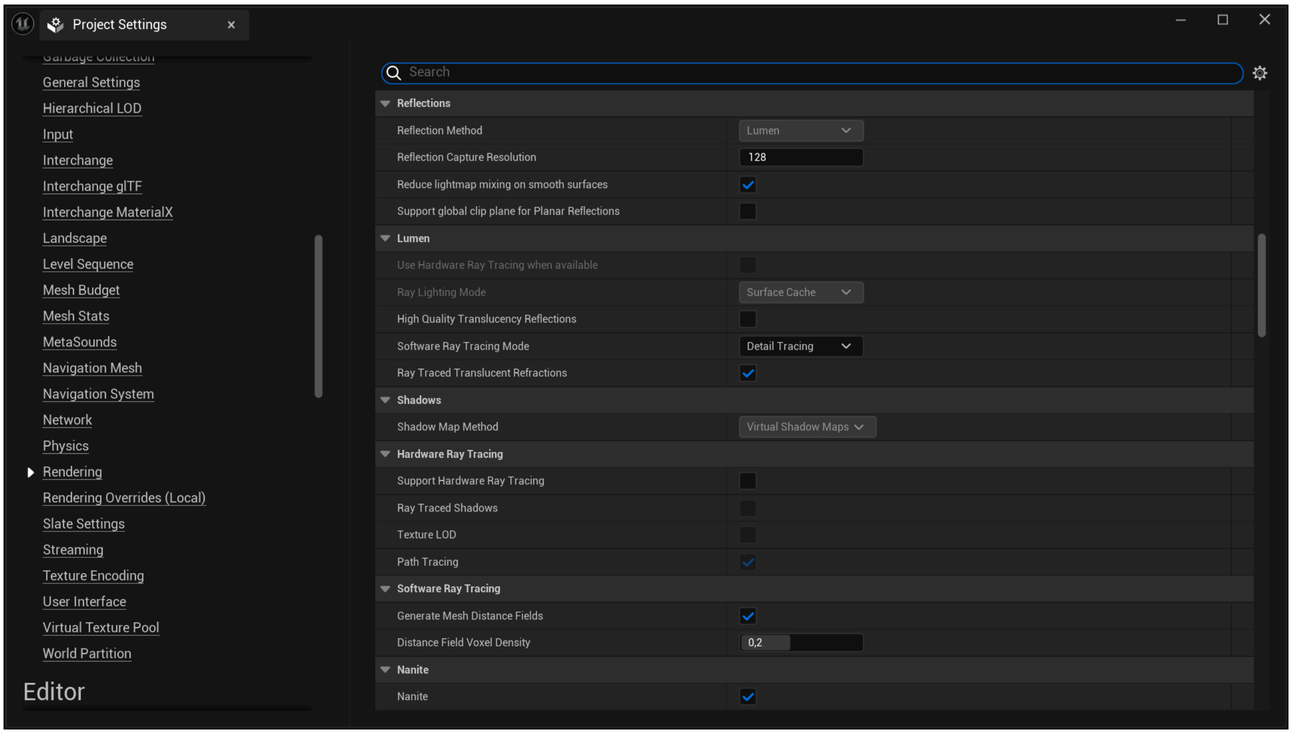
Engine - Rendering

Rendering settings.

These settings are saved in DefaultEngine.ini, which is currently writable.

Mobile	
Mobile Shading	Forward Shading
Allow Deferred Shading on OpenGL	☐
Enable GPUScene on Mobile	☐
Mobile Anti-Aliasing Method	Multisample Anti-Aliasing (MSAA)
Mobile Float Precision Mode	Use Half-precision
Allow Dithered LOD Transition	☐
Enable virtual texture support on Mobile	☐
Support movable light CSM shader culling	✓
Mobile Local Light Setting	Local Lights Enabled
Enable clustered reflections on mobile forward	☐
Mobile Ambient Occlusion	☐
Mobile DBuffer Decals	☐
Planar Reflection Mode	Usual
Support desktop Gen4 TAA on mobile	✓
Materials	
Game Discards Unused Material Quality Levels	☐
Clear Coat Enable Second Normal	☐
Enable Rough Diffuse Material	☐

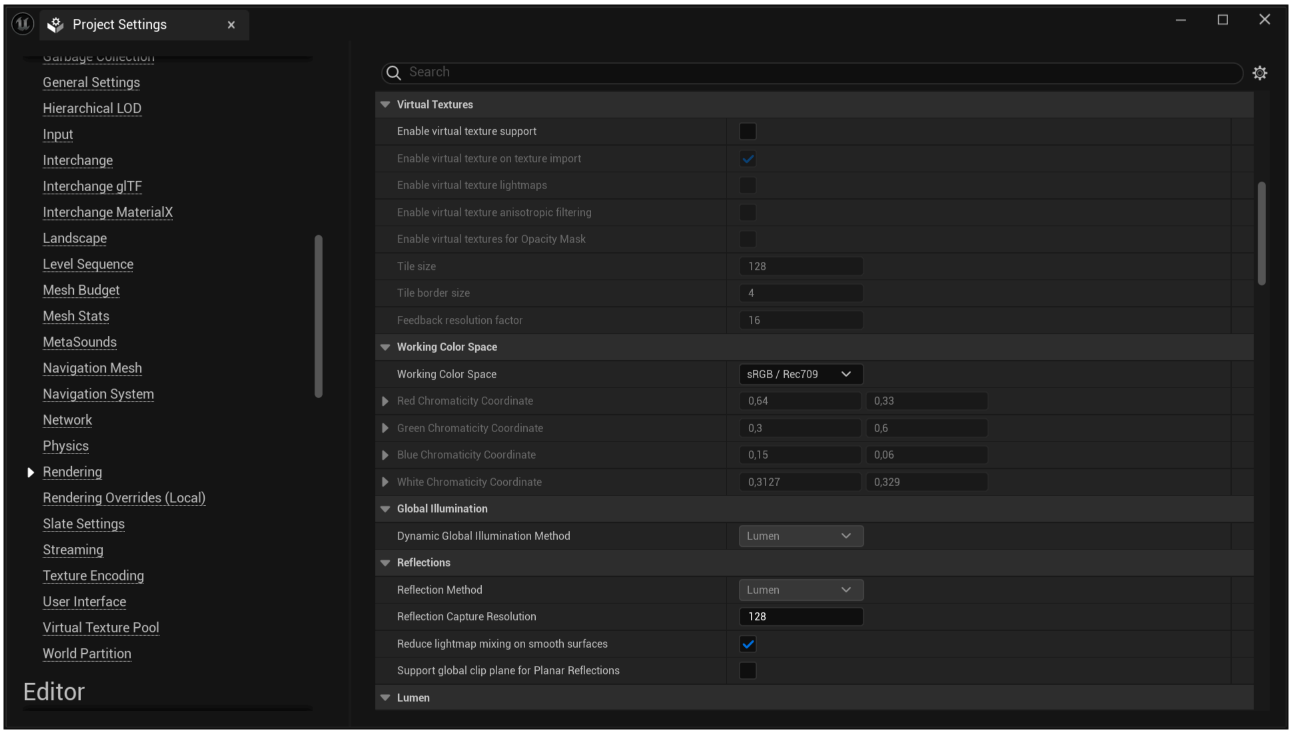
Reflection Capture Resolution: Reduce this value (e.g., 128 or 256) to decrease the memory usage.



Reflections	
Reflection Method	Lumen
Reflection Capture Resolution	128
Reduce lightmap mixing on smooth surfaces	✓
Support global clip plane for Planar Reflections	☐
Lumen	
Use Hardware Ray Tracing when available	☐
Ray Lighting Mode	Surface Cache
High Quality Translucency Reflections	☐
Software Ray Tracing Mode	Detail Tracing
Ray Traced Translucent Refractions	✓
Shadows	
Shadow Map Method	Virtual Shadow Maps
Hardware Ray Tracing	
Support Hardware Ray Tracing	☐
Ray Traced Shadows	☐
Texture LOD	☐
Path Tracing	✓
Software Ray Tracing	
Generate Mesh Distance Fields	✓
Distance Field Voxel Density	0,2
Nanite	
Nanite	✓

Texture Settings

Enable virtual texture support: Disable this setting.



Project Settings

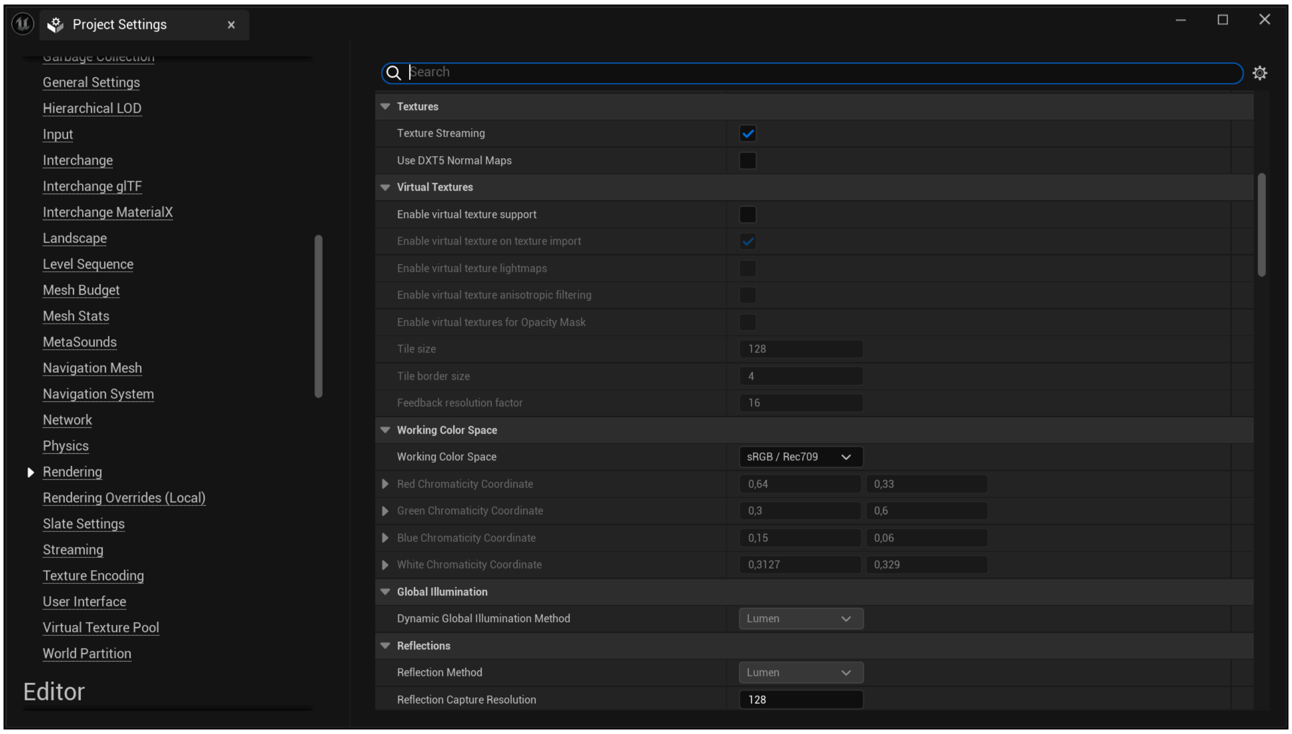
Search

Garbage Collection
General Settings
Hierarchical LOD
Input
Interchange
Interchange glTF
Interchange MaterialX
Landscape
Level Sequence
Mesh Budget
Mesh Stats
MetaSounds
Navigation Mesh
Navigation System
Network
Physics
Rendering
Rendering Overrides (Local)
Slate Settings
Streaming
Texture Encoding
User Interface
Virtual Texture Pool
World Partition

Editor

Virtual Textures	
Enable virtual texture support	☐
Enable virtual texture on texture import	✓
Enable virtual texture lightmaps	☐
Enable virtual texture anisotropic filtering	☐
Enable virtual textures for Opacity Mask	☐
Tile size	128
Tile border size	4
Feedback resolution factor	16
Working Color Space	
Working Color Space	sRGB / Rec709
Red Chromaticity Coordinate	0,64 / 0,33
Green Chromaticity Coordinate	0,3 / 0,6
Blue Chromaticity Coordinate	0,15 / 0,06
White Chromaticity Coordinate	0,3127 / 0,329
Global Illumination	
Dynamic Global Illumination Method	Lumen
Reflections	
Reflection Method	Lumen
Reflection Capture Resolution	128
Reduce lightmap mixing on smooth surfaces	✓
Support global clip plane for Planar Reflections	☐
Lumen	

Texture Streaming: Enable texture streaming to ensure textures load progressively, which helps in reducing memory usage.



Textures
- Texture Streaming: ✓
- Use DXT5 Normal Maps: ☐

Virtual Textures
- Enable virtual texture support: ☐
- Enable virtual texture on texture import: ✓
- Enable virtual texture lightmaps: ☐
- Enable virtual texture anisotropic filtering: ☐
- Enable virtual textures for Opacity Mask: ☐
- Tile size: 128
- Tile border size: 4
- Feedback resolution factor: 16

Working Color Space
- Working Color Space: sRGB / Rec709
- Red Chromaticity Coordinate: 0,64 0,33
- Green Chromaticity Coordinate: 0,3 0,6
- Blue Chromaticity Coordinate: 0,15 0,06
- White Chromaticity Coordinate: 0,3127 0,329

Global Illumination
- Dynamic Global Illumination Method: Lumen

Reflections
- Reflection Method: Lumen
- Reflection Capture Resolution: 128

Texture Quality: Lower the overall texture quality to Medium or Low depending on the target device capabilities.

Texture Compression: Use ASTC compression for Android to ensure the textures are optimized for mobile devices.

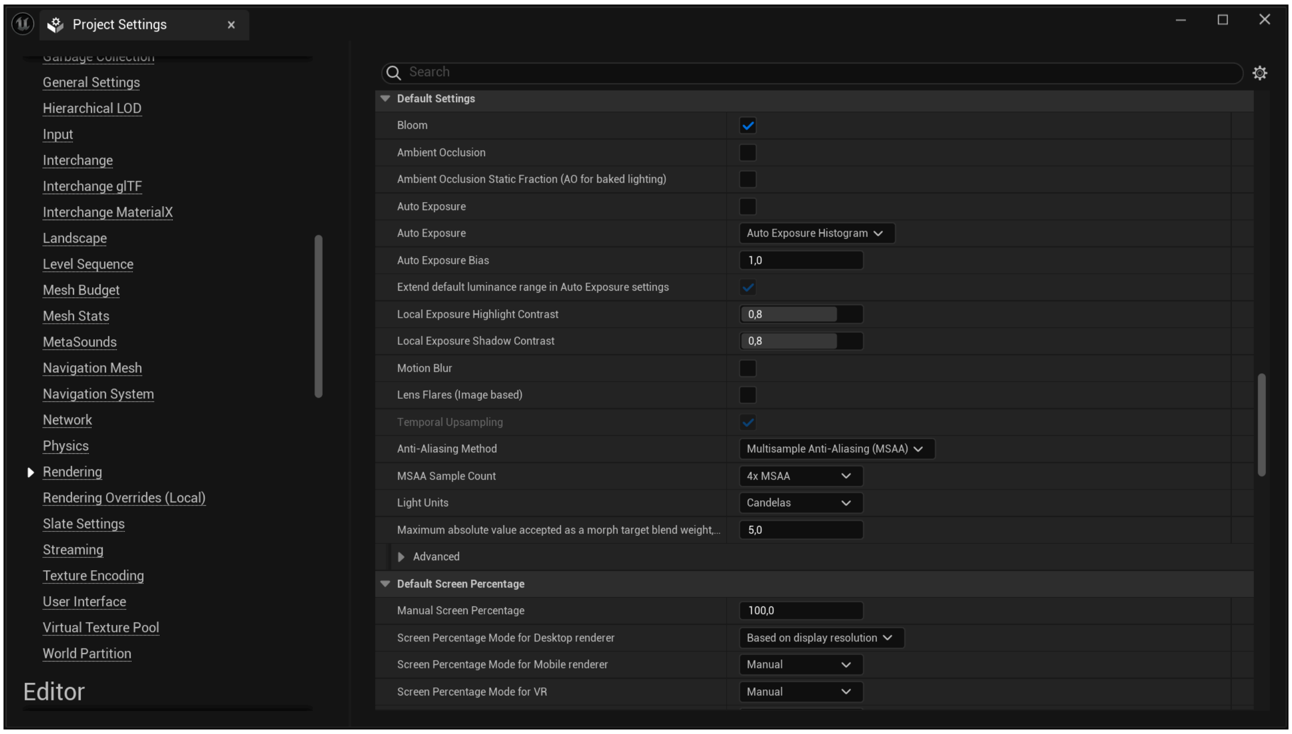
Lighting Settings

Use Static Lighting: Prefer static lighting over dynamic lighting for better performance.

Lightmap Resolution: Use a lower lightmap resolution (e.g., 32 or 64) for mobile to reduce memory usage.

Dynamic Shadows: Disable or minimize the use of dynamic shadows. If required, use CSM (Cascaded Shadow Maps) with low resolution and distance.

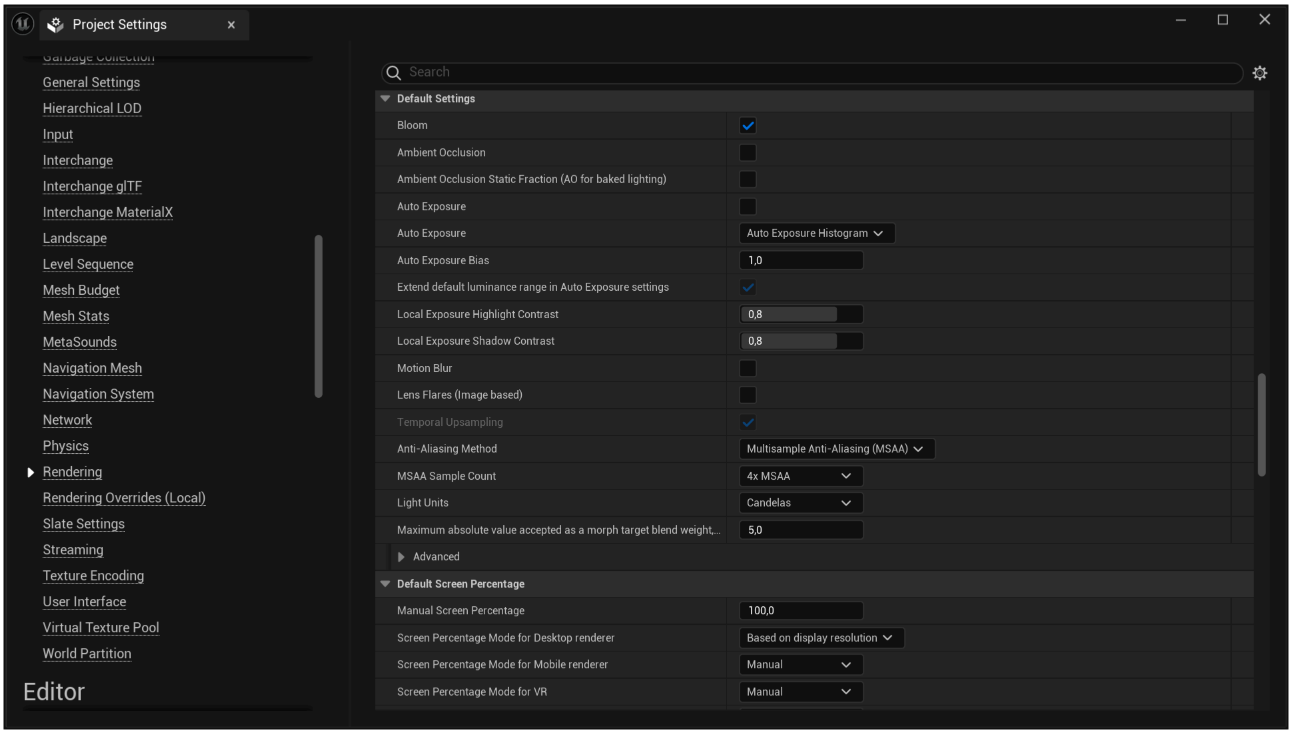
Distance Field Shadows/Ambient Occlusion: Disable these features as they are costly on mobile platforms.



Default Settings	
Bloom	✓
Ambient Occlusion	☐
Ambient Occlusion Static Fraction (AO for baked lighting)	☐
Auto Exposure	☐
Auto Exposure	Auto Exposure Histogram
Auto Exposure Bias	1,0
Extend default luminance range in Auto Exposure settings	✓
Local Exposure Highlight Contrast	0,8
Local Exposure Shadow Contrast	0,8
Motion Blur	☐
Lens Flares (Image based)	☐
Temporal Upsampling	✓
Anti-Aliasing Method	Multisample Anti-Aliasing (MSAA)
MSAA Sample Count	4x MSAA
Light Units	Candelas
Maximum absolute value accepted as a morph target blend weight,...	5,0
Default Screen Percentage	
Manual Screen Percentage	100,0
Screen Percentage Mode for Desktop renderer	Based on display resolution
Screen Percentage Mode for Mobile renderer	Manual
Screen Percentage Mode for VR	Manual

Post-Processing Settings

Bloom, Lens Flares, and Auto Exposure: Minimize or disable these effects as they can be performance-intensive.



Project Settings

General Settings
Hierarchical LOD
Input
Interchange
Interchange glTF
Interchange MaterialX
Landscape
Level Sequence
Mesh Budget
Mesh Stats
MetaSounds
Navigation Mesh
Navigation System
Network
Physics
Rendering
Rendering Overrides (Local)
Slate Settings
Streaming
Texture Encoding
User Interface
Virtual Texture Pool
World Partition

Editor

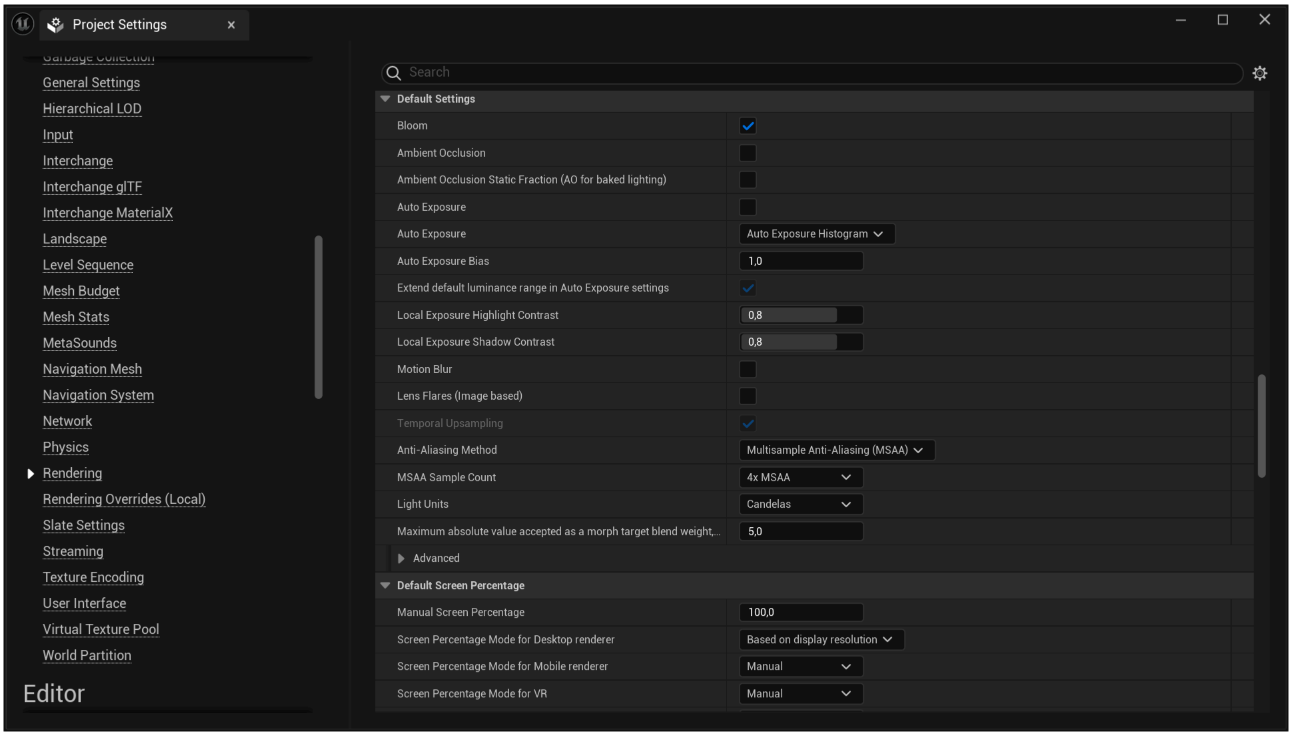
Search

Default Settings	
Bloom	✓
Ambient Occlusion	☐
Ambient Occlusion Static Fraction (AO for baked lighting)	☐
Auto Exposure	☐
Auto Exposure	Auto Exposure Histogram
Auto Exposure Bias	1,0
Extend default luminance range in Auto Exposure settings	✓
Local Exposure Highlight Contrast	0,8
Local Exposure Shadow Contrast	0,8
Motion Blur	☐
Lens Flares (Image based)	☐
Temporal Upsampling	✓
Anti-Aliasing Method	Multisample Anti-Aliasing (MSAA)
MSAA Sample Count	4x MSAA
Light Units	Candelas
Maximum absolute value accepted as a morph target blend weight,...	5,0
Advanced	

Default Screen Percentage	
Manual Screen Percentage	100,0
Screen Percentage Mode for Desktop renderer	Based on display resolution
Screen Percentage Mode for Mobile renderer	Manual
Screen Percentage Mode for VR	Manual

Screen Space Reflections: Disable this setting as it is costly in terms of performance on mobile devices.

Motion Blur: Disable this feature to save on processing power.



Project Settings

General Settings
Hierarchical LOD
Input
Interchange
Interchange glTF
Interchange MaterialX
Landscape
Level Sequence
Mesh Budget
Mesh Stats
MetaSounds
Navigation Mesh
Navigation System
Network
Physics
Rendering
Rendering Overrides (Local)
Slate Settings
Streaming
Texture Encoding
User Interface
Virtual Texture Pool
World Partition

Editor

Search

Default Settings	
Bloom	✓
Ambient Occlusion	☐
Ambient Occlusion Static Fraction (AO for baked lighting)	☐
Auto Exposure	☐
Auto Exposure	Auto Exposure Histogram
Auto Exposure Bias	1,0
Extend default luminance range in Auto Exposure settings	✓
Local Exposure Highlight Contrast	0,8
Local Exposure Shadow Contrast	0,8
Motion Blur	☐
Lens Flares (Image based)	☐
Temporal Upsampling	✓
Anti-Aliasing Method	Multisample Anti-Aliasing (MSAA)
MSAA Sample Count	4x MSAA
Light Units	Candelas
Maximum absolute value accepted as a morph target blend weight,...	5,0
Advanced	

Default Screen Percentage	
Manual Screen Percentage	100,0
Screen Percentage Mode for Desktop renderer	Based on display resolution
Screen Percentage Mode for Mobile renderer	Manual
Screen Percentage Mode for VR	Manual

Materials and Shaders

Material Complexity: Use simple materials with few instructions and limit the number of textures and shader nodes.

Specular Highlights: Consider reducing or disabling specular highlights on materials to save on performance.

LOD (Level of Detail) Models: Ensure that LODs are set up correctly for all models, with appropriate reduction in polygon count for distant objects.

Level of Detail (LOD) Settings

Mesh LODs: Configure LODs for all meshes to reduce polygon count at distances.

Screen Size: Adjust screen size settings for LODs to ensure they switch at appropriate distances for mobile screens.

Engine Scalability Settings

Resolution Scale: Lower the resolution scale (e.g., 70% or 80%) to improve performance while maintaining visual quality.

View Distance: Set to Medium or Low to reduce the amount of detail rendered at long distances.

Shadows: Set to Low or Off for better performance.

Textures: Set to Medium or Low depending on the device's capabilities.

Effects: Set to Low to reduce the complexity of visual effects.

> ⓘ Note
>
> See Game User Settings and Engine Scalability Settings for more details.

Physics and Collision

Physics Simulation: Limit the use of physics simulation where possible, as it can be expensive on mobile devices.

Collision Complexity: Use simple collision meshes instead of complex ones to improve performance.

Audio Settings

Sample Rate: Lower the sample rate to reduce memory usage and processing load.

Number of Audio Channels: Limit the number of audio channels used in the project to reduce CPU usage.

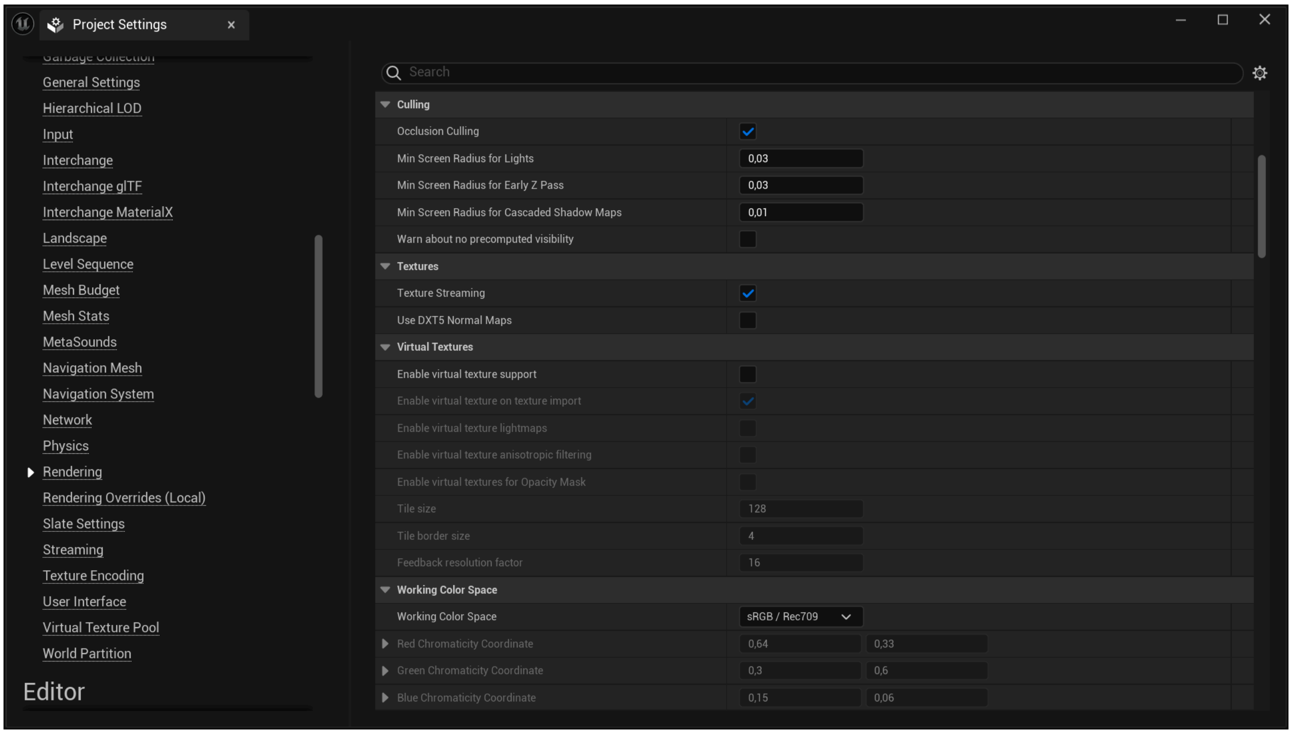
Rendering API

Vulkan vs OpenGL ES: Test your project with both Vulkan and OpenGL ES to see which provides better performance on your target devices. Vulkan often offers better performance but may not be supported on all devices.

Culling

Frustum Culling: Ensure that frustum culling is enabled to avoid rendering objects outside of the camera's view.

Occlusion Culling: Enable occlusion culling to avoid rendering objects that are not visible due to being blocked by other objects.



Culling

Occlusion Culling	✓
Min Screen Radius for Lights	0,03
Min Screen Radius for Early Z Pass	0,03
Min Screen Radius for Cascaded Shadow Maps	0,01
Warn about no precomputed visibility	

Textures

Texture Streaming	✓
Use DXT5 Normal Maps	

Virtual Textures

Enable virtual texture support	
Enable virtual texture on texture import	✓
Enable virtual texture lightmaps	
Enable virtual texture anisotropic filtering	
Enable virtual textures for Opacity Mask	
Tile size	128
Tile border size	4
Feedback resolution factor	16

Working Color Space

Working Color Space	sRGB / Rec709	
Red Chromaticity Coordinate	0,64	0,33
Green Chromaticity Coordinate	0,3	0,6
Blue Chromaticity Coordinate	0,15	0,06

Third-Party Tutorials: Optimizing Your Project for Higher FPS

Optimizing Unreal Engine VR Projects for Higher Framerates (Meta Quest, HTC VIVE, FFR, ETFR, NVIDIA DLSS, AMD FSR, and Intel XeSS Tips Included!)

This beginner-friendly tutorial, covers how to significantly boost the performance of your Unreal Engine VR projects, whether you're building for standalone (mobile) or PCVR (desktop) on devices such as Meta Quest, HTC VIVE, Varjo, or Valve Index.

It covers step-by-step how to:

- Convert a regular project into a VR-ready experience.
- Optimize for both standalone (Android) and PCVR (Windows) platforms.
- Tweak key rendering, lighting, and texture settings for smoother gameplay.
- Configure Meta XR and HTC VIVE OpenXR plugins to fine-tune settings for the best performance possible.
- Utilize and fine-tune engine scalability settings.
- Explore powerful optimization features like DLSS, FSR, XeSS, and Foveated Rendering

Even if you’re new to VR development, this guide breaks it down with visuals, clear examples, and actionable tips to take your framerate from sluggish to silky smooth.

Safe and Reliable Glove Access in Blueprint

Since the Blueprint API uses the underlying C++ API to access the SenseGlove hardware, it often has to deal with C++ pointers. Those who are familiar with C++ and in particular with the Unreal Engine UObject Garbage Collection System are aware that:

- As a general rule of thumb, a pointer should be validated before dereferenced, meaning before accessing the pointer a `NULL` check should be performed, otherwise if the pointer is `NULL` the program is going to crash upon access.
- Unreal implements a garbage collection scheme whereby UObjects that are no longer referenced or have been explicitly flagged for destruction will be cleaned up at regular intervals. The engine builds a reference graph to determine which UObjects are still in use and which ones are orphaned. The ones that are orphaned will be evaluated to `NULL` on the next GC cycle and their allocated memory will be released. Hence, `NULL` checks on UObjects are always mandatory.

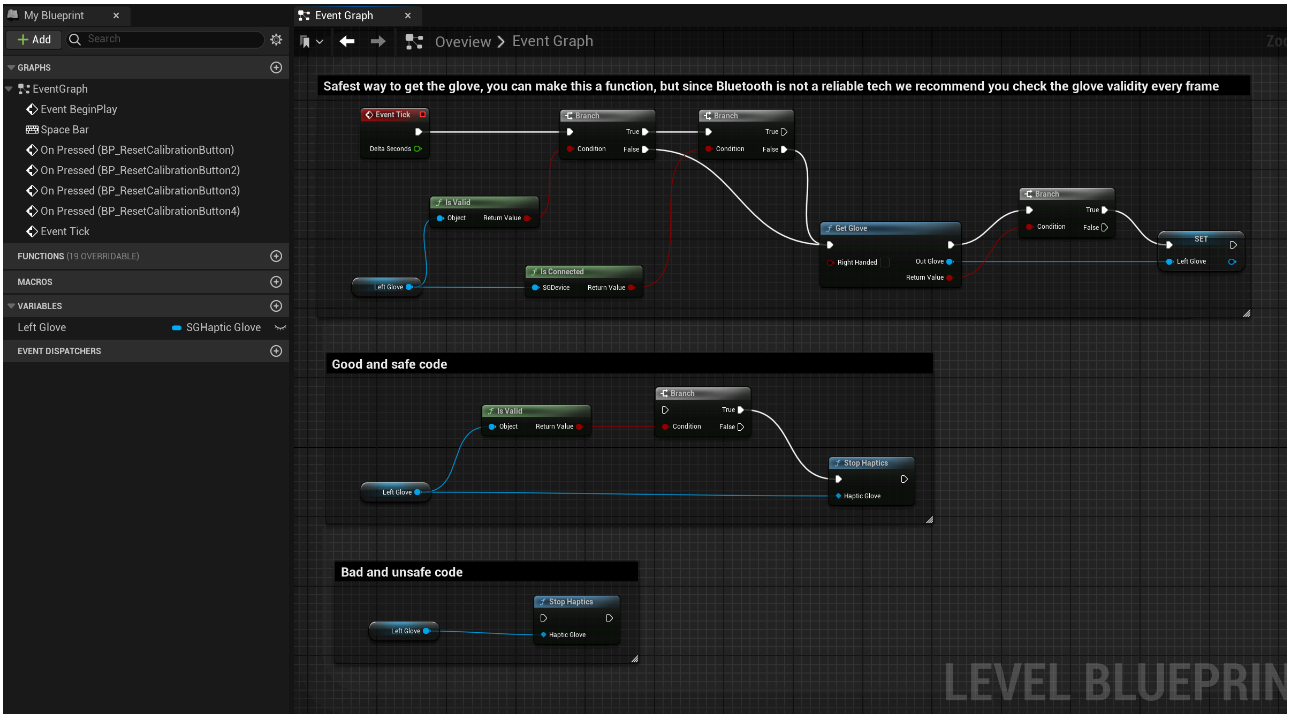
Glove objects inside the SenseGlove Unreal Engine Plugin, utilize the UObject system, and since communication for Nova gloves happens over SenseCom and the Bluetooth protocol, and also the gloves are running on battery, there's always the possibility for a glove variable to become `NULL` and therefore invalidated when the glove hardware for any reason is not accessible.

The recommended way to work with a glove instance without any performance penalty, and in a safe manner in Blueprint is:

1. Cache the glove instance inside a global variable if it passes certain tests so that you don't have to perform all those checks on every access. This usually could happen inside the `Tick` function.
2. The first check inside the `Tick` function is to check whether the cached glove instance is valid. If it's valid we continue to the next step, if not, we ask the API for a new glove instance.

3. If the glove instance is valid, then it's best to perform a connectivity check next. If the glove is connected we don't have to do anything else in regards to obtaining a new glove instance and caching it. If however the glove is not connected, we might ask the API for a new glove instance.
4. If any of the above steps fail, then we can actually ask the API for a new glove instance, and if the result is successful we're going to cache the new glove instance.
5. From here on, anywhere else inside your code, whenever you need to access the glove data or perform an operation like for example sending or stopping haptics you always perform a validity check and only proceed when the glove instance is valid. This way you will always ensure you are accessing the glove instances in a safe and reliable manner, thus avoiding any unexpected behaviors or crashes.

The following Blueprint examples implement the above approach and also demonstrate good and bad glove instance accesses:


Safest way to get the glove, you can make this a function, but since Bluetooth is not a reliable tech we recommend you check the glove validity every frame

Good and safe code

Bad and unsafe code

Roll Your Own Customized Hand Manipulation and Interaction System

The default hand interaction system shipped with the **SenseGlove Unreal Engine Plugin** consists of various components, including `SGPawn`, `SGPlayerController`, `SGVirtualHandComponent`, `SGGrabComponent`, `SGTouchComponent`, and others. This system is very easy to get started with and is thoroughly documented throughout this handbook.

However, this simplicity comes at a cost: limited functionality. At SenseGlove, we prioritize usability and practicality. That said, developing a comprehensive hand interaction system that suits every possible use case is not an easy task. For example, projects such as the VR Expansion Plugin (VRE) — an Epic MegaGrants recipient — have been in development for over a decade, and development is still ongoing. Naturally, such depth also comes with a trade-off: a steep learning curve and reduced beginner-friendliness.

To bridge this gap and serve different groups of users, we provide the `SGPawn` system as a simple, intentionally limited, and beginner-friendly default solution that allows anyone to get up and running quickly.

At the same time, to support intermediate and advanced users, we have aimed for full OpenXR compatibility, opening the door to a wide range of advanced possibilities. Once enabled and loaded in Unreal Engine, the SenseGlove Unreal Engine Plugin registers itself as an `OpenXRHandTracking` provider. This makes it a fully compatible, drop-in replacement for Epic's **OpenXRHandTracking** plugin.

As a result, it can integrate seamlessly with any third-party system or plugin that consumes OpenXR hand-tracking data. Because the SenseGlove plugin is fully OpenXR-compliant, it provides hand-tracking data in the expected OpenXR format and becomes the active provider within Unreal. If your existing interaction system (for example, the VRE plugin) already relies on OpenXR hand-tracking, SenseGlove can function as a direct tracking source instead of a physical hand.

Furthermore, the SenseGlove OpenXR backend allows you to develop and build your own hand interaction system from scratch. This system can operate either via standard OpenXR hand-tracking or with a SenseGlove device interchangeably.

Comparison of Available Approaches

The following table provides an overview and comparison of different hand interaction approaches available within the SenseGlove Unreal Engine Plugin ecosystem when it comes to hand-interaction systems:

	Built-in?	Works out of the box?	Beginner-friendly?	Learnin Curve
SGPawn	✅ Yes	✅ Yes	✅ Most beginer-friendly	✅ Very eas
SenseGlove OpenXR	✅ Yes	❌ Requires Blueprint or C++ coding	✅ Requires a few hours of watching tutotrials	✅ Moderat
VR Expansion Plugin	❌ No	⚠️ Partially – requires setup	❌ Best suited for intermediate or advanced users	⚠️ Steep
Other OpenXR-compatible Plugins	❌ No	❓ Check their documentation	❓ Check their documentation	❓ Check th documentat

Going Beyond SGPawn

In the following sections, we will cover:

- **The Puppeteer (Controller) / Puppet (Pawn) Architecture**: how to customize and control `SGPawn` through events.
- **SGHandTrackerComponent**: how to obtain and consume SenseGlove hand-tracking data, the easy way.
- **SGHapticsComponent**: how to add haptic feedback to your own or third-party hand interaction systems.

We have also covered more in-depth and advanced topics in other parts of this handbook, available in the following sections:

- **OpenXR**: an introduction to OpenXR fundamentals in Unreal Engine.
    - **Consuming FXRHandTrackingState**: explains the data layout of Unreal Engine’s `FXRHandTrackingState`.
        - **Blueprint**: demonstrates how to use `FXRHandTrackingState` data and render a debug hand in Blueprint.
        - **C++**: demonstrates how to use `FXRHandTrackingState` data and render a debug hand in C++.
    - **Third-Party Integrations**: provides a sample Unreal Engine `5.4` project demonstrating how to integrate SenseGlove with the VR Expansion (VRE) Plugin.
    - **Third-Party Tutorials**:— a tutorial series that guides you from beginner to advanced level in using `FXRHandTrackingState` to build your own hand interaction system by animating virtual hand meshes.

SGPawn Events: The Puppeteer (Controller) / Puppet (Pawn) Architecture

The `SGPawn` (SenseGlove Pawn) is intentionally designed as a **data/event-driven puppet**. It detects touch, grab candidates, and hand state, but it does **not make gameplay decisions** on its own. Instead, it delegates the decisions via firing events

Usually these decisions are delegated to the `SGPlayerController` (or your own controller if you want to customize the behaviors), which acts as the **puppeteer** for SGPawn (the **puppet**):

- It registers to `SGPawn` events at the `BeginPlay` event.
- It listens to `SGPawn` events.
- It decides when to grab or release when certain conditions are met.
- It applies gameplay logic.
- It drives haptics or other responses.

That's how `SGPlayerController` works under the hood.

This separation ensures:

- Clean architecture.
- Full and exnsible customization.
- No hidden behavior inside `SGPawn`.
- Deterministic control over interaction rules.

Architecture Overview

```
SGPawn  --->  Emits State Events  --->  SGPlayerController decides what to do
```

`SGPawn`:

- Tracks touch state.
- Tracks grab candidates.
- Tracks grabbed actors.
- Emits events.

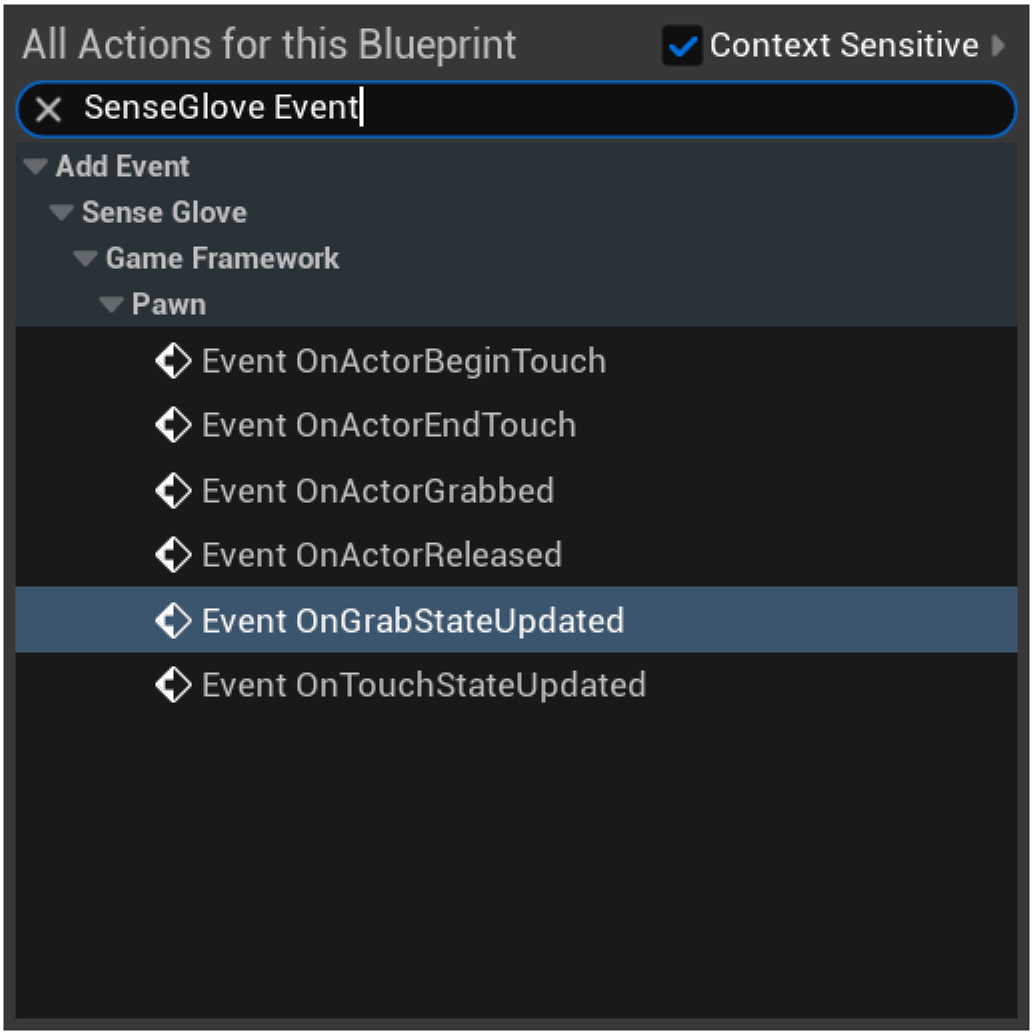
`SGPlayerController`:

- Subscribes to events.
- Calls `Grab()` / `Release()`.
- Applies custom interaction logic.
- Updates haptics.

Exposed Events

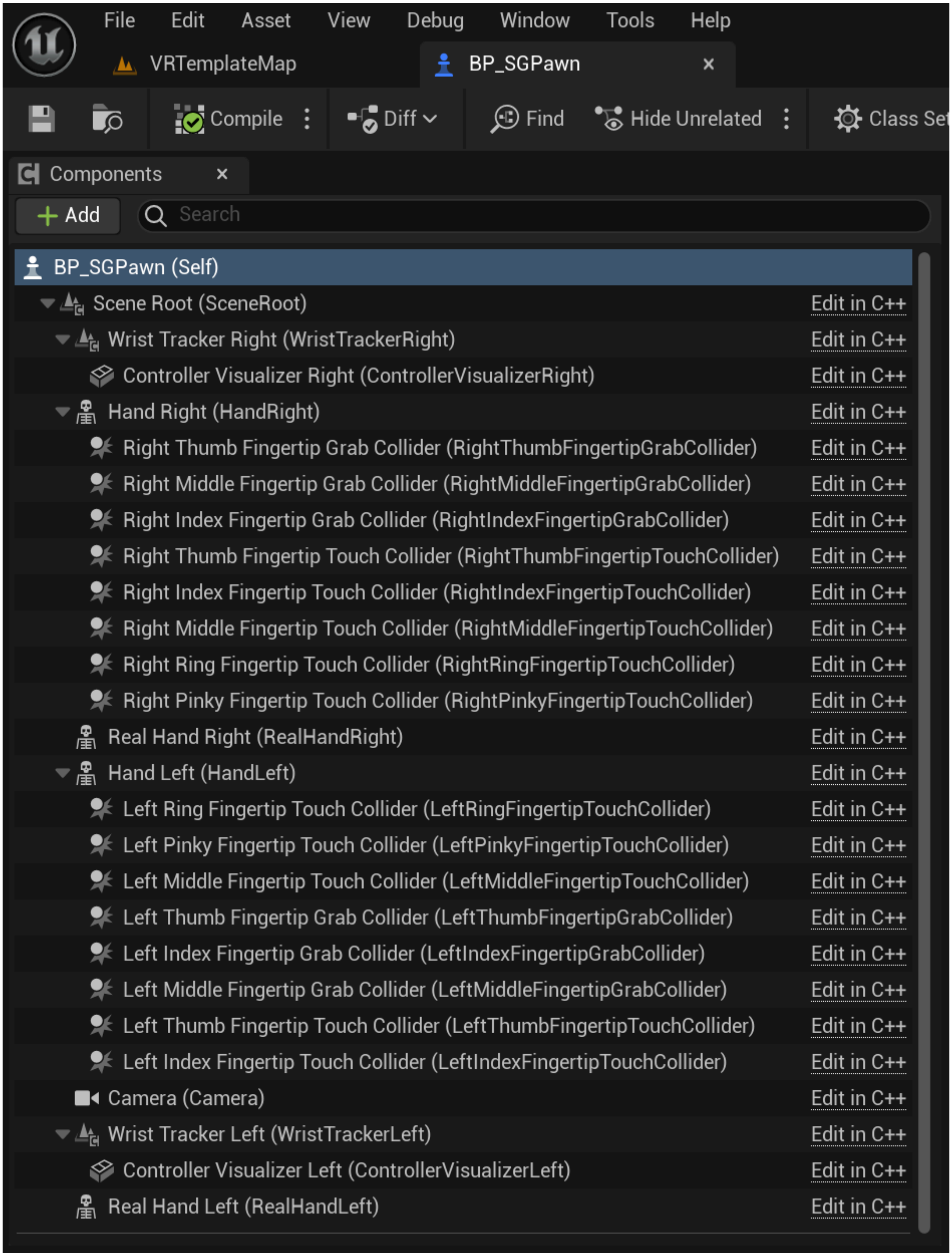
`SGPawn` provides the following event exposed to both C++ and Blueprint:

- `OnGrabStateUpdated`
- `OnTouchStateUpdated`
- `OnActorGrabbed`
- `OnActorReleased`
- `OnActorBeginTouch`
- `OnActorEndTouch`



Important

The current implementation of `SGPawn` relies on `3` grab colliders and `5` touch colliders for grab and touch detection.



BP_SGPawn (Self)
Scene Root (SceneRoot) Edit in C++
Wrist Tracker Right (WristTrackerRight) Edit in C++
Controller Visualizer Right (ControllerVisualizerRight) Edit in C++
Hand Right (HandRight) Edit in C++
Right Thumb Fingertip Grab Collider (RightThumbFingertipGrabCollider) Edit in C++
Right Middle Fingertip Grab Collider (RightMiddleFingertipGrabCollider) Edit in C++
Right Index Fingertip Grab Collider (RightIndexFingertipGrabCollider) Edit in C++
Right Thumb Fingertip Touch Collider (RightThumbFingertipTouchCollider) Edit in C++
Right Index Fingertip Touch Collider (RightIndexFingertipTouchCollider) Edit in C++
Right Middle Fingertip Touch Collider (RightMiddleFingertipTouchCollider) Edit in C++
Right Ring Fingertip Touch Collider (RightRingFingertipTouchCollider) Edit in C++
Right Pinky Fingertip Touch Collider (RightPinkyFingertipTouchCollider) Edit in C++
Real Hand Right (RealHandRight) Edit in C++
Hand Left (HandLeft) Edit in C++
Left Ring Fingertip Touch Collider (LeftRingFingertipTouchCollider) Edit in C++
Left Pinky Fingertip Touch Collider (LeftPinkyFingertipTouchCollider) Edit in C++
Left Middle Fingertip Touch Collider (LeftMiddleFingertipTouchCollider) Edit in C++
Left Thumb Fingertip Grab Collider (LeftThumbFingertipGrabCollider) Edit in C++
Left Index Fingertip Grab Collider (LeftIndexFingertipGrabCollider) Edit in C++
Left Middle Fingertip Grab Collider (LeftMiddleFingertipGrabCollider) Edit in C++
Left Thumb Fingertip Touch Collider (LeftThumbFingertipTouchCollider) Edit in C++
Left Index Fingertip Touch Collider (LeftIndexFingertipTouchCollider) Edit in C++
Camera (Camera) Edit in C++
Wrist Tracker Left (WristTrackerLeft) Edit in C++
Controller Visualizer Left (ControllerVisualizerLeft) Edit in C++
Real Hand Left (RealHandLeft) Edit in C++

On Grab State Updated Event

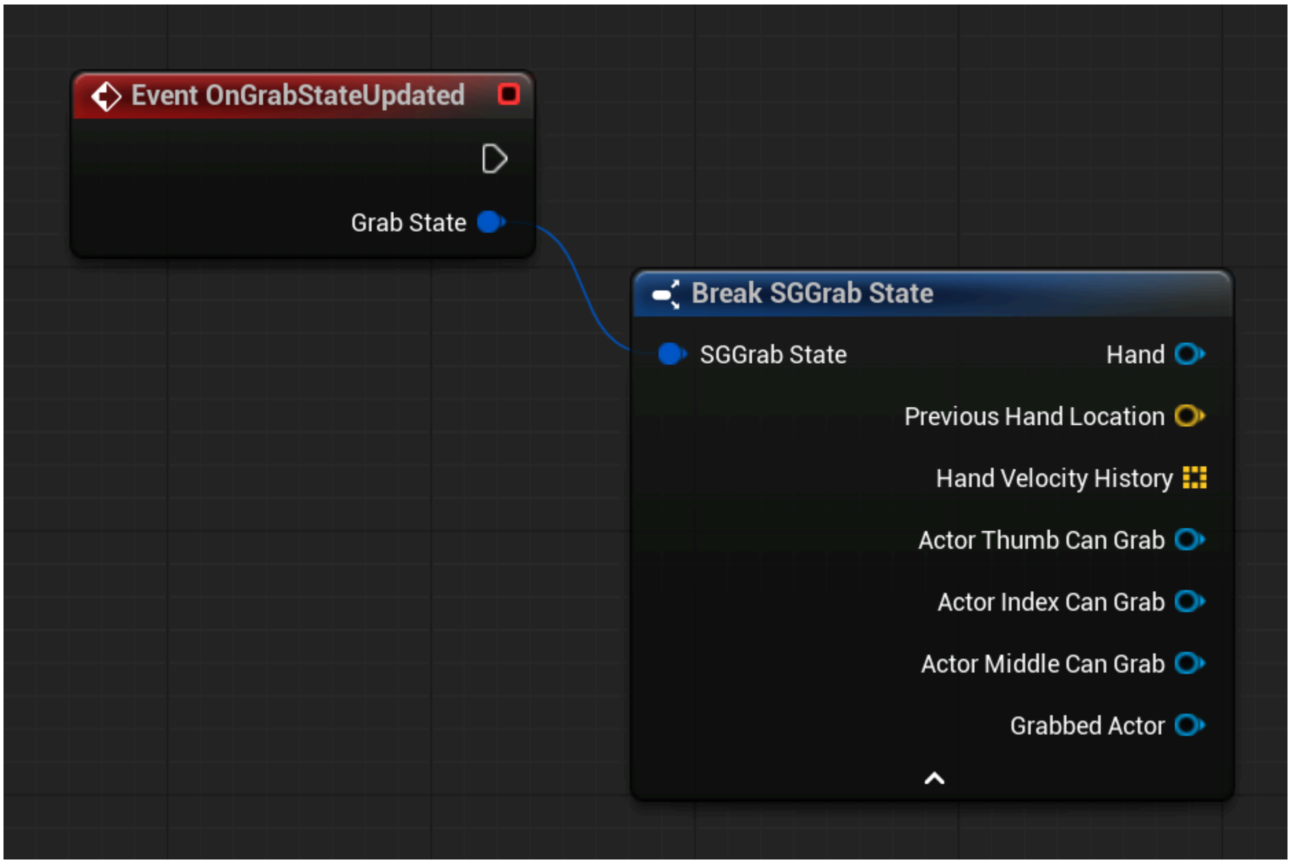
This is the **main decision event** for grabbing logic. The Pawn informs you:

> "Here is the current grab state. You decide what to do."

It is defined in C++ like this:

```
DECLARE_EVENT_OneParam(ASGPawn, FGrabStateUpdatedEvent, const FSGGrabState&
GrabState);
```

In Blueprint, the event appears as shown below:



This event is triggered only when the hand is visible and when any finger on the left or right hand, equipped with a grab collider, begins overlapping (colliding with) or ends overlapping (stops colliding with) an actor that owns an `SGGrabComponent`. So in summary it fires when the following conditions are met:

- The hand is **visible**.
- Any finger (left or right hand) equipped with a **grab collider**:
  - **Begins overlapping** (starts colliding with), or
  - **Ends overlapping** (stops colliding with).
- The overlapped actor owns an `SGGrabComponent`.

Subscribers to this event receive a snapshot of the `FSGGrabState` struct. At the moment the event is fired, the struct contains the following data:

- `Hand`: The `SGVirtualHandComponent` whose grab state was updated due to a finger beginning or ending an overlap with another actor.
- `PreviousHandLocation`: `SGPawn` continuously records hand movement every engine tick. This field stores the hand's location from the previous tick. It can be used to calculate object velocity or apply impulse forces when an object is thrown.
- `HandVelocityHistory`: A history of previous hand locations, up to `SGPawn::MaxNumberOfHandVelocitySamples`. `MaxNumberOfHandVelocitySamples` is a `UPROPERTY` in `SGPawn` that defaults to `10` but can be adjusted as needed.
- `ActorThumbCanGrab`: The actor currently overlapping with the thumb's grab collider. If `null`, the thumb is not overlapping any grabbable actor (which means the actor has an `SGGrabComponent`).
- `ActorIndexCanGrab`: The actor currently overlapping with the index finger's grab collider. If `null`, the index finger is not overlapping any grabbable actor.
- `ActorMiddleCanGrab`: The actor currently overlapping with the middle finger's grab collider. If `null`, the middle finger is not overlapping any grabbable actor.
- `GrabbedActor`: The actor currently being grabbed by this hand. If `null`, the hand is not grabbing anything at that moment.

Here is how the current `SGPlayerController` performs grab detection and instructs the `SGPawn` it controls to execute grab and release actions:

```
void ASGPlayerController::BeginPlay()
{
    Super::BeginPlay();

    ASGPawn* SGPawn{Cast<ASGPawn>(GetPawn())};
    if (!ensureAlwaysMsgf(IsValid(SGPawn), TEXT("%s"), TEXT("ERROR: invalid
SenseGlove pawn!")))
    {
        return;
    }

    SGPawn->OnGrabStateUpdated().AddWeakLambda(
        this, [= SG_CAPTURE_THIS](const FSGGrabState& GrabState) -> void
        {
            if (!IsValid(SGPawn))
            {
                return;
            }

            if (!IsValid(GrabState.Hand))
            {
                return;
            }

            const bool bHandVisible = GrabState.Hand->IsVisible();
            if (!bHandVisible)
            {
                if (SGPawn->IsGrabbing(GrabState.Hand))
                {
                    SGPawn->Release(GrabState.Hand);
                }
                return;
            }

            if (SGPawn->IsGrabbing(GrabState.Hand))
            {
                if (!IsValid(GrabState.ActorThumbCanGrab) ||
                    (GrabState.ActorIndexCanGrab !=
GrabState.ActorThumbCanGrab
                        && GrabState.ActorMiddleCanGrab !=
GrabState.ActorThumbCanGrab))
                {
                    SGPawn->Release(GrabState.Hand);
                }
            }
            else
            {
                if (SGPawn->CanGrab(GrabState.Hand,
GrabState.ActorThumbCanGrab))
                {
                    SGPawn->Grab(GrabState.Hand,
GrabState.ActorThumbCanGrab);
                }
            }
        });
}
```

In this implementation, the `SGPlayerController` listens for grab state updates and determines whether the hand should grab or release an actor based on visibility and finger overlap conditions.

Tip

Haptic feedback is also handled and enforced through `SGPlayerController`. You can review the full implementation in the plguin source code for `SGPlayerController`.

In general, with the current version of the plugin, you can integrate haptic feedback into your own hand interaction system in several ways:

- The `SGHapticsComponent` high-level approach.
- The SenseGlove C++ API:
  - Via the SGHandLayer API.
  - Via the SGHpaticGlove API.
- The SenseGlove Blueprint API:
  - Via the SGHandLayer API which provides a higher-level abstraction compared to the `SGHapticGlove` API.
  - Via the SGHapticGlove API, which offers a lower-level interface than the `SGHandLayer` API and requires some boilerplate code to safely obtain an instance of the desired glove (see Safe and Reliable Glove Access in Blueprint).
- Additionally, there is the `SGTouchComponent`, which provides simplified and limited functionality. On its own, it cannot trigger haptics. It is designed to work in conjunction with the stock `SGPlayerController` shipped with the SenseGlove Unreal Engine plugin.

Touch State Updated Event

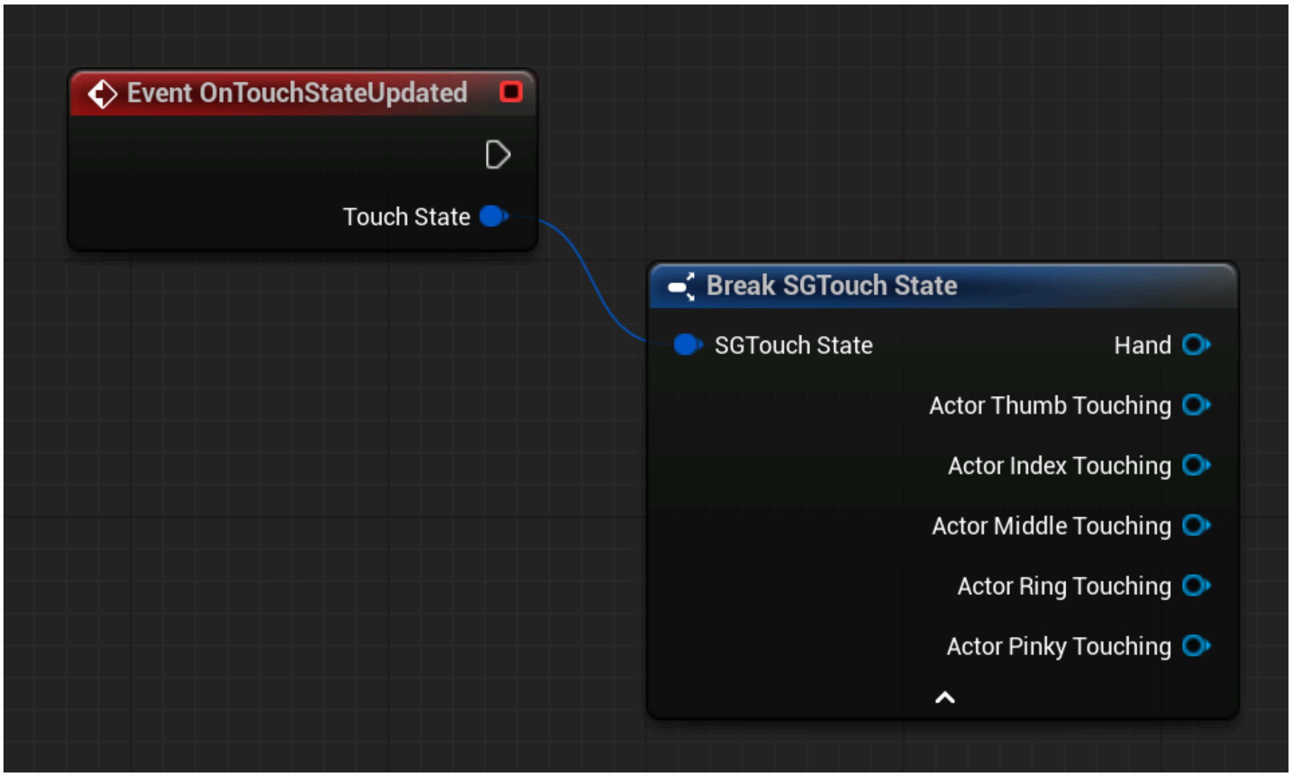
This is the **main decision event** for controlling the touch logic. The Pawn informs you:

> "Here is the current touch state. You decide what to do."

It is defined in C++ like this:

```
DECLARE_EVENT_OneParam(ASGPawn, FTouchStateUpdatedEvent, const FSGTouchState&
TouchState);
```

In Blueprint, the event appears as shown below:



This event is triggered only when the hand is visible and when any finger on the left or right hand, equipped with a grab collider, begins overlapping (colliding with) or ends overlapping (stops colliding with) an actor that owns an `SGGrabComponent`. So in summary it fires when the following conditions are met:

- The hand is **visible**.
- Any finger (left or right hand) equipped with a **touch collider**:
    - **Begins overlapping** (starts colliding with), or
    - **Ends overlapping** (stops colliding with).
- The overlapped actor owns an `SGTouchComponent`.

Subscribers to this event receive a snapshot of the `FSGTouchState` struct. At the moment the event is fired, the struct contains the following data:

- `Hand`: The `SGVirtualHandComponent` whose touch state was updated due to a finger beginning or ending an overlap with another actor.
- `ActorThumbTouching`: The actor currently overlapping with the thumb's touch collider. If `null`, the thumb is not overlapping any touchable actor (which means the actor has an `SGTouchComponent`).
- `ActorIndexTouching`: The actor currently overlapping with the index's touch collider. If `null`, the index is not overlapping any touchable actor.
- `ActorMiddleTouching`: The actor currently overlapping with the middle's touch collider. If `null`, the middle is not overlapping any touchable actor.
- `ActorRingTouching`: The actor currently overlapping with the ring's touch collider. If `null`, the ring is not overlapping any touchable actor.
- `ActorPinkyTouching`: The actor currently overlapping with the pinky's touch collider. If `null`, the pinky is not overlapping any touchable actor.

Here is how the current `SGPlayerController` performs touch detection and instructs the `SGPawn` it controls to apply haptics feedback:

```
void ASGPlayerController::BeginPlay()
{
    Super::BeginPlay();

    ASGPawn* SGPawn{Cast<ASGPawn>(GetPawn())};
    if (!ensureAlwaysMsgf(IsValid(SGPawn), TEXT("%s"), TEXT("ERROR: invalid
SenseGlove pawn!")))
    {
        return;
    }

    SGPawn->OnTouchStateUpdated().AddWeakLambda(
        this, [= SG_CAPTURE_THIS](const FSGTouchState& TouchState) -> void
        {
            if (!IsValid(SGPawn))
            {
                return;
            }

            if (!IsValid(TouchState.Hand))
            {
                return;
            }

            const bool bHandVisible = TouchState.Hand->IsVisible();
            if (!bHandVisible)
            {
                return;
            }

            const bool bGloveConnected = TouchState.Hand->IsGloveConnected();
            if (!bGloveConnected)
            {
                return;
            }

            Pimpl->UpdateHapticsFeedback(TouchState);
        });
}
```

In this implementation, the `SGPlayerController` listens for touch state updates and determines whether the haptic feedbacks should be applied to the glove on that hand, or not. This decision is determined based on various conditions such as hand visibility and finger overlap conditions. Since each fingers haptic feedback application and the type of haptic feedback is decided individually, for the sake of readability the logic has been offloaded to an `SGPlayerController` 's internal function `Pimpl->UpdateHapticsFeedback()` . For example, it applies vibrotactile feedback to eligible fingers like this:

In this implementation, the `SGPlayerController` listens for touch state updates and determines whether haptic feedback should be applied to the glove on that hand. This decision is based on several conditions, such as hand visibility and finger overlap states. Since each finger's haptic feedback and feedback type are evaluated individually, the detailed logic has been offloaded to the internal `SGPlayerController` function `Pimpl->UpdateHapticsFeedback()` for readability and separation of concerns.

For example, vibrotactile feedback is applied to eligible fingers as follows:

```
void ASGPlayerController::FImpl::UpdateHapticsFeedback(const FSGTouchState&
TouchState)
{
    if (!IsValid(TouchState.Hand))
    {
        return;
    }

    USGHapticGlove* Glove{TouchState.Hand->GetConnectedGlove()};
    if (!IsValid(Glove))
    {
        return;
    }

    const bool bGloveConnected = Glove->IsConnected();
    if (!bGloveConnected)
    {
        return;
    }

    // some omitted code due to irrelevance
    ....

    // Send Vibrotactile to the thumb finger if it's touching an actor...
    if (IsValid(TouchState.ActorThumbTouching))
    {
        USGCustomWaveform*
CustomWaveform(GetCustomWaveform(TouchState.ActorThumbTouching));
        Glove->SendCustomWaveform(CustomWaveform,
ESGHapticLocation::ThumbTip);
    }

    // Send Vibrotactile to the index finger if it's touching an actor...
    if (IsValid(TouchState.ActorIndexTouching))
    {
        USGCustomWaveform*
CustomWaveform(GetCustomWaveform(TouchState.ActorIndexTouching));
        Glove->SendCustomWaveform(CustomWaveform,
ESGHapticLocation::IndexTip);
    }

    // Send Vibrotactile to the middle finger if it's touching an actor...
    if (IsValid(TouchState.ActorMiddleTouching))
    {
        USGCustomWaveform*
CustomWaveform(GetCustomWaveform(TouchState.ActorMiddleTouching));
        Glove->SendCustomWaveform(CustomWaveform,
ESGHapticLocation::MiddleTip);
    }

    // Send Vibrotactile to the ring finger if it's touching an actor...
    if (IsValid(TouchState.ActorRingTouching))
    {
        USGCustomWaveform*
CustomWaveform(GetCustomWaveform(TouchState.ActorRingTouching));
        Glove->SendCustomWaveform(CustomWaveform,
ESGHapticLocation::RingTip);
    }

    // Send Vibrotactile to the pinky finger if it's touching an actor...
    if (IsValid(TouchState.ActorPinkyTouching))
    {
        USGCustomWaveform*
CustomWaveform(GetCustomWaveform(TouchState.ActorPinkyTouching));
        Glove->SendCustomWaveform(CustomWaveform,
ESGHapticLocation::PinkyTip);
    }
}
```

As can be seen from the above code, the `SGCustomWaveform` is constructed via a separate helper function:

```
USGCustomWaveform* ASGPlayerController::FImpl::GetCustomWaveform(const
AActor* Actor)
{
    float Amplitude = 0.0f;
    float Duration = 0.0f;
    float Frequency = 0.0f;

    if (IsValid(Actor))
    {
        const USGTouchComponent*
TouchComponent{USGTouchComponent::GetTouchComponent(Actor)};
        if (IsValid(TouchComponent))
        {
            Amplitude = TouchComponent->GetVibrotactileAmplitude();
            Duration = TouchComponent->GetVibrotactileDuration();
            Frequency = TouchComponent->GetVibrotactileFrequency();
        }
    }

    USGCustomWaveform* CustomWaveform{
        USGCustomWaveform::NewCustomWaveform(Owner, Amplitude, Duration,
Frequency)
    };
    return CustomWaveform;
}
```

When it comes to force-feedback, the controller sends force-feedback to all fingers at once, while still constructing the force-feedback levels array via a separate function. `5` elements for `5` fingers indexed from thumb to pinky, where element `0` corresponds to the thumb, `1` to the index finger, and so on, with `4` representing the pinky; see the `SGTouchComponent` documentation for more details. This is how `UpdateHapticsFeedback()` sends force-feedback to the glove:

```
void ASGPlayerController::FImpl::UpdateHapticsFeedback(const FSGTouchState&
TouchState)
{
    if (!IsValid(TouchState.Hand))
    {
        return;
    }

    USGHapticGlove* Glove{TouchState.Hand->GetConnectedGlove()};
    if (!IsValid(Glove))
    {
        return;
    }

    const bool bGloveConnected = Glove->IsConnected();
    if (!bGloveConnected)
    {
        return;
    }

    // Queue the Force-Feedback command...
    TArray<float> ForceFeedbackLevels{
        GetForceFeedbackLevels(
            TouchState.ActorThumbTouching, TouchState.ActorIndexTouching,
TouchState.ActorMiddleTouching,
            TouchState.ActorRingTouching, TouchState.ActorPinkyTouching)
    };
    Glove->QueueForceFeedbackLevels(MoveTemp(ForceFeedbackLevels));

    // Send the haptics commands!
    Glove->SendHaptics();
}
```

Here is the current implementation for `GetForceFeedbackLevels()` :

```
TArray<float> ASGPlayerController::FImpl::GetForceFeedbackLevels(
    const AActor* ActorThumbTouching,
    const AActor* ActorIndexTouching,
    const AActor* ActorMiddleTouching,
    const AActor* ActorRingTouching,
    const AActor* ActorPinkyTouching)
{
    const float ThumbForceFeedbackLevel =
GetForceFeedbackLevel(ActorThumbTouching);
    const float IndexForceFeedbackLevel =
GetForceFeedbackLevel(ActorIndexTouching);
    const float MiddleForceFeedbackLevel =
GetForceFeedbackLevel(ActorMiddleTouching);
    const float RingForceFeedbackLevel =
GetForceFeedbackLevel(ActorRingTouching);
    const float PinkyForceFeedbackLevel =
GetForceFeedbackLevel(ActorPinkyTouching);

    const TArray<float> ForceFeedbackLevels{
        ThumbForceFeedbackLevel,
        IndexForceFeedbackLevel,
        MiddleForceFeedbackLevel,
        RingForceFeedbackLevel,
        PinkyForceFeedbackLevel,
    };

    return ForceFeedbackLevels;
}
```

Tip

You can review the full implementation in the plguin source code for `SGPlayerController`.

In general, with the current version of the plugin, you can integrate haptic feedback into your own hand interaction system in several ways:

- The `SGHapticsComponent` high-level approach.
- The SenseGlove C++ API:
  - Via the SGHandLayer API.
  - Via the SGHpaticGlove API.
- The SenseGlove Blueprint API: SGHandLayer API which provides a higher-level abstraction compared to the `SGHapticGlove` API.

  - Via the SGHapticGlove API, which offers a lower-level interface than the `SGHandLayer` API and requires some boilerplate code to safely obtain an instance of the desired glove (see Safe and Reliable Glove Access in Blueprint).
- Additionally, there is the `SGTouchComponent`, which provides simplified and limited functionality. On its own, it cannot trigger haptics. It is designed to work in conjunction with the stock `SGPlayerController` shipped with the SenseGlove Unreal Engine plugin.

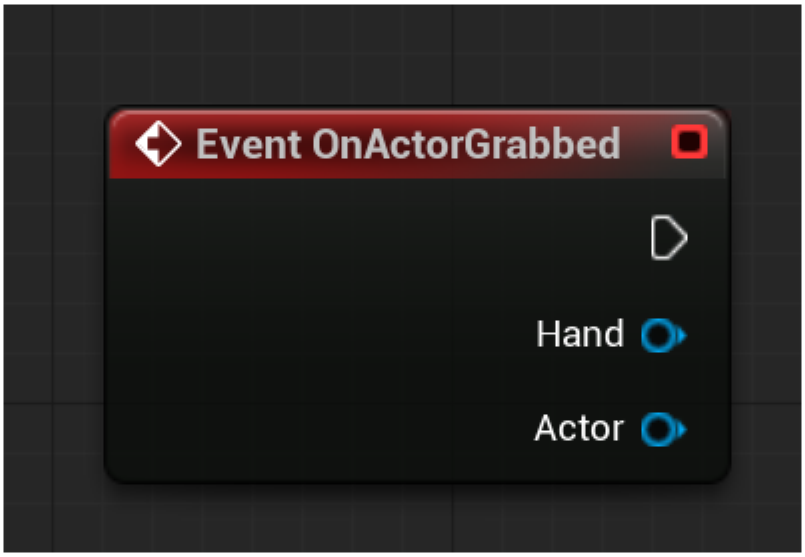
Actor Grabbed Event

This event is triggered whenever a grab is successfully performed by either the left or right hand. Subscribers to this event are notified about **which hand** performed the grab and **which grabbable actor** (an actor that owns an `SGGrabComponent`) was grabbed.

It is defined in C++ like this:

```
DECLARE_EVENT_TwoParams(ASGPawn, FActorGrabbedEvent,
    const USGVirtualHandComponent* Hand,
    const AActor* Actor);
```

In Blueprint, the event appears as shown below:



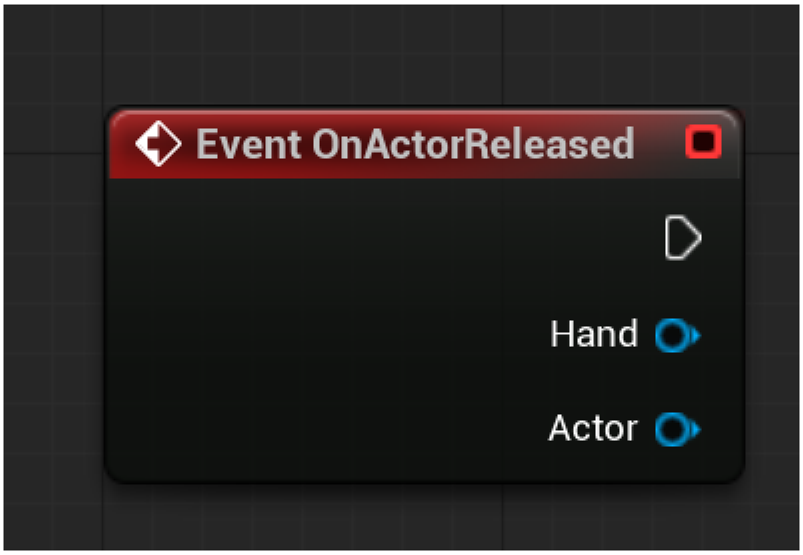
Actor Released Event

This event is triggered whenever a release is successfully performed by either the left or right hand. Subscribers to this event are notified about **which hand** performed the release and **which grabbable actor** (an actor that owns an `SGGrabComponent`) was released.

It is defined in C++ like this:

```
DECLARE_EVENT_TwoParams(ASGPawn, FActorReleasedEvent,
    const USGVirtualHandComponent* Hand,
    const AActor* Actor);
```

In Blueprint, the event appears as shown below:



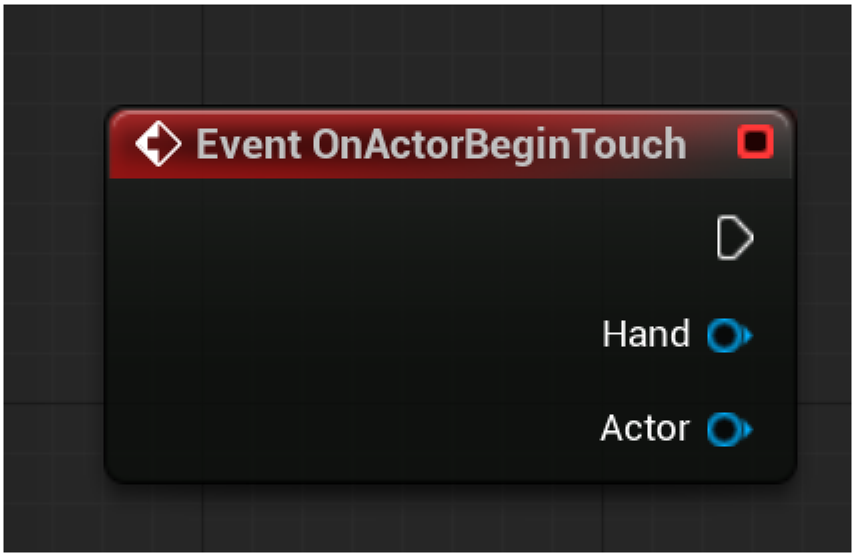
Actor Begin Touch Event

This event is triggered whenever **any finger** on the left or right hand comes into contact with another actor. Subscribers to this event are notified about **which hand** initiated the overlap and **which touchable actor** (an actor that owns an `SGTouchComponent`) was touched.

It is defined in C++ like this:

```
DECLARE_EVENT_TwoParams(ASGPawn, FActorBeginTouchEvent,
    const USGVirtualHandComponent* Hand,
    const AActor* Actor);
```

In Blueprint, the event appears as shown below:



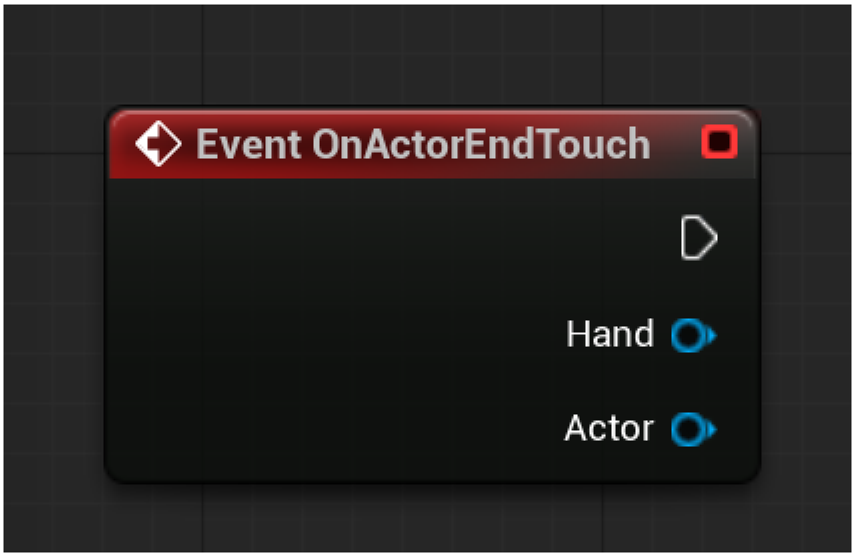
Actor End Touch Event

This event is triggered whenever **any finger** on the left or right hand ends contact with another actor that was previously touched by that finger. Subscribers to this event are notified about **which hand**'s finger ended the overlap and **which touchable actor** (an actor that owns an `SGTouchComponent`) is no longer being touched by that finger.

It is defined in C++ like this:

```
DECLARE_EVENT_TwoParams(ASGPawn, FActorEndTouchEvent,
    const USGVirtualHandComponent* Hand,
    const AActor* Actor);
```

In Blueprint, the event appears as shown below:



SGPawn Grab/Realase-Related Functions

In addition to the events described above, `SGPawn` provides a set of helper functions related to grabbing and releasing actors.

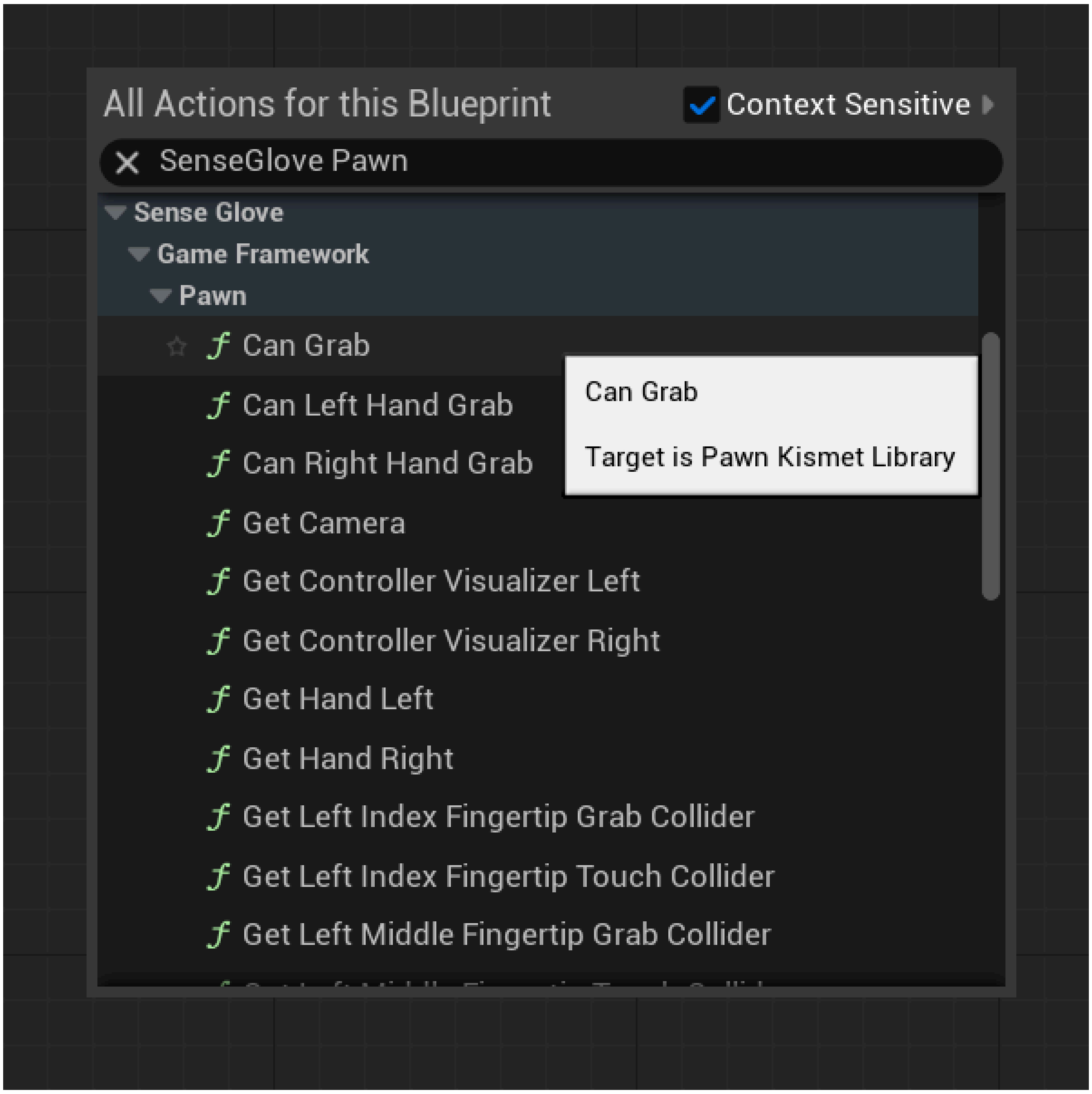
These functions allow you to:

- Check whether a specific hand can grab a given actor.
- Determine whether a hand is currently grabbing an actor.
- Retrieve the currently grabbed actor via an output parameter.
- Trigger grab and release actions programmatically for either hand.

These helper functions are exposed to both C++ and Blueprint:

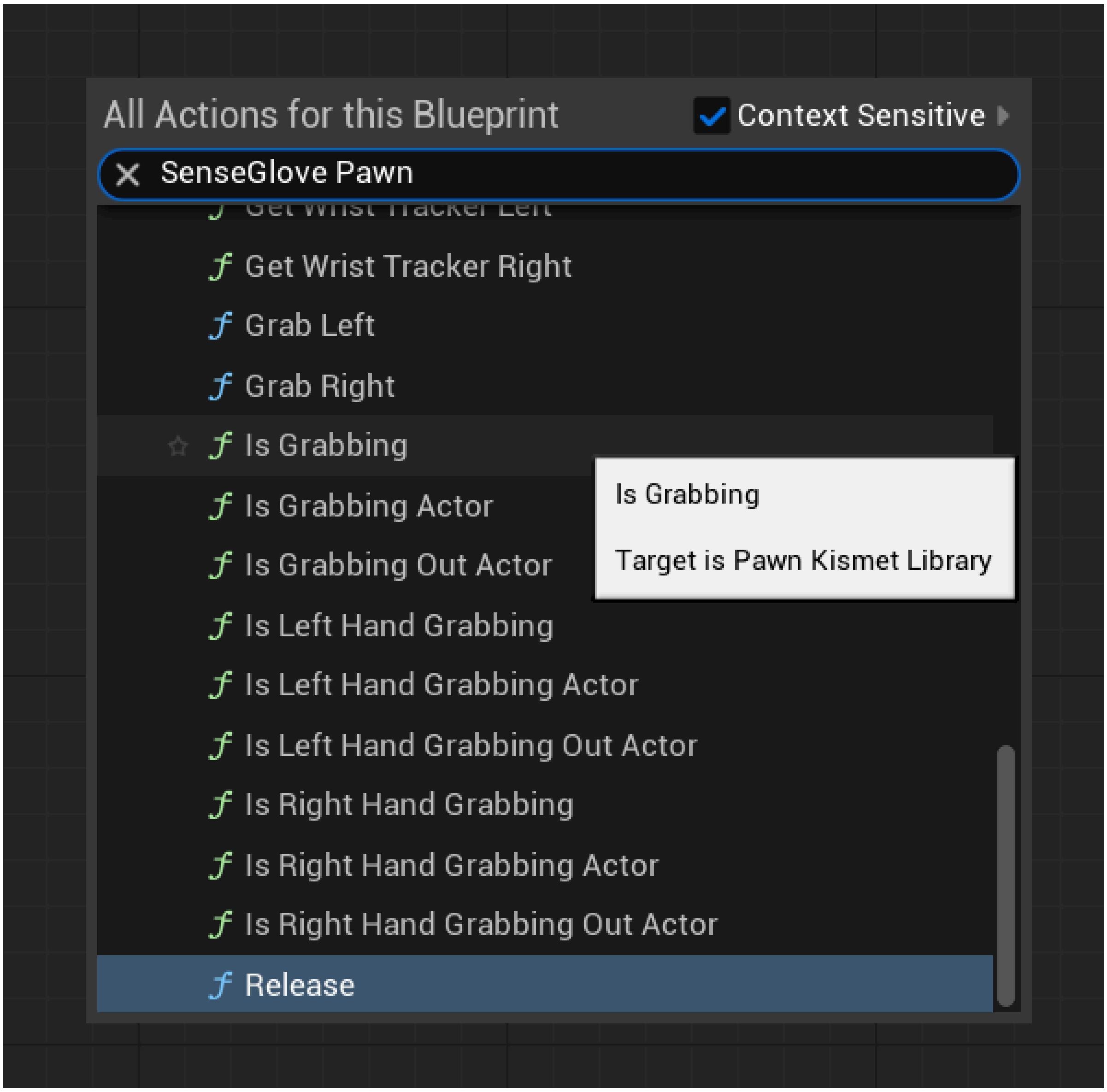
```
public:
    FORCEINLINE bool CanLeftHandGrab(const AActor* Actor) const
    {
        return !IsLeftHandGrabbing() && ((IsValid(Actor) && Actor ==
LeftHandGrabState.ActorThumbCanGrab)
            && (Actor == LeftHandGrabState.ActorIndexCanGrab || Actor ==
LeftHandGrabState.ActorMiddleCanGrab));
    }

    FORCEINLINE bool IsLeftHandGrabbing(const AActor* Actor) const
    {
        return IsValid(Actor) && LeftHandGrabState.GrabbedActor == Actor;
    }

    bool IsLeftHandGrabbing(AActor*& OutActor) const;

    FORCEINLINE bool IsLeftHandGrabbing() const
    {
        return IsValid(LeftHandGrabState.GrabbedActor);
    }

    FORCEINLINE bool CanRightHandGrab(const AActor* Actor) const
    {
        return !IsRightHandGrabbing() && ((IsValid(Actor) && Actor ==
RightHandGrabState.ActorThumbCanGrab)
            && (Actor == RightHandGrabState.ActorIndexCanGrab || Actor ==
RightHandGrabState.ActorMiddleCanGrab));
    }

    FORCEINLINE bool IsRightHandGrabbing(const AActor* Actor) const
    {
        return IsValid(Actor) && RightHandGrabState.GrabbedActor == Actor;
    }

    FORCEINLINE bool IsRightHandGrabbing() const
    {
        return IsValid(RightHandGrabState.GrabbedActor);
    }

    bool IsRightHandGrabbing(AActor*& OutActor) const;

    FORCEINLINE bool CanGrab(const USGVirtualHandComponent* Hand, const
AActor* Actor) const
    {
        return Hand == HandRight ? CanRightHandGrab(Actor) :
CanLeftHandGrab(Actor);
    }
    FORCEINLINE bool IsGrabbing(const USGVirtualHandComponent* Hand, const
AActor* Actor) const
    {
        return Hand == HandRight ? IsRightHandGrabbing(Actor) :
IsLeftHandGrabbing(Actor);
    }

    bool IsGrabbing(const USGVirtualHandComponent* Hand, AActor*& OutActor)
const;

    FORCEINLINE bool IsGrabbing(const USGVirtualHandComponent* Hand) const
    {
        return Hand == HandRight ? IsRightHandGrabbing() :
IsLeftHandGrabbing();
    }

public:
    FORCEINLINE void GrabLeft(AActor* Actor)
    {
        Grab(HandLeft, Actor);
    }

    FORCEINLINE void GrabRight(AActor* Actor)
    {
        Grab(HandRight, Actor);
    }

    void Grab(USGVirtualHandComponent* Hand, AActor* Actor);

    void ReleaseLeft();

    void ReleaseRight();

    FORCEINLINE void Release(const USGVirtualHandComponent* Hand)
    {
        return Hand == HandRight ? ReleaseRight() : ReleaseLeft();
    }
```

The same functions can be searched within the Blueprint Editor or accessed under the `SenseGlove > Game Framework > Pawn` category:


All Actions for this Blueprint
Context Sensitive
SenseGlove Pawn
Sense Glove
Game Framework
Pawn
Can Grab
Can Left Hand Grab
Can Right Hand Grab
Get Camera
Get Controller Visualizer Left
Get Controller Visualizer Right
Get Hand Left
Get Hand Right
Get Left Index Fingertip Grab Collider
Get Left Index Fingertip Touch Collider
Get Left Middle Fingertip Grab Collider

Can Grab

Target is Pawn Kismet Library


All Actions for this Blueprint

Context Sensitive

SenseGlove Pawn

- Get Wrist Tracker Right
- Grab Left
- Grab Right
- Is Grabbing
- Is Grabbing Actor
- Is Grabbing Out Actor
- Is Left Hand Grabbing
- Is Left Hand Grabbing Actor
- Is Left Hand Grabbing Out Actor
- Is Right Hand Grabbing
- Is Right Hand Grabbing Actor
- Is Right Hand Grabbing Out Actor
- Release

Is Grabbing

Target is Pawn Kismet Library

SGHandTrackerComponent

Since `v2.1.0`, the first version to introduce OpenXR support, the **SenseGlove Unreal Engine Plugin** has provided a convenient way to retrieve `FXRHandTrackingState` for SenseGlove devices. This eliminated the need to manually calculate and apply SenseGlove wrist-tracker settings and offsets, or to fetch the `Project Settings > SenseGlove > Tracking Settings > Wrist Tracking Settings` and pass them to `GetWristLocation()` in an additional step, as described in the relevant documentation.

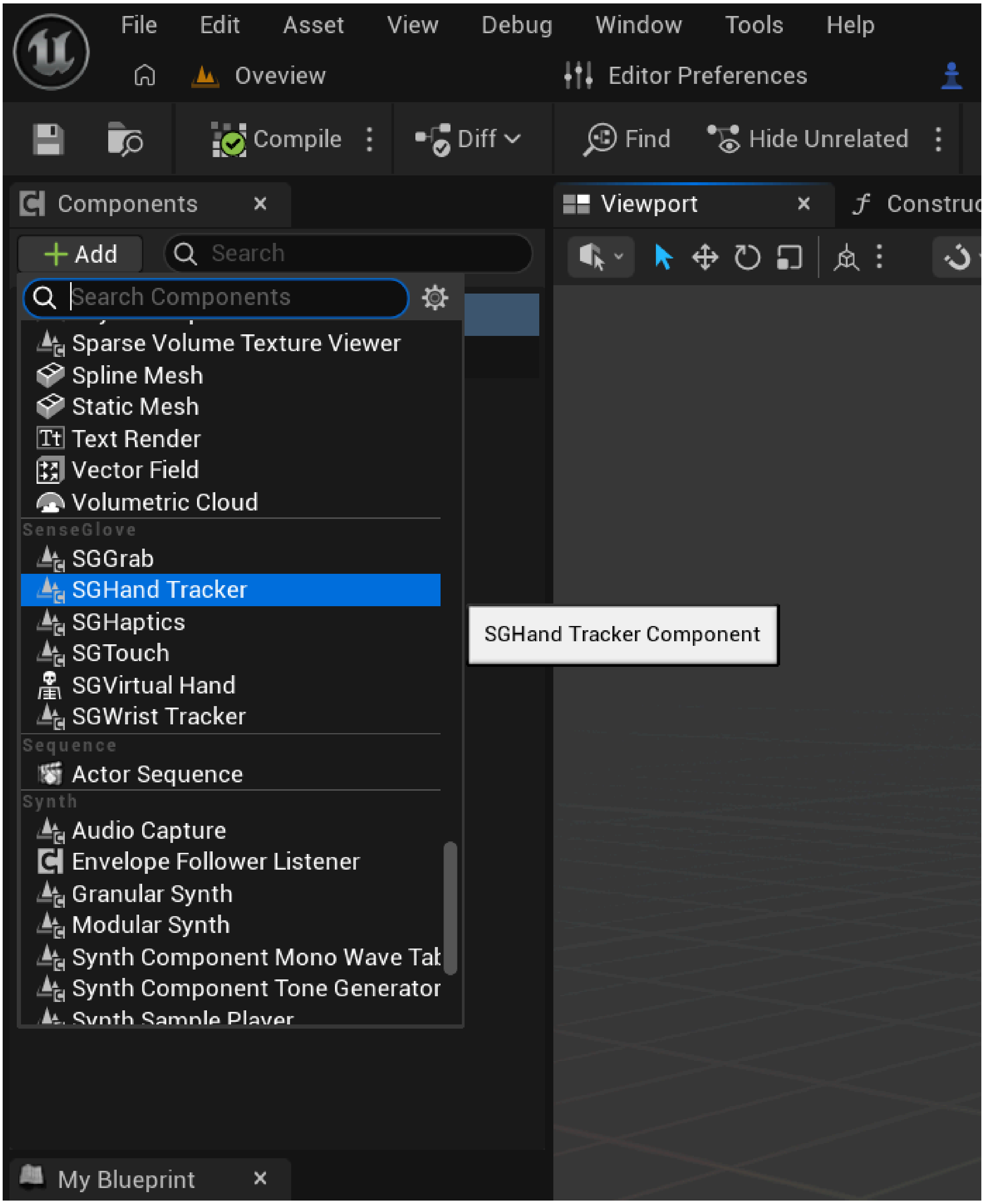
`SGHandTrackerComponent` simplifies this process even further by abstracting all of that away entirely in a high-level manner:

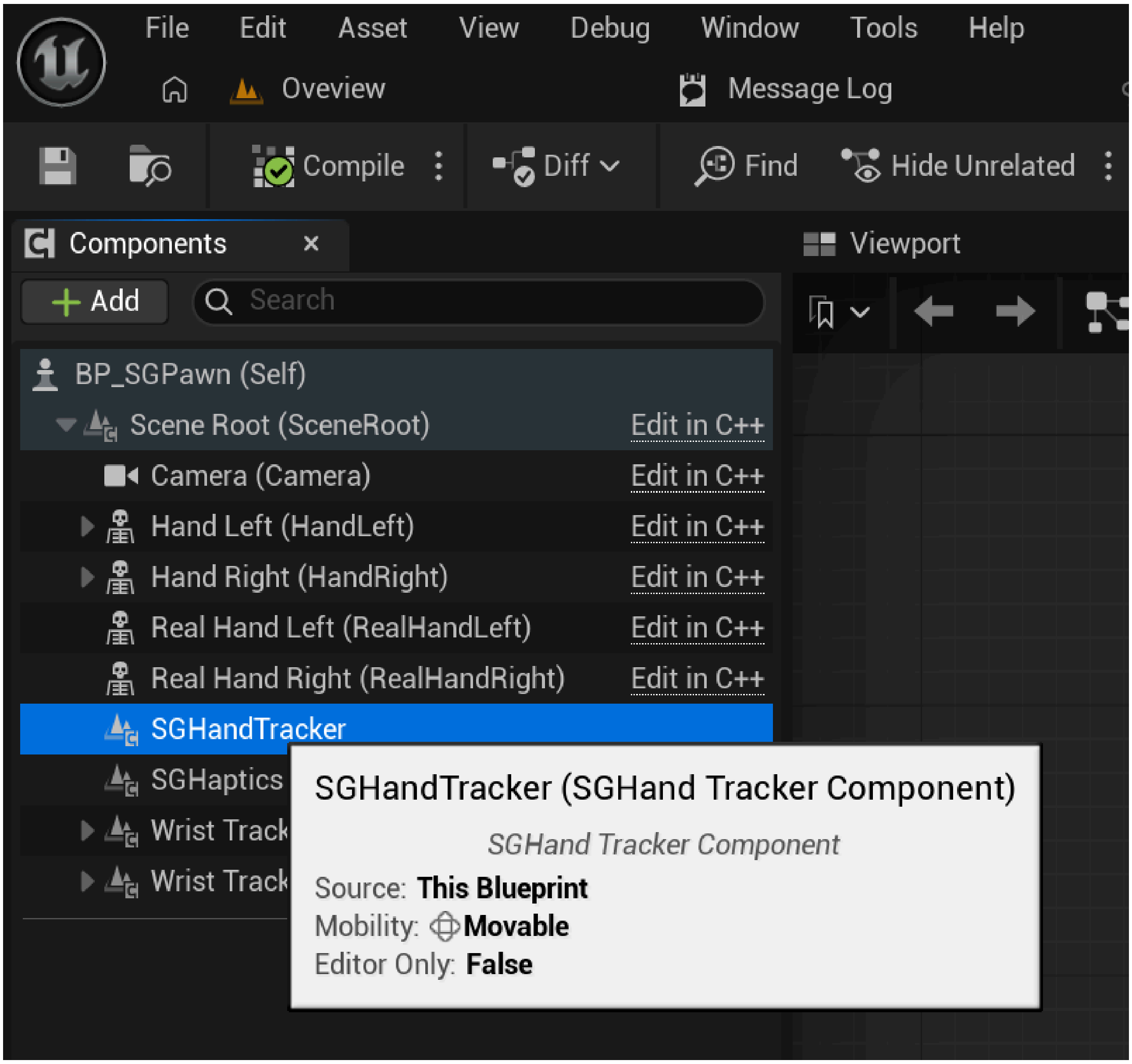
1. Simply add this component to your Pawn class (or any actor that requires hand-tracking data).
2. Configure and adjust its properties.
3. Retrieve the tracking data with a single function call when needed.

The SenseGlove UE Plugin automatically handles all required settings and offset calculations for your positional tracking hardware, regardless of whether you are using pure hand tracking or a SenseGlove device. It also provides an optional debug hand for free, allowing you to visualize the hand-tracking data instantly, without writing a single line of code.

Adding the Component to Your Actors

Adding `SGHandTrackerComponent` is straightforward. In the `Components` panel, click the `Add` button and locate it under the `SenseGlove` section:


File
Edit
Asset
View
Debug
Window
Tools
Help
Oveview
Editor Preferences
Compile
Diff
Find
Hide Unrelated
Components
Viewport
Construc
Add
Search
Search Components
Sparse Volume Texture Viewer
Spline Mesh
Static Mesh
Text Render
Vector Field
Volumetric Cloud
SenseGlove
SGGrab
SGHand Tracker
SGHaptics
SGTouch
SGVirtual Hand
SGWrist Tracker
Sequence
Actor Sequence
Synth
Audio Capture
Envelope Follower Listener
Granular Synth
Modular Synth
Synth Component Mono Wave Tab
Synth Component Tone Generator
Synth Sample Player
SGHand Tracker Component
My Blueprint


File
Edit
Asset
View
Debug
Window
Tools
Help
Oveview
Message Log
Compile
Diff
Find
Hide Unrelated
Components
Viewport
Add
Search
BP_SGPawn (Self)
Scene Root (SceneRoot) Edit in C++
Camera (Camera) Edit in C++
Hand Left (HandLeft) Edit in C++
Hand Right (HandRight) Edit in C++
Real Hand Left (RealHandLeft) Edit in C++
Real Hand Right (RealHandRight) Edit in C++
SGHandTracker
SGHaptics
Wrist Track
Wrist Track

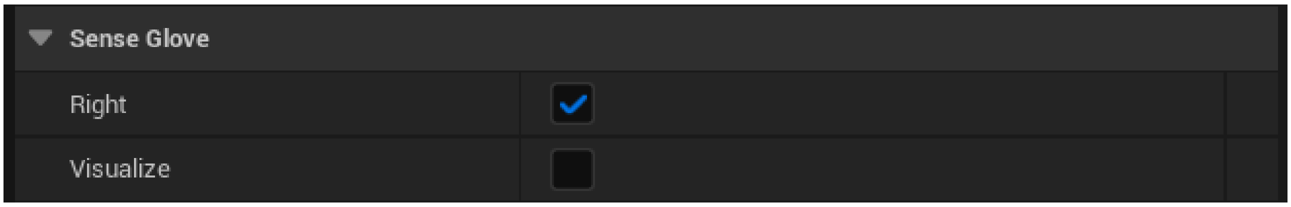
SGHandTracker (SGHand Tracker Component)

SGHand Tracker Component

Source: **This Blueprint**
Mobility: **Movable**
Editor Only: **False**

Blueprint Properties

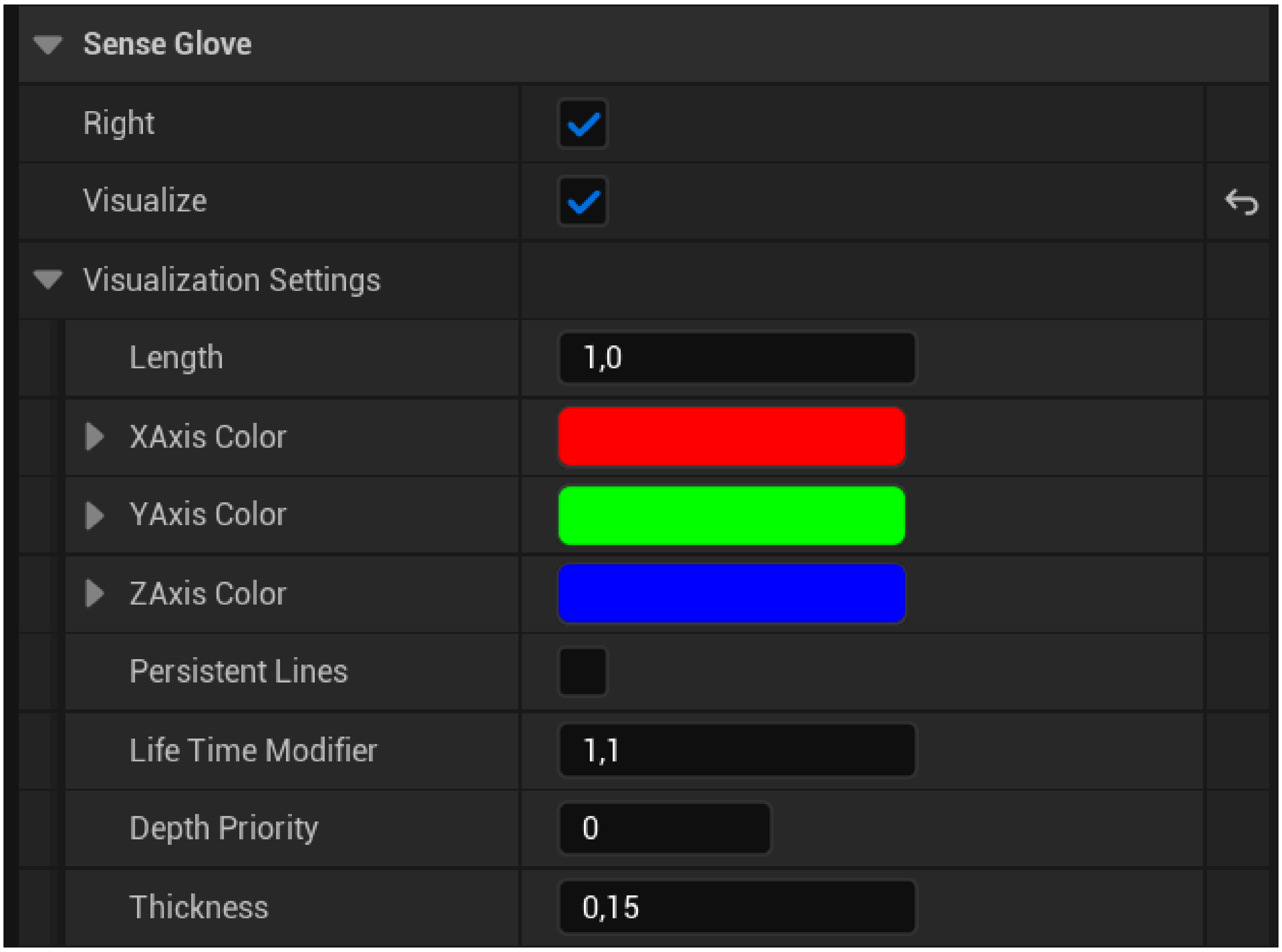
`SGHandTrackerComponent` exposes the following properties through the `Details` panel in Unreal's Blueprint Editor:



Sense Glove

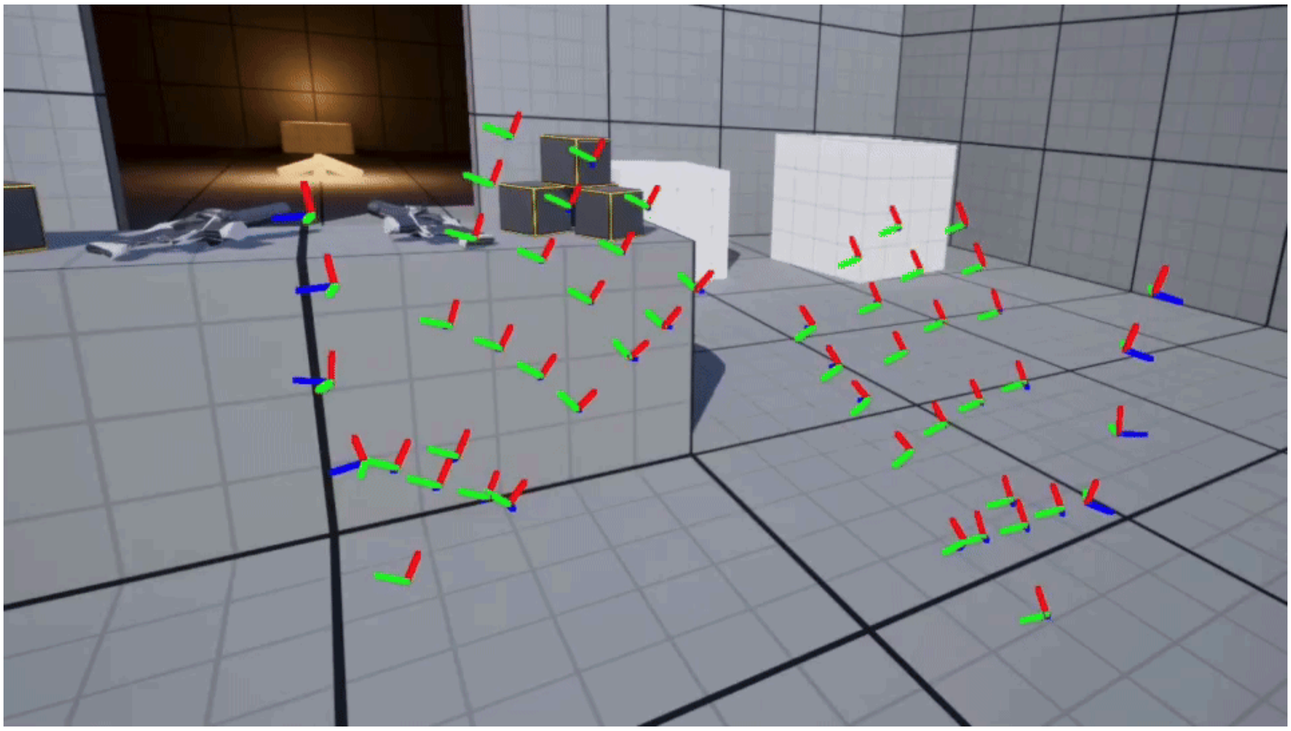
Right	✓
Visualize	☐

- `Right` : If enabled, the component tracks and provides hand-tracking data for the **right** hand. If disabled, it tracks the **left** hand instead.
- `Visualize` : If enabled, the component visualizes the hand-tracking data by rendering a debug hand. The appearance of this debug hand can be further customized, as shown below.



Sense Glove

Right	✓
Visualize	✓
Visualization Settings	
Length	1,0
XAxis Color	🟥
YAxis Color	🟩
ZAxis Color	🟦
Persistent Lines	☐
Life Time Modifier	1,1
Depth Priority	0
Thickness	0,15

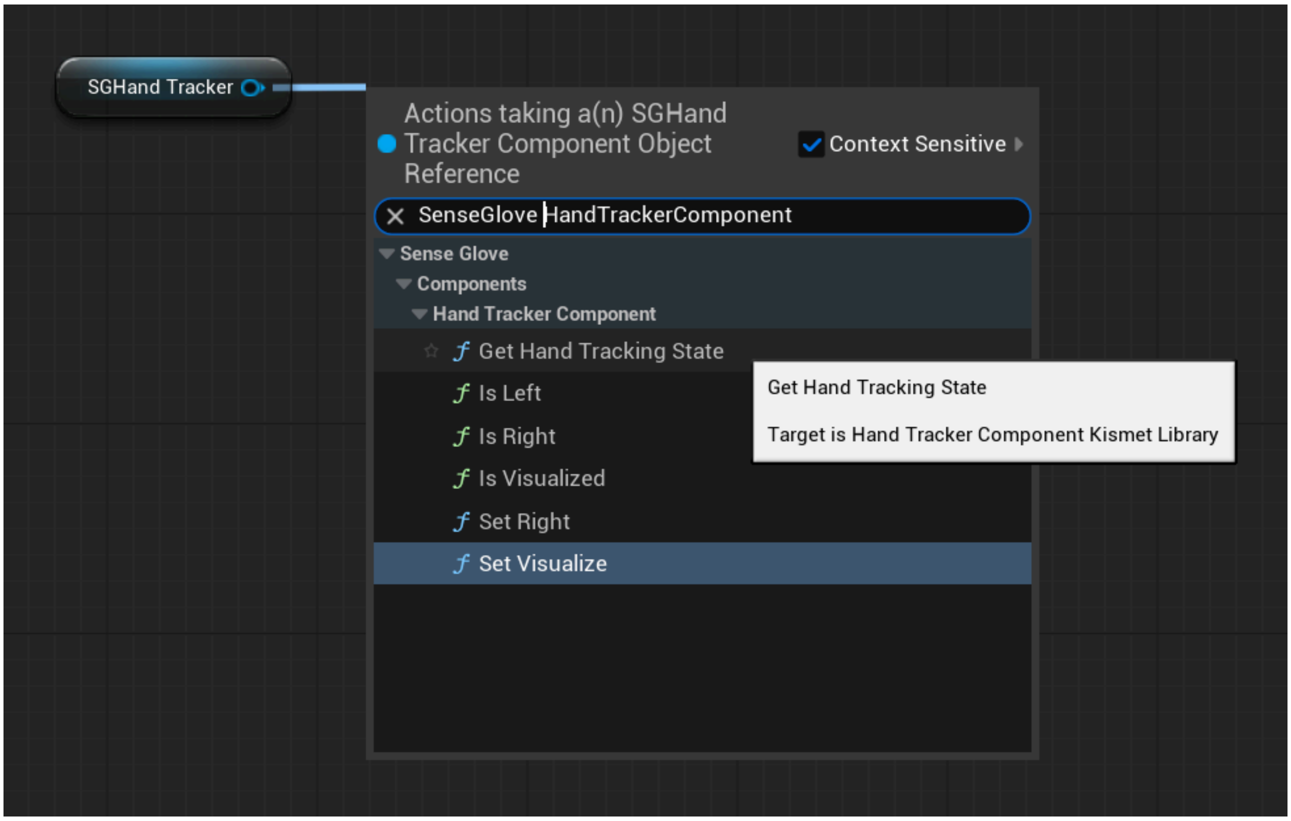


C++ and Blueprint Functions

`SGHandTrackerComponent` provdies the following C++ methods:

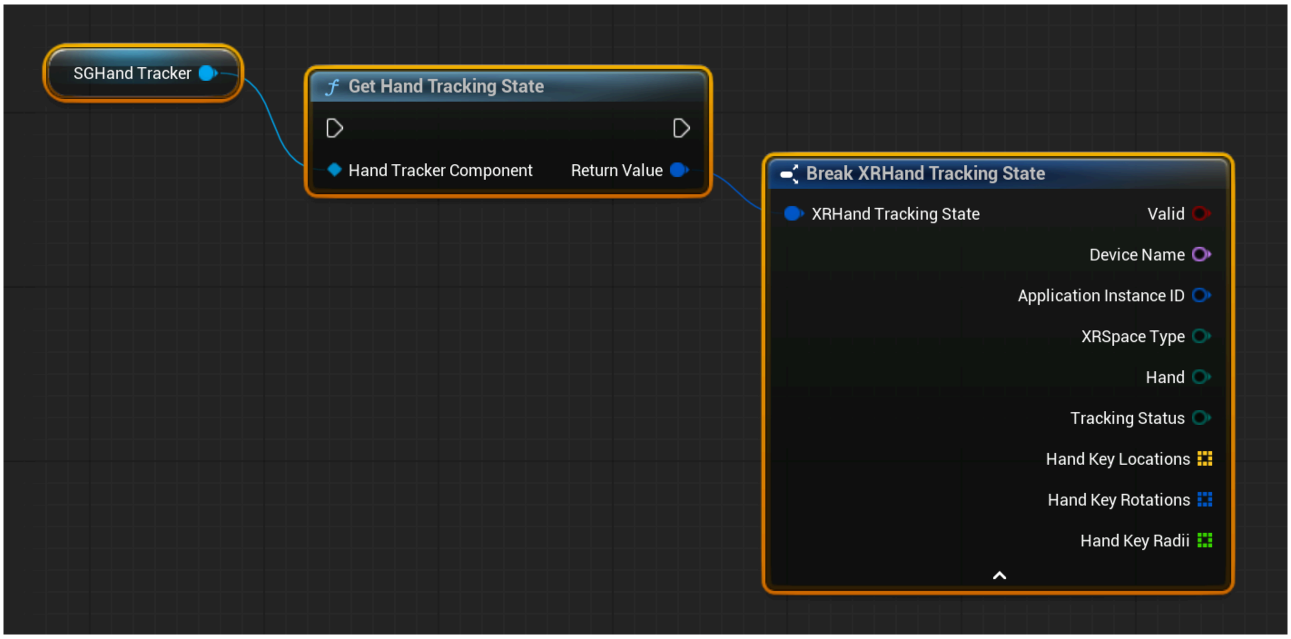
```
public:
    FORCEINLINE bool IsLeft() const
    {
        return !IsRight();
    }

    FORCEINLINE bool IsRight() const
    {
        return bRight;
    }

    void SetRight(const bool bInRight);

    FORCEINLINE bool IsVisualized() const
    {
        return bVisualize;
    }

    FORCEINLINE void SetVisualize(const bool bInVisualize)
    {
        bVisualize = bInVisualize;
    }

public:
    const FXRHandTrackingState& GetHandTrackingState() const;
```

The same set of functions are also exposed to Blueprint:



GetHandTrackingState

The most important function accessible via `SGHandTrackerComponent` is `GetHandTrackingState()` :



This function returns a snapshot of the OpenXR hand-tracking data as an `FXRHandTrackingState` struct.

For more details on what this data contains and how to use it, please refer to the Consuming FXRHandTrackingState section.

SGHapticsComponent

`SGHapticsComponent`, introduced in the SenseGlove Unreal Engine Plugin `v2.8.0`, provides a highly convenient, high-level interface for sending various types of haptic feedback to a SenseGlove device directly from Unreal Engine.

Prior to this release, integrating haptic feedback into a custom hand interaction system was possible in several ways:

- SenseGlove low-level C++ API:
  - Via the SGHandLayer API.
  - Via the SGHpaticGlove API.
- SenseGlove Blueprint API:
  - Via the SGHandLayer API which provides a higher-level abstraction compared to the `SGHapticGlove` API.
  - Via the SGHapticGlove API, which offers a lower-level interface than the `SGHandLayer` API and requires some boilerplate code to safely obtain an instance of the desired glove (see Safe and Reliable Glove Access in Blueprint).
- Additionally, there is the `SGTouchComponent`, which provides simplified and limited functionality. On its own, it cannot trigger haptics. It is designed to work in conjunction with the stock `SGPlayerController` shipped with the SenseGlove Unreal Engine plugin.

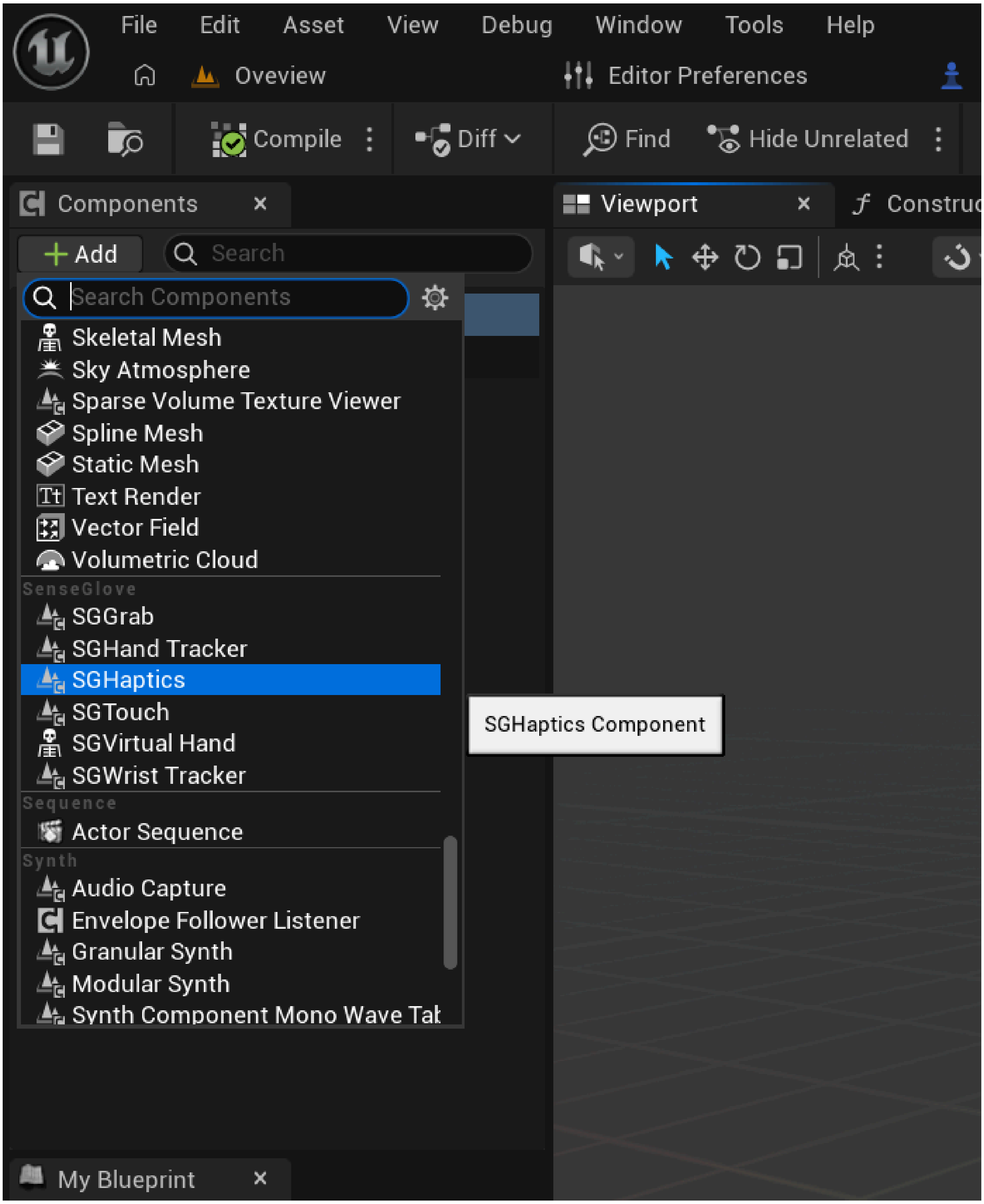
While all of the above approaches remain fully supported, whether in C++ or Blueprint, `SGHapticsComponent` eliminates some of the caveats associated with them, while still giving you full control in a significantly more convenient and streamlined manner.

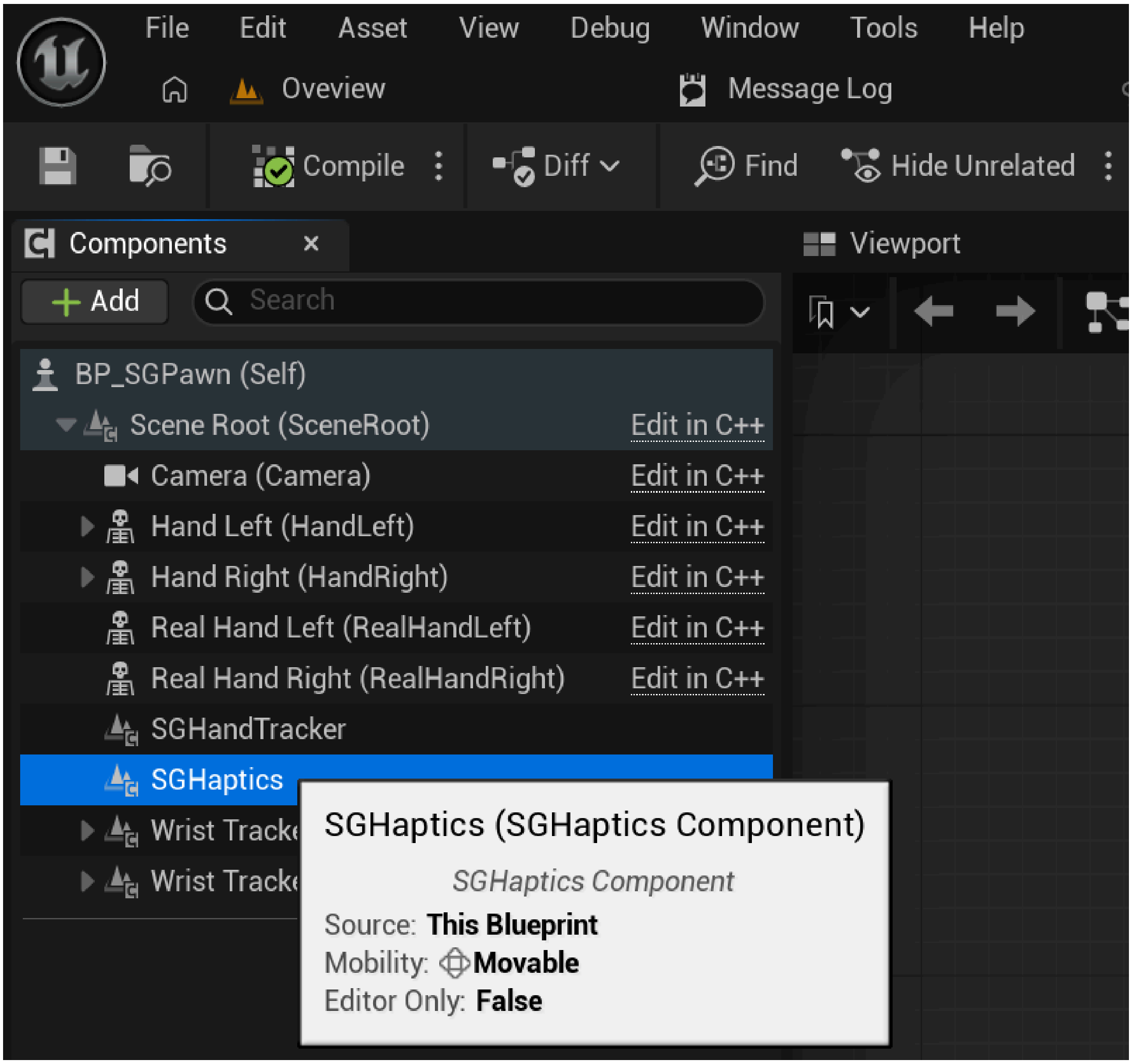
> **Important**
>
> For more detailed information on Nova 2 Glove Vibration Tips & Tricks, please visit the in-depth guide available on SenseGlove Docs.
>
> We strongly recommend reviewing that comprehensive upstream haptics documentation, as this guide focuses solely on applying haptic feedback from

> Unreal Engine.
>
> A solid understanding of the SenseGlove haptics API and its hardware capabilities will help you follow and apply this guide effectively, while also enabling you to troubleshoot haptics-based Unreal Engine projects with confidence.

Adding the Component to Your Actors

Adding `SGHapticsComponent` is straightforward. In the `Components` panel, click the `Add` button and locate it under the `SenseGlove` section:


File
Edit
Asset
View
Debug
Window
Tools
Help
Oveview
Editor Preferences
Compile
Diff
Find
Hide Unrelated
Components
Viewport
Construc
Add
Search
Search Components
Skeletal Mesh
Sky Atmosphere
Sparse Volume Texture Viewer
Spline Mesh
Static Mesh
Text Render
Vector Field
Volumetric Cloud
SenseGlove
SGGrab
SGHand Tracker
SGHaptics
SGTouch
SGVirtual Hand
SGWrist Tracker
Sequence
Actor Sequence
Synth
Audio Capture
Envelope Follower Listener
Granular Synth
Modular Synth
Synth Component Mono Wave Tal
SGHaptics Component
My Blueprint



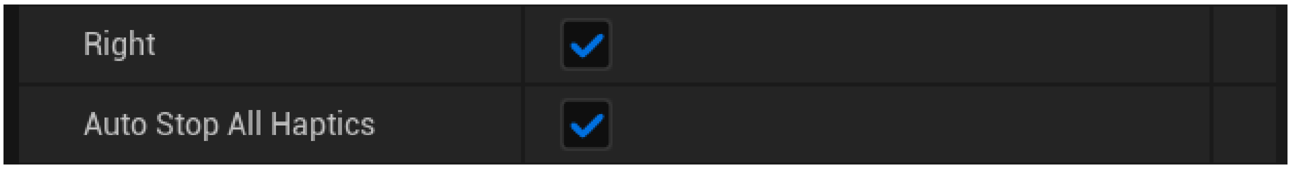
SGHaptics (SGHaptics Component)

SGHaptics Component

Source: **This Blueprint**
Mobility: **Movable**
Editor Only: **False**

Blueprint Properties

`SGHapticsComponent` exposes the following properties through the `Details` panel in Unreal's Blueprint Editor:



Right	✓
Auto Stop All Haptics	✓

- `Right` : If enabled, the component controls haptics feedback for the **right** hand. If disabled, it controls haptics for the **left** hand instead.
- `AutoStopAllHaptics` : If enabled, automatically calls the `StopHaptics()` function when: 1) The component is uninitialized 2) the `EndPlay` event occurs 3) or, the **handedness** changes. This ensures that vibrations won't continue after the simulation ends, or when the active glove it controls, is switched mid-simulation.

C++ and Blueprint Functions

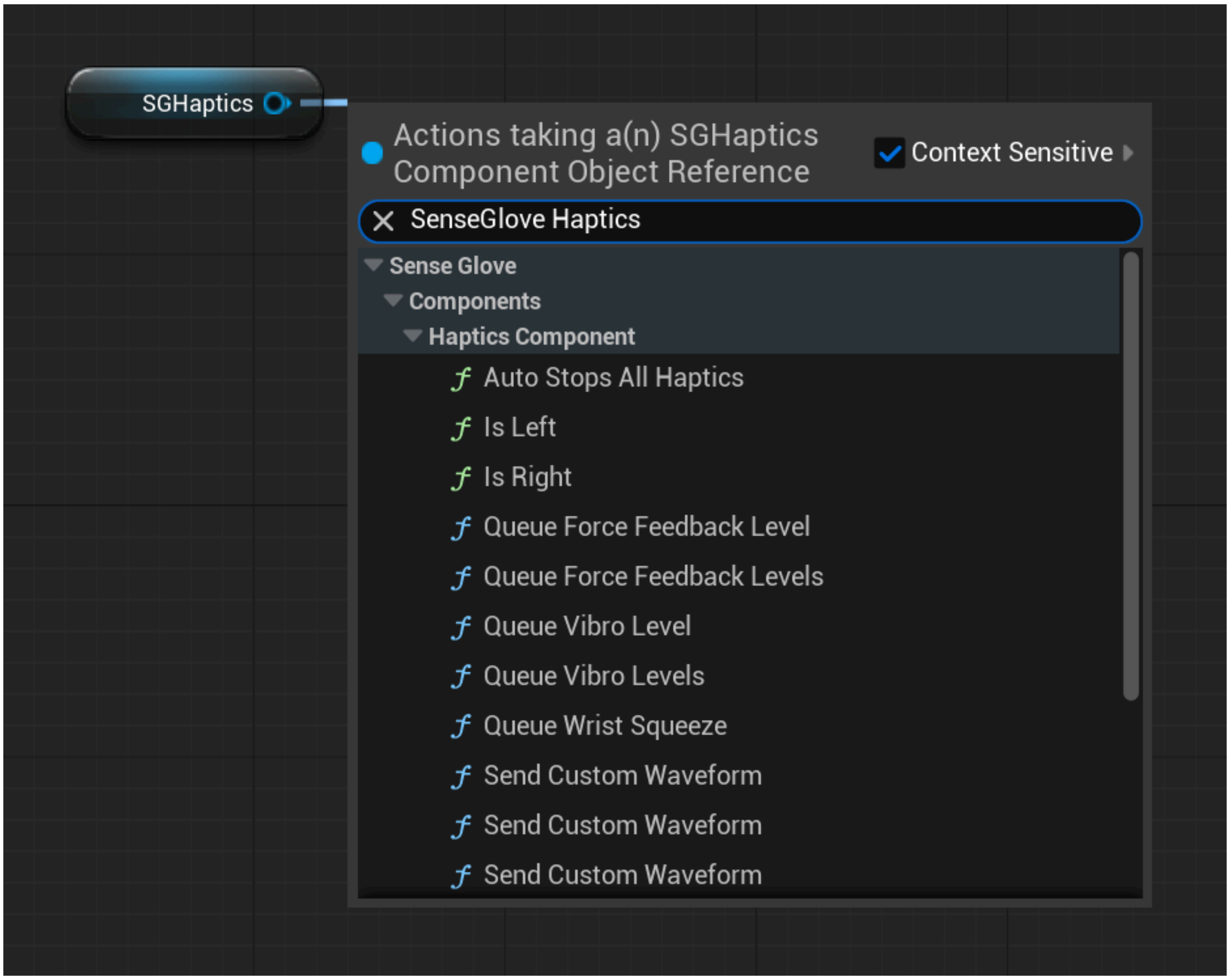
`SGHapticsComponent` provdies the following C++ methods:

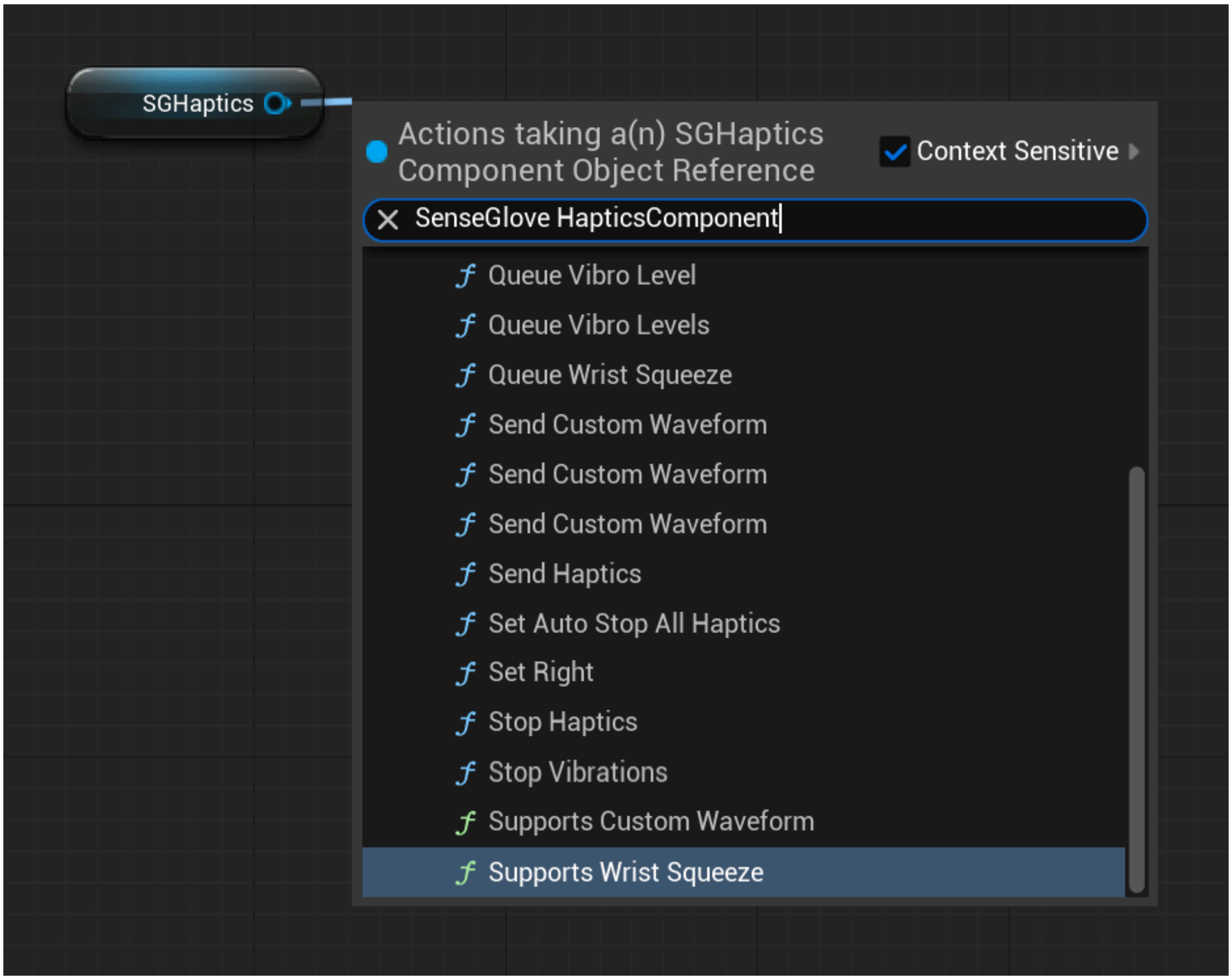
```
public:
    FORCEINLINE bool IsLeft() const
    {
        return !IsRight();
    }

    FORCEINLINE bool IsRight() const
    {
        return bRight;
    }

    void SetRight(const bool bInRight);

    FORCEINLINE bool AutoStopsAllHaptics() const
    {
        return bAutoStopAllHaptics;
    }

    void SetAutoStopAllHaptics(const bool bInAutoStopAllHaptics)
    {
        bAutoStopAllHaptics = bInAutoStopAllHaptics;
    }

public:
    /**
     * Stops all Haptic effects if any are currently playing. Useful at the
end of simulations or when restarting the
     * level.
     */
    void StopHaptics();

    /**
     * Stops only vibrations.
     */
    void StopVibrations();

    /**
     * Take all active commands in the device queue, compile them into one
and send them to the device.
     *
     * @return Returns true if the message was successfully sent to SenseCom.
     */
    bool SendHaptics();

    /**
     * Returns true if the haptic glove supports vibration feedback at the
specified location.
```

```
     *
     * @param AtLocation
     */
    bool SupportsCustomWaveform(ESGHapticLocation AtLocation) const;

    /**
     * Sends a custom waveform to the location specified, provided that the
glove has a motor there, and can support
     * custom waveforms.
     *
     * @param OutWaveform
     * @param Location
     */
    bool SendCustomWaveform(USGCustomWaveform* OutWaveform, ESGHapticLocation
Location);

    /**
     * Sends a custom waveform to the location specified, provided that the
glove has a motor there, and can support
     * custom waveforms.
     *
     * @param Amplitude
     * @param Duration
     * @param Location
     */
    bool SendCustomWaveform(float Amplitude, float Duration,
ESGHapticLocation Location);

    /**
     * Sends a custom waveform to the location specified, provided that the
glove has a motor there, and can support
     * custom waveforms.
     *
     * @param Amplitude
     * @param Duration
     * @param Frequency
     * @param Location
     */
    bool SendCustomWaveform(float Amplitude, float Duration, float Frequency,
ESGHapticLocation Location);

    /**
     * Queue a list of force-feedback levels, between 0.0f and 1.0f. Your
list should be sorted from thumb to pinky.
     *
     * @param Levels01 Array containing the Force-Feedback levels, from 0.0f
(no FFB) to 1.0f. A value < 0.0f will be
     * ignored.
     *
     * @remarks Devices that 'only' have on/off FFB will treat any value >
0.0 as 1.0.
     */
    bool QueueForceFeedbackLevels(const TArray<float>& Levels01);

    /**
     * Set the Force-Feedback value of a particular finger to a specific
level </summary>
     *
     * @param Level01 Value will be clamped between [0...1], where 0.0f means
no Force-Feedback, and 1.0 means full
     * force-feedback.
     * @param Finger The finger to which to send the command.
     */
    bool QueueForceFeedbackLevel(int32 Finger, float Level01);

    /**
     * Queue a list of vibration levels, between 0.0 and 1.0. Your list
should be sorted from thumb to pinky.
     *
     * @param Levels01 Array containing the vibration levels, from 0.0 (no
vibration) to 1.0. A value < 0.0f will be
     * ignored.
     *
     * @remarks Devices that 'only' have on/off FFB will treat any value >
0.0 as 1.0.
     */
    bool QueueVibroLevels(const TArray<float>& Levels01);

    /**
     * Queue a command to set the (continuous) vibration level at a specific
location to a set amplitude.
     *
     * @param Location
     * @param Level01 Value will be clamped between [0...1], where 0.0f means
no vibration, and 1.0 means full
     * vibration.
     */
    bool QueueVibroLevel(ESGHapticLocation Location, float Level01);

    /**
     * Returns true if the chosen glove supports active contact feedback on
the Wrist.
     */
    bool SupportsWristSqueeze() const;

    /**
```

```
	 * Queue a command to set the amount of squeeze level (a.k.a. squeeze-feedback) to the desired level
	 * (0 = no squeeze, 1 = full squeeze) on the wrist, and optionally send it right away.
	 *
	 * @param SqueezeLevel01
	 * @param bSendImmediate
	 */
	bool QueueWristSqueeze(float SqueezeLevel01, bool bSendImmediate);
```

The same set of functions are also exposed to Blueprint:





Quick Blueprint Functions Reference

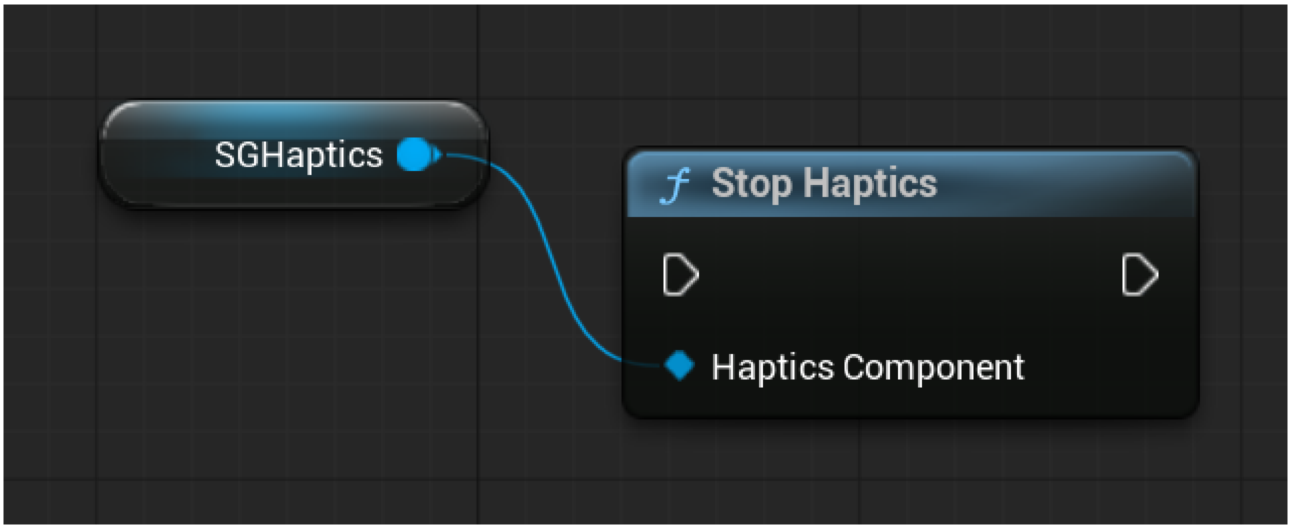
Here is a brief at-a-glance reference of all `SGHapticsComponent` Blueprint functions related to haptic feedback.

Stop Haptics

Stops **all active haptic effects** currently playing on the glove.

This includes:

- Vibrations.
- Force-feedback (FFB).
- Wrist-squeeze.
- Any queued but unsent haptic commands.



Typical Use Cases:

- Resetting the glove at the end of a simulation.
- Restarting a level.
- Emergency stop logic.
- Cleaning up when disabling an actor.

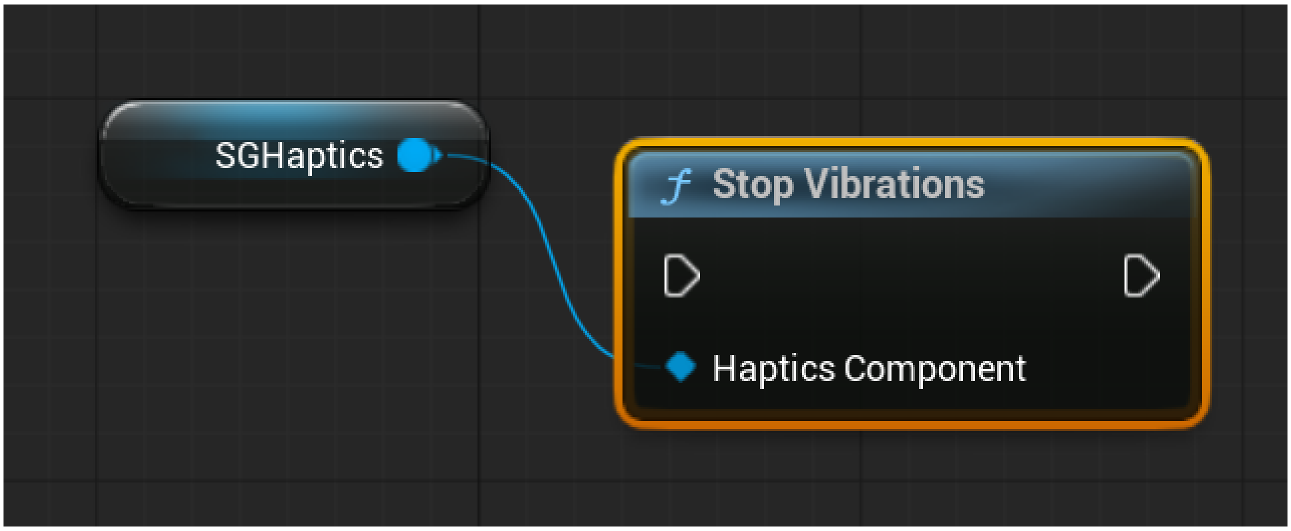
Returns:

This Blueprint node does not return a value.

Stop Vibrations

Stops only **vibration feedback**, without affecting:

- Force-feedback
- Wrist-squeeze



Typical Use Cases:

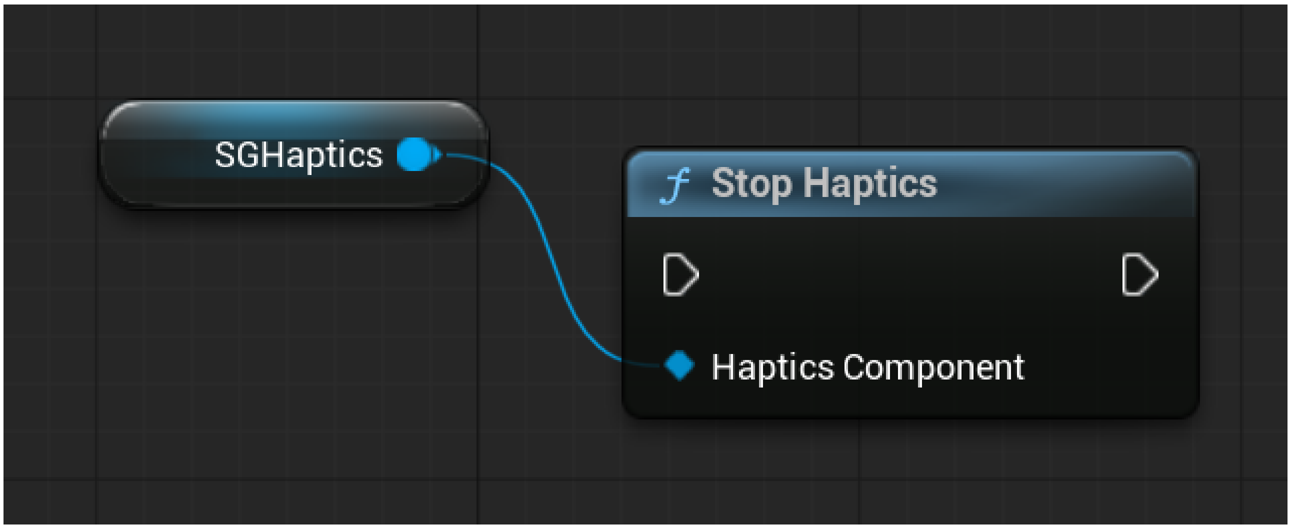
It is useful for example if you want to keep finger resistance active while stopping tactile feedback.

Returns:

This Blueprint node does not return a value.

Send Haptics

Compiles all currently queued haptic commands and sends them to the glove.



The component works using a **queue-based system**:

1. You queue multiple commands (Force-feedback, Vibro, Wrist, etc..)
2. You call **Send Haptics**.
3. Everything is compiled into one device message.

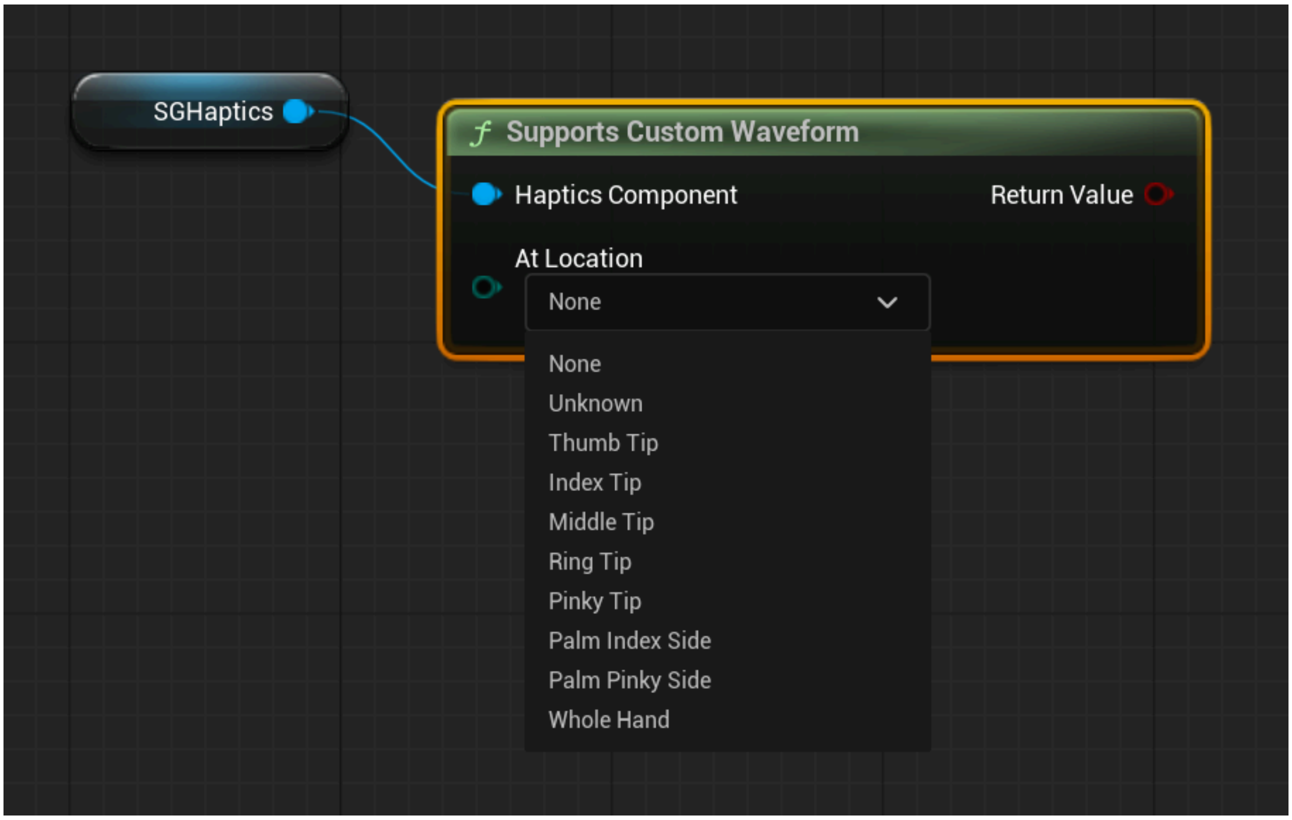
Returns:

- `true` : Indicates message has been successfully sent to SenseCom.
- `false` : Failed to send haptics.

> Caution
>
> Avoid calling `Send Haptics` repeatedly in rapid succession.
>
> For optimal performance, queue all required haptic commands first (Force-Feedback, Vibro, Wrist, etc.), then call `Send Haptics` **once per logical update cycle**.
>
> Continuously queueing commands and flushing them every frame (or multiple times per frame) increases device communication frequency and computational overhead. It may also cause Bluetooth instability and, in extreme cases, lead to the glove disconnecting.
>
> Instead, batch multiple haptic updates together and send them in a single compiled message whenever possible. This reduces processing cost, lowers communication load, and results in more stable and efficient haptic performance.

Supports Custom Waveform

Checks whether the glove supports **custom waveform vibration** at a specific location.



Parameters:

- **At Location:** The vibration location to test (e.g., Thumb Tip, Index Tip, Palm Index Side, etc.).

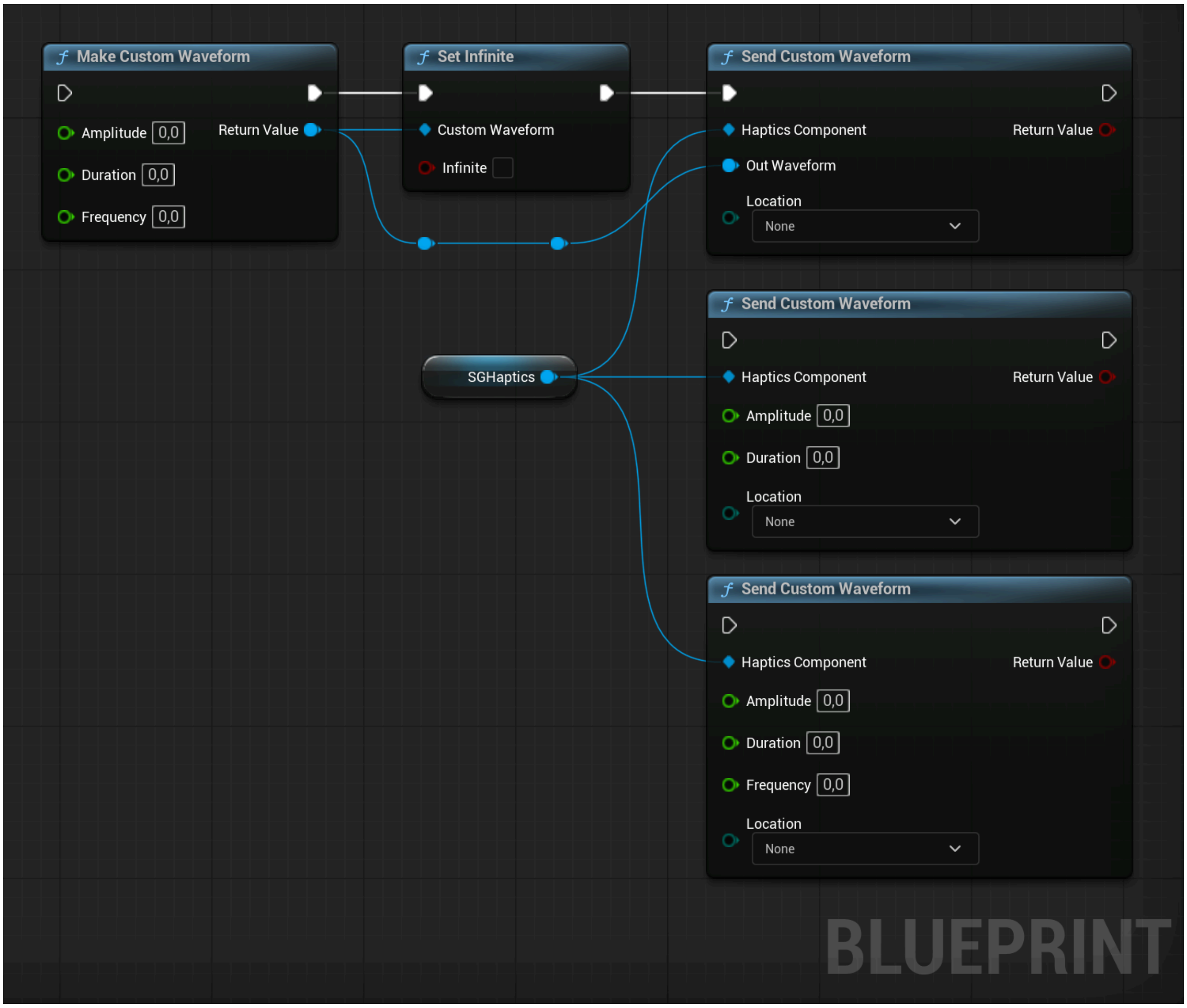
Returns:

- `true` : Custom waveform is supported.
- `false` : Not supported at this location.

You can call this before using `Send Custom Waveform` to see if your glove model at the specified location supports vibration.

Send Custom Waveform

Sends a **custom vibration waveform** to a specific haptic location.



This function has **three overloads** in C++ and is exposed accordingly in Blueprint.

1) Send a Custom Waveform Asset

Parameters:

- **OutWaveform:** A predefined waveform asset that allows you to configure additional custom waveform parameters not available in the other two overloads, giving you more fine-grained control over the vibration's behavior and timing.

Name	Unit	Range	Description
Amplitude		0.0 … 1.0	Vibration intensity
Start Frequency	Hz	10 … 500	Vibration Frequency at the start of the vibration
End Frequency	Hz	10 … 500	Vibration Frequency at the end of the vibration
Attack Time	s	0.0 … 1.0	Time to reach from 0.0 to Amplitude
Sustain Time	s	0.0 … 1.0	Time for which the signal will stay at Amplitude
Decay Time	s	0.0 … 1.0	Time to reach from Amplitude down to 0.0.
Pause Time	s	0.0 … 1.0	Time between each vibration, when repeating the waveform.
Repeat Amount		1 .. 100	How often the waveform is repeated before stopping.
Infinite		True / False	If true, the glove will keep playing this waveform until a new one is played.
Waveform Type	EWaveType	0 .. 5	The shape of the waveform: Sine / Square / SawUp / SawDown / Triangle / Noise.
FrequencySwitchTime*		0.0 … 1.0	At this position in the waveform (0.0 being start, 1.0 being the

Name	Unit	Range	Description
			end), we start multiply the Frequency by FrequencySwitchFactor
FrequencySwitchFactor*		1.0 .. 3.0	How much to multiply the frequency by, after FrequencySwitchTime has passed

- **Location:** Where to play the waveform.

Returns:

- `true` : If command successfully sent.
- `false` : If it fails.

2) Send Amplitude + Duration

Parameters:

- **Amplitude:** Vibration strength (0.0 – 1.0).
- **Duration:** Duration in seconds.
- **Location:** Target haptic location.

Returns:

- `true` : If command successfully sent.
- `false` : If it fails.

3) Send Amplitude + Duration + Frequency

Parameters:

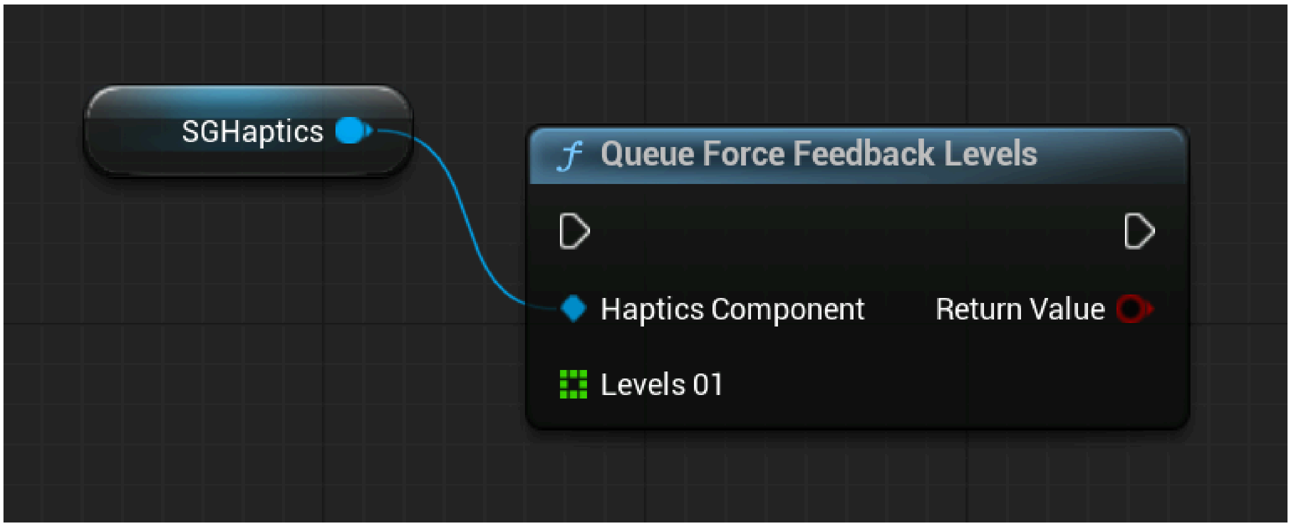
- **Amplitude** — Vibration strength (0.0 – 1.0).
- **Duration** — Duration in seconds.
- **Frequency** — Vibration frequency in Hz.
- **Location** — Target haptic location.

Returns:

- `true` : If command successfully sent.
- `false` : If it fails.

Queue Force Feedback Levels

Queues force-feedback levels for **all fingers** at once.



Parameters:

- **Levels 01:** Array containing the Force-Feedback levels between `0.0` (no FFB) to `1.0` (full FFB); ordered from **Thumb** → **Index** → **Middle** → **Ring** → **Pinky**.

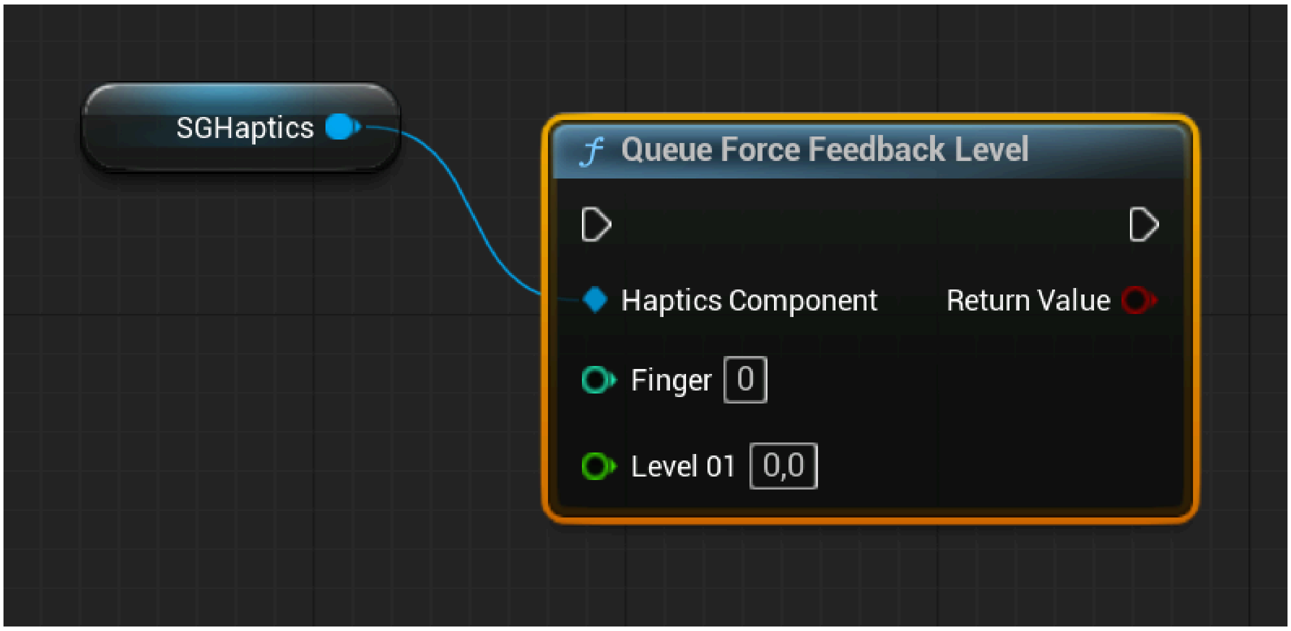
> **Note**
>
> Force-feedback value behavior:
>
> - `0.0` = No resistance.
> - `1.0` = Full resistance.
> - Values `< 0.0` are ignored.
> - Devices that only support on/off FFB treat any value > `0.0` as full force.

Returns:

- `true` : If queued successfully.
- `false` : If it fails.

Queue Force Feedback Level

Queues force-feedback on a **particular finger** to a specific level.



Parameters:

- **Finger:** Index of the finger; indexed from **Thumb** → **Index** → **Middle** → **Ring** → **Pinky**.
- **Level 01:** Value clamped between `0.0` (no FFB) to `1.0` (full FFB).

> ⓘ **Note**
>
> Force-feedback value behavior:
>
> - `0.0` = No resistance.
> - `1.0` = Full resistance.
> - Values `< 0.0` are ignored.
> - Devices that only support on/off FFB treat any value > `0.0` as full force.

Returns:

- `true` : If queued successfully.
- `false` : If it fails.

Queue Vibro Levels

Important

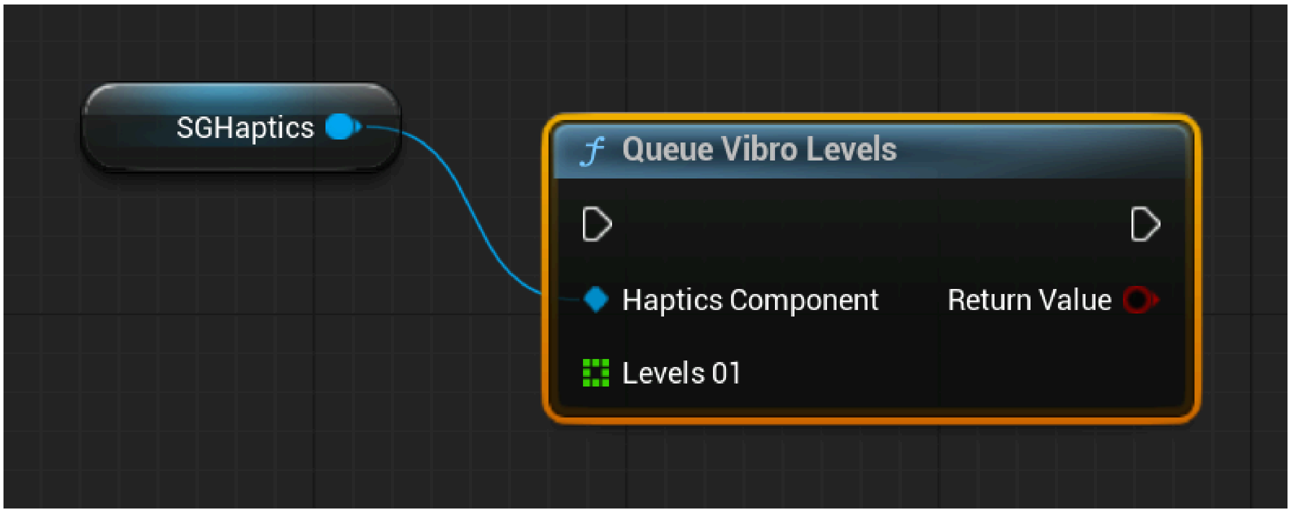
Legacy Function – Use Custom Waveforms Instead

`Queue Vibro Levels` is retained for backward compatibility with older API releases.

Internally, it delegates to `Send Custom Waveform`, which is the recommended method for applying vibrotactile feedback.

For new projects, prefer `Send Custom Waveform`, as it provides more fine-grained control over amplitude, frequency, timing, and waveform shaping.

Queues continuous vibrotactile levels for **all fingers** at once to a set amplitude.



Parameters:

- **Levels 01:** Array containing the vibro levels between `0.0` (no vibration) to `1.0` (full vibration); ordered from **Thumb** → **Index** → **Middle** → **Ring** → **Pinky**.

> **Note**
>
> Force-feedback value behavior:
>
> - `0.0` = No vibration.
> - `1.0` = Full vibration.
> - Values `< 0.0` are ignored.

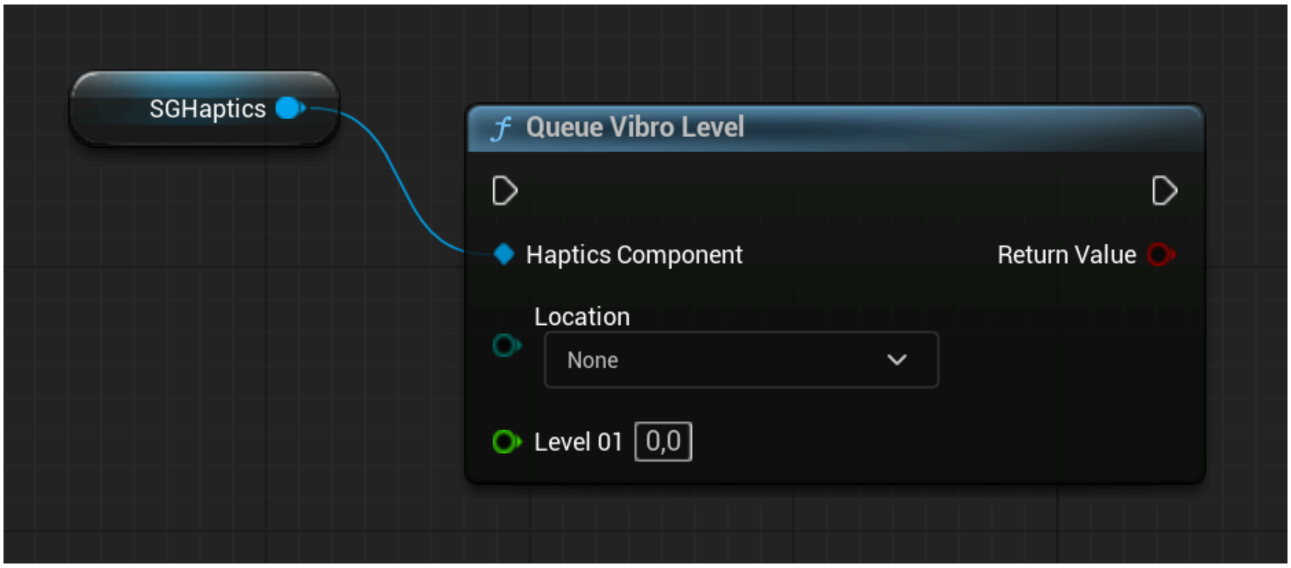
Returns:

- `true` : If queued successfully.
- `false` : If it fails.

Queue Vibro Level

> **Important**
>
> **Legacy Function – Use Custom Waveforms Instead**
>
> `Queue Vibro Level` is retained for backward compatibility with older API releases.
>
> Internally, it delegates to `Send Custom Waveform`, which is the recommended method for applying vibrotactile feedback.
>
> For new projects, prefer `Send Custom Waveform`, as it provides more fine-grained control over amplitude, frequency, timing, and waveform shaping.

Queues continuous vibration at a **specific location** to a set amplitude.



Parameters:

- **Location:** Target location to apply vibration.
- **Level01 (float):** Value clamped between `0.0` (no vibration) to `1.0` (full vibration).

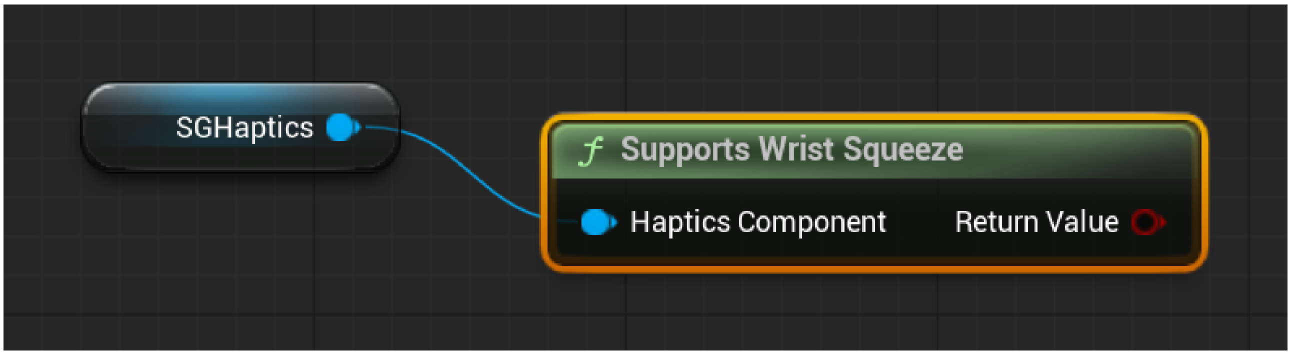
> **Note**
>
> Force-feedback value behavior:
>
> - `0.0` = No vibration.
> - `1.0` = Full vibration.
> - Values `< 0.0` are ignored.

Returns:

- `true` : If queued successfully.
- `false` : If it fails.

Supports Wrist Squeeze

Checks if the connected glove supports **active wrist-squeeze feedback**.

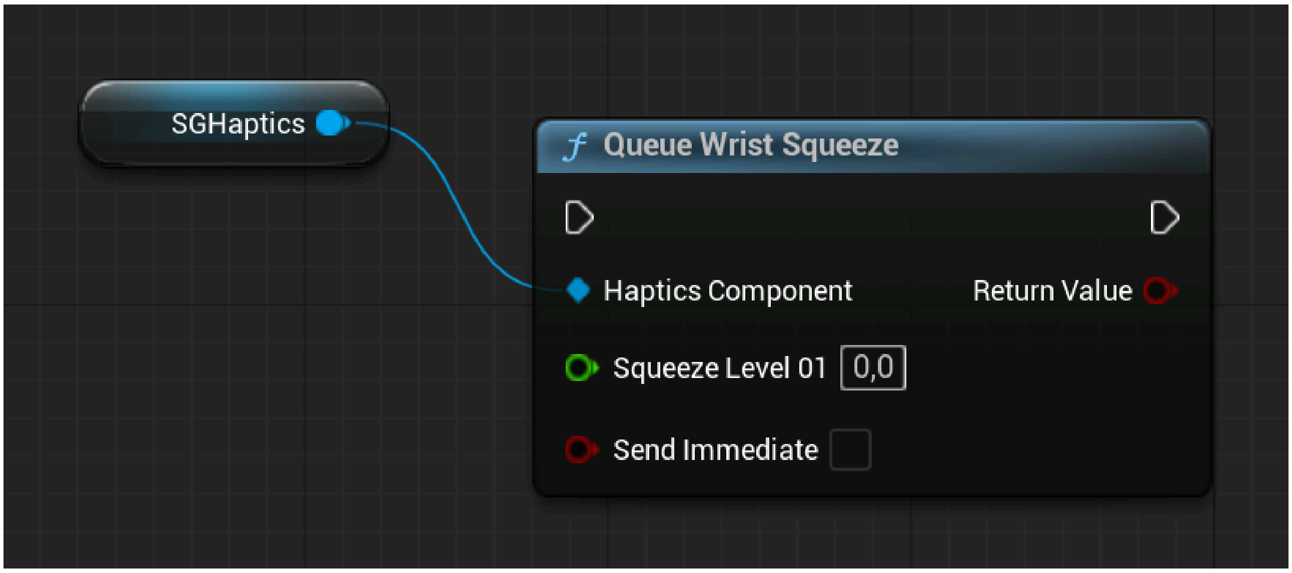


Returns:

- `true` : If wrist-squeeze is supported.
- `false` : If it's not supported.

Queue Wrist Squeeze

Queues a wrist-squeeze feedback at the desired level, and optionally if chosen, sends it right away.



Parameters:

- **Squeeze Level 01:** Value clamped between `0.0` (no squeeze) to `1.0` (full squeeze).

- **Send Immediate** If set to `true`, immediately sends the command, otherwise only queues until `Send Haptics` function is called.

> **Note**
>
> Wrist-squeeze value behavior:
>
> - `0.0` = No squeeze.
> - `1.0` = Full squeeze.

> **Caution**
>
> Avoid using `Send Immediate` unless absolutely necessary.
>
> For optimal performance, queue all haptic commands first and call the `Send Haptics` function once after all commands are prepared.
>
> Sending commands immediately increases device communication frequency and computational overhead. Batching commands using `Send Haptics` reduces processing cost and improves performance.

Returns:

- `true`: If queued successfully.
- `false`: If it fails.

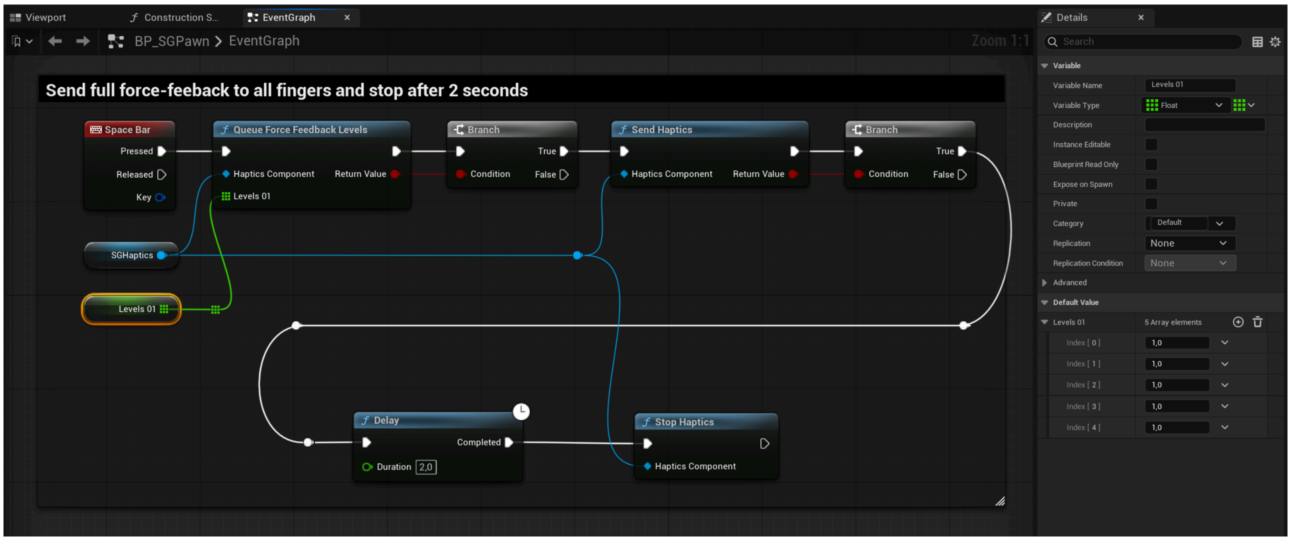
Blueprint Haptics Examples

Below are practical Blueprint examples demonstrating how to combine the different `SGHapticsComponent` functions into complete interaction flows.

Force-Feedback Example

This example demonstrates:

- How to queue force-feedback with **full resistance on all fingers**.
- How to flush all queued haptics (including the recently queued force-feedback) using `Send Haptics`.
- How to stop all haptic effects after `2` seconds, if the send operation succeeds.



Variable

Variable Name	Levels 01
Variable Type	Float
Description	
Instance Editable	☐
Blueprint Read Only	☐
Expose on Spawn	☐
Private	☐
Category	Default
Replication	None
Replication Condition	None

Advanced

Default Value

Levels 01	5 Array elements
Index [0]	1,0
Index [1]	1,0
Index [2]	1,0
Index [3]	1,0
Index [4]	1,0

In this flow:

1. Force-feedback levels are queued for all fingers.
2. `Send Haptics` compiles and sends the command to the glove.
3. If successful, `Stop Haptics` is used to clear all active effects after `2` seconds.

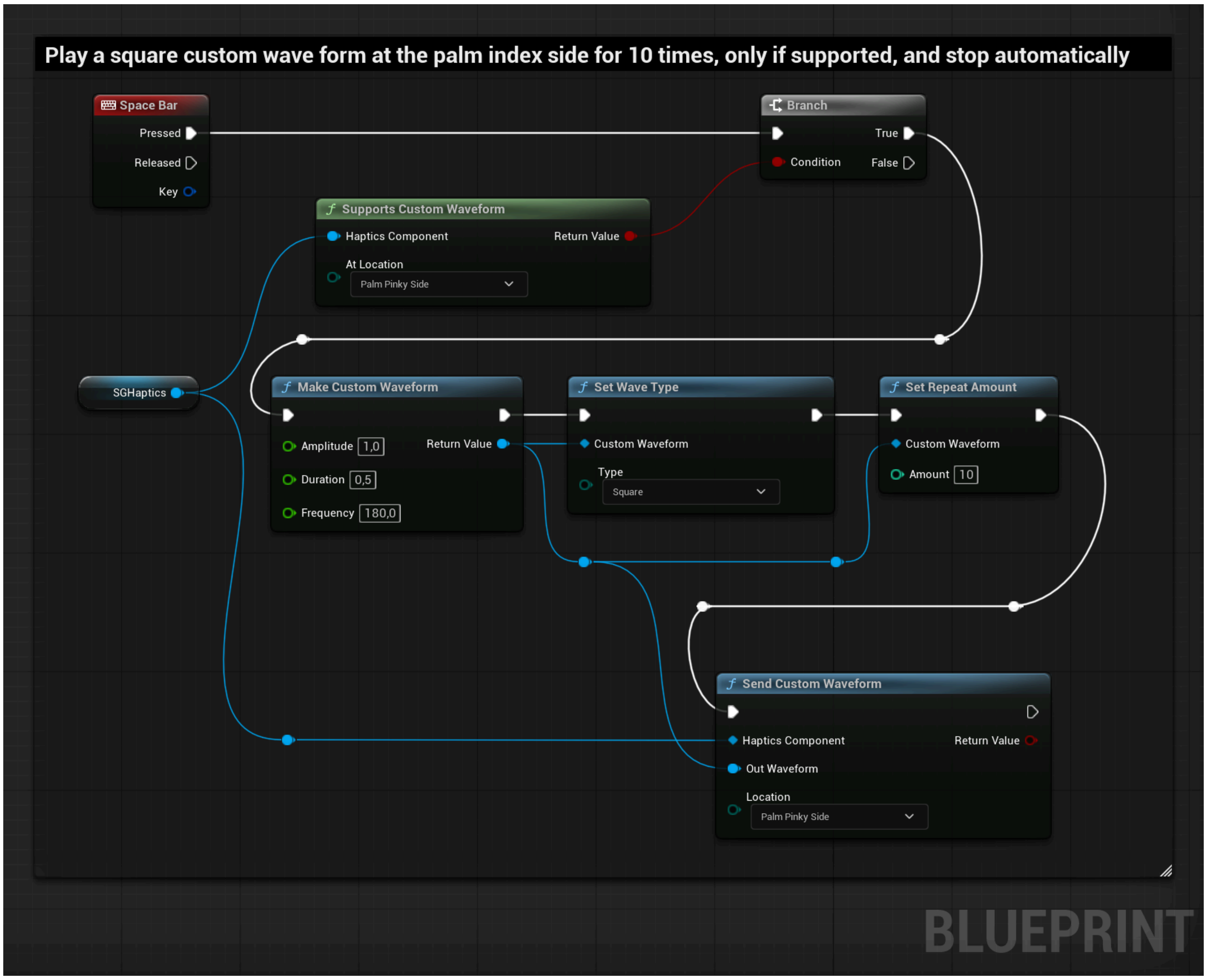
Vibrotactile Example

This example demonstrates:

- How to check if the glove at the current hand supports custom wave forms at the `Plam Pinky Side`.
- If so, it constructs a `SGCustomWaveform` with a duration of `500` milliseconds, amplitude of `1.0` at the frequency of `180.0` (maximum vibration on Nova 2).
- It then sets other parameters such as the `WaveType` to `Square` and the `RepeatAmount` to `10`.
- And, finally sends the custom waveforms to the glove, which is going to stop after `10` times playing.

This example demonstrates:

- How to check whether the current glove supports **custom waveforms** at the `Palm Pinky Side` location.
- How to construct a `SGCustomWaveform` with:
  - `Duration → 500 ms`
  - `Amplitude → 1.0`
  - `Frequency → 180.0 Hz` (maximum vibration on Nova 2)
- How to configure additional parameters such as:
  - `Wave Type → Square`
  - `Repeat Amount → 10`
- How to send the custom waveform to the glove.

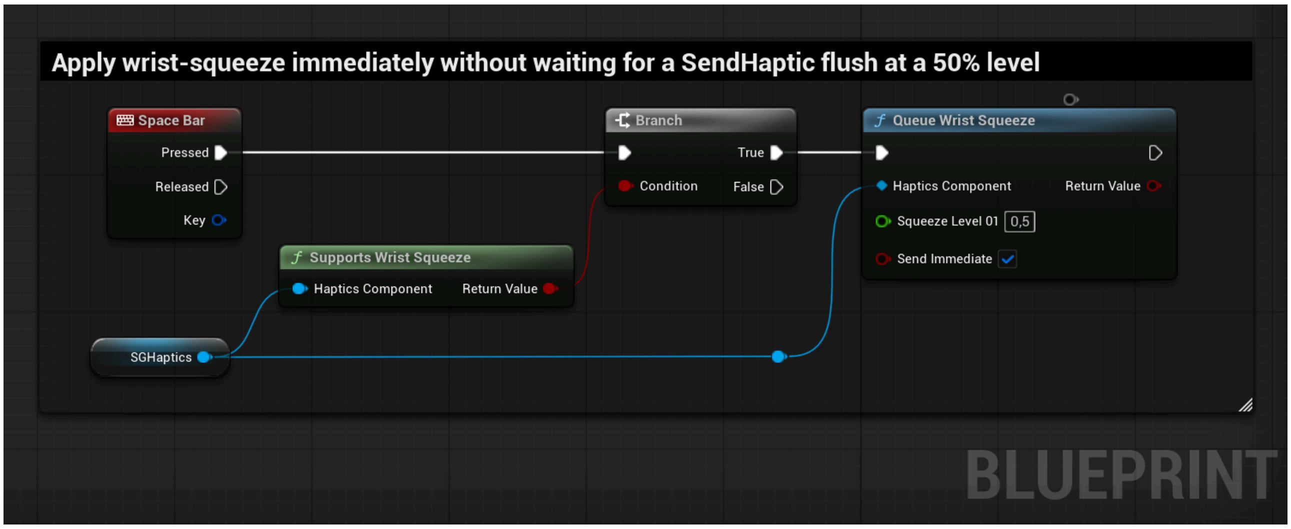


The waveform will automatically stop after playing **10 repetitions**.

Wrist-Squeeze Example

This example demonstrates:

- How to check whether the connected glove supports **wrist squeeze feedback**.
- How to apply a wrist squeeze at **50% intensity**.
- How to send the command immediately without requiring an additional `Send Haptics` call.



Because `Send Immediate` is enabled, the squeeze is transmitted instantly instead of being queued.

Caution

Avoid using `Send Immediate` unless absolutely necessary.

For optimal performance, queue all haptic commands first and call the `Send Haptics` function once after all commands are prepared.

Sending commands immediately increases device communication frequency and computational overhead. Batching commands using `Send Haptics` reduces processing cost and improves performance.

OpenXR

The SenseGlove Unreal Engine Plugin has provided OpenXR-compatible hand tracking by implementing `XR_EXT_hand_tracking` since `v2.1.0`.

Typically a user does not need to know anything about OpenXR to use the plugin, so this section of the handbook is for advanced users who are looking for a way to directly consume the OpenXR data coming directly from either a SenseGlove device or if enabled in the plugin settings from hand-tracking.

Since the SenseGlove Unreal Engine Plugin registers itself as an `OpenXRHandTracking` device it becomes a hand-tracking provider for Unreal Engine, thus the OpenXR data from SenseGlove could always be retrieved from the Unreal Engine's `IXTrackingSystem` with two caveats:

- The first caveat is, if another OpenXR-compatible hand-tracking plugin, e.g. Epic's own OpenXRHandTracking, is enabled simultaneously it's not guaranteed that the `FXRHandTrackingState` struct retrieved from the `IXTrackingSystem::GetHandTrackingState()` method is coming from SenseGlove, as these methods return the first hand-tracking plugin they could find. Thus, SenseGlove provides its own implementation of `GetHandTrackingState()` which guarantees the retrieved `FXRHandTrackingState` is coming from the SenseGlove Unreal Engine Plugin; and this is the preferred way to that.
- The second caveat is, Unreal `IXTrackingSystem::GetHandTrackingState()` method is blind to SenseGlove's wrist-tracking settings and offsets, therefore the returned `FXRHandTrackingState` retrieved using this engine function won't take into account the wrist-tracker offsets, and the hands end up at the wrong orientation or position. In contrast,

Important

In order to retrieve the latest `FXRHandTrackingState` available, The SenseGlove Unreal Engine Plugin provides an alternative implementation for `IXTrackingSystem::GetHandTrackingState()`, which guarantees the OpenXR hand-tracking data is coming from a SenseGlove device and also takes into account the SenseGlove's wrist-tracker offsets automatically.

This is the recommended approach over Unreal Engine's own `IXTrackingSystem::GetHandTrackingState()` or you have to ensure the data received is coming from a SenseGlove device yourself, and also take into account the SenseGlove's wrist-tracker settings and calculate the wrist offsets either using one of the `SGHapticGlove::GetWristLocation()` variants, or manually.

In short `IXTrackingSystem::GetHandTrackingState()` does not respect the offsets from `Project Settings > SenseGlove > Tracking > Wrist-Tracking Settings`

Caution

In order to retrieve the latest `FXRHandTrackingState` available, The SenseGlove Unreal Engine Plugin provides an alternative implementation for `IXTrackingSystem::GetHandTrackingState()`, which guarantees the OpenXR hand-tracking data is coming from a SenseGlove device and also takes into account the SenseGlove's wrist-tracker offsets automatically.

This is the recommended approach over Unreal Engine's own `IXTrackingSystem::GetHandTrackingState()` or you have to ensure the data received is coming from a SenseGlove device yourself, and also take into account the SenseGlove's wrist-tracker settings and calculate the wrist offsets either using one of the `GetWristLocation()` variants, e.g., `SGHandLayer::GetWristLocation()`, `SGHapticGlove::GetWristLocation()`, etc, or manually.

In short `IXTrackingSystem::GetHandTrackingState()` does not respect the offsets from `Project Settings > SenseGlove > Tracking Settings > Wrist Tracking Settings`.

Tip

Starting with version `v2.8.0`, the SenseGlove Unreal Engine Plugin provides a highly convenient, high-level abstraction that eliminates all of the manual steps described above. By using `SGHandTrackerComponent` you can simply add the component to your Pawn class (or any actor that requires hand-tracking data), configure its properties, and retrieve the tracking data with a single function call when needed.

The SenseGlove UE Plugin automatically handles all required settings and offset calculations for your positional tracking hardware, whether you are using pure hand tracking or a SenseGlove device. It also provides an optional debug hand out of the box, allowing you to instantly visualize hand-tracking data without writing a single line of code.

Furthermore, version `v2.8.0` ships with a companion component, `SGHapticsComponent`, which makes it easy to add haptic feedback to your own custom or third-party hand manipulation systems.

Tip

Sometimes, using `IXTrackingSystem::GetHandTrackingState()` is unavoidable, e.g., when using a third-party hand manipulation system such as **VR Expansion Plugin (VRE)**, which internally relies on `IXTrackingSystem::GetHandTrackingState()` to retrieve the hand-tracking data. In that case SenseGlove provides methods such as `SGHandLayer::GetWristLocation()`, `SGHapticGlove::GetWristLocation()`, etc, which you can use to reliably calculate the wrist offsets:

```
// Get the OpenXR hand-tracking data for the right hand
FXRHandTrackingState HandTrackingState;
IXTrackingSystem::GetHandTrackingState(
    GetWorld(),
    EXRSpaceType::UnrealWorldSpace,
    EControllerHand::Right,
    HandTrackingState);

// Return if the struct data is invalid!
if (!bGotHandTrackingState || !HandTrackingState.bValid)
{
    return;
}

// Return if the device is not being tracked!
if (HandTrackingState.TrackingStatus == ETrackingStatus::NotTracked)
{
    return;
}

// Ensure that HandTrackingState.HandKeyLocations has the location data
// for 26 joints!
if (!ensureAlwaysMsgf(HandTrackingState.HandKeyLocations.Num()
                      == EHandKeypointCount,
                      TEXT("Invalid HandKeyLocations count!")))
{
    return;
}

// Ensure that HandTrackingState.HandKeyRotations has the rotation data
// for 26 joints!
if (!ensureAlwaysMsgf(HandTrackingState.HandKeyRotations.Num()
                      == EHandKeypointCount,
                      TEXT("Invalid HandKeyRotations count!")))
{
    return;
}

{
    // FQuat variant of FSGHandLayer::GetWristLocation()
    // Get the OpenXR hand-tracking data with the correct wrist offsets
for
    // Meta Quest 3 Controllers
    FVector WristLocation;
    FQuat WristRotation;
    FSGHandLayer::GetWristLocation(
        true, // true for a right-handed glove and false for a left-handed
one
        HandTrackingState.HandKeyLocations[0], // 0 is the Wrist joint
location
        HandTrackingState.HandKeyRotations[0], // 0 is the wrist joint
rotation
        ESGPositionalTrackingHardware::Quest3Controller,
        WristLocation, WristRotation);

    // WristLocation and WristRotation variables now contain the correct
    // OpenXR hand-tracking data with the wrist offsets applied correctly
}

{
    // FRotator variant of FSGHandLayer::GetWristLocation()
    // Get the OpenXR hand-tracking data with the correct wrist offsets
for
    // Meta Quest 3 Controllers
    FVector WristLocation;
    FRotator WristRotation;
    FSGHandLayer::GetWristLocation(
        true, // true for a right-handed glove and false for a left-handed
one
        HandTrackingState.HandKeyLocations[0], // 0 is the Wrist joint
location
        HandTrackingState.HandKeyRotations[0].Rotator, // 0 is the wrist
joint rotation
        ESGPositionalTrackingHardware::Quest3Controller,
        WristLocation, WristRotation);

    // WristLocation and WristRotation variables now contain the correct
    // OpenXR hand-tracking data with the wrist offsets applied correctly
}
```

In the next sections we'll see:

- How we can directly consume the `FXRHandTrackingState` on UE `5.5` to draw and animate debug virtual hands in both Blueprint and C++.
- The Consuming OpenXR Hand-Tracking Data tutorial series provides a comprehensive introduction to virtual reality, OpenXR hand-tracking, and gesture detection in Unreal Engine. Additionally, this tutorial series covers procedural virtual hand mesh animation using the OpenXR hand-tracking data.

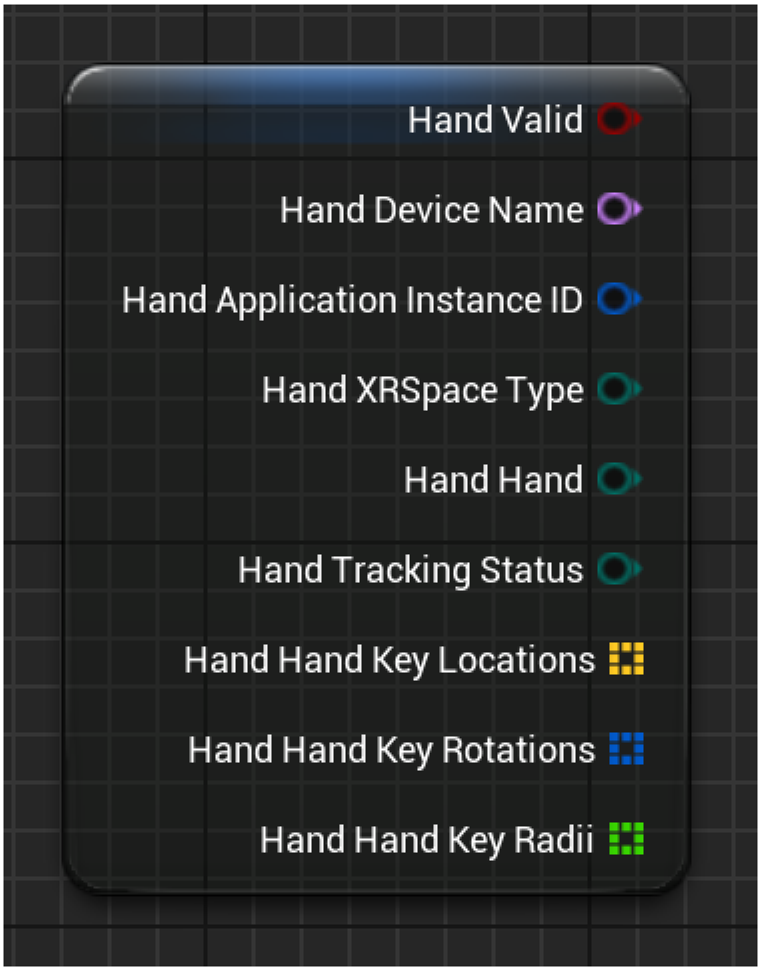
Consuming FXRHandTrackingState

> **Tip**
>
> Starting with version `v2.8.0`, a highly convenient high-level approach for retrieving `FXRHandTrackingState` has been introduced. It allows you to obtain OpenXR hand-tracking data without worrying about the caveats mentioned in the `main OpenXR section`.
>
> `SGHandTrackerComponent` abstracts these complexities away and provides correctly adjusted hand-tracking data with a single function call.
>
> This version also introduces a companion `SGHapticsComponent` is also available for easily integrating haptic feedback.

Taking a closer look at the `FXRHandTrackingState` declaration inside the Unreal Engine's `HeadMountedDisplay` module at `Engine/Source/Runtime/HeadMountedDisplay/Public/HeadMountedDisplayTypes.h`, figuring out the data structure might not seem very straightforward:

```
USTRUCT(BlueprintType)
struct FXRHandTrackingState
{
    GENERATED_USTRUCT_BODY();

    // The state is valid if poses have ever been provided.
    UPROPERTY(BlueprintReadOnly, Category = "XR")
    bool bValid = false;
    UPROPERTY(BlueprintReadOnly, Category = "XR")
    FName DeviceName;
    UPROPERTY(BlueprintReadOnly, Category = "XR")
    FGuid ApplicationInstanceID;

    UPROPERTY(BlueprintReadOnly, Category = "XR")
    EXRSpaceType XRSpaceType = EXRSpaceType::UnrealWorldSpace;

    UPROPERTY(BlueprintReadOnly, Category = "XR")
    EControllerHand Hand = EControllerHand::Left;

    UPROPERTY(BlueprintReadOnly, Category = "XR")
    ETrackingStatus TrackingStatus = ETrackingStatus::NotTracked;

    // The indices of this array are the values of EHandKeypoint (Palm,
Wrist, ThumbMetacarpal, etc).
    UPROPERTY(BlueprintReadOnly, Category = "XR")
    TArray<FVector> HandKeyLocations;

    // The indices of this array are the values of EHandKeypoint (Palm,
Wrist, ThumbMetacarpal, etc).
    UPROPERTY(BlueprintReadOnly, Category = "XR")
    TArray<FQuat> HandKeyRotations;

    // The indices of this array are the values of EHandKeypoint (Palm,
Wrist, ThumbMetacarpal, etc).
    UPROPERTY(BlueprintReadOnly, Category = "XR")
    TArray<float> HandKeyRadii;
};
```

Which on the Blueprint side it looks like this:



But, fear not, we've got you covered!

FXRHandTrackingState in Unreal Engine

`FXRHandTrackingState` is a structure in Unreal Engine designed to hold detailed information about the state of a hand-tracking device at a given moment. This structure is essential for handling hand-tracking inputs in virtual reality (VR) applications, providing the necessary data to accurately track and represent the user's hand movements and actions within the virtual environment.

Structure Members of FXRHandTrackingState

- **bValid**
  - **Description**: A boolean flag indicating whether the data is valid or not.
  - **Usage**: This is used to check if the motion controller data is correctly initialized and can be used for further processing.
- **DeviceName**
  - **Type**: `FName`
  - **Description**: The name of the device.
  - **Usage**: Identifies which device the data is coming from, useful when multiple devices are in use.
- **ApplicationInstanceID**
  - **Type**: `FString`
  - **Description**: A unique identifier for the application instance.
  - **Usage**: Helps in differentiating data from different instances of an application, ensuring the correct instance processes the data.
- **XRSpaceType**
  - **Type**: `EXRSpaceType`
  - **Description**: Enum specifying the type of XR space being used (e.g., unreal world or tracking space).
  - **Usage**: Specifies the coordinate system the XR Device is tracking itself in.
- **Hand**
  - **Type**: `EControllerHand`
  - **Description**: Enum indicating which hand is being tracked (left or right).
  - **Usage**: Helps identify whether the hand-tracking data pertains to the left or right hand, essential for hand-specific actions or interactions.
- **TrackingStatus**
  - **Type**: `EXRTrackingStatus`

  - **Description**: Enum indicating the tracking status of the hand-tracking device.
  - **Usage**: Shows whether the hand-tracking device is being tracked accurately, with possible statuses like `Tracked`, `NotTracked`, etc.

- **HandKeyLocations**
  - **Type**: `TArray<FVector>`
  - **Description**: An array of vectors representing key locations of the hand.
  - **Usage**: Provides detailed locations of key points on the hand, useful for precise hand-tracking and interaction.

- **HandKeyRotations**
  - **Type**: `TArray<FQuat>`
  - **Description**: An array of quaternions representing key rotations of the hand.
  - **Usage**: Complements the hand key locations with rotational data, ensuring accurate representation of hand movements.

- **HandKeyRadii**
  - **Type**: `TArray<float>`
  - **Description**: An array of floats representing the radii of key points of the hand.
  - **Usage**: Gives the size of the hand key points, aiding in collision detection and interaction fidelity.

Organization of FXRHandTrackingState

The structure is organized to encapsulate all relevant data needed for hand-tracking in a coherent and accessible manner. Boolean flag `bValid` provides quick checks on the state of the controller data. Identifiers `DeviceName` and `ApplicationInstanceID` ensure the correct association of data. Arrays `HandKeyLocations`, `HandKeyRotations`, and `HandKeyRadii` allow detailed hand-tracking, which is critical for immersive VR experiences. Lastly, the tracking status `TrackingStatus` informs the system of the reliability of the data being processed and whether the hands are actively being tracked or they are inactive at the moment.

Processing the Data for Drawing and Animating a Virtual Hand

In order to draw and animate a virtual hand in real-time whether the data is coming from hand-tracking or a SenseGlove device, we could consume the data from the `HandKeyLocations` and `HandKeyRotations` fields of the `FXRHandTrackingState` struct.

Both `HandKeyLocations` and `HandKeyRotations` contain 26 elements as defined by OpenXR's `XR_HAND_JOINT_COUNT_EXT` and `XrHandJointLocationsEXT`, etc.

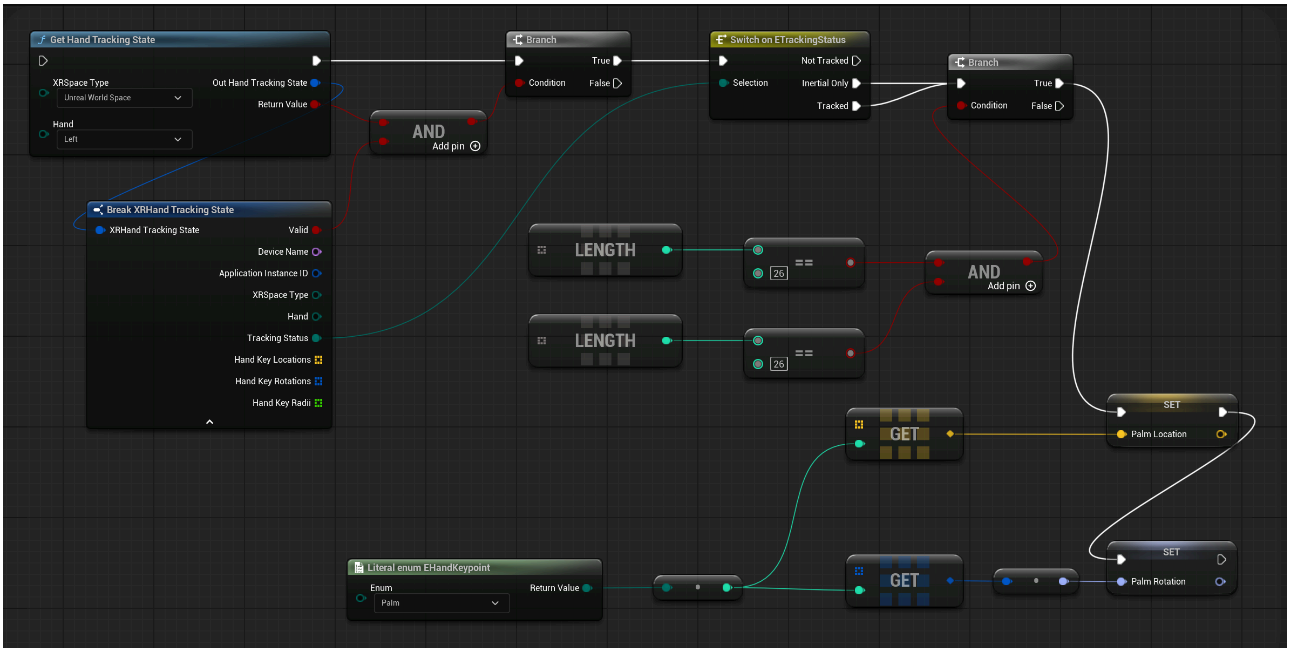
Unreal Engine also provides an enum called `EHandKeypoint` naming the 26 joints, and the equivalent of `XR_HAND_JOINT_COUNT_EXT` as `EHandKeypointCount` inside `Engine/Source/Runtime/HeadMountedDisplay/Public/HeadMountedDisplayTypes.h` as follows:

```
/**
 * Transforms that are tracked on the hand.
 * Matches the enums from WMR to make it a direct mapping
 */
UENUM(BlueprintType)
enum class EHandKeypoint : uint8
{
    Palm,
    Wrist,
    ThumbMetacarpal,
    ThumbProximal,
    ThumbDistal,
    ThumbTip,
    IndexMetacarpal,
    IndexProximal,
    IndexIntermediate,
    IndexDistal,
    IndexTip,
    MiddleMetacarpal,
    MiddleProximal,
    MiddleIntermediate,
    MiddleDistal,
    MiddleTip,
    RingMetacarpal,
    RingProximal,
    RingIntermediate,
    RingDistal,
    RingTip,
    LittleMetacarpal,
    LittleProximal,
    LittleIntermediate,
    LittleDistal,
    LittleTip
};

const int32 EHandKeypointCount = static_cast<int32>(EHandKeypoint::LittleTip)
+ 1;
```

So, getting the any joint's location or rotation is as easy as casting the enum value and passing it as the array index.

```
    FXRHandTrackingState HandTrackingState;
    const bool bGotHandTrackingState = FSGXRTracker::GetHandTrackingState(
        GetWorld(), EXRSpaceType::UnrealWorldSpace, EControllerHand::Left,
HandTrackingState);

    // Return if the struct data is invalid!
    if (!bGotHandTrackingState || !HandTrackingState.bValid)
    {
        return;
    }

    // Return if the device is not being tracked!
    if (HandTrackingState.TrackingStatus == ETrackingStatus::NotTracked)
    {
        return;
    }

    // Ensure that HandTrackingState.HandKeyLocations has the location data
    // for 26 joints!
    if (!ensureAlwaysMsgf(HandTrackingState.HandKeyLocations.Num()
                         == EHandKeypointCount,
                         TEXT("Invalid HandKeyLocations count!")))
    {
        return;
    }

    // Ensure that HandTrackingState.HandKeyRotations has the rotation data
    // for 26 joints!
    if (!ensureAlwaysMsgf(HandTrackingState.HandKeyRotations.Num()
                         == EHandKeypointCount,
                         TEXT("Invalid HandKeyRotations count!")))
    {
        return;
    }

    static constexpr int32 PalmIndex = static_cast<int32>
(EHandKeypoint::Palm);

    const FVector& PalmLocation{
        HandTrackingState.HandKeyLocations[PalmIndex]
    };
    const FRotator& PalmRotation{
        HandTrackingState.HandKeyRotations[PalmIndex].Rotator()
    };
```

The equivalent Blueprint code for the above looks something like this:



OK, now that we've got a glimpse of how the virtual hand's joint data could be processed we are going to draw and animate a virtual hand in both Blueprint and C++ in the upcoming sections.

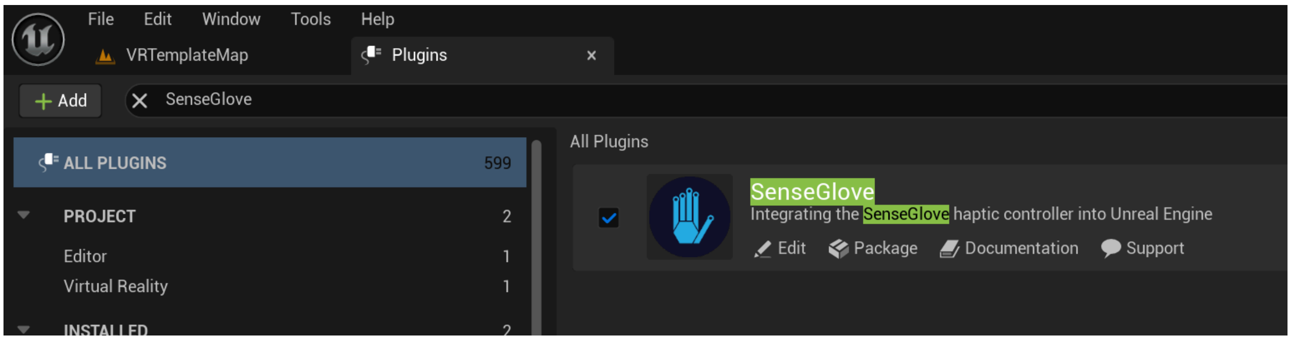
Consuming FXRHandTrackingState in Blueprint

Before continuing this section, please ensure you've studied the Consuming FXRHandTrackingState section, first.

> **Tip**
>
> Starting with version `v2.8.0`, a highly convenient high-level approach for retrieving `FXRHandTrackingState` has been introduced. It allows you to obtain OpenXR hand-tracking data without worrying about the caveats mentioned in the `main OpenXR section`.
>
> `SGHandTrackerComponent` abstracts these complexities away and provides correctly adjusted hand-tracking data with a single function call.
>
> This version also introduces a companion `SGHapticsComponent` is also available for easily integrating haptic feedback.

Drawing and Animating Virtual Hands

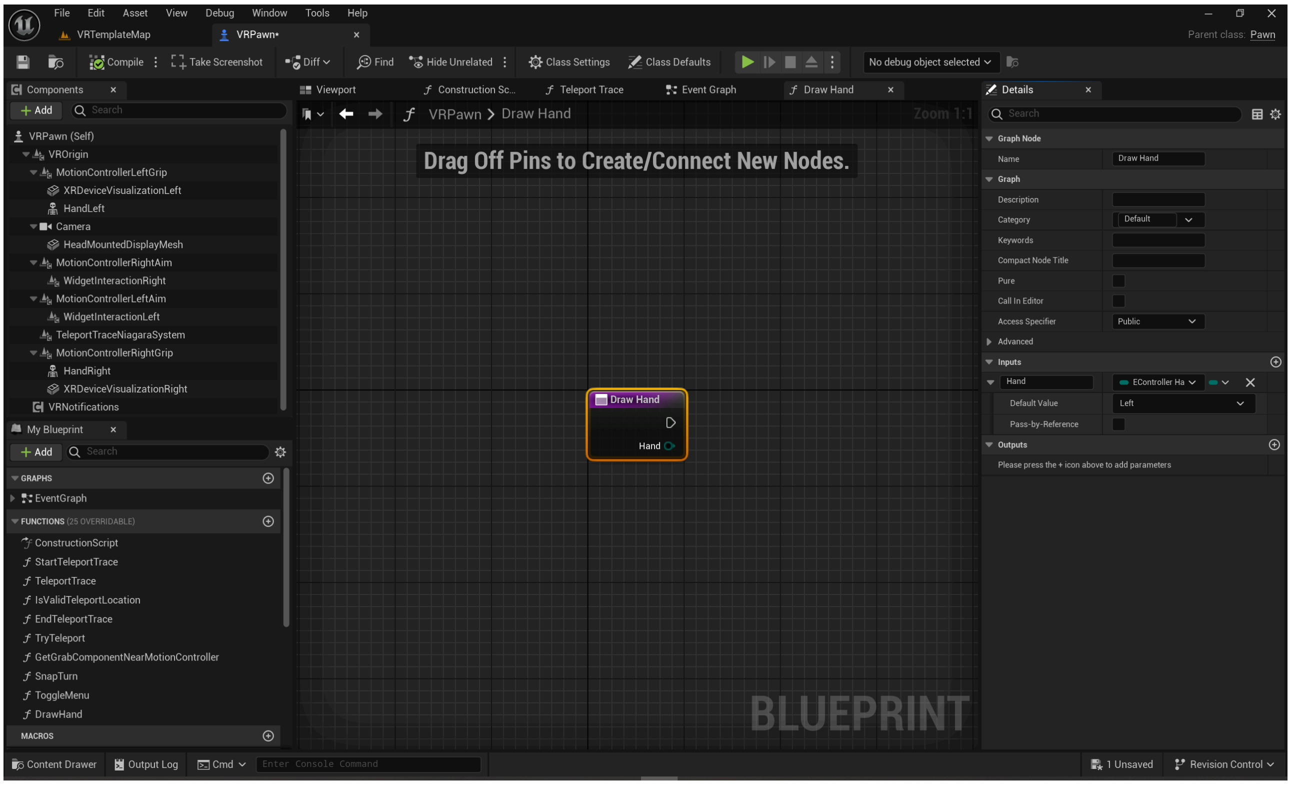
1. Create a new Virtual Reality project based the Unreal VR Template.
2. Make sure the SenseGlove UnrealEngine plugin is installed and enabled inside your new project.



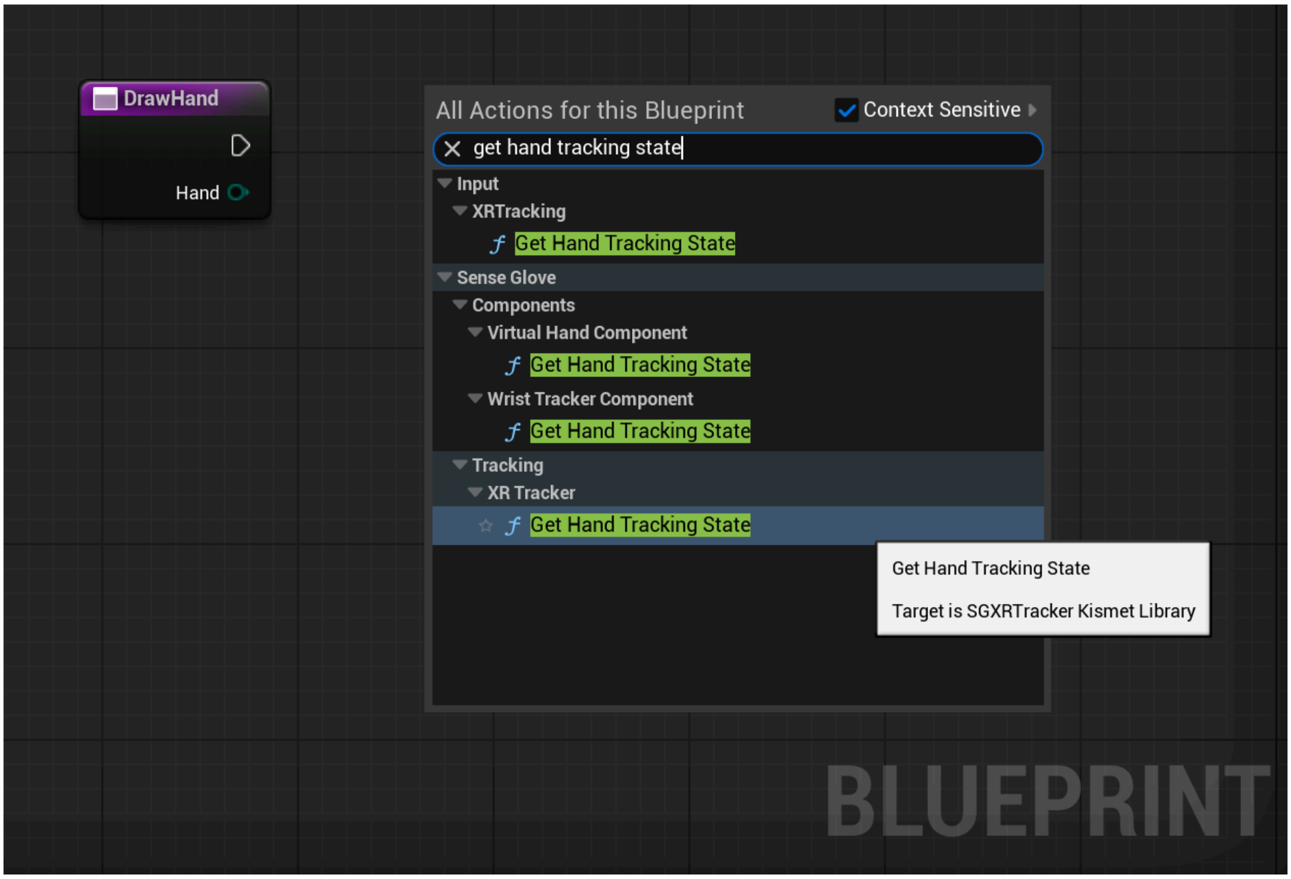
3. You could use either hand-tracking or a SenseGlove device as the input data, or both of the inside the same project. Whether you would like to use hand-tracking or a SenseGlove device, please make sure the required steps are taken for each of those first.

4. You could add the required Blueprint code for drawing virtual hands to either your Level Buleprint or the VRPawn Blueprint Class located at `/Content/VRTemplate/Blueprints/VRPawn`. In this guide we are going to add the code to our VRPawn.

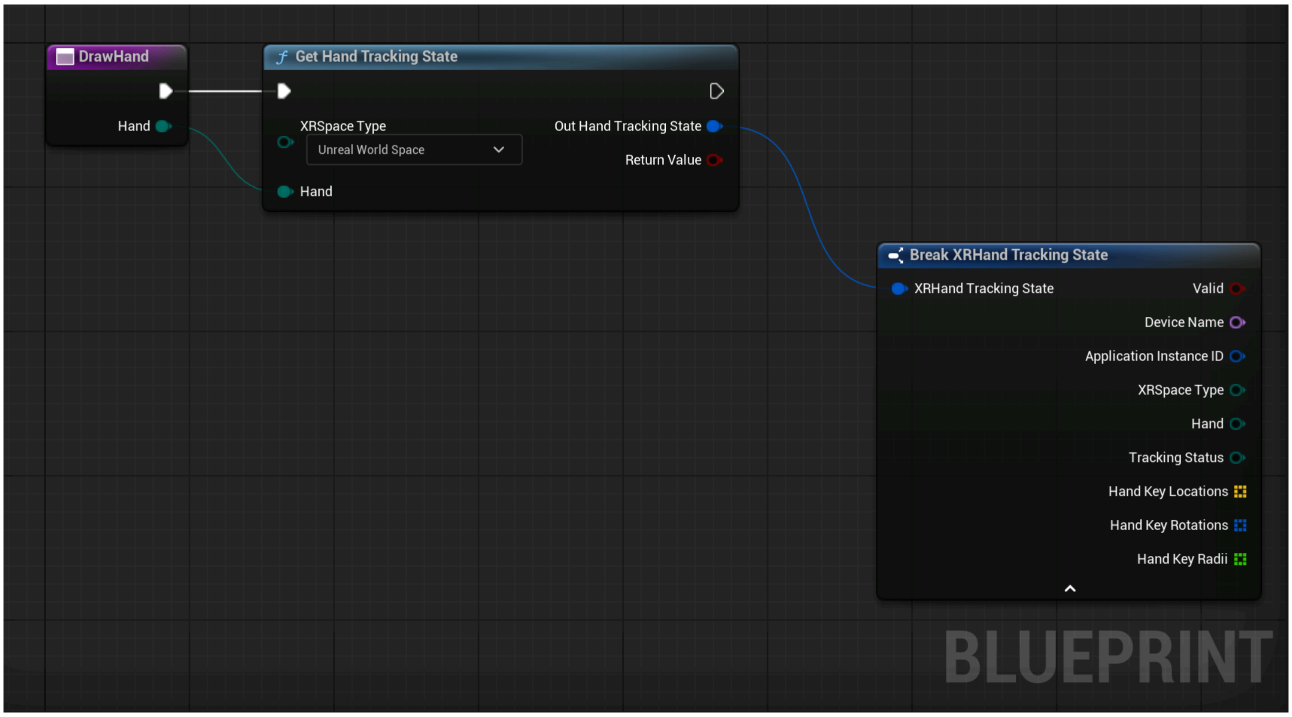
5. Add a new function named `Draw Hand` with an input parameter of type `EController Hand` named `Hand`.



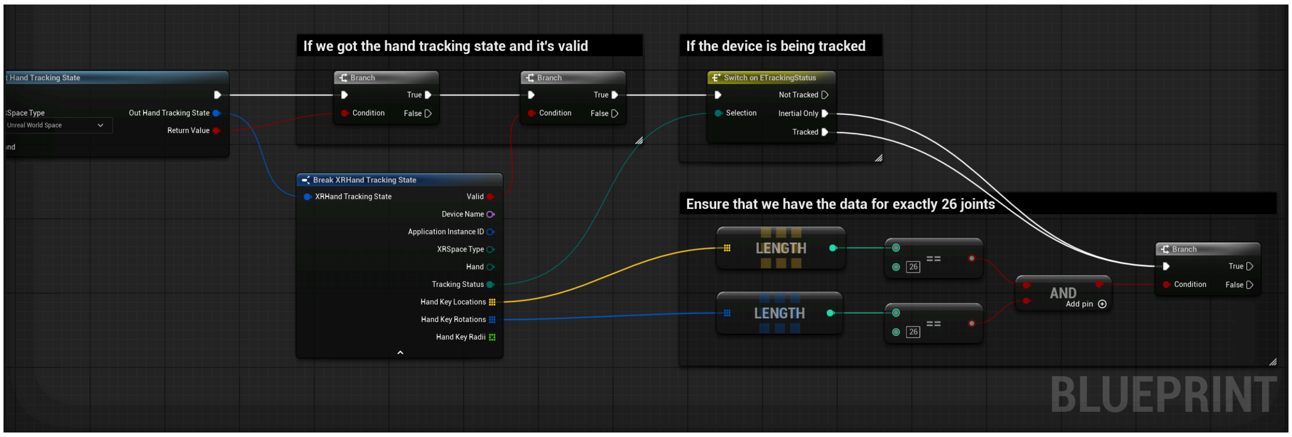
6. Inside this function's event graph add a `Get Hand Tracking State` node from `SenseGlove > Tracking > XR Tracker > Get Hand Tracking State`.



7. Then connect the functions `Hand` input parameter to the `Get Hand Tracking State`'s `Hand` input and right-click on the `OutHandTrackingState` parameter and use the `Break XRHandTrackingState` node to break the struct to it's fields.

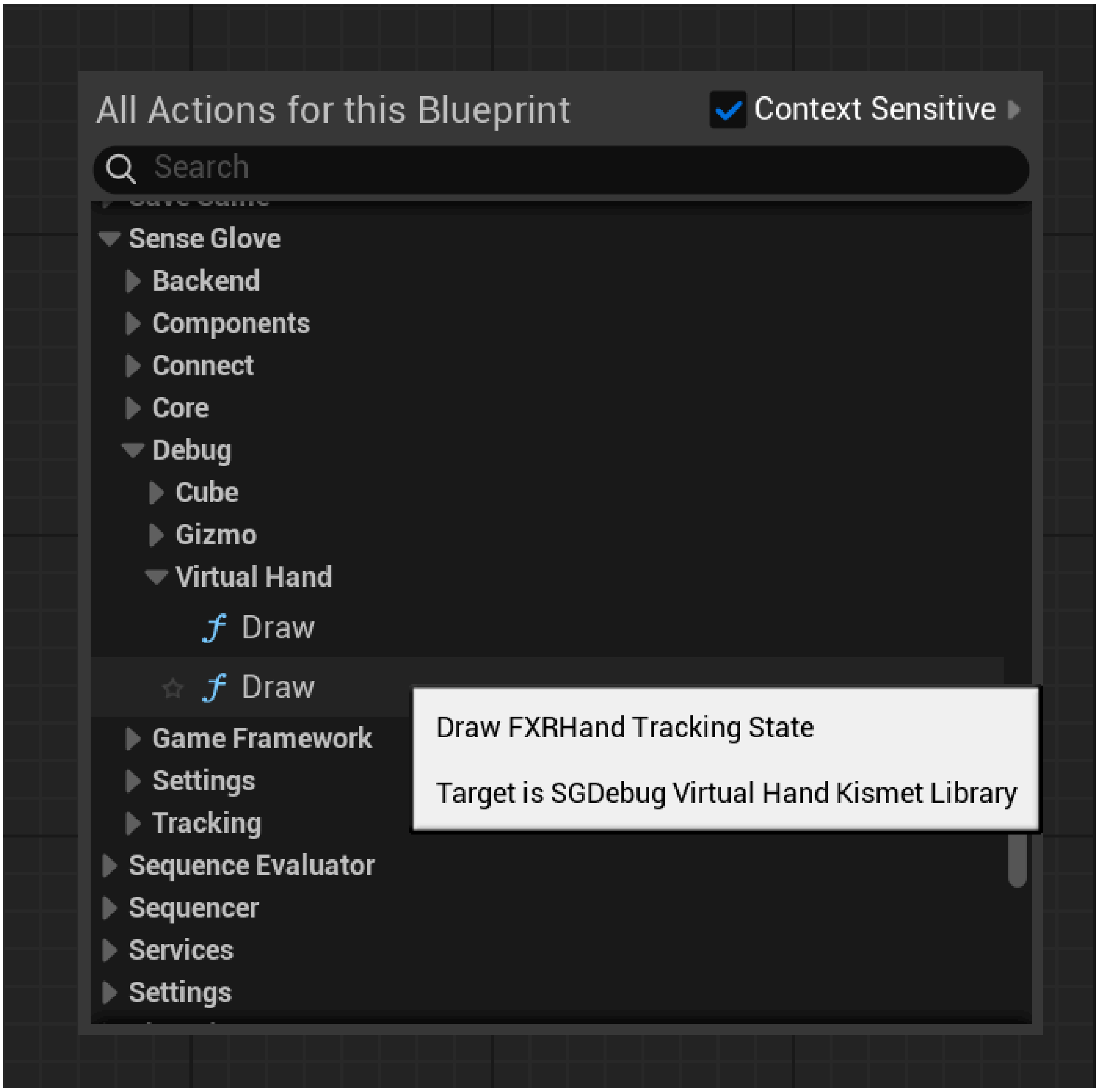


8. After this, we need to perform data validation by checking the return status of the `Get Hand Tracking State` function and `FXRHandTrackingState`'s `Valid` field. Then, we check if the hand-tracking device is being tracked and indeed coming from a hand-tracking source. And, finally, we check whether we have the positions and rotations for exactly `26` joints or not.


If we got the hand tracking state and it's valid
If the device is being tracked
Ensure that we have the data for exactly 26 joints
Branch
Condition
True
False
Switch on ETrackingStatus
Selection
Not Tracked
Inertial Only
Tracked
Break XRHand Tracking State
LENGTH
LENGTH
==
26
AND
Add pin
BLUEPRINT

9. OK, now it's time to draw the joints! If we check out the SenseGlove Debug module's draw option, we notice there are various ways to draw the debug virtual hand. Drawing a cube or a gizmo per joint, or draw the whole hand all at once by passing the retrieved `FXRHandTrackingState` to the

`DebugVirtualHand::Draw` function! But, since the point of this tutorial is to learn how to consume the `FXRHandTrackingState` we ignore the last option. Between the debug cubes or gizmos, we are going to choose the gizmos since they better represent the rotations than the cubes.

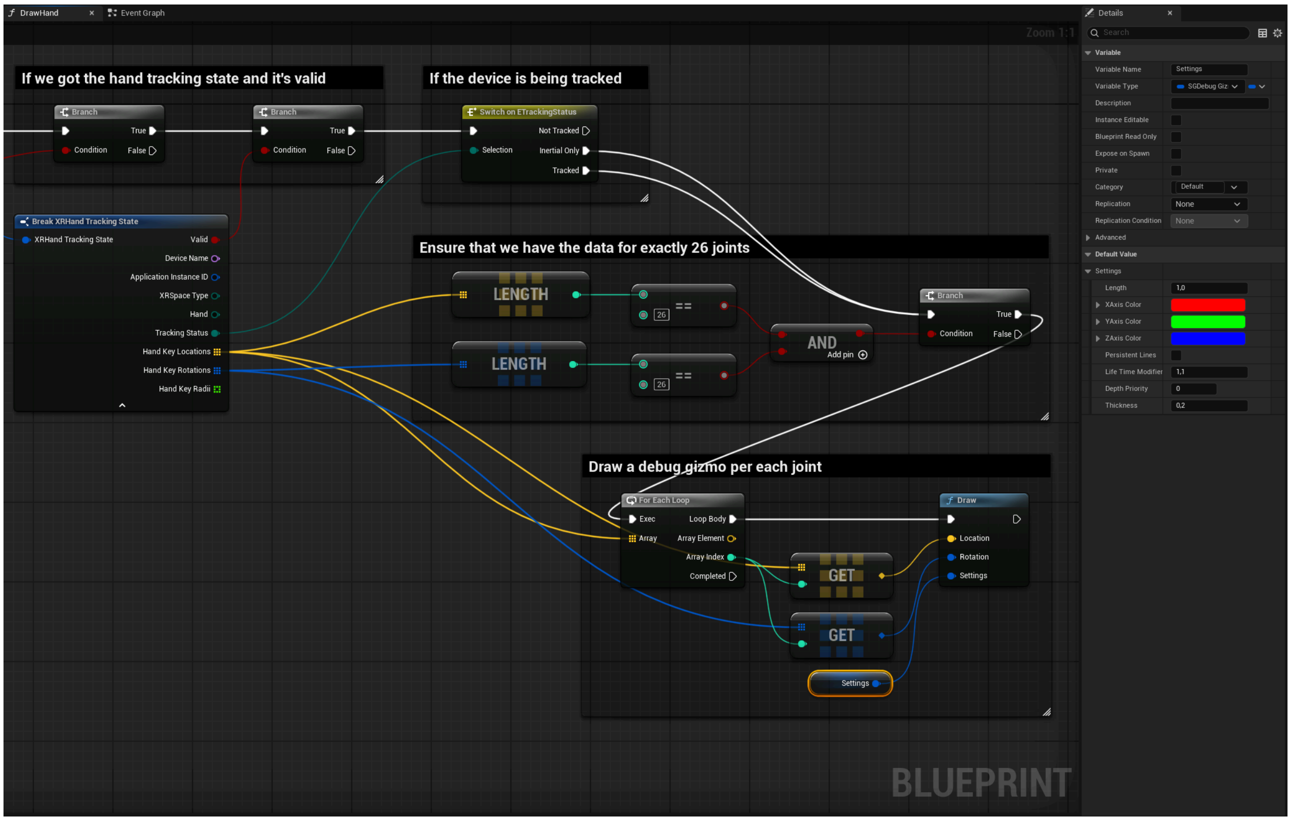


Draw FXRHand Tracking State

Target is SGDebug Virtual Hand Kismet Library

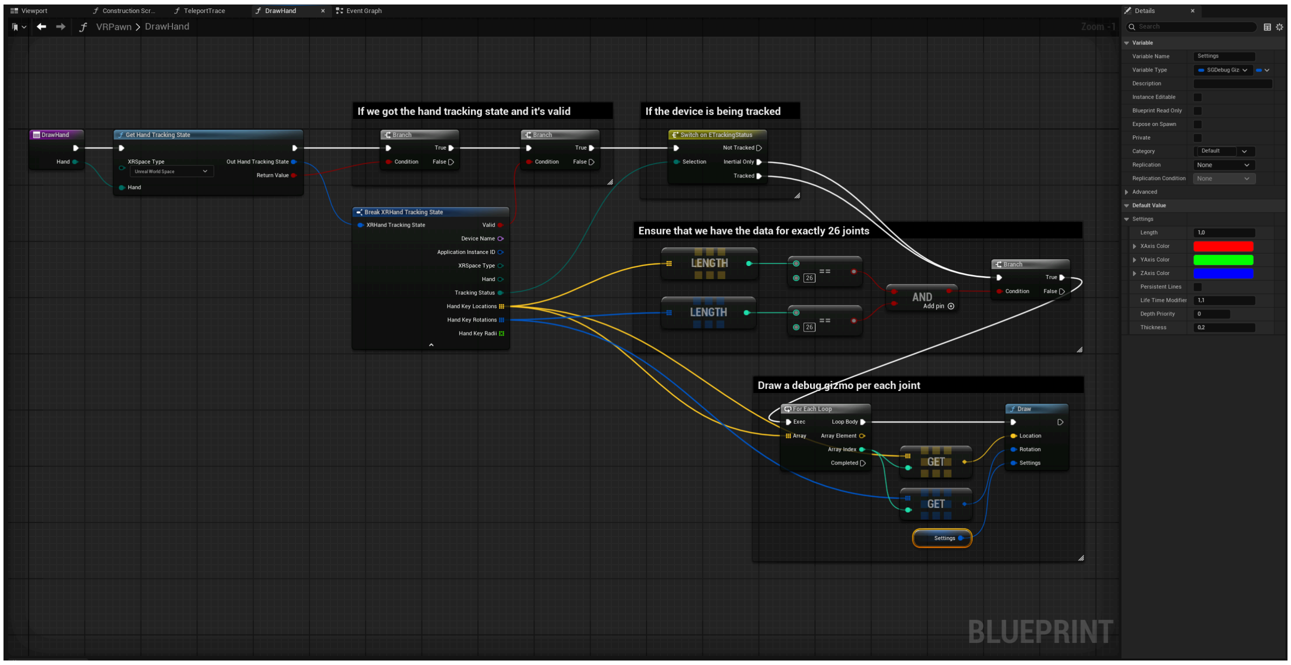
10. In the last step inside the `Draw Hand` function, in order to draw a virtual hand with `26` joints, we have to first iterate through either of the `Hand Key Positions` or `Hand Key Rotations` arrays from the `FXRHandTrackingState` struct.

Since we made sure both arrays have `26` elements before we reached this step, it's safe to just iterate over one and use the `Array Index` inside a `For Each Loop` or a `For Loop` to access the position and rotation of every joint. Then we use each array `Get (a ref)` method to access the position and rotation data inside the loop and call the `Draw` function from `SenseGlove > Debug > Gizmo` per every joint. Please note that there are two `Draw` functions and the only difference between the two is that one accepts an `FQuat` and the other a `FRotator` for its `Rotation` input parameter. In this case, we use the `FQuat` variant to avoid an extra conversion to `FRotator`. Also, please adjust the `Thickness` option for the `Settings` parameter from `1.0` to `0.2`, as the default value might be too thick for drawing a joint gizmo.

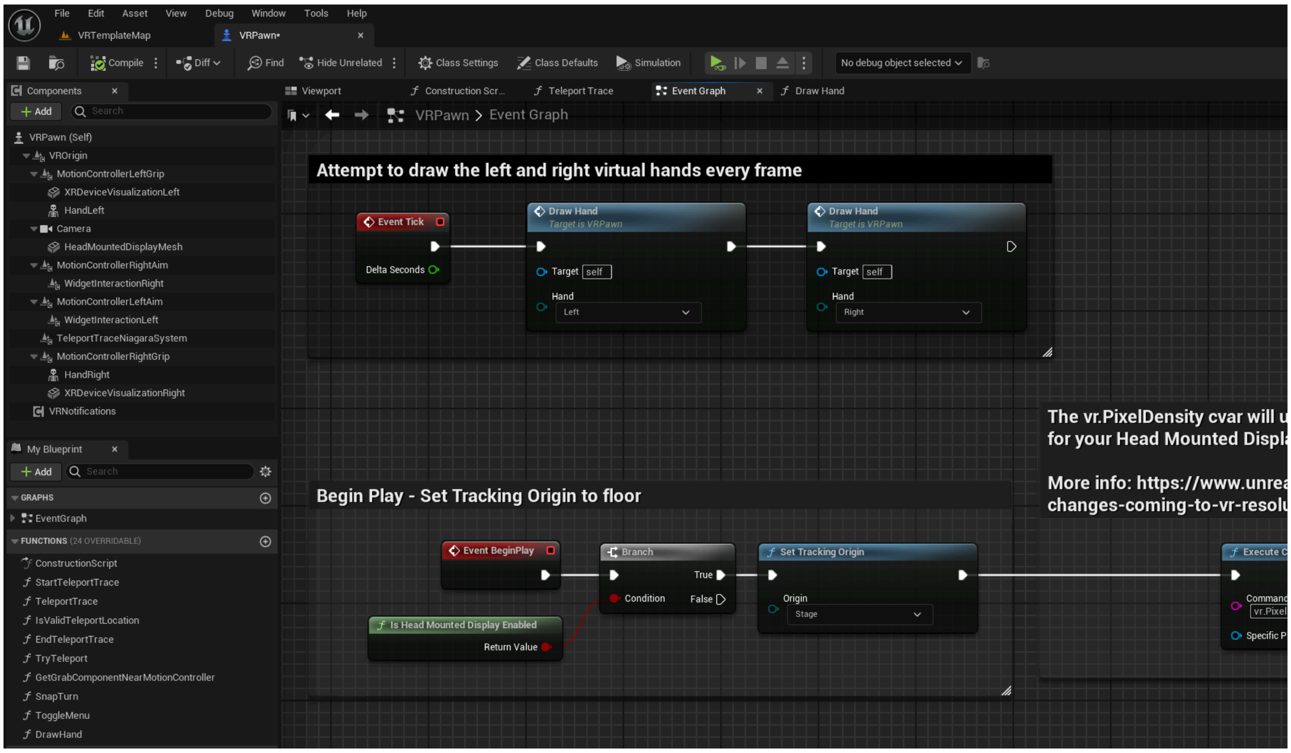


Variable	
Variable Name	Settings
Variable Type	SGDebug Giz
Description	
Instance Editable	☐
Blueprint Read Only	☐
Expose on Spawn	☐
Private	☐
Category	Default
Replication	None
Replication Condition	None
Advanced	
Default Value	
Settings	
Length	1,0
XAxis Color	🟥
YAxis Color	🟩
ZAxis Color	🟦
Persistent Lines	☐
Life Time Modifier	1,1
Depth Priority	0
Thickness	0,2

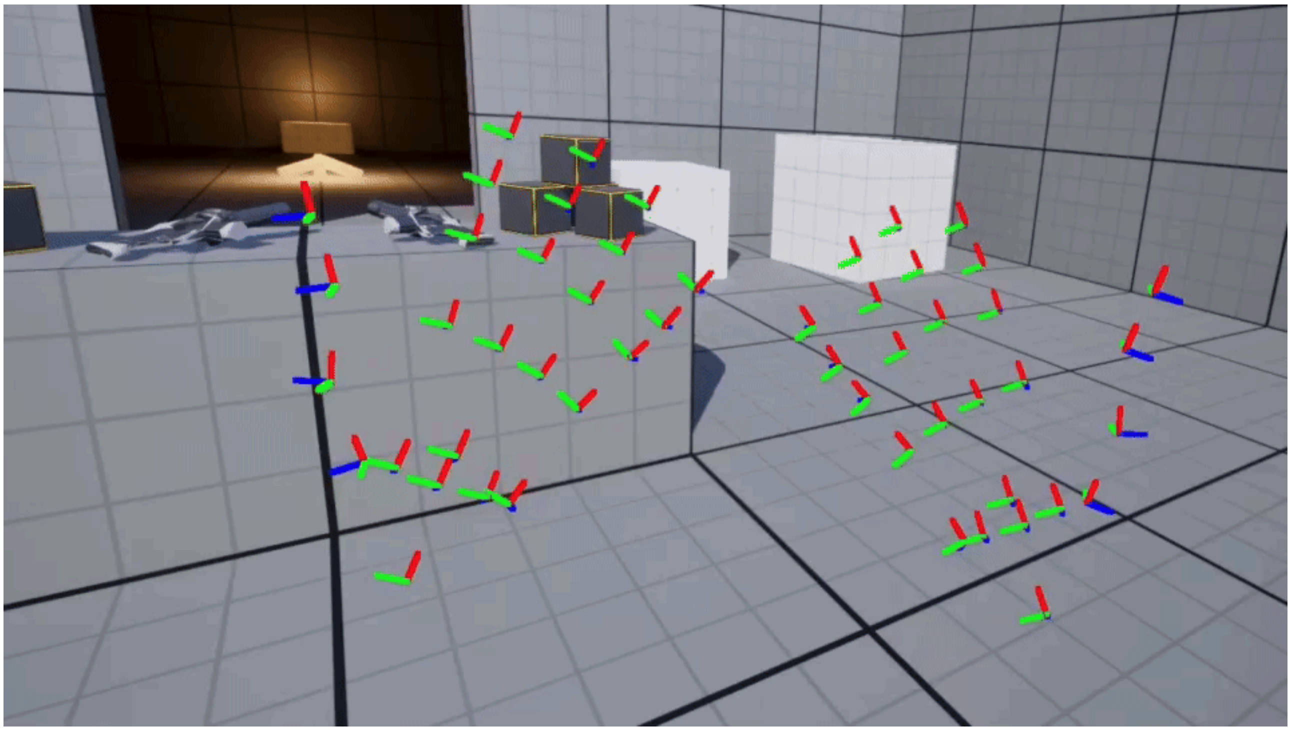
11. Well, now the full implementation for the `Draw Hand` function insde the `VRPawn` should look something like this:


If we got the hand tracking state and it's valid
If the device is being tracked
Ensure that we have the data for exactly 26 joints
Draw a debug gizmo per each joint
BLUEPRINT

12. Finally, go back to `VRPawn` 's event graph and the following code to the `Tick` event. Basically what we do here is call our newly implemented `Draw Hand` twice, once for each hand.


Attempt to draw the left and right virtual hands every frame
Event Tick
Delta Seconds
Draw Hand
Target is VRPawn
Target self
Hand
Left
Right
Begin Play - Set Tracking Origin to floor
Event BeginPlay
Branch
Is Head Mounted Display Enabled
Set Tracking Origin
Origin
Stage
The vr.PixelDensity cvar will u
for your Head Mounted Displ
More info: https://www.unre
changes-coming-to-vr-resolu

13. Now, go back to the `VRTemplateMap` and use the VR Preview button to run the game. If everything's done correctly, you should be able to see the virtual hands inside your VR simulation.



Consuming FXRHandTrackingState in C++

Before continuing this section, please ensure you've first studied the Consuming FXRHandTrackingState section.

Tip

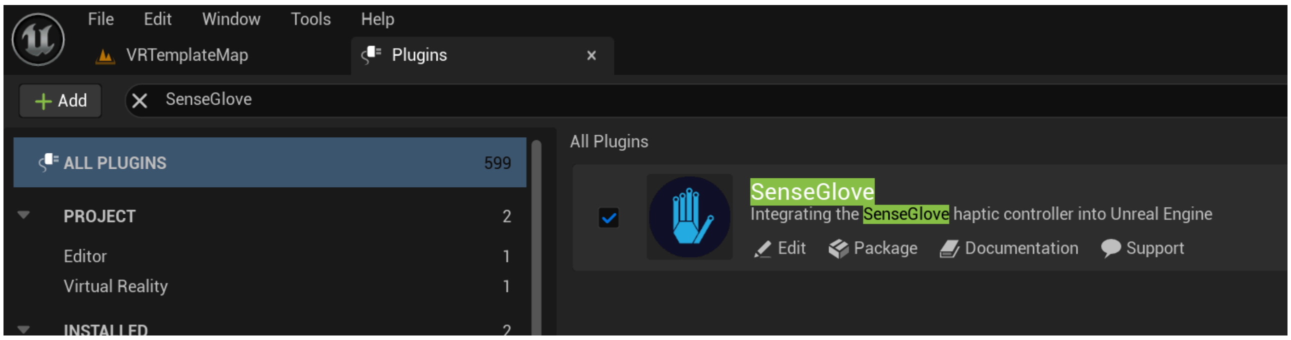
Starting with version `v2.8.0`, a highly convenient high-level approach for retrieving `FXRHandTrackingState` has been introduced. It allows you to obtain OpenXR hand-tracking data without worrying about the caveats mentioned in the `main OpenXR section`.

`SGHandTrackerComponent` abstracts these complexities away and provides correctly adjusted hand-tracking data with a single function call.

This version also introduces a companion `SGHapticsComponent` is also available for easily integrating haptic feedback.

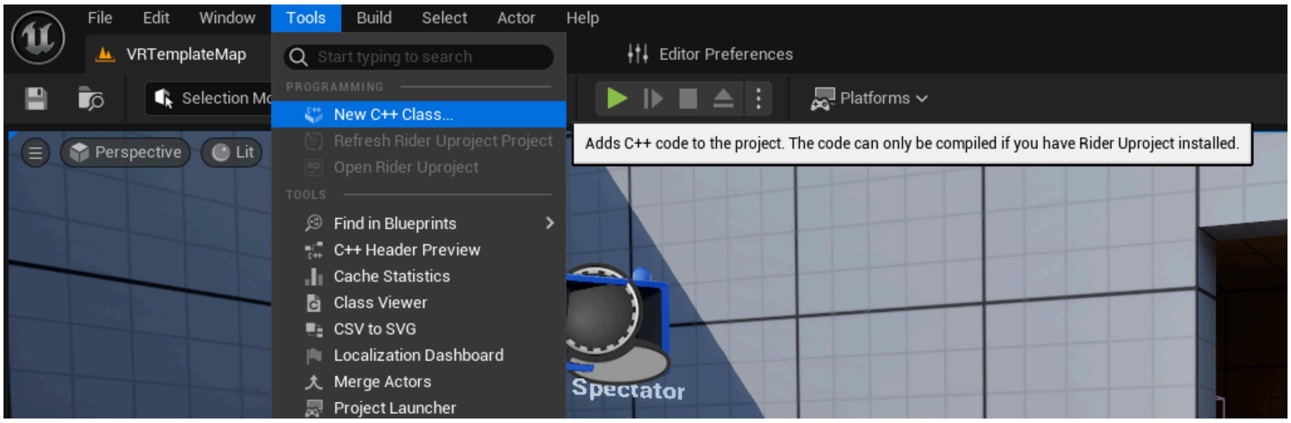
Drawing and Animating Virtual Hands

1. Create a new Virtual Reality project based the Unreal VR Template.
2. Make sure the SenseGlove UnrealEngine plugin is installed and enabled inside your new project.



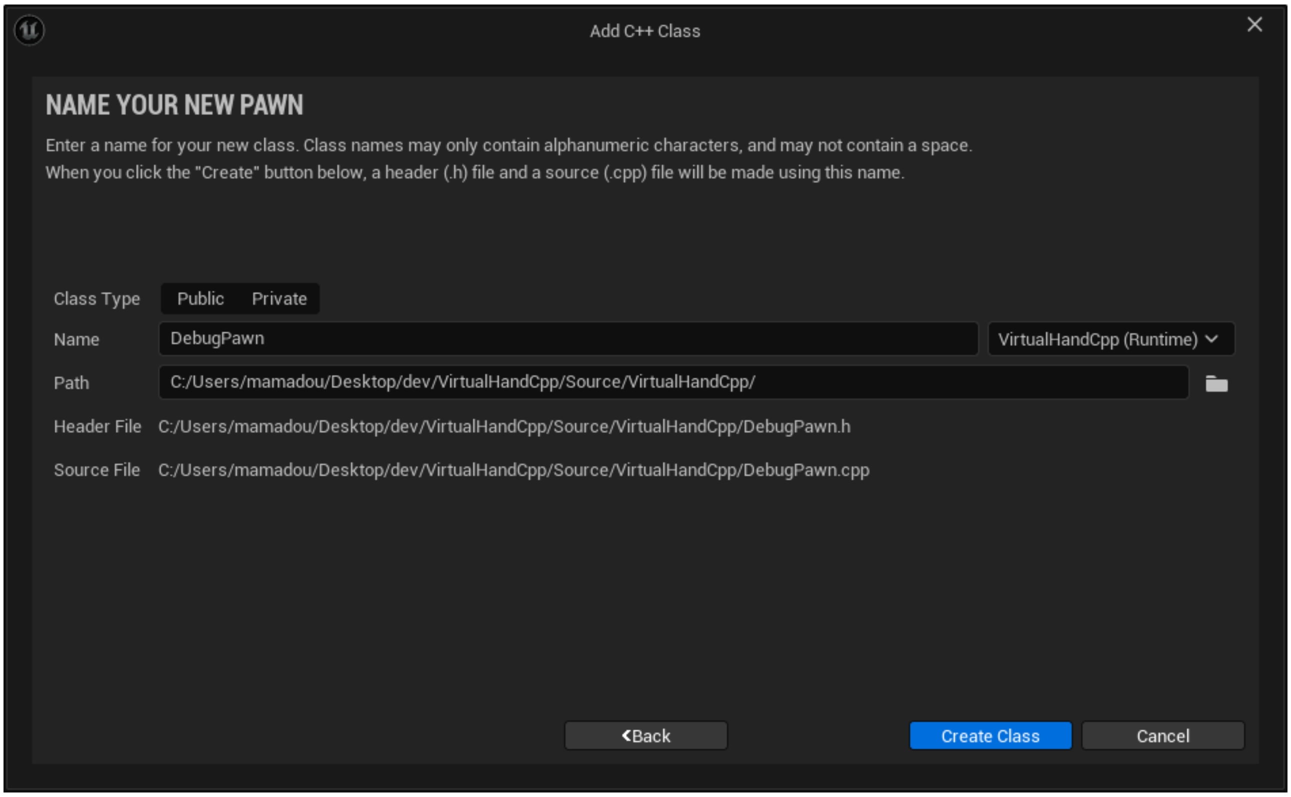
3. You could use either hand-tracking or a SenseGlove device as the input data, or both of the inside the same project. Whether you would like to use hand-tracking or a SenseGlove device, please make sure the required steps are taken for each of those first.

4. From the `Tools` menu choose `New C++ class...`.



Adds C++ code to the project. The code can only be compiled if you have Rider Uproject installed.

5. Choose the Unreal Engine's `APawn` class as the parent class for the new C++ pawn class.

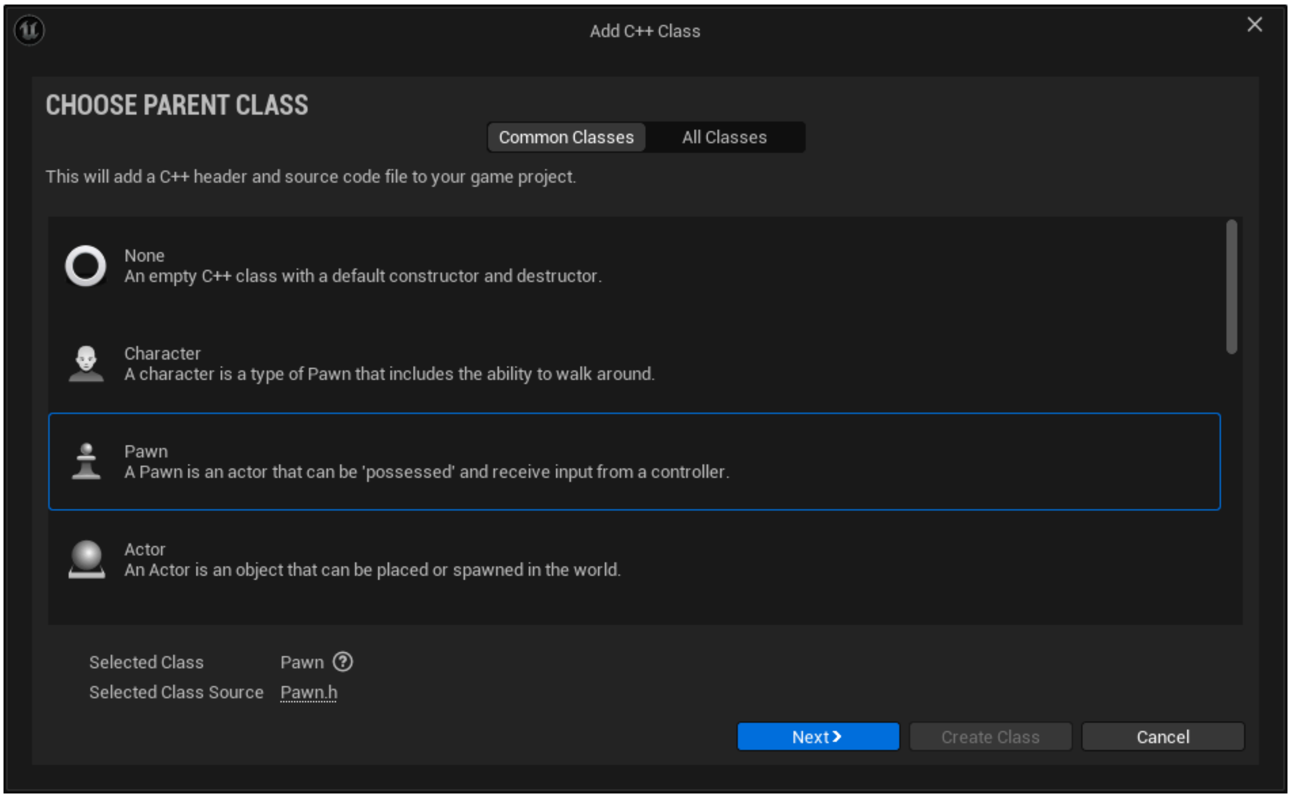

Add C++ Class

NAME YOUR NEW PAWN

Enter a name for your new class. Class names may only contain alphanumeric characters, and may not contain a space.
When you click the "Create" button below, a header (.h) file and a source (.cpp) file will be made using this name.

Class Type	Public Private
Name	DebugPawn — VirtualHandCpp (Runtime)
Path	C:/Users/mamadou/Desktop/dev/VirtualHandCpp/Source/VirtualHandCpp/
Header File	C:/Users/mamadou/Desktop/dev/VirtualHandCpp/Source/VirtualHandCpp/DebugPawn.h
Source File	C:/Users/mamadou/Desktop/dev/VirtualHandCpp/Source/VirtualHandCpp/DebugPawn.cpp

6. Name the new pawn class `DebugPawn`.



Add C++ Class

CHOOSE PARENT CLASS

This will add a C++ header and source code file to your game project.

None
An empty C++ class with a default constructor and destructor.

Character
A character is a type of Pawn that includes the ability to walk around.

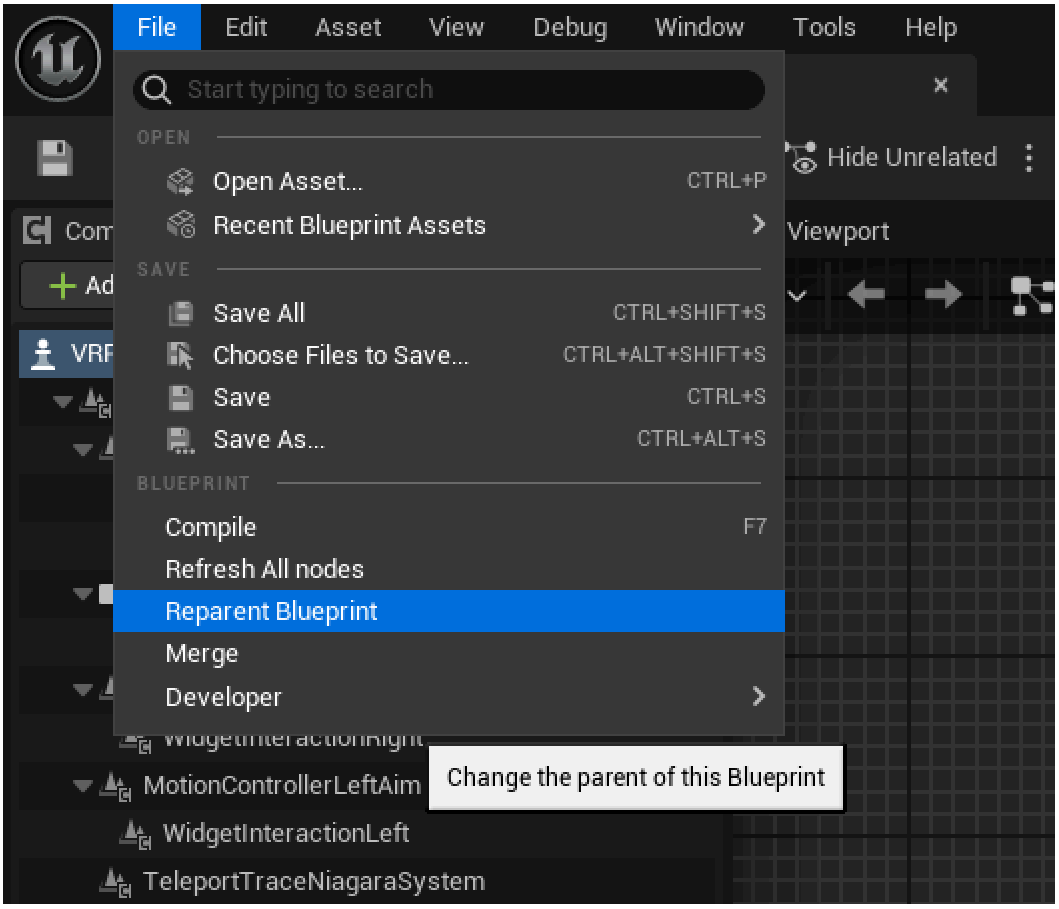
Pawn
A Pawn is an actor that can be 'possessed' and receive input from a controller.

Actor
An Actor is an object that can be placed or spawned in the world.

Selected Class Pawn
Selected Class Source Pawn.h

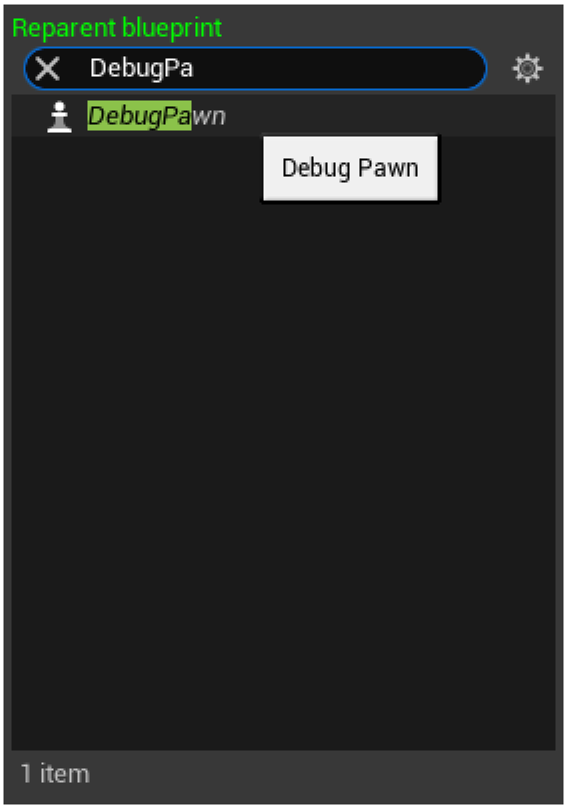
7. Since we have created a new C++ class, this converts the current Blueprint VRTemplateMap project to a C++ one. That's why the Unreal Editor will give us a few prompts regarding opening the project in the default IDE and rebuilding the code. It might be simpler to just close the editor, then rebuild the source code inside your favorite IDE, and then start the editor with the converted project again.

8. Find and open the VRPawn Blueprint Class located at `/Content/VRTemplate/Blueprints/VRPawn` inside the Blueprint Editor and from the `File` menu choose the `Reparent Blueprint` class.

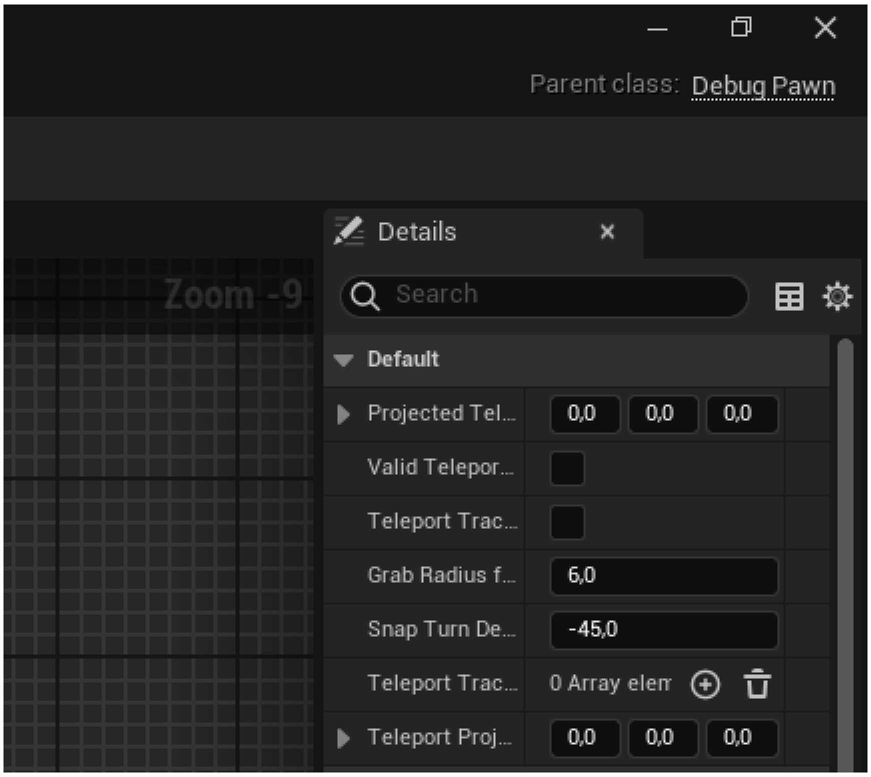


Change the parent of this Blueprint

9. In the new `Reparent blueprint` window choose `DebugPawn` as the new parent.



10. By looking at the `Parent Class` label located under the Blueprint Editor window control buttons verify that the `ADebugPawn` class has been set as the new parent.


Parent class: Debug Pawn

Details

Default

Projected Tel...	0,0	0,0	0,0
Valid Telepor...	☐		
Teleport Trac...	☐		
Grab Radius f...	6,0		
Snap Turn De...	-45,0		
Teleport Trac...	0 Array elem		
Teleport Proj...	0,0	0,0	0,0

11. Locate the project's main Build file, in our case `VirtualHandCpp/Source/VirtualHandCpp/VirtualHandCpp.Build.cs` and add the `InputDevice`, `OpenXRHMD`, `SenseGloveBuildHacks`, `SenseGloveDebug`, `SenseGloveSettings`, and `SenseGloveTracking` modules as either a private or public dependency.

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

using UnrealBuildTool;

public class VirtualHandCpp : ModuleRules
{
    public VirtualHandCpp(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicDependencyModuleNames.AddRange(new string[] { "Core",
"CoreUObject", "Engine", "InputCore" });

        PrivateDependencyModuleNames.AddRange(new string[]
        {
            "InputDevice",
            "OpenXRHMD",
            "SenseGloveBuildHacks",
            "SenseGloveDebug",
            "SenseGloveSettings",
            "SenseGloveTracking"
        });

        // Uncomment if you are using Slate UI
        // PrivateDependencyModuleNames.AddRange(new string[] { "Slate",
"SlateCore" });

        // Uncomment if you are using online features
        // PrivateDependencyModuleNames.Add("OnlineSubsystem");

        // To include OnlineSubsystemSteam, add it to the plugins section in
your uproject file with the Enabled attribute set to true
    }
}
```

12. Locate the C++ header and source file for the `ADebugPawn` inside the project in your C++ IDE. In our case they are located at `VirtualHandCpp/Source/VirtualHandCpp/DebugPawn.h` and `VirtualHandCpp/Source/VirtualHandCpp/DebugPawn.cpp` .

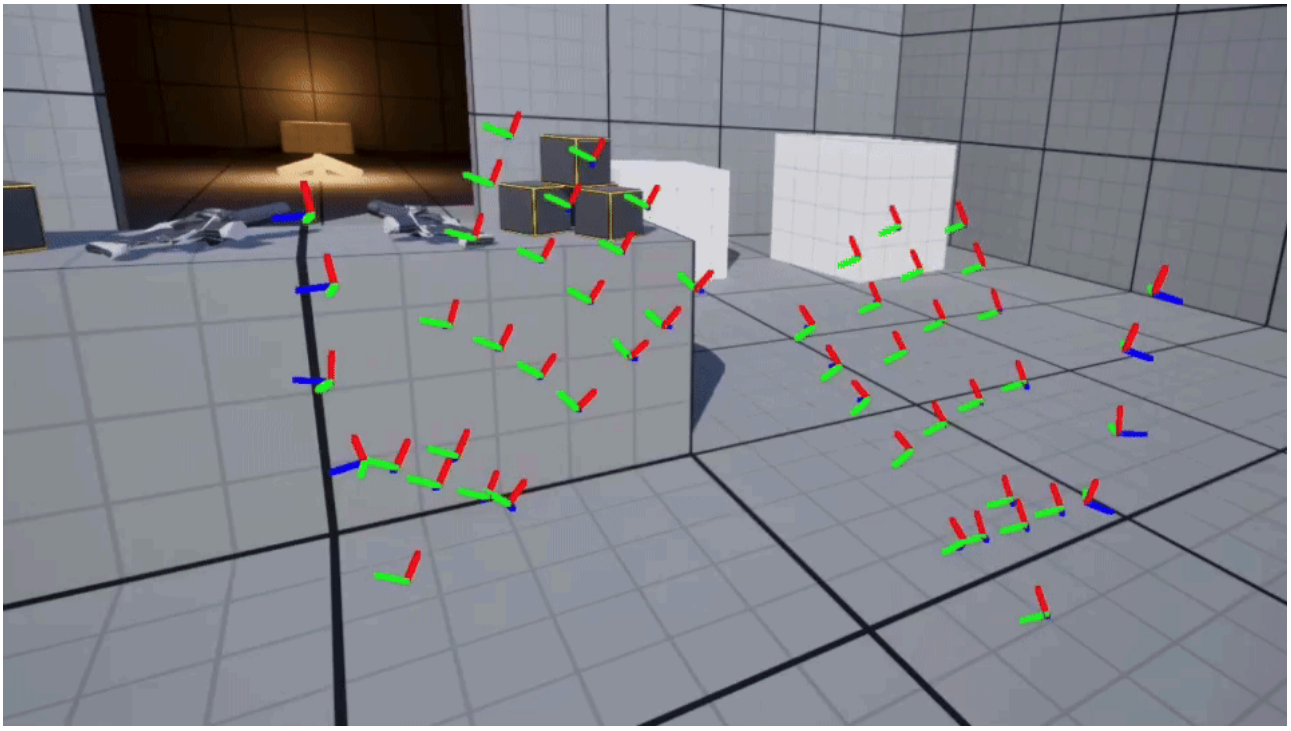
13. Modify the `DebugPawn.h` header file to look like this:

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

#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Pawn.h"

#include "SGSettings/SGDebugGizmoSettings.h"

#include "DebugPawn.generated.h"

UCLASS()
class VIRTUALHANDCPP_API ADebugPawn : public APawn
{
    GENERATED_BODY()

private:
    // The virtual hand drawing settings.
    UPROPERTY(EditDefaultsOnly, Category="DebugPawn",
        meta=(AllowPrivateAccess="false"))
    FSGDebugGizmoSettings HandDrawingSettings;

public:
    // Sets default values for this pawn's properties
    ADebugPawn();

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;

public:
    // Called every frame
    virtual void Tick(float DeltaTime) override;

    // Called to bind functionality to input
    virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override;

private:
    // The method responsible for drawing a virtual hand.
    void DrawHand(EControllerHand Hand) const;
};
```

14. Modify the `DebugPawn.cpp` implementation file to look like this:

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


#include "DebugPawn.h"

#include "SGDebug/SGDebugGizmo.h"
#include "SGTracking/SGXRTracker.h"

// Sets default values
ADebugPawn::ADebugPawn()
{
    // Set this pawn to call Tick() every frame.  You can turn this off to
improve performance if you don't need it.
    PrimaryActorTick.bCanEverTick = true;

    // Set the default virtual hand drawing settings.
    HandDrawingSettings = FSGDebugGizmoSettings{
        1.0f,
        FColor{255, 0, 0, 255},
        FColor{0, 255, 0, 255},
        FColor{0, 0, 255, 255},
        false,
        1.1f,
        0,
        0.2f,
    };
}

// Called when the game starts or when spawned
void ADebugPawn::BeginPlay()
{
    Super::BeginPlay();
}

// Called every frame
void ADebugPawn::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    // Attempt at drawing the left/right virtual hands every frame.
    DrawHand(EControllerHand::Left);
    DrawHand(EControllerHand::Right);
}

// Called to bind functionality to input
void ADebugPawn::SetupPlayerInputComponent(UInputComponent*
PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);
}

void ADebugPawn::DrawHand(const EControllerHand Hand) const
{
    // Get the world and cache it, if it's null we return early.
    UWorld* World{GetWorld()};
    if (!IsValid(World))
    {
        return;
    }

    FXRHandTrackingState HandTrackingState;
    const bool bGotHandTrackingState = FSGXRTracker::GetHandTrackingState(
        World, EXRSpaceType::UnrealWorldSpace, Hand, HandTrackingState);

    // Return if the struct data is invalid!
    if (!bGotHandTrackingState || !HandTrackingState.bValid)
    {
        return;
    }

    // Return if the device is not being tracked!
    if (HandTrackingState.TrackingStatus == ETrackingStatus::NotTracked)
    {
        return;
    }

    // Ensure that HandTrackingState.HandKeyLocations has the location data
    // for 26 joints!
    if (!ensureAlwaysMsgf(HandTrackingState.HandKeyLocations.Num()
                          == EHandKeypointCount,
                          TEXT("Invalid HandKeyLocations count!")))
    {
        return;
    }

    // Ensure that HandTrackingState.HandKeyRotations has the rotation data
    // for 26 joints!
    if (!ensureAlwaysMsgf(HandTrackingState.HandKeyRotations.Num()
                          == EHandKeypointCount,
                          TEXT("Invalid HandKeyRotations count!")))
    {
        return;
    }
    // Iterate over the hand joint locations and rotations!
    for (int32 JointIndex = 0; JointIndex < EHandKeypointCount; ++JointIndex)
    {
        const FVector& JointLocation{
            HandTrackingState.HandKeyLocations[JointIndex]
        };
        const FQuat& JointRotation{
            HandTrackingState.HandKeyRotations[JointIndex]
        };

        // Draw a single joint's gizmo!
        // Please note that we could alternatively:
        // Use FSGDebugCube::Draw() to draw a cube.
        // Or use the FSGDebugVirtualHand::Draw() method and pass the
        // HandTrackingState directly to draw the virtual hand
        // all at once without iterating the joints. But, that's not
        // goal of this tutorial.
        FSGDebugGizmo::Draw(World, JointLocation, JointRotation,
HandDrawingSettings);
    }
}
```

15. Now, rebuild the source code and go back to the `VRTemplateMap`, then use the VR Preview button to run the game. If everything's done correctly, you should be able to see the virtual hands inside your VR simulation.



Third-Party OpenXR Integrations

The **SenseGlove Unreal Engine Plugin** registers itself as an `OpenXRHandTracking` provider, making it a fully compatible, drop-in replacement for Epic's own **OpenXRHandTracking** plugin in Unreal Engine. This allows it to integrate seamlessly with any third-party system or plugin that can consume OpenXR hand-tracking data.

One notable example is the open-source, MIT-licensed VR Expansion Plugin (VRE).

> **Important**
>
> As explained in the Third-Party Tutorials: Consuming OpenXR Hand-Tracking Data section, it's entirely possible to build your own custom hand interaction system without relying on SGPawn or any third-party OpenXR-compatible interaction plugin altogether.
>
> If your project requires finer-grained control than what these solutions offer, the tutorials in that section will guide you through understanding the OpenXR hand-tracking data format in Unreal Engine and help you implement a fully tailored interaction system from the ground up in a few hours.

As the SenseGlove Unreal Engine Plugin is fully OpenXR-compliant, it provides OpenXR hand-tracking data in the expected format and takes over as the active provider within Unreal. If your existing interaction system (e.g. VRE plugin) already uses OpenXR hand-tracking, SenseGlove will function as a direct tracking source instead of a real hand.

> **Note**
>
> Since most hand-tracking systems are not capable of haptics feedback, integrating SenseGlove's haptic feedback requires a small amount of additional effort.
>
> The SenseGlove API is fully exposed to Unreal Engine via C++ and Blueprint, so triggering haptic feedback is as simple as calling a function. For more information, refer to the Blueprint Changes section below.

Important

If you're using a third-party OpenXR hand interaction system, configuring the Wrist-Tracker Hardware Settings will likely have no effect, and your hand offsets may appear at the incorrect location in the scene.

This is because those settings are only recognized by SenseGlove's native actors and components such as `SGPawn`, `SGWristTrackerComponent`, etc. Most third-party plugins are unaware of these settings. As a result, you'll need to figure out how to manually apply the appropriate offsets within your chosen OpenXR hand interaction system.

For example, the VRE plugin provides similar configurations in their plugin's settings section. For more information refer to the Changing Wrist-Tracker Offsets section below.

Comparison of Supported OpenXR Hand-Interaction Systems

	Built-in?	Works out of the box?	Beginner-friendly?	Learning Curve
SGPawn	✅ Yes	✅ Yes	✅ Most beginer-friendly	✅ Very eas
SenseGlove OpenXR	✅ Yes	❌ Requires Blueprint or C++ coding	✅ Requires a few hours of watching tutotrials	✅ Moderat

	Built-in?	Works out of the box?	Beginner-friendly?	Learning Curve
VR Expansion Plugin	❌ No	⚠️ Partially – requires setup	❌ Best suited for intermediate or advanced users	⚠️ Steep
Other OpenXR-compatible Plugins	❌ No	❓ Check their documentation	❓ Check their documentation	❓ Check th documentat

VR Expansion Plugin

The **VR Expansion Plugin (VRE)** is a robust, community-driven plugin for Unreal Engine that focuses on advanced VR interaction and gameplay mechanics. It is open-source (MIT licensed), actively maintained, and has received support from Epic via the MegaGrants program.

Designed to extend Unreal's capabilities for virtual reality, VRE offers a modular set of tools covering:

- Multiplayer and networking
- Locomotion systems
- Object gripping and interaction
- Custom movement and physics handling

The plugin is particularly useful for teams building sophisticated VR experiences. While it's beginner-friendly to an extent, its depth and flexibility are **best suited for intermediate to advanced Unreal Engine developers**. Whether you're prototyping with built-in features or extracting specific systems for your own framework, VRE offers a rich foundation for VR development.

ⓘ Note

> For support and assistance with the VRE plugin, you can join its active and welcoming Discord community, known for being responsive and supportive.

SGVRETemplate Demo Scene

To showcase how SenseGlove can be integrated with OpenXR-compatible third-party interaction systems, SenseGlove provides a ready-to-use VR Expansion Plugin Integration Demo for Unreal Engine 5.4.

This repository includes UE `5.4` –compatible versions of both the **SenseGlove** and **VR Expansion** plugins, with all necessary setup and configuration already in place. Simply download the project and it should run out of the box, allowing you to explore the integration without additional setup.

> Note
>
> SenseGlove provides this demo to demonstrate the potential for integrating with third-party OpenXR-based hand interaction systems. Please note that the **VR Expansion Plugin** is a third-party solution, and as such, **we do not offer official support for it**.
>
> For help with the VRE plugin, refer to its documentation at **vreue4.com** and consider joining the **official VRE Discord community**, which is active, supportive, and very responsive.

SGVRETemplate Modifications

The **SGVRETemplate** is built on top of the VR Expansion Plugin Example Template. However, since the original template is not directly compatible with SenseGlove, several adjustments were necessary.

In addition, a few known issues with OpenXR support in the VR Expansion Plugin for Unreal Engine `5.4` required us to modify the plugin itself to ensure smooth integration.

Below is an overview of the key modifications made to both the project template and this version of the VRE plugin.

Blueprint Changes

- **Content/VRE/Core/Character/BP_VRCharacter**: Four functions were added: `SendVibration`, `SendFFB`, `SendSqueeze`, and `ResetHaptics`. These functions retrieve the glove instance and send the appropriate haptic command to it. In the `OnPossessed` event, `Load Controller by Name` was added along with a string uproperty `Tracking Offset`, which is used to load the correct tracking offsets based on the selected profile.

> **Note**
>
> If you'd like to implement your own haptic functions, the most convenient approach is to safely acquire a glove instance. Once you have the glove instance, applying haptic feedback is as simple as calling the appropriate function.
>
> SenseGlove supports three types of haptics: Vibrations, Force-feedback, and Wrist-squeeze.
>
> - Using `Send Custom Waveform`, you can send vibrations to the glove instance.
> - Using `Queue Command Force Feedback Levels`, you can send force-feedback.
> - Using `Queue Command Wrist Squeeze`, you can send a wrist-squeeze command to the glove.
>
> Each of these functions can be called directly on the glove instance to trigger the desired haptic feedback.

- **Content/VRE/Core/GraspingHands/GraspingHandManny**: In the `SetupFingerAnimations` function, replace the hardcoded check for `HandType == Left` with a string comparison: convert the enum to a string and check if it contains `"Left"`. This allows compatibility with alternative tracking sources such as `"Left Foot"`.

C++ Changes

- **Plugins/VRExpansionPlugin/Source/VRExpansionPlugin/Public/Grippables/HandSocketComponent.h**: The following line was added as a public `UPROPERTY` in the header file:

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Hand Animation")
float HandAnimationProgress = 0.0f;
```

- **Plugins/VRExpansionPlugin/Source/VRExpansionPlugin/Private/Grippables/HandSocketComponent.cpp**: In the function `bool UHandSocketComponent::GetBlendedPoseSnapShot(FPoseSnapshot& PoseSnapShot, USkeletalMeshComponent* TargetMesh, bool bSkipRootBone, bool bFlipHand)`, the `TrackLocation` calculation was modified from:

```
if (TrackIndex != INDEX_NONE && (!bSkipRootBone || TrackIndex != 0))
{
    double TrackLocation = 0.0f;
    HandTargetAnimation->GetBoneTransform(LocalTransform,
FSkeletonPoseBoneIndex(TrackMap[TrackIndex].BoneTreeIndex), TrackLocation,
false);
}
else
{
```

To:

```
if (TrackIndex != INDEX_NONE && (!bSkipRootBone || TrackIndex != 0))
{
    double TrackLocation = HandTargetAnimation->GetPlayLength() *
HandAnimationProgress;
    HandTargetAnimation->GetBoneTransform(LocalTransform,
FSkeletonPoseBoneIndex(TrackMap[TrackIndex].BoneTreeIndex), TrackLocation,
false);
}
else
{
```

- **Plugins/VRExpansionPlugin/Source/VRExpansionPlugin/Private/GripMotionControllerComponent.cpp**: In the function `void`

`UGripMotionControllerComponent::GetCurrentProfileTransform(bool bBindToNoticationDelegate)` , the following logic was updated from:

```
if (HandType == EControllerHand::Left || HandType == EControllerHand::AnyHand
|| !VRSettings->bUseSeperateHandTransforms)
{
    NewControllerProfileTransform = VRSettings-
>CurrentControllerProfileTransform;
}
else if (HandType == EControllerHand::Right)
{
    NewControllerProfileTransform = VRSettings-
>CurrentControllerProfileTransformRight;
}
```

To:

```
if (UEnum::GetDisplayValueAsText(HandType).ToString().Contains("Left") ||
HandType == EControllerHand::AnyHand || !VRSettings-
>bUseSeperateHandTransforms)
{
    NewControllerProfileTransform = VRSettings-
>CurrentControllerProfileTransform;
}
else if (UEnum::GetDisplayValueAsText(HandType).ToString().Contains("Right"))
{
    NewControllerProfileTransform = VRSettings-
>CurrentControllerProfileTransformRight;
}
```

The following function was also updated; from:

```
void UGripMotionControllerComponent::GetHandType(EControllerHand& Hand)
{
    if (!IMotionController::GetHandEnumForSourceName(MotionSource, Hand))
    {
        // Check if the palm motion source extension is being used
        // I assume eventually epic will handle this case
        if (MotionSource.Compare(FName(TEXT("RightPalm"))) == 0 ||
MotionSource.Compare(FName(TEXT("RightWrist"))) == 0)
        {
            Hand = EControllerHand::Right;
        }
        // Could skip this and default to left now but would rather check
        else if (MotionSource.Compare(FName(TEXT("LeftPalm"))) == 0 ||
MotionSource.Compare(FName(TEXT("LeftWrist"))) == 0)
        {
            Hand = EControllerHand::Left;
        }
        else
        {
            Hand = EControllerHand::Left;
        }
    }
}
```

To:

```
void UGripMotionControllerComponent::GetHandType(EControllerHand& Hand)
{
    if (!IMotionController::GetHandEnumForSourceName(MotionSource, Hand))
    {
        // Check if the palm motion source extension is being used
        // I assume eventually epic will handle this case
        if (MotionSource.Compare(FName(TEXT("RightPalm"))) == 0 ||
MotionSource.Compare(FName(TEXT("RightWrist"))) == 0 ||
MotionSource.ToString().Contains("Right"))
        {
            Hand = EControllerHand::Right;
        }
        // Could skip this and default to left now but would rather check
        else if (MotionSource.Compare(FName(TEXT("LeftPalm"))) == 0 ||
MotionSource.Compare(FName(TEXT("LeftWrist"))) == 0 ||
MotionSource.ToString().Contains("Left"))
        {
            Hand = EControllerHand::Left;
        }
        else
        {
            Hand = EControllerHand::Left;
        }
    }
}
```

Changing Wrist-Tracker Offsets

If you are using wrist-tracking hardware supported by the SenseGlove plugin, you can change the offsets inside `BP_VRCharacter` using the uproperty `Tracking Offset` typing or copying any of the following, depending on your hardware:

- **SenseGlove_Quest3**: The wrist-tracking controller profile for for Meta Quest3.
- **SenseGlove_ViveWristTrackers**: The wrist-tracking controller profile for HTC VIVE wrist trackers.

Changing Motion Source

In `BP_VRCharacter`, you can change the wrist-tracking motion source for each hand. This is required depending on which tracker you are using.

Adding More Gestures

In the `GraspingHandManny` Blueprint, we've created a simple function called `SaveHandPose`. If you press the `Space Bar` while the game is running, it will save the current pose of the corresponding hand. The pose is stored in a gestures database located under `Content/SenseGlove` with the default name `NewHandPose`. You should rename the pose to something meaningful when you intend to use it.

It's helpful to add an Event Dispatcher to the `GraspingHandManny` Blueprint, which is triggered in the Event Graph by the `On New Gesture Detected` event from the `OpenXRHandPose` component. This system is index-based rather than name-based, so keep that in mind when adding more dispatchers. By default, we've included examples for `Teleport`, `Grab`, `Release`, and `Use`.

Video Summary

This short video provides an overview of some of the key changes and modifications behind the SGVRETemplate demo scene, mentioned above.

SGVRETemplate Demo Calibration Scene

The SGVRETemplate includes a basic Calibration Scene located at `Content/SenseGlove/Maps/Calibration`. Inside this level, you'll find a copy of `Content/SenseGlove/Blueprints/Calibration/BP_Calibrator` Blueprint responsible for transitioning to your desired target map after the calibration process is complete. You can configure the target map directly within this Blueprint by adjusting the `Level to Load` uproperty.

Third-Party Tutorials: Consuming OpenXR Hand-Tracking Data

Introduction to Virtual Reality, OpenXR Hand-Tracking, and Gesture Detection in Unreal Engine

In this tutorial, you'll learn how to get the OpenXR Hand-Tracking Data from the Unreal Engine API and how to consume it to draw virtual hand models using cubes (as hand joints). It will also dive into gesture recognition by implementing a simple pinch gesture recognition.

In the first part, it will focus on UE `4.26` to `5.4` API. And, in the second part, you'll learn how to update the project to work with `5.5`.

Procedural Virtual Hand Mesh Animation Using OpenXR Hand-Tracking Data

Building on the Introduction to Virtual Reality, OpenXR Hand-Tracking, and Gesture Recognition in Unreal Engine tutorial, this slightly more advanced tutorial will dive deeper into the following topics:

- Transitioning seamlessly between motion controller and hand-tracking modes in Unreal Engine.
- Adding custom debugging gizmos to improve development and testing workflows.
- Visualizing debug virtual hands by incorporating the custom gizmos.
- Animating virtual hand meshes with OpenXR hand-tracking data, moving beyond basic joint representation with cubes.
- Re-using and adapting the gesture recognition code from the introductory tutorial to integrate with the new animated virtual hand meshes. This guide will help you take your VR projects to the next level with polished and practical implementations.

Part 1:

Part 2:

Low-level Blueprint API

Unfortunately, due to Unreal Engine's limited availability of automated documentation generation tools, there is no updated online documentation for the SenseGlove Blueprint API. However, this does not mean that no documentation is available. In fact, most of the Blueprint code is already documented within the relevant header files. Any modules with the `Kismet` postfix in the name contain the Blueprint documentation. For example, the Blueprint documentation for the `Core` module can be found inside the `Source/SenseGloveCoreKismet/Public/SGCoreKismet` directory.

There is also an outdated Blueprint documentation hosted on GitLab. This documentation was generated for the early releases of the plugin using kamrann/KantanDocGenPlugin and kamrann/KantanDocGenTool, which is no longer maintained.

Efforts are ongoing to generate comprehensive documentation using PsichiX/unreal-doc, but progress has been hindered by various known issues.

There are also other outdated materials that might still be partially relevant. These include an example Unreal Engine Blueprint project and a video tutorial:

Low-level C++ API

Due to Unreal Engine's limited availability of automated documentation generation tools, there is no updated online documentation for the SenseGlove Unreal Engine C++ API. However, this does not mean that no documentation is available. A significant portion of the API is documented within the relevant header files. For example, the C++ API documentation for the `Core` module can be found inside the `Source/SenseGloveCore/Public/SGCore` directory.

Efforts are ongoing to generate comprehensive documentation using PsichiX/unreal-doc, but progress has been hindered by various known issues.

Nevertheless, since this plugin builds on top of the SGConnect and SGCoreCpp third-party C++ libraries, the upstream documentation provides detailed information on various aspects of the underlying SenseGlove C++ API.

There are also other outdated materials that might still be partially relevant. These include an example Unreal Engine C++ project and a video tutorial:

Platform Support Matrix

Platform Support Matrix

	Windows (MSVC 2017)	Windows (MSVC 2019)	Windows (MSVC 2022)	Linux x86-64 (Native Toolchain)	Linux AArch6 (Native Toolchai
5.7	❌	❌	✅ v2.8.x	✅ v2.8.x	✅ v2.8.
5.6	❌	❌	✅ v2.8.x	✅ v2.8.x	✅ v2.8.
5.5	❌	❌	✅ v2.8.x	✅ v2.8.x	✅ v2.8.
5.4	❌	❌	⚠️ v2.7.x	⚠️ v2.7.x	⚠️ v2.7.
5.3	❌	⚠️ v2.6.x	⚠️ v2.6.x	⚠️ v2.6.x	⚠️ v2.6.
5.2	❌	⚠️ v2.4.x	⚠️ v2.4.x	⚠️ v2.4.x	⚠️ v2.4.
5.1	❌	⚠️ v2.0.x	⚠️ v2.0.x	⚠️ v2.0.x	⚠️ v2.0.
5.0	❌	⚠️ v1.6.x	⚠️ v1.6.x	⚠️ v1.6.x	⚠️ v1.6.
4.27	⚠️ v1.4.x	⚠️ v1.4.x	⚠️ v1.4.x	⚠️ v1.4.x	⚠️ v1.4.
4.26	⚠️ v1.0.x	⚠️ v1.0.x	❌	⚠️ v1.0.x	❌
4.25	⚠️ v1.0.x	⚠️ v1.0.x	❌	⚠️ v1.0.x	❌

	Windows (MSVC 2017)	Windows (MSVC 2019)	Windows (MSVC 2022)	Linux x86-64 (Native Toolchain)	Linux AArch6 (Native Toolchai
4.24	⚠️ v1.0.x	⚠️ v1.0.x	❌	⚠️ v1.0.x	❌
4.23	⚠️ v1.0.x	⚠️ v1.0.x	❌	⚠️ v1.0.x	❌
4.22	⚠️ v1.0.x	⚠️ v1.0.x	❌	⚠️ v1.0.x	❌

- ✅ Supported
- ⚠️ End-of-life (EOL) release that is not supported anymore and might be lacking features
- ❌ Not supported at all
- ❓ Unknown or untested

Remarks:

- Per Epic's Marketplace Guidelines in regards to Code Plugins (sections `2.6.3`.d and `3.1.b`), we are only able to distribute or update the SenseGlove plugin for the last `3` stable versions of Unreal Engine. As a result, we won't be able to publish updates or bug fixes for the older versions of the Engine except on rare occasions and only through our official repository on Microsoft Azure DevOps.
- All third-party libraries on Windows built against Windows SDK `10.0`.
- Oculus and VIVE support is only provided through the recommended Android NDK versions by Epic Games.
- wjwwood/serial requires Android NDK API Level `28+` in order to be built successfully.
- All third-party libraries target Android NDK API Level `29`, thus any project relying on the plug-in should be build with the same NDK API Level.
- Meta Quest PCVR-mode support is only provided through the Epic OpenXR plugin. Furthermore, the Standalone-mode support is also provided through the Epic OpenXR plugin only. Please note that enabling the Meta XR plugin will

result in crashes or unexpected behavior. Meta XR plugin compatibility is being worked on at the moment and might be supported in the future.

- While HTC VIVE PCVR-mode support is only provided through the Epic OpenXR plugin, the standalone-mode support is only provided through the official HTC `ViveOpenXR` plugin and no other plugin is supported.

Planned Features Completion Status

Planned Features Completion Status

Implemented as of v2.8.x

- ✅ SenseGlove Unreal Engine Handbook, a comprehensive guide to the SenseGlove Unreal Engine Plugin accessible online via unreal.docs.senseglove.com, and also in PDF and EPUB formats.
- ✅ Full SenseGlove low-level core API access through Unreal C++.
- ✅ Full SenseGlove low-level core API access through Blueprint.
- ✅ DK 1 Support.
- ✅ Nova 1 Support.
- ✅ Nova 2 Support.
- ✅ Support for Microsoft Windows as a development platform.
- ✅ Support for GNU/Linux as a development platform.
- ✅ Support for Microsoft Windows as a deployment platform.
- ✅ Support for GNU/Linux x64 as a deployment platform.
- ✅ Support for GNU/Linux AArch64 as a deployment platform.
- ✅ Support for Android as a deployment platform.
- ✅ Support for Oculus Quest 2, Oculus Quest Pro, Oculus Quest 3, and Oculus Quest 3s.
- ✅ Support for HTC VIVE Pro, HTC VIVE Focus 3, HTC XR Elite, and HTC VIVE Focus Vision.
- ✅ Support for HTC VIVE Trackers and HTC VIVE Wrist Trackers.
- ✅ Support for both Bluetooth Serial and Bluetooth Low Energy.
- ✅ On-device calibration for Android without the need for SenseCom.
- ✅ Haptic feedback including force feedback, buzz, and thumper commands.
- ✅ A customizable Grab component that could be added to any actor.
- ✅ A customizable Touch component that could be added to any actor.
- ✅ Ability to grab, release, and throw objects around.
- ✅ Separation of the real and virtual hand rendering.

- ✅ An out-of-the-box customizable SGPawn with the ability to be extended in C++ and Blueprint.
- ✅ Easy wrist/hand tracking debugging using the SenseGlove Debug module.
- ✅ A generic Settings module with the ability to override settings.
- ✅ C++/Blueprint interaction events such as OnGrabStateUpdated, OnTouchStateUpdated, OnActorGrabbed, OnActorReleased, OnActorBeginTouch, and OnActorEndTouch.
- ✅ A fall back to HMD and wrist tracker hardware auto-detection mechanism when automatic detection of the wrist tracker hardware is desired.
- ✅ OpenXR-compatible hand tracking (XR_EXT_hand_tracking) support.
- ✅ `FXRMotionControllerData` compatible hand animation system on UE versions `5.2`, `5.3`, and `5.4`.
- ✅ `FXRHandTrackingState` compatible hand animation system on UE versions `5.5+`.
- ✅ `FXRMotionControllerData` compatible wrist tracking system on UE versions `5.2`, `5.3`, and `5.4`.
- ✅ `FXRHandTrackingState` compatible wrist tracking system on UE versions `5.5+`.
- ✅ `FXRMotionControllerData` compatible hand interaction manipulation system on UE versions `5.2`, `5.3`, and `5.4`.
- ✅ `FXRHandTrackingState` compatible hand interaction manipulation system on UE versions `5.5+`.
- ✅ Ability to fallback to hand tracking when a glove is not present and use the bare hands for interactions, or a combination of glove and hand tracking if no motion controller input is detected.
- ✅ The SenseGlove grab/touch sockets one-click-setup ability on any Epic-compliant virtual hand mesh from within the Unreal Editor's Content Browser, Skeleton Editor, or Skeletal Mesh Editor.
- ✅ A flexible virtual hand animation system that can take the mesh bone's transforms into account for a more reliable hand animation.
- ✅ Ability to manage the Engine Scalability Settings through the SenseGlove plugin in order to change the graphics settings on the fly.
- ✅ Ability to automatically ask for the required permissions on Android when the plugin is enabled in any UE project.
- ✅ `ViveOpenXR` plugin compatibility.

- ✅ A highly convenient OpenXR hand-tracking data provider component to easily feed SenseGlove data to your own custom or third-party hand-manipulation systems.
- ✅ A highly convenient haptics component to easily add haptics to your own custom or third-party hand-manipulation systems.

Upcoming features planned for the v2.9.x release

Planned features long-term

- ⬛ Get tracking input from sources other than a SenseGlove device.
- ⬛ Be able to assign behaviors to different objects (meshes) in the scene (e.g. Slider, Hinge, basic Grabables, etc).
- ⬛ Make it so developers can define or extend their own behavior(s) to an object through Code / Blueprint (e.g. I want a car door that is like a slider, but follows a path rather than a straight line).
- ⬛ Make the hand(s) able to push around physics-driven objects (for as much as their behaviors allow) (in backlog).
- ⬛ Be able to grab objects with up to 2 hands (and move them around with both hands at the same time in a way that seems realistic).
- ⬛ Ensure that our virtual hands (and the objects they hold) do not phase through other physics objects (e.g. walls and tables).
- ⬛ Allow other scripts to force a grab and/or release to occur (for example, when you place it apart at the designated location, it gets removed from your hand and snaps into place).
- ⬛ Have some form of weight simulation by making certain objects harder to push, lowering manipulation speed, or making objects only moveable with two hands.
- ⬛ (Optional) Make it so the fingers of your virtual hands do not clip inside the meshes you are holding (certain people see this as an indicator of how fast the Force-Feedback activates - but it's basically just rendering).

Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[2.8.0] - 2026-02-24

This minor release introduces high-level haptics and hand-tracking components, simplifies OpenXR integration around `FXRHandTrackingState`, and removes legacy `FXRMotionControllerData` APIs. It also includes ABI-breaking changes, Unreal Engine deprecations, documentation expansions, and various internal improvements.

Added

- Added a `USGHapticsComponent` to allow sending a variety of haptic feedbacks (force-feedback, vibrotactile, custom waveforms, and wrist-squeeze) to the gloves without touching SenseGlove's low-level API.
- Added a `USGHandTrackerComponent` to allow retrieval or visualization of `FXRHandTrackingState` without relying on low-level SenseGlove API or UE's generic `GetHandTrackingState()` functionality. This is useful when developing a custom hand-interaction system using SenseGlove/UE OpenXR API or interfacing with third-party OpenXR-compatible plugins such as VR Expansion Plugin (VRE). Using this component removes the need to calculate the wrist offsets manually, or an extra call to `USGHapticGlove::GetWristLocation()`, in comparison to when the `FXRHandTrackingState` is retrieved via UE's `GetHandTrackingState()`.
- A new `FSGDebugVirtualHand::Draw()` overload has been added to allow visualizing `FSGDebugGizmoSettings` directly. This is used internally by the new `USGHandTrackerComponent` to visualize its `FXRHandTrackingState` if `bVisualize` is enabled.

- Added a new `FSGHandLayer::GetWristLocation()` overload to allow passing `FRotator` s instead of `FQuat` s as input or output parameters.

Fixed

- Fix some copyright notices with wrong copyright owner. These propably has happend during bulk replaces with class or struct names.
- Additional minor fixes and improvements that may not be listed here.

Changed

- The rename of `SGDeviceList::SenseCommRunning()` to `SGDeviceList::SenseComRunning()` — previously listed as part of the v2.7.0 release — was not actually included in the `master` branch due to a missed commit during cherry-picking from `dev` to `master` . Since Microsoft Azure DevOps Repositories tags are always created from `master` , the `v2.7.0` , `v2.7.1` , `v2.7.2` , and `v2.7.3` tags also do not contain this change. The rename is, however, included in the Unreal Engine `5.7` , `5.6` , `5.5` , and `5.4` packages submitted to Epic's Fab Store, as those archives were built from their respective engine-specific branches, which already contained the change. The rename is now correctly applied to the `master` branch and the `v2.8.0` tag as part of the `v2.8.0` release. If you are using the source code directly from `master` , or from one of the mentioned `v2.7.x` tags obtained via Microsoft Azure DevOps Repositories (instead of the version distributed via Fab or other engine-specific branches), this introduces an ABI and API breaking change affecting both C++ and Blueprint code if your exsiting plugin version `v2.7.x` does not already include this rename.
- List existing SenseGlove components under `SenseGlove` `ClassGroup` inside Unreal's Blueprint Editor. This chagnes `ClassGroup` for `USGGrabComponent` , `USGTouchComponent` , `USGVirtualHandComponent` , and `USGWristTrackerComponent` .
- Force `USGVirtualHandComponent` and `USGWristTrackerComponent` to update their XR hand-tracking data when their handedness is updated.
- Bumped the SenseGlove Unreal Engine Marketplace Packager to `v0.6.2-b675bab` .

- Bumped the copyright years.

Removed

- Dropped support for Unreal Engine `5.4`, which was already deprecated via the `v2.7.x` release series.
- Dropped support for Epic Native/Cross Toolchains `v22` (previously used for building UE `5.3` and `5.4` Linux dependencies), as they were already deprecated via previous releases.
- Removed support for the deprecated `FXRMotionControllerData`. The plugin now exclusively uses `FXRHandTrackingState` (introduced in Unreal Engine 5.5+ and supported by The SenseGlove Unreal Engine Plugin since `v2.2.0` for OpenXR hand tracking. This affects only projects that directly consume `FXRMotionControllerData` from the SenseGlove plugin in their own custom hand-tracking or interaction systems. Please see the v2.7.x to v2.8.x migration guide for more details.
- Removed SenseGlove's `GetMotionControllerData()`; the alternative implementation to `IXTrackingSystem::GetMotionControllerData()`. You can now use SenseGlove's `GetHandTrackingState()` instead of Unreal's `IXTrackingSystem::GetHandTrackingState()` which guarantees the OpenXR hand-tracking data is coming from a SenseGlove device and also takes into account the SenseGlove's wrist-tracker offsets automatically.

Documentation

- Added a v2.7.x to v2.8.x migration guide for transitioning from `FXRMotionControllerData` to `FXRHandTrackingState` for projects that directly consume SenseGlove OpenXR data.
- Added documentation section Roll Your Own Hand Manipulation System.
- Added documentation section SGPawn Events.
- Added documentation section SGHandTrackerComponent.
- Added documentation section SGHapticsComponent.
- Updated Enabling XR_EXT_hand_tracking on VR Headsets documentation, replacing `FXRMotionControllerData` usage with `FXRHandTrackingState`.

- Updated Setup the Virtual Hand Meshes documentation, replacing `FXRMotionControllerData` usage with `FXRHandTrackingState`.
- Updated Plugin Configuration > Plugin Settings > Virtual Hand > Mesh documentation, replacing `FXRMotionControllerData` usage with `FXRHandTrackingState`.
- Updated Advanced Topics > OpenXR documentation section with more relevant and plenty of useful information reflecting the recent changes.
- Updated Advanced Topics > OpenXR > Consuming FXRHandTrackingState documentation section with information reflecting the recent changes.
- Updated Advanced Topics > OpenXR > Consuming FXRHandTrackingState > Blueprint documentation section with information reflecting the recent changes.
- Updated Advanced Topics > OpenXR > Consuming FXRHandTrackingState > C++ documentation section with information reflecting the recent changes.
- Removed the **Consuming FXRMotionControllerData** documentation section as it's no longer relevant.
- Removed the **Consuming FXRMotionControllerData > Blueprint** documentation section as it's no longer relevant.
- Removed the **Consuming FXRMotionControllerData > C++** documentation section as it's no longer relevant.
- Ensure deterministic dependency builds by pinning Rust toolchain.
- Changelog errata fixes.
- Minor changelog formatting fixes.
- Additional minor fixes and improvements that may not be listed here.

[2.7.3] - 2026-02-17

This patch release resolves documentation build pipeline failures by updating broken mdBook tooling.

Documentation

- Bumped mdBook to `v0.5.2`

- Bumped the Michael-F-Bryan/mdbook-epub crate to `21a1c8134134201a2d555313447c96e56e2a8996`, which addresses issue #133. This allowed upgrading mdBook from `0.4.49` to `v0.5.2` without breaking images inside the generated ePub version of the handbook.
- Bumped HollowMan6/mdbook-pdf to `v0.1.13`.
- Removed lambdalisue/rs-mdbook-alerts as it's incompatible with mdBook `v0.5.x`. The good news is that mdBook itself is now able to support this feature natively.

[2.7.2] - 2026-02-17

This patch release resolves build issues affecting Unreal Engine `5.7` on Linux `Arm64` architecture.

Fixed

- Fixed an issue in UE `5.7` projects where Linux `Arm64` builds were incorrectly linking against third-party libraries compiled with Epic Native/Cross Toolchain `v25` (shipped with UE `5.6`), instead of the required Toolchain `v26` binaries (shipped with UE `5.7`).

[2.7.1] - 2025-12-09

This patch release addresses a severe regression affecting UE `5.5` projects with specific settings.

Fixed

- Resolved a critical deadlock between the rendering and game threads in Unreal Engine `5.5` that when `IHeadMountedDisplay::GetHMDMonitorInfo()` is invoked from `FSGXRTracker::GetControllerTransform()` in certain conditions; specifically while SenseCom is running and gloves are present and connected. This issue is related to critical deadlock issue (UE-212224), occurring during PipelinedFrameState acquisition, which had already been addressed for the plugin versions running on UE `5.6+`. Because it only gets triggered under specific project settings, we were previously unaware that it also affected IHeadMountedDisplay::GetHMDMonitorInfo() in UE `5.5`.

Documentation

- Locked the mdbook-pdf crate to `v0.1.11` due to the following `v0.1.12` panick during PDF generation: called `Result::unwrap()` on an `Err` value: Unable to deserialize the `RenderContext`.

[2.7.0] - 2025-11-18

This minor release focuses on delivering performance improvements, made possible by major optimizations in the underlying proprietary SenseGlove libraries, while also introducing some breaking changes.

Added

- Added support for UE `5.7`.
- Added support for Android NDK `r27c`, which is the recommend Android SDK since UE `5.6.1` and the default for UE `5.7+` for Android Standalone builds.
- Added support for Epic Cross and Native Toolchains `v26`, which is shipped with the UE `5.7` and GNU/Linux support.
- Added prebuilt binaries for third-party library {fmt} Formatting Library on all supported platforms. This third-party library is a required dependency of SenseGlove libraries >= `v2.300.0`.
- Added prebuilt binaries for third-party library: Loguru Logging Library on all supported platforms. This third-party library is a required dependency of SenseGlove libraries >= `v2.300.0`.
- Added third-party module `SGCommonThirdPartyLibs`.

- Added third-party module `SGConnectShmThirdPartyLibs`.
- Added third-party module `SGCoreShmThirdPartyLibs`.
- Added third-party module `SGCoreThirdPartyHeaders`.
- Added third-party module `SGFmtThirdPartyLibs`.
- Added third-party module `SGLogThirdPartyLibs`.
- Added third-party module `SGLoguruThirdPartyLibs`.
- Added third-party module `SGWjwwoodSerialThirdPartyLibs` to replace `SGSerialThirdPartyLibs` while retaining `SGSerialThirdPartyLibs` for a different purpose. See the relevant comment in the Changed section below.
- Added UPROPERTY `USGTouchComponent::VibrotactileAmplitude`.
- Added UPROPERTY `USGTouchComponent::VibrotactileFrequency`.
- Added method `USGTouchComponent::GetVibrotactileAmplitude()`.
- Added method `USGTouchComponent::SetVibrotactileAmplitude()`.
- Added method `USGTouchComponent::GetVibrotactileFrequency()`.
- Added method `USGTouchComponent::SetVibrotactileFrequency()`.
- Added Blueprint-accessible method `USGTouchComponentKismetLibrary::GetVibrotactileAmplitude()`.
- Added Blueprint-accessible method `USGTouchComponentKismetLibrary::SetVibrotactileAmplitude()`.
- Added Blueprint-accessible method `USGTouchComponentKismetLibrary::GetVibrotactileFrequency()`.
- Added Blueprint-accessible method `USGTouchComponentKismetLibrary::SetVibrotactileFrequency()`.

Fixed

- Fix a typo in the function name `SGDeviceList::SenseCommRunning()`. This fix breaks ABI and API compatibility with previous versions of the plugin and affects both C++ and Blueprint code.

Changed

- `SGDeviceList::SenseCommRunning()` has been renamed to `SGDeviceList::SenseComRunning()` due to a typo. This change breaks ABI and

API compatibility with previous versions of the plugin and affects both C++ and Blueprint code.

- Bumped the SenseGlove libraries to `v2.305.3-17a820b6e`. This release of the SenseGlove libraries disables RTTI/Exceptions for the most parts and isolates it to a minor portion of the code base, which yields noticable performance gains. Futhermore, some optiomizations are done in the multi-threaded code, such as replacing Spinlocks with Adaptive Mutexes (a hybrid Mutex/Spinlock).
- As a result of SenseGlove libraries >= `v2.300.0` changing it's directory structure, the `ThirdParty` folder's directory structure has been revamped.
- Renamed third-party module `SGSerialThirdPartyLibs` to `SGWjwwoodSerialThirdPartyLibs` since SenseGlove libraries >= `v2.300.0` ships a new static library named `sgserial`. Thus, to avoid confusion and naming conflicts the third-party `serial` static library is now provided by the `SGWjwwoodSerialThirdPartyLibs` module and `sgserial` is provided by the `SGSerialThirdPartyLibs` module.
- `FSGGloveTrackingSettings::GloveConnectivityCheckInterval` settings have been renamed to `FSGGloveTrackingSettings::DataRetrievalRefreshRate` for adoption other than glove connectivity use cases.
- `USGVirtualHandComponent::GetMotionControllerData()` signature has changed.
- `USGVirtualHandComponent::GetHandTrackingState()` signature has changed.
- `USGWristTrackerComponent::GetMotionControllerData()` signature has changed.
- `USGWristTrackerComponent::GetHandTrackingState()` signature has changed.
- `USGVirtualHandComponentKismetLibrary::GetMotionControllerData()` signature has changed.
- `USGVirtualHandComponentKismetLibrary::GetHandTrackingState()` signature has changed.
- `SGPawn` and `SGTouchComponent` no longer use the legacy `QueueVibroLevels` method for applying vibrotactile feedback. Instead it's been replaced with `SendCustomWaveform`.
- Changed `USGTouchComponent::VibrotactileDuration` UPROPERTY's maximum value to `1.0f`. Previously it was uncapped, and now any value beyond `1.0f` seconds is clamped.

Removed

- Dropped support for Unreal Engine `5.3`, which was already deprecated in the `v2.6.x` release series.
- Dropped support for MSVC `v142` (Visual Studio 2019), since UE `5.3` was the last supported version relying on it.
- Cleaned up remnants of the long-removed Unreal Engine `5.2` from third-party module `*.Build.cs` files.
- Cleaned up remnants of the long-removed Unreal Engine `5.2` from `SenseGlove.Build.cs`, `SenseGloveKismet.Build.cs`, `SenseGloveTracking.Build.cs`, files.
- Cleaned up remnants of the long-removed Unreal Engine `5.2` from `SenseGloveTracking` module.
- `USGVirtualHandComponent::GetMotionControllerState()` has been removed.
- `USGWristTrackerComponent::GetMotionControllerState()` has been removed.
- `USGVirtualHandComponentKismetLibrary::GetMotionControllerState()` has been removed.
- `USGWristTrackerComponentKismetLibrary::GetMotionControllerState()` has been removed.
- `USGTouchComponent::VibrotactileLevel` UPROPERTY has been removed.
- `USGTouchComponent::GetVibrotactileLevel()` method has been removed.
- `USGTouchComponent::SetVibrotactileLevel()` method has been removed.
- `USGTouchComponentKismetLibrary::GetVibrotactileLevel()` method has been removed and is no longer available to Blueprint.
- `USGTouchComponentKismetLibrary::SetVibrotactileLevel()` method has been removed and is no longer available to Blueprint.

Deprecated

- **This is the last major/minor release to support Unreal Engine** `5.4`, and its support will be removed in future minor or major releases. This is **important to keep in mind if your target development and deployment platform is HTC VIVE in Standalone Mode**. Unfortunately, HTC has not released any updates to their HTC ViveOpenXR plugin since December 6, 2024. Their latest release [1] [2], ViveOpenXR Plugin `v2.5.0`, supports only Unreal Engine `5.3` and `5.4`. HTC VIVE PCVR Mode is unaffected and will remain fully functional because, on Microsoft Windows, it is supported via the OpenXRViveTracker Plugin, which is bundled with Unreal Engine and officially maintained by Epic Games. If you still intend to target HTC in Standalone Mode, you are welcome to continue using the latest SenseGlove Unreal Engine Plugin `v2.7.x`, which will retain HTC Standalone Mode support. However, please keep in mind that once newer versions of the SenseGlove Unreal Engine Plugin are released and UE `5.4` is no longer supported, the latest release of the plugin supporting UE `5.4` will not receive new features, hardware support, or bug fixes. If at any point in the future HTC releases a new version of their ViveOpenXR plugin that supports any Unreal Engine version we actively support, in accordance with our support policy and Platform Support Matrix, we will make every reasonable effort to reintroduce HTC Standalone Mode support.

Documentation

- Added {fmt} Formatting Library License section.
- Added Loguru Logging Library License section.
- Updated Setting up the Touch System section to reflect the recent touch system changes.
- Added a third-party tutorial to the Android Standalone Mode Deployment section for UE `5.7` which provides a one-click Android development and build environment setup, instructions on how to setup and use the new UE `5.7` project launcher, and how to enabled OpenXR hand-tracking support on Meta Quest and HTC VIVE devices.

[2.6.3] - 2025-06-27

This patch release contains no changes to the plugin code. It includes only documentation updates and improvements.

Documentation

- Added a new section to the handbook titled Third-Party OpenXR Integrations section.
- Additional minor fixes and improvements that may not be listed here.

[2.6.2] - 2025-06-10

This patch release contains no changes to the plugin code. It includes only documentation updates and improvements.

Documentation

- Added a third-party tutorial to the Optimizing Your Project for Higher FPS section.

[2.6.1] - 2025-06-05

This patch release addresses several critical build and linking issues.

Fixed

- Fix a linking issue on GNU/Linux with UE `5.6` where SenseGlove libraries were built against dynamic versions of `libc++` and `libc++abi` libraries rather than the static versions.
- Fix an issue where SenseGlove libraries for some targets were not actually built with `C++20` and still were built against `C++17`.
- Fix an issue where SenseGlove libraries for some targets were built or linked with incorrect settings.
- Additional minor fixes and improvements that may not be listed here.

Changed

- Bumped the SenseGlove libraries to `v2.204.0-3a37b1977`.

Removed

- SenseGlove plugin no longer ships Boost or wjwwood's Serial Communication Library header files as SenseGlove public headers shipped with `v2.204.0-3a37b1977` render them redundant. This significantly removes clutter, free up disk space, and speed up builds to some extent.

Documentation

- Fix some changelog typos.
- Additional minor fixes and improvements that may not be listed here.

[2.6.0] - 2025-06-04

This minor release delivers broad compatibility, stability, and maintainability enhancements, focusing on bringing full Unreal Engine `5.6` support, C++20 migration, and resolving various GNU/Linux build issues.

Added

- Added support for Epic Native Toolchain `v25`, which will be shipped with the upcoming UE `5.6`.

Fixed

- Resolved GNU/Linux build issues for Unreal Engine `5.5` and `5.6` caused by incorrect linkage to GNU/GCC's `libstdc++` instead of LLVM/Clang's `libc++`.

- Fix `SGLog` build issues on GNU/Linux with UE `5.6`.
- Fix type conversion safety and consistency issues across all `SGLog` formatters.
- Resolved a critical deadlock between the rendering and game threads in Unreal Engine `5.6` that occurrs when `IHeadMountedDisplay::GetHMDMonitorInfo()` is invoked from `FSGXRTracker::GetControllerTransform()`. This is similar to another critical deadlock issue (UE-212224), occurring during PipelinedFrameState acquisition, addressed in the `v2.5.0` release.
- Additional minor fixes and improvements that may not be listed here.

Changed

- Replaced Epic Native Toolchain `v24` support with Epic Native Toolchain `v25` due to the fact that now `v25` is the default Linux toolchain for UE `5.6`.
- Revamped `FSGHMDTracker` to resolve a critical deadlock between the rendering and game threads in Unreal Engine `5.6` that occurrs when `IHeadMountedDisplay::GetHMDMonitorInfo()` is invoked from `FSGXRTracker::GetControllerTransform()`. This is similar to another critical deadlock issue (UE-212224), occurring during PipelinedFrameState acquisition, addressed in the `v2.5.0` release.
- Revamped the UBT logic for importing third-party headers and libraries by introducing third-party dependency modules, `SGBleThirdPartyLibs`, `SGConnectThirdPartyHeaders`, `SGConnectThirdPartyLibs`, `SGCoreThirdPartyLibs`, and `SGSerialThirdPartyLibs`, which significantly reduces UBT boilerplate code and increases maintainability.
- Bumped the SenseGlove libraries to `v2.203.0-f3d3e676`.
- SenseGlove libraries have migrated to `C++20` from `C++17`.
- Revamped the `SGLog` logging utility class to use `TUniquePtr` instead of `std::unique_ptr`.
- `SGLog` now relies on `TAtomic<bool>` for thread-safe initialization.
- `SGBackend` now relies on `TAtomic<bool>` for thread-safe initialization.
- `USGBackend::IsBackendInitialized()` is no longer inlined and the initialization flag has been moved to the private implementation of `USGBackend`.
- Bumped the SenseGlove Unreal Engine Marketplace Packager to `v0.6.0-4108c6f`.

Removed

- Dropped support for Epic Native Toolchain `v24`, which was last shipped with the preview release of UE `5.6`, but has been removed from the `5.6` branch on GitHub.

Deprecated

- This is the last minor release to support Unreal Engine `5.3` and its support will be removed from the next minor or major releases.

Documentation

- Revised the outdated Plugin Directory Structure section to accurately reflect the latest changes to the `Source/ThirdParty` directory layout changes.
- Lock the mdbook crate version to `v0.49.0` to avoid layout issues introduced with `v0.50.0`.
- Additional minor fixes and improvements that may not be listed here.

[2.5.0] - 2025-05-09

This minor release primarily focuses on bringing Bluetooth Low Energy support to the SenseGlove Unreal Engine integration.

Added

- Added support for Epic Native Toolchain `v24`, which will be shipped with the upcoming UE `5.6`.

Fixed

- Backend initialization error handling on Android.
- Fix a critical issue introduced by UE `5.5` that also affects the upcoming UE `5.6`. This is known as issue `UE-212224`, which leads to a deadlock during `PipelinedFrameState` acquisition between the game and rendering threads.
- Additional minor fixes and improvements that may not be listed here.

Changed

- The error codes returned from `FSGConnectJNI::Initialize()` and `FSGCoreJNI::Initialize()` have been changed. This is a breaking change for any code that relies on handling the return codes from those functions.
- Bumped the SenseGlove libraries to the `v2.200.0-0cb715d0` release with BLE (Bluetooth Low Energy) support.

Removed

- Dropped support for Unreal Engine `5.2` and Epic Native Toolchain `v21` (previously used for building UE `5.2` Linux dependencies), as they were already deprecated in the `v2.4.x` release series.

Documentation

- Revamped SenseCom documentation in order to divide the SenseCom instructions section into Bluetooth Low Energy instructions for SenseCom and Bluetooth Serial instructions for SenseCom sections.
- Added Bluetooth Low Energy instructions for SenseCom.
- Added Bluetooth Serial instructions for Android.
- Added SGBLE and SGBLExx Rust Dependency Licenses.
- Bumped the mdbook-epub crate to `cac03b7f4b151f106f7f05b13da4c33fc098dd2c`.
- List the third-party tutorials inside the Extra Resources section in a categorized manner.

- Improved changelog formatting.
- Additional minor fixes and improvements that may not be listed here.

[2.4.2] - 2025-02-17

This is a patch release to address minor issues in the SenseGlove Unreal Engine Handbook, with no modifications to the plugin code.

Documentation

- List the third-party tutorials with a description in their corresponding parent sections.
- Additional minor fixes and improvements that may not be listed here.

[2.4.1] - 2025-02-14

This is a patch release to address minor issues in the SenseGlove Unreal Engine Handbook, with no modifications to the plugin code.

Documentation

- Fix a bug that breaks the custom CSS styles on the Handbook's release URLs (e.g. `https://unreal.docs.senseglove.com/2.4/`) by reverting an unintentional change from the `v2.4.0`.
- Applied a minor Handbook Makefile fix.
- Additional minor fixes and improvements that may not be listed here.

[2.4.0] - 2025-02-14

This minor release brings various improvements and, notably, the first version to add support for VIVE standalone mode with ViveOpenXR compatibility.

Added

- Added the ePub version of the SenseGlove Unreal Engine Handbook.
- Added `FSGConnectJNI::Initialize()` and `FSGCoreJNI::Initialize()` return codes to Android logs for detailed debugging purposes through `adb logcat`.
- Introduced compatibility with the `ViveOpenXR` plugin.
- Added the `FSGPluginUtils` utility class for other plugins or modules availability detection such as `Meta XR` and `ViveOpenXR`.
- Added various HTC HMDs auto-detection support on Android.
- Added support for HTC VIVE Focus Vision HMD auto-detection.
- Added support for HTC Vive Wrist Trackers support on Android using the official `ViveOpenXR` plugin's OpenXR positional tracking provider `OpenXRViveWristTracker`.
- Added enum `ESGOpenXRPositionalTrackingProvider`.
- Added the `OpenXRPositionalTrackingProvider` option to the plugin's wrist-tracker settings to either manually set the positional tracking provider or auto-detect it based on a combination of tracker hardware settings, targeted platform, available OpenXR plugins, or the auto-detected HMD hardware.

Fixed

- Fix a critical issue where `HandStates->GetTransform(KeyPoint)` was incorrectly resolving to `(&HandStates[0])->GetTransform(KeyPoint)`, causing both hands to use the left hand's wrist transform under specific conditions. This occurred when the `bFallbackToHandTrackingIfNoGloveDetected` option was enabled, two gloves were present, and no hardware wrist-tracking device was active, resulting in both hands overlapping at the same transform.
- `FSGArrayUtils` optimizations that affect the plugin performance as a whole.
- Additional minor fixes and improvements that may not be listed here.

Changed

- Now the motion sources for the wrist-tracking hardware or hand-tracking are queried and populated dynamically rather than relying on the hardcoded `EControllerHand` enum. This allows the SenseGlove Unreal Engine Plugin to integrate better into other plugins such as `ViveOpenXR`, which when enabled, provides many more options as the motion source for their various wrist-tracking hardware.
- `FSGWristTrackingSettings::LeftHandMotionSource` and `FSGWristTrackingSettings::RightHandMotionSource` types have changed from `EControllerHand` to `FName`.
- Bumped the SenseGlove libraries to `v2.105.3-97ea18cb`.
- Bumped the SenseGlove Unreal Engine Marketplace Packager to `v0.5.0-7df1183`.
- Bumped the copyright years.
- This is the last release to support Unreal Engine `5.2`. From `v2.5.x` onwards only UE `5.3` and newer will be supported.
- The `ESGViveHMDDetectionPriority` enum items have changed and are no longer backward-compatible.

Deprecated

- This is the last minor release to support Unreal Engine `5.2` and its support will be removed from the next minor or major releases.

Documentation

- Reintroduced the Handbook in ePub format.
- Revamped the Enabling XR_EXT_hand_tracking OpenXR Extension on VR Headsets and Deploying to Android (Standalone) documentation, and added the `ViveOpenXR`-compatibile instructions as well.
- Significantly improved the Setting Up the Wrist Tracking Hardware section by providing more detailed documentation and example configuration per HMD and wrist tracking hardware.

- Clarified how to set up the VIVE Wrist Trackers in various configurations.
- Added HTC VIVE specific optimization tips for running in standalone mode.
- Fixed a few broken URLs caused by bad markdown formatting.
- Applied various Handbook Makefile fixes.
- Clarified the HTC VIVE standalone support status in the platform support matrix.
- Bumped the mdbook-alerts crate to `v0.7.x`.
- Reintroduced mdbook-epub at `d1536bbbdc1ca00320522ad73a967e15057ef573` from the `master` branch as the blocking issues in #115 have been address in `1ca2a860f6ed405c00914a3aadd8057d5050b29b`.
- Added third-party tutorials to the following sections: Enabling XR_EXT_hand_tracking on VR Headsets, Deploying to Android (Standalone), and OpenXR.
- Additional minor fixes and improvements that may not be listed here.
- List the VRExpansionPlugin demo in the Extra Resources section.

[2.3.2] - 2025-01-28

This patch release addresses a critical issue backported from the upcoming `2.4.x` release to `2.3.x`.

Fixed

- Fix a critical issue backported from the upcoming `2.4.x` release where `HandStates->GetTransform(KeyPoint)` was incorrectly resolving to `(&HandStates[0])->GetTransform(KeyPoint)`, causing both hands to use the left hand's wrist transform under specific conditions. This occurred when the `bFallbackToHandTrackingIfNoGloveDetected` option was enabled, two gloves were present, and no hardware wrist-tracking device was active, resulting in both hands overlapping at the same transform.

[2.3.1] - 2024-11-27

This patch release addresses a few issues with SenseGlove Sockets Editor.

Fixed

- Additional minor fixes and improvements that may not be listed here.

Changed

- The SenseGlove Sockets Editor now calculates hand bone reference transforms using the current virtual hand mesh being edited, rather than the reference mesh, when adding SenseGlove sockets.

[2.3.0] - 2024-11-13

This minor release includes some improvements and adds official Unreal Engine `5.5` Fab support.

Added

- Added `USGAndroidPermissions` to the `SenseGloveAndroid` module, enhancing the plugin's permission request process on Android. Now, a pop-up prompts the user to grant permissions, preventing silent crashes when permissions haven't been granted beforehand.
- Added Unreal Engine `5.5` Fab support.

Fixed

- Fix UE `5.5` deprecation warnings inside `USGVirtualHandComponent`.
- Additional minor fixes and improvements that may not be listed here.

[2.2.2] - 2024-11-08

This patch release addresses a few issues with both glove and hand-tracking.

Fixed

- Fixed a chain of critical bugs that gets triggered due to `GloveConnectivityCheckInterval` getting passed as seconds to the engine rather than milliseconds. Thus, the default or any large value for `GloveConnectivityCheckInterval` causes noticeable long delays between glove-connectivity-check intervals and consequently renders the hand-tracking state invalid in certain situations when the `bFallbackToHandTrackingIfNoGloveDetected` option is false.

[2.2.1] - 2024-10-23

This patch release focuses exclusively on updates to the documentation.

Documentation

- Updated all URLs, screenshots, and tutorials to reflect the transition from the Unreal Engine Marketplace to Fab, Epic's new unified content marketplace.
- Revised documentation now points to the new home of the SenseGlove Unreal Engine Plugin on Fab, ensuring users have access to the latest resources and information.

[2.2.0] - 2024-10-22

This is a minor release with some breaking API and ABI changes, focusing mainly on migrating away from the deprecated `FXRMotionControllerData` in favor of `FXRMotionControllerState` and `FXRHandTrackingState` on Unreal Engine `5.5+`.

Added

- Completed support for the upcoming Unreal Engine `5.5` release.
- Added `USGVirtualHandComponent::GetMotionControllerState()` and the equivalent Blueprint function `UVirtualHandComponentKismetLibrary::GetMotionControllerState` on UE `5.5+`.
- Added `USGVirtualHandComponent::GetHandTrackingState()` and the equivalent Blueprint function `UVirtualHandComponentKismetLibrary::GetHandTrackingState` on UE `5.5+`.
- Added `USGWristTrackerComponent::GetMotionControllerState()` and the equivalent Blueprint function `UWristTrackerComponentKismetLibrary::GetMotionControllerState` on UE `5.5+`.
- Added `USGWristTrackerComponent::GetHandTrackingState()` and the equivalent Blueprint function `UWristTrackerComponentKismetLibrary::GetHandTrackingState` on UE `5.5+`.
- Added a variant of `FSGDebugVirtualHand::Draw()` and the equivalent Blueprint function `USGDebugVirtualHandKismetLibrary::Draw_FXRHandTrackingState()` which accept `FXRHandTrackingState` on UE `5.5+`.
- Added the new member `bTracked` to the `FSGXRHandState` struct.
- Added `FSGXRTracker::GetMotionControllerState()` and the equivalent Blueprint function `USGXRTrackerKismetLibrary::GetMotionControllerState()`.
- Added `FSGXRTracker::GetHandTrackingState()` and the equivalent Blueprint function `USGXRTrackerKismetLibrary::GetHandTrackingState()`.

Fixed

- Additional minor fixes and improvements that may not be listed here.

Changed

- Replaced all internal usages of the `FXRMotionControllerData` struct with either `FXRMotionControllerState` or `FXRHandTrackingState` on UE `5.5+`.
- Deprecated `USGVirtualHandComponent::GetMotionControllerData()` on UE `5.5+`.

- Deprecated `USGWristTrackerComponent::GetMotionControllerData()` on UE `5.5+`.
- Deprecated the variant of `FSGDebugVirtualHand::Draw()` which accepts `FXRMotionControllerData` as a parameter on UE `5.5+`.
- Renamed `USGDebugVirtualHandKismetLibrary::Draw` to `USGDebugVirtualHandKismetLibrary::Draw_FXRMotionControllerData` for more clarification.
- Renamed an `FSGXRHandState` member from `bReceivedJointPoses` to `bHasReceivedJointPoses`.
- Changed the `FSGXRTracker::GetAllKeypointStates()` signature on UE `5.5+` to match the `IHandTracker` interface API changes.
- The animation system on UE `5.5+` has been revamped to utilize `FXRHandTrackingState` instead of `FXRMotionControllerData`.
- The wrist tracking system on UE `5.5+` has been revamped to utilize `FXRHandTrackingState` instead of `FXRMotionControllerData`.
- The hand interaction manipulation on UE `5.5+` has been revamped to utilize `FXRHandTrackingState`.
- The virtual hand debugging system on UE `5.5+` has been revamped to utilize `FXRHandTrackingState`.

Documentation

- Added the documentation on consuming the `FXRHandTrackingState` struct in both Blueprint and C++.
- Updated the documentation on consuming the `FXRMotionControllerData` struct.
- Additional minor documentation fixes and improvements that may not be listed here.

[2.1.4] - 2024-10-22

This is a bugfix release that delivers some documentation fixes.

Documentation

- Updated the documentation on consuming the `FXRMotionControllerData` struct.
- Additional minor documentation fixes and improvements that may not be listed here.

[2.1.3] - 2024-10-11

This bugfix release centers on adding initial support for the upcoming Unreal Engine `5.5`.

Added

- Added initial support for the upcoming Unreal Engine `5.5` release. Please note that, while the plugin is functional, a few adjustments are still required to address deprecation warnings. Specifically, the `FXRMotionControllerData` struct needs to be replaced with the newly introduced `FXRMotionControllerState` and `FXRHandTrackingState` structs, along with adjustments to adhere to the new hand-tracking API changes.
- Added support for Epic Native Toolchain `v23`.

Fixed

- Fix a bug inside `USGVirtualHandComponent::PostEditChangeProperty()` where the get member name check happens against the wrong class and member names.
- Additional minor fixes and improvements that may not be listed here.

Changed

- The SenseGlove libraries have been updated to `v2.105.0-02a2e508`.

[2.1.2] - 2024-09-02

This is a bugfix release that addresses a few non-critical issues and documentation fixes.

Fixed

- Fix a bug where the hands are always visible even when `bVisibleWhenHandDataUnavailable` is disabled.
- Fix a bug where the `HandVisibilityChangedEvent` event is not triggered on the virtual hand component visibility changes.
- Fix the wrong script name for `USGHMDTrackerKismetLibrary`.
- Fix the wrong script name for `USGXRTrackerKismetLibrary` .
- Fix `LogPython: Warning: 'SGHMDTrackerKismetLibrary' and 'SGXRTrackerKismetLibrary' have the same name (SenseGloveHeadMountDisplayKismetLibrary) when exposed to Python. Rename one of them using 'ScriptName' meta-data` when packaging the game.
- Fix the non-existent default hand-mesh warnings polluting the logs when packaging the game.
- Expanded the clickable area on the handbook index page revision buttons.
- Minor documentation fixes.

[2.1.1] - 2024-08-18

This is a bugfix release with no actual plugin code changes, mostly addressing issues in the documentation and third-party dependencies caused by source control merge conflicts.

Fixed

- Fix the messed up changelog file caused by cherry-picking merge conflicts between the dev branch and the master branch.

- Fix a bug that causes a handbook revision mismatch when deploying the handbook from the dev branch.
- Fix a bug where `SG_GIT_IS_SHALLOW_CLONE` while building the handbook is always set to `yes` even if it's not a shadow clone because `SG_DOT_GIT_SHALLOW_FILE` evaluates to an empty string when the `.git/shallow` file does not exist.
- Fix some documentation typos.

Removed

- Removed Android NDK `r25` `armv7` and `x86` dependencies brought back by mistake while merging `v2.1.0` from the dev branch to the master branch.

[2.1.0] - 2024-08-16

This is a minor release focusing mainly on bringing OpenXR-compatible hand tracking support (`XR_EXT_hand_tracking`) and Head-mounted Display automatic detection for adjusting wrist tracker offsets automatically at runtime.

Added

- Added SenseGloveTracking and module which provides OpenXR-compatible hand tracking by implementing `XR_EXT_hand_tracking` support, HMD auto-detection, and SenseGlove device tracking.
- Added `USenseGloveTrackingKismet` module in order to expose part of the SenseGloveTracking functionality to Blueprint.
- Added `FSGXRTracker` , the underlying main class that implements the OpenXR compatibility.
- Added `USGXRTrackerKismetLibrary` in order to allow Blueprint to retrieve the `FXRMotionControllerData` directly from our tracking module.
- Added the `SGTrackingTypes` header to the `SenseGloveTypes` module in order to define and share `SenseGloveTracking` module types through this header across the plugin modules.

- A fallback to HMD and wrist tracker hardware auto-detection mechanism has been added to be triggered in situations when automatic detection of the wrist tracker hardware is desired, e.g., either by not setting it explicitly, or setting it to the default None value. Please note that this is still highly experimental and HTC VIVE Focus 3 and HTC XR Elite cannot be distinguished in the current iteration. Though, since the tracker devices and offsets for both headsets are the same in the end it does not make a difference if both headsets are detected as each other.
- Added `ESGHeadMountedDisplayDevice` enum with supported HMDs list.
- Added `ESGViveHMDDetectionPriority` enum in order to choose which headset we attempt to detect between VIVE Focus 3 and VIVE XR Elite as we cannot distinguish them, yet.
- Added the `FSGHMDTracker` utility class, in order to easily gather information about the HMD device at runtime.
- Added `USGHMDTrackerKismetLibrary` which exposes the equivalent C++ HMD auto-detection functionality to Blueprint.
- Added `FSGHMDTrackingSettings` config struct.
- Added the `FSGGloveTracer` utility class, in order to easily check the left or right glove connectivity or retrieve the connected glove instances.
- Added `USGGloveTrackerKismetLibrary` which exposes the equivalent C++ functionality to Blueprint.
- Added `FSGGloveTrackingSettings` config struct.
- Added `FSGTrackingSettings` config struct.
- Added `FSGHandTrackingSettings` config struct.
- Added `FSGWristTrackingDebuggingSettings` config struct.
- Added `FSGVirtualHandSettings` config struct.
- Added `FSGVirtualHandAnimationSettings` config struct.
- Added `FSGVirtualHandDebuggingSettings` config struct.
- Added `FSGVirtualHandGrabSettings` config struct.
- Added `FSGVirtualHandHapticsSettings` config struct.
- Added `FSGVirtualHandMeshSettings` config struct.
- Added `FSGVirtualHandPhalangesLengthSettings` config struct.
- Added `FSGVirtualHandTouchSettings` config struct.
- Added `USGVirtualHandComponent::OnHandVisibilityChanged()` event in order to notify other components/actors whenever the virtual hand mesh appears or

disappears (for example, this could happen when a glove is connected/disconnected).

- `GetMotionControllerData()` has been introduced to the `USGVitualHandComponent` in order to retrieve the OpenXR-compatible glove data in Unreal's `FXRMotionControllerData` format.
- Added `FSGVirtualHandAnimInstanceProxy::GetMotionControllerData()` and many more accessor methods usable only by child classes to allow consumption of the data required for manipulating the virtual hand mesh animations.
- `GetMotionControllerData()` has been introduced to the `USGWristTrackerComponent` in order to retrieve the OpenXR-compatible glove data in Unreal's `FXRMotionControllerData` format.
- Added `USGGrabComponent::SimulatePhysics()` method.
- Added `FSGDebugCube`.
- Added `FSGDebugCubeSettings`.
- Added the `SenseGloveDebugKismet` module in order to allow drawing of debugging, cubes, gizmos, and virtual hands from Blueprint.
- Added `USGDebugCubeKismetLibrary` in order to expose the `FSGDebugCube` functionalities to Blueprint.
- Added `USGDebugGizmoKismetLibrary` in order to expose the `FSGDebugGizmo` functionalities to Blueprint.
- Added `USGDebugVirtualHandKismetLibrary` in order to expose the `FSGDebugVirtualHand` functionalities to Blueprint.
- Added a new static `Draw()` method overload to `DebugGizmo` which allows passing an `FQuat` instead of a `FRotator`.
- Introduced a new `FXRMotionControllerData` compatible hand animation system with the ability to take the mesh bone's transforms into account for a more reliable hand animation.
- Introduced a new `FXRMotionControllerData` compatible wrist tracking system.
- Introduced a new `FXRMotionControllerData` compatible hand interaction manipulation system.
- Added the ability to fallback to hand tracking when a glove is not present and use the bare hands for interactions, or a combination of glove and hand tracking if no motion controller input is detected.
- Added the SenseGlove grab/touch sockets one-click-setup ability on any Epic-compliant virtual hand mesh from within the Unreal Editor's Content Browser,

Skeleton Editor, or Skeletal Mesh Editor by extending the Unreal Editor.

- Added `FSGAssetUtils` editor-only class.
- Added `FSGContentBrowserExtension` editor-only class.
- Added `FSGPluginStyle` editor-only class.
- Added `FSGSocketsEdito` r editor-only class.
- Added `FSGSocketsEditorCommands` editor-only class.
- Added the `FSGInitializationSettings` config struct in order to control how the plugin is initialized.
- Introduced the `FSGGameUserSettings` for managing the Engine Scalability Settings through the SenseGlove plugin in order to change the graphics settings on the fly.
- Added `USGGameUserSettingsKismetLibrary` in order to allow all the Engine Scalability Settings to be managed from the Blueprint side.
- Added `FSGGameUserSettingsSettings` config struct.
- Added the SenseGlove console commands: `SG_GetEngineScalabilitySettings()` and `SG_SetEngineScalabilitySettings(Scalability)`.
- Added `SGHardwareBenchmarkingSettings` config struct.
- Introduced `ESGEngineScalabilitySettings` enum.
- Added `FSGVirtualHandSettingsOverrides` config struct used by the new settings override system.
- Added `SGWristTrackingSettingsOverrides` config structured by the new settings override system.
- Added support for Android API level `32` in addition to the API level `29`.
- Introduced the SenseGlove Unreal Engine Handbook as an attempt at documenting the SenseGlove Unreal Engine Plugin.
- Merged the `pack` utility branch to the plugin's source code at `/Packager` which adds the SenseGlove Unreal Engine Marketplace Packager `v0.4.0-a65bb20` binaries and configurations.

Fixed

- Fixed a bug when the virtual hand inside the game is not visible but still collides with other objects inside the scene, mistakenly triggering events like `OnGrabStateUpdated` and `OnTouchStateUpdated`.

- Fixed a bug where `USGGrabComponent` 's `bAffectPhysicsState` does not enables physics on its owning actor at `BeginPlay()` .
- Fixed various wrong Kismet script names and their class exports.
- Fixed the display name for various overloads of the Blueprint-exposed function `Queue Command Vibro Level` to expose sensible display names.
- Some Android UPL tweaks, permission, and build fixes.
- Many other large and small fixes and improvements that might not be listed here.
- A few small bugfixes that have already been backported to the `v2.0.x` series.

Changed

- Now, if `bValidateIfDefaultClassesAreSGCompliant` option from `FSGInitializationSettings` is enabled (default) the SenseGlove plugin checks for default SenseGlove-compliant `GameMode` , `GameInstance` , etc, at module initialization and tries to set to default, native SenseGlove classes, if any of those default classes are not a SenseGlove or a SenseGlove-derived class.
- The `USGSettings` has been fully revamped with more customizations added and categorized in a different manner adding many new structs and removing some, in order to have fine-grained control over the various aspects and functionality of the plugin components.
- The `USGSettings` constructor visibility has been changed from public to private.
- The Settings override system has been overhauled as well affecting how we override settings from the `USGVirtualHandComponent` and `USGWristTrackerComponent` .
- The SenseGlove libraries have been updated to `v2.104.1-55fddbd2` .
- `GetHandPose()` has been replaced by `GetMotionControllerData()` inside `USGVirtualHandComponent` (see the relevant entry in the Added and Removed sections).
- Many functions inside `USGVirtualHandComponent` for retrieving bone names or reference transforms has been renamed to return different data types; e.g. `GetLeftHandFingerBoneNames()` , `GetRightHandFingerBoneNames()` , `GetLeftHandFingerBoneName()` , and `GetRightHandFingerBoneName()` renamed to `GetLeftHandBoneNames()` , `GetRightHandBoneNames()` , `GetLeftHandBoneName()` , and `GetRightHandBoneName()` respectively.

- `bHiddenInGameIfNoGloveDetected` uproperty from `USGVirtualHandComponent` has been renamed to `bVisibleWhenHandDataUnavailable` and accordingly all of its getters and setters; `bVisibleWhenHandDataUnavailable = false` now acts as `bHiddenInGameIfNoGloveDetected = true`, and vice-versa.
- `USGWristTrackerComponent` now uses `FXRMotionControllerData` for wrist tracking instead of calculating the wrist location by calling the SenseGlove API.
- `FSGVirtualHandAnimInstanceProxy` now relies on `FXRMotionControllerData` to animate the hands instead of a `TMap` of bone names and rotations which allows it to also apply the bone locations.
- The new OpenXR animation system now takes into account the mesh bone's transforms for a more reliable hand animation.
- `FSGDebugVirtualHand::Draw()` now accepts a `FXRMotionControllerData` parameter instead of all `WristLocation`, `WristRotation`, `JointPositions`, and `JointRotations` parameters.
- `FSGDebugVirtualHandSettings` has been renamed to `FSGVirtualHandDebuggingSettings`.
- The value for `USGGrabComponent`'s `AttachmentSocketName` uproperty now defaults to the value of the plugin's `GrabAttachPointSocketName` instead of `Name_NONE`.
- The `SGGrabComponent` now enables `bGravityEnabled`, `bSimulatePhysics`, and calls `WakeRigidBody` on its owning actor at `BeginPlay()` if `bAffectPhysicsState` is enabled.
- Updated the Directory Structure section of the main README file to reflect the latest toolchain support status.
- The `/CHANGELOG.md` file has been migrated to `/Handbook/src/overview/changelog.md`
- The `/LICENSE.md` file has been migrated to `/Handbook/src/license/senseglove-unreal-engine-plugin.md`
- The `/LICENSE-THIRD-PARTY.md` file has been migrated to `/Handbook/src/license/third-party.md` and every third-party component's license has been split; adding `/Handbook/src/license/senseglove-sdk.md` for the SenseGlove SDK, `/Handbook/src/license/boost-cpp-libraries.md` for the Boost C++ Libraries, and `/Handbook/src/license/serial-communication-library.md` for the Serial Communication Library.
- The Platform Support Matrix section of the main README file has been migrated to `/Handbook/src/overview/platform-support-matrix.md`.

- The Planned Features Completion Status section of the main README file has been migrated to `/Handbook/src/overview/planned-features-completion-status.md`.
- The Directory Structure section of the main README file has been migrated to `/Handbook/src/overview/directory-structure.md`.
- The SenseGlove settings' main config struct is now marked as DefaultConfig which means it does not require to be saved when settings are changed and they take effect immediately as the user updates them.
- Replaced all bitfield uproperties with booleans.
- Changed the DocsURL from the old Blueprint docs website to the new SenseGlove Unreal Engine Handbook website.
- The Blueprint signature for various overloads of the Blueprint-exposed function `Queue Command Vibro Level` has been changed to expose sensible display names.

Removed

- Dropped support for Unreal Engine `5.1` and Epic Native Toolchain `v20` (used to build UE `5.0` and `5.1` Linux dependencies).
- Removed the Allbreaker virtual hand model as it's no longer compatible with the SenseGlove plugin.
- Removed `ASGVirtualHandActor` as it was experimental and we no longer maintain it and haven't been doing so for a long time.
- Removed `FSGVirtualHandAnimInstanceProxy::GetBonesRotations()`.
- Removed `USGVirtualHandComponent::GetHandPose()` and it's no longer possible to get the hand pose data from `USGVirtualHandComponent` as `GetHandPose()` has been removed. If you need it, you could always use the SenseGlove low-level API to retrieve it from the glove.
- Removed also `GetFingerBoneName()`, `GetFingerBoneRefTransform()`, `GetFingerBoneRefRotation()` and `GetFingerBoneRefRotation()` from `USGVirtualHandComponent`.
- Removed some remnants of UE `5.1` and older releases from the C++ code.
- Removed the `pack` utility branch and merge it to the plugin's source code at `/Packager`.

Known Issues

- With the new OpenXR release, the separation of the real and virtual hand rendering is broken. The reason is the animation system now uses the OpenXR data in the world transforms which yields better animations, but comes at the cost of overriding the the hand position set by the wrist tracker component's position and rotation. If `FXRMotionControllerData` is invalid and `bVisibleWhenHandDataUnavailable` is enabled for example, the system works as expected, since the animation system won't proceed to animate the hand meshes without valid `FXRMotionControllerData`. Since the animation system is only aware of the hand mesh it's animating versus the real hand and virtual hand meshes it means either it should become aware of the physics events like begin and end overlap events and also the real vs virtual hands, or it should resort back to animating the virtual hand meshes in local or component space. This release marks this feature as broken for now until we come up with a reasonable solution in the future.
- The `UXRDeviceVisualizationComponent` provided by Unreal Engine is used in the `SGPawn` class as `ControllerVisualizerLeft` and `ControllerVisualizerRight` for implementing the wrist tracking hardware visualization feature. However, it is not compatible with the new OpenXR system in certain scenarios. For instance, when the motion controllers serve as wrist tracking hardware since the SenseGlove plugin is now introduced to the engine as an `OpenXRHandTracking` system, it causes the `UXRDeviceVisualizationComponent` to visualize the wrist tracking hardware at coordinates (`0.0f`, `0.0f`, `0.0f`) instead of their actual location and rotation in the world. This happens because the component incorrectly registers them as inactive, possibly because it's assumed hand tracking and motion controllers cannot be in use at the same time. Currently, we use this feature solely for debugging, and we have an alternative in the form of wrist-tracking debug gizmos, which can be toggled on or off via the settings system. In future releases, we might remove this feature due to its incompatibility, unless we find a solution to make the `UXRDeviceVisualizationComponent` work with the new system. Alternatively, we may develop our own version of the `UXRDeviceVisualizationComponent`.
- Although the SenseGlove OpenXR implementation is fully compatible with the `IOpenXRHMD` interface and the `FOpenXRHMD` `XRTrackingSystem`, it is not compatible with the `FOculusXRHMD` backend provided by the Meta XR plugin. The

same issue likely applies to the VIVE OpenXR plugin. So, if these plugins are enabled in your project, the SenseGlove OpenXR will not function as intended, effectively breaking the plugin's functionality. It seems these plugins are necessary in order to make the fallback to the hand-tracking feature work on Android. While we may add support and compatibility with Meta XR and VIVE OpenXR plugins in the future, for the time being, if your project requires these plugins, we advise continuing with the `v2.0.x` release of the SenseGlove Unreal Engine plugin until this issue is addressed.

[2.0.8] - 2024-07-15

This is a bugfix release that contains a somewhat important bugfix backported from the next release of the plugin as documented below.

Fixed

- Fix a bug where the `SGPawn` right-hand grab colliders' default size is mistakenly set to the default value for the left-hand grab colliders at CDO initialization time.

[2.0.7] - 2024-05-29

This is a bugfix release with no actual plugin code changes, only fixing issues with binary assets incompatible with UE versions earlier than `5.4`.

Fixed

- Make the Allbreaker assets compatible with UE5.1+ again as the `v2.0.5` update breaks compatibility with UE versions earlier than `5.4`, thus leaving the engine unable to load those assets.

[2.0.6] - 2024-05-29

This is a bugfix release with no actual plugin code changes, only removing development/test assets from UE `5.3` that were never meant to be shipped.

Removed

- Removed the dev/test virtual hand models that leaked into the `5.3` branch.

Fixed

[2.0.5] - 2024-05-22

This is a bugfix release with no actual plugin code changes, only focusing on fixing the Allbreaker virtual hand model issues.

Fixed

- Fix the wrong palm bone names on the Allbreaker virtual hand models.

[2.0.4] - 2024-05-17

This is a bugfix release with no actual plugin's code change.

Fixed

- Fix our in-house Unreal Engine Marketplace submission tool's configurations where the Content folder (containing the Allbreaker hand model) is mistakenly

ignored during the submission. This release reintroduces the Virtual Hand Model and its material missing from the previous release.
- Fix the `SenseGlove.uproject`'s wrong versioning submitted to the Unreal Engine Marketplace.

[2.0.3] - 2024-05-15

This is a bugfix release addressing mostly RunUAT build issues on Unreal Engine `5.4`.

Fixed

- Fix UE `5.4` RunUAT build issue: "Asking CppCompileEnvironment for a single Architecture, but it has multiple Architectures (arm64, x64)", affecting `SenseGloveConnectImpl` and `SenseGloveCoreImpl` modues.
- Improved target platform detection when building `SenseGloveConnectImpl` and `SenseGloveCoreImpl` modules and also distinguishing the `x64` builds from `arm64` on Microsoft Windows.
- Fix other UE `5.4` RunUAT build issues, mostly caused by missing headers.

Removed

- Removed support for Android `armeabi-v7a` and `x86` architectures as they are no longer supported by the supported engine versions.

[2.0.2] - 2024-04-25

This is a patch release with no code changes.

Added

- Introduce official Unreal Engine `5.4` support to the Unreal Engine Marketplace.

Changed

- Updated the Platform Support Matrix with the latest changes. This is the last release to support Unreal Engine `5.1` as we no longer are able to push updates for this release to the Unreal Engine Marketplace. The `v2.0.1` release for Unreal Engine `5.1` can be obtained from the Unreal Engine Marketplace, and `v2.0.2` through our Microsoft Azure DevOps repositories. Please note that there are no actual code changes between these two releases and in terms of functionality they are almost identical.

[2.0.1] - 2024-04-15

This is a bugfix release.

Fixed

- Fix a bug inside both `SGVirtualHandComponent` and `SGWristTrackerComponent` where the connected glove's `UObject` instance gets destroyed and re-instantiated every frame. With this fix now the glove instance will be created or destroyed only when a glove connects to or disconnects from the system.
- Update the outdated Platform Support Matrix and its remarks section to reflect the latest status information.
- Fix the wrong header file description sections for the header files inside `SenseGloveKismet/Public/SGKismet/`.

Changed

- SenseGlove libraries have been updated to `v2.102.0-35d4de3f`.

- Together, SenseGlove libraries `v2.102.0-35d4de3f` and SenseCom `v1.6.1` remove the need to call ResetCalibration every time and are able to store and load calibration profiles from disk.
- SesenGloveBackend module is no longer calling `FSGHandLayer::ResetCalibration()` on every backend initialization.

[2.0.0] - 2024-03-22

This is the second major release of the SenseGlove Unreal Engine Plugin adding support for Nova 2 with enormous breaking changes to the current C++ and Blueprint APIs.

Added

- Added support for the SenseGlove Nova 2 devices.
- Added support for Quest 3 controllers.
- Various classes have been added to the API in order to implement the new functionalities and features from the latest upstream SenseGlove libraries.
- Added initial support for the upcoming Unreal Engine `5.4` release.
- Added a pair of default production-ready virtual hand meshes for the left and right hands, courtesy of Allbreaker LLC Columbia. For usage and redistribution, please consult the LICENSE-THIRD-PARTY.md file.

Fixed

- A few critical bug fixes that have already been backported to the `v1.x.x` series through `v1.9.3` to `v1.9.8` releases.
- Revamped the way we do `FVector <-> SGVect3D`, `FQuat <-> SGQuat`, and SenseGlove <-> Unreal Engine angles conversions in order to properly translate between the SenseGlove and Unreal Engine coordinate systems.
- Allow the C++ compiler the opportunity to perform RVO/NRVO if applicable.
- Fix the modules' order inside the `.uplugin` file.

- Fix a build issue inside `FSGArrayUtils::FromStdVector()` introduced by newer MVSC updates due to stricter implicit `uint64` to `int32` conversions.
- Fix a build issues inside `FSGArrayUtils` when performing non-Unity builds due to the missing `<string>` header.
- Fix other build issues in `USGDevice`, `USGNovaGloveSensorData`, `FSGDeviceImpl`, and `FSGSenseGloveVarsImpl` when performing non-Unity builds due to the missing relevant headers.
- Fix changelog formatting.
- Some other improverment and fixes.

Changed

- SenseGlove libraries have been updated to `v2.101.12-62b1be11`.
- The SenseGlove Unreal Engine Plugin now declares the OpenXR plugin as a dependency, so that the OpenXR plugin will be enabled automatically as soon as the SenseGlove Unreal Engine Plugin gets enabled.
- Various classes and parts of the API have been changed in order to reflect and adhere to upstream SenseGlove libraries.
- Reverse the Platform Support Matrix order from newer Unreal Engine versions to the older ones.
- Clarify the engine support policy in the main readme file by adding the corresponding references from the Epic Marketplace Guidelines and a URL to their guidelines page.
- The `SGTouchComponent` uproperties BuzzDuration and BuzzLevel now utilize different different names in order to correspond to the underlying API changes. They have been renamed to `VibrotactileDuration` and `VibrotactileLevel`.
- The `SGTouchComponent` uproperties ForceFeedbackLevel and BuzzLevel (now `VibrotactileLevel`) parameters type have changed from `int32` to `float` with the value range varying between `0.0f` to `1.0f` instead of `1` to `100` in order to correspond to the underlying API changes.
- The `SGVirtualHandComponent` now assumes the default grab point's name as `GenericGrabPoint` instead of `GrabPoint` as default if not specified in the Unreal Blueprint Editor.
- The `SGPawn` on UE `5.2+` now utilizes `UXRDeviceVisualizationComponent` in order to properly display the controller meshes shipped with Unreal Engine's OpenXR

plugin, or a user-provided mesh. On UE `5.1` this could still be set on the `WristTrackerLeft` and `WristTrackerRight` components. Please note that despite the fact that on UE `5.2+` it's still possible to utilize the `WristTrackerLeft` and `WristTrackerRight` for setting the controller meshes, this has been deprecated in UE `5.2+` and is no longer supported.

Removed

- Various classes and parts of the API have been removed in order to reflect and adhere to upstream SenseGlove libraries.
- Removed the redundant `SGIC_int32_Ref` interop type.

[1.9.8] - 2024-03-12

This is a bugfix release that contains bugfixes backported from the next major release of the plugin as documented below.

Fixed

- Fix a bug where the right-hand mesh is always hidden inside the game no matter whether the right glove is connected or not.
- Fix a crash inside the `USGHandPose::FromHandAngles()` method.
- Some performance optimizations by utilizing `MoveTemp` in return statements.
- Some improvements applied to the source code.
- Some other minor fixes.

Changed

- The BonesRotations TMap is no longer a public field of `FSGVirtualHandAnimInstanceProxy` and instead could be retrieved by calling the `GetBonesRotations()` method.

[1.9.7] - 2024-02-18

This is a bugfix release that contains bugfixes backported from the next major release of the plugin as documented below.

Fixed

- Fix various bugs inside the `SGPlayerController` which occur when the thumb and pinky fingers are simultaneously touching different `SGTouchComponents`, or only one of them is in touch with such a component. In this case pinky's buzz and force-feedback levels are determined from the `SGTouchComponent` that is in collision with the thumb instead of the one that is touched by the pinky. Or, the pinky could ignore the buzz and force-feedback level if the thumb is not in collision with an `SGTouchComponent`. Or, the pinky could have reacted with a buzz or force feedback while only the thumb is in contact with an `SGTouchComponent`.
- Fix the `BuzzDuration` uproperty range in order not to get clamped at `100.0f` and also use `float` values for `ClampMin` and `UIMin` specifiers instead of integer values.

[1.9.6] - 2024-02-14

This is a bugfix release.

Fixed

- Fix a few critical bugs inside the `NovaGlove` class where the higher levels of the API including constructors, Parse, and `NewNovaGlove` methods mistakenly instantiate a `SenseGloveImpl` class instead of a `NovaGloveImpl` class.

[1.9.5] - 2024-02-09

This is a bugfix release.

Fixed

- Fix a wrong type-casting inside `SGDeviceModel::ParseFirmware()` where `OutMainVersion` and `OutSubVersion` arguments are getting passed to the lower levels of the API. This could potentially result in a segfault at the FFI boundary between lower and higher levels of the API.

[1.9.4] - 2024-02-08

This is a bugfix release addressing mostly Blueprint API issues with ABI breaking changes inside the Blueprint layer, backported from the next major release of the plugin as documented below.

Fixed

- Fix the Blueprint Parse function signature for the `NovaGloveInfoKismetLibrary` where the `OutGloveInfo` passed by the caller was never actually assigned as it was not getting passed by reference.
- Changelog formatting.

[1.9.3] - 2024-02-03

This is a hotfix release addressing a few critical issues that might result in crashes or malfunctions for users of the low-level SenseGlove API, backported from the next major release of the plugin as documented below.

Fixed

- Fix a potential memory corruption inside one of the `SGBasicHandModel` constructors where the StartPositions parameter gets passed as the StartRotations parameter to lower levels of the API.
- Fix a potential memory corruption inside one of the `SGSenseGloveInfo` constructors where the StartPositions parameter gets passed as the Functions parameter to lower levels of the API.
- Fix a potential memory corruption where inside the `SGHapticGloveCalibrationSequence::GetCurrentInstruction()` method, the return statement of the function is getting assigned to the const parameter `NextStepKey`, thus the return statement of the function will always be empty as well.
- Fix a potential memory corruption where inside one of the overloads of the `SGSenseGloveImpl::GetGlovePose()` method, the out parameter of the method is getting passed as the SensorData parameter to the lower levels of the API.
- Fix multiple Equals methods for a few classes such as `SGInterpolationSet`, `SGNovaGloveHandProfile`, `SGNovaGloveInfo`, `SGSenseGloveHandProfile`, `SenseGloveInfo`, `SenseGlovePose`, where the Equal method compares the current instance against itself instead of the other instance passed to as the parameter to the method.
- Removed a redundant code statement inside the `SGNovaGloveImpl::GetSubFirmwareVersion()` method.
- Some minor const correctness fixes.
- Some other minor code fixes and improvements.
- Fix the wrong version numbers inside the paltform support matrix and the main `.uplugin` file.
- Minor changelog fixes.
- Bumped the copyright years.

[1.9.2] - 2023-11-03

Added

- Added a list of planned features and their completion status to the main README file.

Fixed

- A bug where the released actor is going to be `NULL` whenever the `OnActorReleased` event fires.

[1.9.1] - 2023-10-11

Fixed

- Add the missing Unreal Engine C++ header to files that rely on the `ENGINE_*_VERSION` macros in order to fix the Epic Store build failures on UE `5.3`.

[1.9.0] - 2023-10-10

Changed

- The `BlueprintImplementableEvent` ufunction specifier for the `OnGrabStateUpdated`, `OnTouchStateUpdated`, `OnActorGrabbed`, `OnActorReleased`, `OnActorBeginTouch`, and `OnActorEndTouch` events have been changed to BlueprintNativeEvent in order to allow them to be implemented from the child

C++ classes as well. This won't break any existing Blueprint code that relies on the previous BlueprintImplementableEvent signature.

Fixed

- Add a missing release note entry for the `v1.8.0` release to the changelog file.

[1.8.0] - 2023-10-10

Added

- Introduced new `SGPawn` events: `OnActorGrabbed`, `OnActorReleased`, `OnActorBeginTouch`, and `OnActorEndTouch`.
- Exposed `OnGrabStateUpdated`, `OnTouchStateUpdated`, `OnActorGrabbed`, `OnActorReleased`, `OnActorBeginTouch`, and `OnActorEndTouch` events to Blueprint as `BlueprintImplementableEvent`.

Fixed

- Fix a bug where the `OnTouchStateUpdated` event is mistakenly triggered instead of the `OnGrabStateUpdated` when the right thumb fingertip grab collider overlaps with a grabbable actor.
- Fix the `DECLARE_EVENT` macro signature for `OnGrabStateUpdated` and `OnTouchStateUpdated` events.

[1.7.0] - 2023-09-14

Added

- Introduce `SGGameInstance`, a customized SenseGlove game instance for future use.
- Added the new `SenseGloveBackend` and `SenseGloveBackendKismet` modules.
- Added `SG_CPP20` C++ macro for C++20 detection, which is now default from UE `5.3` onwards.
- Added `SG_CAPTURE_THIS` C++ macro as a workaround for `error C4855: implicit capture of 'this' via '[=]' is deprecated in /std:c++20` in order to build the same lambda captures without extra `#ifdef`s on all supported engine versions.

Changed

- SenseGlove libraries have been updated to `v2.12.0-19c9854`.
- `SGCoreImpl`/`SGPlatform` has been moved to `SGBuildHacks`/`SGPlatform`.

Fixed

- Proper initialization of the SenseGlove backend in order to fix a bug in certain situations where `SGConnect::Init()` gets called every frame.
- Some other minor fixes and improvements.

[1.6.1] - 2023-08-14

Fixed

- Fix Unreal Engine `5.0` build issues.

- Minor documentation fixes.

[1.6.0] - 2023-08-14

Added

- Added support for the upcoming Unreal Engine `5.3`.
- Now, the hand's velocity is applied to grabbed actors after being released from the hand.
- Introduce the real hands to the `SenseGlove` module (`SGPawn`) API.
- Added separation of the virtual and real hand rendering.

Fixed

- Fix the wrong default debug virtual hand gizmo colors when initialized using the default constructor.
- Some minor performance fixes and improvements.

Changed

- SenseGlove libraries have been updated to `v2.11.0-b775a05`.

[1.5.3] - 2023-07-19

This is a hotfix release mostly addressing Android Bluetooth performance issues.

Fixed

- Minor changelog fixes.

Changed

- SenseGlove libraries have been updated to `v2.10.1-3b0e7c9`.

[1.5.2] - 2023-07-19

This is a hotfix release mostly addressing Android-related issues.

Fixed

- Fix a build issue with Android shipping builds due to `sgconnect.jar` not getting copied automatically in the AFSProject which is compiled for shipping builds when `AndroidFileServer (AFS)` is enabled.
- Minor changelog fixes and some source code formatting fixes.

[1.5.1] - 2023-07-13

This is a hotfix release addressing a few critical issues introduced by the recent changes.

Fixed

- Fix a wrist tracker bug where left and right hands' wrist trackers are mistakenly tracking the opposite hand's motion source.
- Fix a bug where the right hand is not able to do grab or release.

[1.5.0] - 2023-06-16

This release breaks ABI/API compatibility with the previous versions in some areas as documented below.

Added

- Added HTC VIVE Focus 3 positional tracking hardware enum.
- Added support for the Meta Quest Pro, HTC VIVE, and HTC VIVE Focus 3 positional tracking hardware.
- Added two options to the wrist tracker settings (to the global plugin settings and the overrides in the wrist tracker component) in order to be able to specify a custom motion source for the left and right hands, so that it allows SteamVR-based trackers such as HTC VIVE or HTC VIVE Focus 3 to operate with the `SGPawn`.

Fixed

- Fix a bug where SteamVR trackers such as HTC VIVE and HTC VIVE Focus 3's wrist orientation and location were not being tracked.

Changed

- Fully refactored the top-level configurations in the settings system into ustructs.
- SenseGlove libraries have been updated to `v2.10.0-12133ac`.

Removed

- Dropped support for the Epic Native Toolchain `v19`, MSVC `v141` (Visual Studio 2017), and thus Unreal Engine `4.27` as it has been marked as deprecated since `v1.4.x`.

- Removed any kind of support for Oculus Touch (Oculus Rift S and Oculus Quest 1) positional tracking hardware, thus the enum as well.
- Removed any kind of support for Pico Neo 2 positional tracking hardware, thus the enum as well.
- Removed any kind of support for Pico Neo 3 positional tracking hardware, thus the enum as well.

[1.4.3] - 2023-06-01

This is a hotfix release addressing a critical Android crash.

Fixed

- Fix a critical Android crash that happens where the default development hand meshes are not found, which means almost always since we don't ship any default virtual hand mesh at the moment.
- Minor changelog release formatting fix in order to stay consistent.

[1.4.2] - 2023-06-01

This is a hotfix release addressing a few critical issues.

Fixed

- Fix build issues with certain compilers when the Unreal Engine version is older than `5.2`.
- Reintroduced the Virtual Hand and the Wrist Tracker debug gizmos which have temporarily been disabled due to a bug in the settings system.
- Some minor changelog fixes.

[1.4.1] - 2023-05-29

This is a bugfix release with a focus on Android build issues.

Fixed

- Fix an Android Gradle build issue that happens when the game's package name won't start with com.senseglove.*.
- Suppress a grade warning for non-arm64 architectures when the build target is Android.

Removed

- Remove dead Gradle code from the Android module.

[1.4.0] - 2023-05-19

This release breaks ABI/API compatibility with the previous versions.

Added

- Added support for the stable release of Unreal Engine `5.2` (the preview release has been supported since `v1.2.0`).
- Added Linux `AArch64` platform support.
- Added a new Grab component that can turn any actor into a grabbable object.
- Added a new Touch component that enables haptic feedback such as Buzz and Force-Feedback commands.
- Added an optional feature in order to automatically stop all haptics on the EndPlay event, wherever the virtual hand component is used. By default, it's enabled.

Fixed

- Fix Blueprint signatures for `USGVirtualHandComponentKismetLibrary` and make all the Blueprint exposed functions static.

Changed

- SenseGlove libraries have been updated to `v2.7.1-965f90c` with support for Linux `AArch64`.
- The Virtual Hand and the Wrist Tracker debug gizmos (the intended use is only for SenseGlove developers for really low-level stuff; thus won't affect the users of the plugin at all) have been disabled and will be ignored due to an esoteric bug in the settings systems which has been scheduled to be fixed in the future releases.

Removed

- Removed the redundant SenseGloveCoreTypes module which causes all kinds of packaging issues with certain versions of the engine.

Deprecated

- This is the last release to support Unreal Engine `4.27` and please keep in mind that the current release is not obtainable through the Unreal Engine Marketplace. The latest published version on the Marketplace for `4.27` is `v1.3.1`. Per Epic's Marketplace policy regarding Code Plugins, we are only able to distribute or update the SenseGlove plugin for the last three stable versions of Unreal Engine. As a result, we won't be able to publish updates or bug fixes for the older versions of the Engine except on rare occasions and only through our official repository on Microsoft Azure DevOps.

[1.3.1] - 2023-04-28

Fixed

- Fix RunUAT build issues caused by missing headers.
- Minor documentation fixes.

[1.3.0] - 2023-04-28

This release breaks ABI/API compatibility with the previous versions in addition to breaking coordinates systems conversions between Unreal Engine and the SenseGlove libraries.

Added

- A new generic SenseGlove Debug module.
- A debug virtual hand.

Fixed

- Fix the wrist tracker miscalculations for the Quest 2 controllers (other headsets might need fixing as well, in that case, future releases will address that).
- Minor code improvement and fixes.
- Minor documentation fixes.

Changed

- Breaking API/ABI changes in the Settings and the main `SenseGlove` module due to some settings refactoring.
- Breaking changes in the SenseGlove/Unreal coordinates systems conversions due to underlying changes in the SenseGlove Core Libraries.

- SenseGlove libraries have been updated to `v2.6.0-aac3d56`.

[1.2.1] - 2023-03-30

Fixed

- Fix RunUAT build issues with Android.

[1.2.0] - 2023-03-28

This release breaks ABI/API compatibility with the previous versions.

Added

- Android/Oculus on-device glove calibration.
- Introduced the animated Virtual Hand Model (as a set of virtual hand and wrist tracker components and an actor) with in-editor animation availability.
- Introduced `SGPawn`, `SGPlayerController`, `SGGameModeBase`, etc classes.
- Added an internal `SenseGloveCoreTypes` module in order to share common SenseGloveCore types between various modules.
- Segregated Android binaries for NDK `r21e` (UE `4.27` and `5.0`) and `r25b` (UE `5.1`, `5.2`).
- Fully functional and stable Linux development support.
- Fully functional and stable Unreal Engine `5.2` preview support has been added.
- Added a Plugin's settings manager and two new modules SenseGloveSettings and SenseGloveSettingsKismet.

Changed

- SenseGlove libraries have been updated to the Linux-aware version: `v2.5.0-8069342`.
- API has changed to use degrees instead of radians.
- SGCoordinates utility class name has been changed to `SGAngles` and now the plugin API uses degrees in contrast of SenseGlove libraries by default.
- Migrate common nested array types into the SenseGloveTypes module from the SenseGloveCore module.

Removed

- Removed a few thousand lines of archaic pre-public-release dead code.
- Dropped Android NDK `r21b` binaries used by the older engine versions.
- Purged the dead code for dropped engine versions by `v1.1.1` (`4.22`, `4.23`, `4.24`, `4.25`, and `4.26`) that carried over to the current version.
- Removed redundant `SGConnectImpl` / `SGPlatform`.
- Removed redundant `SGTypes` / `SGConnectTypes`.

Known Issues

- Wrist Tracker's offsets are a bit off (e.g. on Quest 2), scheduled to be fixed in the next patch release.

[1.1.1] - 2023-02-07

Added

- Initial support for the upcoming Unreal Engine `5.2`.
- Add support for Android `armeabi-v7a` with `neon`, `x86-64`, and `x86` builds in addition to `arm64-v8a`.

Fixed

- Fix various Android build issues.
- Some minor fixes and improvements.

Changed

- Bump SenseGlove libraries to `v2.1.2-95ec6e7`.

[1.1.0] - 2023-02-03

Added

- Whitelist Android as a target platform.
- Introduce Android support.
- Add third-party library SGConnect for Android `v1.1.0`.

Fixed

- Fix Android build issues caused by the log module.

Changed

- SGConnect and SGCore libraries have been updated to `v2.1.1-0569c74`.

Removed

- Removed the enum utils class due to `ANY_PACKAGE` deprecation warnings in Unreal Engine `5.1`.

- Support for older versions of the Engine (namely, `4.22`, `4.23`, `4.24`, `4.25`, and `4.26`) has been dropped.

[1.0.4] - 2022-12-02

This is a minor release focusing mostly on adherence to the Unreal Engine Marketplace Guidelines based on the feedback from Epic Games.

Added

- Added support for MSVC 2017.

Changed

- Updated SenseGlove libraries (SGCore/SGConnect) to `v2.0.4`.

[1.0.3] - 2022-11-29

This is a minor release focusing on adherence to the Unreal Engine Marketplace Guidelines based on the feedback from Epic Games.

Changed

- Adjust Config/FilterPlugin.ini in order to conform to Epic's Market Place Guidelines.

[1.0.2] - 2022-11-27

This is a minor release focusing on adherence to the Unreal Engine Marketplace Guidelines based on the feedback from Epic Games.

Added

- Added the newly acquired Unreal Engine Market Place Offer ID to the `.uplugin` file.
- List the dotfiles inside the FilterPlugin.ini file as well.
- Add the copyright notice to the source files missing it.
- Add the SenseGlove SDK license to the third-party license file.

Fixed

- Fix the readme typos and errors.
- Minor fixes in the changelog for previous releases.

[1.0.1] - 2022-11-25

Changed

- Exposed SenseGloveTypes as a public dependency in SenseGloveConnect and SenseGloveCore modules, so that the C++ users of the API don't need to explicitly add it as a dependency.
- Cleaned up the redundant headers/modules dependencies from SGCore headers.

Fixed

- Fix RunUAT build issues prior to Epic Store submission.

[1.0.0] - 2022-11-24

Added

- Initial public release of the SenseGlove haptic API for Unreal Engine with support for Microsoft Windows and GNU/Linux.

Directory Structure

```
/
   │
   ├── Config
   │
   ├── Documentation (this will be generated by running the <code>make</code>
command inside the Handbook directory)
   │
   ├── Handbook (this is the mdBook source code, used to generate the
Documentation folder and not distributed to [Fab](https://www.fab.com/))
   │
   ├── Resources
   │
   └── Source (various plug-in modules)
         │
         ├── SenseGlove (the UE-specific high-level API)
         │
         ├── SenseGloveAndroid (the Android-specific module)
         │
         ├── SenseGloveBackend (responsible for initialization and
deinitialization of the backend libraries)
         │
         ├── SenseGloveBackendKismet (exposes Blueprint-specific functionality
from the SenseGloveBackend module)
         │
         ├── SenseGloveBuildHacks (uses Exceptions and RTTI, internally used
for compiler-specific build hacks)
         │
         ├── SenseGloveConnect (exposes part of the SGConnect low-level API to
C++)
         │
         ├── SenseGloveConnectImpl (uses Exceptions and RTTI, intended for
internal use only)
         │
         ├── SenseGloveConnectKismet (SGConnect functionality exposed to
Blueprint)
         │
         ├── SenseGloveCore (exposes part of the SGCoreCpp low-level API to
C++)
         │
         ├── SenseGloveCoreImpl (uses Exceptions and RTTI, intended for
internal use only)
         │
         ├── SenseGloveCoreKismet (SGCoreCpp functionality exposed to
Blueprint)
         │
         ├── SenseGloveDebug (a utility debug module)
         │
         ├── SenseGloveDebugKismet (exposes Blueprint-specific functionality
from the SenseGloveDebug module)
         │
         ├── SenseGloveEditor (the Editor module)
         │
         ├── SenseGloveInterop (internally used for interoperability between
RTTI disabled/enabled modules)
         │
         ├── SenseGloveKismet (exposes Blueprint-specific functionality from
the SenseGlove module)
         │
         ├── SenseGloveLog (the internal log module)
         │
         ├── SenseGloveSettings (the plugin's settings manager)
         │
         ├── SenseGloveSettingsKismet (exposes Blueprint-specific
functionality from the SenseGloveSettings module)
         │
         ├── SenseGloveTracking (provides XR_EXT_hand_tracking support, HMD
auto-detection, and SenseGlove device tracking)
         │
         ├── SenseGloveTrackingKismet (exposes Blueprint-specific
functionality from the SenseGloveTracking module)
         │
         ├── SenseGloveTypes (exposes various enums from the backend libraries
and also types from the SenseGlove module)
         │
         ├── SenseGloveUtils (the internal utility module)
         │
         └── ThirdParty (3rd-party dependencies)
               │
               ├── android (.jar file Java libraries for Android)
               │     │
               │     ├── debug
               │     │
               │     └── release
               │
               ├── include (header files)
               │     │
               │     └── SenseGlove
               │           │
               │           ├── BLE (SGBLE headers)
               │           │
               │           ├── Common (SGCommon headers)
               │           │
               │       │
               │       ├── Connect (SGConnect headers)
               │       ├── Core (SGCore headers)
               │       └── Log (SGLog headers)
               ├── lib (platform-specific pre-built binary dependencies)
               │   ├── android
               │   │   └── r25b (Android NDK r25b dependencies for UE
 5.1+)
               │   │       ├── aarch64 (64-bit ARM variant of Android)
               │   │       │   ├── debug
               │   │       │   └── release
               │   │       └── x86-64 (64-bit x86-64 variant of Android)
               │   │           ├── debug
               │   │           └── release
               │   ├── linux
               │   │   ├── rustc (GNU/Linux binary dependencies built with
 Rust)
               │   │   │   ├── aarch64 (dependencies targeting GNU/Linux
 AArch64 architecture)
 │   │   │   │   ├── debug
 │   │   │   │   └── release
               │   │   │   └── x86-64 (dependencies targeting GNU/Linux
 x86-64 architecture)
 │   │   │       ├── debug
 │   │   │       └── release
               │   │   ├── v22 (5.4 GNU/Linux dependencies)
               │   │   │   ├── aarch64 (dependencies targeting GNU/Linux
 AArch64 architecture)
               │     │     │     │     │
               │     │     │     │     ├── debug
               │     │     │     │     │
               │     │     │     │     └── release
               │     │     │     │
               │     │     │     └── x86-64 (dependencies targeting GNU/Linux
 x86-64 architecture)
               │     │     │           │
               │     │     │           ├── debug
               │     │     │           │
               │     │     │           └── release
               │     │     │
               │     │     ├── v23 (UE 5.5 GNU/Linux dependencies)
               │     │     │     │
               │     │     │     ├── aarch64 (dependencies targeting GNU/Linux
 AArch64 architecture)
               │     │     │     │     │
               │     │     │     │     ├── debug
               │     │     │     │     │
               │     │     │     │     └── release
               │     │     │     │
               │     │     │     └── x86-64 (dependencies targeting GNU/Linux
 x86-64 architecture)
               │     │     │           │
               │     │     │           ├── debug
               │     │     │           │
               │     │     │           └── release
               │     │     │
               │     │     └── v25 (UE 5.6 GNU/Linux dependencies)
               │     │           │
               │     │           ├── aarch64 (dependencies targeting GNU/Linux
 AArch64 architecture)
               │     │           │     │
               │     │           │     ├── debug
               │     │           │     │
               │     │           │     └── release
               │     │           │
               │     │           └── x86-64 (dependencies targeting GNU/Linux
 x86-64 architecture)
               │     │                 │
               │     │                 ├── debug
               │     │                 │
               │     │                 └── release
               │     │
               │     └── windows
               │           │
               │           ├── v143 (Microsoft Visual Studio 2022
 dependencies)
               │           │     └── x86-64 (dependencies targeting Microsoft
 Windows x86-64 architecture)
               │           │           ├── debug
               │           │           └── release
               │           └── rustc (Microsoft Windows binary dependencies
 built with Rust)
               │                 └── x86-64 (dependencies targeting Microsoft
 Windows x86-64 architecture)
               │                       ├── debug
               │                       └── release
               ├── SGBleThirdPartyLibs (Third-party module providing SGBLE
 headers and libraries)
               ├── SGConnectThirdPartyHeaders (Third-party module providing
 SGConnect headers)
               ├── SGConnectThirdPartyLibs (Third-party module providing
 SGConnect headers and libraries)
               ├── SGCoreThirdPartyLibs (Third-party module providing SGCore
 headers and libraries)
               └── SGSerialThirdPartyLibs (Third-party module providing
 github.com/wjwwood/serial headers and libraries)
```

Extra Resources

There are various resources available for older versions of the SenseGlove Unreal Engine Plugin prior to `v2.1.x` that might still be partially relevant. These include example projects, demo scenes, and tutorials. Plans are underway to provide new example projects, demo scenes, and tutorials for the latest release. In the meantime, the outdated resources can still be beneficial

Examples and Demo Projects

- A basic OpenXR-compatible Blueprint demo demonstrating basic functionality such as grab/release, touch with buzz and force-feedback, etc (compatible with versions `v2.1.0+`).
- A basic Blueprint demo demonstrating basic functionality such as grab/release, touch with buzz and force-feedback, etc (compatible with versions >= `v1.4.x` and <= `v2.0.x)` .
- Example C++ API Project (only compatible with early `v1.x.x` releases)
- Example Blueprint API Project (only compatible with early `v1.x.x` releases)

Third-Party OpenXR Integration Demos

- A VRExpansionPlugin Integration Demo for UE `5.4`
  - Documentation

Tutorials

- Finding out your SenseGlove plugin version
- Plugin installation guide for Microsoft Windows
- C++ & Blueprint examples for Microsoft Windows
- Plugin and examples installation guide for GNU/Linux

- How to connect to Nova gloves on GNU/Linux using Blueman Bluetooth Manager
- How to connect to Nova gloves on GNU/Linux using command-line
- The basic C++ and Blueprint API usage
- How to setup the virtual hand model & the SenseGlove pawn
- How to deploy to Oculus Quest 2 and Android
- Setting up Grabbing and Haptic Feedback functionalities (SGBasicDemo)
- Setting up VIVE Pro & VIVE Trackers in Unreal Engine
- Setting up VIVE Focus 3 & VIVE Wrist Trackers in Unreal Engine
- SGBasicDemo: setup throwing objects and physics settings for the real and virtual hands
- SGBasicDemo v2: upgrading your projects to the SenseGlove Unreal Engine Plugin v2.0.0

Third-Party Tutorials

OpenXR Tutorials

- Introduction to Virtual Reality, OpenXR Hand-Tracking, and Gesture Detection in Unreal Engine
- Procedural Virtual Hand Mesh Animation Using OpenXR Hand-Tracking Data - Part 1
- Procedural Virtual Hand Mesh Animation Using OpenXR Hand-Tracking Data - Part 2
- Unreal Engine OpenXR Hand-Tracking on Android with Meta XR (Quest 3S/3/Pro/2) and HTC VIVE OpenXR (Focus Vision/XR Elite/Focus 3) Plugins

Android (Meta Quest / HTC VIVE) Tutorials

- Build & Deploy Unreal Engine 5.5 Projects APK to Android & Meta Quest 3S/3/Pro/2 in Standalone Mode
- Unreal Engine OpenXR Hand-Tracking on Android with Meta XR (Quest 3S/3/Pro/2) and HTC VIVE OpenXR (Focus Vision/XR Elite/Focus 3) Plugins

VR Optimization

- Optimizing Unreal Engine VR Projects for Higher Framerates (Meta Quest, HTC VIVE, FFR, ETFR, NVIDIA DLSS, AMD FSR, and Intel XeSS Tips Included!)

SenseGlove Unreal Engine Plugin License

The SenseGlove Unreal Engine Plugin is licensed under the terms of the MIT License. Below is the MIT License:

Please note that while the SenseGlove Unreal Engine Plugin is made available under the MIT License, it utilizes a few third-party libraries with permissive free licenses as well, in order to power various components. For a list of these libraries and their own respective open-source licenses take a look at the third-party licenses, please.

SenseGlove Unreal Engine Handbook License

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 -- License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

   1. If You Share the Licensed Material (including in modified form), You must:

      a. retain the following if it is supplied by the Licensor with the Licensed Material:

         i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

         ii. a copyright notice;

         iii. a notice that refers to this Public License;

         iv. a notice that refers to the disclaimer of warranties;

         v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

Third Party Licenses

Please note that while the SenseGlove Unreal Engine Plugin is made available under the MIT License, it utilizes a few third-party libraries with permissive free licenses as well, in order to power various components.

The following third-party software are used and shipped with the SenseGlove Unreal Engine Plugin:

- The SenseGlove SDK (a.k.a. SenseGlove Backend Libraries, or SenseGlove Core Libraries)
- SGBLE and SGBLExx Rust Dependencies
- The Boost C++ Libraries
- The {fmt} Formatting Library
- The Loguru Logging Library License
- The Serial Communication Library

For more information consult their own respective open-source licenses, please.

SenseGlove SDK License

```
SENSEGLOVE SDK LICENSE

Copyright (c) 2020 - 2026 SenseGlove

Purchase of the Product does not entitle you to ownership or a license to any
software generated by SenseGlove for use with the Product (the “Software”).
To the extent that SenseGlove, in its sole discretion, grants you access to
any
such Software, the Software is licensed by us or by the relevant
licensor/owner
subject to the relevant end-user license agreement or other license terms
included with the Product and/or on the SenseGlove Websites including the
Github
page of SenseGlove (the “License Terms“).

Specifically, SenseGlove shall have sole discretion to determine and change
the
availability, nature, features, content, versioning of any Software that it
makes available to you, for download through the the Github page of
SenseGlove
or otherwise (including the SenseGlove software developer kit (“SDK”)).
Purchase of a Product does not entitle you to access to any specific
features,
content or version of the SDK, including and especially versions of the SDK
that
have not yet been made available to the public. SenseGlove will have no
obligation to provide any updates or upgrades to any Software it makes
available
to you, but in the event that it does, such updates, upgrades and any
documentation will be subject to the License Terms available at
https://www.senseglove.com/solutions/.

Except to the extent expressly provided by us in writing or under the License
Terms, the Software is provided “AS IS” without any warranties, terms or
conditions as to quality, fitness for purpose, non-infringement, performance
or
correspondence with description and we do not offer any warranties or
guarantees
in relation to the Software installation, configuration or error/defect
correction.
```

SGBLE and SGBLExx Rust Dependency Licenses

SGBLE and SGBLExx, components of the SenseGlove SDK, are built using the Rust programming language. These libraries rely on a variety of open-source Rust crates. To ensure commercial developers can use our SDK without licensing concerns, we strictly rely on crates with **permissive licenses at runtime**.

Some dependencies, like `r-efi`, are dual-licensed and allow permissive use. The **only copyleft tool** used is `cbindgen`, which is solely a **build-time dependency**. It is employed to automatically generate C FFI bindings for SGBLE, which is implemented in pure Rust. Since `cbindgen` is not required at runtime and acts more like a transpiler (source-to-source compiler), it does **not introduce any licensing risks** for commercial applications.

Below is a comprehensive list of all third-party crates and their corresponding licenses used in SGBLE and SGBLExx:

hook-registry, smallvec, smol_str, socket2, stable_deref_trait, static_assertions, syn, tempfile, thiserror, thiserror, thiserror-impl, thiserror-impl, thread_local, toml, toml_datetime, toml_edit, ttf-parser, type-map, typenum, unicode-segmentation, unicode-width, unicode-xid, url, utf16_iter, utf8_iter, utf8parse, uuid, version_check, wasm-bindgen, wasm-bindgen-backend, wasm-bindgen-futures, wasm-bindgen-macro, wasm-bindgen-macro-support, wasm-bindgen-shared, web-sys, web-time, webbrowser, weezl, wgpu, wgpu-core, wgpu-hal, wgpu-types, winapi, winapi-i686-pc-windows-gnu, winapi-x86_64-pc-windows-gnu, windows, windows, windows-core, windows-core, windows-implement, windows-implement, windows-interface, windows-interface, windows-result, windows-result, windows-strings, windows-sys, windows-sys, windows-sys, windows-targets, windows-targets, windows-targets, windows_aarch64_gnullvm, windows_aarch64_gnullvm, windows_aarch64_gnullvm, windows_aarch64_msvc, windows_aarch64_msvc, windows_aarch64_msvc, windows_i686_gnu, windows_i686_gnu, windows_i686_gnu, windows_i686_gnullvm, windows_i686_msvc, windows_i686_msvc, windows_i686_msvc, windows_x86_64_gnu, windows_x86_64_gnu, windows_x86_64_gnu, windows_x86_64_gnullvm, windows_x86_64_gnullvm, windows_x86_64_gnullvm, windows_x86_64_msvc, windows_x86_64_msvc, windows_x86_64_msvc, write16, x11rb, x11rb-protocol
Apache-2.0 OR MIT OR Zlib (12): bytemuck, bytemuck_derive, cursor-icon, glow, miniz_oxide, objc2-app-kit, objc2-core-bluetooth, objc2-core-foundation, objc2-core-graphics, objc2-io-surface, raw-window-handle, xkeysym
BSD-2-Clause (1): arrayref
BSD-3-Clause (3): jni-utils, tiny-skia, tiny-skia-path
BSL-1.0 (2): clipboard-win, error-code
CC0-1.0 (1): hexf-parse
ISC (1): libloading
MIT (91): android-properties, block, block2, block2, bytes, calloop, calloop-wayland-source, cfg_aliases, combine, dashmap, dispatch, dlib, endi, generic-array, glutin-winit, is-terminal, libredox, malloc_buf, memoffset, mio, nix, objc, objc-sys, objc2, objc2, objc2-app-kit, objc2-cloud-kit, objc2-contacts, objc2-core-data, objc2-core-image, objc2-core-location, objc2-encode, objc2-foundation, objc2-foundation, objc2-link-presentation, objc2-metal, objc2-quartz-core, objc2-symbols, objc2-ui-kit, objc2-uniform-type-identifiers, objc2-user-notifications, orbclient, ordered-float, quick-xml, quick-xml, redox_syscall, redox_syscall, sctk-adwaita, serde-xml-rs, simd-adler32, slab, smithay-client-toolkit, smithay-clipboard, strict-num, strsim, strum, strum_macros, synstructure, tiff, tokio, tokio-macros, tokio-stream, tokio-util, tracing, tracing-attributes, tracing-core, uds_windows, wayland-backend, wayland-client, wayland-csd-frame, wayland-cursor, wayland-protocols, wayland-protocols-plasma, wayland-protocols-wlr, wayland-scanner, wayland-sys, winnow, x11-dl, xcursor, xdg-home, xkbcommon-dl, xml-rs, zbus, zbus-lockstep, zbus-lockstep-macros, zbus_macros, zbus_names, zbus_xml, zvariant, zvariant_derive, zvariant_utils
MIT OR Unlicense (7): aho-corasick, byteorder-lite, memchr, same-file, termcolor, walkdir, winapi-util
MPL-2.0 (1): cbindgen
N/A (2): sgble, sgblueman

```
Unicode-3.0 (19): icu_collections, icu_locid, icu_locid_transform,
icu_locid_transform_data, icu_normalizer, icu_normalizer_data,
icu_properties, icu_properties_data, icu_provider, icu_provider_macros,
litemap, tinystr, writeable, yoke, yoke-derive, zerofrom, zerofrom-derive,
zerovec, zerovec-derive
Zlib (2): foldhash, slotmap
```

For detailed terms of use, please refer to the license files in each project's upstream repository.

Boost C++ Libraries License

{fmt} Formatting Library License

Loguru Logging Library License

Serial Communication Library License

Build Information

	The SenseGlove Unreal Engine Handbook
Handbook Revision	2.8
Handbook Revision URL	https://unreal.docs.senseglove.com/2.8
Handbook PDF URL	https://unreal.docs.senseglove.com/2.8/the-senseglove-unreal-engine-handbook-2.8.pdf
Handbook ePub URL	https://unreal.docs.senseglove.com/2.8/the-senseglove-unreal-engine-handbook-2.8.epub
Git Branch	master
Git Tag	v2.8.0
Git Commit	e6156463
Git Commits Since Tag	0
Git Tree State	clean
Git Is Shallow Clone	no
Git Latest Remote Tag	v2.8.0
Git Version	v2.8.0
Git Version Major	2
Git Version Minor	8
Git Version Patch	0
Plugin Version	v2.8.0
Plugin Version Major	2

The SenseGlove Unreal Engine Handbook	
Plugin Version Minor	8
Plugin Version Patch	0
Build Host	mamadou-dev-pc
Build Time	Tue Feb 24, 2026 11:29 CET +0100