Table of contents

Getting Started With Unreal SDK

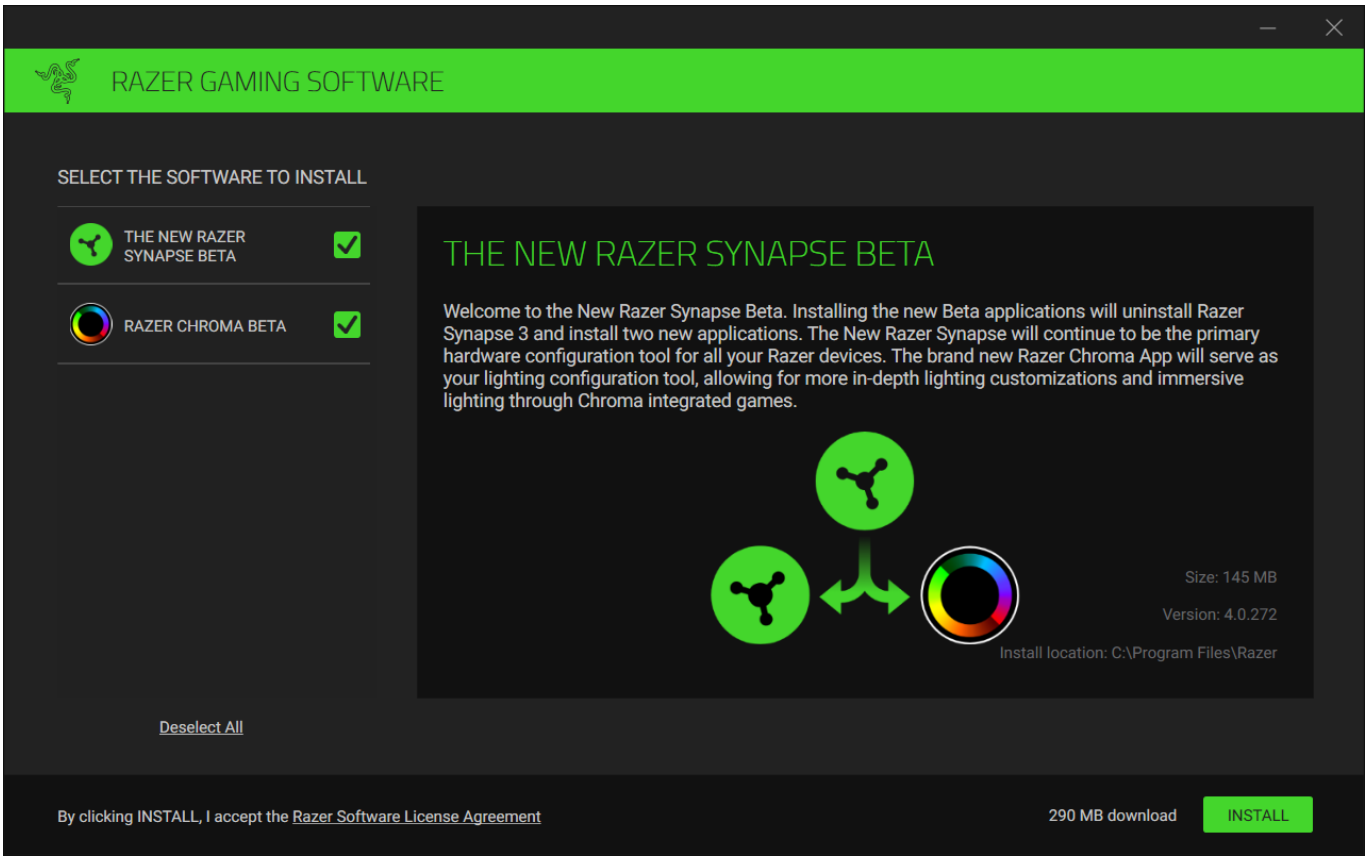
This Chroma SDK plugin has been tested with `Unreal` versions 4.21 through 5.4.

User Privacy

Note: The Chroma SDK requires only the minimum amount of information necessary to operate during initialization, including the title of the application or game, description of the application or game, application or game author, and application or game support contact information. This information is displayed in the Chroma app. The Chroma SDK does not monitor or collect any personal data related to users.

Dependencies

To use the Chroma SDK first install the new Razer Synapse and Chroma App.



- If you don't have Chroma hardware, you can see Chroma effects with the Chroma Emulator

---

SDK Integration

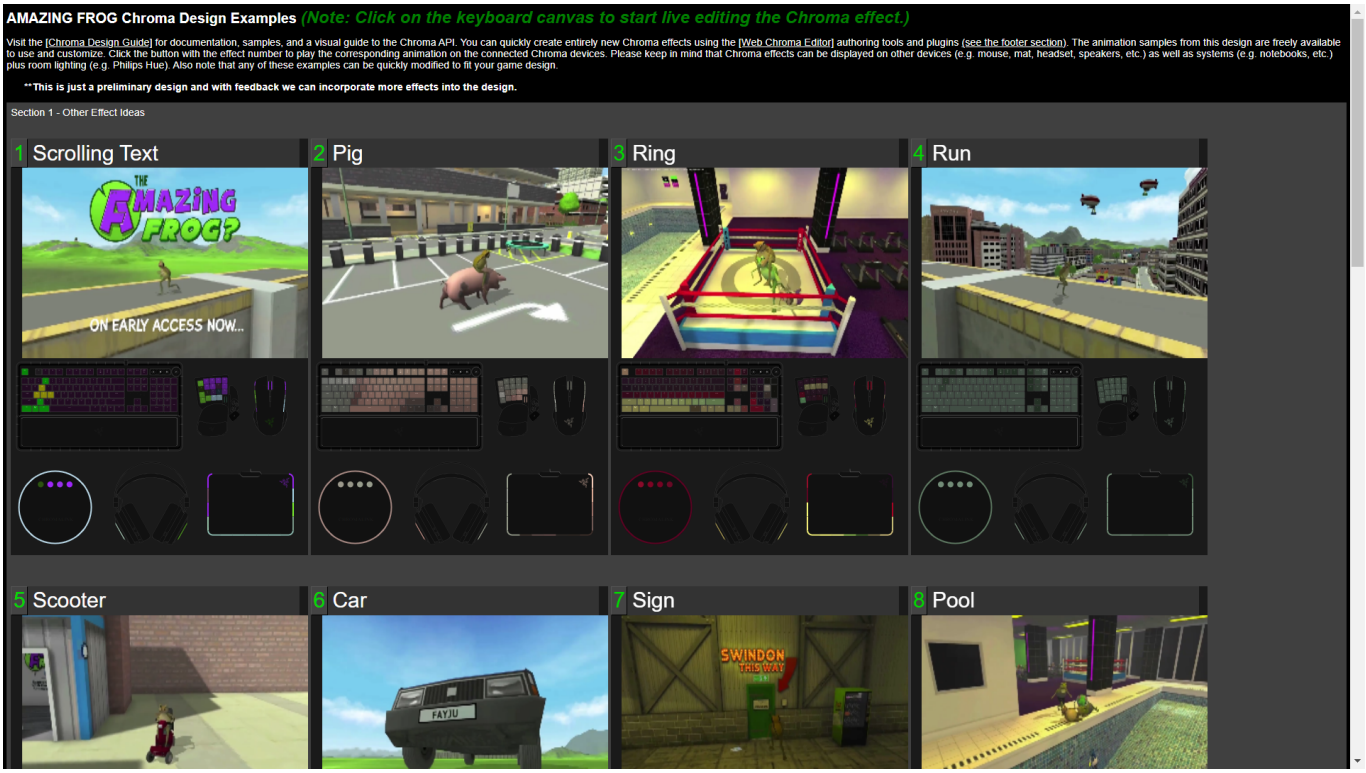
The SDK integration process involves the following:

1. Chroma Design
2. Revisions
3. Sample Project
4. Tools
5. Integration
6. Testing
7. Haptic Design

8. Modding

Chroma Design

The Chroma Design is the starting point. The team provides 15 sample effects that play on an animated web page. The sample effects correspond to short gameplay video clips and give an idea to the type of animation that could play for a set of game events. The samples are available to use for the specified effect or can be used for any other effect which is completely up to the developer. The developer may ask for effect revisions or additional sample effects. If gameplay video is not available, the developer can provide a description or reference art to conceptualize the desired effect.



Revisions

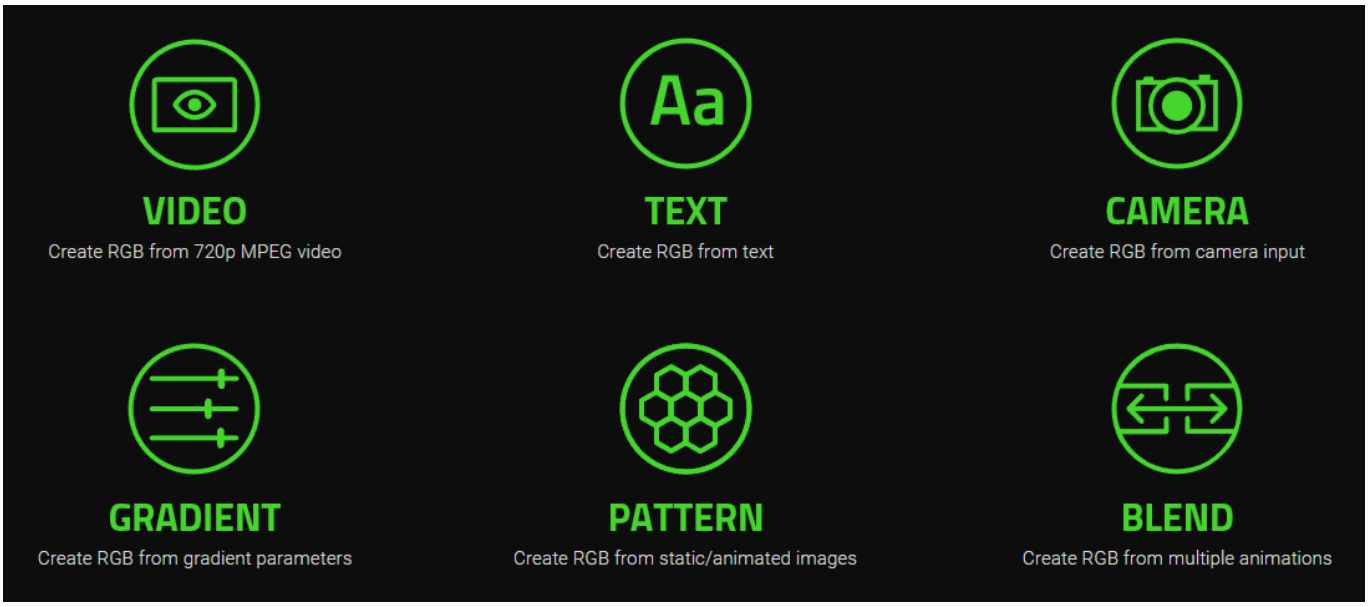
Some Chroma Designs require revisions to add more requested effects or to make changes through the feedback of reviewing the Chroma Design. Revisions can be requested which result in a subset of alterations from the previous design or add completely new game events. **Fill out the Chroma_Sensa_Template_Developers.xlsx Template which provides all the necessary fields for making design requests and revisions.**

Sample Project

The developer specifies which game engine is used by the game so that a sample project can be shared with sample code for the specified engine. The sample project will have the same effects that were defined in the Chroma Design and ported to the target language/game engine. The sample project will include a plugin to add the Chroma SDK to the specified game engine, and the ported sample code and sample animations from the `Chroma Design`.

Tools

- The Web Chroma Editor creates Chroma animations and code snippets from several input sources. Designers can create Chroma animations without writing any code. The toolset can use input sources as video, text, camera, web cam, desktop capture, gradients, patterns, images, and blended animations.



- The Chroma Design Converter can automatically port a web based Chroma Design to several languages and game engines.
- The Synesthesia Console can generate haptic configurations automatically for your Chroma integration.

Integration

The integration process can be as easy as copy and paste from the sample project into the game code. Most likely, it's a matter of finding game triggers in the game code to find the optimal place to add a call to `PlayAnimation()`. The typical Chroma integration process lasts 3 - 5 days for a single developer. Haptics integration can take 0 days by using automatic mode. Manually adding haptics can take about the same amount of work as Chroma to add the calls to `SetEventName()` in the right places. Chroma and haptics are independent meaning sometimes they play together and sometimes they play separately, which is completely up to the designer. **In most cases for game engines after the game build completes, the Chroma animations need to be copied to the animation folder within the game's content folder.**

Testing

The team can provide QA on the game build when integration has completed. Steam beta keys and Epic Store beta keys make testing possible before a game launches. This can be a good way to provide design revisions by testing and giving feedback on the build. To support the QA process, it will be important to include a level selector and potentially console commands that make it easy to navigate the build to test the game triggers at the right moments to validate the visuals work as expected. Beta key access is limited to the engineering and QA review team.

Haptic Design

Just like Chroma Designs, the Haptic Design can be provided by the team. Adding haptic support does not require adding assets to the game. Haptics can be added to a game without code changes and after the game has released. Haptics can be added through creation of a haptic configuration file. Developers can use the

Synesthesia Console which automates creation of the haptic configuration file within `HapticFolders` and will add some mockup haptic files (simple haptic effect which can be edited with Haptic Composer) when event names follow a naming convention. Haptic configuration files are automatically distributed by the team through `Chroma App` updates.

Modding

The decision to add Chroma mod support for a title is completely up to the developer. If the developer decides to block modding, Chroma animations can be loaded from a byte array which sandboxes and protects against any modifications to the Chroma animation assets. If the developer wants to use modding, Chroma animation assets are placed within the installation directory. Modders can modify the Chroma animations assets that are loaded by the title. The API provides `CloseAnimation` which reloads the Chroma animation from disk. This allows Chroma animations to be modified externally without needing to relaunch the title. Chroma animation playback also supports relative paths from the content folder. Relative paths can be used to organize several mods within the content folder. The title can have a configuration menu that switches between mod subfolder names which changes the relative path for loading the Chroma animations. The C++ Chroma Mod Sample shows how relative paths can be used to detect and use mods, which is applicable for any game or custom engine.

General

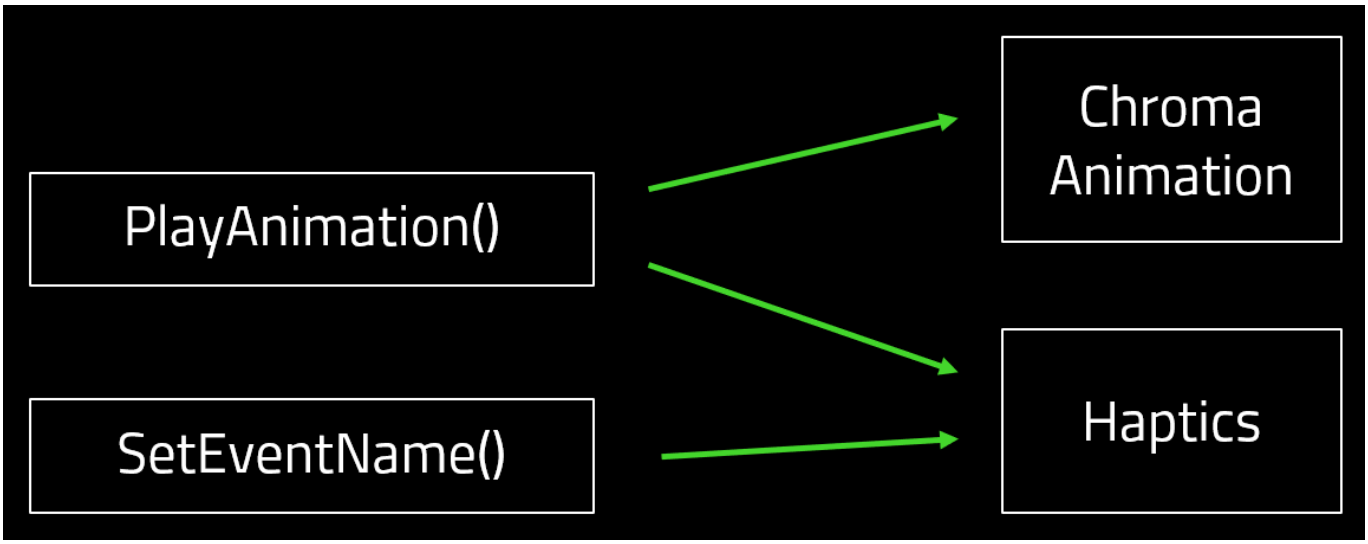
- The Chroma SDK allows an application or game to set the details in the Chroma Apps list within the `Chroma App`.
- The Chroma Plugin `Unreal SDK for Chroma` is available at Unreal_ChromaSDK on Github.

This document provides a guide to integrating Chroma RGB using the Chroma Unreal SDK. Chroma can be included through premade Chroma animations or APIs. Here is the list of available methods:

- Initialize SDK: Initialize the Chroma SDK to use the library.
- Is Active: Check if the app/game has Chroma focus.
- Is Connected: Check if Chroma hardware is connected.
- Play Chroma Animation: Playback a Chroma animation asset.
- Set Event Name: Name a game event or game trigger in order to also add Haptics to the Chroma event.
- Use Forward Chroma Events: Enable or disable automatic invocation of `SetEventName()` when invoking `PlayAnimation()` using the animation name.

Chroma Sensa

Chroma Sensa is the combination of Chroma and Razer Sensa HD Haptics in a single SDK. By integrating RGB lighting and haptics into game environments and events, players can enjoy a truly immersive gaming experience. The `Chroma SDK` is capable of playing Chroma animations and haptics on the Razer Sensa HD Haptics devices. The default mode allows automatic triggering of haptics effects when Chroma animations are played with `PlayAnimation()`. Manual mode is set by `UseForwardChromaEvents(false)` and haptics can be triggered independently of Chroma animations with SetEventName().



Event names can follow a naming convention which assists with the generation of the haptics configuration for your title. Event names are specified with the `SetEventName()` method. The event name suffix can be left off or used to prepopulate common settings for `_ON`, `_OFF`, and `_MERGE`.

- "Jump" - (without a suffix) Existing haptics stop, the named haptic plays to completion and then ends
- "Attack_ON" - Existing haptics continue to play, the named haptic plays as a continuous looping haptic
- "Attack_OFF" - Existing haptics continue to play, the named looping haptic stops
- "Punch_MERGE" - Existing haptics continue to play, the named haptic plays to completion and ends
- "Block_MERGE" - Existing haptics continue to play, the named haptic plays to completion and ends

Upon completion of Chroma and haptic implementation, the list of Chroma events and game triggers should be shared with the team to be add to the game's Chroma Workshop entry.

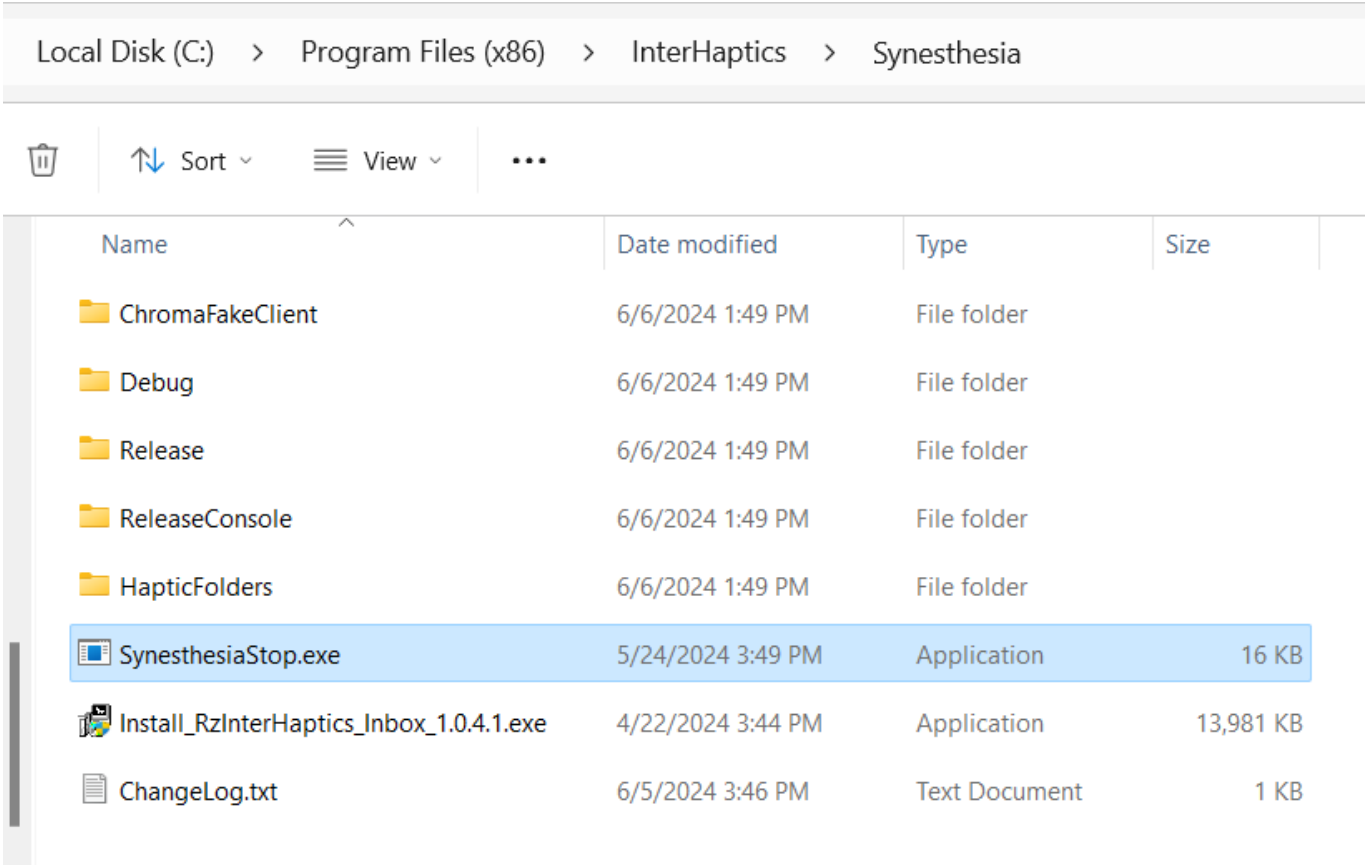
Targeting features can be **optionally** described for each haptics effect.

- "Target" defaults to `"All"`. GroupID options can be found at https://www.interhaptics.com/doc/interhaptics-engine/#groupid
- "Spatialization" defaults to `"Global"`. Other LateralFlag options can be found at https://www.interhaptics.com/doc/interhaptics-engine/#lateralflag
- "Gain" defaults to 1.0.

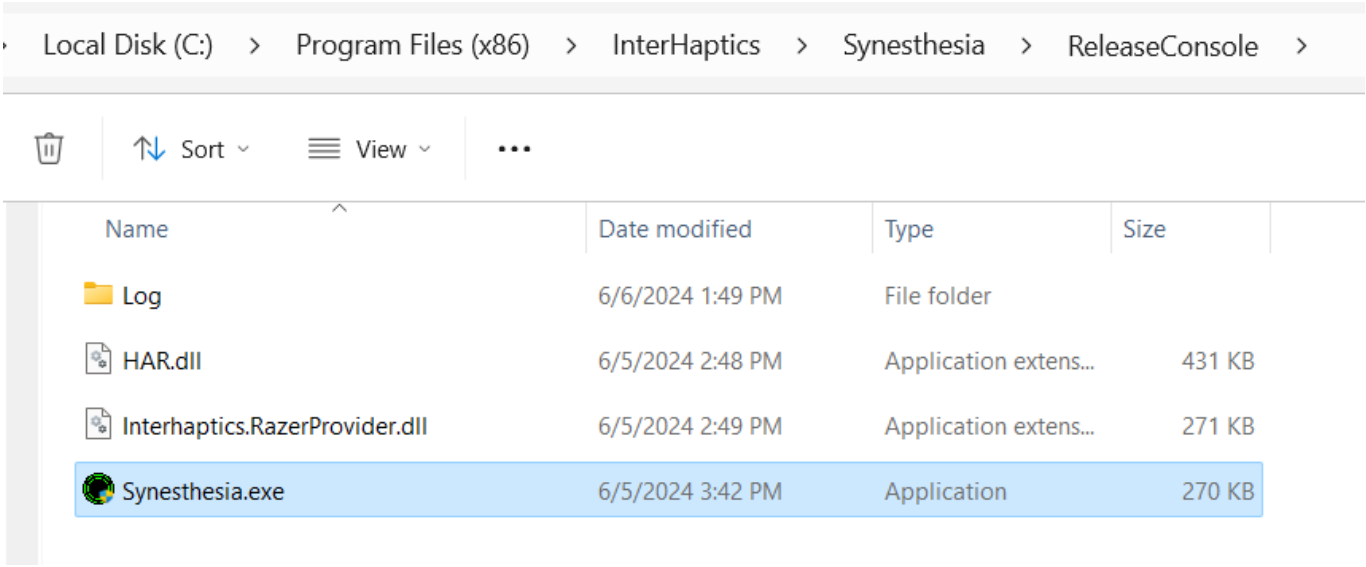
Synesthesia

The Synesthesia Console makes creating the haptics configuration for game integration super easy. Download and run the installer to get started creating a haptics config.

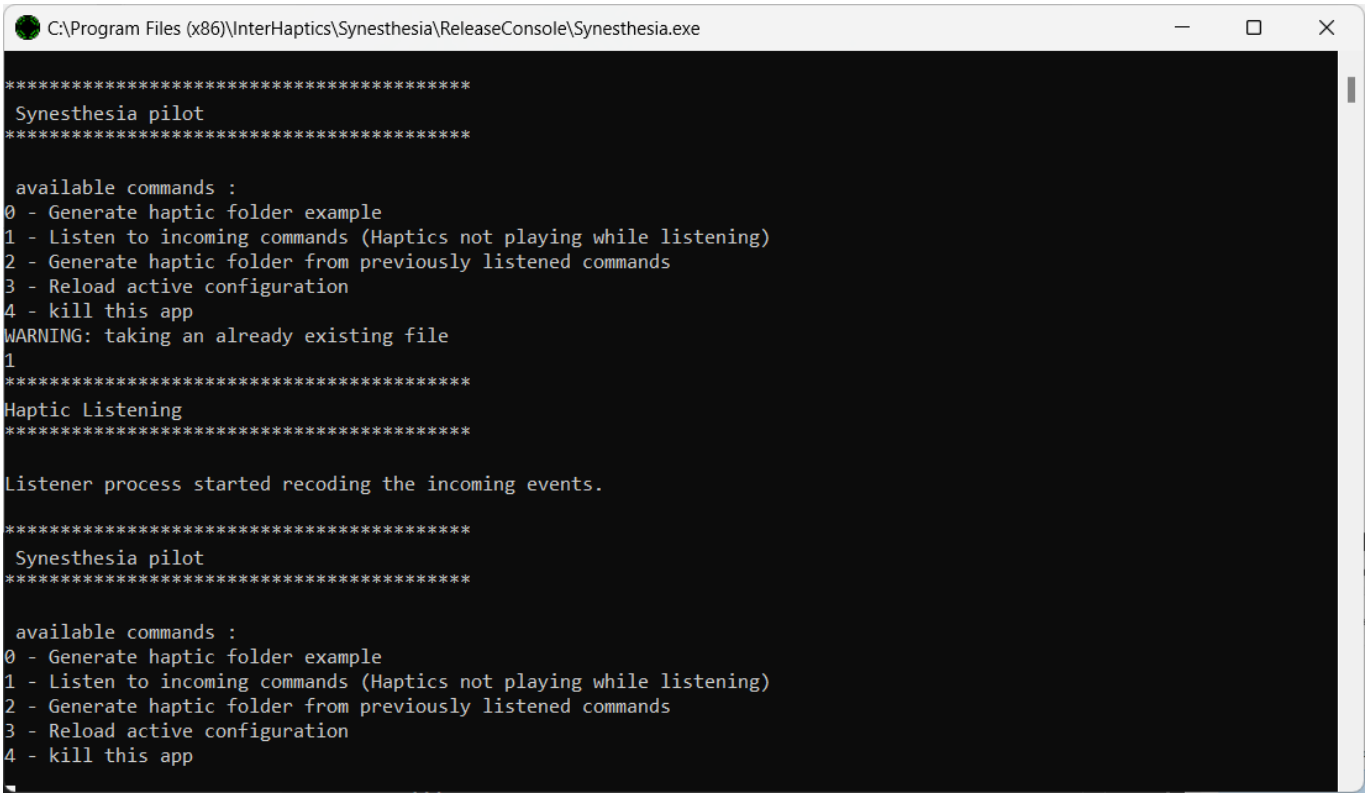
1. Run `SynesthesiaStop.exe` to stop any existing background or haptic consoles



2. Run the `Synesthesia Console` for the interactive prompt



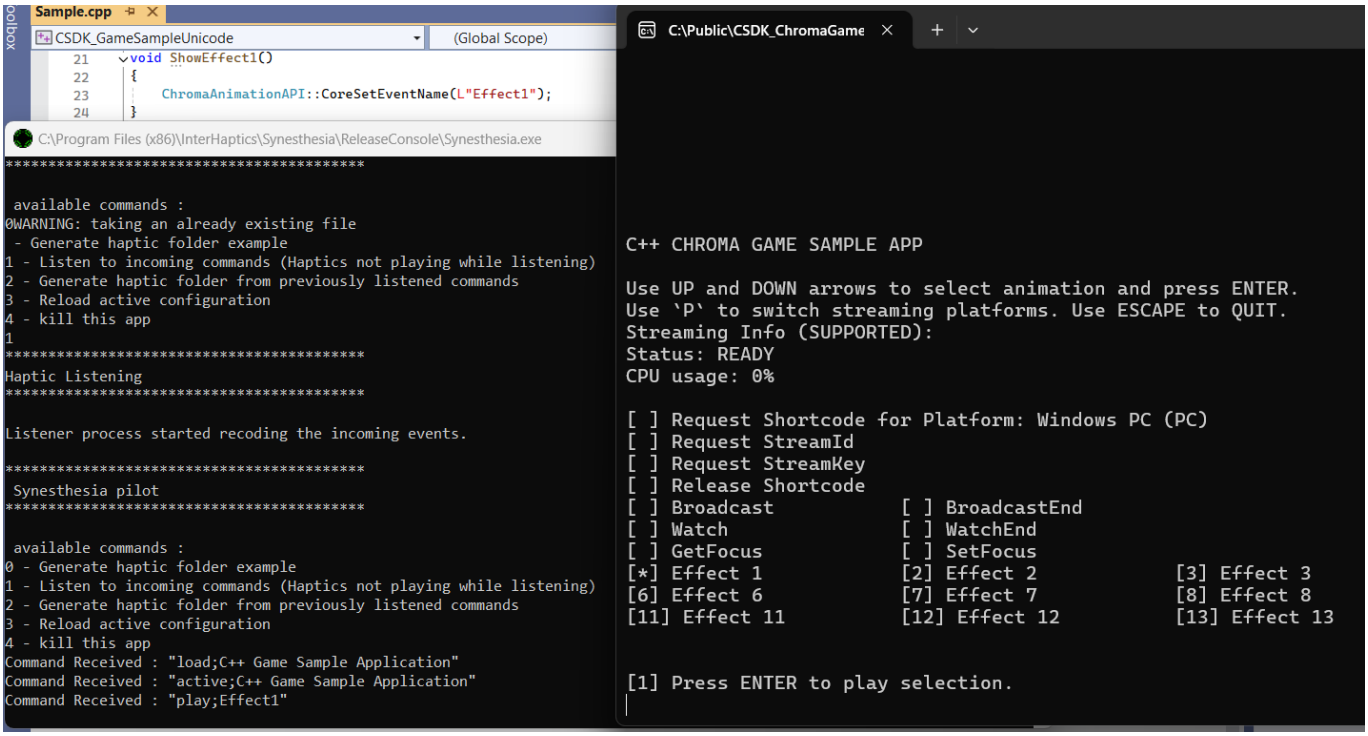
3. Enter option `1` and press `Enter` to listen for incoming commands



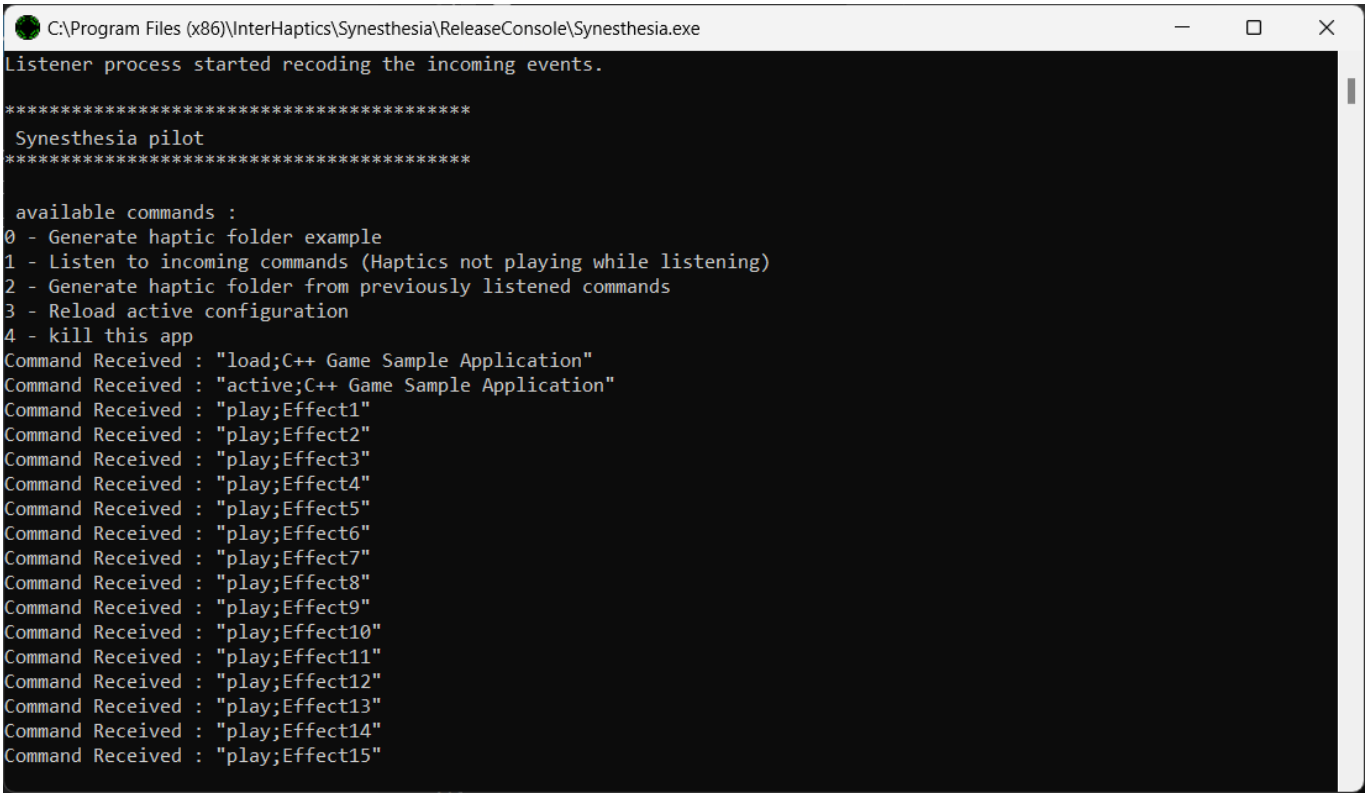
4. Launch your game that uses `PlayAnimation` or `SetEvent` directly to trigger haptic commands.

When the application launches and initializes Chroma, the command to `load` the haptic configuration file is sent. When the application receives Chroma focus, the `active` command is sent. When `PlayAnimation` or `SetEvent` is called, the `play` command is sent.

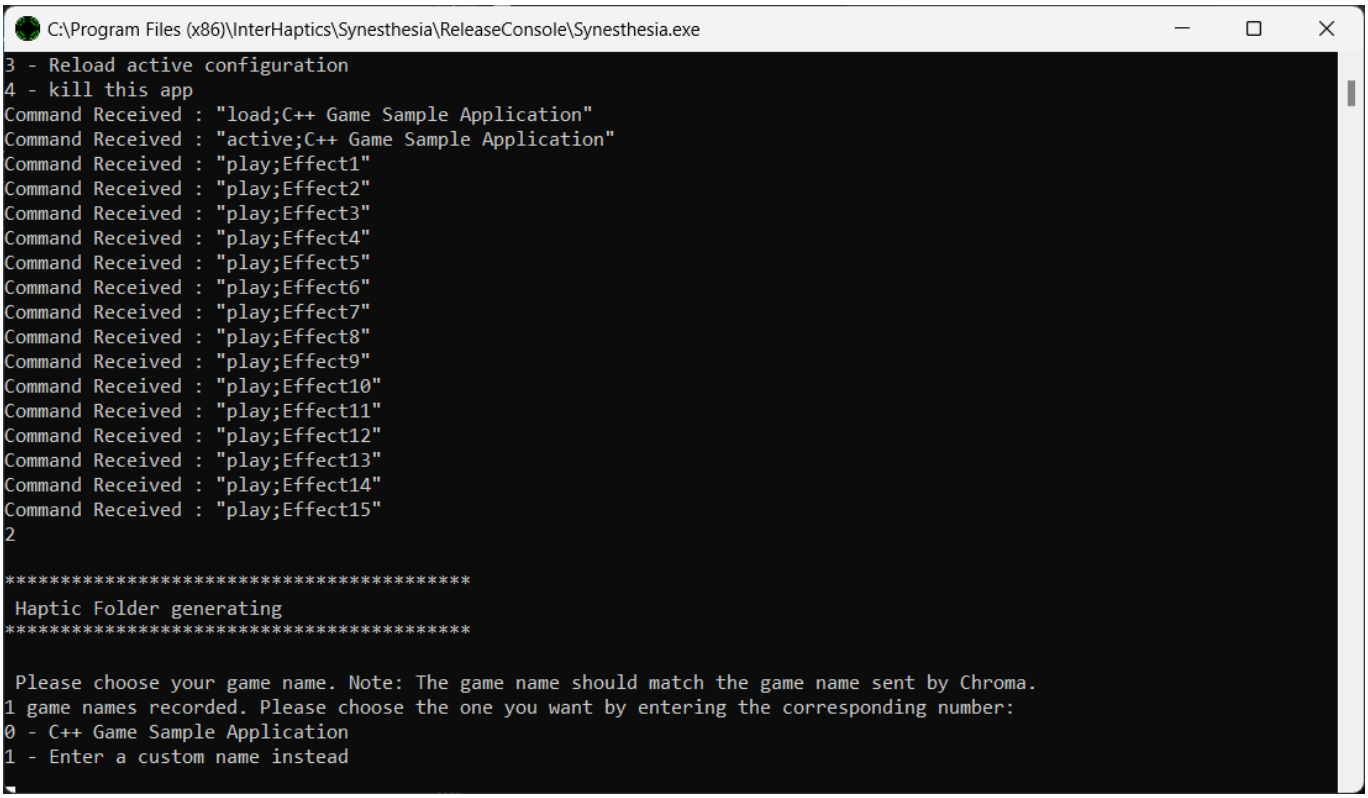
```
Command Received : "load;C++ Game Sample Application"
Command Received : "active;C++ Game Sample Application"
Command Received : "play;Effect1"
```



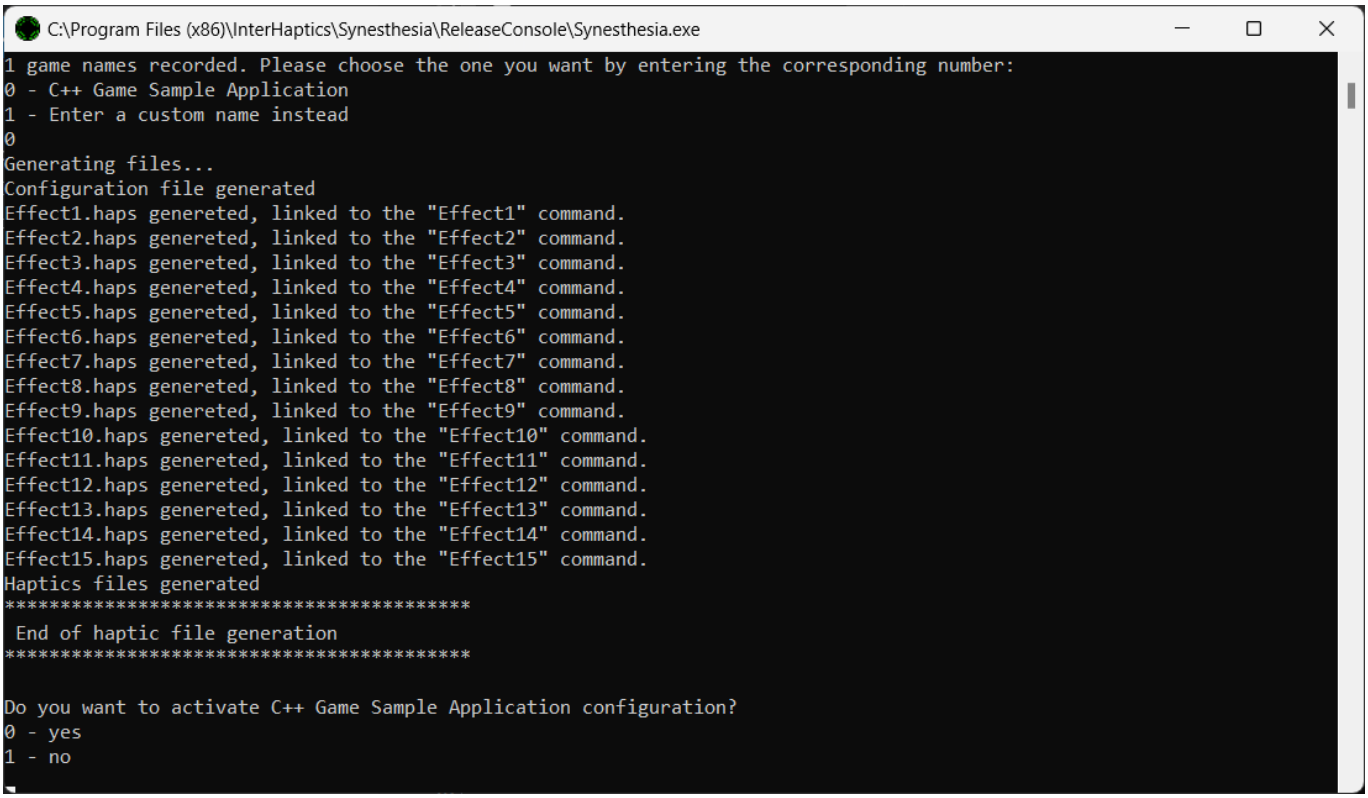
5. Play through all the game triggers to send any possible commands the game might use. This will be useful for generating the haptic configuration next.



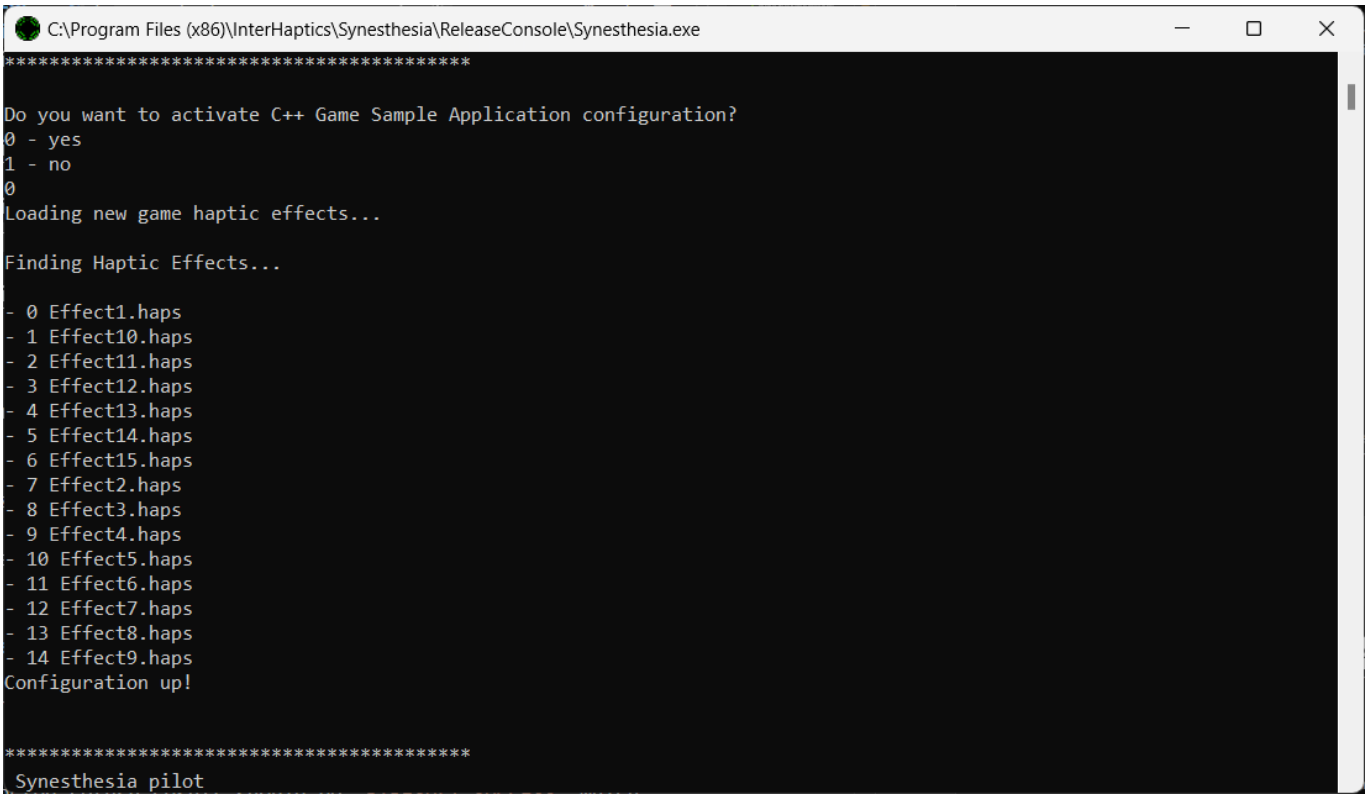
6. Enter option `2` and press `Enter` to generate the haptics configuration



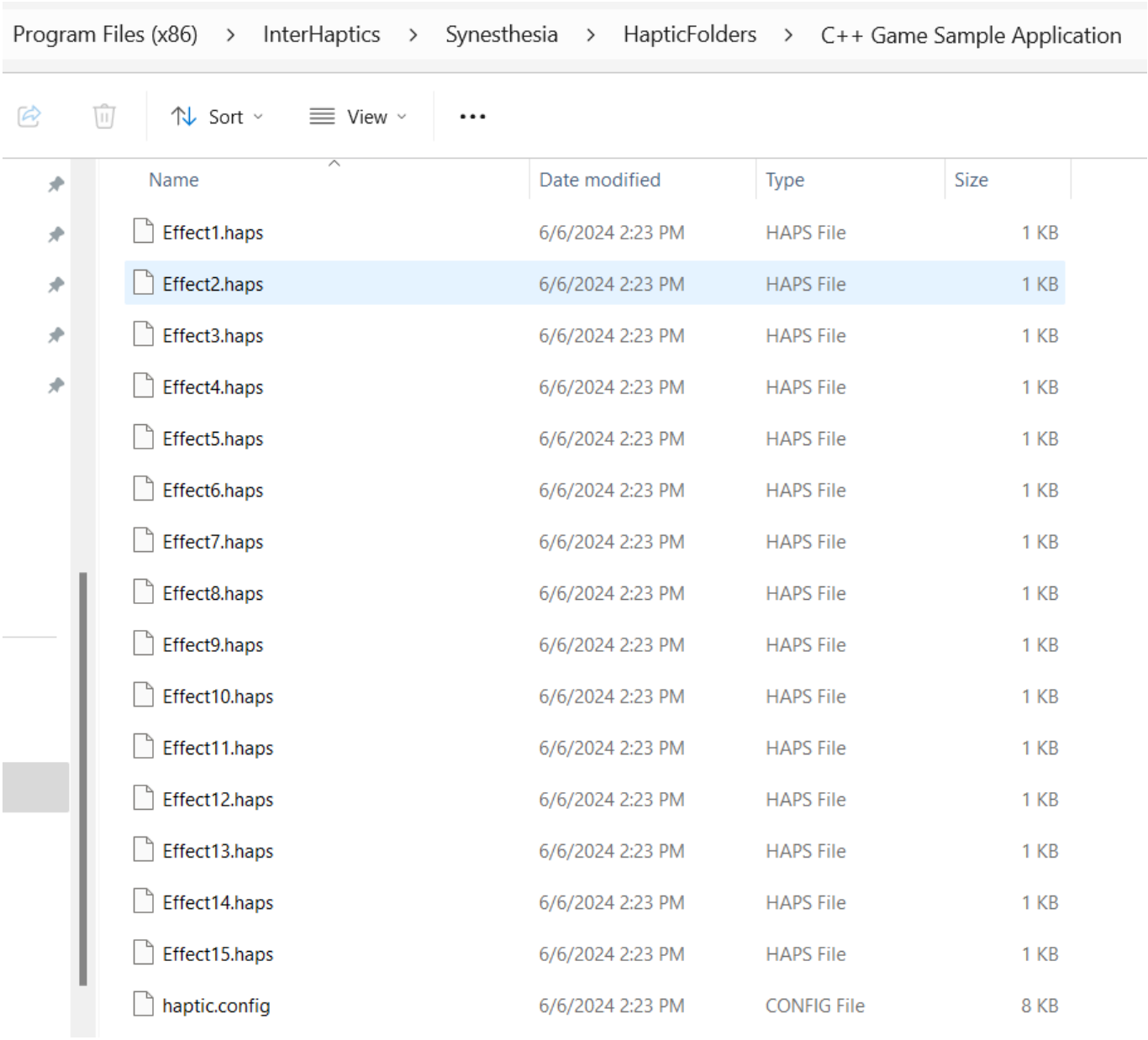
7. Enter option `0` and press `Enter` to use the detected application name used by the Chroma initialization



8. Enter option `0` and press `Enter` to use activate the new haptic configuration file. Now when the game triggers haptic events, the configured haptic events will play.



The `haptic.config` and `haps` default haptics effects were generated in the `HapticFolders` by the console.



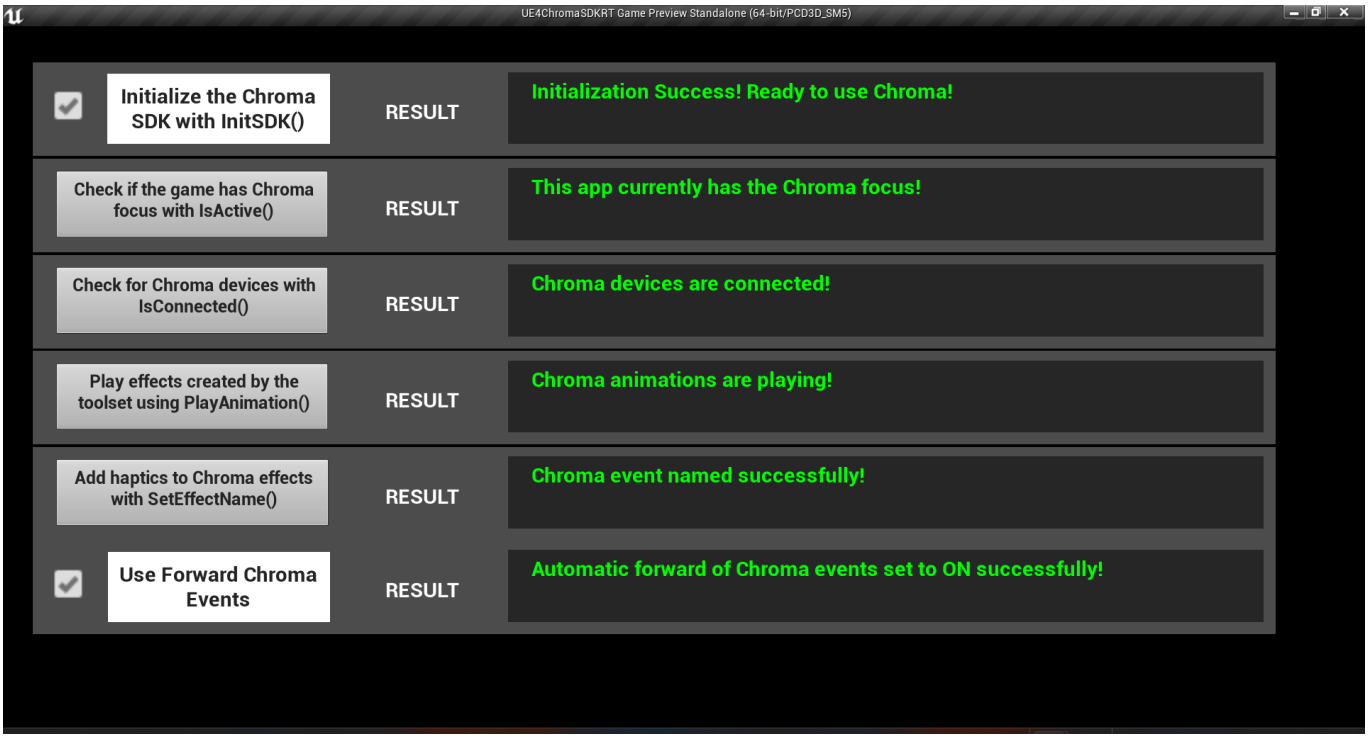
The `haptic.config` contains default targeting for the generated entries for each detected command.

```
{
    "ExternalCommands": [
        {
            "External_Command_ID": "Effect1",
            "Haptic_Events": [
                {
                    "Haptic_Effect": "Effect1",
                    "Loop": 1,
                    "Mixing": "Override",
                    "Targeting": [
                        {
                            "Gain": 1.0,
                            "Spatialization": "Global",
                            "Target": "All"
                        }
                    ]
                }
```

```
            ]
        },
        ...
    ]
}
```

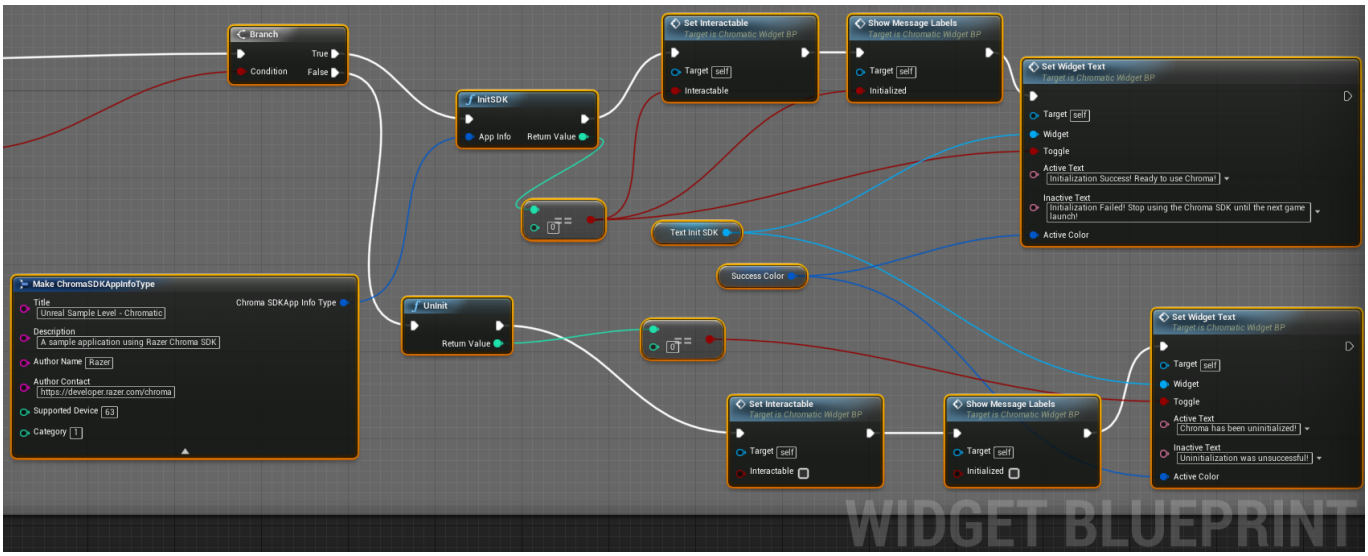
Chromatic Level

- The following APIs are demonstrated in the `Chroma_Sample\Content\Levels\Chromatic_Level.umap` sample level and `Chroma_Sample\Content\UI\ChromaticWidget_BP.uasset` widget blueprint.



Initialize SDK

Initialize the Chroma SDK in order to utilize the API. The `InitSDK` method takes an `AppInfo` parameter which defines the application or game details that will appear in the `Chroma App` within the `Chroma Apps` tab. The expected return result should be zero for success which indicates the API is ready for use. If a non-zero result is returned, the Chroma implementation should be disabled until the next time the application or game is launched. Reasons for failure are likely to be the user does not have the `Synapse` or the `Chroma App` installed. After successfully initializing the Chroma SDK, wait approximately 100 ms before playing Chroma animations.




```
FChromaSDKAppInfoType appInfo;
appInfo.Title = "Unreal Sample Scene - Chromatic";
appInfo.Description = "A sample application using Razer Chroma SDK";
appInfo.Author_Name = "Razer";
appInfo.Author_Contact = "https://developer.razer.com/chroma";

//appInfo.SupportedDevice =
//    0x01 | // Keyboards
//    0x02 | // Mice
//    0x04 | // Headset
//    0x08 | // Mousepads
//    0x10 | // Keypads
//    0x20   // ChromaLink devices
appInfo.SupportedDevice = (0x01 | 0x02 | 0x04 | 0x08 | 0x10 | 0x20);
//    0x01 | // Utility. (To specifiy this is an utility application)
//    0x02   // Game. (To specifiy this is a game);
appInfo.Category = 1;

int result = UChromaSDKPluginBPLibrary::ChromaSDKInitSDK(appInfo);
if (result == 0)
{
    // Init Success! Ready to use the Chroma SDK!
}
else
{
    // Init Failed! Stop using the Chroma SDK until the next game launch!";
}
```

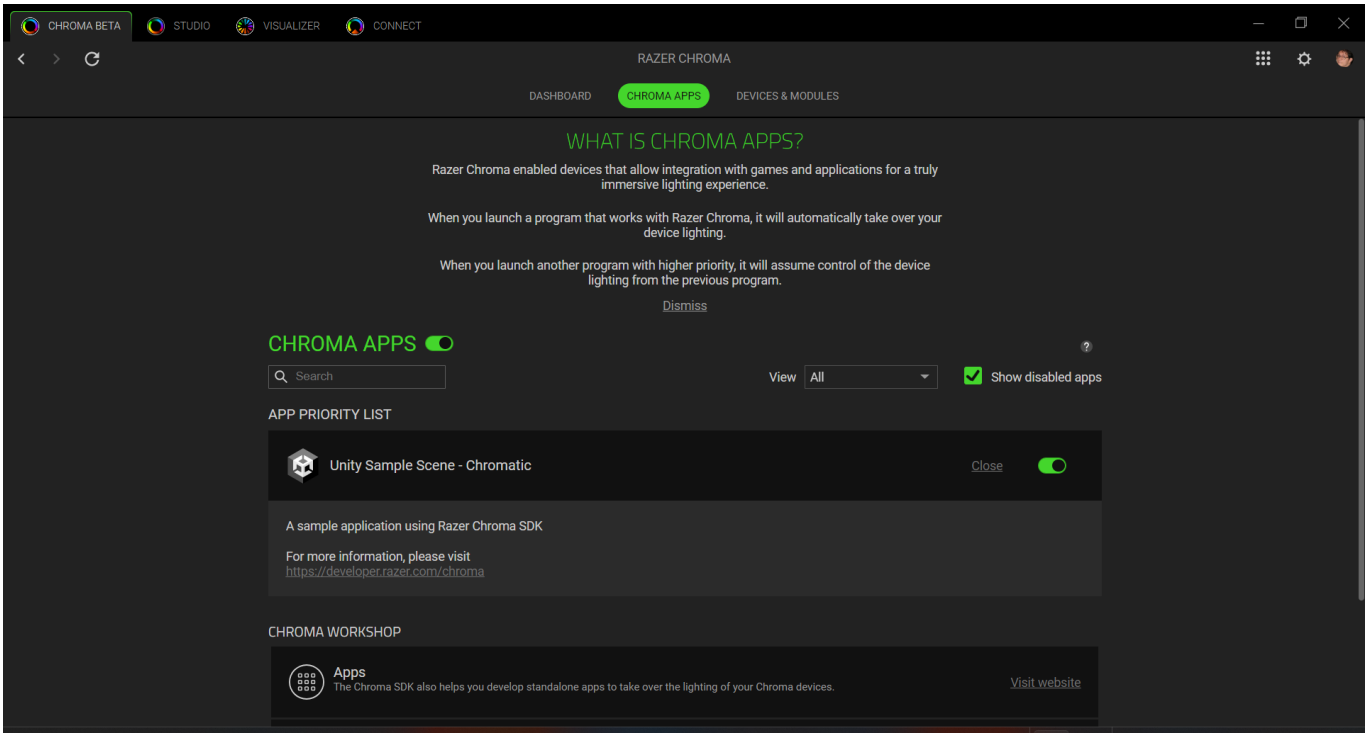
Applications should uninitialize the Chroma SDK with Uninit() for a clean exit. Uninitialization is only needed if the Chroma SDK was successfully initialized.

```
int result = UChromaSDKPluginBPLibrary::ChromaSDKUnInit();
if (result == 0)
{
    // Chroma has been uninitialized!
```

```
}
else
{
    // Uninitialization was unsuccessful!
}
```

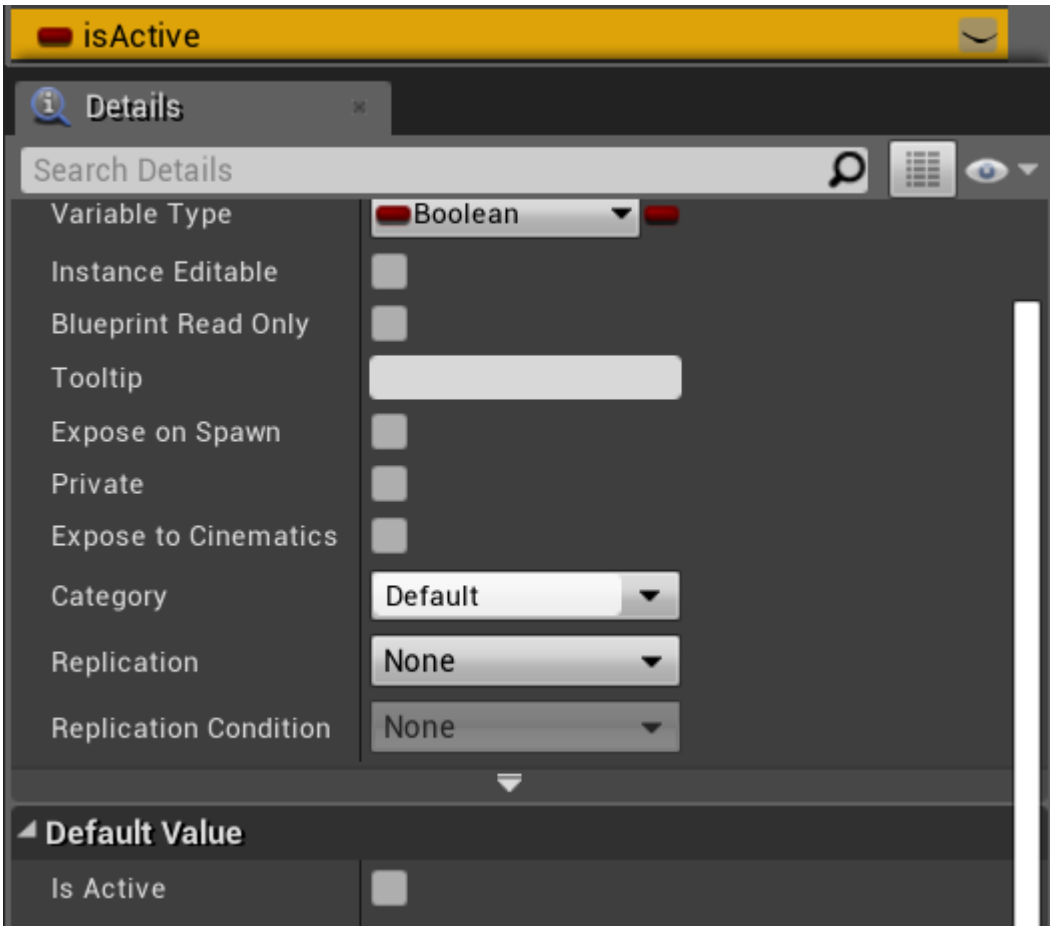
Is Active

Many applications and games can use the Chroma SDK at the same time, yet only one can have the Chroma focus. The `APP PRIORITY LIST` defines the priority order and the highest on the list receives the Chroma focus when more than one are actively using the Chroma SDK. Users can adjust the priority order by dragging and dropping or toggling the app completely off.

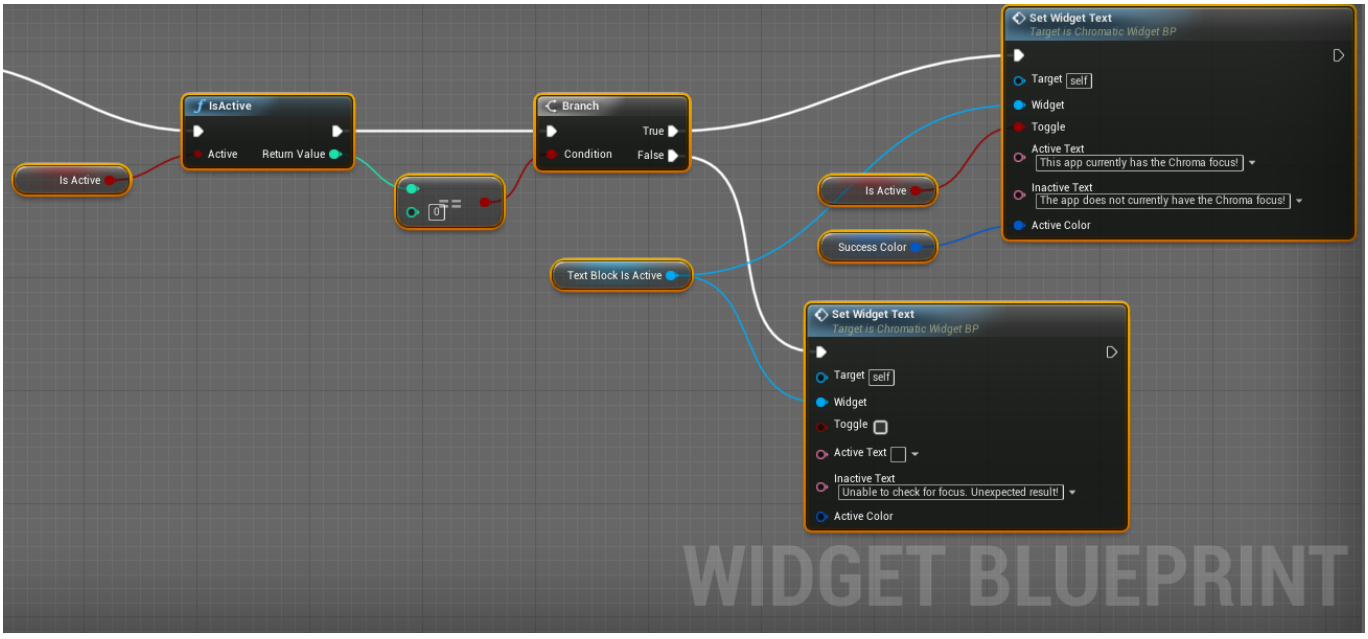


The IsActive() method allows an application or game to check if it has Chroma focus at a time. This allows the title to free up overhead when Chroma is not in use. If a title uses this to check for focus, the state should be periodically checked to turn Chroma back on when focus is returned. When active returns false, the title can stop playing Chroma animations, disable idle animations, and inactivate dynamic Chroma to free up some overhead. Keep in mind that some apps use Chroma notifications so they will only briefly take Chroma focus and then return it typically over a 5 second period.

Blueprints should define a bool variable to pass by reference.



Check the state of the variable when the result is successful.



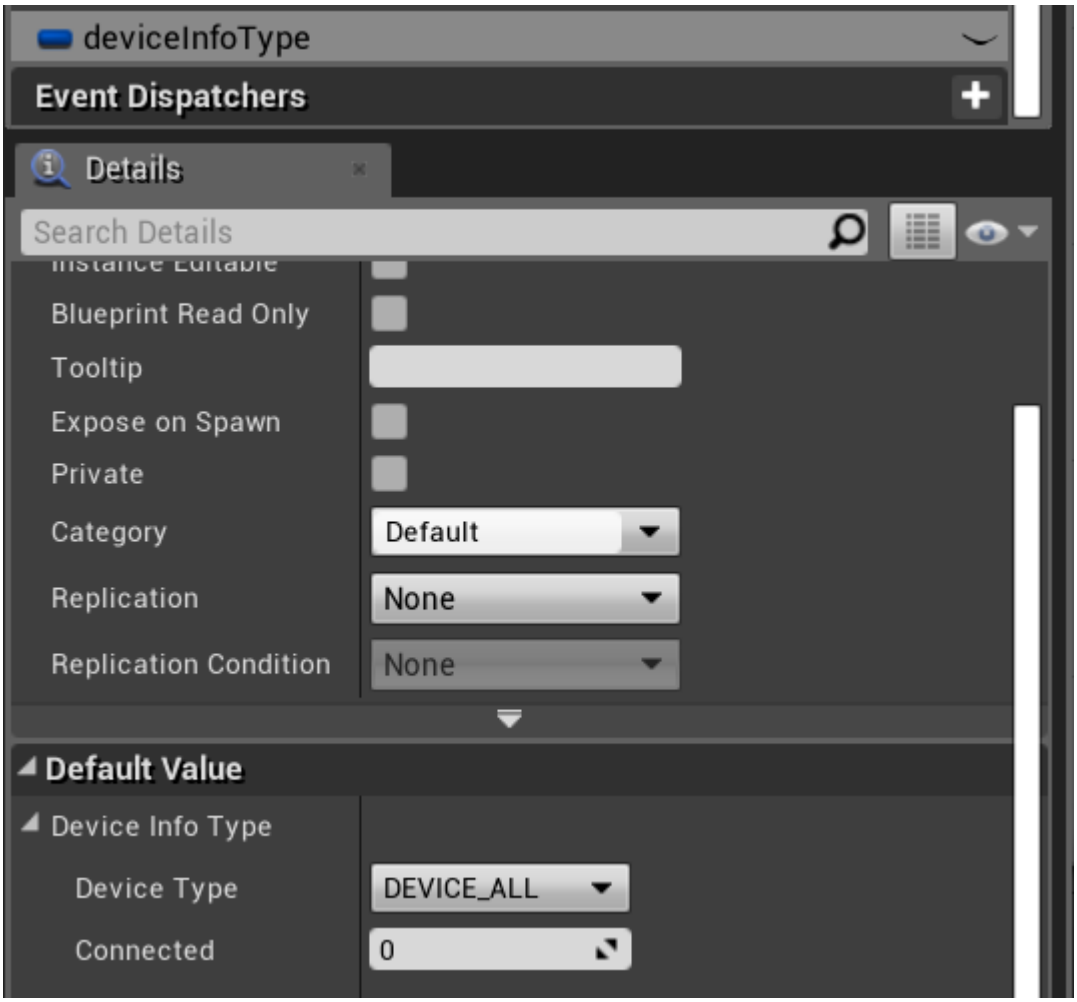

```
bool isActive;
int result = UChromaSDKPluginBPLibrary::IsActive(isActive);
if (result == 0)
{
    if (isActive)
    {
        // The game currently has the Chroma focus!
    }
```

```
        else
        {
            // The game does not currently have the Chroma focus!"
        }
    }
    else
    {
        // Unable to check for Chroma focus. Unexpected result!
    }
```

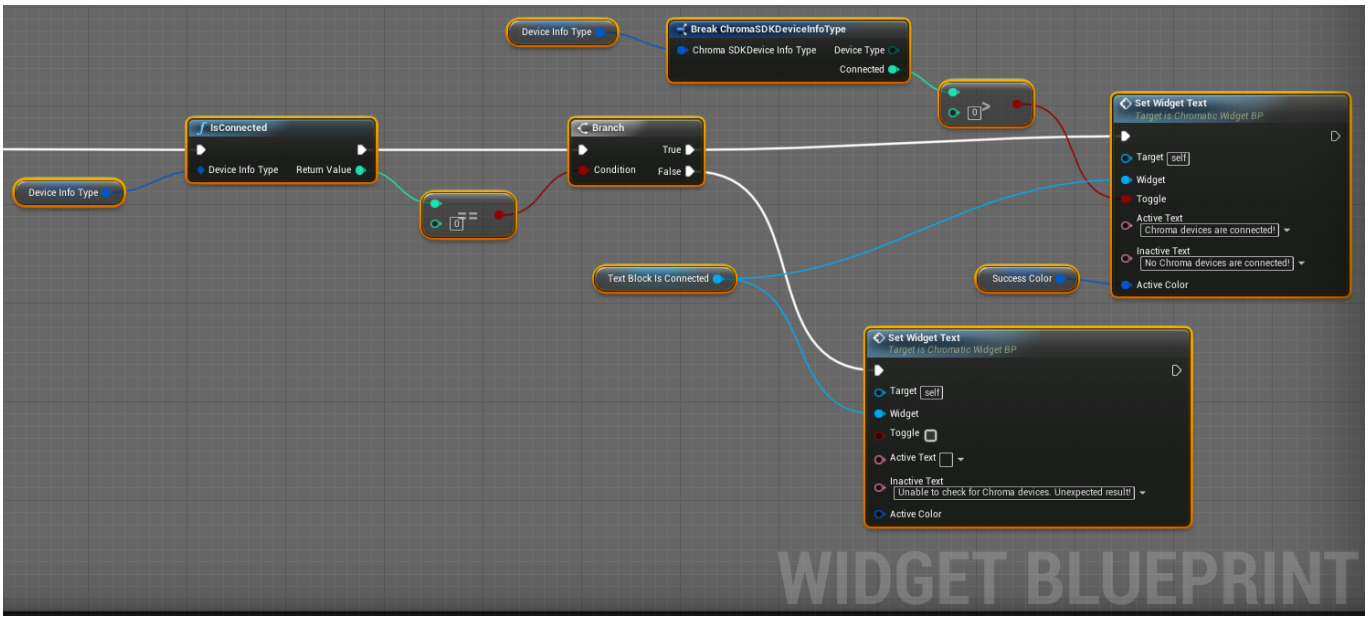
Is Connected

To further reduce overhead, a title can check if supported devices are connected before showing Chroma effects. The IsConnected() method can indicate if supported devices are in use to help determine if Chroma should be active. Games often will include a menu settings option to toggle Chroma RGB support, with being on by default as an additional way that users can minimize overhead.

Blueprints should define a FChromaSDKDeviceInfoType variable to pass by reference.



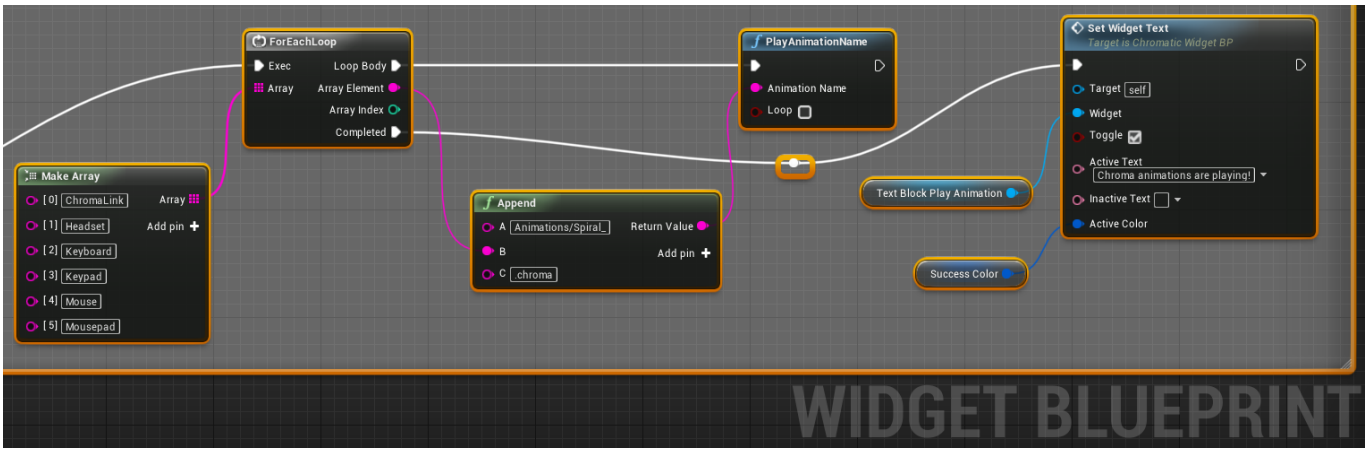
Check the state of the variable when the result is successful.




```
FChromaSDKDeviceInfoType deviceInfoType;
deviceInfoType.DeviceType = EChromaSDKCoreDeviceTypeEnum::DEVICE_ALL;
int result = UChromaSDKPluginBPLibrary::IsConnected(deviceInfoType);
if (result == 0)
{
    if (deviceInfoType.Connected > 0)
    {
        // Chroma devices are connected!
    }
    else
    {
        // "No Chroma devices are connected!";
    }
}
else
{
    // "Unable to check for Chroma devices. Unexpected result!";
}
```

Play Chroma Animation

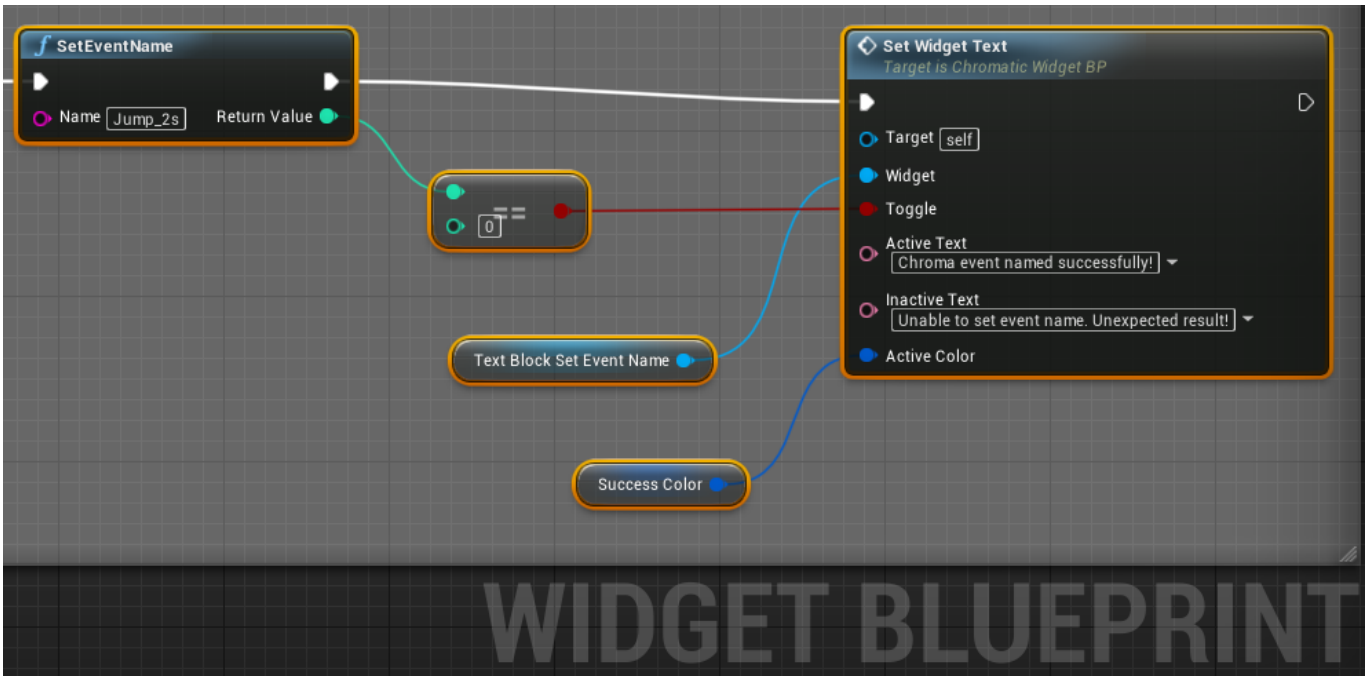
The Chroma SDK supports playing premade Chroma animations which are placed in the `Content` folder or subfolders within. Chroma animations can be created in the web authoring tools, or dynamically created and modified using the API. Call PlayAnimation() to play Chroma animations with or without looping. Animations have a device category, and playing an animation will stop an existing animation from playing before playing the new animation for the given device category. The animation name is file path of the Chroma animation relative to the `Content` folder.




```
bool loop = false;
TArray<FString> deviceCategories =
{
    "ChromaLink",
    "Headset",
    "Keyboard",
    "Keypad",
    "Mouse",
    "Mousepad",
};
for (int i = 0; i < deviceCategories.Num(); ++i)
{
    FString animationName = "Animations/Spiral_" + deviceCategories[i] +
".chroma";
    UChromaSDKPluginBPLibrary::PlayAnimationName(animationName, loop);
}
```

Set Event Name

Chroma events can be named to add supplemental technology to your lighting experience. By naming game events and game triggers, the event name can be used as a lookup to play things like haptics effects. `Jump_2s` could be used when playing a Chroma animation of a jump effect that lasts for 2 seconds. Using "Jump_2s" a corresponding haptic effect with similar duration can be added with the Chroma effect to enhance emersion for the title. No other APIs are required to add haptics effects other than to invoke SetEventtName(). To stop haptics playback use SetEventName() with an empty string. A Chroma animation does not need to be playing in order to trigger haptics manually with SetEventName().

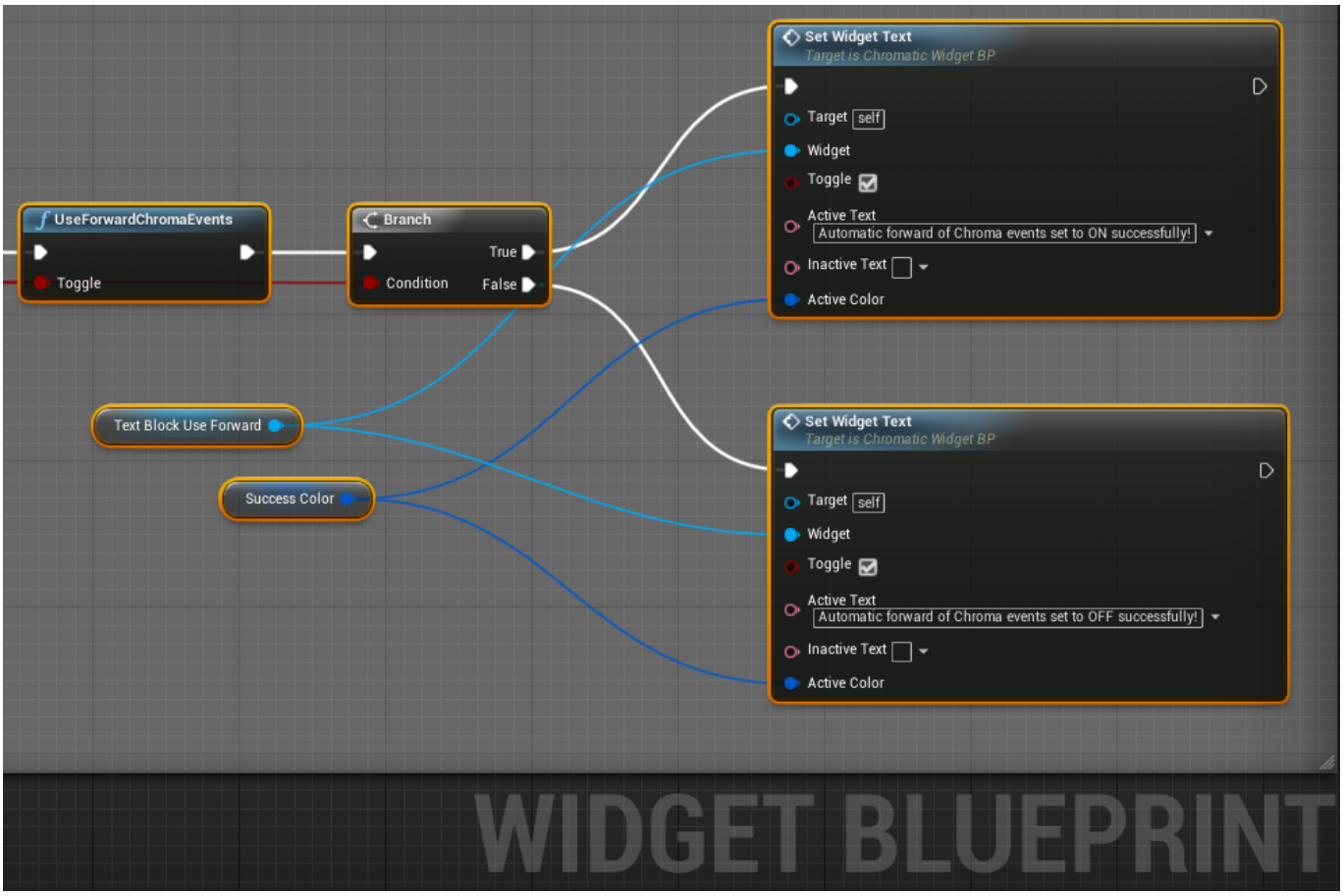



```
int result = UChromaSDKPluginBPLibrary::SetEventName(L"Jump_2s");
if (result == 0)
{
    // Chroma event named successfully!"
}
else
{
    // Unable to set event name. Unexpected result!"
}

// Stop haptic playback
result = UChromaSDKPluginBPLibrary::SetEventName(L"");
if (result == RZRESULT_SUCCESS)
{
    // Haptics stopped successfully!"
}
else
{
    // Unable to stop haptics. Unexpected result!"
}
```

Use Forward Chroma Events

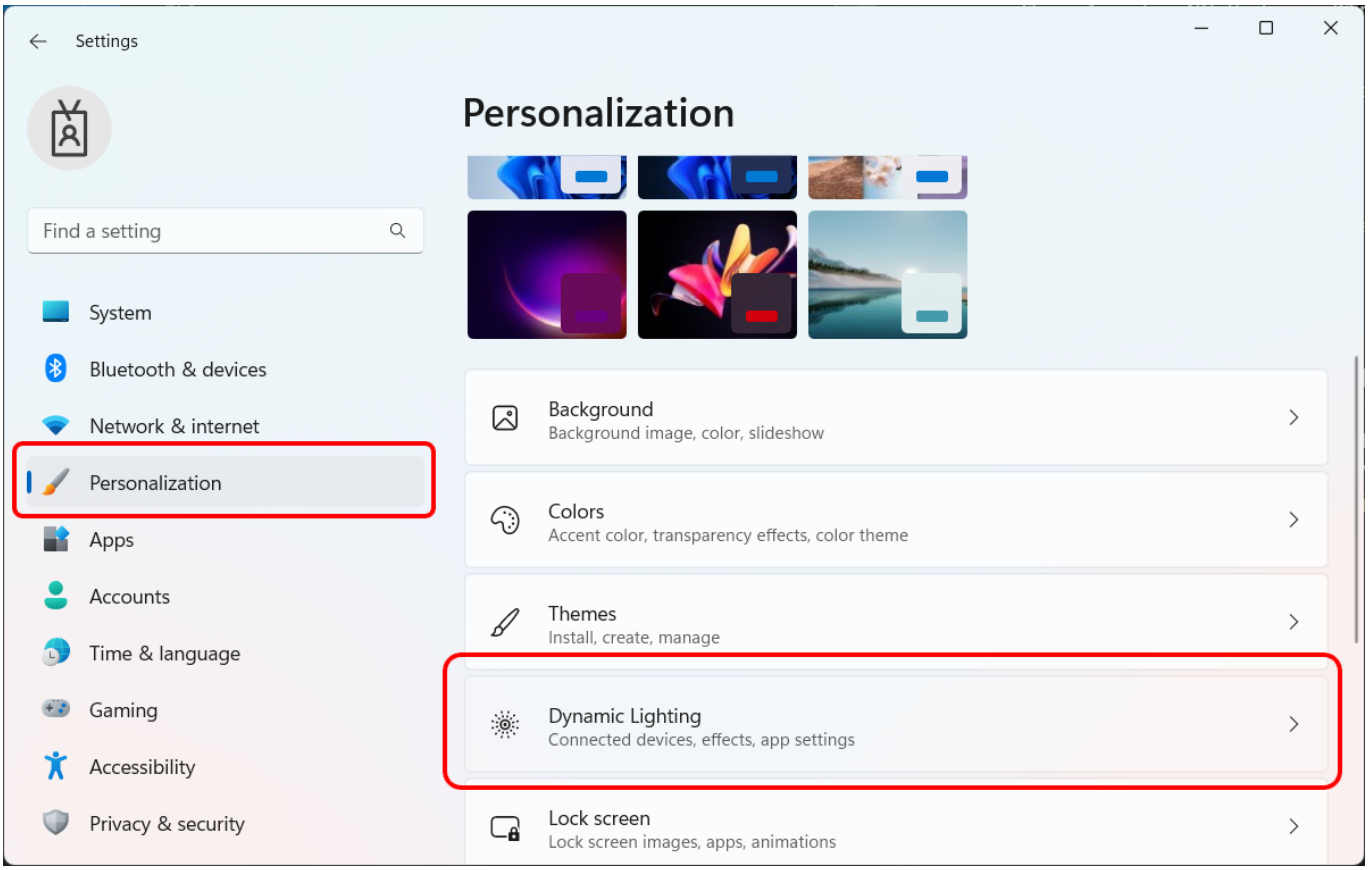
By default when PlayAnimation is called, the animation name is automatically sent to SetEffectName(). In order to disable the default behaviour set the toggle to false. PlayAnimation() as shown above is called for each device category. It will be more efficent to use SetEventName() once for the Chroma animation set. Manual mode gives the title explicit control over when SetEventName() is called.



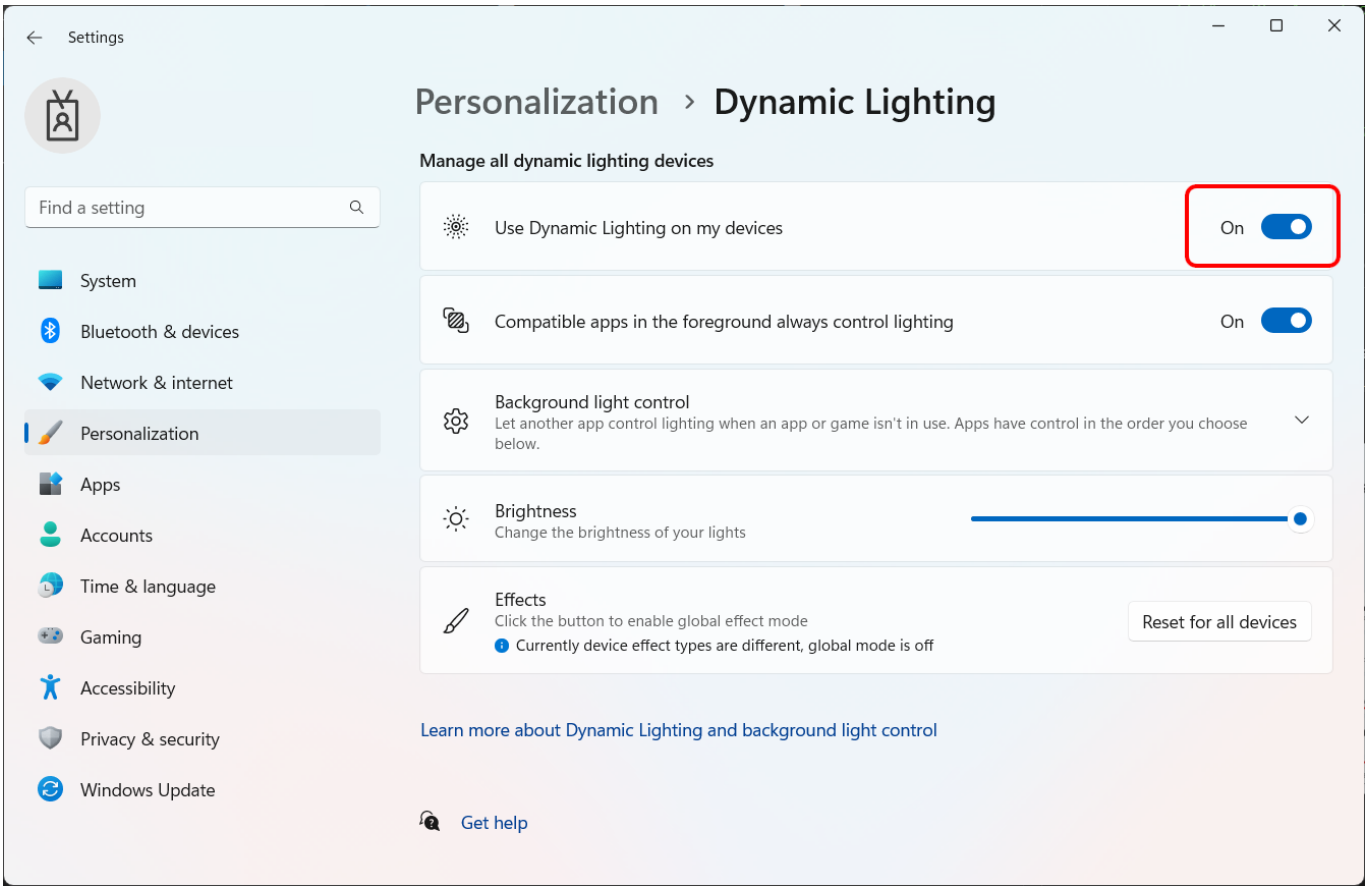

```
bool toggle = false; // manual mode
UChromaSDKPluginBPLibrary::UseForwardChromaEvents(toggle);
if (toggle)
{
    // When PlayAnimation is used, the name is sent to SetEventName().
}
else
{
    // The PlayAnimation name is not forwarded.
}
```

Microsoft Dynamic Lighting

Windows 11 launched Microsoft Dynamic Lighting which is built-in to the Windows Settings Personalization on Windows. Microsoft DL became generally available in `Windows 11 22H2`. See the list of supported devices.



For HID compatible devices, with `Dynamic Lighting` set to `ON` and `Chroma App` set as the ambient controller, Chroma effects will display on DL compatible hardware. No extra coding is required to add this compatibility. `Chroma App` handles Chroma compatibility with DL and it is completely automatic.



See Also

Docs:

- Chroma Animation Guide - Visual examples of the Chroma Animation API methods

Plugins:

- CChromaEditor - C++ library for playing and editing Chroma animations

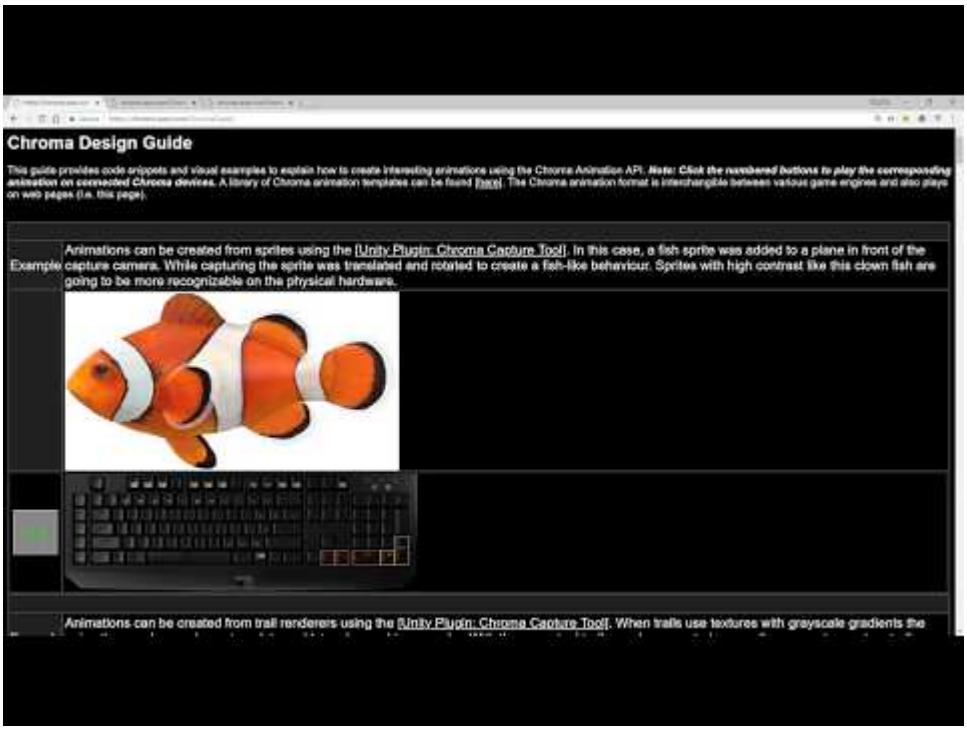
Overview

`Chroma_Sample` provides a runtime module for using the `ChromaSDK`. The runtime module provides a blueprint library and C++ methods for playing Chroma animations. See the Chroma Guide for details on how to make visually interesting Chroma animations using the plugin API.

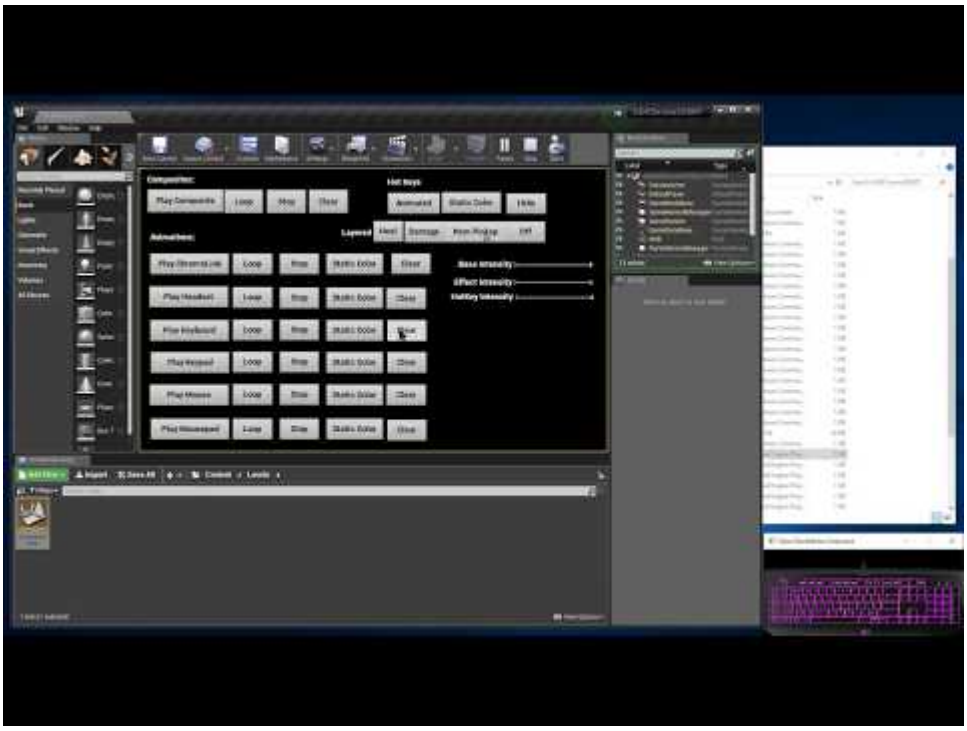
Tutorials

Videos

Chroma Design Guide

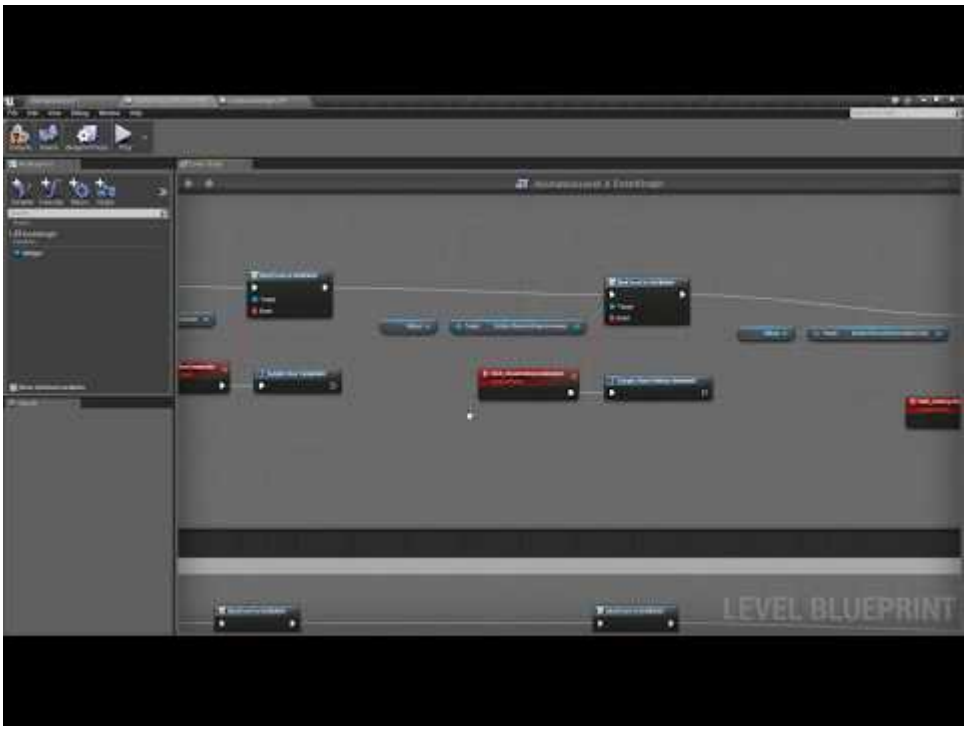


Chroma_Sample Plugin Setup for your specific version of UE.



Chroma_Sample Overview

Simplify UI Blueprints



Supported versions

This project is checked in under `UE 4.21`. To use a later version of Unreal, open the Chroma_Sample/Chroma_Sample.uproject project file in a text editor and change the `EngineAssociation` to the target version.

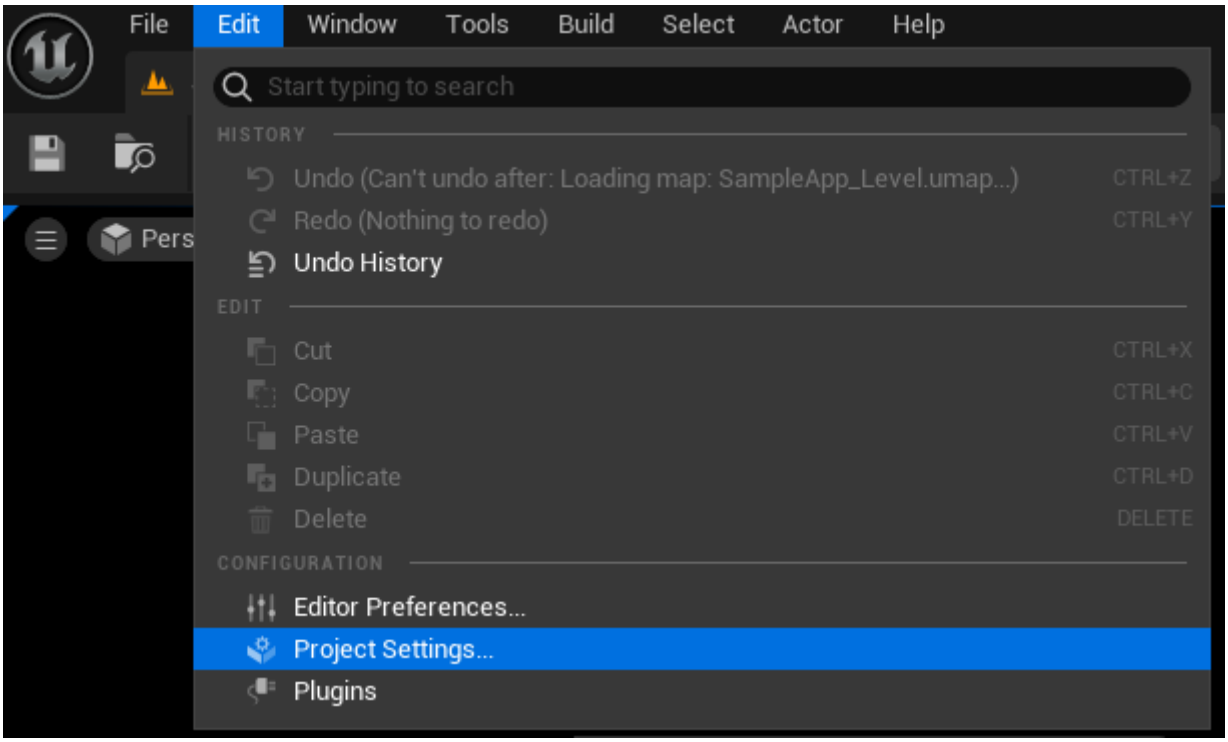
```
"EngineAssociation": "4.21",
```

To update the plugin version, open Chroma_Sample/Plugins/ChromaSDKPlugin/ChromaSDKPlugin.uplugin in a text editor and set the target version.

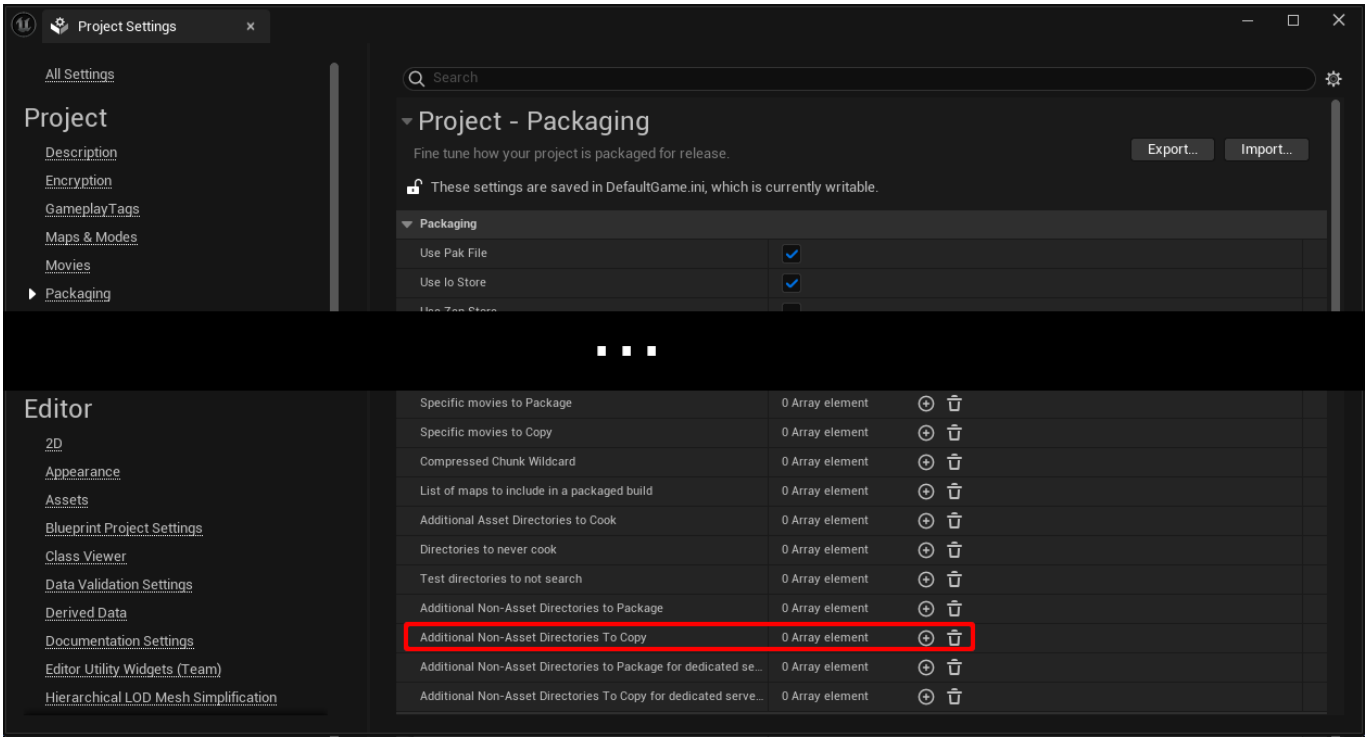
```
"EngineVersion": "4.21.0",
```

Packaging

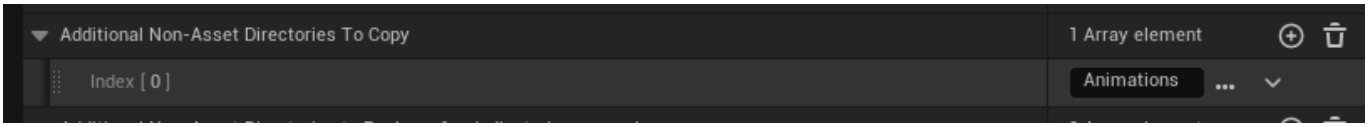
Edit the project settings in order to include Chroma animation files within the content folder.



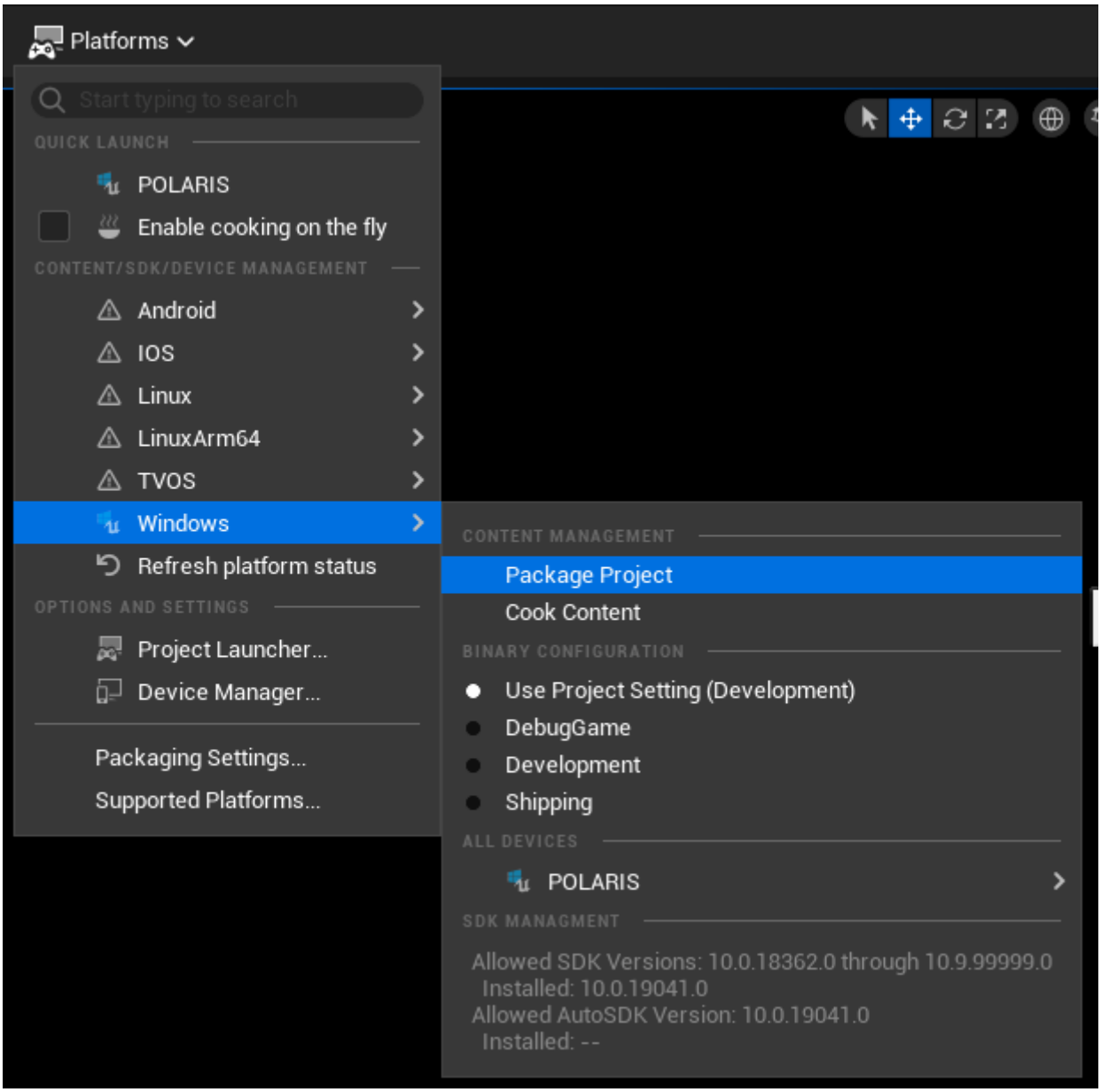
Find the Project - Packaging section and scroll down to `Additional Non-Asset Directories to Copy`.



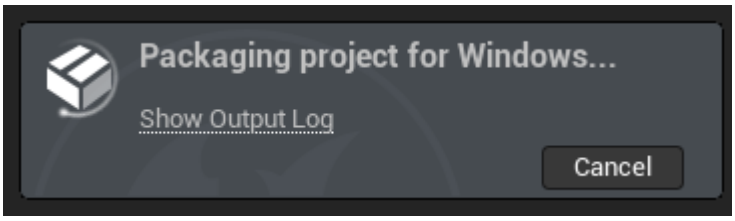
Add an entry to the location within your Content subfolder where you placed the Chroma animation files.



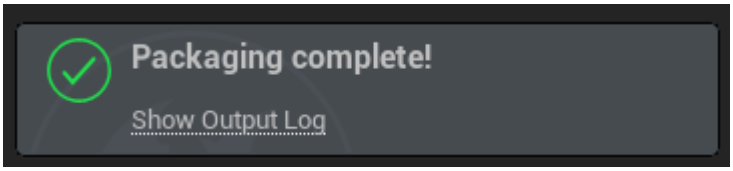
The packaging UI is different depending on your version of `Unreal Editor`. In 5.4, the `Platforms` dropdown is available on the main toolbar. Select `Windows->Package Project`.



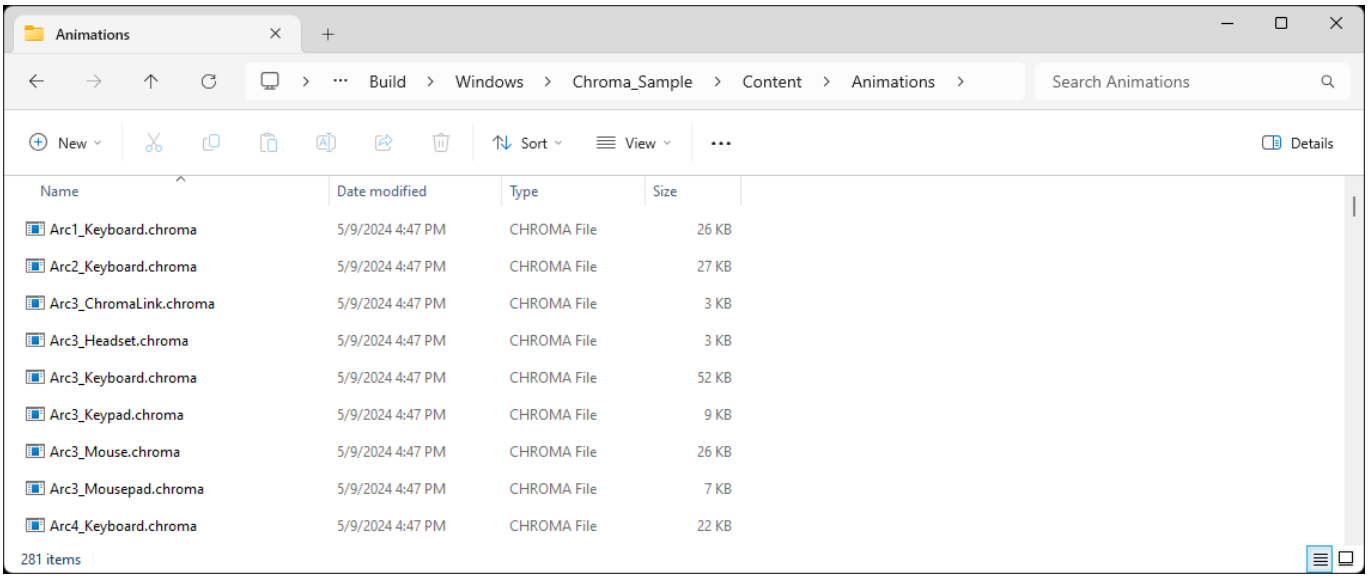
`Package project for Windows...` may appear for a few minutes.



And then finally complete.



After completing packaging for the Windows platforms the Chroma animation content will be included in the build automatically.



Security

The C++ Chroma Editor Library loads the core Razer DLL `RzChromatic.dll` and the Razer stream library `RzChromaStreamPlugin.dll`. To avoid a 3rd party injecting malicious code, the C++ Chroma Editor Library checks for a valid signature on the Razer libraries. The DLL issuer is validated to be `Razer USA Ltd.` Init and InitSDK will return `RZRESULT_DLL_INVALID_SIGNATURE` if the signature check fails.

The sample apps use the `CHECK_CHROMA_LIBRARY_SIGNATURE` preprocessor definition to enable signature checking on the Chroma Editor Library. Signature checking can be used on the Razer libraries downloaded from Github releases.

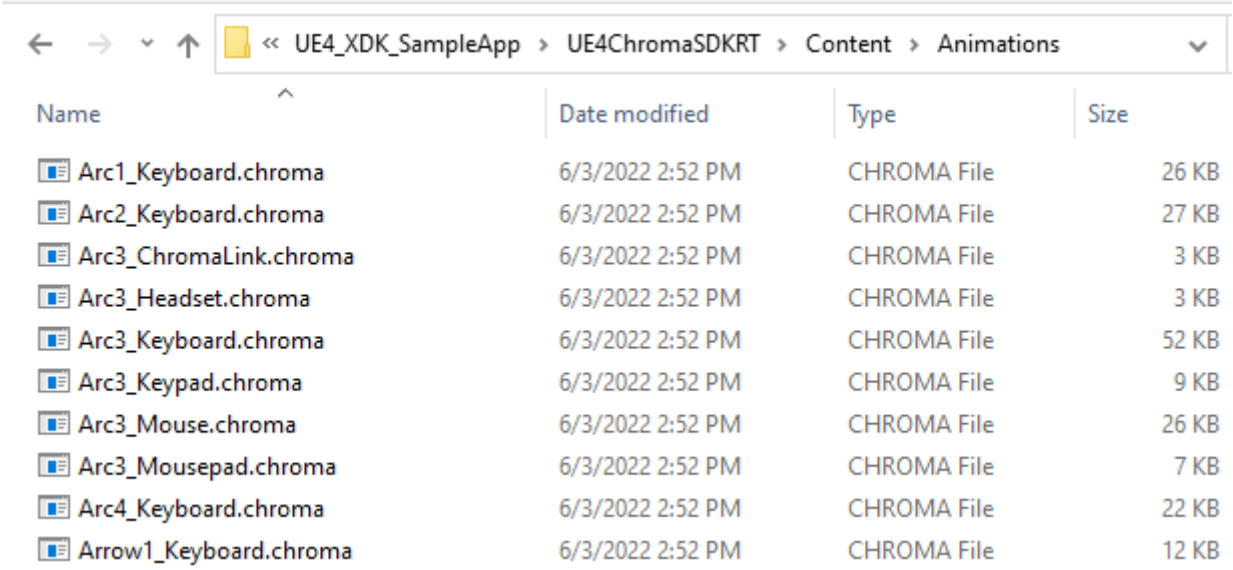
```
#ifdef CHECK_CHROMA_LIBRARY_SIGNATURE
    // verify the library has a valid signature
    _sInvalidSignature = !VerifyLibrarySignature::VerifyModule(path);
#endif
```

Chroma Editor Library

The Chroma Editor Library is a helper library for Chroma animation playback and realtime manipulation of Chroma animations.

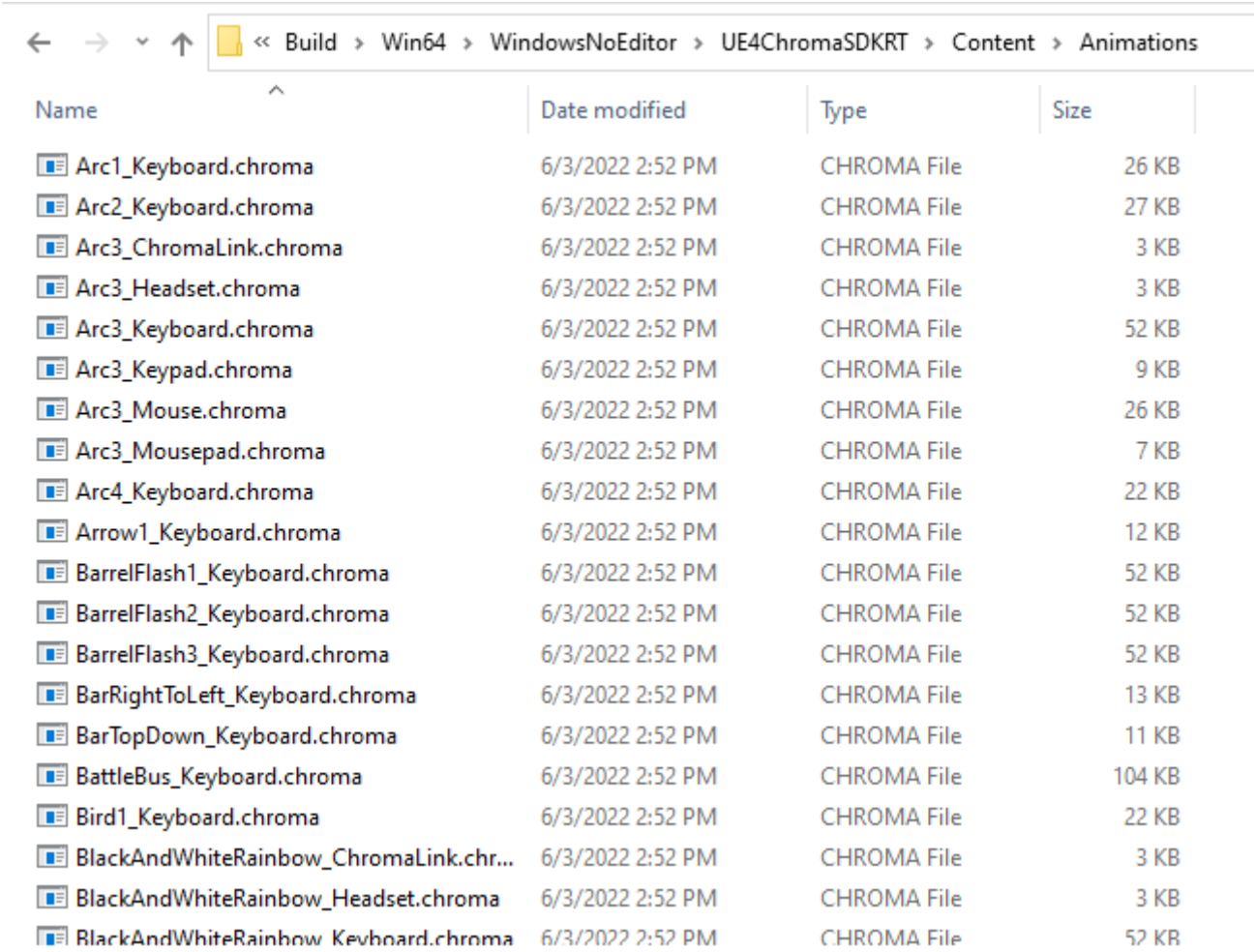
In the UE Editor, Chroma animations files are placed within the project content folder. Animation paths used in the Chroma API are relative to the content folder.

```
Chroma_Sample\Content
```



In a standalone PC or Cloud build, Chroma animation files may need to be copied to within the build content folder.

```
WindowsNoEditor\Chroma_Sample\Content
```



The latest versions of the `Chroma Editor Library` can be found in Releases for `Windows-PC` and `Windows-Cloud`.

The plugin build file Chroma_Sample\Plugins\ChromaSDKPlugin\Source\ChromaSDKPlugin\ChromaSDKPlugin.Build.cs has a preprocessor definition to check the signature of the `Chroma Editor Library`. This a security feature and Chroma libraries won't be loaded that fail to pass the signature validation when this flag is enabled.

```
PrivateDefinitions.Add("CHECK_CHROMA_LIBRARY_SIGNATURE=1");
PublicDefinitions.Add("CHECK_CHROMA_LIBRARY_SIGNATURE=1");
```

Video: **UE Chroma Animation Sample App - Streaming on Windows PC and Cloud**

Windows PC

For `Windows PC` builds the `RzChromaSDK.dll` and `RzChromaStreamPlugin.dll` are not packaged with the build. These libraries are automatically updated and managed by Synapse and the Chroma Connect module. Avoid including these files in your build folder for `Windows PC` builds.

Within the `Chroma Plugin` the `Chroma Editor Library` files (`CChromaEditorLibrary.dll` and `CChromaEditorLibrary64.dll`) are part of the plugin's binary folders on Windows.

32-bit libraries

```
Project Folder\Plugins\ChromaSDKPlugin\Binaries\Win32\CChromaEditorLibrary.dll
Build
Folder\WindowsNoEditor\Chroma_Sample\Plugins\ChromaSDKPlugin\Binaries\Win32\CChrom
aEditorLibrary.dll
```

64-bit libraries

```
Project Folder\Plugins\ChromaSDKPlugin\Binaries\Win64\CChromaEditorLibrary64.dll
Build
Folder\WindowsNoEditor\Chroma_Sample\Plugins\ChromaSDKPlugin\Binaries\Win64\CChrom
aEditorLibrary64.dll
```

Windows Cloud

`Windows Cloud` builds run on cloud platforms using `Windows` such as `Amazon Luna`, `Microsoft Game Pass`, and `NVidia GeForce Now`. Game instances run in the cloud without direct access to Chroma hardware. Chroma effects stream across the Internet to reach your local machine and connected hardware. No extra code is required to add Cloud support. In the case with `NVidia GeForce Now`, the cloud runs the same Epic Games and Steam builds as the desktop version and support Chroma streaming. Viewers can watch the cloud stream via the Razer Stream Portal.

Plugin Structure

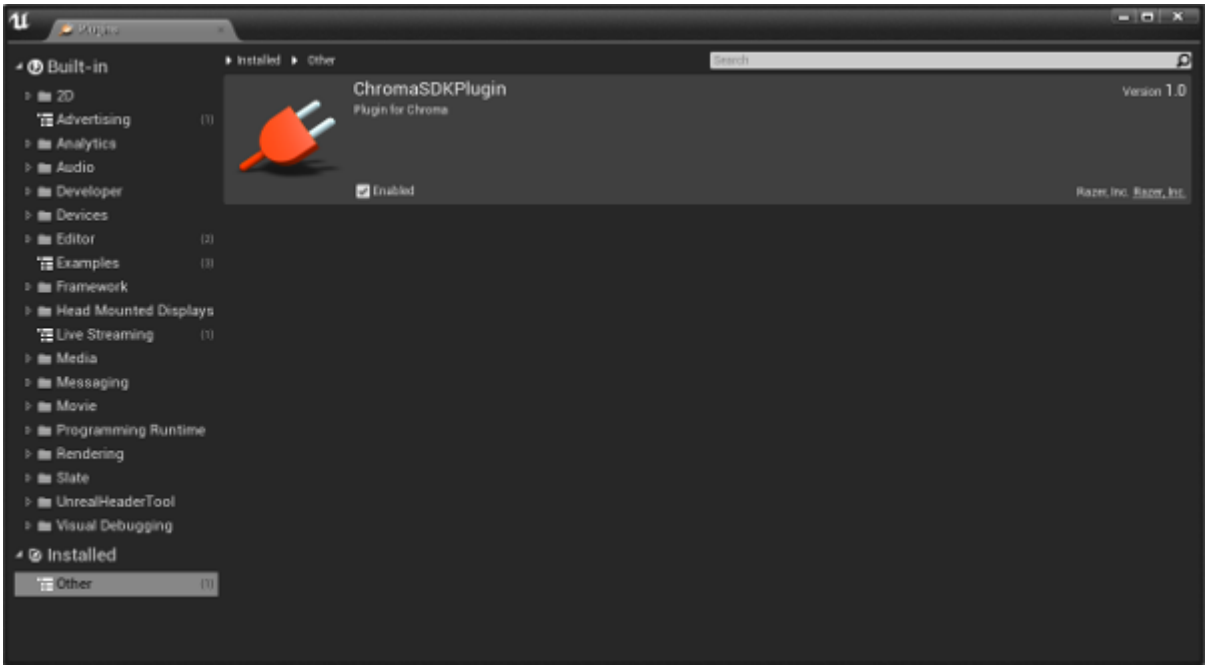
Plugin Definition: `Plugins/ChromaSDKPlugin/ChromaSDKPlugin.uplugin`

Plugin Source: `Chroma_Sample/Plugins/ChromaSDKPlugin/Source/ChromaSDKPlugin/`

Headers: `Chroma_Sample/Plugins/ChromaSDKPlugin/Source/ChromaSDKPlugin/Public/`

Implementation: `Chroma_Sample/Plugins/ChromaSDKPlugin/Source/ChromaSDKPlugin/Private/`

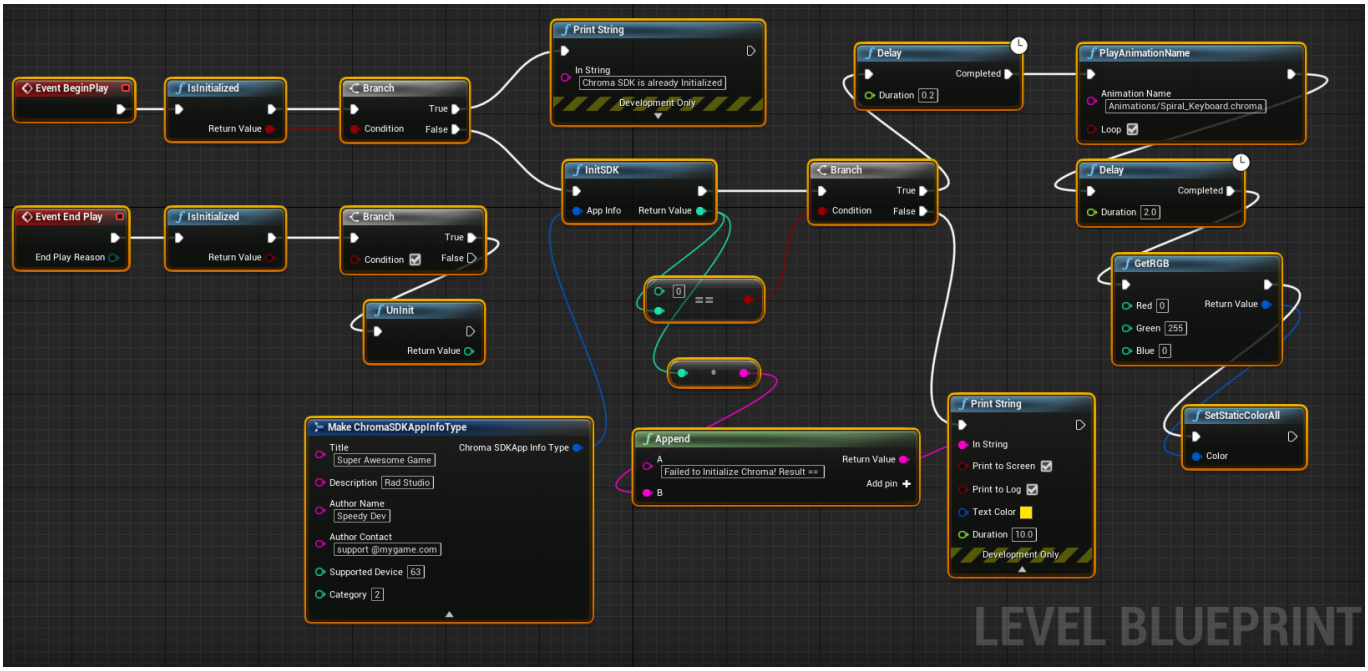
Plugin appears in Window->Plugins



Sample Blueprint Init / Uninit Setup

`Event BeginPlay` invokes `InitSDK` passing the `AppInfo` that provides the information that displays within `Synapse->Connect->Apps`. `InitSDK` returns `0` upon success after a 100ms delay the Chroma API is ready to use. If `InitSDK` returns nonzero, avoid further calls to the Chroma API. After success, make a call to

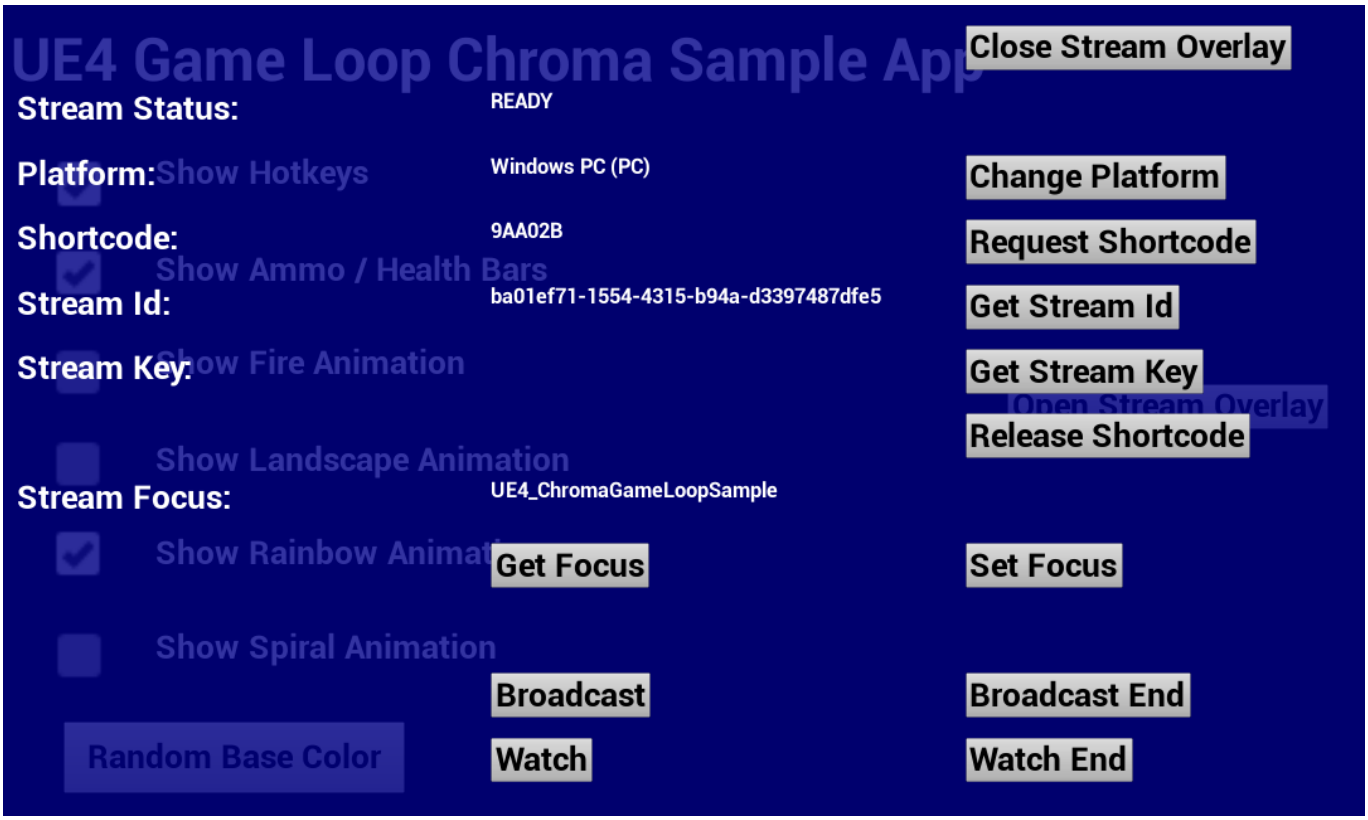
`SupportsStreaming` and save the result. If `SupportsStreaming` returns true, the streaming API can be used for broadcasting Chroma.



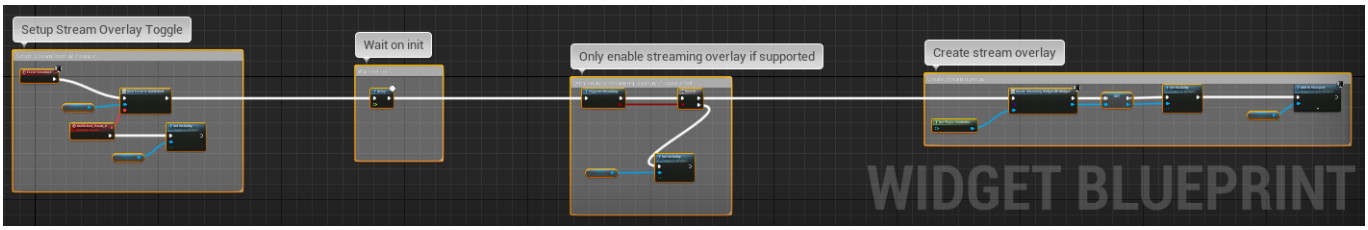
Samples

The project has a few sample levels.

Samples share the same `Stream Overlay` logic defined in the Chroma_Sample/Content/UI/StreamingWidget_BP.uasset Widget Blueprint.

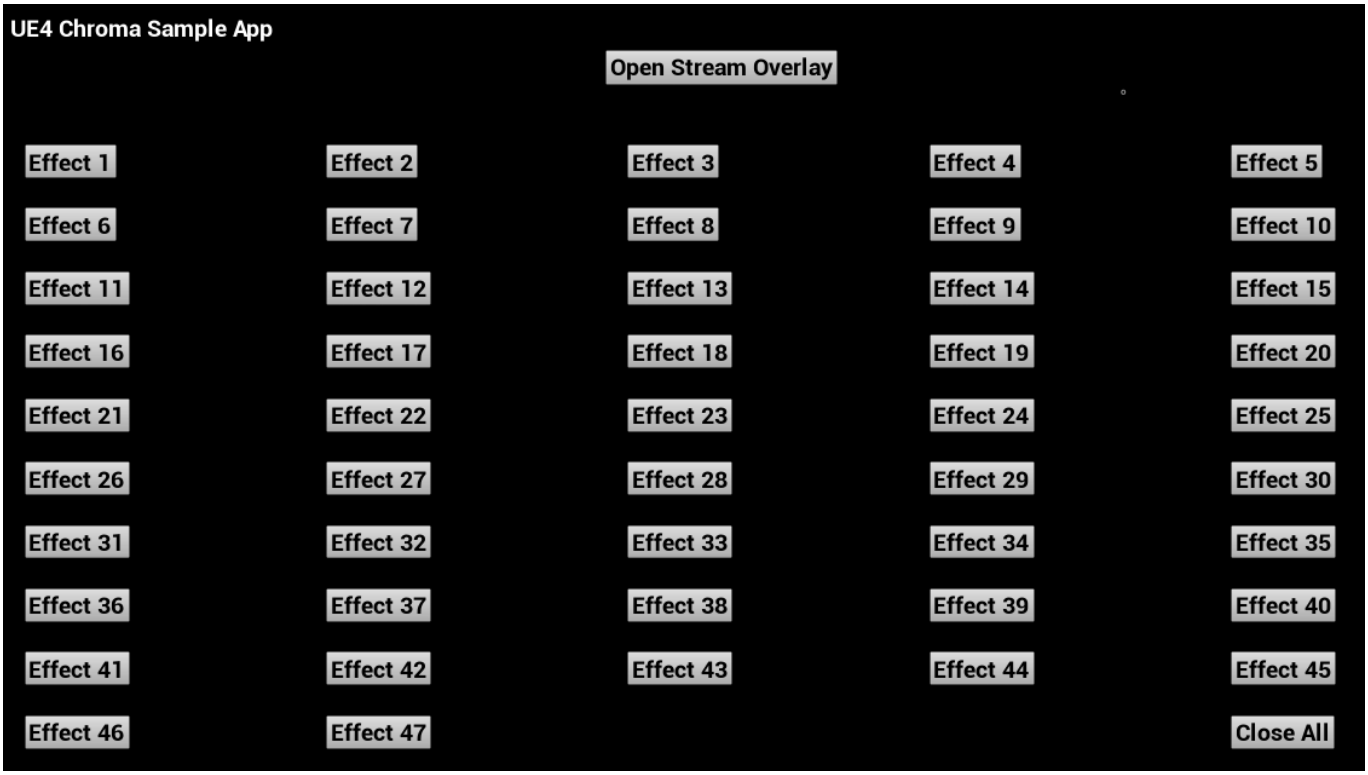


Sample UI event construction checks if streaming is supported before showing the button that displays the sample stream overlay.



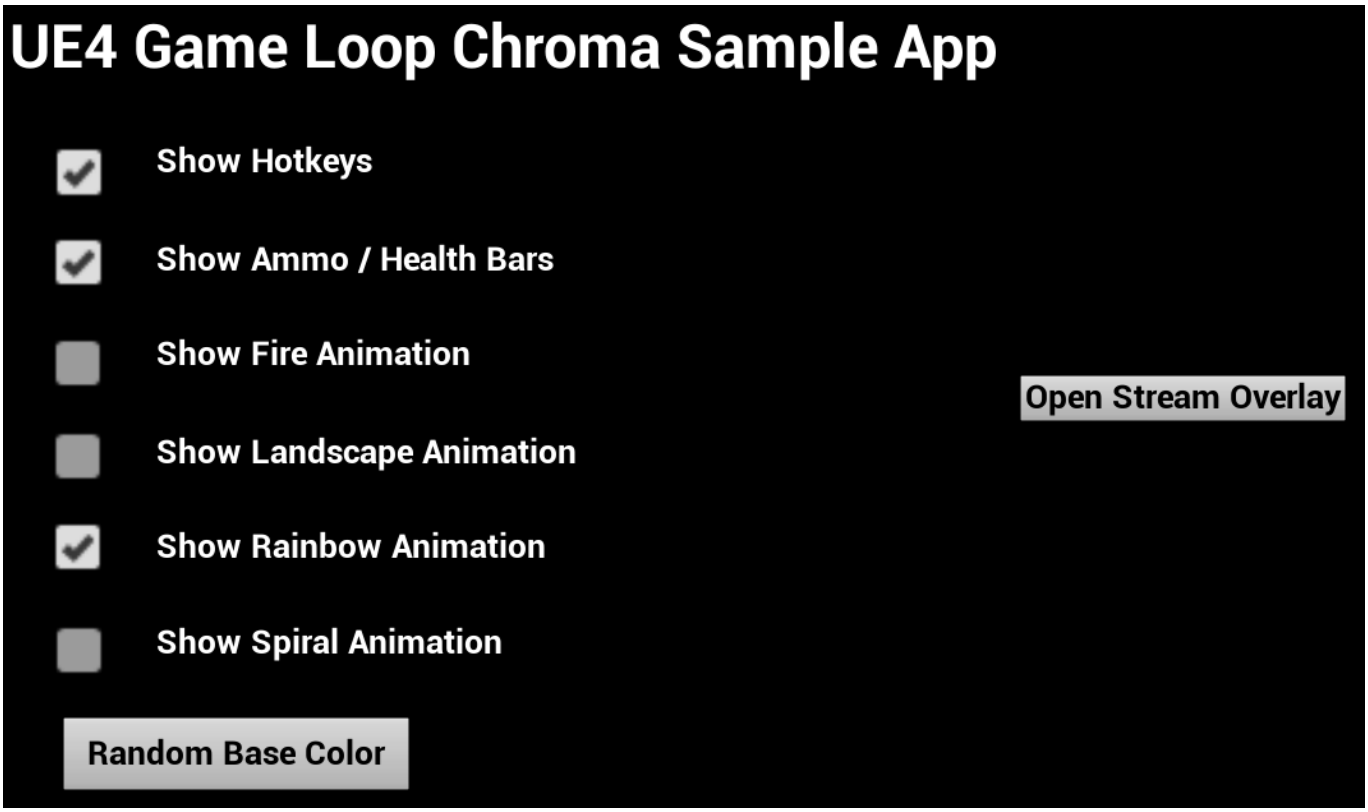
UE Chroma Sample App

The Chroma_Sample/Content/Levels/SampleApp_Level.umap level shows the sample animations from the Chroma Animation Guide. The level blueprint uses BP functions defined in the Chroma_Sample/Source/Chroma_Sample/SampleAppChromaBP.h header and implemented in the Chroma_Sample/Source/Chroma_Sample/SampleAppChromaBP.cpp source.



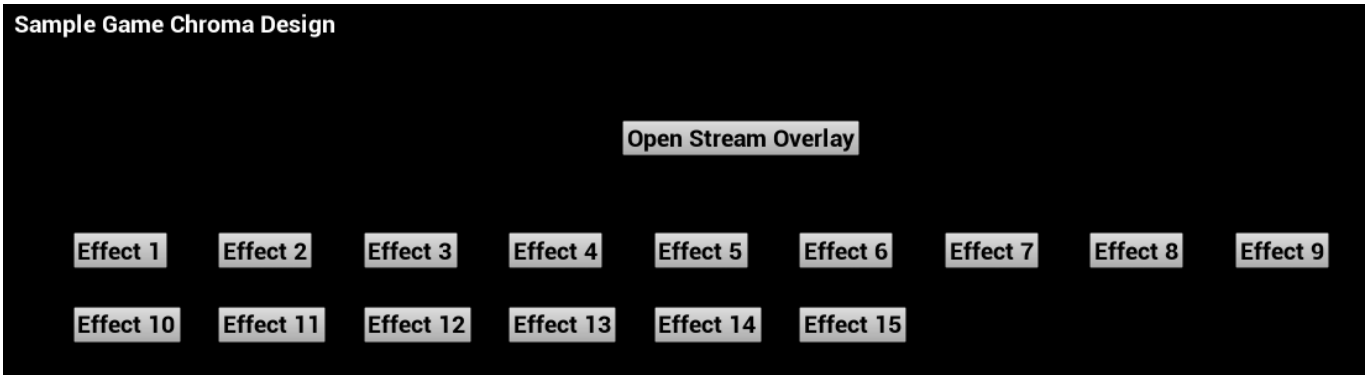
UE Game Loop Chroma Sample App

The Chroma_Sample/Content/Levels/SampleGameLoopLevel.umap level shows how to dynamically set color effects directly through the API and while also playing several animations at the same time using various blending operations. This sample shows how to do Chroma effects without using premade Chroma animations. Chroma animations can be used as source color information when doing dynamic blending. The level blueprint uses BP functions defined in the Chroma_Sample/Source/Chroma_Sample/SampleGameLoopChromaBP.h header and implemented in the Chroma_Sample/Source/Chroma_Sample/SampleGameLoopChromaBP.cpp source.



UE Sample Game Chroma Design

The Chroma_Sample/Content/Levels/SampleGameLevel.umap level is a template intended to work with the automated Chroma Design Converter for quickly porting sample effects from HTML5 to Unity. The level blueprint uses BP functions defined in the Chroma_Sample/Source/Chroma_Sample/SampleGameChromaBP.h header and implemented in the Chroma_Sample/Source/Chroma_Sample/SampleGameChromaBP.cpp source. Chroma Design samples are commonly created with 15 sample effects which is why the template has that many buttons to play the sample effects from the ported code. The Chroma Design Converter is not limited to just 15 sample effects and can generate more effect code from the input HTML5 script.



Unreal Compatibility

- Note: Enum syntax - Enums are namespaced and types use the `EChromaSDKKeyboardKey::Type` syntax to avoid collisions.
- Note: No const enum types or passing enums by reference in function parameters - Avoid use of const enum types because that seems to crash in UE 4.5.

Full API

`Chroma_Sample` is a Blueprint API library with methods that expose the CChromaEditor library.

- Take a look at the code from SampleAppChromaBP.cpp. These sample effects show how the blueprint library was used to create the `Chroma` effects that correspond to the `Guide`.
- AddNonZeroAllKeys
- AddNonZeroAllKeysAllFrames
- AddNonZeroAllKeysAllFramesName
- AddNonZeroAllKeysAllFramesOffset
- AddNonZeroAllKeysAllFramesOffsetName
- AddNonZeroAllKeysName
- AddNonZeroTargetAllKeysAllFrames
- AddNonZeroTargetAllKeysAllFramesName
- AddNonZeroTargetAllKeysAllFramesOffset
- AddNonZeroTargetAllKeysAllFramesOffsetName
- AppendAllFrames
- AppendAllFramesName
- ClearAll
- ClearAnimationType
- CloseAll
- CloseAnimation
- CloseAnimationName
- CopyAllKeys
- CopyAllKeysName
- CopyAnimation
- CopyAnimationName
- CopyKeyColor
- CopyKeyColorName
- CopyKeysColor
- CopyKeysColorAllFrames
- CopyKeysColorAllFramesName

- CopyKeysColorName
- CopyNonZeroAllKeys
- CopyNonZeroAllKeysAllFrames
- CopyNonZeroAllKeysAllFramesName
- CopyNonZeroAllKeysAllFramesOffset
- CopyNonZeroAllKeysAllFramesOffsetName
- CopyNonZeroAllKeysName
- CopyNonZeroAllKeysOffset
- CopyNonZeroAllKeysOffsetName
- CopyNonZeroKeyColor
- CopyNonZeroKeyColorName
- CopyNonZeroTargetAllKeys
- CopyNonZeroTargetAllKeysAllFrames
- CopyNonZeroTargetAllKeysAllFramesName
- CopyNonZeroTargetAllKeysAllFramesOffset
- CopyNonZeroTargetAllKeysAllFramesOffsetName
- CopyNonZeroTargetAllKeysName
- CopyZeroTargetAllKeysAllFrames
- CopyZeroTargetAllKeysAllFramesName
- DuplicateFirstFrame
- DuplicateFirstFrameName
- DuplicateFrames
- DuplicateFramesName
- DuplicateMirrorFrames
- DuplicateMirrorFramesName
- FadeEndFrames
- FadeEndFramesName
- FadeStartFrames

- FadeStartFramesName
- FillColor
- FillColorAllFrames
- FillColorAllFramesName
- FillColorAllFramesRGB
- FillColorAllFramesRGBName
- FillColorName
- FillColorRGB
- FillColorRGBName
- FillNonZeroColor
- FillNonZeroColorAllFrames
- FillNonZeroColorAllFramesName
- FillNonZeroColorAllFramesRGB
- FillNonZeroColorAllFramesRGBName
- FillNonZeroColorName
- FillNonZeroColorRGB
- FillNonZeroColorRGBName
- FillRandomColors
- FillRandomColorsAllFrames
- FillRandomColorsAllFramesName
- FillRandomColorsBlackAndWhite
- FillRandomColorsBlackAndWhiteAllFrames
- FillRandomColorsBlackAndWhiteAllFramesName
- FillRandomColorsBlackAndWhiteName
- FillRandomColorsName
- FillThresholdColorsAllFrames
- FillThresholdColorsAllFramesName
- FillThresholdColorsAllFramesRGB

- FillThresholdColorsAllFramesRGBName
- FillThresholdColorsMinMaxAllFramesRGB
- FillThresholdColorsMinMaxAllFramesRGBName
- FillThresholdColorsRGB
- FillThresholdColorsRGBName
- FillThresholdRGBColorsAllFramesRGB
- FillThresholdRGBColorsAllFramesRGBName
- FillZeroColor
- FillZeroColorAllFrames
- FillZeroColorAllFramesName
- FillZeroColorAllFramesRGB
- FillZeroColorAllFramesRGBName
- FillZeroColorName
- FillZeroColorRGB
- FillZeroColorRGBName
- GetAnimation
- GetAnimationCount
- GetAnimationId
- GetAnimationName
- GetCurrentFrame
- GetCurrentFrameName
- GetFrameCount
- GetFrameCountName
- GetFrameDuration
- GetFrameDurationName
- GetKeyColor
- GetKeyColorName
- GetMaxColumn

- GetMaxLeds
- GetMaxRow
- GetPlayingAnimationCount
- GetPlayingAnimationId
- GetRGB
- GetTotalDuration
- GetTotalDurationName
- InsertDelay
- InsertDelayName
- InsertFrame
- InsertFrameName
- InvertColorsAllFrames
- InvertColorsAllFramesName
- IsActive
- IsConnected
- IsInitialized
- Lerp
- LerpColor
- LoadAnimation
- LoadAnimationName
- MakeBlankFrames
- MakeBlankFramesName
- MakeBlankFramesRandom
- MakeBlankFramesRandomBlackAndWhite
- MakeBlankFramesRandomBlackAndWhiteName
- MakeBlankFramesRandomName
- MakeBlankFramesRGB
- MakeBlankFramesRGBName

- MultiplyColorLerpAllFrames
- MultiplyColorLerpAllFramesName
- MultiplyIntensity
- MultiplyIntensityAllFrames
- MultiplyIntensityAllFramesName
- MultiplyIntensityAllFramesRGB
- MultiplyIntensityAllFramesRGBName
- MultiplyIntensityColor
- MultiplyIntensityColorAllFrames
- MultiplyIntensityColorAllFramesName
- MultiplyIntensityColorName
- MultiplyIntensityName
- MultiplyIntensityRGB
- MultiplyIntensityRGBName
- MultiplyNonZeroTargetColorLerpAllFrames
- MultiplyNonZeroTargetColorLerpAllFramesName
- MultiplyTargetColorLerpAllFrames
- MultiplyTargetColorLerpAllFramesName
- OffsetColors
- OffsetColorsAllFrames
- OffsetColorsAllFramesName
- OffsetColorsName
- OffsetNonZeroColors
- OffsetNonZeroColorsAllFrames
- OffsetNonZeroColorsAllFramesName
- OffsetNonZeroColorsName
- OpenAnimationFromMemory
- OverrideFrameDurationName

- PlayAnimation
- PlayAnimationName
- PreviewFrame
- PreviewFrameName
- ReduceFrames
- ReduceFramesName
- ReverseAllFrames
- ReverseAllFramesName
- SetChromaCustomColorAllFramesName
- SetChromaCustomFlagName
- SetCurrentFrame
- SetCurrentFrameName
- SetEventName
- SetIdleAnimationName
- SetKeyColor
- SetKeyColorAllFrames
- SetKeyColorAllFramesName
- SetKeyColorName
- SetKeyNonZeroColor
- SetKeyNonZeroColorName
- SetKeyRowColumnColorName
- SetKeysColor
- SetKeysColorAllFrames
- SetKeysColorAllFramesName
- SetKeysColorAllFramesRGB
- SetKeysColorAllFramesRGBName
- SetKeysColorName
- SetKeysColorRGB

- SetKeysColorRGBName
- SetKeysNonZeroColor
- SetKeysNonZeroColorAllFrames
- SetKeysNonZeroColorAllFramesName
- SetKeysNonZeroColorName
- SetStaticColor
- SetStaticColorAll
- StopAll
- StopAnimation
- StopAnimationType
- StreamBroadcast
- StreamBroadcastEnd
- StreamGetAuthShortcode
- StreamGetFocus
- StreamGetId
- StreamGetKey
- StreamGetStatusString
- StreamReleaseShortcode
- StreamSetFocus
- StreamWatch
- StreamWatchEnd
- SubtractNonZeroAllKeys
- SubtractNonZeroAllKeysAllFrames
- SubtractNonZeroAllKeysAllFramesName
- SubtractNonZeroAllKeysAllFramesOffset
- SubtractNonZeroAllKeysAllFramesOffsetName
- SubtractNonZeroAllKeysName
- SubtractNonZeroTargetAllKeysAllFrames

- SubtractNonZeroTargetAllKeysAllFramesName
- SubtractNonZeroTargetAllKeysAllFramesOffset
- SubtractNonZeroTargetAllKeysAllFramesOffsetName
- TrimEndFrames
- TrimEndFramesName
- TrimFrame
- TrimFrameName
- TrimStartFrames
- TrimStartFramesName
- UnloadAnimation
- UnloadAnimationName
- UseForwardChromaEvents
- UseIdleAnimation
- UseIdleAnimations
- UsePreloading
- UsePreloadingName

AddNonZeroAllKeys

Add source color to target where color is not black for frame id, reference source and target by id.

```
void UChromaSDKPluginBPLibrary::AddNonZeroAllKeys(int32 sourceAnimationId,
    int32 targetAnimationId, int32 frameId);
```

AddNonZeroAllKeysAllFrames

Add source color to target where color is not black for all frames, reference source and target by id.

```
void UChromaSDKPluginBPLibrary::AddNonZeroAllKeysAllFrames(int32
sourceAnimationId,
    int32 targetAnimationId);
```

AddNonZeroAllKeysAllFramesName

Add source color to target where color is not black for all frames, reference source and target by name.

```
void UChromaSDKPluginBPLibrary::AddNonZeroAllKeysAllFramesName(const FString&
    sourceAnimationName, const FString& targetAnimationName);
```

AddNonZeroAllKeysAllFramesOffset

Add source color to target where color is not black for all frames starting at offset for the length of the source, reference source and target by id.

```
void UChromaSDKPluginBPLibrary::AddNonZeroAllKeysAllFramesOffset(int32
sourceAnimationId,
    int32 targetAnimationId, int32 offset);
```

AddNonZeroAllKeysAllFramesOffsetName

Add source color to target where color is not black for all frames starting at offset for the length of the source, reference source and target by name.

```
void UChromaSDKPluginBPLibrary::AddNonZeroAllKeysAllFramesOffsetName(const
    FString& sourceAnimationName, const FString& targetAnimationName, int32
    offset);
```

AddNonZeroAllKeysName

Add source color to target where color is not black for frame id, reference source and target by name.

```
void UChromaSDKPluginBPLibrary::AddNonZeroAllKeysName(const FString&
sourceAnimationName,
    const FString& targetAnimationName, int32 frameId);
```

AddNonZeroTargetAllKeysAllFrames

Add source color to target where the target color is not black for all frames, reference source and target by id.

```
void UChromaSDKPluginBPLibrary::AddNonZeroTargetAllKeysAllFrames(int32
sourceAnimationId,
    int32 targetAnimationId);
```

AddNonZeroTargetAllKeysAllFramesName

Add source color to target where the target color is not black for all frames, reference source and target by name.

```
void UChromaSDKPluginBPLibrary::AddNonZeroTargetAllKeysAllFramesName(const
    FString& sourceAnimationName, const FString& targetAnimationName);
```

AddNonZeroTargetAllKeysAllFramesOffset

Add source color to target where the target color is not black for all frames starting at offset for the length of the source, reference source and target by id.

```
void UChromaSDKPluginBPLibrary::AddNonZeroTargetAllKeysAllFramesOffset(int32
    sourceAnimationId, int32 targetAnimationId, int32 offset);
```

AddNonZeroTargetAllKeysAllFramesOffsetName

Add source color to target where the target color is not black for all frames starting at offset for the length of the source, reference source and target by name.

```
void UChromaSDKPluginBPLibrary::AddNonZeroTargetAllKeysAllFramesOffsetName(const
    FString& sourceAnimationName, const FString& targetAnimationName, int32
    offset);
```

AppendAllFrames

Append all source frames to the target animation, reference source and target by id.

```
void UChromaSDKPluginBPLibrary::AppendAllFrames(int32 sourceAnimationId,
    int32 targetAnimationId);
```

AppendAllFramesName

Append all source frames to the target animation, reference source and target by name.

```
void UChromaSDKPluginBPLibrary::AppendAllFramesName(const FString&
sourceAnimationName,
    const FString& targetAnimationName);
```

ClearAll

`PluginClearAll` will issue a `CLEAR` effect for all devices.

```
void UChromaSDKPluginBPLibrary::ClearAll();
```

ClearAnimationType

`PluginClearAnimationType` will issue a `CLEAR` effect for the given device.

```
void UChromaSDKPluginBPLibrary::ClearAnimationType(EChromaSDKDeviceEnum::Type
    device);
```

CloseAll

`PluginCloseAll` closes all open animations so they can be reloaded from disk. The set of animations will be stopped if playing.

```
void UChromaSDKPluginBPLibrary::CloseAll();
```

CloseAnimation

Closes the `Chroma` animation to free up resources referenced by id. Returns the animation id upon success. Returns negative one upon failure. This might be used while authoring effects if there was a change necessitating re-opening the animation. The animation id can no longer be used once closed.

```
void UChromaSDKPluginBPLibrary::CloseAnimation(const int32 animationId);
```

CloseAnimationName

Closes the `Chroma` animation referenced by name so that the animation can be reloaded from disk.

```
void UChromaSDKPluginBPLibrary::CloseAnimationName(const FString& animationName);
```

CopyAllKeys

Copy source animation to target animation for the given frame. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyAllKeys(int32 sourceAnimationId, int32
    targetAnimationId, int32 frameId);
```

CopyAllKeysName

Copy source animation to target animation for the given frame. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyAllKeysName(const FString&
sourceAnimationName,
    const FString& targetAnimationName, int32 frameId);
```

CopyAnimation

Copy animation to named target animation in memory. If target animation exists, close first. Source is referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyAnimation(int32 sourceAnimationId, const
    FString& targetAnimationName);
```

CopyAnimationName

Copy animation to named target animation in memory. If target animation exists, close first. Source is referenced by name.

```
void UChromaSDKPluginBPLibrary::CopyAnimationName(const FString&
sourceAnimationName,
    const FString& targetAnimationName);
```

CopyKeyColor

Copy animation key color from the source animation to the target animation for the given frame. Reference the source and target by id.

```
void UChromaSDKPluginBPLibrary::CopyKeyColor(int32 sourceAnimationId, int32
    targetAnimationId, int32 frameIndex, EChromaSDKKeyboardKey::Type key);
```

CopyKeyColorName

Copy animation key color from the source animation to the target animation for the given frame.

```
void UChromaSDKPluginBPLibrary::CopyKeyColorName(const FString&
sourceAnimationName,
    const FString& targetAnimationName, const int32 frameIndex,
EChromaSDKKeyboardKey::Type
    key);
```

CopyKeysColor

Copy animation color for a set of keys from the source animation to the target animation for the given frame. Reference the source and target by id.

```
void UChromaSDKPluginBPLibrary::CopyKeysColor(int32 sourceAnimationId, int32
    targetAnimationId, int32 frameIndex, const
TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys);
```

CopyKeysColorAllFrames

Copy animation color for a set of keys from the source animation to the target animation for all frames. Reference the source and target by id.

```
void UChromaSDKPluginBPLibrary::CopyKeysColorAllFrames(int32 sourceAnimationId,
    int32 targetAnimationId, const
TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys);
```

CopyKeysColorAllFramesName

Copy animation color for a set of keys from the source animation to the target animation for all frames. Reference the source and target by name.

```
void UChromaSDKPluginBPLibrary::CopyKeysColorAllFramesName(const FString&
    sourceAnimationName, const FString& targetAnimationName, const
TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys);
```

CopyKeysColorName

Copy animation color for a set of keys from the source animation to the target animation for the given frame. Reference the source and target by name.

```
void UChromaSDKPluginBPLibrary::CopyKeysColorName(const FString&
sourceAnimationName,
    const FString& targetAnimationName, const int32 frameIndex, const
TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys);
```

CopyNonZeroAllKeys

Copy source animation to target animation for the given frame. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeys(int32 sourceAnimationId,
    int32 targetAnimationId, int32 frameId);
```

CopyNonZeroAllKeysAllFrames

Copy nonzero colors from a source animation to a target animation for all frames. Reference source and target by id.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeysAllFrames(int32
sourceAnimationId,
    int32 targetAnimationId);
```

CopyNonZeroAllKeysAllFramesName

Copy nonzero colors from a source animation to a target animation for all frames. Reference source and target by name.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeysAllFramesName(const FString&
    sourceAnimationName, const FString& targetAnimationName);
```

CopyNonZeroAllKeysAllFramesOffset

Copy nonzero colors from a source animation to a target animation for all frames starting at the offset for the length of the source animation. The source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeysAllFramesOffset(int32
    sourceAnimationId, int32 targetAnimationId, int32 offset);
```

```
```

CopyNonZeroAllKeysAllFramesOffsetName

Copy nonzero colors from a source animation to a target animation for all frames starting at the offset for the length of the source animation. The source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeysAllFramesOffsetName(const
    FString& sourceAnimationName, const FString& targetAnimationName, int32
    offset);
```

CopyNonZeroAllKeysName

Copy nonzero colors from source animation to target animation for the specified frame. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeysName(const FString&
sourceAnimationName,
    const FString& targetAnimationName, int32 frameId);
```

CopyNonZeroAllKeysOffset

Copy nonzero colors from the source animation to the target animation from the source frame to the target offset frame. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeysOffset(int32 sourceAnimationId,
    int32 targetAnimationId, int32 frameId, int32 offset);
```

CopyNonZeroAllKeysOffsetName

Copy nonzero colors from the source animation to the target animation from the source frame to the target offset frame. Source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroAllKeysOffsetName(const FString&
    sourceAnimationName, const FString& targetAnimationName, int32 frameId,
    int32 offset);
```

CopyNonZeroKeyColor

Copy animation key color from the source animation to the target animation for the given frame where color is not zero.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroKeyColor(int32 sourceAnimationId,
    int32 targetAnimationId, int32 frameIndex, EChromaSDKKeyboardKey::Type
    key);
```

CopyNonZeroKeyColorName

Copy animation key color from the source animation to the target animation for the given frame where color is not zero.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroKeyColorName(const FString&
sourceAnimationName,
    const FString& targetAnimationName, const int32 frameIndex,
EChromaSDKKeyboardKey::Type
    key);
```

CopyNonZeroTargetAllKeys

Copy nonzero colors from the source animation to the target animation where the target color is nonzero for the specified frame. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroTargetAllKeys(int32 sourceAnimationId,
    int32 targetAnimationId, int32 frameId);
```

CopyNonZeroTargetAllKeysAllFrames

Copy nonzero colors from the source animation to the target animation where the target color is nonzero for all frames. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroTargetAllKeysAllFrames(int32
    sourceAnimationId, int32 targetAnimationId);
```

CopyNonZeroTargetAllKeysAllFramesName

Copy nonzero colors from the source animation to the target animation where the target color is nonzero for all frames. Source and target are referenced by name.

```

```

```
void UChromaSDKPluginBPLibrary::CopyNonZeroTargetAllKeysAllFramesName(const
    FString& sourceAnimationName, const FString& targetAnimationName);
```

CopyNonZeroTargetAllKeysAllFramesOffset

Copy nonzero colors from the source animation to the target animation where the target color is nonzero for all frames. Source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroTargetAllKeysAllFramesOffset(int32
    sourceAnimationId, int32 targetAnimationId, int32 offset);
```

CopyNonZeroTargetAllKeysAllFramesOffsetName

Copy nonzero colors from the source animation to the target animation where the target color is nonzero for all frames starting at the target offset for the length of the source animation. Source and target animations are referenced by name.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroTargetAllKeysAllFramesOffsetName(const
    FString& sourceAnimationName, const FString& targetAnimationName, int32
    offset);
```

CopyNonZeroTargetAllKeysName

Copy nonzero colors from the source animation to the target animation where the target color is nonzero for the specified frame. The source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::CopyNonZeroTargetAllKeysName(const FString&
    sourceAnimationName, const FString& targetAnimationName, int32 frameId);
```

CopyZeroTargetAllKeysAllFrames

Copy nonzero color from source animation to target animation where target is zero for all frames. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::CopyZeroTargetAllKeysAllFrames(int32
sourceAnimationId,
    int32 targetAnimationId);
```

CopyZeroTargetAllKeysAllFramesName

Copy nonzero color from source animation to target animation where target is zero for all frames. Source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::CopyZeroTargetAllKeysAllFramesName(const
    FString& sourceAnimationName, const FString& targetAnimationName);
```

DuplicateFirstFrame

Duplicate the first animation frame so that the animation length matches the frame count. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::DuplicateFirstFrame(int32 animationId, int32
    frameCount);
```

DuplicateFirstFrameName

Duplicate the first animation frame so that the animation length matches the frame count. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::DuplicateFirstFrameName(const FString&
animationName,
    int32 frameCount);
```

DuplicateFrames

Duplicate all the frames of the animation to double the animation length. Frame 1 becomes frame 1 and 2. Frame 2 becomes frame 3 and 4. And so on. The animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::DuplicateFrames(int32 animationId);
```

DuplicateFramesName

Duplicate all the frames of the animation to double the animation length. Frame 1 becomes frame 1 and 2. Frame 2 becomes frame 3 and 4. And so on. The animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::DuplicateFramesName(const FString& animationName);
```

DuplicateMirrorFrames

Duplicate all the animation frames in reverse so that the animation plays forwards and backwards. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::DuplicateMirrorFrames(int32 animationId);
```

DuplicateMirrorFramesName

Duplicate all the animation frames in reverse so that the animation plays forwards and backwards. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::DuplicateMirrorFramesName(const FString&
    animationName);
```

FadeEndFrames

Fade the animation to black starting at the fade frame index to the end of the animation. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FadeEndFrames(int32 animationId, int32 fade);
```

FadeEndFramesName

Fade the animation to black starting at the fade frame index to the end of the animation. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FadeEndFramesName(const FString& animationName,
    int32 fade);
```

FadeStartFrames

Fade the animation from black to full color starting at 0 to the fade frame index. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FadeStartFrames(int32 animationId, int32
    fade);
```

FadeStartFramesName

Fade the animation from black to full color starting at 0 to the fade frame index. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FadeStartFramesName(const FString& animationName,
    int32 fade);
```

FillColor

Set the RGB value for all colors in the specified frame. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillColor(int32 animationId, int32 frameId,
    const FLinearColor& colorParam);
```

FillColorAllFrames

Set the RGB value for all colors for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillColorAllFrames(int32 animationId, const
    FLinearColor& colorParam);
```

FillColorAllFramesName

Set the RGB value for all colors for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillColorAllFramesName(const FString&
animationName,
    const FLinearColor& colorParam);
```

FillColorAllFramesRGB

Set the RGB value for all colors for all frames. Use the range of 0 to 255 for red, green, and blue parameters. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillColorAllFramesRGB(int32 animationId,
    int32 red, int32 green, int32 blue);
```

FillColorAllFramesRGBName

Set the RGB value for all colors for all frames. Use the range of 0 to 255 for red, green, and blue parameters. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillColorAllFramesRGBName(const FString&
    animationName, int32 red, int32 green, int32 blue);
```

FillColorName

Set the RGB value for all colors in the specified frame. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillColorName(const FString& animationName,
    int32 frameId, const FLinearColor& colorParam);
```

FillColorRGB

Set the RGB value for all colors in the specified frame. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillColorRGB(int32 animationId, int32 frameId,
    int32 red, int32 green, int32 blue);
```

FillColorRGBName

Set the RGB value for all colors in the specified frame. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillColorRGBName(const FString& animationName,
    int32 frameId, int32 red, int32 green, int32 blue);
```

FillNonZeroColor

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors in the specified frame. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColor(int32 animationId, int32
    frameId, const FLinearColor& colorParam);
```

FillNonZeroColorAllFrames

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColorAllFrames(int32 animationId,
    const FLinearColor& colorParam);
```

FillNonZeroColorAllFramesName

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColorAllFramesName(const FString&
    animationName, const FLinearColor& colorParam);
```

FillNonZeroColorAllFramesRGB

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors for all frames. Use the range of 0 to 255 for red, green, and blue parameters. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColorAllFramesRGB(int32 animationId,
    int32 red, int32 green, int32 blue);
```

FillNonZeroColorAllFramesRGBName

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors for all frames. Use the range of 0 to 255 for red, green, and blue parameters. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColorAllFramesRGBName(const FString&
    animationName, int32 red, int32 green, int32 blue);
```

FillNonZeroColorName

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors in the specified frame. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColorName(const FString& animationName,
    int32 frameId, const FLinearColor& colorParam);
```

FillNonZeroColorRGB

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors in the specified frame. Use the range of 0 to 255 for red, green, and blue parameters. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColorRGB(int32 animationId, int32
    frameId, int32 red, int32 green, int32 blue);
```

FillNonZeroColorRGBName

This method will only update colors in the animation that are not already set to black. Set the RGB value for a subset of colors in the specified frame. Use the range of 0 to 255 for red, green, and blue parameters. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillNonZeroColorRGBName(const FString&
animationName,
    int32 frameId, int32 red, int32 green, int32 blue);
```

FillRandomColors

Fill the frame with random RGB values for the given frame. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillRandomColors(int32 animationId, int32
    frameId);
```

FillRandomColorsAllFrames

Fill the frame with random RGB values for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillRandomColorsAllFrames(int32 animationId);
```

FillRandomColorsAllFramesName

Fill the frame with random RGB values for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillRandomColorsAllFramesName(const FString&
    animationName);
```

FillRandomColorsBlackAndWhite

Fill the frame with random black and white values for the specified frame. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillRandomColorsBlackAndWhite(int32 animationId,
    int32 frameId);
```

FillRandomColorsBlackAndWhiteAllFrames

Fill the frame with random black and white values for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillRandomColorsBlackAndWhiteAllFrames(int32
    animationId);
```

FillRandomColorsBlackAndWhiteAllFramesName

Fill the frame with random black and white values for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillRandomColorsBlackAndWhiteAllFramesName(const
    FString& animationName);
```

FillRandomColorsBlackAndWhiteName

Fill the frame with random black and white values for the specified frame. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillRandomColorsBlackAndWhiteName(const
    FString& animationName, int32 frameId);
```

FillRandomColorsName

Fill the frame with random RGB values for the given frame. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillRandomColorsName(const FString& animationName,
    int32 frameId);
```

FillThresholdColorsAllFrames

Fill all frames with RGB color where the animation color is less than the RGB threshold. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsAllFrames(int32 animationId,
    int32 threshold, const FLinearColor& colorParam);
```

FillThresholdColorsAllFramesName

Fill all frames with RGB color where the animation color is less than the RGB threshold. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsAllFramesName(const FString&
    animationName, int32 threshold, const FLinearColor& colorParam);
```

FillThresholdColorsAllFramesRGB

Fill all frames with RGB color where the animation color is less than the threshold. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsAllFramesRGB(int32 animationId,
    int32 threshold, int32 red, int32 green, int32 blue);
```

FillThresholdColorsAllFramesRGBName

Fill all frames with RGB color where the animation color is less than the threshold. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsAllFramesRGBName(const
    FString& animationName, int32 threshold, int32 red, int32 green, int32
    blue);
```

FillThresholdColorsMinMaxAllFramesRGB

Fill all frames with the min RGB color where the animation color is less than the min threshold AND with the max RGB color where the animation is more than the max threshold. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsMinMaxAllFramesRGB(int32
    animationId, int32 minThreshold, int32 minRed, int32 minGreen, int32 minBlue,
    int32 maxThreshold, int32 maxRed, int32 maxGreen, int32 maxBlue);
```

FillThresholdColorsMinMaxAllFramesRGBName

Fill all frames with the min RGB color where the animation color is less than the min threshold AND with the max RGB color where the animation is more than the max threshold. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsMinMaxAllFramesRGBName(const
    FString& animationName, int32 minThreshold, int32 minRed, int32 minGreen,
    int32 minBlue, int32 maxThreshold, int32 maxRed, int32 maxGreen, int32
    maxBlue);
```

FillThresholdColorsRGB

Fill the specified frame with RGB color where the animation color is less than the RGB threshold. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsRGB(int32 animationId,
    int32 frameId, int32 threshold, int32 red, int32 green, int32 blue);
```

FillThresholdColorsRGBName

Fill the specified frame with RGB color where the animation color is less than the RGB threshold. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillThresholdColorsRGBName(const FString&
    animationName, int32 frameId, int32 threshold, int32 red, int32 green,
    int32 blue);
```

FillThresholdRGBColorsAllFramesRGB

Fill all frames with RGB color where the animation color is less than the RGB threshold. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillThresholdRGBColorsAllFramesRGB(int32
    animationId, int32 redThreshold, int32 greenThreshold, int32 blueThreshold,
    int32 red, int32 green, int32 blue);
```

FillThresholdRGBColorsAllFramesRGBName

Fill all frames with RGB color where the animation color is less than the RGB threshold. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillThresholdRGBColorsAllFramesRGBName(const
    FString& animationName, int32 redThreshold, int32 greenThreshold, int32
    blueThreshold, int32 red, int32 green, int32 blue);
```

FillZeroColor

Fill the specified frame with RGB color where the animation color is zero. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillZeroColor(int32 animationId, int32 frameId,
    const FLinearColor& colorParam);
```

FillZeroColorAllFrames

Fill all frames with RGB color where the animation color is zero. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillZeroColorAllFrames(int32 animationId,
    const FLinearColor& colorParam);
```

FillZeroColorAllFramesName

Fill all frames with RGB color where the animation color is zero. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillZeroColorAllFramesName(const FString&
    animationName, const FLinearColor& colorParam);
```

FillZeroColorAllFramesRGB

Fill all frames with RGB color where the animation color is zero. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillZeroColorAllFramesRGB(int32 animationId,
    int32 red, int32 green, int32 blue);
```

FillZeroColorAllFramesRGBName

Fill all frames with RGB color where the animation color is zero. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillZeroColorAllFramesRGBName(const FString&
    animationName, int32 red, int32 green, int32 blue);
```

FillZeroColorName

Fill the specified frame with RGB color where the animation color is zero. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillZeroColorName(const FString& animationName,
    int32 frameId, const FLinearColor& colorParam);
```

FillZeroColorRGB

Fill the specified frame with RGB color where the animation color is zero. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::FillZeroColorRGB(int32 animationId, int32
    frameId, int32 red, int32 green, int32 blue);
```

FillZeroColorRGBName

Fill the specified frame with RGB color where the animation color is zero. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::FillZeroColorRGBName(const FString& animationName,
    int32 frameId, int32 red, int32 green, int32 blue);
```

GetAnimation

Get the animation id for the named animation.

```
int32 UChromaSDKPluginBPLibrary::GetAnimation(const FString& animationName);
```

GetAnimationCount

`PluginGetAnimationCount` will return the number of loaded animations.

```
int32 UChromaSDKPluginBPLibrary::GetAnimationCount();
```

GetAnimationId

`PluginGetAnimationId` will return the `animationId` given the `index` of the loaded animation. The `index` is zero-based and less than the number returned by `PluginGetAnimationCount`. Use `PluginGetAnimationName` to get the name of the animation.

```
int32 UChromaSDKPluginBPLibrary::GetAnimationId(const FString& animationName);
```

GetAnimationName

`PluginGetAnimationName` takes an `animationId` and returns the name of the animation of the `.chroma` animation file. If a name is not available then an empty string will be returned.

```
FString UChromaSDKPluginBPLibrary::GetAnimationName(const int32 animationId);
```

GetCurrentFrame

Get the current frame of the animation referenced by id.

```
int32 UChromaSDKPluginBPLibrary::GetCurrentFrame(int32 animationId);
```

GetCurrentFrameName

Get the current frame of the animation referenced by name.

```
int32 UChromaSDKPluginBPLibrary::GetCurrentFrameName(const FString&
animationName);
```

GetFrameCount

Returns the frame count of a `Chroma` animation upon success. Returns negative one upon failure.

```
int32 UChromaSDKPluginBPLibrary::GetFrameCount(const int32 animationId);
```

GetFrameCountName

Returns the frame count of a `Chroma` animation upon success. Returns negative one upon failure.

```
int32 UChromaSDKPluginBPLibrary::GetFrameCountName(const FString& animationName);
```

GetFrameDuration

Returns the duration of an animation frame in seconds upon success. Returns zero upon failure.

```
float UChromaSDKPluginBPLibrary::GetFrameDuration(int32 animationId, int32
    frameId);
```

GetFrameDurationName

Returns the duration of an animation frame in seconds upon success. Returns zero upon failure.

```
float UChromaSDKPluginBPLibrary::GetFrameDurationName(const FString&
animationName,
    int32 frameId);
```

GetKeyColor

Get the color of an animation key for the given frame referenced by id.

```
FLinearColor UChromaSDKPluginBPLibrary::GetKeyColor(int32 animationId, int32
    frameIndex, EChromaSDKKeyboardKey::Type key);
```

GetKeyColorName

Get the color of an animation key for the given frame referenced by name.

```
FLinearColor UChromaSDKPluginBPLibrary::GetKeyColorName(const FString&
animationName,
    const int32 frameIndex, EChromaSDKKeyboardKey::Type key);
```

GetMaxColumn

Returns the `MAX COLUMN` given the `EChromaSDKDevice2DEnum` device as an integer upon success. Returns negative one upon failure.

```
int32 UChromaSDKPluginBPLibrary::GetMaxColumn(EChromaSDKDevice2DEnum::Type
    device);
```

GetMaxLeds

Returns the MAX LEDS given the `EChromaSDKDevice1DEnum` device as an integer upon success. Returns negative one upon failure.

```
int32 UChromaSDKPluginBPLibrary::GetMaxLeds(EChromaSDKDevice1DEnum::Type
    device);
```

GetMaxRow

Returns the `MAX ROW` given the `EChromaSDKDevice2DEnum` device as an integer upon success. Returns negative one upon failure.

```
int32 UChromaSDKPluginBPLibrary::GetMaxRow(EChromaSDKDevice2DEnum::Type
    device);
```

GetPlayingAnimationCount

`PluginGetPlayingAnimationCount` will return the number of playing animations.

```
int32 UChromaSDKPluginBPLibrary::GetPlayingAnimationCount();
```

GetPlayingAnimationId

`PluginGetPlayingAnimationId` will return the `animationId` given the `index` of the playing animation. The `index` is zero-based and less than the number returned by `PluginGetPlayingAnimationCount`. Use `PluginGetAnimationName` to get the name of the animation.

```
int32 UChromaSDKPluginBPLibrary::GetPlayingAnimationId(int32 index);
```

GetRGB

Get the RGB color given red, green, and blue.

```
FLinearColor UChromaSDKPluginBPLibrary::GetRGB(int32 red, int32 green, int32
    blue);
```

GetTotalDuration

Returns the total duration of an animation in seconds upon success. Returns zero upon failure.

```
float UChromaSDKPluginBPLibrary::GetTotalDuration(int32 animationId);
```

GetTotalDurationName

Returns the total duration of an animation in seconds upon success. Returns zero upon failure.

```
float UChromaSDKPluginBPLibrary::GetTotalDurationName(const FString&
animationName);
```

InsertDelay

Insert an animation delay by duplicating the frame by the delay number of times. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::InsertDelay(int32 animationId, int32 frameId,
    int32 delay);
```

InsertDelayName

Insert an animation delay by duplicating the frame by the delay number of times. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::InsertDelayName(const FString& animationName,
    int32 frameId, int32 delay);
```

InsertFrame

Duplicate the source frame index at the target frame index. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::InsertFrame(int32 animationId, int32 sourceFrame,
    int32 targetFrame);
```

```
```

InsertFrameName

Duplicate the source frame index at the target frame index. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::InsertFrameName(const FString& animationName,
    int32 sourceFrame, int32 targetFrame);
```

InvertColorsAllFrames

Invert all the colors for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::InvertColorsAllFrames(int32 animationId);
```

InvertColorsAllFramesName

Invert all the colors for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::InvertColorsAllFramesName(const FString&
    animationName);
```

IsActive

Direct access to low level API.

```
int32 UChromaSDKPluginBPLibrary::IsActive(UPARAM(ref) bool& active);
```

IsConnected

Direct access to low level API.

```
int32 UChromaSDKPluginBPLibrary::IsConnected(UPARAM(ref) FChromaSDKDeviceInfoType&
    deviceInfoType);
```

IsInitialized

Returns true if the plugin has been initialized. Returns false if the plugin is uninitialized.

```
bool UChromaSDKPluginBPLibrary::IsInitialized();
```

Lerp

Do a lerp math operation on a float.

```
float UChromaSDKPluginBPLibrary::Lerp(float start, float end, float amt);
```

LerpColor

Lerp from one color to another given t in the range 0.0 to 1.0.

```
FLinearColor UChromaSDKPluginBPLibrary::LerpColor(FLinearColor colorParam1,
    FLinearColor colorParam2, float t);
```

LoadAnimation

Loads `Chroma` effects so that the animation can be played immediately. Returns the animation id upon success. Returns negative one upon failure.

```
void UChromaSDKPluginBPLibrary::LoadAnimation(const int32 animationId);
```

LoadAnimationName

Load the named animation.

```
void UChromaSDKPluginBPLibrary::LoadAnimationName(const FString& animationName);
```

MakeBlankFrames

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color defaults to color. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MakeBlankFrames(int32 animationId, int32
    frameCount, float duration, const FLinearColor& colorParam);
```

MakeBlankFramesName

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color defaults to color. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MakeBlankFramesName(const FString& animationName,
    int32 frameCount, float duration, const FLinearColor& colorParam);
```

MakeBlankFramesRandom

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color is random. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MakeBlankFramesRandom(int32 animationId,
    int32 frameCount, float duration);
```

MakeBlankFramesRandomBlackAndWhite

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color is random black and white. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MakeBlankFramesRandomBlackAndWhite(int32
    animationId, int32 frameCount, float duration);
```

MakeBlankFramesRandomBlackAndWhiteName

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color is random black and white. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MakeBlankFramesRandomBlackAndWhiteName(const
    FString& animationName, int32 frameCount, float duration);
```

MakeBlankFramesRandomName

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color is random. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MakeBlankFramesRandomName(const FString&
    animationName, int32 frameCount, float duration);
```

MakeBlankFramesRGB

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color defaults to color. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MakeBlankFramesRGB(int32 animationId, int32
    frameCount, float duration, int32 red, int32 green, int32 blue);
```

MakeBlankFramesRGBName

Make a blank animation for the length of the frame count. Frame duration defaults to the duration. The frame color defaults to color. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MakeBlankFramesRGBName(const FString&
animationName,
    int32 frameCount, float duration, int32 red, int32 green, int32 blue);
```

MultiplyColorLerpAllFrames

Multiply the color intensity with the lerp result from color 1 to color 2 using the frame index divided by the frame count for the `t` parameter. Animation is referenced in id.

```
void UChromaSDKPluginBPLibrary::MultiplyColorLerpAllFrames(int32 animationId,
    const FLinearColor& colorParam1, const FLinearColor& colorParam2);
```

MultiplyColorLerpAllFramesName

Multiply the color intensity with the lerp result from color 1 to color 2 using the frame index divided by the frame count for the `t` parameter. Animation is referenced in name.

```
void UChromaSDKPluginBPLibrary::MultiplyColorLerpAllFramesName(const FString&
    animationName, const FLinearColor& colorParam1, const FLinearColor&
colorParam2);
```

MultiplyIntensity

Multiply all the colors in the frame by the intensity value. The valid the intensity range is from 0.0 to 255.0. RGB components are multiplied equally. An intensity of 0.5 would half the color value. Black colors in the frame will not be affected by this method.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensity(int32 animationId, int32
    frameId, float intensity);
```

MultiplyIntensityAllFrames

Multiply all the colors for all frames by the intensity value. The valid the intensity range is from 0.0 to 255.0. RGB components are multiplied equally. An intensity of 0.5 would half the color value. Black colors in the frame will not be affected by this method.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityAllFrames(int32 animationId,
    float intensity);
```

MultiplyIntensityAllFramesName

Multiply all the colors for all frames by the intensity value. The valid the intensity range is from 0.0 to 255.0. RGB components are multiplied equally. An intensity of 0.5 would half the color value. Black colors in the frame will not be affected by this method.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityAllFramesName(const FString&
    animationName, float intensity);
```

MultiplyIntensityAllFramesRGB

Multiply all frames by the RBG color intensity. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityAllFramesRGB(int32 animationId,
    int32 red, int32 green, int32 blue);
```

MultiplyIntensityAllFramesRGBName

Multiply all frames by the RBG color intensity. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityAllFramesRGBName(const
    FString& animationName, int32 red, int32 green, int32 blue);
```

MultiplyIntensityColor

Multiply the specific frame by the RBG color intensity. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityColor(int32 animationId,
    int32 frameId, const FLinearColor& colorParam);
```

MultiplyIntensityColorAllFrames

Multiply all frames by the RBG color intensity. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityColorAllFrames(int32 animationId,
    const FLinearColor& colorParam);
```

MultiplyIntensityColorAllFramesName

Multiply all frames by the RBG color intensity. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityColorAllFramesName(const
    FString& animationName, const FLinearColor& colorParam);
```

MultiplyIntensityColorName

Multiply the specific frame by the RBG color intensity. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityColorName(const FString&
    animationName, int32 frameId, const FLinearColor& colorParam);
```

MultiplyIntensityName

Multiply all the colors in the frame by the intensity value. The valid the intensity range is from 0.0 to 255.0. RGB components are multiplied equally. An intensity of 0.5 would half the color value. Black colors in the frame will not be affected by this method.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityName(const FString&
animationName,
    int32 frameId, float intensity);
```

MultiplyIntensityRGB

Multiply the specific frame by the RBG color intensity. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityRGB(int32 animationId,
    int32 frameId, int32 red, int32 green, int32 blue);
```

MultiplyIntensityRGBName

Multiply the specific frame by the RBG color intensity. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MultiplyIntensityRGBName(const FString&
    animationName, int32 frameId, int32 red, int32 green, int32 blue);
```

MultiplyNonZeroTargetColorLerpAllFrames

Multiply all frames by the color lerp result between color 1 and 2 using the frame color value as the `t` value. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MultiplyNonZeroTargetColorLerpAllFrames(int32
    animationId, const FLinearColor& colorParam1, const FLinearColor&
colorParam2);
```

MultiplyNonZeroTargetColorLerpAllFramesName

Multiply all frames by the color lerp result between color 1 and 2 using the frame color value as the `t` value. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MultiplyNonZeroTargetColorLerpAllFramesName(const
    FString& animationName, const FLinearColor& colorParam1, const FLinearColor&
    colorParam2);
```

MultiplyTargetColorLerpAllFrames

Multiply all frames by the color lerp result between color 1 and 2 using the frame color value as the `t` value. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::MultiplyTargetColorLerpAllFrames(int32
animationId,
    const FLinearColor& colorParam1, const FLinearColor& colorParam2);
```

MultiplyTargetColorLerpAllFramesName

Multiply all frames by the color lerp result between color 1 and 2 using the frame color value as the `t` value. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::MultiplyTargetColorLerpAllFramesName(const
    FString& animationName, const FLinearColor& colorParam1, const FLinearColor&
    colorParam2);
```

OffsetColors

Offset all colors in the frame using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetColors(int32 animationId, int32 frameId,
    int32 red, int32 green, int32 blue);
```

OffsetColorsAllFrames

Offset all colors for all frames using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetColorsAllFrames(int32 animationId,
    int32 red, int32 green, int32 blue);
```

OffsetColorsAllFramesName

Offset all colors for all frames using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetColorsAllFramesName(const FString&
    animationName, int32 red, int32 green, int32 blue);
```

OffsetColorsName

Offset all colors in the frame using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetColorsName(const FString& animationName,
    int32 frameId, int32 red, int32 green, int32 blue);
```

OffsetNonZeroColors

This method will only update colors in the animation that are not already set to black. Offset a subset of colors in the frame using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetNonZeroColors(int32 animationId, int32
    frameId, int32 red, int32 green, int32 blue);
```

OffsetNonZeroColorsAllFrames

This method will only update colors in the animation that are not already set to black. Offset a subset of colors for all frames using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetNonZeroColorsAllFrames(int32 animationId,
    int32 red, int32 green, int32 blue);
```

OffsetNonZeroColorsAllFramesName

This method will only update colors in the animation that are not already set to black. Offset a subset of colors for all frames using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetNonZeroColorsAllFramesName(const FString&
    animationName, int32 red, int32 green, int32 blue);
```

OffsetNonZeroColorsName

This method will only update colors in the animation that are not already set to black. Offset a subset of colors in the frame using the RGB offset. Use the range of -255 to 255 for red, green, and blue parameters. Negative values remove color. Positive values add color.

```
void UChromaSDKPluginBPLibrary::OffsetNonZeroColorsName(const FString&
animationName,
    int32 frameId, int32 red, int32 green, int32 blue);
```

OpenAnimationFromMemory

Opens a Chroma animation data from memory so that it can be played. Data is a pointer to BYTE array of the loaded animation in memory. Name will be assigned to the animation when loaded. Returns an animation id >= 0 upon success. Returns negative one if there was a failure. The animation id is used in most of the API methods.

```
void UChromaSDKPluginBPLibrary::OpenAnimationFromMemory(const TArray<uint8>&
    data, const FString& animationName);
```

OverrideFrameDurationName

Override the duration of all frames with the duration value. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::OverrideFrameDurationName(const FString&
    animationName, float duration);
```

PlayAnimation

Plays the Chroma animation. This will load the animation, if not loaded previously. Returns the animation id upon success. Returns negative one upon failure.

```
void UChromaSDKPluginBPLibrary::PlayAnimation(const FString& animationName,
    bool loop);
```

PlayAnimationName

PluginPlayAnimationName automatically handles initializing the ChromaSDK. The named .chroma animation file will be automatically opened. The animation will play with looping on or off.

```
void UChromaSDKPluginBPLibrary::PlayAnimationName(const FString& animationName,
    bool loop);
```

PreviewFrame

Displays the Chroma animation frame on Chroma hardware given the frameId. Returns the animation id upon success. Returns negative one upon failure.

```
int32 UChromaSDKPluginBPLibrary::PreviewFrame(int32 animationId, int32 frameId);
```

PreviewFrameName

Displays the `Chroma` animation frame on `Chroma` hardware given the `frameId`. Animaton is referenced by name.

```
void UChromaSDKPluginBPLibrary::PreviewFrameName(const FString& animationName,
    int32 frameId);
```

ReduceFrames

Reduce the frames of the animation by removing every nth element. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::ReduceFrames(int32 animationId, int32 n);
```

ReduceFramesName

Reduce the frames of the animation by removing every nth element. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::ReduceFramesName(const FString& animationName,
    int32 n);
```

ReverseAllFrames

Reverse the animation frame order of the `Chroma` animation. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::ReverseAllFrames(int32 animationId);
```

ReverseAllFramesName

Reverse the animation frame order of the `Chroma` animation. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::ReverseAllFramesName(const FString&
animationName);
```

SetChromaCustomColorAllFramesName

When custom color is set, the custom key mode will be used. The animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::SetChromaCustomColorAllFramesName(const
    FString& animationName);
```

SetChromaCustomFlagName

Set the Chroma custom key color flag on all frames. `True` changes the layout from grid to key. `True` changes the layout from key to grid. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::SetChromaCustomFlagName(const FString&
animationName,
    bool flag);
```

SetCurrentFrame

Set the current frame of the animation referenced by id.

```
void UChromaSDKPluginBPLibrary::SetCurrentFrame(int32 animationId, int32
    frameId);
```

SetCurrentFrameName

Set the current frame of the animation referenced by name.

```
void UChromaSDKPluginBPLibrary::SetCurrentFrameName(const FString& animationName,
    int32 frameId);
```

SetEventName

Direct access to low level API.

```
int32 UChromaSDKPluginBPLibrary::SetEventName(const FString& name);
```

SetIdleAnimationName

When the idle animation is used, the named animation will play when no other animations are playing. Reference the animation by name.

```
void UChromaSDKPluginBPLibrary::SetIdleAnimationName(const FString&
animationName);
```

SetKeyColor

Set animation key to a static color for the given frame.

```
void UChromaSDKPluginBPLibrary::SetKeyColor(int32 animationId, int32 frameIndex,
    EChromaSDKKeyboardKey::Type key, const FLinearColor& colorParam);
```

SetKeyColorAllFrames

Set the key to the specified key color for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::SetKeyColorAllFrames(int32 animationId,
    EChromaSDKKeyboardKey::Type key, const FLinearColor& colorParam);
```

SetKeyColorAllFramesName

Set the key to the specified key color for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::SetKeyColorAllFramesName(const FString&
    animationName, EChromaSDKKeyboardKey::Type key, const FLinearColor&
colorParam);
```

SetKeyColorName

Set animation key to a static color for the given frame.

```
void UChromaSDKPluginBPLibrary::SetKeyColorName(const FString& animationName,
    const int32 frameIndex, EChromaSDKKeyboardKey::Type key, const FLinearColor&
    colorParam);
```

SetKeyNonZeroColor

Set animation key to a static color for the given frame if the existing color is not already black.

```
void UChromaSDKPluginBPLibrary::SetKeyNonZeroColor(int32 animationId, int32
    frameIndex, EChromaSDKKeyboardKey::Type key, const FLinearColor& colorParam);
```

SetKeyNonZeroColorName

Set animation key to a static color for the given frame if the existing color is not already black.

```
void UChromaSDKPluginBPLibrary::SetKeyNonZeroColorName(const FString&
animationName,
    const int32 frameIndex, EChromaSDKKeyboardKey::Type key, const FLinearColor&
    colorParam);
```

SetKeyRowColumnColorName

Set animation key by row and column to a static color for the given frame.

```
void UChromaSDKPluginBPLibrary::SetKeyRowColumnColorName(const FString&
    animationName, const int32 frameIndex, const int32 row, const int32 column,
    const FLinearColor& colorParam);
```

SetKeysColor

Set an array of animation keys to a static color for the given frame. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::SetKeysColor(int32 animationId, int32 frameIndex,
    const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>& keys, const
FLinearColor&
    colorParam);
```

SetKeysColorAllFrames

Set an array of animation keys to a static color for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::SetKeysColorAllFrames(int32 animationId,
    const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>& keys, const
FLinearColor&
    colorParam);
```

SetKeysColorAllFramesName

Set an array of animation keys to a static color for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::SetKeysColorAllFramesName(const FString&
    animationName, const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys, const FLinearColor& colorParam);
```

SetKeysColorAllFramesRGB

Set an array of animation keys to a static color for all frames. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::SetKeysColorAllFramesRGB(int32 animationId,
    const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>& keys, int32 red,
    int32 green, int32 blue);
```

SetKeysColorAllFramesRGBName

Set an array of animation keys to a static color for all frames. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::SetKeysColorAllFramesRGBName(const FString&
    animationName, const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys, int32 red, int32 green, int32 blue);
```

SetKeysColorName

Set an array of animation keys to a static color for the given frame.

```
void UChromaSDKPluginBPLibrary::SetKeysColorName(const FString& animationName,
    const int32 frameIndex, const
TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys, const FLinearColor& colorParam);
```

SetKeysColorRGB

Set an array of animation keys to a static color for the given frame. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::SetKeysColorRGB(int32 animationId, int32
    frameIndex, const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>& keys,
    int32 red, int32 green, int32 blue);
```

SetKeysColorRGBName

Set an array of animation keys to a static color for the given frame. Animation is referenced by name.

```
void UChromaSDKPluginBPLibrary::SetKeysColorRGBName(const FString& animationName,
    const int32 frameIndex, const
TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys, int32 red, int32 green, int32 blue);
```

SetKeysNonZeroColor

Set an array of animation keys to a static color for the given frame if the existing color is not already black.

```
void UChromaSDKPluginBPLibrary::SetKeysNonZeroColor(int32 animationId, int32
    frameIndex, const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>& keys,
    const FLinearColor& colorParam);
```

SetKeysNonZeroColorAllFrames

Set an array of animation keys to a static color for the given frame where the color is not black. Animation is referenced by id.

```
void UChromaSDKPluginBPLibrary::SetKeysNonZeroColorAllFrames(int32 animationId,
    const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>& keys, const
FLinearColor&
    colorParam);
```

SetKeysNonZeroColorAllFramesName

Set an array of animation keys to a static color for all frames if the existing color is not already black. Reference animation by name.

```
void UChromaSDKPluginBPLibrary::SetKeysNonZeroColorAllFramesName(const FString&
    animationName, const TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys, const FLinearColor& colorParam);
```

SetKeysNonZeroColorName

Set an array of animation keys to a static color for the given frame if the existing color is not already black. Reference animation by name.

```
void UChromaSDKPluginBPLibrary::SetKeysNonZeroColorName(const FString&
animationName,
    const int32 frameIndex, const
TArray<TEnumAsByte<EChromaSDKKeyboardKey::Type>>&
    keys, const FLinearColor& colorParam);
```

SetStaticColor

Sets the target device to the static color.

```
void UChromaSDKPluginBPLibrary::SetStaticColor(EChromaSDKDeviceEnum::Type
    device, const FLinearColor& color);
```

SetStaticColorAll

Sets all devices to the static color.

```
void UChromaSDKPluginBPLibrary::SetStaticColorAll(const FLinearColor& color);
```

StopAll

`PluginStopAll` will automatically stop all animations that are playing.

```
void UChromaSDKPluginBPLibrary::StopAll();
```

StopAnimation

Stops animation playback if in progress. Returns the animation id upon success. Returns negative one upon failure.

```
void UChromaSDKPluginBPLibrary::StopAnimation(const FString& animationName);
```

StopAnimationType

`PluginStopAnimationType` automatically handles initializing the `ChromaSDK`. If any animation is playing for the `deviceType` and `device` combination, it will be stopped.

```
```

```
void UChromaSDKPluginBPLibrary::StopAnimationType(EChromaSDKDeviceEnum::Type
    device);
```

StreamBroadcast

Begin broadcasting Chroma RGB data using the stored stream key as the endpoint. Intended for Cloud Gaming Platforms, restore the streaming key when the game instance is launched to continue streaming. streamId is a null terminated string streamKey is a null terminated string StreamGetStatus() should return the READY status to use this method.

```
void UChromaSDKPluginBPLibrary::StreamBroadcast(const FString& streamId,
    const FString& streamKey);
```

StreamBroadcastEnd

End broadcasting Chroma RGB data. StreamGetStatus() should return the BROADCASTING status to use this method.

```
void UChromaSDKPluginBPLibrary::StreamBroadcastEnd();
```

StreamGetAuthShortcode

shortcode: Pass the address of a preallocated character buffer to get the streaming auth code. The buffer should have a minimum length of 6. length: Length will return as zero if the streaming auth code could not be obtained. If length is greater than zero, it will be the length of the returned streaming auth code. Once you have the shortcode, it should be shown to the user so they can associate the stream with their Razer ID StreamGetStatus() should return the READY status before invoking this method. platform: is the null terminated string that identifies the source of the stream: { GEFORCE_NOW, LUNA, STADIA, GAME_PASS } title: is the null terminated string that identifies the application or game.

```
FString UChromaSDKPluginBPLibrary::StreamGetAuthShortcode(const FString&
    platform, const FString& title);
```

StreamGetFocus

focus: Pass the address of a preallocated character buffer to get the stream focus. The buffer should have a length of 48 length: Length will return as zero if the stream focus could not be obtained. If length is greater than zero, it will be the length of the returned stream focus.

```
FString UChromaSDKPluginBPLibrary::StreamGetFocus();
```

StreamGetId

Intended for Cloud Gaming Platforms, store the stream id to persist in user preferences to continue streaming if the game is suspended or closed. shortcode: The shortcode is a null terminated string. Use the shortcode that authorized the stream to obtain the stream id. streamId should be a preallocated buffer to get the stream key. The buffer should have a length of 48. length: Length will return zero if the key could not be obtained. If the length is greater than zero, it will be the length of the returned streaming id. Retrieve the stream id after authorizing the shortcode. The authorization window will expire in 5 minutes. Be sure to save the stream key before the window expires. StreamGetStatus() should return the READY status to use this method.

```
FString UChromaSDKPluginBPLibrary::StreamGetId(const FString& shortcode);
```

StreamGetKey

Intended for Cloud Gaming Platforms, store the streaming key to persist in user preferences to continue streaming if the game is suspended or closed. shortcode: The shortcode is a null terminated string. Use the shortcode that authorized the stream to obtain the stream key. If the status is in the BROADCASTING or WATCHING state, passing a NULL shortcode will return the active streamId. streamKey should be a preallocated buffer to get the stream key. The buffer should have a length of 48. length: Length will return zero if the key could not be obtained. If the length is greater than zero, it will be the length of the returned streaming key. Retrieve the stream key after authorizing the shortcode. The authorization window will expire in 5 minutes. Be sure to save the stream key before the window expires. StreamGetStatus() should return the READY status to use this method.

```
FString UChromaSDKPluginBPLibrary::StreamGetKey(const FString& shortcode);
```

StreamGetStatusString

Convert StreamStatusType to a printable string

```
FString UChromaSDKPluginBPLibrary::StreamGetStatusString(const
EChromaSDKStreamStatusEnum::Type
    status);
```

StreamReleaseShortcode

This prevents the stream id and stream key from being obtained through the shortcode. This closes the auth window. shortcode is a null terminated string. StreamGetStatus() should return the READY status to use this method. returns success when shortcode has been released

```
bool UChromaSDKPluginBPLibrary::StreamReleaseShortcode(const FString& shortcode);
```

StreamSetFocus

The focus is a null terminated string. Set the focus identifer for the application designated to automatically change the streaming state. Returns true on success.

```
bool UChromaSDKPluginBPLibrary::StreamSetFocus(const FString& streamFocus);
```

StreamWatch

Begin watching the Chroma RGB data using streamID parameter. streamId is a null terminated string. StreamGetStatus() should return the READY status to use this method.

```
void UChromaSDKPluginBPLibrary::StreamWatch(const FString& streamId, int32
    timestamp);
```

StreamWatchEnd

End watching Chroma RGB data stream. StreamGetStatus() should return the WATCHING status to use this method.

```
void UChromaSDKPluginBPLibrary::StreamWatchEnd();
```

SubtractNonZeroAllKeys

Subtract the source color from the target color for the frame where the target color is not black. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroAllKeys(int32 sourceAnimationId,
    int32 targetAnimationId, int32 frameId);
```

SubtractNonZeroAllKeysAllFrames

Subtract the source color from the target color for all frames where the target color is not black. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroAllKeysAllFrames(int32
sourceAnimationId,
    int32 targetAnimationId);
```

SubtractNonZeroAllKeysAllFramesName

Subtract the source color from the target color for all frames where the target color is not black. Source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroAllKeysAllFramesName(const
    FString& sourceAnimationName, const FString& targetAnimationName);
```

SubtractNonZeroAllKeysAllFramesOffset

Subtract the source color from the target color for all frames where the target color is not black starting at offset for the length of the source. Source and target are referenced by id.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroAllKeysAllFramesOffset(int32
    sourceAnimationId, int32 targetAnimationId, int32 offset);
```

SubtractNonZeroAllKeysAllFramesOffsetName

Subtract the source color from the target color for all frames where the target color is not black starting at offset for the length of the source. Source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroAllKeysAllFramesOffsetName(const
    FString& sourceAnimationName, const FString& targetAnimationName, int32
    offset);
```

SubtractNonZeroAllKeysName

Subtract the source color from the target color for the frame where the target color is not black. Source and target are referenced by name.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroAllKeysName(const FString&
    sourceAnimationName, const FString& targetAnimationName, int32 frameId);
```

SubtractNonZeroTargetAllKeysAllFrames

Subtract the source color from the target color where the target color is not black for all frames. Reference source and target by id.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroTargetAllKeysAllFrames(int32
    sourceAnimationId, int32 targetAnimationId);
```

SubtractNonZeroTargetAllKeysAllFramesName

Subtract the source color from the target color where the target color is not black for all frames. Reference source and target by name.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroTargetAllKeysAllFramesName(const
    FString& sourceAnimationName, const FString& targetAnimationName);
```

SubtractNonZeroTargetAllKeysAllFramesOffset

Subtract the source color from the target color where the target color is not black for all frames starting at the target offset for the length of the source. Reference source and target by id.

```
void UChromaSDKPluginBPLibrary::SubtractNonZeroTargetAllKeysAllFramesOffset(int32
    sourceAnimationId, int32 targetAnimationId, int32 offset);
```

SubtractNonZeroTargetAllKeysAllFramesOffsetName

Subtract the source color from the target color where the target color is not black for all frames starting at the target offset for the length of the source. Reference source and target by name.

```
void
UChromaSDKPluginBPLibrary::SubtractNonZeroTargetAllKeysAllFramesOffsetName(const
    FString& sourceAnimationName, const FString& targetAnimationName, int32
    offset);
```

TrimEndFrames

Trim the end of the animation. The length of the animation will be the lastFrameId plus one. Reference the animation by id.

```
void UChromaSDKPluginBPLibrary::TrimEndFrames(int32 animationId, int32
lastFrameId);
```

TrimEndFramesName

Trim the end of the animation. The length of the animation will be the lastFrameId plus one. Reference the animation by name.

```
void UChromaSDKPluginBPLibrary::TrimEndFramesName(const FString& animationName,
    int32 lastFrameId);
```

TrimFrame

Remove the frame from the animation. Reference animation by id.

```
void UChromaSDKPluginBPLibrary::TrimFrame(int32 animationId, int32 frameId);
```

TrimFrameName

Remove the frame from the animation. Reference animation by name.

```
void UChromaSDKPluginBPLibrary::TrimFrameName(const FString& animationName,
    int32 frameId);
```

TrimStartFrames

Trim the start of the animation starting at frame 0 for the number of frames. Reference the animation by id.

```
void UChromaSDKPluginBPLibrary::TrimStartFrames(int32 animationId, int32
    numberOfFrames);
```

TrimStartFramesName

Trim the start of the animation starting at frame 0 for the number of frames. Reference the animation by name.

```
void UChromaSDKPluginBPLibrary::TrimStartFramesName(const FString& animationName,
    int32 numberOfFrames);
```

```

```

UnloadAnimation

Unloads `Chroma` effects to free up resources. Returns the animation id upon success. Returns negative one upon failure. Reference the animation by id.

```
void UChromaSDKPluginBPLibrary::UnloadAnimation(const int32 animationId);
```

UnloadAnimationName

Unload the animation effects. Reference the animation by name.

```
void UChromaSDKPluginBPLibrary::UnloadAnimationName(const FString& animationName);
```

UseForwardChromaEvents

On by default, `UseForwardChromaEvents` sends the animation name to `CoreSetEventName` automatically when `PlayAnimationName` is called.

```
void UChromaSDKPluginBPLibrary::UseForwardChromaEvents(bool toggle);
```

UseIdleAnimation

When the idle animation flag is true, when no other animations are playing, the idle animation will be used. The idle animation will not be affected by the API calls to PluginIsPlaying, PluginStopAnimationType, PluginGetPlayingAnimationId, and PluginGetPlayingAnimationCount. Then the idle animation flag is false, the idle animation is disabled. `Device` uses `EChromaSDKDeviceEnum` enums.

```
void UChromaSDKPluginBPLibrary::UseIdleAnimation(EChromaSDKDeviceEnum::Type
    device, bool flag);
```

UseIdleAnimations

Set idle animation flag for all devices.

```
void UChromaSDKPluginBPLibrary::UseIdleAnimations(bool flag);
```

UsePreloading

Set preloading animation flag, which is set to true by default. Reference animation by id.

```
void UChromaSDKPluginBPLibrary::UsePreloading(int32 animationId, bool flag);
```

UsePreloadingName

Set preloading animation flag, which is set to true by default. Reference animation by name.

```
void UChromaSDKPluginBPLibrary::UsePreloadingName(const FString& animationName,
    bool flag);
```
