WebUI Plugin

The WebUI plugin supports Windows, Mac, Linux, Android (4.19) and iOS (4.21). Also starting with version **4.19** you must ==download and install the **JsonLibrary**== plugin since it is required.

 4.16   4.19   4.21

EPIC GAMES DESTROYED DESKTOP PERFORMANCE IN VERSIONS 4.24 TO 4.26
(see table of contents)

DOWNLOAD

This plugin can be downloaded from GitHub at the following address:

https://github.com/tracerinteractive/UnrealEngine/releases

4.19.0   Latest release

🏷 4.19.0 -○- 8e4560b

▪ tracerinteractive released this 9 hours ago

∨ Assets 5

📦 HttpLibrary-4.19.zip	20.4 MB
📦 JsonLibrary-4.19.zip	19.3 MB
📦 WebUI-4.19.zip	17.9 MB

Each version is also compatible with all minor engine updates which means the 4.19.0 version of the plugin will work with any corresponding hotfix such as 4.19.1 or 4.19.2 as well.

You must have a GitHub account linked to your Epic Games account!
Setup Instructions: https://www.unrealengine.com/ue4-on-github

Otherwise you will receive the previous 404 error if you are not signed in with a linked account.
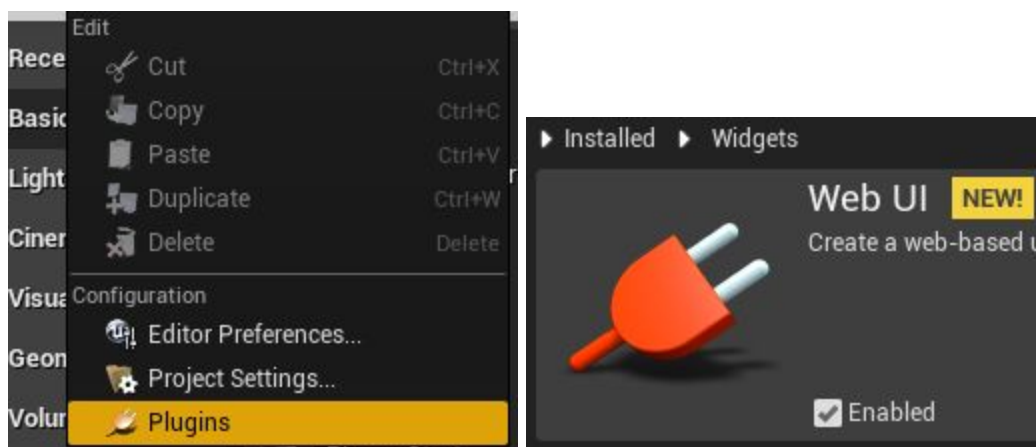
INSTALL

To install the WebUI plugin extract the downloaded files to the following engine folder:

This PC > Local Disk (C:) > Program Files > Epic Games > UE_4.19 > Engine > Plugins > Runtime

Name	Date modified	Type	Size
Synthesis	3/14/2018 5:34 PM	File folder	
WebBrowserWidget	3/14/2018 5:34 PM	File folder	
WebUI	3/14/2018 5:46 PM	File folder	
WindowsDeviceProfileSelector	3/14/2018 5:34 PM	File folder	

Also take note of the **UE_4.19** directory in the screenshots. You need to change this folder to the version that corresponds to the plugin version that was downloaded. *If you did not install your engine to the default directory then navigate to your custom installation folder instead.*

am Files > Epic Games > UE_4.19 > Engine > Plugins > Runtime > WebUI

Name	Date modified	Type
Binaries	3/14/2018 5:46 PM	File folder
Intermediate	3/14/2018 5:46 PM	File folder
Source	3/14/2018 5:46 PM	File folder
WebUI.uplugin	3/14/2018 5:46 PM	UPLUGIN File

Then open your project and go to the "Plugins" option in the edit drop-down. Click on the "Widgets" category and enable the WebUI plugin if it is not already enabled.
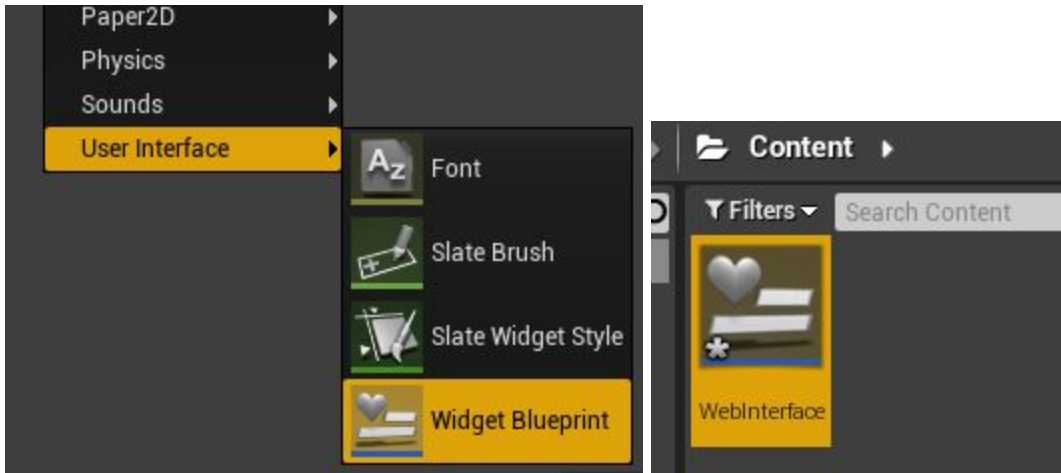
You have now successfully installed the WebUI plugin. Restart the editor to continue.
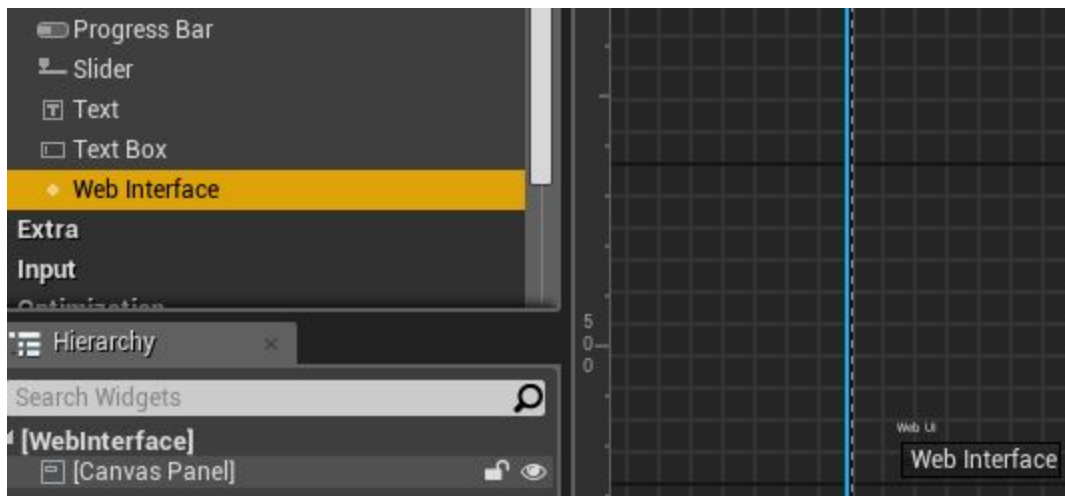
TABLE OF CONTENTS

SETUP

Once the WebUI plugin is installed and enabled start by creating a custom user widget. In this example we'll create one called **WebInterface**.


Now open the **WebInterface** blueprint to begin editing. Drag and drop a _Web Interface_ component into the canvas panel.

Select the *Web UI* component in the canvas panel and set a variable name. In this example we'll use the name **Browser** for this component.
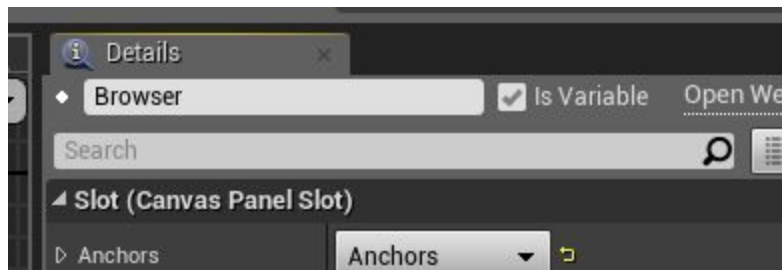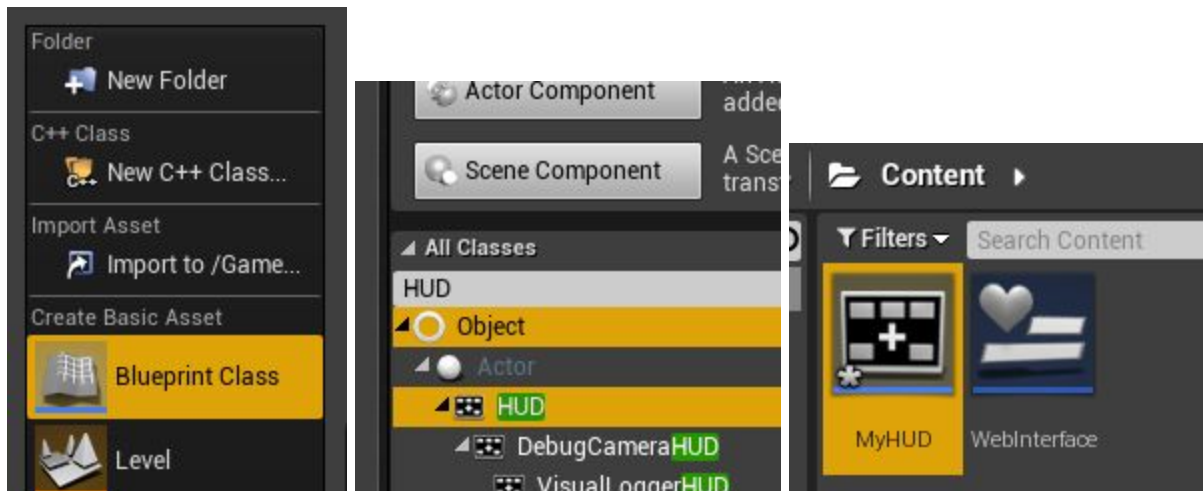

Next click the anchors drop-down and select the "snap to all edges" option in the bottom right and then set all the offsets to zero.
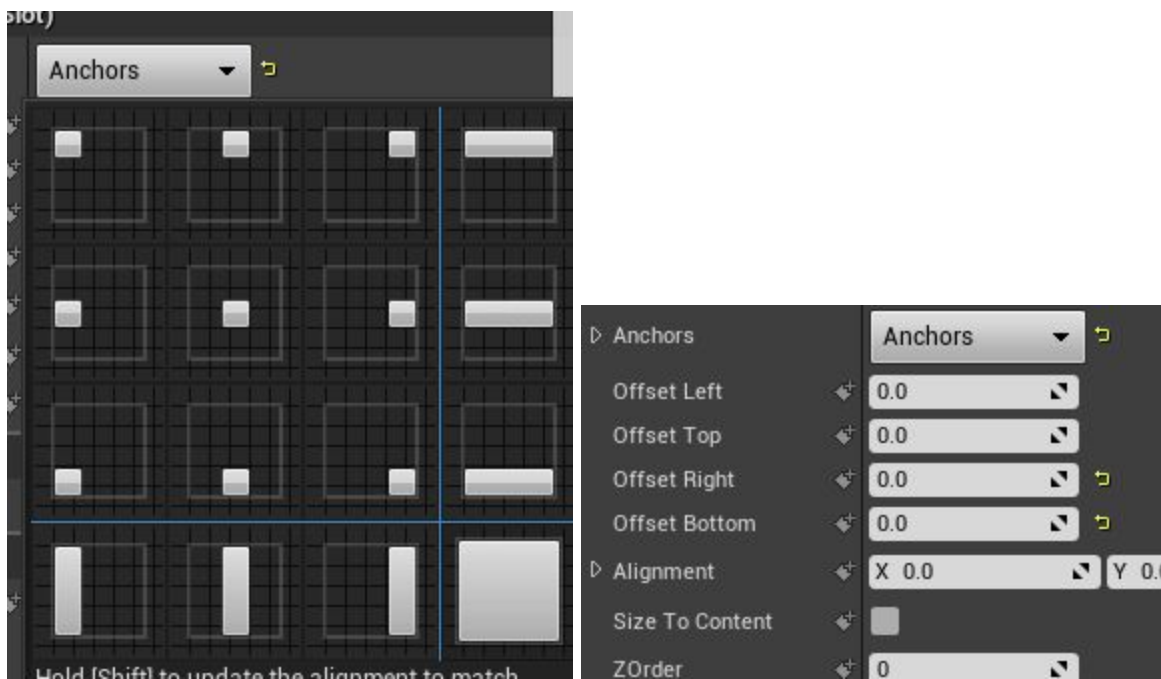

The *Web UI* component should now be full screen. Click the *Compile* and *Save* buttons and then close this blueprint.

Create a new blueprint class and pick the parent class. In this example we will select the _HUD_ class since it's the most appropriate and use the name **MyHUD** for this asset. _Note that a widget can be added to the viewport from any blueprint so you could use an existing blueprint instead._


Now open the **MyHUD** blueprint to begin editing and click the "Event Graph" tab. Drag an execution line from the _BeginPlay_ event and select the "Create Widget" node. Then click the "Select Class" drop-down and select the **WebInterface** widget.
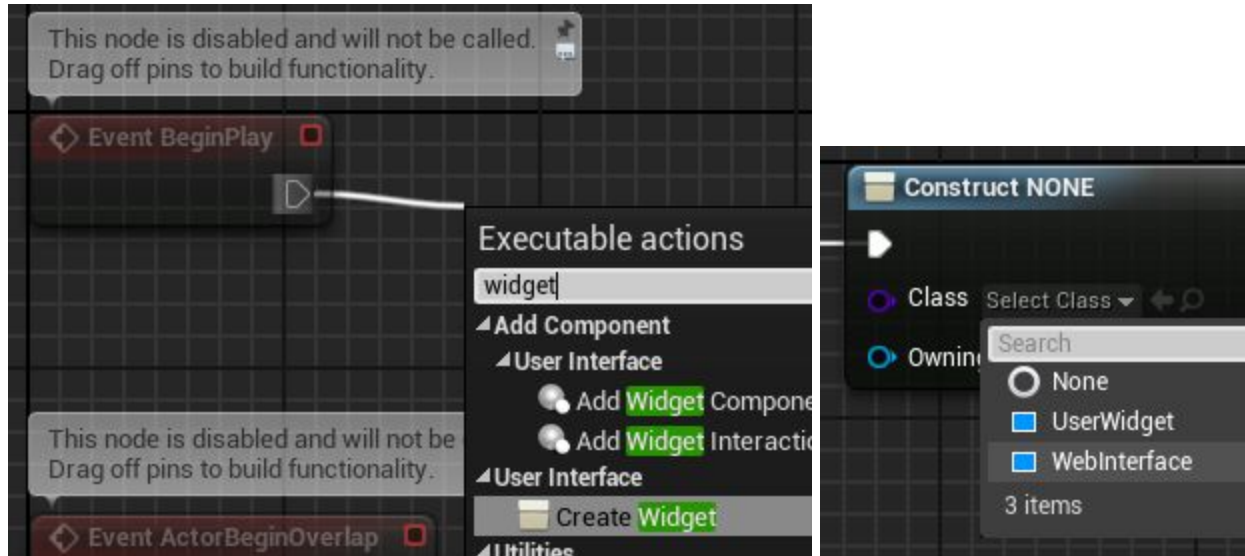
Next drag a connection from the "Owning Player" pin and select the *Get Owning Player Controller* node.
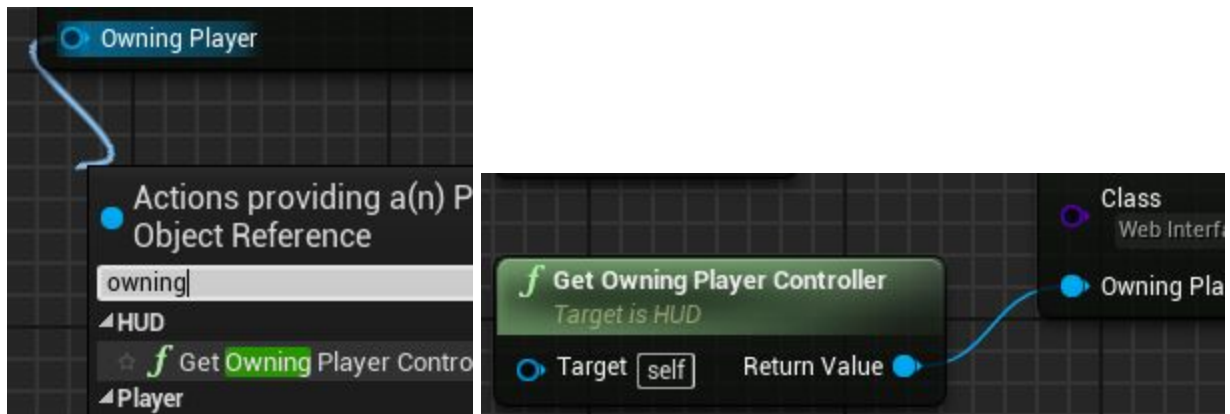
Then drag another connection from the "Return Value" pin and select the "Promote to variable" option in the drop-down. In this example we'll use the name **MyWidget** for this variable.
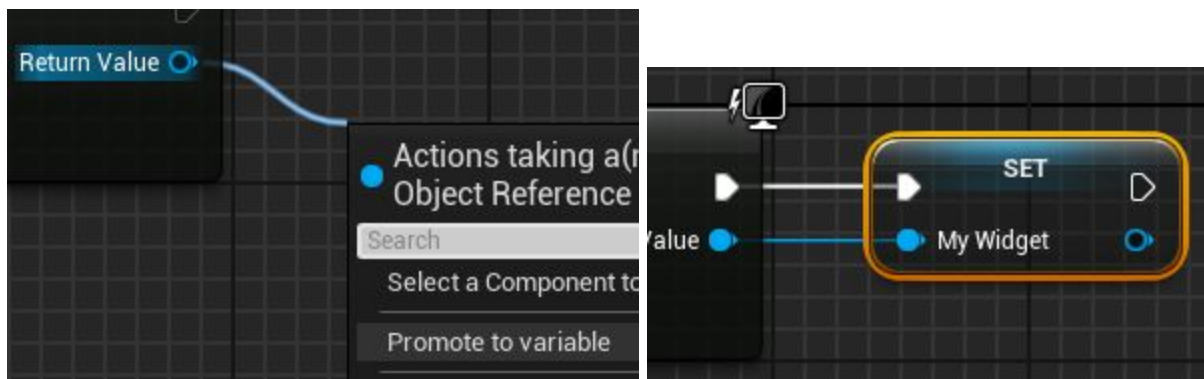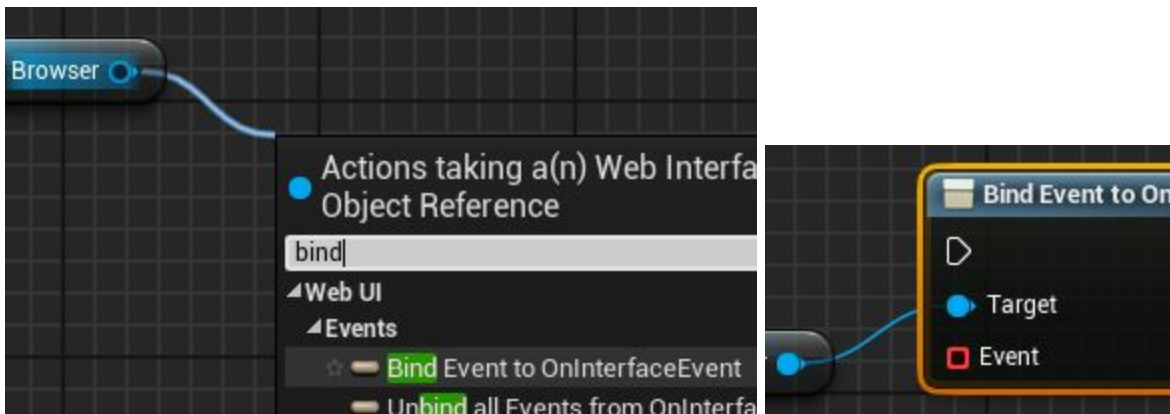

Now read the value of the **Browser** variable by dragging a connection from the **MyWidget** pin.
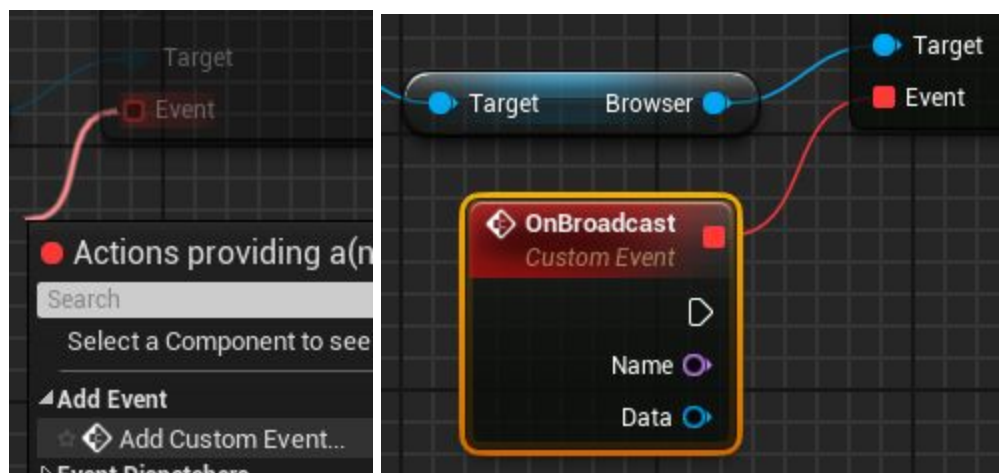
Next drag a connection from the **Browser** variable and select the "Bind Event to OnInterfaceEvent" node.

Also be sure to connect the execution pin from the **MyWidget** node to this node.

Now drag a delegate connection from the "Event" pin and select "Add Custom Event.." from the drop-down. In this example we'll use the name **OnBroadcast** for this event.
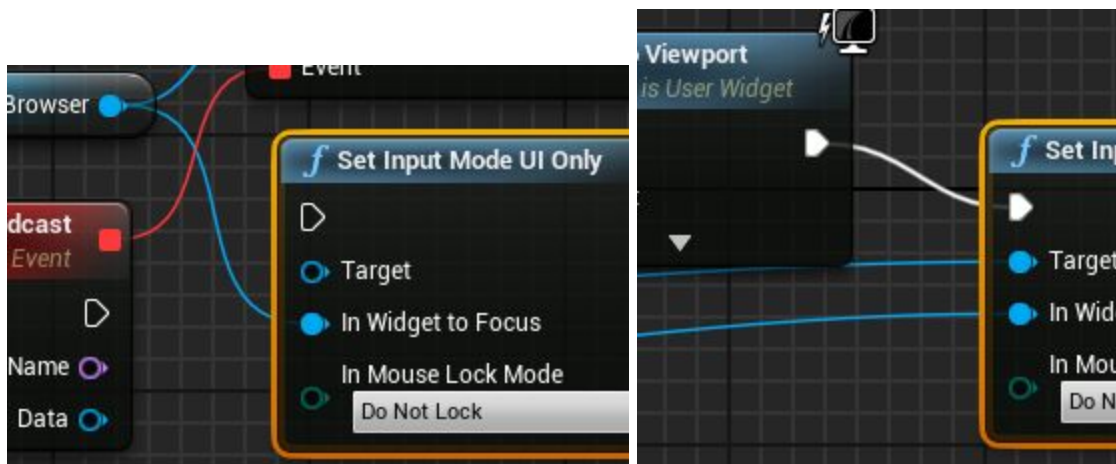
Then drag another connection from the **MyWidget** variable and select the "Add to Viewport" node in the drop-down.
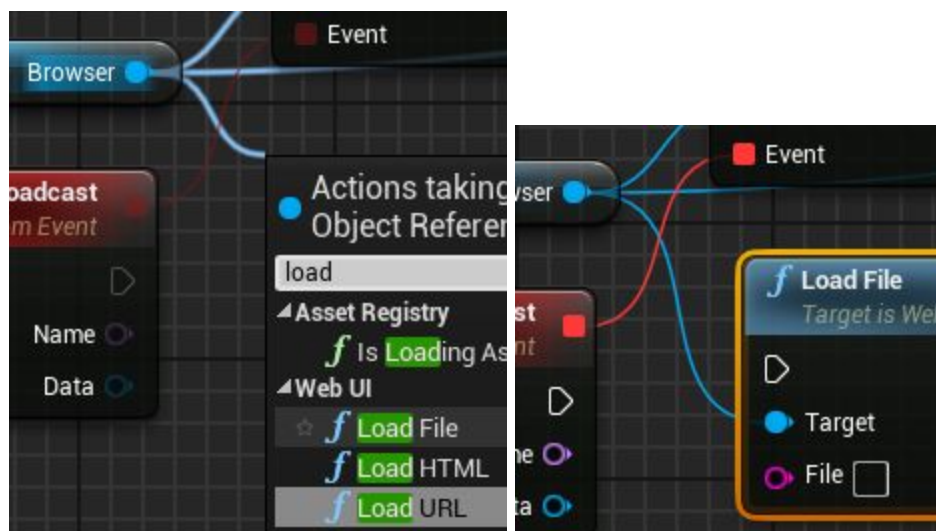

Move this node to the right and connect it to the *Bind Event to OnInterfaceEvent* node. Now drag a connection from the **Browser** variable and select the "Set Input Mode UI Only" node.

Move this node to the right as well and connect it to the *Add to Viewport* node. Then connect the "Target" pin to the *Get Owning Player Controller* node previously created.
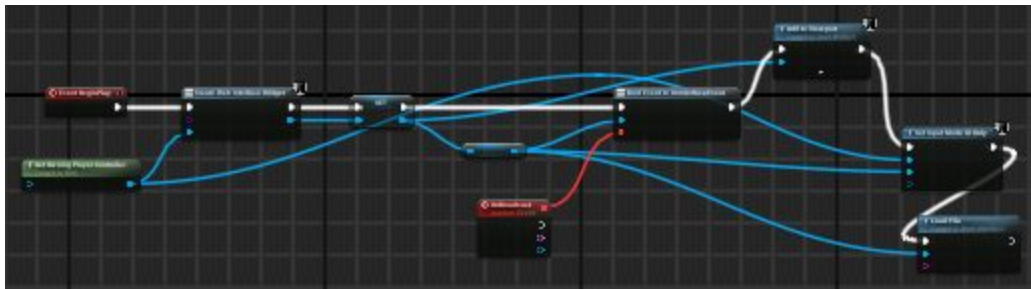

Now drag another connection from the **Browser** variable and select the "Load File" node.

Move this node to the right and connect it to the *Set Input Mode UI Only* node. Then enter a filename from the **/UI** directory. Try using the *sample.html* file from the example project:

You should now have a blueprint that looks similar to the following screenshot:


This is the baseline functionality required for the **WebInterface** widget. Click the *Compile* and *Save* buttons and then close this blueprint.

To use this in the game create another blueprint class and select the "Game Mode Base" class as the parent. In this example we'll use the name **MyGameMode** for this blueprint. Then select your HUD in the details section.
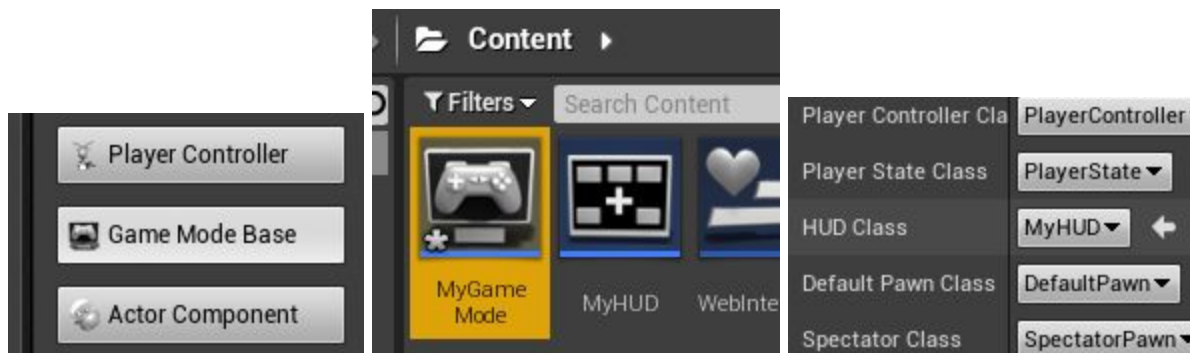

Click the *Compile* and *Save* buttons and then close this blueprint. *If you have an existing level in which you'd like to load your interface, skip the following section and use your own custom level instead. You could also set this game mode as the default game mode in your project settings as shown below:*
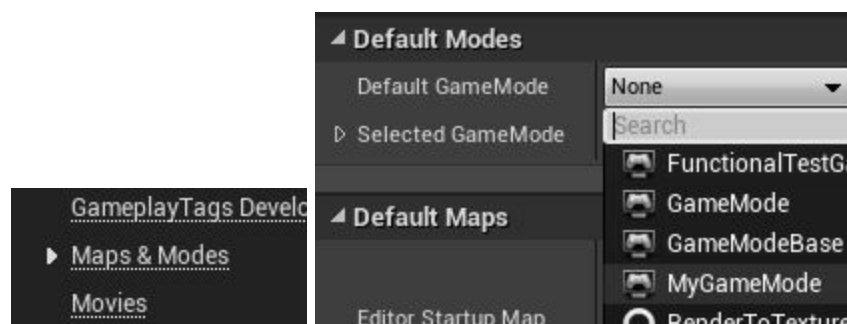
Now create a level asset to load your interface. In this example we'll use the name **MyMap** for this level. Once created double click on the level to open it in the editor.
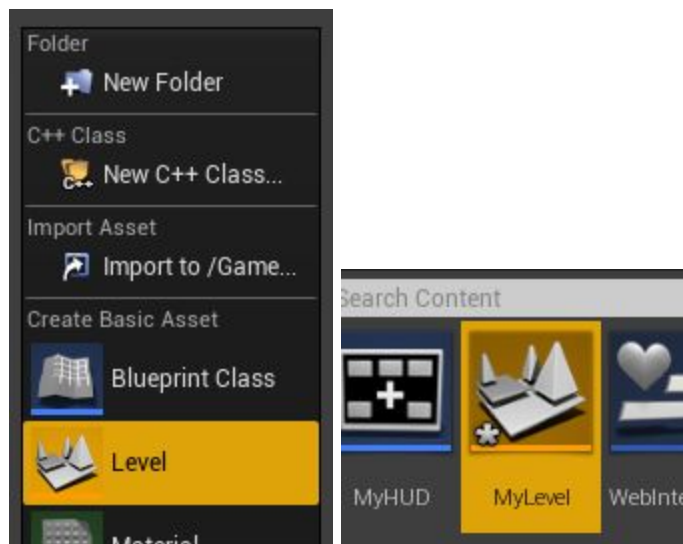
In the "World Settings" tab select **MyGameMode** from the drop-down under _Game Mode Override_. Then click "Save Current" in the top left to save your map. Now your WebUI is ready for testing, just click the "Play" button to begin!
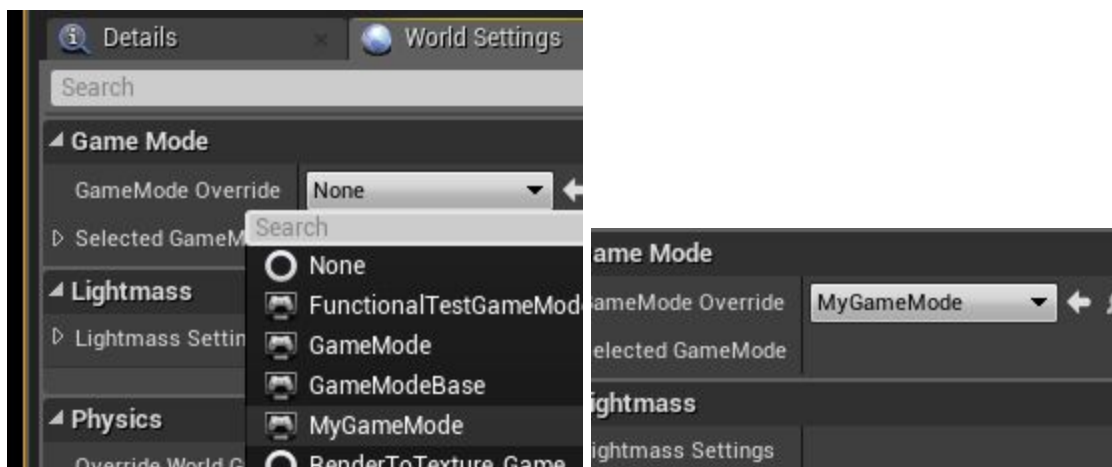
DATA

The browser can send data to the game as JSON using a custom blueprint event. Start by dragging a connection from the "Name" pin of the **OnBroadcast** event previously created and selecting the "Switch on Name" node.


Click the "Add pin +" button and with the node selected uncheck the *Has Default Pin* option in the details panel on the right. Type a name for the pin based on the desired functionality and add more as necessary. In this example **print** will be used to debug the "Data" pin. Drag a connection from this pin to traverse and access objects, arrays, and primitive data types sent from JavaScript.
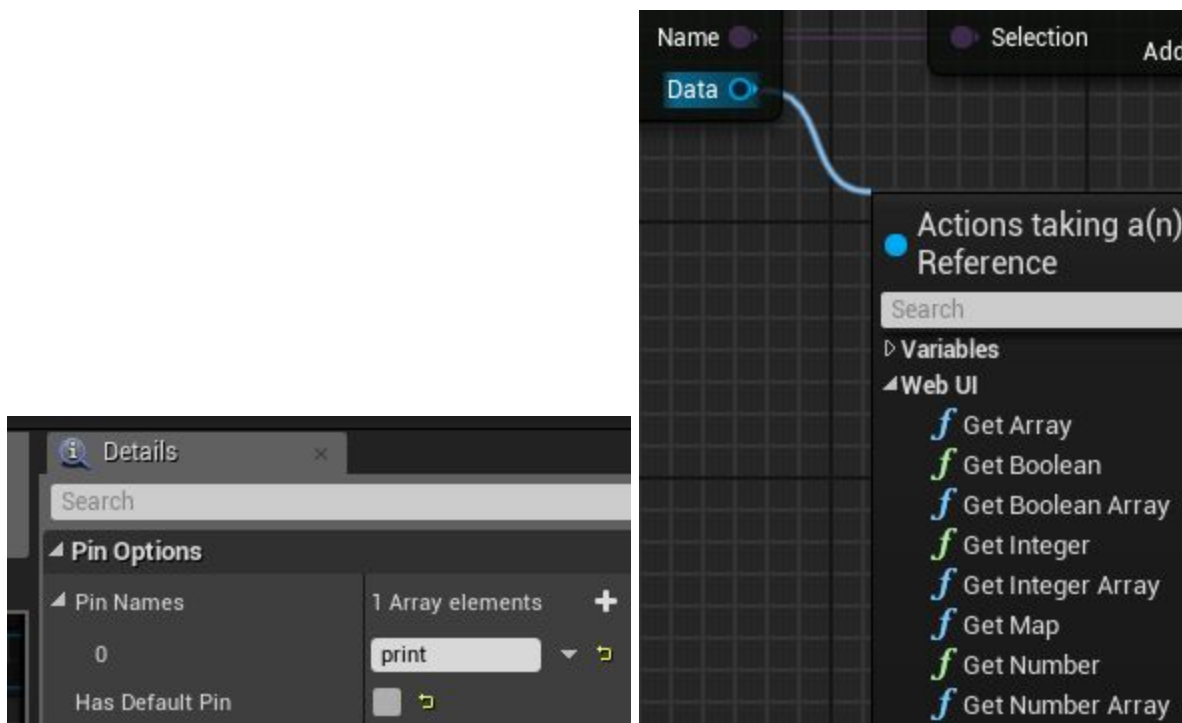
After selecting the "Stringify" node and printing the "Return Value" pin your **OnBroadcast** event should look similar to the following screenshot:
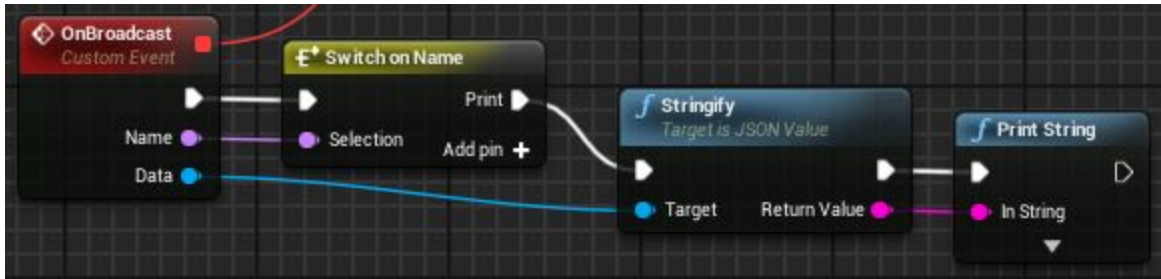
This event is triggered by calling the `ue.interface.broadcast` function in JavaScript. The first argument is the "Name" of the event and must be a string. The second argument is provided through the "Data" pin and must be a valid JSON string. A global `ue4()` helper function is defined to automatically *JSON.stringify(...)* the second argument.

```javascript
jQuery(function()
{
    $("#debugButton").click(function(e)
    {
        var posX = $(this).position().left,
            posY = $(this).position().top;

        // transmit data to the game
        ue4("print", { "posX": posX, "posY": posY });
    });
});
```

This script should be used to define the global `ue4()` helper function on page load and is <mark>required for mobile support</mark>. *The source code for this script is provided on the following page*.

```javascript
"object"!=typeof ue||"object"!=typeof ue.interface?("object"!=typeof ue&&(ue={}),
ue.interface={},ue.interface.broadcast=function(e,t){if("string"==typeof e){
var o=[e,""];void 0!==t&&(o[1]=t);var n=encodeURIComponent(JSON.stringify(o));
"object"==typeof history&&"function"==typeof history.pushState?(history.pushState(
{},"","#"+n),history.pushState({},"","#"+encodeURIComponent("[]"))):
(document.location.hash=n,document.location.hash=encodeURIComponent("[]"))}}):
function(e){ue.interface={},ue.interface.broadcast=function(t,o){
"string"==typeof t&&(void 0!==o?e.broadcast(t,JSON.stringify(o)):
e.broadcast(t,""))}}(ue.interface),ue4=ue.interface.broadcast;
```

The `ue.interface.broadcast` function is <mark>not available on Android or iOS</mark>. Therefore a workaround has been developed that supports mobile browsers. More details can be found in the source code of the global helper function.

The **ue4()** helper function is a wrapper that calls **ue.interface.broadcast** using a valid JSON string. An optional second argument allows commands to be triggered without providing data. This function also falls back to transmitting JSON via URL changes on mobile platforms.

```javascript
if (typeof ue != "object" || typeof ue.interface != "object")
{
    if (typeof ue != "object")
        ue = {};

    // mobile
    ue.interface = {};
    ue.interface.broadcast = function(name, data)
    {
        if (typeof name != "string")
            return;

        var args = [name, ""];
        if (typeof data != "undefined")
            args[1] = data;

        var hash = encodeURIComponent(JSON.stringify(args));
        if (typeof history == "object" && typeof history.pushState == "function")
        {
            history.pushState({}, "", "#" + hash);
            history.pushState({}, "", "#" + encodeURIComponent("[]"));
        }
        else
        {
            document.location.hash = hash;
            document.location.hash = encodeURIComponent("[]");
        }
    };
}
else
    (function(obj)
    {
        // desktop
        ue.interface = {};
        ue.interface.broadcast = function(name, data)
        {
            if (typeof name != "string")
                return;

            if (typeof data != "undefined")
                obj.broadcast(name, JSON.stringify(data));
            else
                obj.broadcast(name, "");
        };
    })(ue.interface);

// create the global ue4(...) helper function
ue4 = ue.interface.broadcast;
```
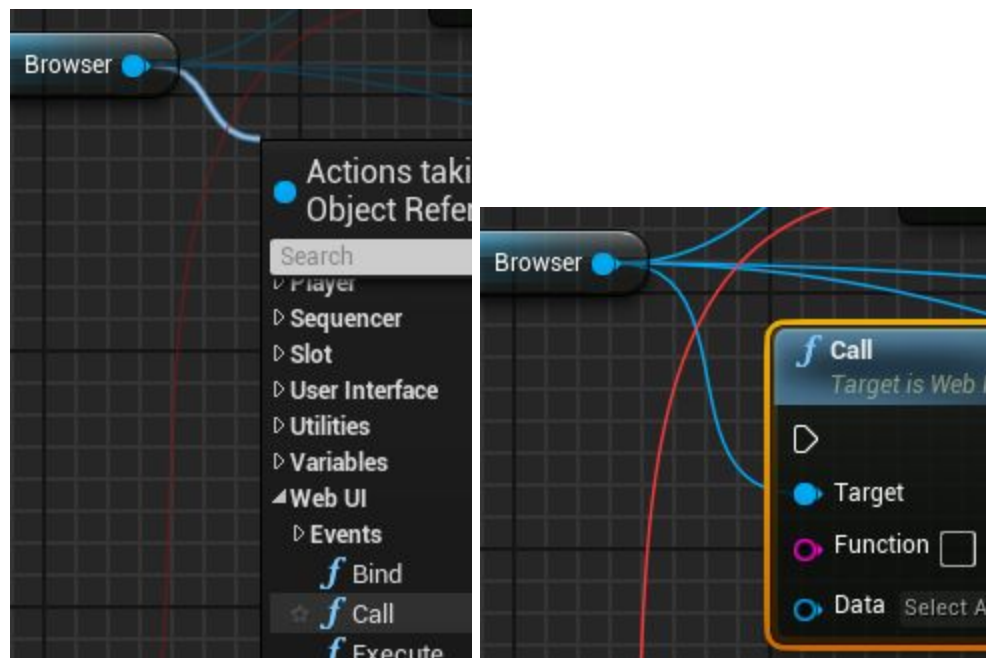
The game can also send data to the browser as JSON through a function call. Start by dragging a connection from the **Browser** variable and selecting the "Call" node.

Type the name of the function that will be executed in JavaScript. In this example the *setFPS(...)* function will be called. Now connect a JSON value to the "Data" pin. This can be a float, integer, string, boolean, or even complex types such as an array or map.


These functions must be defined in JavaScript on the global `ue.interface` object. The JSON provided to the "Data" pin is passed as the only argument to the function. It will be automatically deserialized into the appropriate data type.

```javascript
// called in-game via blueprints
ue.interface.setFPS = function(fps)
{
    // set element text
    $("#fpsMeter").text(fps.toFixed(1) + " FPS");
};
```

Using these functions developers can quickly and easily send data between the game and browser through JSON.
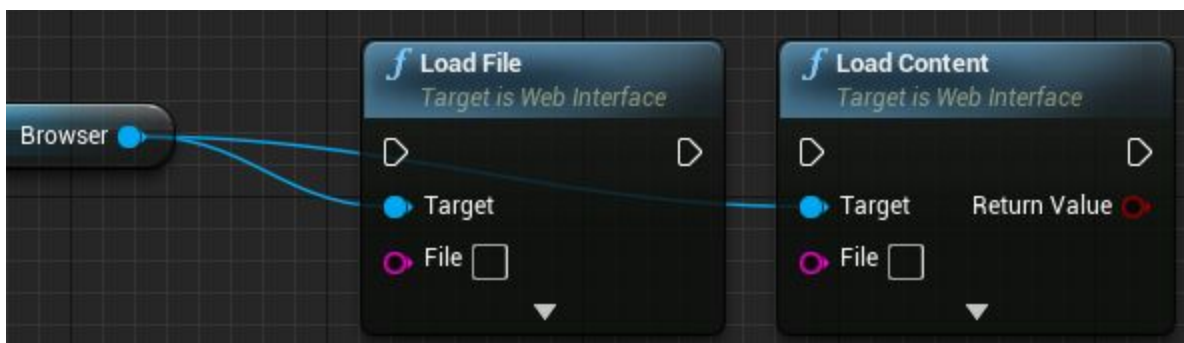
LOAD

The browser can load files or content when dragging a connection from the **Browser** variable and selecting one of the "Load" functions:
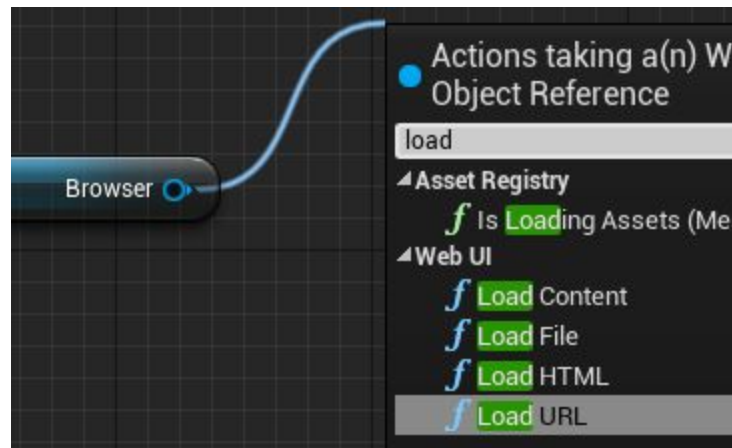

The "Load URL" and "Load HTML" nodes are pretty straightforward. You can either provide a raw HTML string or any URL which will be loaded in the browser:
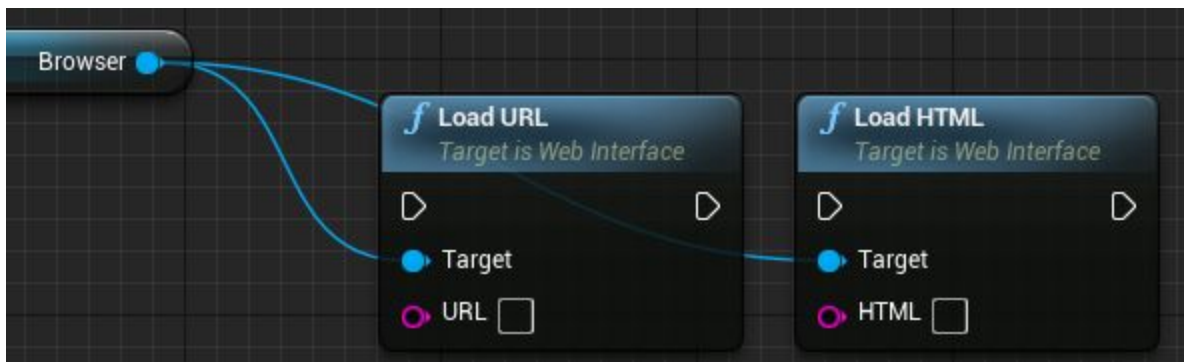

However the "Load File" and "Load Content" nodes are a bit more complicated:
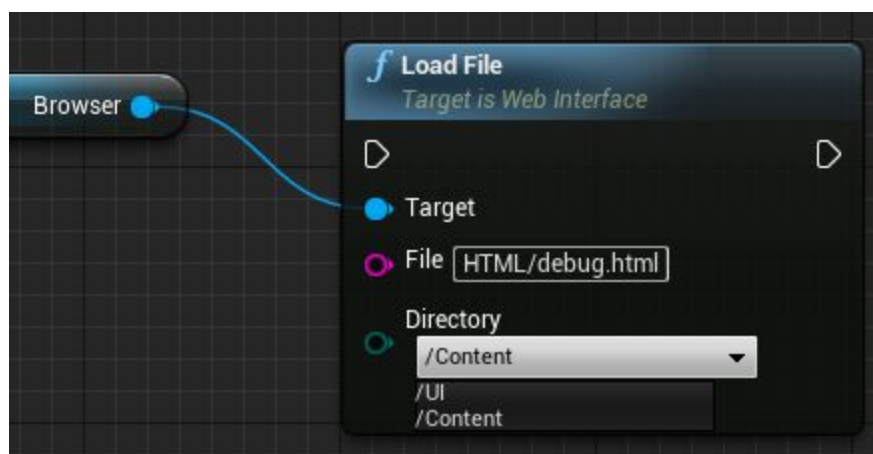
The "Load File" function is the equivalent of using $file:///$ and loads HTML files directly in the browser. By default it will load files from the **/UI** directory in the root of your project:
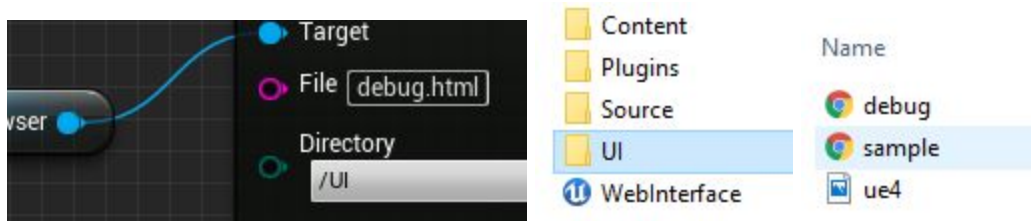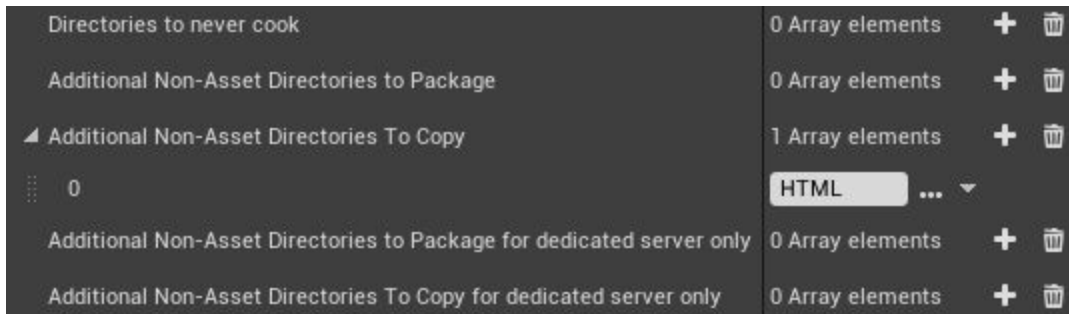

This also allows the use of relative paths in the HTML (such as `<img src="ue4.png" />`) to access images, scripts, and stylesheets. When shipping or packaging the **/UI** directory should be copied to the same level as the **/Binaries** and **/Content** folders within your game folder:
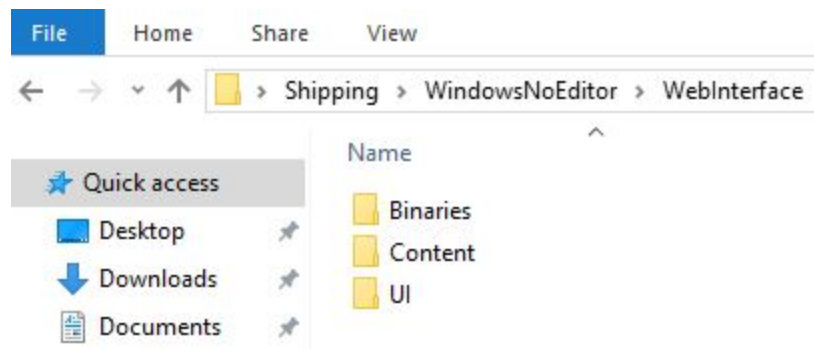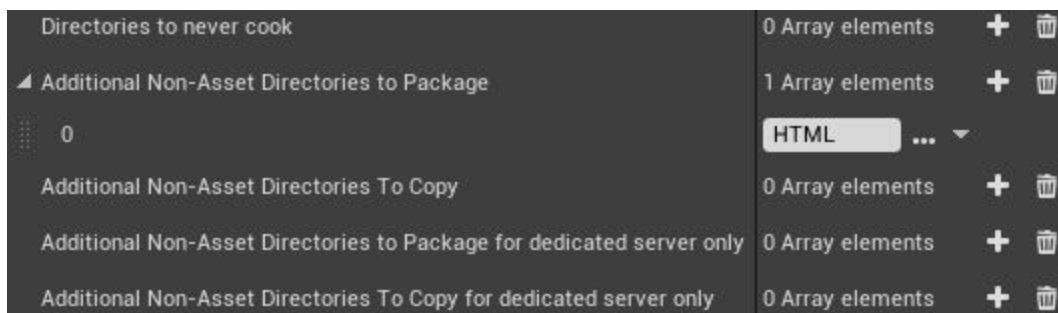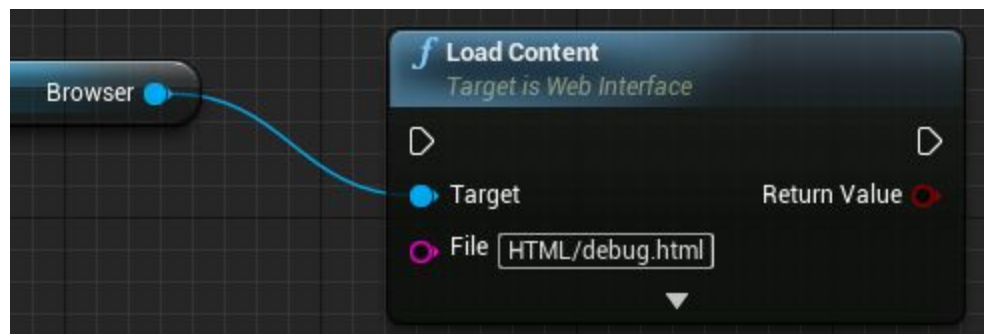

If you prefer to have HTML files inside the **/Content** directory there is an option for this as well under the advanced display. Then instead of manually copying the **/UI** folder you can use the "Additional Non-Asset Directories To Copy" option in the project settings to automatically copy a specific folder from the **/Content** directory:

Directories to never cook		0 Array elements	+	🗑
Additional Non-Asset Directories to Package		0 Array elements	+	🗑
▲ Additional Non-Asset Directories To Copy		1 Array elements	+	🗑
⁞ 0		HTML ... ▾		
Additional Non-Asset Directories to Package for dedicated server only	0 Array elements	+	🗑
Additional Non-Asset Directories To Copy for dedicated server only	0 Array elements	+	🗑

However the "Load File" function <mark>cannot access HTML files inside `.pak` files</mark> (even if they are located inside the **/Content** directory). Therefore the "Load Content" function was added to access files using the "Additional Non-Asset Directories to Package" option instead:

| Directories to never cook		0 Array elements	+	🗑
▲ Additional Non-Asset Directories to Package		1 Array elements	+	🗑
⁞ 0		HTML ... ▾		
Additional Non-Asset Directories To Copy		0 Array elements	+	🗑
Additional Non-Asset Directories to Package for dedicated server only	0 Array elements	+	🗑
Additional Non-Asset Directories To Copy for dedicated server only	0 Array elements	+	🗑

This option packages a folder from the **/Content** directory into the `.pak` file(s) of your shipped game. But since this content is not loaded using $file:///$ the HTML <mark>cannot access local files</mark> such as images, scripts, and stylesheets. However the "Load Content" function does have an option under the advanced display which allows JavaScript files from the **/Content** directory to be executed within the context of the browser:
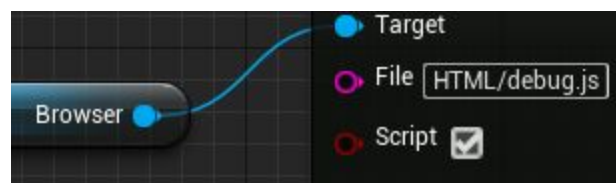
CALLBACKS

Starting with version 4.24 the `ue.interface.broadcast` function includes an optional third argument that must be a string. It specifies a function name on the `ue.interface` object that the engine can choose to call back with an optional argument. Here is an example of calling the callback from blueprints if one was provided:
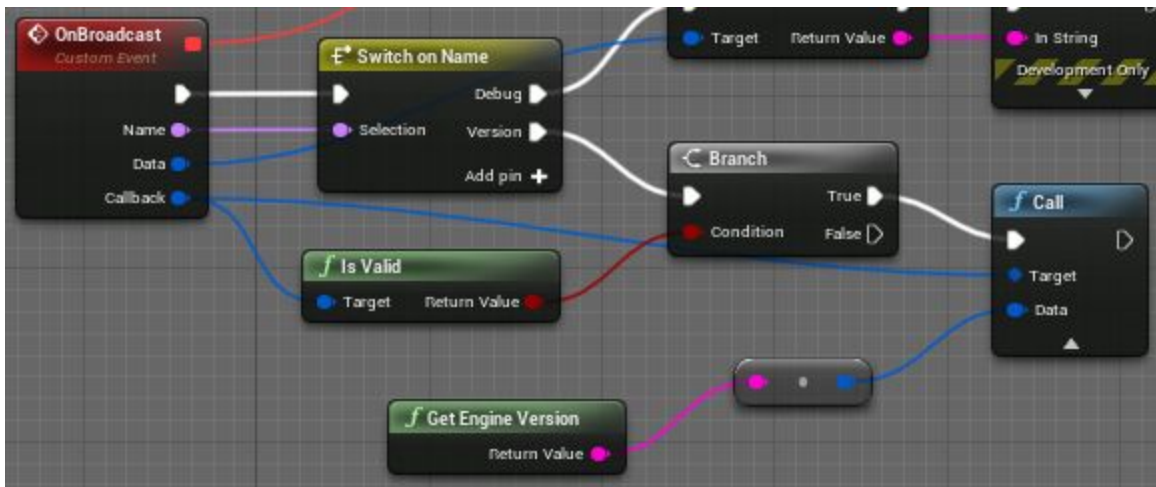
This event can be triggered from JavaScript by the global `ue4()` helper function with the callback function as either the second or third argument:

```javascript
ue4("version", function(v)
{
    if (typeof v == "string")
        document.body.innerText = "Unreal Engine " + v.split('-')[0];
});
```

An optional timeout period can be provided after the function. If no timeout is provided the default value defined in the helper function is 1 second. The temporary callback function will be automatically deleted after the timeout period.

Here is an example with input data and a callback timeout of 3 seconds:

```javascript
ue4("version", {}, function(v)
{
    if (typeof v == "string")
        document.body.innerText = "Unreal Engine " + v.split('-')[0];
}, 3);
```

This script can be used to define a global **ue4()** helper function that registers a temporary callback function with an optional timeout period. *The source code for this script is provided in the example project.*

```
"object"!=typeof ue&&(ue={}),uuidv4=function(){
return"10000000-1000-4000-8000-100000000000".replace(/[018]/g,function(t){
return(t^crypto.getRandomValues(new Uint8Array(1))[0]&15>>t/4).toString(16)})},
ue4=function(r){return"object"!=typeof ue.interface||"function"!=typeof
ue.interface.broadcast?(ue.interface={},function(t,e,n,o){var u,i;"string"==typeof
t&&("function"==typeof e&&(o=n,n=e,e=null),u=[t,"",r(n,o)],void 0!==e&&(u[1]=e),
i=encodeURIComponent(JSON.stringify(u)),"object"==typeof
history&&"function"==typeof history.pushState?(history.pushState({},"","#"+i),
history.pushState({},"","#"+encodeURIComponent("[]"))):(document.location.hash=i,
document.location.hash=encodeURIComponent("[]")))}):(i=ue.interface,
ue.interface={},function(t,e,n,o){var u;"string"==typeof t&&("function"==typeof
e&&(o=n,n=e,e=null),u=r(n,o),void 0!==e?i.broadcast(t,JSON.stringify(e),u):
i.broadcast(t,"",u))})});var i}(function(t,e){if("function"!=typeof t)return"";var
n=uuidv4();return ue.interface[n]=t,setTimeout(function(){delete ue.interface[n]},
1e3*Math.max(1,parseInt(e)||0)),n});
```

The **uuidv4** function generates a cryptographically strong UUID that serves as a temporary function name on the **ue.interface** object. This property will be automatically deleted after the designed timeout period which defaults to 1 second.
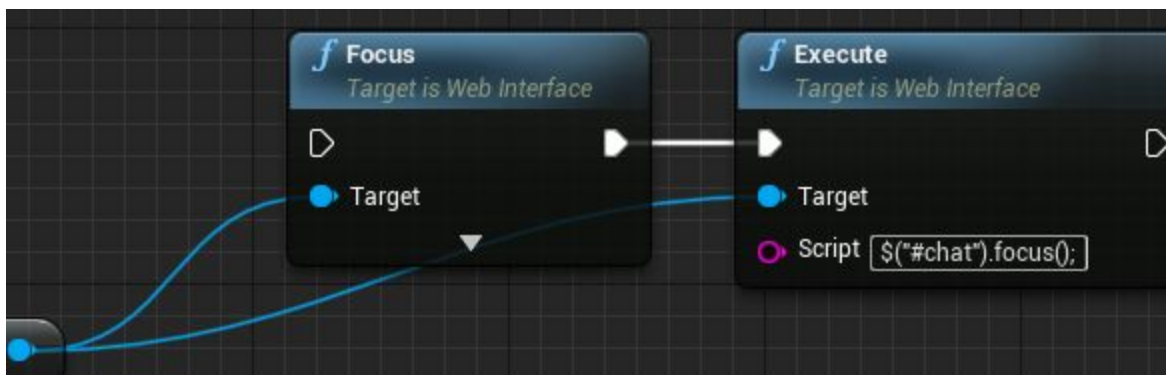
FOCUS

There are "Focus" and "Unfocus" functions available to change keyboard and mouse focus:
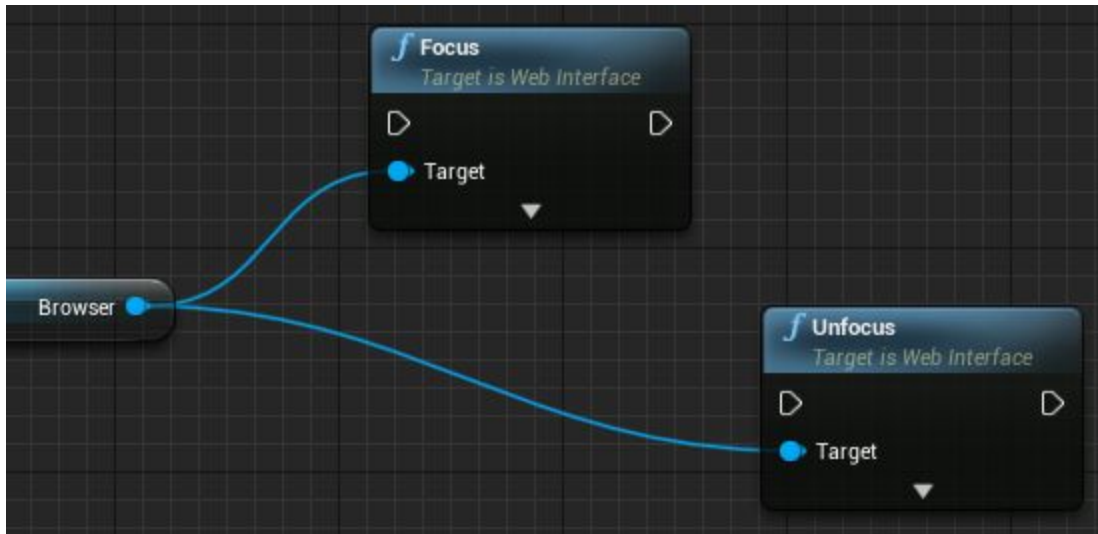

The "Focus" node provides direct focus into the viewport of the browser. This method is more advanced than the default functions provided by the engine since they require the user to click on the interface before the browser receives keyboard focus. However this function provided by the plugin allows JavaScript to focus `input` and `textbox` elements. This means you can set focus from the game directly into HTML elements (for example a chat widget) when combined with a JavaScript focus event as shown below:

This node will also automatically show the mouse cursor along with making the interface fully interactable. The "Unfocus" node will then automatically hide the mouse cursor and move focus from the browser back to the game. These two nodes are commonly used for switching between the game and any in-game settings (usually via the escape key in most games).

Another node called "Reset Mouse Position" is provided to move the mouse (when it is visible) to the center of the screen. This is commonly called after the "Focus" node and is recommended if the mouse was not initially visible when setting focus to the interface:
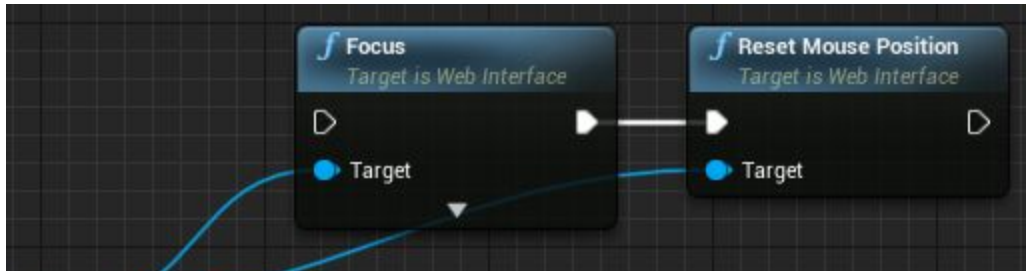
TRANSPARENCY

The widget can be configured to support clicking behind the transparent parts of the interface for games that require the mouse cursor to always be visible. This setting is available in the widget blueprint under the "Behavior" section:
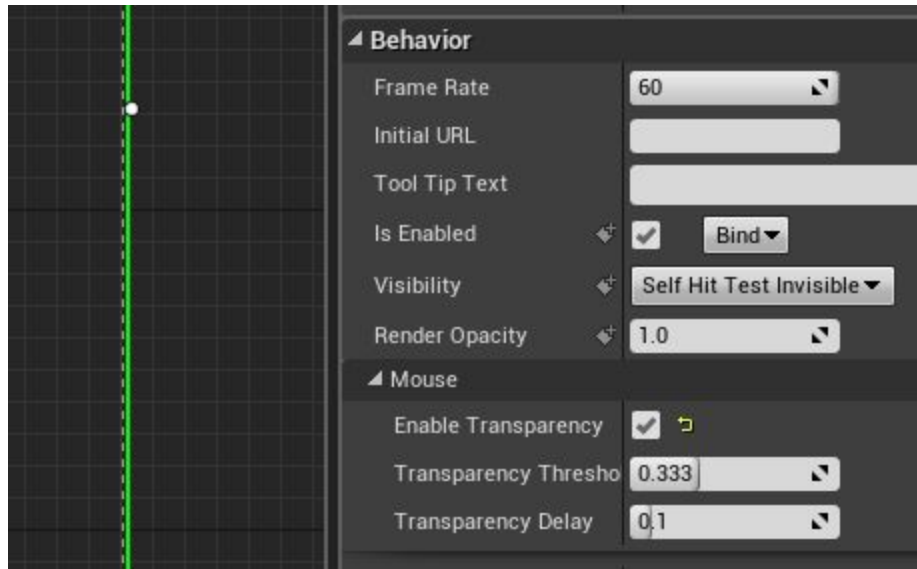

This directs the underlying slate widget to sample the pixel under the mouse position on each game tick and dynamically toggles the widget between *Hit Test Invisible* and *Visible* modes.

It is also recommended to enable mouse click/over events in your game when utilizing this functionality since *ActorOnClicked* and *ActorBeginCursorOver* or *ActorEndCursorOver* events are automatically built-in to actor blueprints:

These events are already included with the engine and you can find them in the *Mouse Input* section under *Add Event* when right-clicking in the event graph of any actor blueprint:
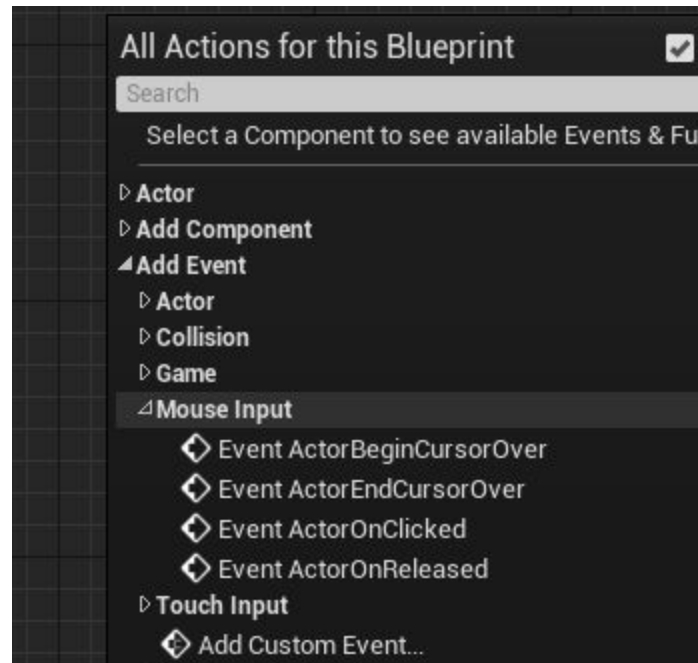
Note that you must *Enable Click Events* or *Enable Mouse Over Events* in the "Mouse Interface" section of your player controller for these events to trigger in the engine:
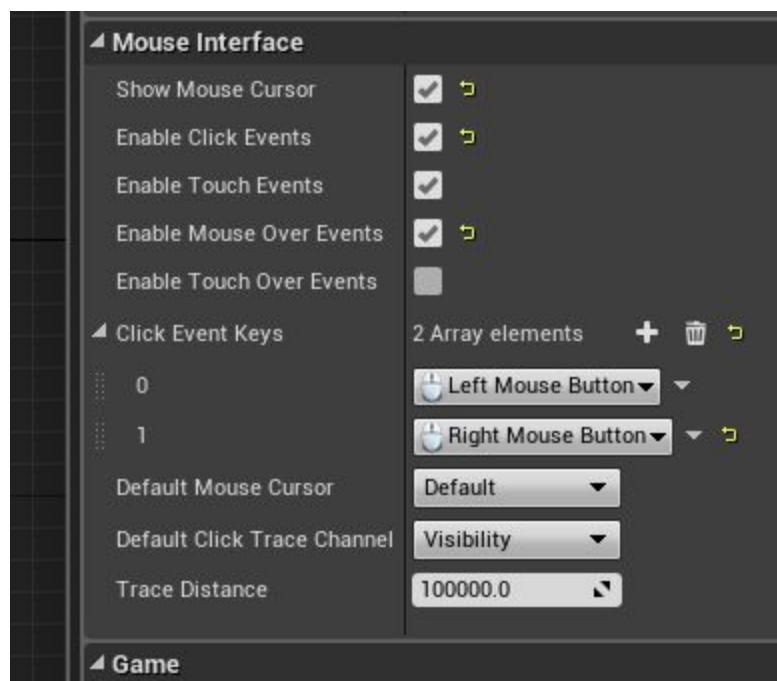
CURSOR

This plugin does not support custom mouse cursors due to a bug in the engine. However if you manually compiled from source the following changes can be implemented.

To fix this issue with custom mouse cursors find the *CEFWebBrowserWindow.cpp* file in the */Engine/Source/Runtime/WebBrowser/Private/CEF* directory of the engine source. Perform a search for the method **FCEFWebBrowserWindow::OnCursorChange(...)** shown below:

```
1695    void FCEFWebBrowserWindow::OnCursorChange(CefCursorHandle CefCursor,
1696    {
1697        switch (Type) {
1698            // Map the basic 3 cursor types directly to Slate types on a
1699            case CT_NONE:
1700                Cursor = EMouseCursor::None;
1701                break;
1702            case CT_POINTER:
1703                Cursor = EMouseCursor::Default;
1704                break;
1705            case CT_IBEAM:
1706                Cursor = EMouseCursor::TextEditBeam;
1707                break;
1708            #if PLATFORM_WINDOWS || PLATFORM_MAC
1709            // Platform specific support for native cursor types
1710            default:
1711                {
1712                    TSharedPtr<ICursor> PlatformCursor = FSlateApplicatio
1713
1714                    if (PlatformCursor.IsValid())
1715                    {
1716                        PlatformCursor->SetTypeShape(EMouseCursor::Custor
1717                        Cursor = EMouseCursor::Custom;
1718                    }
1719                }
1720                break;
1721            #else
1722            // Map to closest Slate equivalent on platforms where native
1723            case CT_VERTICALTEXT:
1724                Cursor = EMouseCursor::TextEditBeam;
1725                break;
```

Note the `#if PLATFORM_WINDOWS || PLATFORM_MAC` on line 1708. It essentially bypasses the native cursor management of the engine by manually setting the cursor of the operating system unless it is the default pointer or text editor beam.

Therefore you should change this line to `#if 0` which prevents this block (with absolutely no global setting to toggle it) from being compiled. The following example demonstrates this change to the engine source:

```
1705            case CT_IBEAM:
1706                Cursor = EMouseCursor::TextEditBeam;
1707                break;
1708        #if 0
1709        // Platform specific support for native cursor types
1710        default:
1711            {
1712                TSharedPtr<ICursor> PlatformCursor = FSlateApplicatic
1713
1714                if (PlatformCursor.IsValid())
1715                {
1716                    PlatformCursor->SetTypeShape(EMouseCursor::Custor
1717                    Cursor = EMouseCursor::Custom;
1718                }
1719            }
1720            break;
1721        #else
1722        // Map to closest Slate equivalent on platforms where native
1723        case CT_VERTICALTEXT:
1724            Cursor = EMouseCursor::TextEditBeam;
1725            break;
1726        case CT_EASTRESIZE:
1727        case CT_WESTRESIZE:
1728        case CT_EASTWESTRESIZE:
1729        case CT_COLUMNRESIZE:
1730            Cursor = EMouseCursor::ResizeLeftRight;
1731            break;
```

Note the block between lines 1709 and 1720 is now grayed out and the cursor of the web browser process will be appropriately mapped to the engine cursor types.

You can now build the UE4 project to use custom mouse cursors with the WebUI plugin:
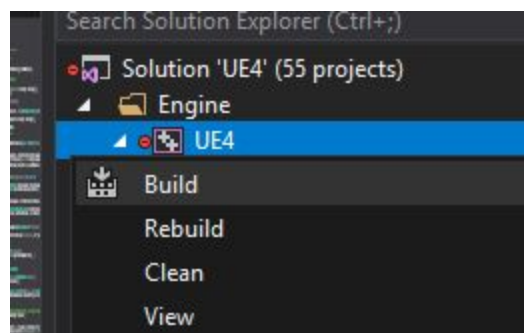
TEXTURE

The texture data of the browser can be accessed using the "Read Texture Pixel" node or the "Read Texture Pixels" node as shown in the following example:
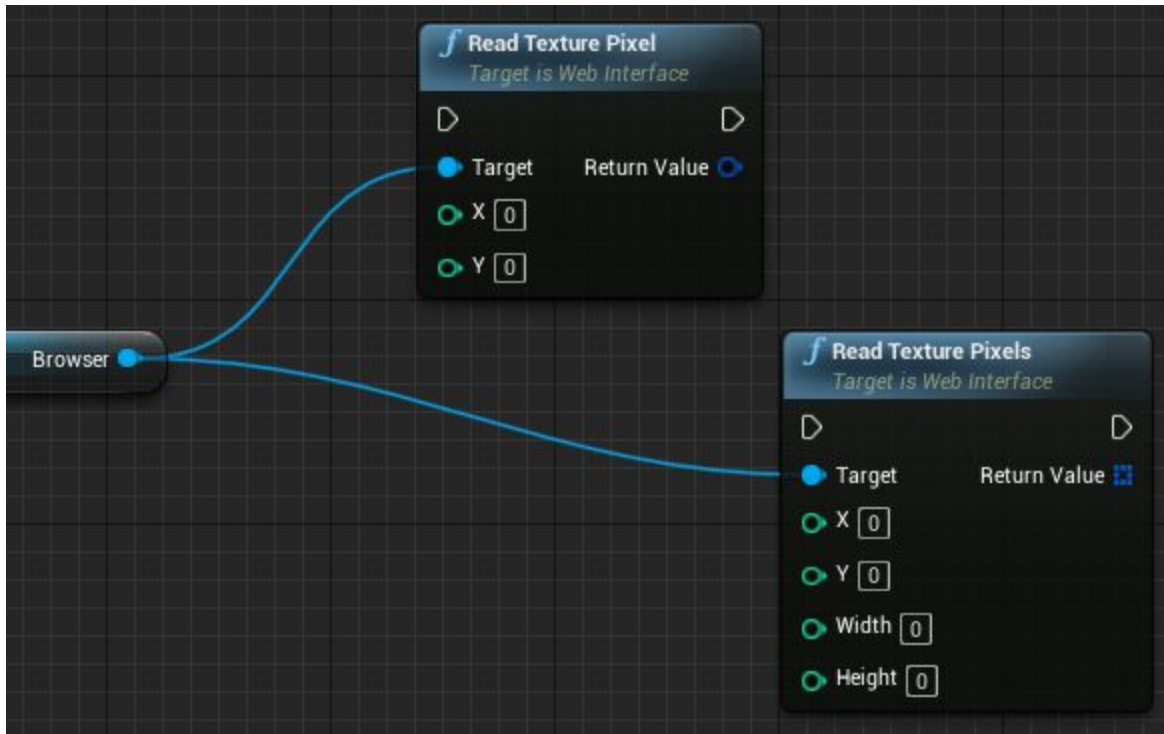
These nodes will return either a single color or an array of colors, respectively. Note that there is no render target created to access this texture data. Instead these functions directly execute commands on the render thread using the underlying texture reference of the browser.

Also keep in mind that the size of the browser texture may not always match the size of the widget in the viewport. Therefore the "Get Texture Width" and "Get Texture Height" nodes are provided to access the size of the underlying browser texture:
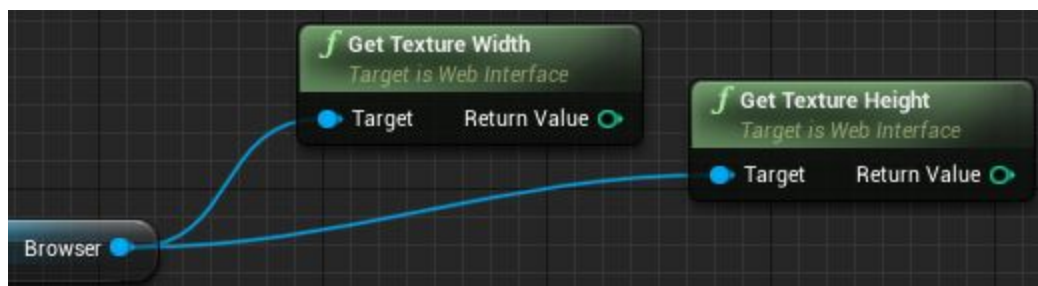
TOOLS

The browser can be debugged by using CTRL+SHIFT+I when focused. This key combination opens the Chrome DevTools and is only available in development and debug builds.
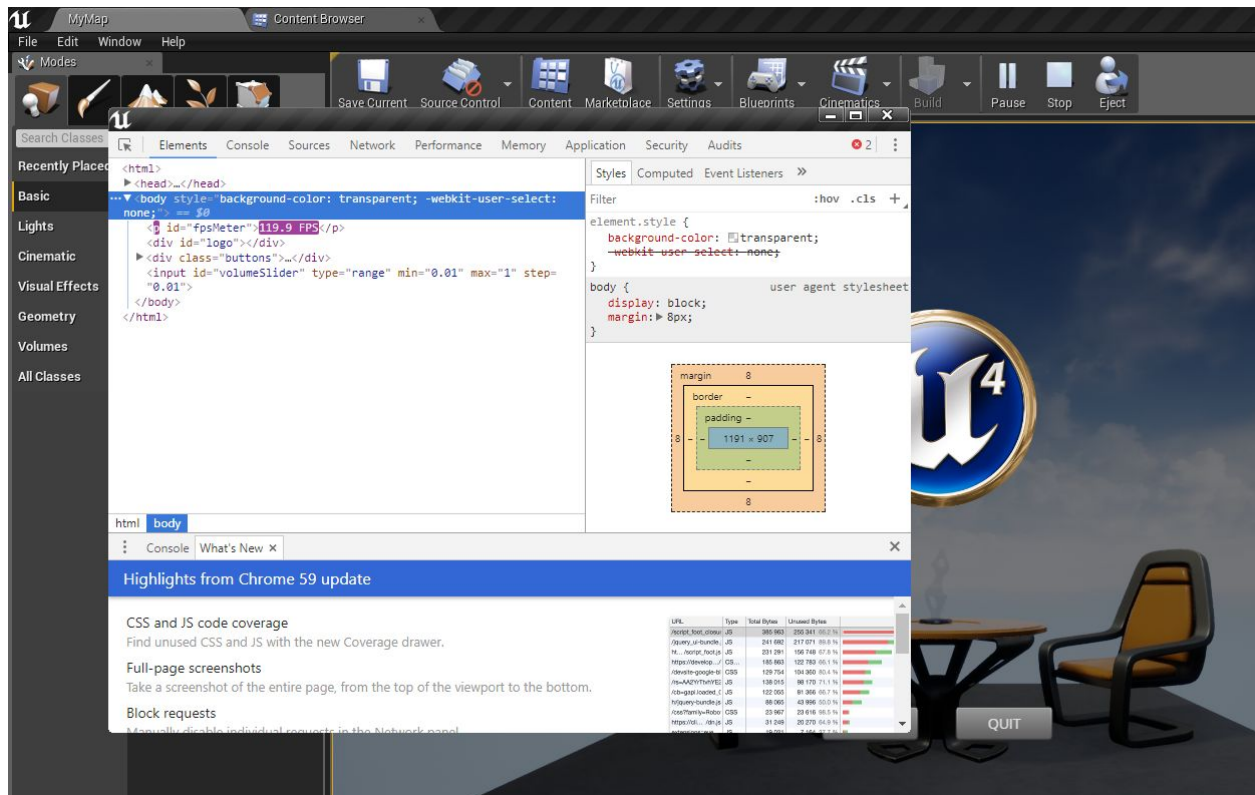
You can find documentation on the Chrome DevTools at the following address:
https://developers.google.com/web/tools/chrome-devtools/

WEBGL

This plugin does not support WebGL on desktop platforms in versions 4.23 and below due to the configuration of the engine. However if you manually compiled from source the following changes can be implemented *at the cost of performance (especially for fullscreen interfaces)*.

Open the *CEFBrowserApp.cpp* file in the */Engine/Source/Runtime/WebBrowser/Private/CEF* directory. Search for the **FCEFBrowserApp::OnBeforeCommandLineProcessing(...)** method as shown in the following example:

```
16    void FCEFBrowserApp::OnBeforeCommandLineProcessing(const CefString&
17    {
18        CommandLine->AppendSwitch("disable-gpu");
19        CommandLine->AppendSwitch("disable-gpu-compositing");
20    #if !PLATFORM_MAC
21        CommandLine->AppendSwitch("enable-begin-frame-scheduling");
22    #endif
23    }
```

Note the two switches `disable-gpu` and `disable-gpu-compositing` appended to the browser subprocess command-line. You will need to remove these two lines of code as shown below:

```
16    void FCEFBrowserApp::OnBeforeCommandLineProcessing(const CefString&
17    {
18        //CommandLine->AppendSwitch("disable-gpu");
19        //CommandLine->AppendSwitch("disable-gpu-compositing");
20    #if !PLATFORM_MAC
21        CommandLine->AppendSwitch("enable-begin-frame-scheduling");
22    #endif
23    }
```

You can now build the UE4 project to use the WebUI plugin with WebGL enabled:
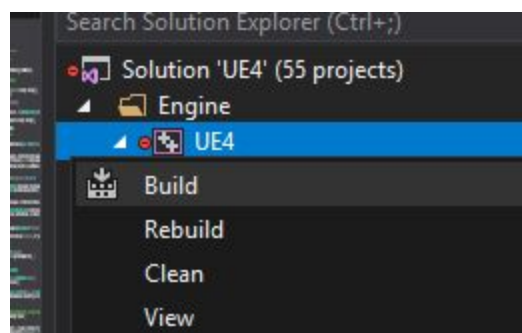
PERFORMANCE

The engine installations for versions 4.24 and 4.25 have WebGL enabled by default which significantly degrades performance (especially for fullscreen interfaces). However if you manually compiled from source the following changes can be implemented.

Open the *CEFBrowserApp.cpp* file in the */Engine/Source/Runtime/WebBrowser/Private/CEF* directory. Search for the **FCEFBrowserApp::OnBeforeCommandLineProcessing(...)** method as shown in the following example:

```
16    void FCEFBrowserApp::OnBeforeCommandLineProcessing(const CefString&
17    {
18        CommandLine->AppendSwitch("enable-gpu");
19        CommandLine->AppendSwitch("enable-gpu-compositing");
20        CommandLine->AppendSwitch("enable-begin-frame-scheduling");
21    }
```

Note the two switches `enable-gpu` and `enable-gpu-compositing` appended to the browser subprocess command-line. You will need to change these two switches to `disable-gpu` and `disable-gpu-compositing` as shown below:

```
16    void FCEFBrowserApp::OnBeforeCommandLineProcessing(const CefString&
17    {
18        CommandLine->AppendSwitch("disable-gpu");
19        CommandLine->AppendSwitch("disable-gpu-compositing");
20        CommandLine->AppendSwitch("enable-begin-frame-scheduling");
21    }
```

You can now build the UE4 project to use the WebUI plugin with WebGL disabled:
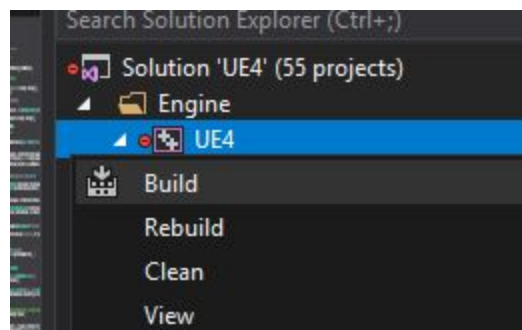
COMPILE

This plugin can be manually compiled for other platforms or engine versions. First open the command prompt (by searching for "cmd" in the start menu) and type the following command:

```
cd "C:\Program Files\Epic Games\UE_4.21"
```

You can copy this command and paste it by right-clicking on the command prompt. Also take note of the **UE_4.21** directory. You need to change this folder to the version that corresponds to the engine version you are using. *If you did not install your engine to the default directory then type the path to your custom installation folder instead.*

Press ENTER to run the command. You should now see output similar to the following:
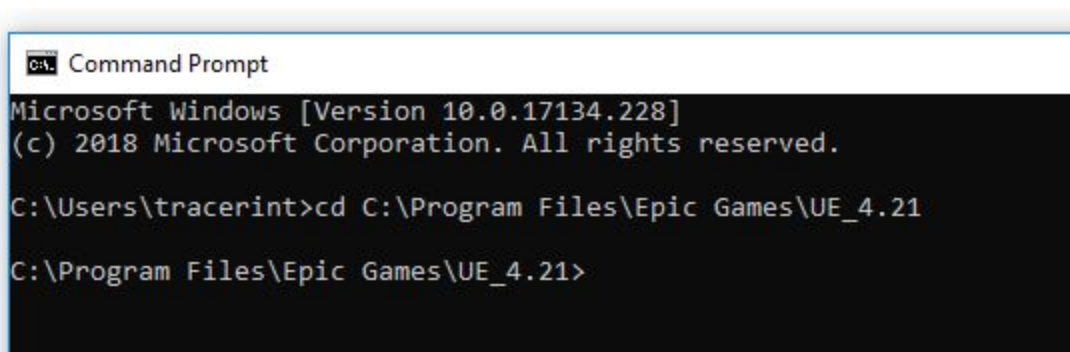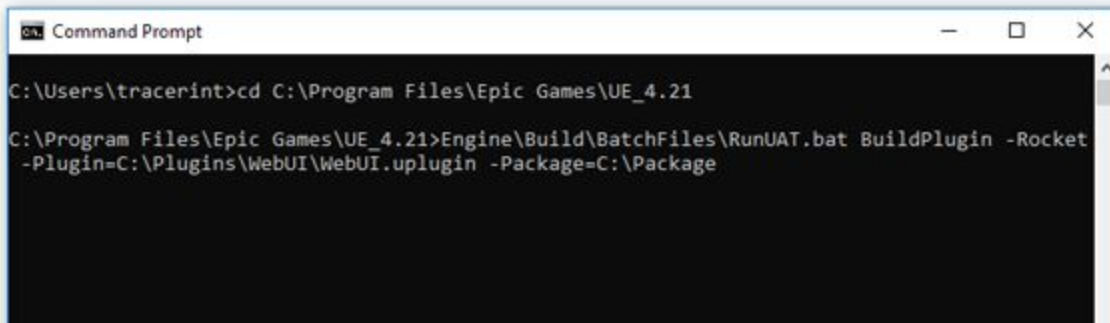
Then type the following command to build the plugin:

```
Engine\Build\BatchFiles\RunUAT.bat BuildPlugin -Rocket -Plugin="..." -Package="..."
```

Replace the first "..." with the path to `WebUI.uplugin` and the last "..." with the path to a temporary "package" folder. You can also drag and drop the `.uplugin` file or your temporary folder directly onto the command prompt and it will automatically type the path:
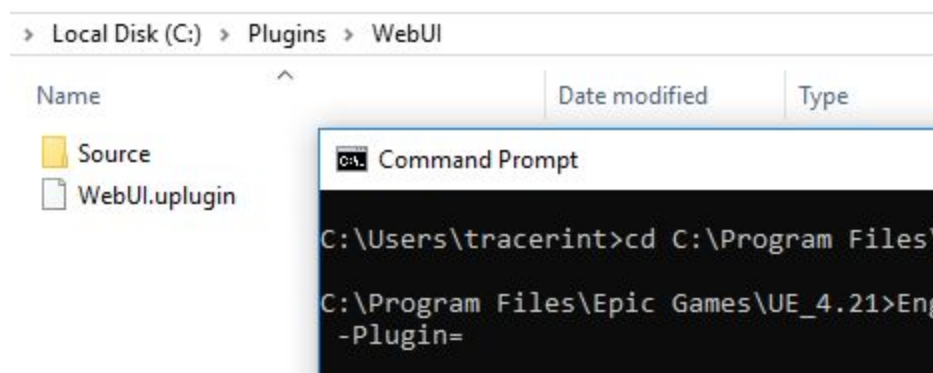
Make sure these paths are <mark>not inside the engine directory</mark> or the build will fail. Once you have the full command typed out it should look similar to the following:

If your machine does not support Mac or Linux builds then you will most likely have to remove the `"Mac"` and `"IOS"` or `"Linux"` platforms from WebUI.uplugin before compiling:

Now hit the ENTER key to run the command. If everything was setup correctly you'll see many different versions of the plugin being compiled for various platforms.

Once the build is complete you should see the "BUILD SUCCESSFUL" message:

Check your temporary "package" folder to ensure it looks similar to the following:

Then copy the files in this temporary folder into your engine installation directory. You must do this beforehand if you are compiling any other plugins that require the WebUI plugin.

You have now successfully compiled the WebUI plugin for your engine version.