Coherent UI for Unreal Engine 4

2.5.5.3

Contents

Chapter 1

Copyright

Chapter 2

Requirements

To use Coherent UI SDK in your project, please make sure that the following minimum system requirements are met by your development environment and are compatible with your project requirements for the end-users.

2.1 Software Requirements

Windows

- Windows XP (required Service Pack 3)
- Windows Vista (required Service Pack 2)
- Windows 7 (required Service Pack 1)
- Windows 8
- Windows 8.1

Visual Studio 2010, 2012, 2013 is required to build Coherent UI SDK. DirectX SDK is required for older versions of Visual Studio to build some of the samples.

Note

- All Windows systems must have the latest Service Pack provided by Microsoft installed.
- DirectX shared textures require Windows Vista or later.
- Latest drivers provided by the hardware vendor are required.
- For Windows 8 and 8.1 only desktop mode applications are supported
- WinRT is not supported
- The Coherent UI Samples require the Microsoft DirectX End-User Runtime (June 2010) to work compile (or at least d3dx9_43.dll and D3DCompiler_43.dll should be added to the output folder).

Linux

Generally any distribution similar to Ubuntu 12.04 LTS.

- glibc 2.14 or later
- libstdc++6 or later
- GTK 2.24.10 or later
- GCC 4.6.3 or later

Note

Only the latest video card drivers are supported. OpenGL 3.0 support is required.

Mac OS X

- 10.7 or later
- XCode 4.4 or later

Note

Limited support for Mac OS 10.6.8 is available, but extensive testing should be performed, and some features might not work.

2.2 Hardware Requirements

- Intel or AMD CPU architectures are supported (no ARM support)
- Dual-core CPU is recommended
- 512 MB RAM
- 350 MB HDD space for development
- 40-80 MB HDD space for runtime libraries

Video adapter:

- Support for shader model 3.0 is required
- DirectX 9.0c or later support
- OpenGL 3.0∗ or later support
- Full non-power-of-two textures support
- Dedicated video card recommended
- Certain features of Coherent UI may be available on earlier versions of OpenGL but we don't test that configuration and results may be unpredictable.
- On configurations with multiple video cards - SLI, Optimus, etc., the faster card should be enabled, and your users should make sure that the same adapter runs both your application and the Coherent UI rendering processes.

Chapter 3

Coherent UI for Unreal Engine 4 plugin

3.1 Introduction

This page will introduce you on how to set-up and use the Coherent UI middleware library with Unreal Engine 4.

Coherent UI is a modern HTML5-based user interface middleware library for realtime applications. It has been pre-integrated with Unreal Engine 4 and all the source of the integration plugin is available to Coherent UI and UE4 licensees.

Coherent UI is based on browser-like technology and can also be used as an in-game browser. It can load and render any page from the web and supports all modern authentication methods and protocols.

This guide requires basic knowledge of both Coherent UI and Unreal Engine 4. For more info on the discussed topics, please refer to the Coherent UI Documentation and the Unreal Engine Documentation.

3.2 Sample game

To quickly try out Coherent UI with the provided sample - try the provided *CoUITestFPS* game.

After the *Installer* has completed, it'll add the game to the UE4 solution in Visual Studio.

1. Build the UE4 Engine
2. Build the *CoUITestFPS* game
3. Run the UE4 Engine
4. Click *Projects->Browse* and browse to the *CoUITestFPS.uproject* file
5. Open the game.

If you try to directly run the game from VS before having it previously opened, you might receive an error *Failed to open descriptor file*.

The game shows a simple menu, HUD and if you press "I", it'll pop-up an in-game inventory in 3D. If you press "Tab" you'll be presented with a simple in-game menu.

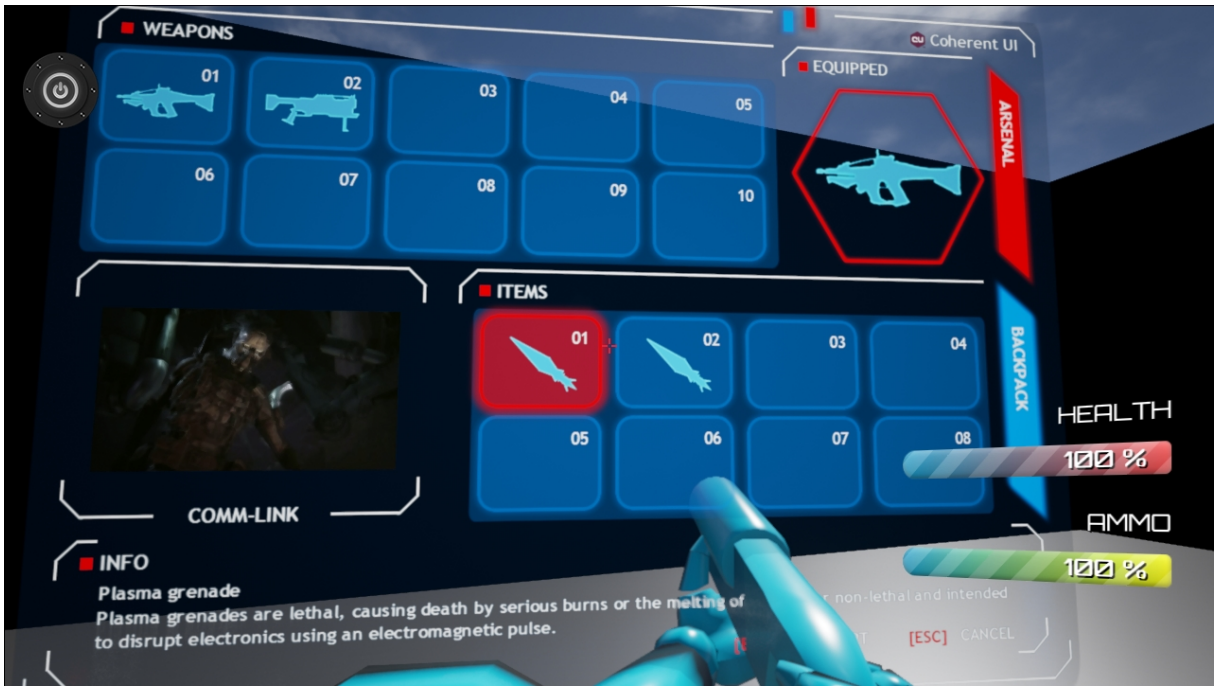


Figure 3.1: Sample UI

3.3 Plugins structure

The Coherent UI integration is divided in two main parts: a "ThirdParty" dependency for UE4 that is the Coherent UI C++ DLL itself along with it's dependencies and an Engine Plugin that provides the "glue" between UE4 and Coherent UI. This plugin exposes several classes and Components specific for the development with UE4 and takes care of all the initialization and management of Coherent UI.

3.4 Setting up the ThirdParty dependency

Coherent UI has to be added as a "ThirdParty" dependency to UE4. This will make sure that all DLL are loaded when the game starts and they will be referenced by the engine correctly. Currently, Coherent UI uses 2 plugins, one for the actual gameplay code, and another Engine plugin that sets up the dependency to the core Coherent UI library.

The Coherent UI installator will make sure that both the Coherent UI core plugin and the gameplay plugins are installed correctly.

Install steps:

1. Run the provided installer.
2. When asked what to install, check "UnrealEngine Plugin".
3. Select which UE4 version you'd like to install the plugin to.
4. Click install.

#There are two ways to add the Coherent UI plugin to your game. Automatic and Manual:

Automatic: Use the Coherent UI installer check the Add to Existing game

option and then activate the plugin from the editor with the following steps:

1. Open the Unreal Engine Editor
2. Click on the Window menu
3. Click Plugins
4. Find the Coherent UI in the list of plugins

Manually edit the plugin registration files

1. Open "YourGame/Source/YourGame/YourGame.Build.cs". Add two private dependency modules - *CoherentUIPlugin* and *CoherentUI*. This can be done by adding these lines in your constructor:

```
PrivateDependencyModuleNames.AddRange(
    new string[] { "CoherentUIPlugin", "CoherentUI" }
);
```

2. Open the Unreal Engine Editor
3. Click on the Window menu
4. Click Plugins
5. Find the Coherent UI in the list of plugins

Note: In case than you can not activate the plugin from the editor, you can also activate it by addiding the following lines in the .uproject file of your game.

```
"Plugins": [
    {
        "Name": "CoherentUIPlugin",
        "Enabled": true
    }
]
```

This will tell the engine that it should load the compiled CoherentUIPlugin dll if your project is using Coherent UI.

Note: CoherentUI will look for resources loaded through custom protocols (such as coui://) in "YourGame/Content/UIResources" folder.

3.5 Adding the Coherent UI plugin to your blueprint game

Using the installer

1. Run the provided installer.
2. When asked what to install, check "Add Coherent UI to an existing game" and **uncheck** "Generate Project Files".
3. Browse to the root directory of your game.
4. Click install.

These are almost the same steps as in the previous section but generating project files will fail if your game has no C++ code.

One last step

Open the editor and enable the Coherent UI plugin for your game using the **Window** -> **Plugins** dialog. Coherent UI plugin is in the Installed / User Interface section.

Note: Without C++ code, you won't be able to distribute your game. See the next section for details.

3.6 Distributing (Packaging) Coherent UI games

UE4 currently doesn't build plugins in blueprint-only projects, thus you can't use CoherentUI (or any other runtime plugin) without any C++ code. You can fix this by adding a simple dummy C++ class to your project which is described in the next paragraph. If your game already has C++ code, skip the next section.

Fix for blueprint-only games

1. Open the UE editor and go to **File** -> **Add Code to Project**
2. Select empty class.
3. Click "Next" until you're asked "Would you like to edit the code now?". Click *yes*. (or click *no* and open the solution in Visual Studio manually)
4. Build the solution from Visual Studio (**CTRL + SHIFT + B**). Don't worry if you get a linker error for "CoherentUIPlugin.dll"

Required steps for distribution

1. Make sure your "UIResources" folder is located in "YourGameDir\Content".
2. Go to **File** -> **Package Project** -> **Packaging settings** and expand the **Packaging** options.
3. Add "UIResources" under **Additional Non-Asset Directories to Package**.
4. Package your game from the editor.
5. Copy the "host" folder from "YourUnrealEngineDir/Engine/Binaries/ThirdParty/CoherentUI/YourPlatform" ("YourPlatform" should be Win32/Win64/Mac/etc.) to "YourPackageDir/WindowsNoEditor/Engine/Binaries/ThirdParty/CoherentUI/YourPlatform"
6. Copy the contents of "YourUnrealEngineDir/Engine/Binaries/ThirdParty/CoherentUI/YourPlatform" (without the "host" folder) next to the game executable (so that "CoherentUI.dll" is next to "YourGame.exe").

Note: Previous CoherentUI versions loaded UIResources from the root of your game directory (as opposed to "YourGameDir\Content"). That behaviour is still supported but it is **deprecated** and will be removed in future versions. If you are still using that behaviour, you are required to manually copy the "UIResources" folder to "YourPackageDir/WindowsNoEditor/YourGameDir" (next to "Content" and "Binary").

Your package directory structure should resemble this:

```
|-YourPackageDir
|-WindowsNoEditor
    |-Engine
        |-Build
        |-Binaries
            |-ThirdParty
              |-CoherentUI
                |-Win64
                  |-host
                |-<any_other_platform>
                  |-host
    |-YourGameDir
      |-Saved
```

```
|-Content
  |-UIResources
|-Binaries
  |-Win64
    |-CoherentUI64.dll
    |-YourGame.exe
  |-Win32
    |-CoherentUI.dll
    |-YourGame.exe
  |-Mac
    |-libCoherentUI.dylib
    |-YourGame.exe
  |-<any_other_platform>
    |-<CoherentUI client library>
    |-YourGame.exe
```

3.7 Coherent UI Usage

The current plugin allows to easily add Coherent UI Views to UE4 HUDs as well as surface on the world - that is having UI Views as textures on objects.

All exported Components inherit the "CoherentUIViewListener" class. It takes care to listen manage the creation/destruction of Coherent UI View resources and textures and listens to View events. Some of the View events are also broad-casted through Unreal Engine Delegates. Almost all interactions can be achieved through the Unreal Editor and the Blueprint Editor, and all View properties are exposed to them.

Coherent UI comes to Unreal with an editor plugin that eases your workflow. The next section describes the editor plugin. For detailed information about each feature scroll past the section.

3.8 Coherent UI Editor Plugin

Coherent UI's Editor Plugin adds pulldown on the main toolbar in your Unreal Editor, to the right of the Window menu.

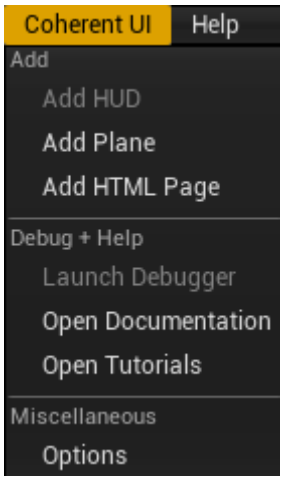


Figure 3.2: Coherent UI's Editor Menu

Here's an overview of the functionality provided by each item in the menu:

Add HUD

Changes the **GameMode**'s HUD class to the one provided with Coherent UI - **CoherentUIGameHUD** and adds a HTML file named *hud.html* to *YourGame/Content/UIResources*.

This item is only available if the current default **GameMode** is a Blueprint class. To see or change your current default **GameMode**, go to **Edit** -> **Project Settings** -> **Maps and Modes** -> **Selected GameMode**. **GameMode** classes written in C++ can't be changed from the editor which is why to use this option you would need to create a new blueprint that inherits from `AGameMode`. For further information, please refer to the UE4 docs.

By default, **CoherentUIGameHUD** will try to load *hud.html* (and use several others sensible default settings). If you'd like to customize the HUD's settings, you can do so with the following Blueprint:

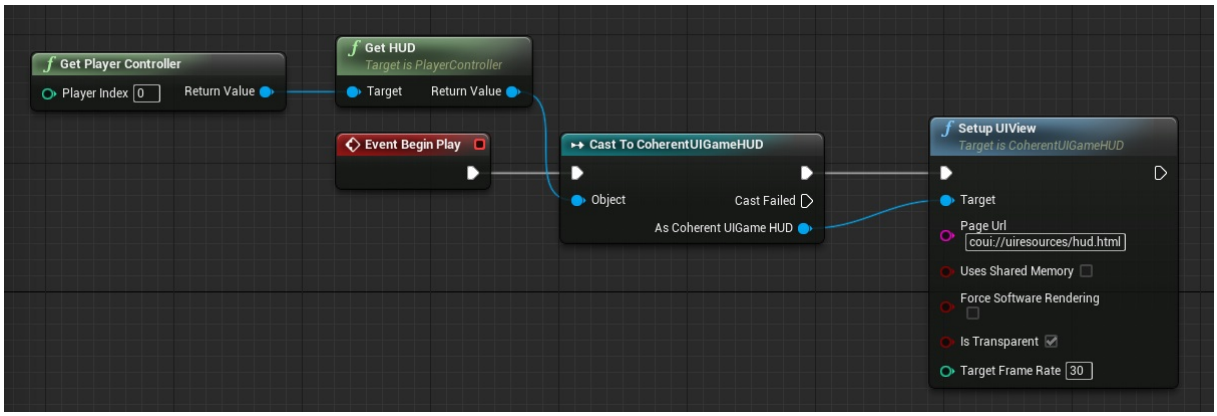


Figure 3.3: Blueprint HUD with CoherentUIGameHUD HUD class

It's important to call `SetupUIView` as shown in the image above during `Event Begin Play` or otherwise the HUD will fallback to its default settings.

Add Plane

Adds a 3D Plane with a **CoherentUIComponent** to the scene, facing current camera's position. The plane is an instance of *CoherentUIPlane* - an asset of ours that will be copied to *YourGame/Content* if it's missing.

Add Page

Adds a HTML page to *YourGame/Content/UIResources* and launches it in your browser.

Launch Inspector

Launches Coherent UI GT's Inspector. You can change the path to the debugger in the settings (see below).

This item is only available if the game is currently running in PIE.

Open Documentation

Opens Coherent UI's documentation on http://coherent-labs.com.

Open Tutorials

Opens Coherent UI's tutorials on http://coherent-labs.com.

Settings

Opens Coherent UI's settings (you can also access them from **Edit** -> **Project Settings** -> **Plugins** -> **Coherent UI**. This enables you to change the Debugger's port and path.

3.9 Coherent UI Views

At the core of Coherent UI stays the so called *View*. A Coherent UI View is basically an HTML5 page and the JavaScript context for it. Coherent UI will render the View, execute the JavaScript and provide UE4 with a texture with what it has drawn.

Views provide methods to change the current page, use local resources, interact with JavaScript. In UE4 you'll seldom have to use them directly as they are encapsulated in Components that can be manipulated in code or Blueprint.

3.10 Coherent UI Components

The Coherent UI UE4 plugin has two main Components - *UCoherentUIHUD* and *UCoherentUIComponent* and an actor - *ACoherentUISystem*. This document introduces in detail all of them. There's also the *UCoherentUILiveView-Component* component which allows easy transfer of image data from UE4 into your HTML page.

All Coherent UI UE4 Components inherit from the *UCoherentBaseComponent* Component. It should *not* be used directly by itself. It exports all the most important Coherent UI properties and events to the UE4 Engine and Blueprint Editor.

Coherent UI renders all pages out-of-process and hence needs some time to start-up. All Components provide an important *IsReadyToCreateView* method. You shouldn't create the Components *View* before it is ready to do so.

The *system* actor takes care of properties that are global for all the Components. If you don't create one, a default will be created in the game for you.

3.11 Coherent UI Views for HUDs (C++)

The "CoherentUIHUD" component can be used to easily add Coherent UI Views to a game HUD. A sample Game accompanies this code and details can be seen there. The Component must be added to the HUD Actor of the game and than the HUD is in charge of creating it's View and must call the "CoherentUIHUD::DrawHUD" method when inside it's own "DrawHUD" method. Details can be seen in the *CoUITestFPSHUD.h* and *CoUITestFPSHUD.cpp* files.

The *UCoherentUIHUD* class is a Component that takes care of loading the HTML page with the HUD interface and can draw it in the Canvas of the HUD Actor. In the sample the *CoherentUIHUD* Component is created and added to the ACoUITestFPSHUD Actor with the following code:

```
CoherentUIHUD = PCIP.CreateDefaultSubobject<UCoherentUIHUD>(this, TEXT("CoherentUIHUD"));
```

Then on every *DrawHUD* event the Coherent HUD is checked for readiness and eventually draw the HUD.

```
if (CoherentUIHUD)
{
    if (CoherentUIHUD->IsReadyToCreateView() && !CoherentUIHUD->HasRequestedView())
    {
        CreateView();
    }
    // check that the size matches
    auto view = CoherentUIHUD->GetView();
    if (view)
    {
        if (view->GetWidth() != Canvas->ClipX
            || view->GetHeight() != Canvas->ClipY)
        {
```

```
            CoherentUIHUD->Resize(Canvas->ClipX, Canvas->ClipY);
        }
    }
    CoherentUIHUD->DrawHUD(Canvas, 0, 0);
}
```

In the snippet above you can see that the Component is checked for readiness and if it has already requested the creation of a *View* for itself. It it hasn't, it'll call the *CreateView* method. If we already have a *View* we check if it has to be resized (for instance after we've resized the game window) and calls the *DrawHUD* method with the current Canvas and the coordinates on the screen where we want our *View* to draw itself.

The following code will request the creation of a View:

```
void ACoUITestFPSHUD::CreateView()
{
    Coherent::UI::ViewInfo info;
    info.Width = Canvas->ClipX;
    info.Height = Canvas->ClipY;
    info.TargetFrameRate = 60;
    info.UsesSharedMemory = true;
    info.IsTransparent = true;
    info.SupportClickThrough = true;

    CoherentUIHUD->ReadyForBindings.AddDynamic(this, &ACoUITestFPSHUD::BindUI);
    CoherentUIHUD->CreateHUDView(info, TEXT("coui://UIResources/HUD/hud.html"));
}
```

All properties are self-explanatory - for more detail on each of them please consult the Coherent UI C++ documentation.

The line

```
CoherentUIHUD->CreateHUDView(info, TEXT("coui://UIResources/HUD/hud.html"));
```

is the one that actually initiates the creation of the *View*. The "coui://UIResources/HUD/hud.html" page is the initial page that will be loaded. Note that *coui* is a special protocol that denotes local resources in the game. *coui* is read through a *FCoherentFileHandler* that for UE4 can be found in the *CoUISystemHolder.cpp* file. It will take care to read all files relative to the game's current content directory. Coherent UI supports all standard protocols too, so you can load pages from the Internet.

The more interesting snippet above is the line:

```
CoherentUIHUD->ReadyForBindings.AddDynamic(this, &ACoUITestFPSHUD::BindUI);
```

It binds a method that will be called when the *View* is ready to accept its *bindings*. *Binding* is the facility that allows the communication between C++ and JavaScript code in the page. For more information please refer to the *Scripting* section of this document.

3.12 Coherent UI Views for HUDs (Blueprints)

There are 2 ways to add a HUD View using blueprints.

The first way is by using a predefined Coherent UI HUD class.

You'll need a game mode override in the Editor's World Settings which uses the *CoherentUIGameHUD* HUD class.

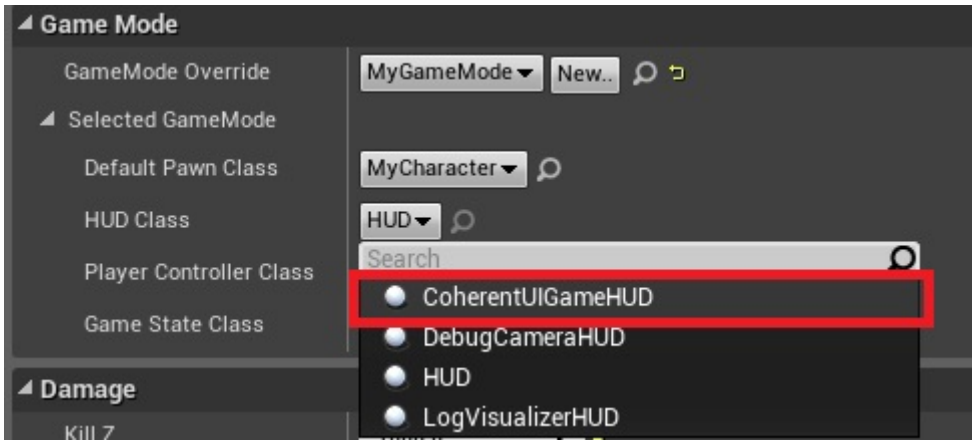


Figure 3.4: World Settings custom HUD class

Then, you need to initialize the HUD using a blueprint similar to this one:

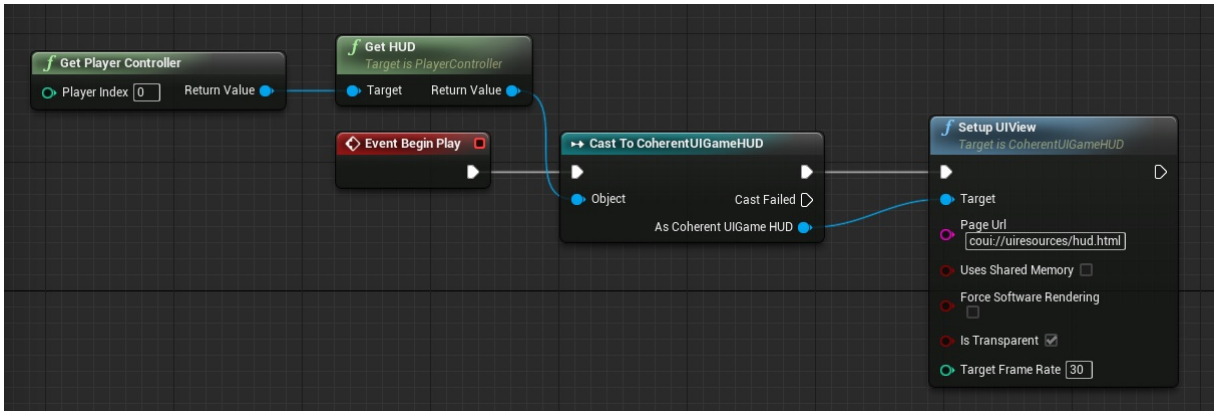


Figure 3.5: Blueprint HUD with CoherentUIGameHUD HUD class

If you don't initialize the HUD, Coherent UI will try to load your hud from **coui://uiresources/hud.html** and use default view settings.

The second way is by adding a Coherent UI component to the HUD.

If you have a custom HUD class that you want to use instead of the *CoherentUIGameHUD* one, you can use this option.

Start by creating a new game mode (e.g. "MyGameMode"), then create a new blueprint whose parent class is the HUD class you want to use. In the following example we use the generic *HUD* as a parent for blueprint:

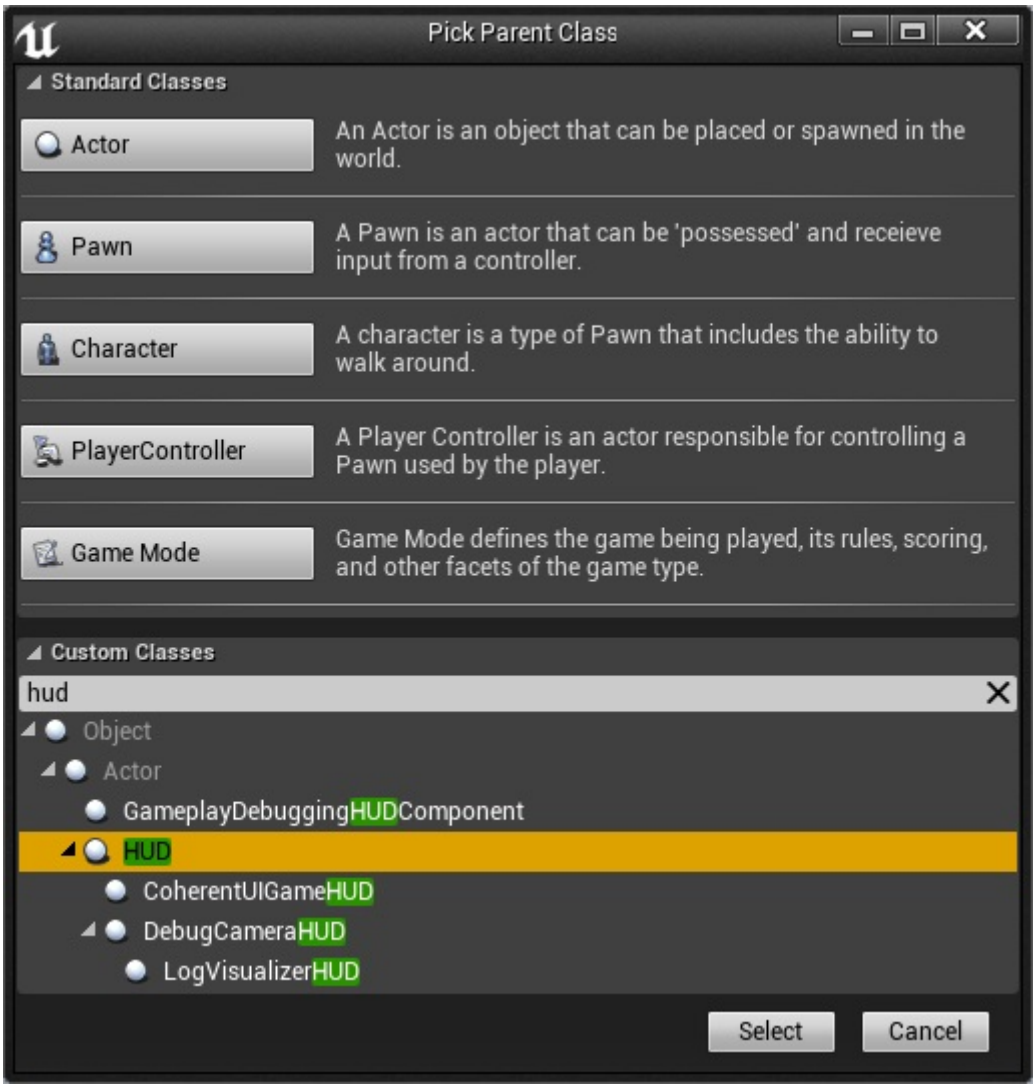


Figure 3.6: HUD class with parent

Use it in the game mode override:

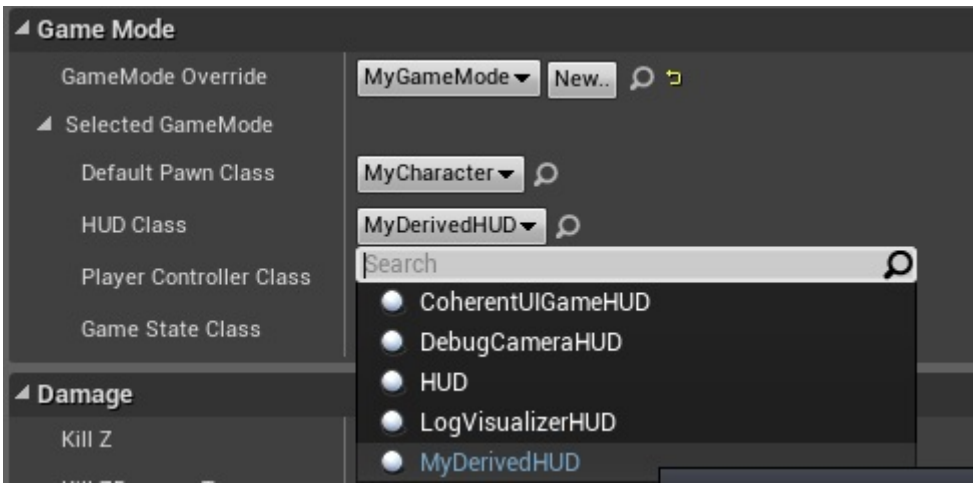


Figure 3.7: HUD class in world settings

Add a *CoherentUI* component to the HUD blueprint you just created:

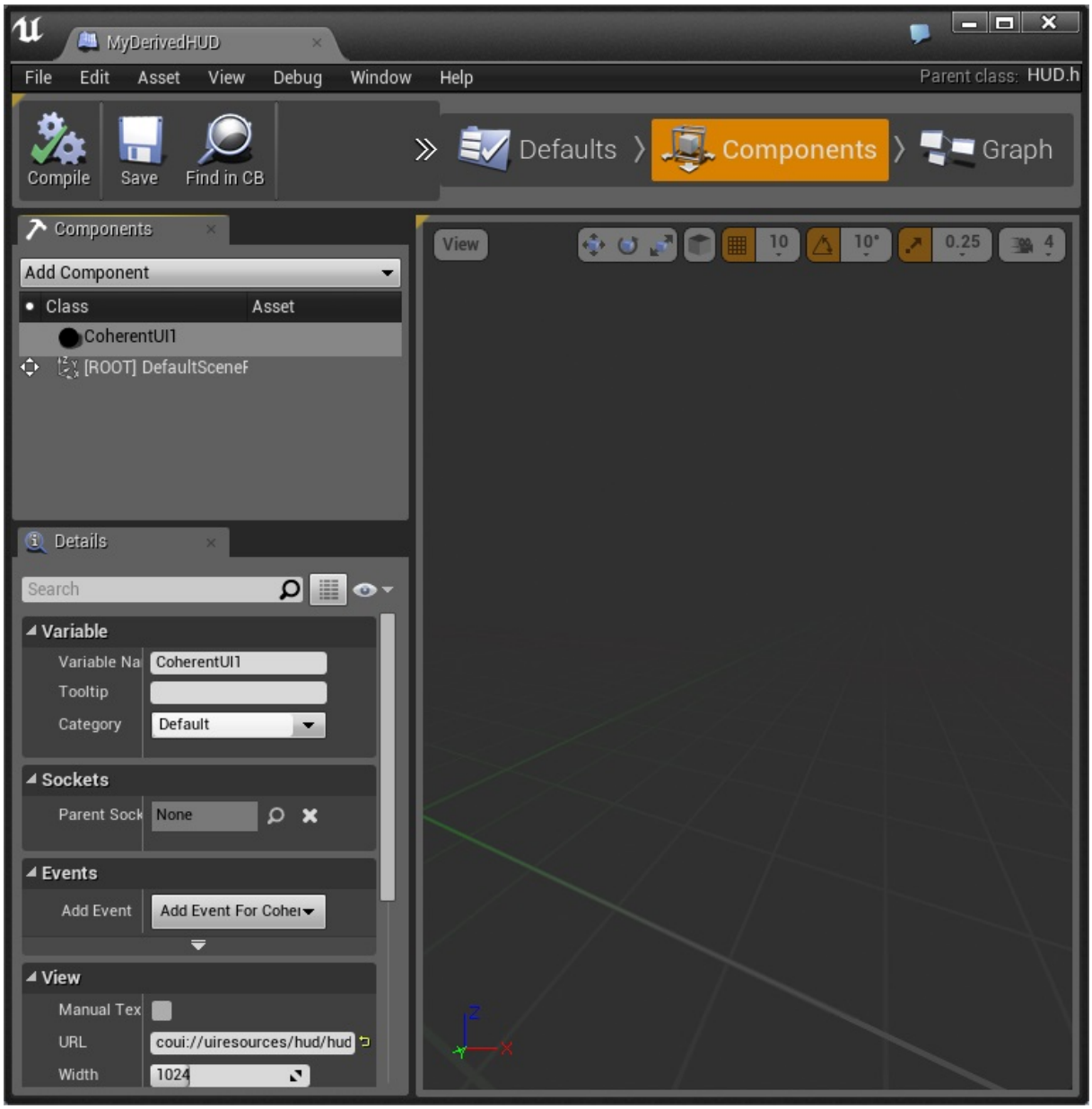


Figure 3.8: Coherent UI component in HUD blueprint

Finally, just draw the texture of the Coherent UI view in the Draw HUD event:

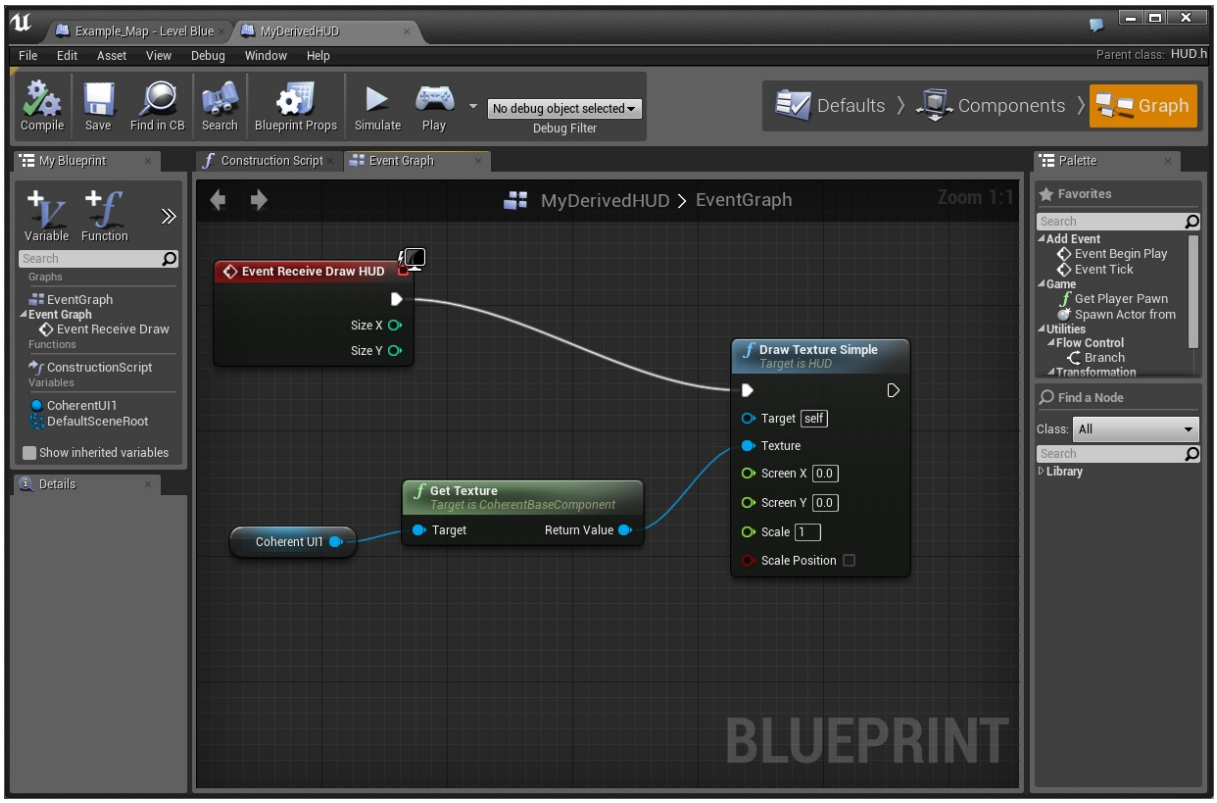


Figure 3.9: HUD blueprint graph

3.13 Coherent UI Views for in-game Surfaces

Coherent UI Views can be used as textures on in-game objects to create interactive displays, call-outs and many other interesting effects. The component used is called "CoherentUIComponent". The steps to add a View to an object in the world are straightforward.

1. Create a Material in the Editor and connect a Texture2D object to it's "Base Color" input (or any field you need)
2. Make the Texture a "Parameter" by right-clicking on it and selecting "Convert to Parameter".
3. Name the Parameter "UITexture". The Coherent UI Component will later enumerate the materials and search for a parameter named so that it will dynamically update with the View texture.
4. Create a "Material Instance" from said Material
5. Find the object you want to apply the View to and create a Blueprint for it.
6. Set the material to the just created Material Instance
7. Add the "CoherentUIComponent" to the Blueprint and set all the parameters you need for it - size, URL etc.

Now you can use the Blueprint and add it in the world. The UI Component will automatically update the "UITexture" parameter with the texture of the View.

A material setup for use by Coherent UI can be found in the sample game and is named *CoherentUIMaterial*. In the sample game please check the *CoUIPlane* and the *StaticActorCoherent* objects that have the material setup and a "CoherentUIComponent" and can show pages on the in-world. The *StaticActorCoherent* is a cube that shows web pages on all its faces.

The properties of each *View* can be edited in the Bluperint editor in the *Components* menu under the "CoherentUI-Component" *View* section.

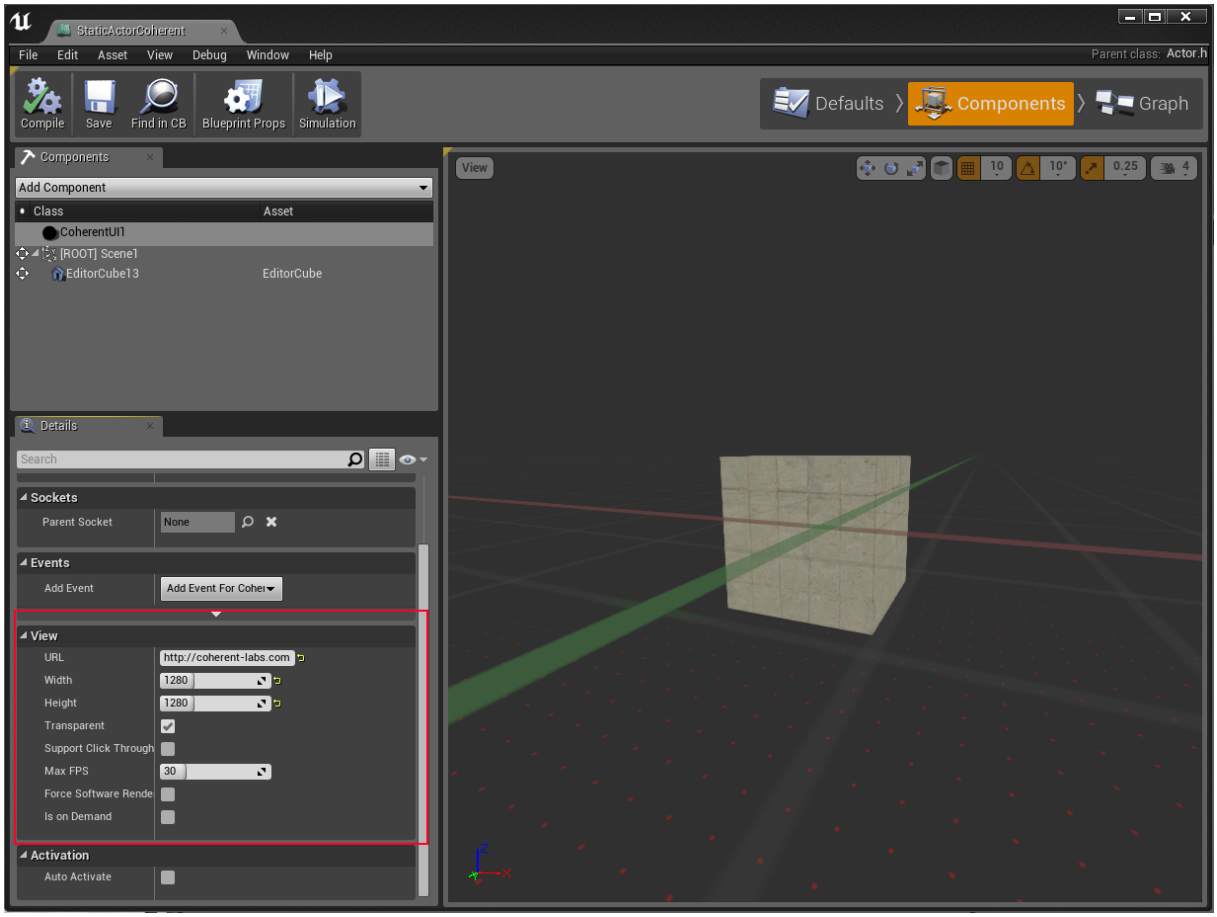


Figure 3.10: View properties

3.14 Coherent UI Live Game Views

Live Game Views are Coherent UI's high performance way to transfer image data from your game engine into a HTML canvas in your UI.

To use them in an actor/blueprint, you need to do the following:

- Add a Coherent UI component
- Add a Coherent UI Live View component (it depends on the Coherent UI component)
- Set up the UTexture that is going to be transferred to the HTML UI
- Add code in your HTML page that will render the image data into a canvas

The Live Game View has the following properties:

- LinkName: Unique string identifier of the Live Game View
- Texture: The UTexture instance that will be sent to the HTML UI. **Warning**: currently only RGBA textures are supported.

- UpdateEveryFrame: Whether the texture is sent every frame in the component's `TickComponent` function. If set to false, you'll need to call `UCoherentUILiveViewComponent::QueueUpdateLiveViewOnRenderThread` yourself whenever you want to queue an update.

In the HTML there's one requirement: you need to have `coherent.js` in the page. Whenever UE4 sends texture data, you'll receive the `onEngineImageDataUpdate` callback on your <canvas> objects with two arguments - the name of the link and the `ImageData` object. The following sample demonstrates placing the received data in the canvas:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <script type="text/javascript" src="js/jquery-2.0.3.min.js"></script>
        <script type="text/javascript" src="js/coherent.js"></script>
    </head>
    <body>
        <div id="live-view-container">
            <canvas id="live-view-canvas" width="384" height="384"
                style="border:1px solid #d3d3d3; ">
                Your browser does not support the HTML5 canvas tag.
            </canvas>
        </div>

        <script type="text/javascript">
            $(document).ready(function() {
                var c = document.getElementById("live-view-canvas");
                c.onEngineImageDataUpdated = function (linkName, image) {
                    // If you have more than 1 live view in your page,
                    // you'll need to check the name of the live view link
                    var ctx = c.getContext("2d");

                    // It's important to clear the canvas before drawing
                    ctx.clearRect(0, 0, c.width, c.height);
                    ctx.putImageData(image, 0, 0);
                }
            });
        </script>
    </body>
</html>
```

Coherent UI Live Game Views pause

After you have added a Live Game View to an actor, you can specify the behaviour of the View during a pause. There is an option called UpdateWhenPaused in the properties of the LiveViewComponent which, depending on whether it is set or not, will update the Live View.

3.15 Coherent UI System

To use Coherent UI during gameplay a *UISystem* object has to be created and live somewhere. The plugin automatically takes care of the creation and management of the UISystem, using a default set of options. This is done by spawning a *ACoherentUISystem* Actor in the world that will set everything up.

3.16 Using custom UI System

If you would like to customize the UI System properties, you can spawn a *ACoherentUISystem* Actor of your own and edit its properties in the Defaults tab of the Blueprint Editor. The System properties define many aspects of the runtime such as to whether support cookies, the UI Debugger port etc. If you don't add a *ACoherentUISystem* Actor, then a default one will be created as soon as you try to create a View.

Note: The easiest way to spawn a Custom UI System is to drag the "CoUISystemBlueprint" blueprint into the world in editor mode and then edit its settings by clicking the "Edit CoUISystemBlueprint" in the Scene Outliner window.

Coherent UI behaviour when the UE4 engine is paused

Sometimes it may be the case that the UI should be paused when the UE4 engine is paused. There is an option for that, which requires using a custom Coherent UISystem (Refer to the *Using custom UI System* section for details).

Note: The default value of the UpdateWhenPaused option is set true. To change it:

1. Create an instance of the CoUISystemBlueprint in the world.
2. Locate the Scene Outliner tab in the UE4 editor and find the instance of CoUISystemBlieprint
3. Click "Edit CoUISystemBlueprint"
4. Uncheck the UpdateWhenPaused option in System Settings section.

3.17 Input overview

The input in Coherent UI is managed through a Slate widget (`SCoherentUIInputForward`). This widget is an empty widget that covers the whole screen and forwards input to a Coherent UI View. For simplicity we have created the `ACoherentUIInputActor` actor, which encapsulates the logic behind the widget and is accessible through both C++ and blueprints. All you have to do is spawn the actor and call its initialize method.

Keyboard, Mouse and Joystick focus in UE4

Keyboard focus, in UE4 terms, means that the Coherent UI Input Forward widget has keyboard focus. When the widget has keyboard focus it is able to receive keyboard events, otherwise it is not.

Mouse events don't require focus and they are propagated to the widgets under the cursor until one of the widgets handles the event. The Coherent UI Input Forward widget is generally on the top of the viewport so it can process mouse events first. The processing in the widget works as follows:

- If the widget does not have **keyboard** focus and the widget isn't set to always accept mouse events anyway, the event is unhandled. By default, the widget does **not** always allow mouse events so you need to have keyboard focus to receive them.
- Check the pixel under the cursor for every HUD view. If a solid pixel is found (solid meaning having an alpha value greater than the click-through alpha threshold specified in the view component) the event is handled.
- If none of the HUD views should handle the mouse event, a raycast through the mouse position is made. Depending on the widget settings it can trace either multiple objects, or just the first object the ray hits. If the hit object has a Coherent UI component the event is handled.
- If neither a HUD view or a 3D object view should receive the mouse event, it's left unhandled and it's propagated to the next widgets.

Joystick input works by default when the viewport widget has focus. Keyboard focus implies joystick focus, which means that if the Coherent UI widget has keyboard focus, the joystick events will stop working. If that's not intended, the input actor has a method for keeping joystick focus to the viewport after setting keyboard focus.

View focus

Each view can be focused, regardless of the widget focus in UE4. This is core functionality and it's independent of any other focus mechanics. The view focus controls the color of selected text, whether you have a blinking caret in input fields and the like.

Here are a few examples for views that have focus (left) and ones that don't (right):

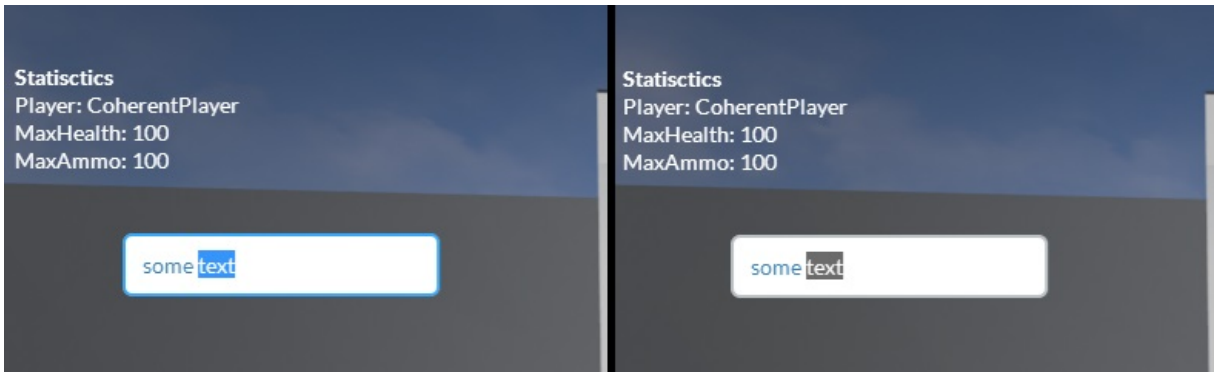


Figure 3.11: Selected text

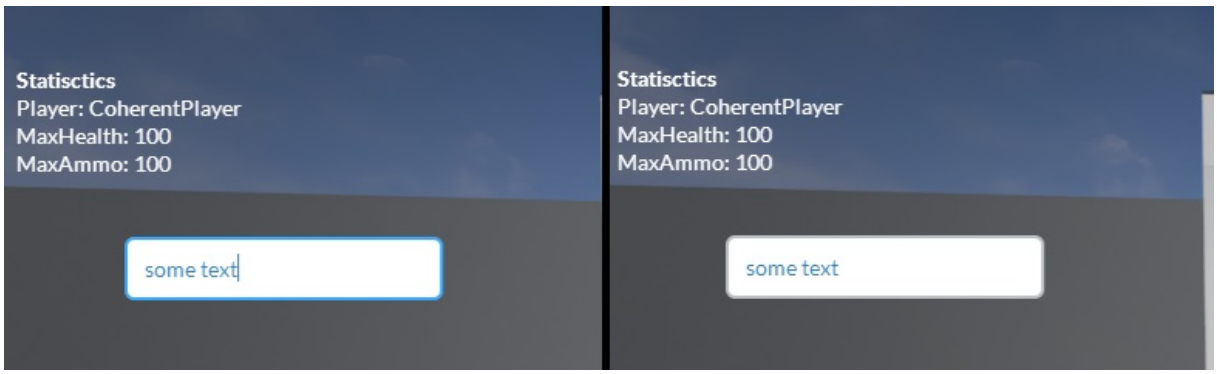


Figure 3.12: Blinking caret

The current implementation of the input actor/widget allow for only one focused view at a time. If you need some custom behaviour you'll have to modify the plugin's code.

Each view component has a property (`Receive Input`) that controls whether a specific view takes part in the mouse event forwarding.

3.18 Input (C++)

The input in Coherent UI is managed through a Slate widget (`SCoherentInputForward`). This widget is an empty widget that covers the whole screen and forwards input to a Coherent UI View. The widget should be added in the front of the viewport so it can process mouse and keyboard messages first. Due to Unreal Engine 4's architecture, the widget can process *keyboard* messages **only if it's focused**. To focus the widget, you can use the `FSlateApplication::Get().SetKeyboardFocus(Widget)` method. Focus management must be implemented by the client.

You can also show the system mouse cursor using `FSlateApplication::Get().ResetToDefaultInputSettings();`. If you want to set focus back to the game, `FSlateApplication::Get().SetFocusToGameViewport();` will do that.

If the `SCoherentInputForward` widget has focus, then keyboard events are sent to the *focused* Coherent UI View. To focus a view, you should click anywhere on it.

Mouse events are sent only to the Coherent UI View that is below the cursor. The code is inside the `SCoherentInputForward::OnMouseEvent` method. The method does roughly the following:

- Checks if the mouse is over a solid pixel of any of the HUD Views, using the `View::IssueMouseOnUIQuery`, `View::FetchMouseOnUIQuery` and `View::IsMouseOnView` APIs. If any of the HUD

Views is below the mouse, the appropriate mouse event is sent to it.

- If none of the HUD Views should receive the event, a raycast through the world's geometry is done. Note that the raycast is configured to check only bodies of the **ECC_WorldDynamic** physics type. You can change that by using the `SCoherentInputForward::SetRaycastCollisionGroup` API. Currently, the collision group is set for all objects. If an object is hit and it has a `UCoherentUIComponent`, the mouse coordinates re converted in Coherent UI View space and sent to the View. The conversion is done using the UV coordinates of the static mesh. By default, the first UV channel is used, but that can be changed with `SCoherentInputForward::SetRaycastUVChannel`.

To setup the widget in the *CoUITestFPSSample* the following code is added in CoUITestFPSCharacter.h:

```
TSharedPtr<class SCoherentInputForward> CoherentInputForwardWidget;
```

Then, in the end of the `ACoUITestFPSCharacter::PostInitializeComponents` in `CoUITestFP-SCharaceter.cpp`:

```
CoherentInputForwardWidget = SNew(SCoherentInputForward).Owner(this);
if (GEngine->IsValidLowLevel())
{
    GEngine->GameViewport->AddViewportWidgetContent(
        SNew(SWeakWidget).PossiblyNullContent(
            CoherentInputForwardWidget.ToSharedRef()));
}
if (CoherentInputForwardWidget.IsValid())
{
    CoherentInputForwardWidget->SetVisibility(EVisibility::Visible);
}
```

The code above creates and adds the `SCoherentInputForward` widget to the front of the viewport. After that the widget takes care for sending input to Coherent UI as explained above.

Of course, you can also use the Coherent UI `View` API to create your own input events and send them to Coherent UI. The `View` provides the `MouseEvent`, `KeyEvent` and `TouchEvent` methods that send input to the it. Each Coherent UI input event has to be populated with properties that can be trivially translated from the UE4 events.

3.19 Input (Blueprints)

To add input in a blueprint only game, you need to spawn an actor that will set up the `SCoherentInputForward` widget. This widget is an empty widget that covers the whole screen and forwards input to a Coherent UI View. The widget should be added in the front of the viewport so it can process mouse and keyboard messages first. In fact, due to Unreal Engine 4's architecture, the widget can process *keyboard* messages **only if it's focused**.

Here's a sample blueprint that enables the input:

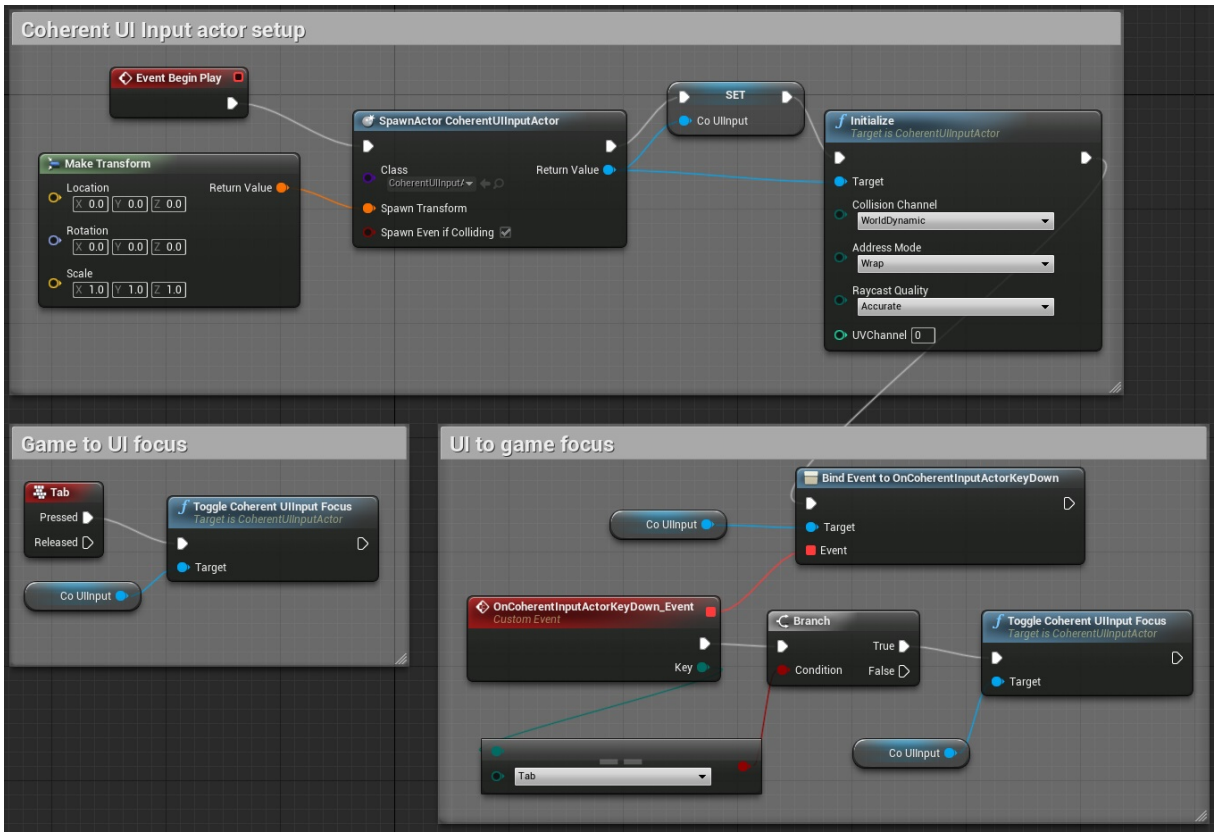


Figure 3.13: Blueprint Input

The actor setup part spawns and initializes the actor, which in turn adds the `SCoherentInputForward` widget to the viewport.

Note

At any time there should be **only one** input actor spawned, otherwise the input might not work correctly.

The "Game to UI Focus" part is for managing input focus - when you press *Tab*, the character input will stop and the UI will take it. This would seem to be enough, but currently UE4 stops firing key events when the focus is not on the viewport so we need other means to get focus back to the game. This is shown in the "UI to game focus" part. The Coherent UI input actor provides events for key/mouse up/down - in the sample we're using the key down event. When it's fired there's a check for the key enum to see if it's the *Tab* key, and if it is, the focus is toggled.

The "Toggle Coherent UI Input Focus" function provides means for toggling input without tracking it. If you want to, there are also functions for querying and setting the focus explicitly.

3.20 Input use cases

Here we outline some common usages of the input system. The examples are given using blueprints and the Coherent UI Input actor as that maps directly to C++ code. We'll assume that you already have a spawned and initialized Coherent UI Input actor:

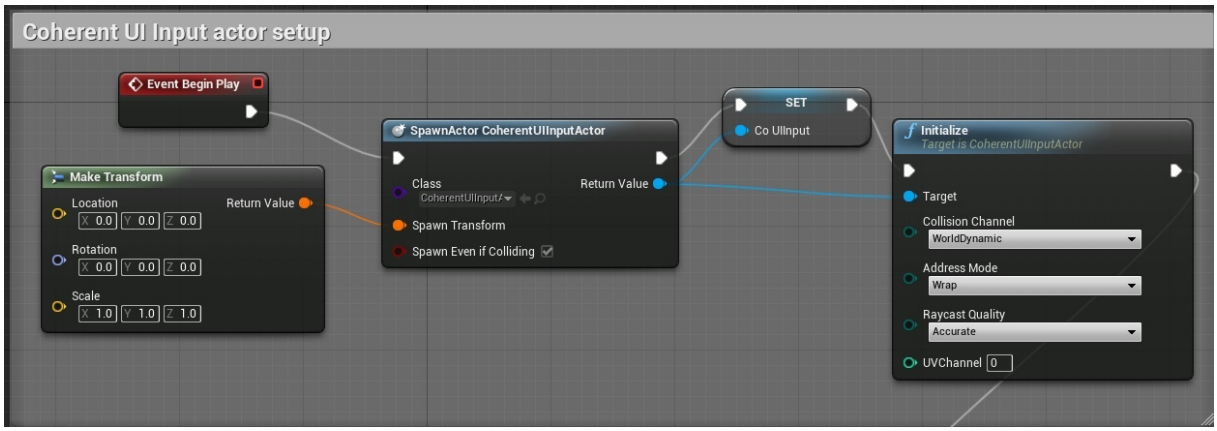


Figure 3.14: Initialize Input Actor

Toggling input focus between the UI and the Game

Toggling input focus can be done 2 ways:

- Clicking with the mouse on a Coherent UI view gives focus to the view since the Unreal's Slate system will focus the widget that handles the click. If you click back somewhere in the game viewport, the game will take focus.

- The "click to focus" method works, but it in a more realistic use case you'd want to be able to change the focus programatically. You can do that using the Coherent UI input actor's `ToggleCoherentUIInputFocus` method. The actor keeps track of the current focus internally so it can toggle between game and UI at any time. If you want to set it explicitly, you can use `SetCoherentUIInputFocus`, or query the state with `IsCoherentUIFocused`.

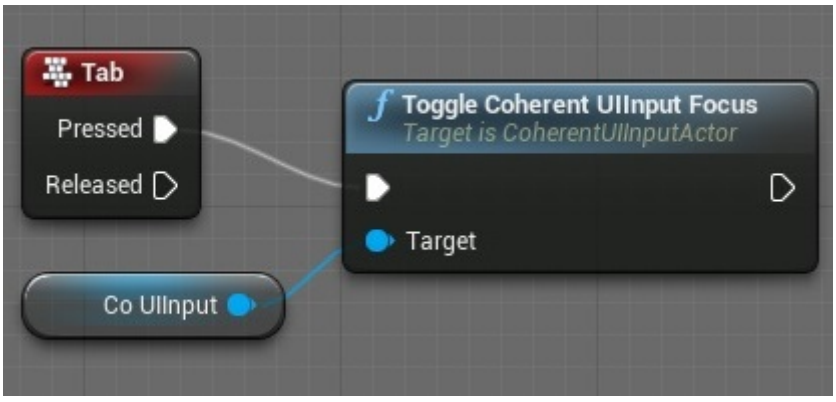


Figure 3.15: Toggle focus

After the Coherent UI Input Actor has focused the input forwarding widget, you need to set which view should receive the keyboard input. This can be done with the `SetCoherentUIViewFocus` method. An alternative is to simply let the user click on the input field that she wants to type in.

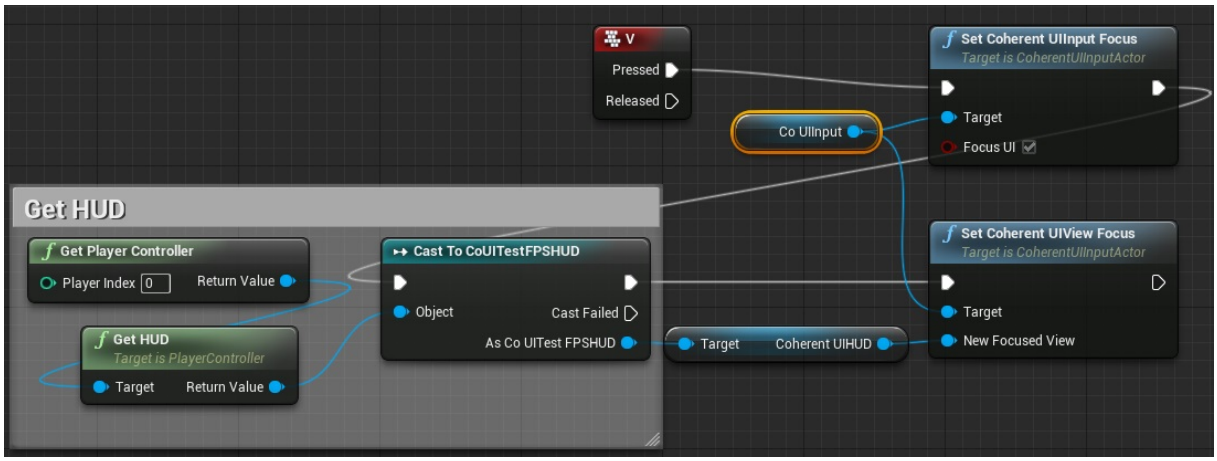


Figure 3.16: Set view focus

Propagating keyboard and joystick events when UI is focused

When the UI is focused, keyboard events are consumed by the Coherent UI GT Input forward widget by default. You can change that using the `SetInputPropagationBehaviour` method. The available options are to forward keyboard events, joystick events, both or none to other widgets.

This can be useful if you need to have your action handlers (e.g. "Jump", "Fire", etc.) executed even if the focused widget isn't the game viewport.

Note that joystick forwarding is only taken into account when using the Coherent UI GT Input actor's methods for setting (or toggling) focus.

Figure 3.17: Input propagation

Forwarding mouse input events to Coherent UI regardless of the focused widget

If you want your mouse input to be forwarded to Coherent UI first for hover events, but still receive keyboard events in your game, then you can use the `AlwaysAcceptMouseInput` method. When set to `true`, mouse events will be received by Coherent UI, regardless of whether the input forward widget is focused or not.

Figure 3.18: Always accept mouse input

Including or removing specific Coherent UI components in the input forwarding

To control whether a Coherent UI view receives mouse events you can change the View component's `Receive Input` property. Only when this property is set to `true` will the component be considered for receiving input events.

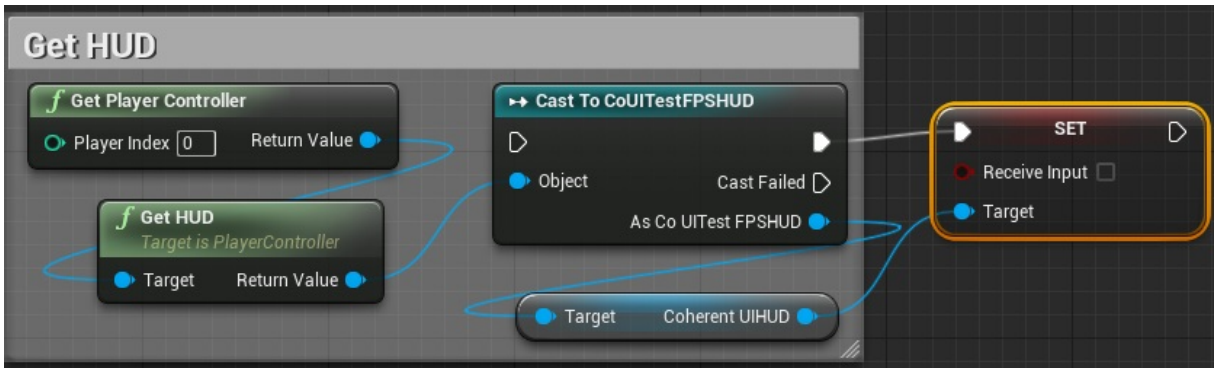


Figure 3.19: Set Receive input property dynamically with BP

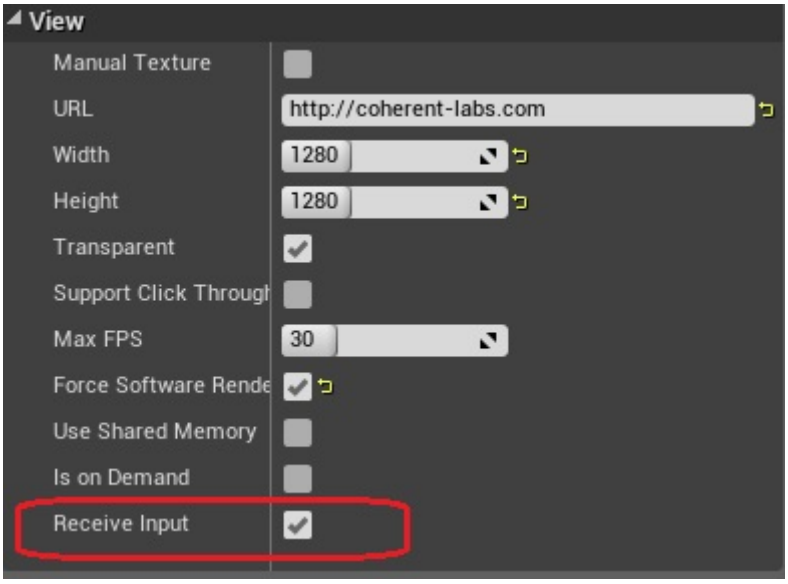


Figure 3.20: Set receive input property from components tab

Input on multiple Coherent UI components in the 3D world

When you have multiple views on 3D objects and they are overlapping when viewed with the current camera, the front one will receive the mouse input events, and the back one will get nothing. If, for example, you have disabled input on the front one, the back one will still get nothing, because 3D objects under the cursor are gathered with a raycast from UE4 and which is independent from the input forward widget's logic. For performance reasons, by default the raycast returns only the first hit object. In a scenario like above, when you have 2 objects and you want to forward input to the back one when the front one is marked as not receiving input, you have to change the raycast type from returning single object to returning multiple objects.

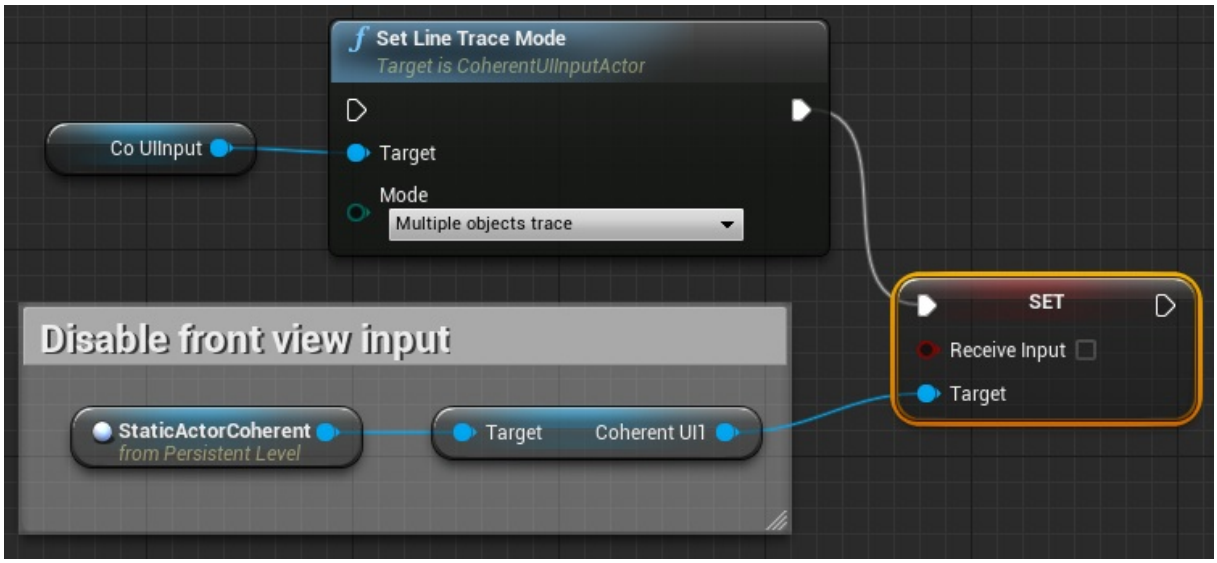


Figure 3.21: Back view takes mouse input

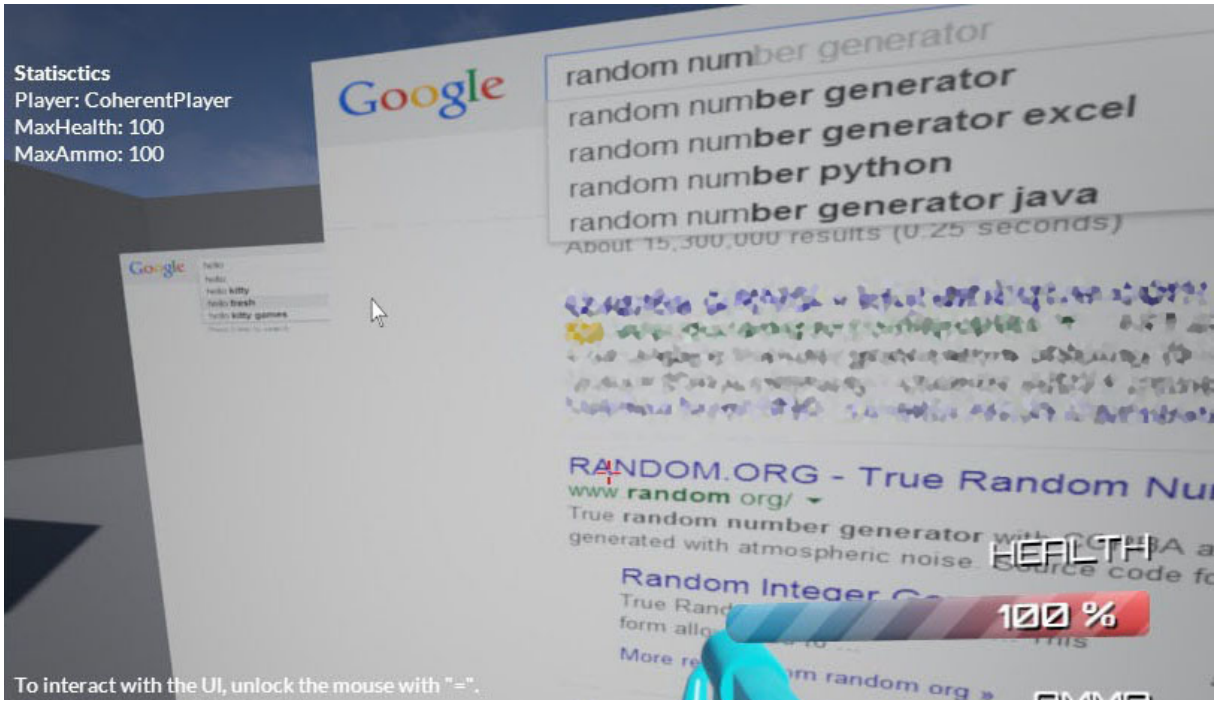


Figure 3.22: Back view takes mouse input

Capturing mouse input on transparent parts of the HUD

It's common in some cases (e.g. in-game menu) that you'd pause your game and show a menu that doesn't cover the whole screen.

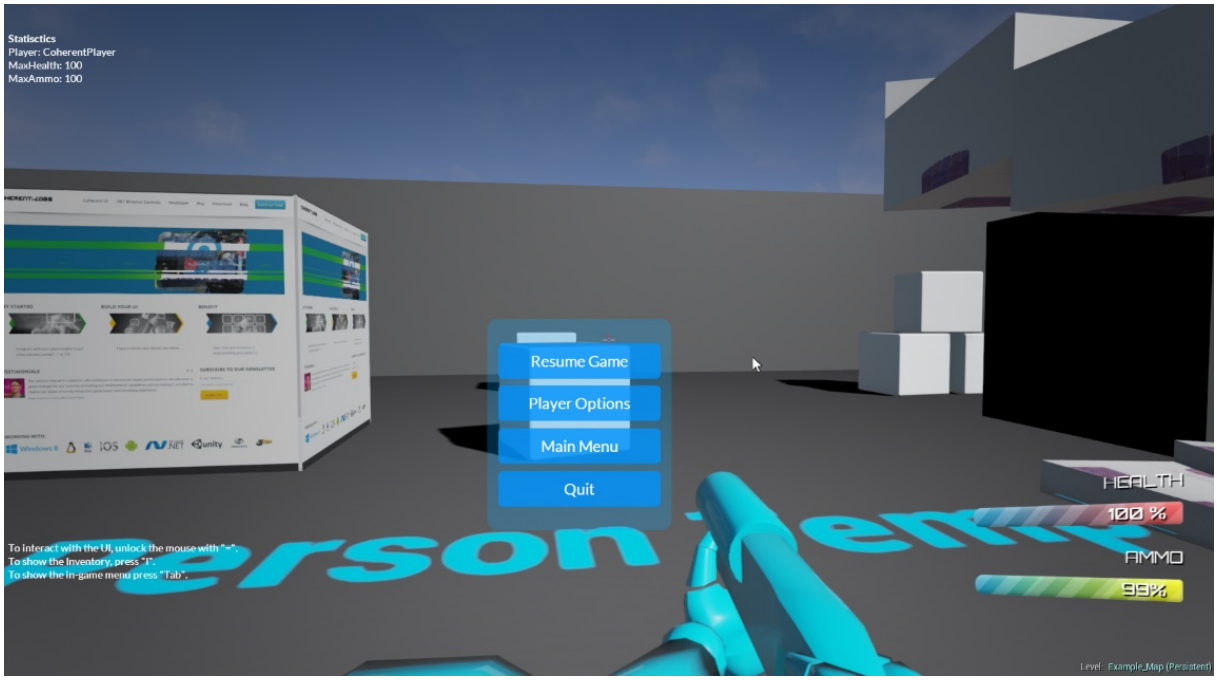


Figure 3.23: In-game menu

With the default implementation of the input widget, mouse events will be passed on the viewport (or other 3D views) when they happen on a transparent area of the HUD. This may not be desired with in-game menus, since the designer would likely want to have the menu focused until the player explicitly resumes the game.

There's a simple solution for that - setting the click-through alpha threshold. This value determines the maximum alpha value of a pixel so it's considered transparent and input is passed on the next handler. If you set that to 0, only completely transparent pixels pass input. If you set that to a negative value, e.g. -1, every pixel is assumed to be solid since its alpha is in the range [0, 1], thus never passing input to other handlers.

In our sample game we have the following code for capturing input when the in-game menu is shown by pressing the *Tab* key (see `ACoUITestFPSCharacter::OnTabReleased`):

```
Coherent::UI::View* View = GetCoherentUIView();
// Set the alpha threshold to a negative value when the in-game menu
// is shown so if you click on a transparent area, you don't return
// to the game.
View->SetClickThroughAlphaThreshold(-1.0f);
```

When the in-game menu is closed in `ACoUITestFPSCharacter::OnCloseIGMenu`, the alpha threshold is set back to the default. If you show the mouse cursor again using the `*=*` key, which toggles game/UI focus, you'll be able to interact with other 3D surface Coherent UI views, unlike when you open the in-game menu.

```
Coherent::UI::View* View = GetCoherentUIView();
View->SetClickThroughAlphaThreshold(0.0f);
```

You can also do this with Blueprints as the `CoherentUIComponent` has the whole `Coherent::UI::View API` exposed.

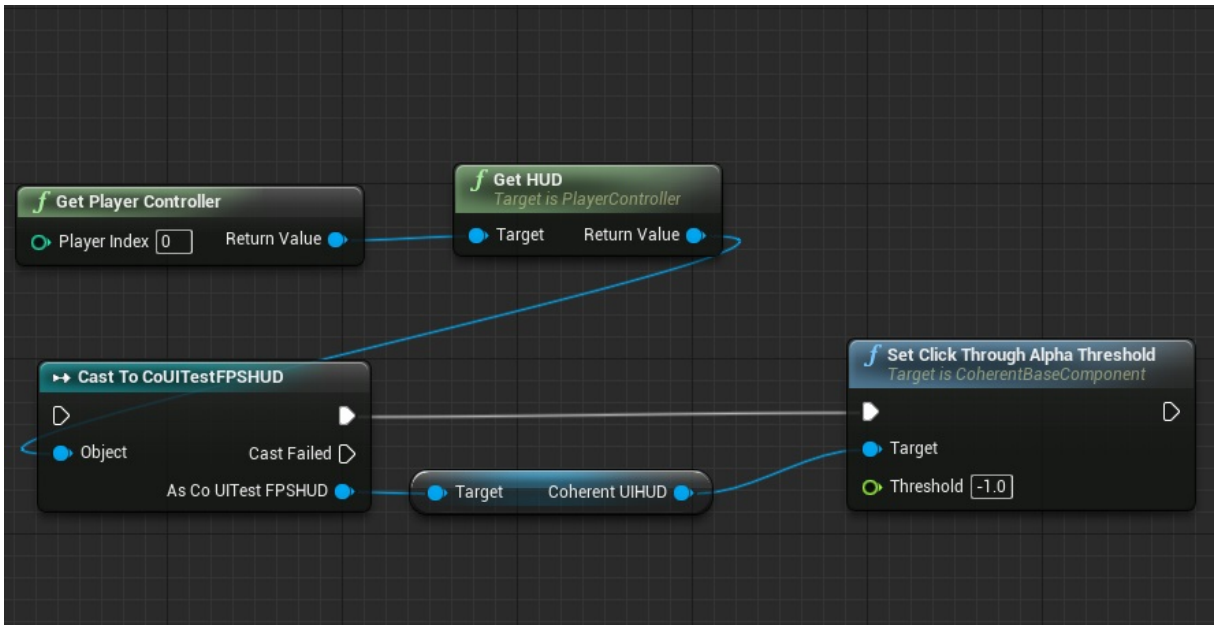


Figure 3.24: Set alpha threshold

3.21 UI Scripting with C++

Note: UI Scripting can be made with Blueprints too. Refer to the *UI Scripting with Blueprints* section for details.

Binding is the Coherent UI name for the communication between the game and the pages JavaScript. In this document we'll use the terms *binding* and *UI scripting* interchancably.

Coherent UI provides a very powerful *binding* framework that can be used to connect the C++ logic of the game with the UI JavaScript code and vice-versa.

A detailed guide on Binding is also given in the C++ documentation of Coherent UI.

To use the Coherent UI Binding feature, your page MUST include the coherent.js file.

To trigger events from C++ to JavaScript you can use the *TriggerEvent* method of the *View*.

```
void ACoUITestFPSCharacter::OnToggleGameMenu()
{
    Coherent::UI::View* View = GetCoherentUIView();
    if (View)
    {
        View->TriggerEvent("ToggleMenu");
    }
}
```

Any parameters you pass to it will also go to the JavaScript code. In JavaScript you must define which function will be called when an event is triggered from C++. The *coherent.js* file exposes a special JavaScript object named *engine* that provides the glue and connections between events in C++ and JS. You must use that object to define all your event handlers. The *on* method will bind an event to a JS handler.

```
engine.on('ToggleMenu', toggleMenu);
```

In this case as soon as C++ calls *TriggerEvent("ToggleMenu")*, the *toggleMenu* function will be executed in JavaScript.

Triggering events from JavaScript to C++ is analogous. First in C++ you need to define what method (or UE4 delegate) will handle a specific event triggered from JS.

Defining C++ handlers **must** happed after the *View* has triggered the *ReadyForBindings* event. If you want to handle events from JS, you must subscribe to it and in its handler declare all your bindings. In the *ACoUITestFPSHUD* Actor in the sample game this is done with the following line:

```
CoherentUIHUD->ReadyForBindings.AddDynamic(this, &ACoUITestFPSHUD::BindUI);
```

Now as soon as the *View* has been loaded it will invoke the *BindUI* method and there you can declare all the bindings for the *View*.

The *BindUI* method is straightforward:

```
void ACoUITestFPSHUD::BindUI(int32 frameid, const FString& path, bool isMain)
{
    CoherentUIHUD->GetView()->BindCall("CallFromJavaScript",
        Coherent::UI::MakeHandler(&CalledFromJSSampleDelegate,
        &(FCalledFromJSSample::ExecuteIfBound)));
    CoherentUIHUD->GetView()->BindCall("CalledFromJSString",
        Coherent::UI::MakeHandler(this,
        &ACoUITestFPSHUD::CalledFromJSStringHandler));
}
```

Essentially we say: "When JavaScript fires the CallFromJavaScript event, in C++ you must execute the CalledFromJSSampleDelegate delegate". The same applies for the second binding but in that case it'll call the *CalledFromJSStringHandler* method of the class.

In JavaScript you just have to use:

```
engine.call("CallFromJavaScript", 123);
engine.call("CalledFromJSString", "Hello from JavaScript!");
```

Note that you can again pass arguments from JS to C++ but the handler signatures *must* coincide with the arguments passed, otherwise an error is generated.

In the sample game the handlers for those methods are very simple:

```
void ACoUITestFPSHUD::CalledFromJSHandler(int32 number)
{
    UE_LOG(LogScript, Log, TEXT("UE4 Delegate called from JavaScript"));
}

void ACoUITestFPSHUD::CalledFromJSStringHandler(const FString& str)
{
    UE_LOG(LogScript, Log, TEXT("String received from JS: %s"), *str);
}
```

We just log that we were called from JS, so that we know everything works fine. Coherent UI support passing parameters to/from JS for all primitive types, STL containers as well as the UE4 FString type. To use those types you must include the relevant headers available in the *Coherent/UI/Binding* folder.

Coherent UI also support binding user defined types. For a detailed guide on that topic please consult the Coherent UI C++ documentation.

3.22 Blueprint integration

Almost all *View* methods have been exposed to the Blueprint visual editing system. Coherent UI Components also trigger dynamic multicast delegates on a variety of events that users can subscribe to and use either from C++ or Blueprint.

A sample of the Blueprint usage is given if you open the Blueprint editor on the *StaticActorCoherent* object in the sample game.

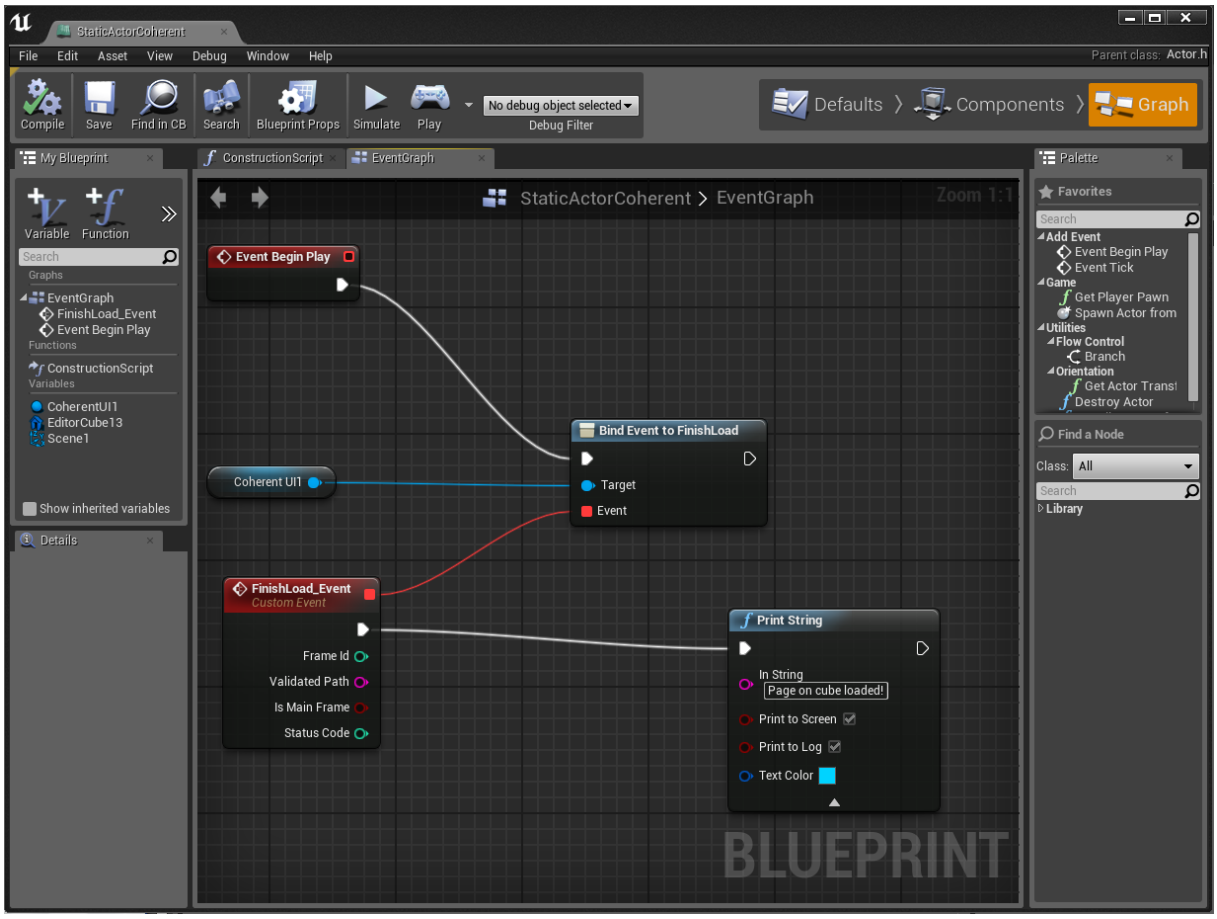


Figure 3.25: Blueprint load

In this case the *FinishLoad* event (a dynamic multicast delegate) of the Coherent UI Component will trigger a *Print String* with the text "Page on cube loaded!". The *FinishLoad* is triggered by the Component when the page it is loading has been successfully loaded. Another event we've already used in the previous section is the *ReadyForBindings* event that we bound through C++.

For a full list of the event exposed by the Coherent UI Components please refer to the *CoherentUIBaseComponent.h* file or review them in the Blueprint editor.

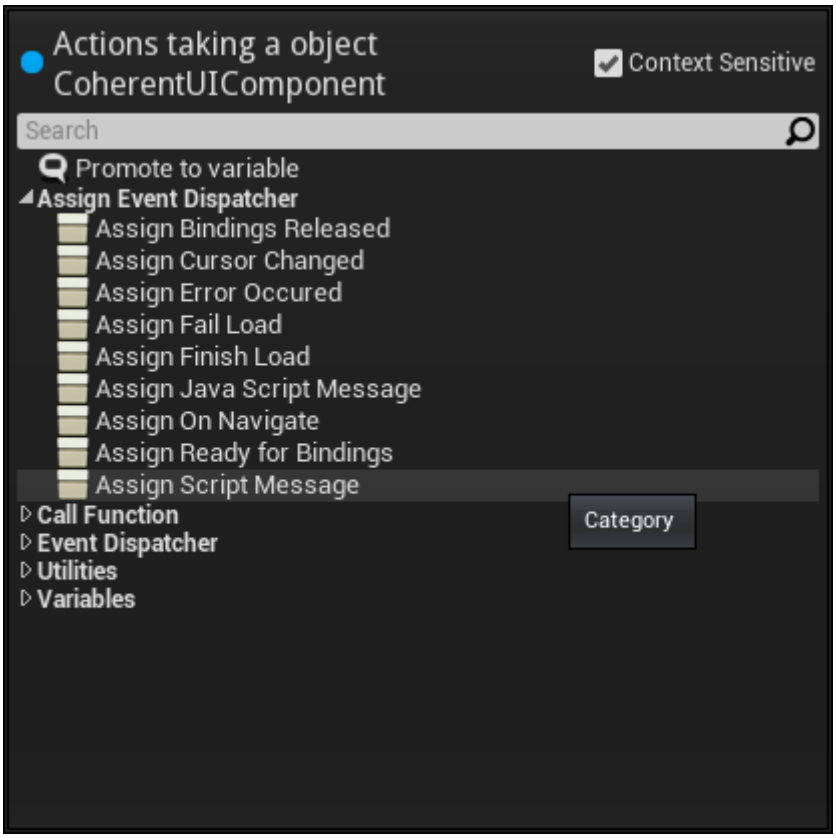


Figure 3.26: Component events

The Components also expose most of the functions of the *View* to Blueprint. With them you can change the current page of the view, resize it, get all its current properties etc. All functions are available under the *View* category.

For a full list of the provided methods please refer to the *CoherentUIBaseComponent.h* file or review them in the Blueprint editor.

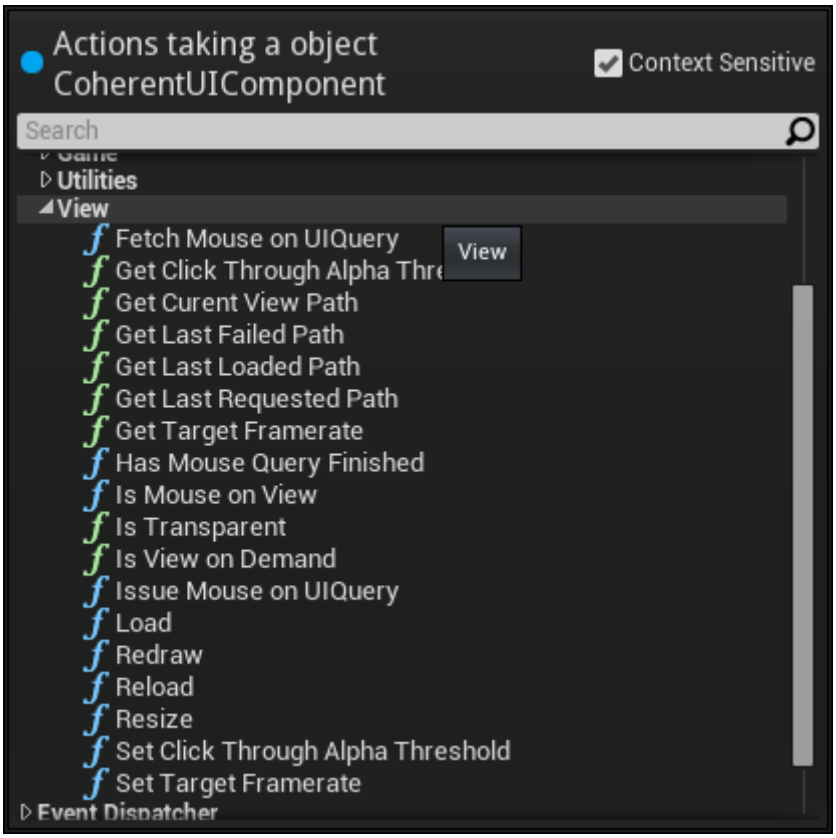


Figure 3.27: Component functions

3.23 UI Scripting via Blueprints

Binding is the Coherent UI name for the communication between the game and the pages JavaScript. In this document we'll use the terms *binding* and *UI scripting* interchancably.

Coherent UI supports scripting the UI via Blueprints. This allows developers to expose game objects to the UI to populate the interface and trigger actions from the UI like starting a new game when the player clicks "Start game" in a menu.

With Coherent UI you can script the whole UI via Blueprints without C++ code.

Developers can trigger events in the UI JavaScript code via Blueprints. They can also listen to events triggered in JavaScript and implement game logic on such events. Developers can expose primitive types and any UObject to JavaScript to populate the UI.

In this section we'll introduce a small example that shows two-way communication between Blueprints and the UI JavaScript. The example is available in the Coherent UI Sample game.

Calling UI scripts from Blueprints

The first example show sending game data to the UI. We'll make available the whole "PlayerCharacter" object to the UI and we'll show on-screen the player name, the max health she can have and the max ammo she can carry. In the "hud.html" page you can see that there is a "playerInfo" element that contains the player data we want to show. We'll populate this data from the game through Blueprints.

1. Open the Example_Map map of the sample game
2. Open the Level Blueprint

3. You'll see the Blueprint already created. You can study it or directly delete it and re-create it better understand what it does.

4. Add a Begin Play Event

5. Add a Get Player Controller node

6. Add a Get HUD Node

7. Add a Cast to CoUITestFPSHUD Node

8. Connect the nodes. Now you have the HUD object for this scene.

9. From the CoUITestFPSHUD object get the Coherent UI HUD View. That is get the Coherent UI HUD property.

10. Drag a pin from the Coherent UI HUD object and select "Assign UI Scripting Ready". The "UI Scripting Ready" event will be triggered when the page is ready to receive events. Note that you MUST add this event manually to your JavaScript code in the UI AFTER all you initial "engine.on" event subscriptions.

11. Now we want to create our event and send our data to the UI. Drag the Coherent UI HUD pin and create a "Create JS Event" node. This node produces the object that will represent our event.

12. Connect the UIScriptingReady_Event node with the "Create JS Event". This means that as soon as the page is ready to receive events, we'll send it one.

13. Add a "Sequence" node and connect it after the "Create JS Event". This node defines the actions that happen after we've created our event.

14. Add a "Get Player Character" node.

15. Drag the "Return value" pin of the "Create JS Event" node and create a "Add Object" node. The JS event object that we just created can carry an arbitrary amount of arguments that will be available to the JavaScript of the page. Each parameter must be added to the event with the appropriate "Add XXXX" node before the event is triggered. The order by which arguments are added to the event will be the order the JavaScript code receives them.

16. Connect the "Get Player Character" return value to the "Object" pin of the "Add Object". This means that our event will send the whole player character object to JavaScript and we'll be able to use it's properties in the UI.

17. Drag the Coherent UI HUD pin and create a "Trigger JS Event" node. This is the node that will effectively execute the event and send it to JavaScript. It must happen after all arguments have been added.

18. Connect the JS event to the "EventData" pin.

19. Set the event name (the "Name" property) to "SetPlayerState".

20. Connect the "Then 0" pin of the sequence to the "Add Object" node and the "Then 1" to the "Trigger JS event" node.

21. Press Play.

The screenshot below shows the complete Blueprint.

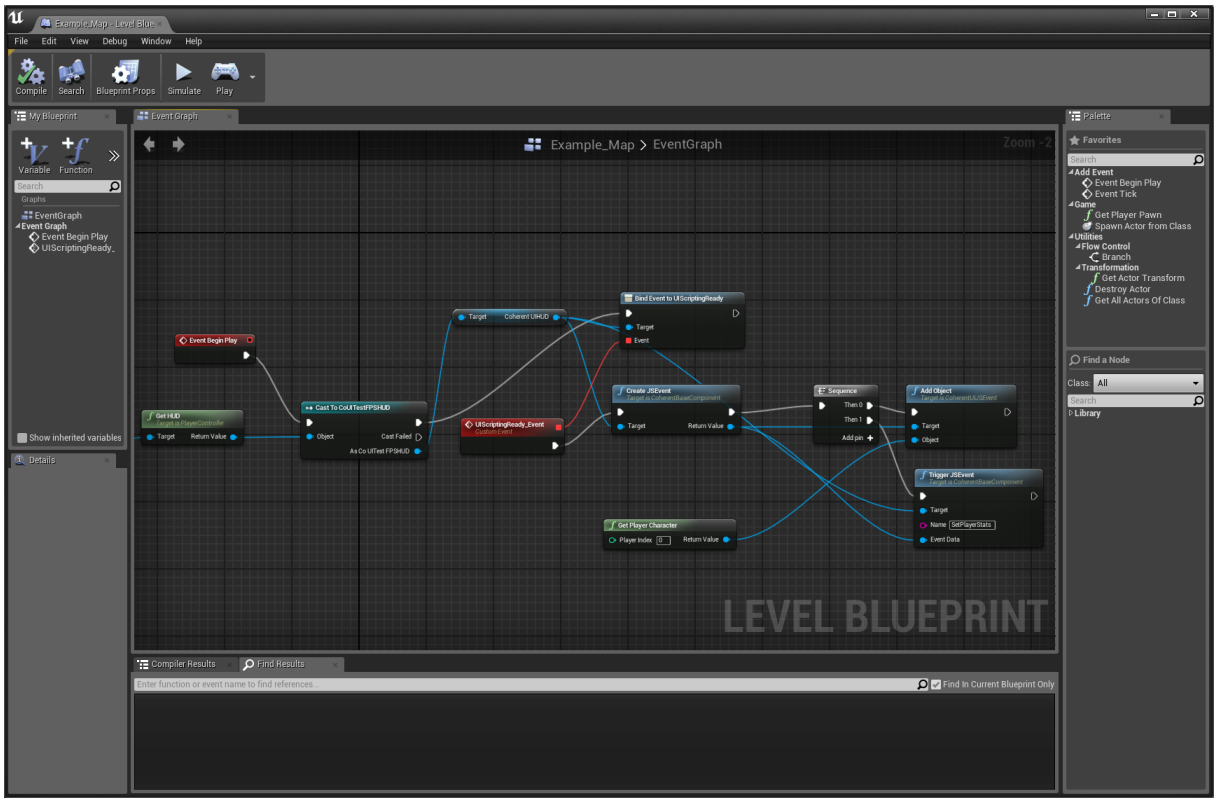


Figure 3.28: Sending events to JS from Blueprints

As soon as the game starts and the HUD loads, the "SetPlayerState" event will be triggered and the corresponding JavaScript code that populates the "Statistics" fields in the UI will execute.

In JavaScript we've added a handler for the "SetPlayerState" event.

```
engine.on('SetPlayerStats', function (character) {
    $("#playerName").html(character.PlayerName);
    $("#playerMaxHealth").html(character.MaxHealth);
    $("#playerMaxAmmo").html(character.MaxAmmo);

    $("#playerInfo").css("display", "initial");
});
// IMPORTANT: Signal the game we are redy to receive events
// We have setup all JS event listeners (the 'engine.on' calls)
engine.call("UIScriptingReady");
```

The "character" variable contains all the properties of the APlayerCharacter actor in the game. Now we can use them to populate the "Statistics" of our UI.

This is how some of the properties of the "character" parameter look in the Coherent UI JavaScript debugger. Their names are the same as in the game.

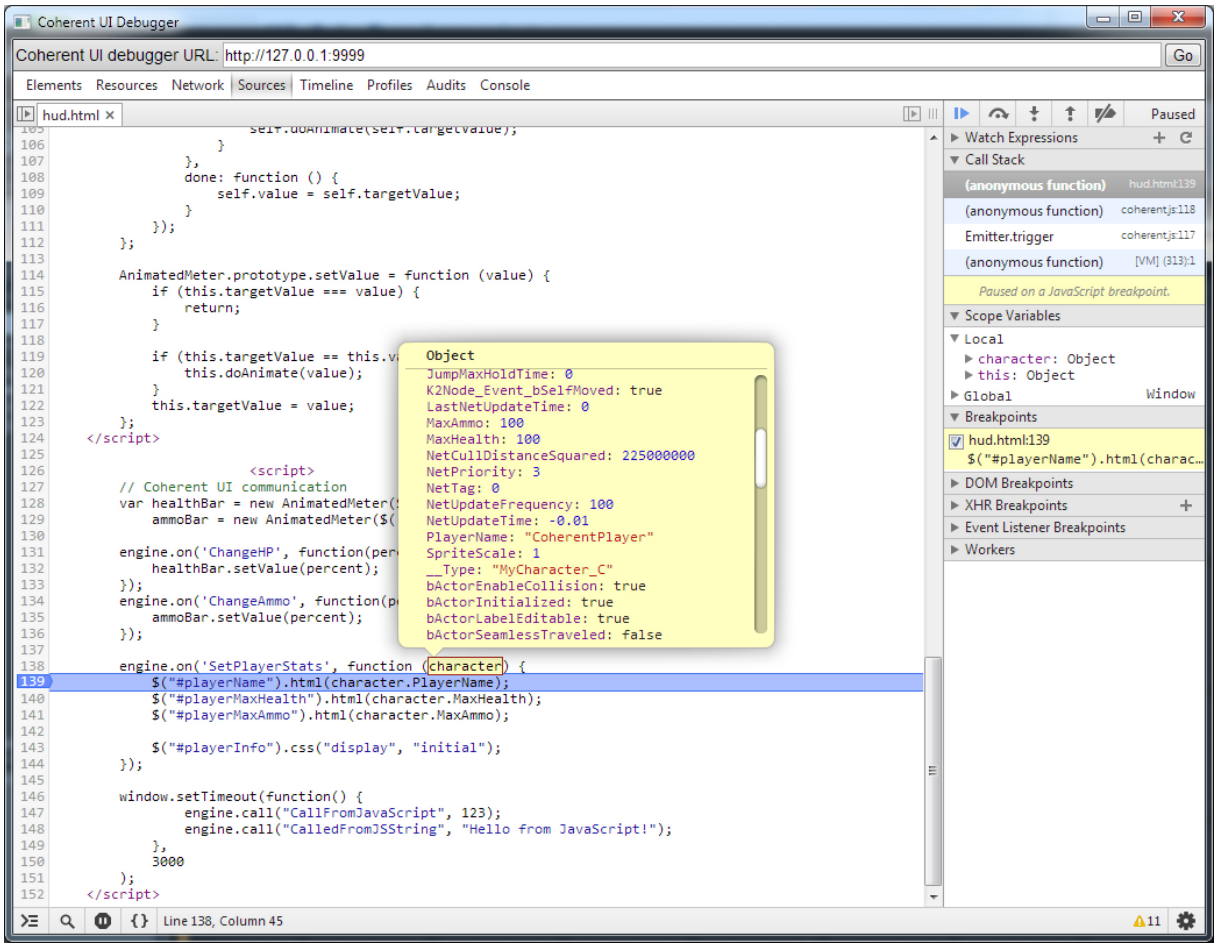


Figure 3.29: The character object in JavaScript

Note that when exporting UObjects, only their primitive type UPROPERTY tagged fields are exported. UObjects contained in exported UObjects are not recursively exported because they might contain circular dependencies and cause memory outages.

The object in JavaScript is a *copy* of the original one, so changing directly its properties won't affect the object in the game. You can however call events back in the game passing parameters and use that mechanism to update logic in the game from the UI. This is explained in the next section.

Calling Blueprints from UI scripts

This section show the other way of communication - from the UI to the game. We'll use the Menu level from the Coherent UI Sample game. When the player clicks on the "Start game" button we'll send an event from JavaScript to the game and change the level.

First let's see what happens in JavaScript when the player clicks "Start game":

```
document.getElementById('play').onclick = function () {
    engine.call('onStartGame', 'Example_Map');
};
```

The code is pretty self-explanatory - when the 'play' button is clicked call an event in the game called "onStartGame". "Example_Map" is a parameter for this event and is the name of the level to load.

Now lets create the Blueprint that will handle this event and actually load the new level.

1. Open the "Menu_Map" level in the Sample game

2. Open the Level Blueprint

3. You'll see the Blueprint already created. You can study it or directly delete it and re-create it better understand what it does.

4. Add a Begin Play Event

5. Add a Get Player Controller node

6. Add a Get HUD Node

7. Add a Cast to CoUITestFPSHUD Node

8. Connect the nodes. Now you have the HUD object for this scene.

9. From the CoUITestFPSHUD object get the Coherent UI HUD View. That is get the Coherent UI HUD property.

10. Drag a pin from the Coherent UI HUD object and select "Assign JavaScript Event". The "Assign JavaScript Event" event will be triggered when the page send an event to the game.

11. Drag a pin from the "JavaScriptEvent_Event" node and assign it a "Switch on String". The "JavaScriptEvent" will fire each time that a call is made from JavaScript. We are however interested in only one type of event - "onStartGame".

12. Drag the pin from the "JavaScriptEvent_Event" Payload property. This is an object of type "CoherentUIJS-Payload" that contains the name of the event fired as well as all the parameters passed by JavaScript. Get the "Event Name" property and connect it to the "Switch on String" node.

13. Add a pin to the "Switch on String" node and set its name to "onStartGame". The node to the JS event node.

14. Create a "Sequence" node and connect it to the "Switch on String". Our sequence will be getting the name of the level passed from JS through the Payload object and then loading a new level with that name.

15. Drag a pin from the "Payload" object and create a "Get String" node. This will be the node that retrieves the name of our level. Coherent UI supports all primitive UE4 types as well as UObjects. Note the "index" property of this node. It tells the node which parameter to select from the ones passed by JavaScript. You can pass multiple parameters, their order is the from left to right as passed in JS starting from 0. If we had a second parameter of type "int" we'd have to add a second "Get Int 32" node with index = 1.

16. Add an "Open Level" node. Connect the pin "Return value" from the "Get String" node to the "Level Name".

17. Play

This is how the final blueprint looks like:

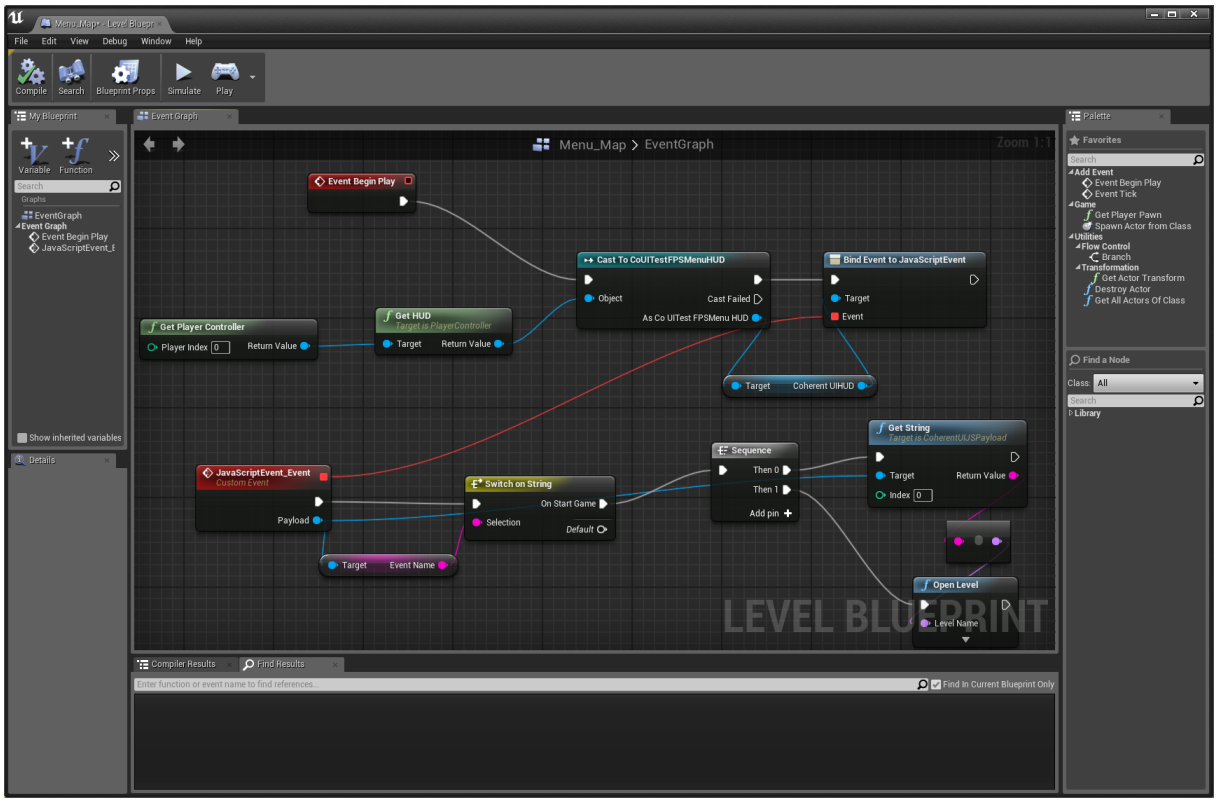


Figure 3.30: JavaScript to BP menu

Remember that Coherent UI supports as many arguments as you need in both directions of scripting and you can send also UObjects both ways - they work automatically.

Triggering *Custom Events* in Blueprints

Coherent UI supports triggering `Custom Events` defined in Blueprints. The event must be declared in the Blueprint that contains the `UCoherentUIComponent` component. To use this functionallity, simply right click on the *Event Graph* and choose *Add Event -> Custom Event* from the menu. Name the event and add any input arguments. Say, the event is *CreateCharacter* and the argument is its name. Here is how the *custom event* looks in the Blueprint editor:

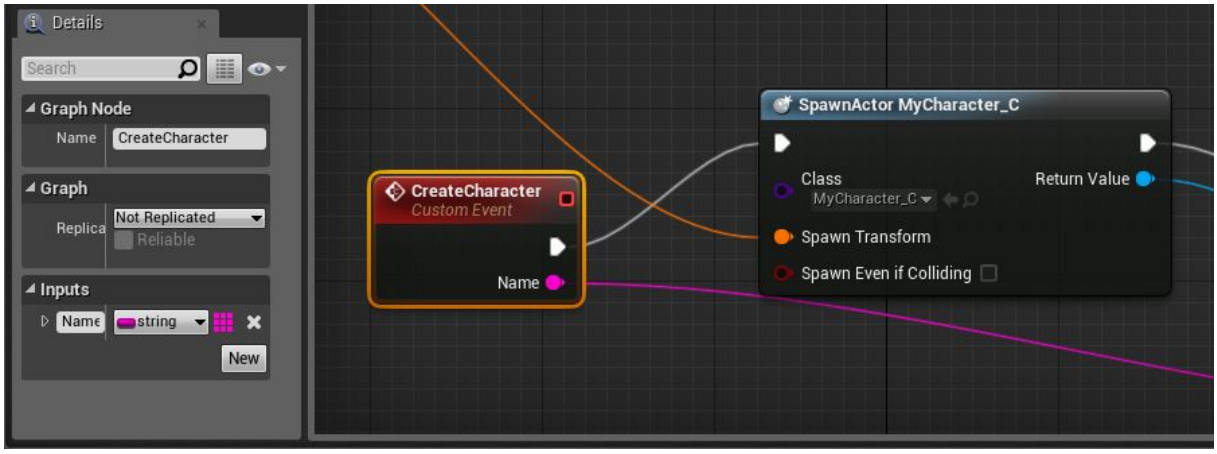


Figure 3.31: Custom event for character creation

Now you can trigger this event from your UI using:

```
engine.trigger('CreateCharacter', 'Joe Monster');
```

Coherent UI will automatically trigger the custom event with all arguments from JavaScript. Please note that currently only primitive types are supported as arguments - `bool`, `byte`, `int`, `string`, `float`.

UObject scripting

Coherent UI supports auto-magic export of UObjects from the game to the UI. Just send a UObject parameter and all its UProperties will be available in the UI page. Only first level properties are sent, UObjects inside other UObjects are not recursively expanded.

You can also updates values in UObjects from JavaScript. To do this create a JavaScript object with the properties you want to change named the same way as in the game. For instance:

```
var mycharacter = {
    PlayerName: "MyJSPlayer",
    MaxAmmo: 800
};
```

Now you can send an event to the game with this object:

```
engine.call("UpdatePlayer", mycharacter);
```

In the game Blueprint when you receive the event you can create a "Read Object" node from the "CoherentUIJSPaylod" object and pass it the APlayerCharacter of your game. All properties that have the same types and names as the ones passed from JavaScript will be updated. You can do this with any UObject. As long as the names and types of the objects coincide, they'll be updated by the "Read Object" node.

Remarks

Currently returning values from the game to JavaScript events is not supported for Blueprints (it is possible with the C++ interface) but will soon be available.

Objects sent to JavaScript are copies of the game versions. Sending smaller objects less often is preferable as it incurs less resource usage.

As a general note the best way to structure the scripting is to create blueprint functions for every interaction and call them when needed from the master Blueprint. This makes the master Event Graph much less cluttered and more readable.

3.24 Coherent UI Debugger

Coherent UI comes bundled with the powerful Coherent UI Debugger. It is modeled after the Chrome dev tools and provides an easy way to inspect elements in the page, debug JavaScript, check the performance of the page. The Debugger allows live editing and debugging of the UI.

You can find the Debugger in the CouiTestFPS/Tools/Debugger folder.

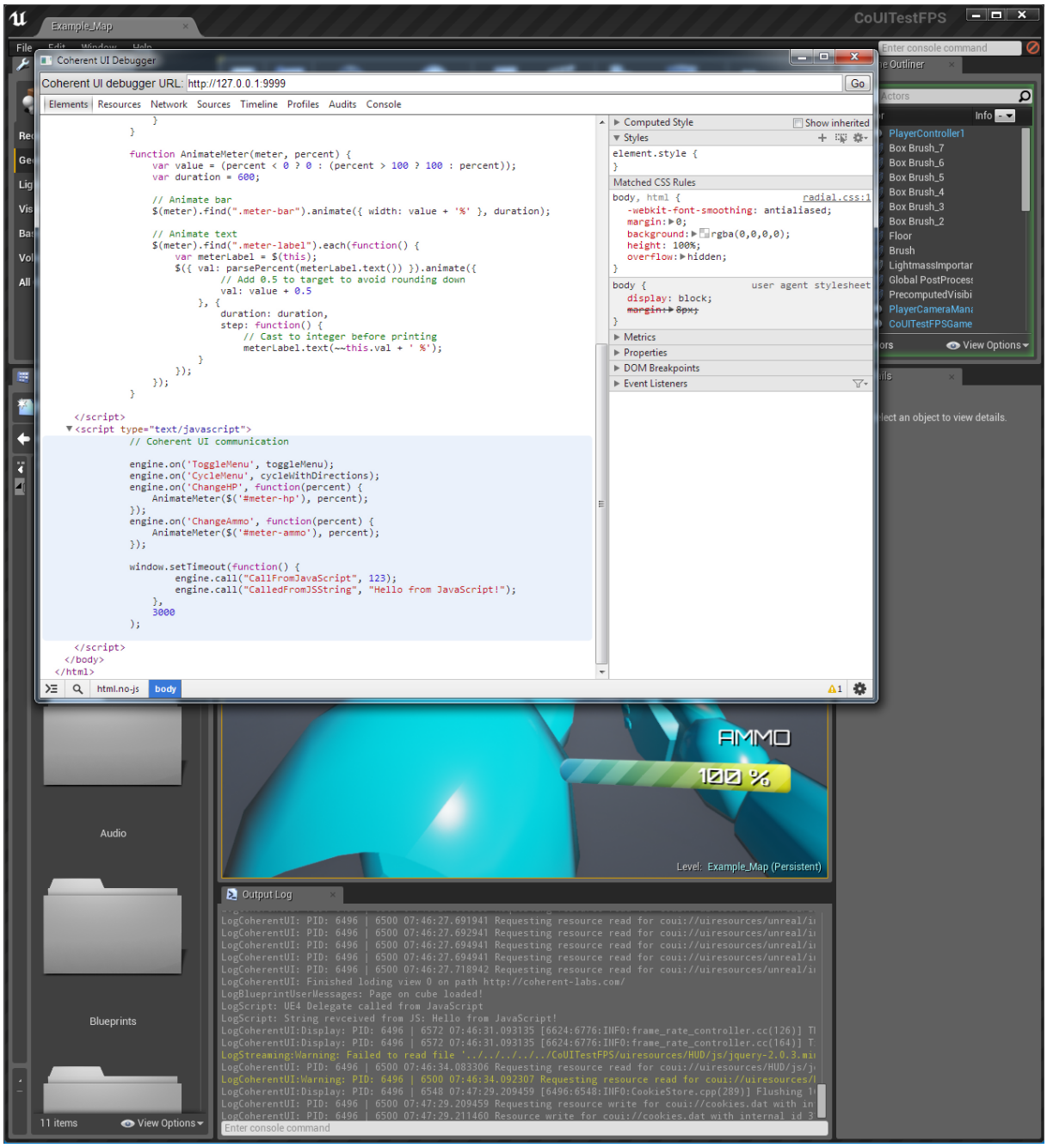


Figure 3.32: Debugger

To use the Debugger start it while your UI is running in the UE4 Editor or in your game. By default the port 9999 is used for the Debugger to connect to but you can change it by modifying the *DebuggerPort* property of the *A-CoherentUISystem* Actor. When connected, the Debugger will offer a list of currently active Views, you have to select the one you want to work on.

The *Elements* tab provides an overview of the page along with the properties and styles of all the elements in the page. When you hover over the elements in the Debugger, they are highlighted in the game as well.

The *Resource* tab shows all the loaded sources and images in the page.

The *Sources* tab can be used to debug JavaScript code. You can put breakpoints and inspect JavaScript variables and code flow.

The *Timeline* tab allows profiling the page. It shows detailed stats on how much time different operations have taken.

The *Profile* tab can be used to profile detailed aspects of the page as the JavaScript execution or the CSS selector. It also allows checking for JavaScript memory wastages.

The *Console* tab shows a JavaScript console that you can use to easily execute code in the page, debug and check properties of objects.

3.25 Coherent UI Menu

Coherent UI Menu allows you to start immediately using Coherent UI for the menus in your game. It also allows to create the menu and implement its logic without leaving the Unreal Engine 4 editor. This menu can be used for a quick prototype, as a substitute for the final menus during development and even for the final game after some styling. You can see the menu in action in the UE4Editor by loading the *MenuBP_Map* level.

To create a menu, you create `CoherentUIMenuInfo` and call `Setup UI Menu` with it and the view that should show the menu. You can create the `CoherentUIMenuInfo` either using the `Make CoherentUIMenuInfo` function or as a varible in the Blueprint. `Setup UI Menu` will return the `CoherentUIMenu` instance you can use to control the menu.

The properties of `CoherentUIMenuInfo` are:

- `Menu ID` - the unique id of this menu in its view. This id is used internally by `CoherentUIMenu`, but can be used from your JavaScript code to manipulate the menu or to listen for its events. See the *JavaScript API* for more details.

- `Parent Element ID` - the id of the element in which to show the menu. You can control the position of the menu by changing the position of the parent element.

- `Visible` - whether the menu is visible by default or not. You can later change the visiblity of the menu using the `Show` and `Hide` methods.

- `Buttons` - the list of buttons in the menu. Please note, that currently the list of the buttons can not be changed after the menu has been created.

Here is how a quit game menu might look like:

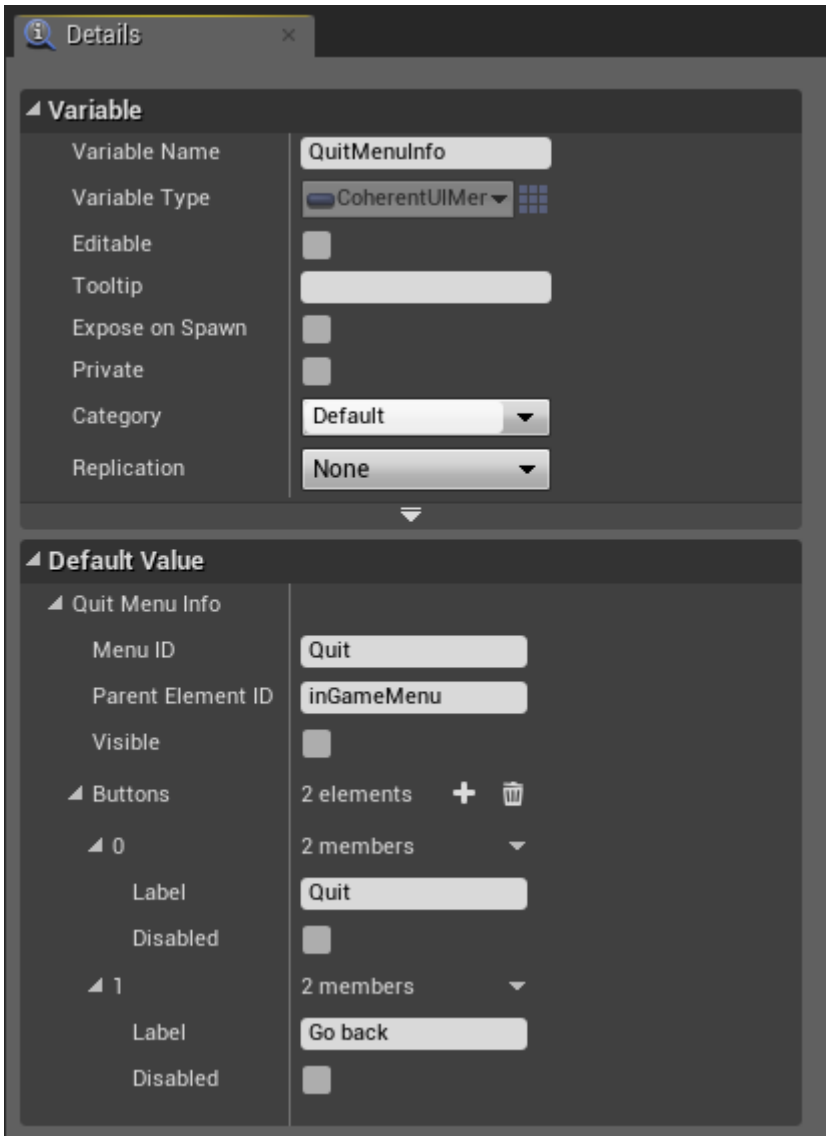


Figure 3.33: Setting up menu in the editor

Binding actions to events

The `CoherentUIMenu` fires the `MenuButtonClicked` event button when a button is clicked passing the label of the click button as an argument. So to handle a click on a menu, you can attach a delegate to the `MenuButtonClicked` event, switch on the button label and call any code you need.

Here is how the final menu looks in the Blueprint editor:

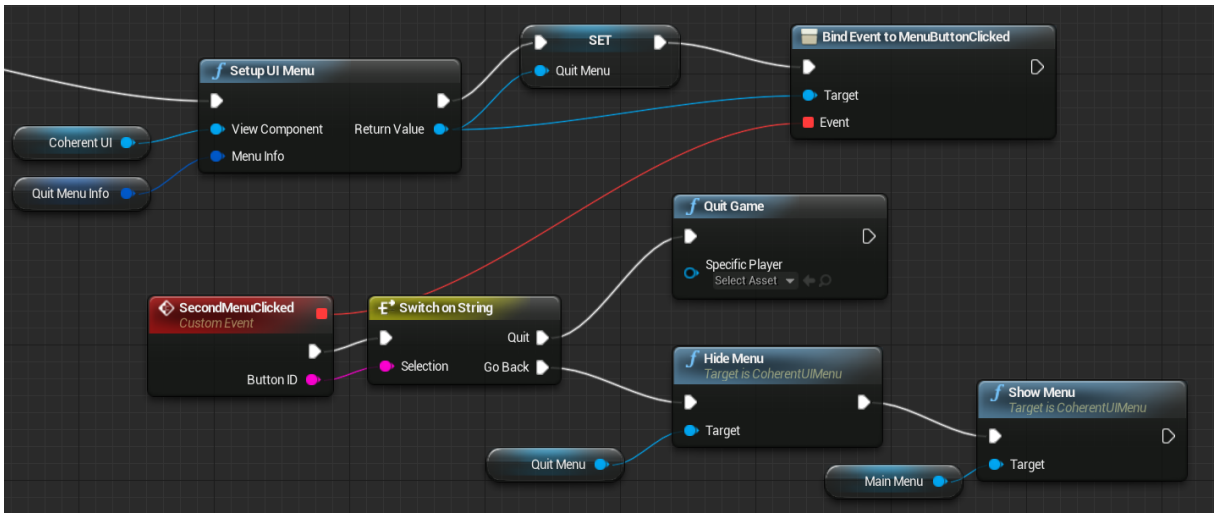


Figure 3.34: Setting up menu handlers in the editor

Setting up the HTML

Using the Coherent UI Menu requires some JavaScript and CSS styles to be present in the view.
The required stylesheets are:

```
<link rel="stylesheet" type="text/css" href="components/flat-ui-official/bootstrap/css/bootstrap.css" />
<link rel="stylesheet" type="text/css" href="components/flat-ui-official/css/flat-ui.css" />
<link rel="stylesheet" type="text/css" href="coherent/css/menu_style.css" />
```

And JavaScript resources:

```
<script type="text/javascript" src="coherent.js"></script>
<script type="text/javascript" src="coherent/js/game_menu.js"></script>
```

Please note that *game_menu.js* depends on *coherent.js*, so it is included after *coherent.js*.

JavaScript API

The JavaScript API of the menu kit consist of the following events:

- `cui.MenuButtonClicked` - triggered on button clicked with the *Id* of the menu and the *Label* of the button
- `cui.ShowMenu` - triggered to show the menu with the menu *Id*
- `cui.HideMenu` - triggered to hide the menu with the menu *Id*

You can attach to and trigger these events using the `engine.on` and `engine.trigger` functions.

3.26 Mac OS X Support

Coherent UI supports running in the UE4Editor on Mac OS X as well as in packaged games.

Currently there is no installator for Mac OS X, but all necessary files are installed by the Windows installator. The easiest way to install Coherent UI on Mac OS X is to transfer the installed files from Windows either by your source control system or by simply copying the files.

Using Coherent UI with source checkout

You need to generate the sample project first. To do so, make sure you add a file named *UE4Games.uprojectdirs* in the root UE4 folder and add a line with the path to the folder, containing the CoUITestFPS folder. After that, run the *GenerateProjects.command* script and after it's finished, open the *UE4.xcodeproject* file. You should have a scheme named *CoUITestFPSEditor*. Select that scheme and build it.

Using Coherent UI with binary release

The sample project is a modification of the First Person C++ template. To run it, start the UE4Editor.app and locate the CoUITestFPS.uproject file in the "Open Project" selection. You can also do that by right clicking on the CoUITestFPS.uproject file in the Finder and opening it with the UE4 version that has Coherent UI installed.

After you open the project, you need to enable the Coherent UI plugin. To do so, go to *Window->*Plugins*, find the entry for Coherent UI plugin and enable it. You'll need to restart the Editor and then you can run the sample.

3.27 Linux Support

Coherent UI supports cross compilation to Linux. After setting up your Unreal Engine 4 for Linux, you can cross compile our sample game for Linux. Due to limitations in the current plugin support of Unreal Engine 4, some additional steps are required to make the game work after being packaged. The steps are:

1. Copy *libCoherentUI.so* from <Engine Directory>/Engine/Source/ThirdParty/CoherentUI/lib/Linux to <Game Directory>/<Game>/Binaries/Linux/ThirdParty/CoherentUI/lib/Linux/
2. Copy the *host* folder from <Engine Directory>/Engine/Binaries/ThirdParty/CoherentUI/Linux to <Game Directory>/Engine/Binaries/ThirdParty/CoherentUI/.
3. Copy the *UIResources* folder as <Game Directory>/Content/uiresources*. Please note that *uiresources* is in lowercase.

3.28 Performance considerations

Coherent UI provides a very fast out-of-process rendering scheme. Each *View* is drawn in it's own process and communication with the game is performed via high-performance IPC. As Coherent UI can be used as an in-game browser, the external processes provide better encapsulation and security for the game. Memory spaces are also cleanly divided.

The asynchronous model guarantees that the UI or web pages will never tax the game too much, so in a situation where computational resources are scarce, the UI might run slower but it will never "steal" resources from the game slowing it down.

By default Coherent UI renders GPU-accelerated. This means that UE4 and Coherent UI have to divide the GPU time between themselves. As UE4 is the main process, it WILL always get the rendering resources that it needs and might cut down the share that Coherent UI requires if GPU resources are scarce. Event though Coherent UI needs very little GPU time, if UE4 consumes all of it, the UI might "starve" and the frame-rate of the UI might decrease. This is perfectly fine in extraordinary situations when UE4 might need all the GPU - for instance when an exceptionally heavy scene is loaded or an effect shown. The UI might slow down a little but the game will run perfectly.

If you notice slow-downs in the UI it might be a case when UE4 is starving Coherent UI on GPU time. Usually this happens when *VSync* is off. Enabling *Vsync* solves any hiccups 90% of time. Enabling *VSync* is usually ok for most games as there is no visually perceivable difference if the game runs on more than 60 FPS (monitors usually refresh at that rate). *VSync* also saves power and hence battery life on laptops. One case when a disabled *VSync* might be required is in competitive multiplayer games where some player prefer to have it off in order to diminish the lag from the input.

Coherent UI provides also a very fast Software renderer. If your game is very heavily GPU bound you can try setting your *Views* to software rendering. This is done through the "Force Software Rendering" property. Note that when in software mode, not all rendering features are available in the *View*. For more information please refer to the C++ documentation.

To force Coherent UI to not let UE4 starve it on GPU resources you can also enable the "Is on Demand" property. In that case UE4 will wait on every frame for Coherent UI to render it's frame.

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

Class Documentation

6.1 ACoherentUISystem Class Reference

`#include <CoherentUISystem.h>`

Inherits AActor.

Public Attributes

- bool AllowCookies
- FString CookiesResource
- bool EnableProxy
- FString CachePath
- FString HTML5LocalStoragePath
- bool ForceDisablePluginFullscreen
- bool DisableWebSecurity
- bool SupportProprietaryCodecs
- bool UpdateWhenPaused
- TEnumAsByte
  < ELoggingEnum::LoggingSeverity > LogSeverity

6.1.1 Detailed Description

This Actor encapsulates a Coherent UI System. If you need to modify any of the default values of the System you can put this Actor in the world and it will be picked up by all created Views. If no Actor is present, a default one will be created.

6.1.2 Member Data Documentation

6.1.2.1 bool ACoherentUISystem::AllowCookies

Allows setting cookies

6.1.2.2 FString ACoherentUISystem::CachePath

Path used for cahced resources

6.1.2.3 FString ACoherentUISystem::CookiesResource

The name of the file used for cookies

6.1.2.4 bool ACoherentUISystem::DisableWebSecurity

Disables web security

6.1.2.5 bool ACoherentUISystem::EnableProxy

Enables support for proxy

6.1.2.6 bool ACoherentUISystem::ForceDisablePluginFullscreen

Disables fulscreen functionality on all plugins

6.1.2.7 FString ACoherentUISystem::HTML5LocalStoragePath

Path for HTML5 local storage

6.1.2.8 TEnumAsByte<ELoggingEnum::LoggingSeverity> ACoherentUISystem::LogSeverity

Verbosity of the Coherent UI log

6.1.2.9 bool ACoherentUISystem::SupportProprietaryCodecs

Enables support for proprietary codecs. NB: You must provide the proper ffmpeg build yourself.

6.1.2.10 bool ACoherentUISystem::UpdateWhenPaused

If true, the system will be updated even when the game is paused

6.2 FMediaDevice Struct Reference

`#include <CoherentMediaRequestHandler.h>`

Public Attributes

- FString Name
- FString Id

6.2.1 Detailed Description

A media device on the system

6.2.2 Member Data Documentation

6.2.2.1 FString FMediaDevice::Id

the device ID

6.2.2.2 FString FMediaDevice::Name

the human readable name of the device

6.3 ICoherentUIPlugin Class Reference

`#include <ICoherentUIPlugin.h>`

Inherits IModuleInterface.

Static Public Member Functions

- static ICoherentUIPlugin & Get ()
- static bool IsAvailable ()

6.3.1 Detailed Description

The public interface to this module. In most cases, this interface is only public to sibling modules within this plugin.

6.3.2 Member Function Documentation

6.3.2.1 static ICoherentUIPlugin& ICoherentUIPlugin::Get ( ) [inline], [static]

Singleton-like access to this module's interface. This is just for convenience! Beware of calling this during the shutdown phase, though. Your module might have been unloaded already.

Returns

Returns singleton instance, loading the module on demand if needed

6.3.2.2 static bool ICoherentUIPlugin::IsAvailable ( ) [inline], [static]

Checks to see if this module is loaded and ready. It is only valid to call Get() if IsAvailable() returns true.

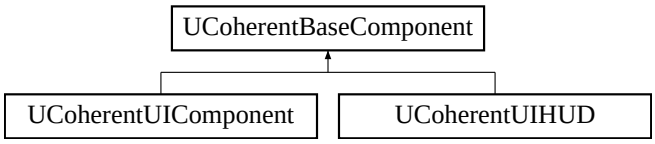
Returns

True if the module is loaded and ready to use

6.4 UCoherentBaseComponent Class Reference

`#include <CoherentBaseComponent.h>`

Inheritance diagram for UCoherentBaseComponent:



Public Member Functions

- UCoherentMediaRequestHandler * GetMediaRequestHandler ()
- void Resize (int32 resizeWidth, int32 resizeHeight)
- void IssueMouseOnUIQuery (float normX, float normY)
- bool HasMouseQueryFinished ()
- void FetchMouseOnUIQuery ()
- bool IsMouseOnView ()
- void SetClickThroughAlphaThreshold (float threshold)
- float GetClickThroughAlphaThreshold () const
- void SetTargetFramerate (int32 target)
- int32 GetTargetFramerate () const
- bool IsViewOnDemand () const
- bool IsTransparent () const
- void Redraw () const
- void Load (const FString &path)
- void Reload (bool ignoreCache)
- FString GetCurentViewPath () const
- FString GetLastRequestedPath () const
- FString GetLastLoadedPath () const
- FString GetLastFailedPath () const
- UTexture2D * GetTexture () const
- UCoherentUIJSEvent * CreateJSEvent ()
- void TriggerJSEvent (const FString &name, UCoherentUIJSEvent *eventData) const
- bool IsReadyForBindings () const
- bool IsReadyForScripting () const

Public Attributes

- FString URL
- bool Transparent
- bool SupportClickThrough
- int32 MaxFPS
- bool ForceSoftwareRendering
- bool UseSharedMemory
- bool IsOnDemand
- bool bReceiveInput
- FUIReadyForBindingsSignature ReadyForBindings
- FUIBindingsReleasedSignature BindingsReleased
- FUIFinishLoadSignature FinishLoad
- FUIFailLoad FailLoad
- FUIOnNavigate OnNavigate
- FUIScriptMessage ScriptMessage
- FUICursorChanged CursorChanged
- FUIJavaScriptMesssage JavaScriptMessage
- FUIOnError ErrorOccured
- FUIOnJavaScriptEvent JavaScriptEvent
- FUIScriptingReady UIScriptingReady
- FUIOnViewCreated UIOnViewCreated

6.4.1 Detailed Description

Base class for all Coherent UI Components

The class exposes all events generated by the Coherent UI View as well as its methods to the Blueprint Editor.

6.4.2 Member Function Documentation

6.4.2.1 UCoherentUIJSEvent* UCoherentBaseComponent::CreateJSEvent ( )

Creates an event that will be executed in JavaScript

6.4.2.2 void UCoherentBaseComponent::FetchMouseOnUIQuery ( )

[DEPRECATED] Waits for the query to finish and fetches its data

6.4.2.3 float UCoherentBaseComponent::GetClickThroughAlphaThreshold ( ) const

Gets the alpha threshold for click-through queries

6.4.2.4 FString UCoherentBaseComponent::GetCurentViewPath ( ) const

Gets the URL that the View is on

6.4.2.5 FString UCoherentBaseComponent::GetLastFailedPath ( ) const

Gets the URL that last failed its loading

6.4.2.6 FString UCoherentBaseComponent::GetLastLoadedPath ( ) const

Gets the URL that was last loaded successfully by the View

6.4.2.7 FString UCoherentBaseComponent::GetLastRequestedPath ( ) const

Gets the URL that was last requested by the user for the View

6.4.2.8 UCoherentMediaRequestHandler* UCoherentBaseComponent::GetMediaRequestHandler ( )

Get the media request handler of the View

6.4.2.9 int32 UCoherentBaseComponent::GetTargetFramerate ( ) const

Gets the current max FPS

6.4.2.10 UTexture2D* UCoherentBaseComponent::GetTexture ( ) const

Gives access to the UI texture

6.4.2.11 bool UCoherentBaseComponent::HasMouseQueryFinished ( )

Checks if the last issued query has finished

6.4.2.12 bool UCoherentBaseComponent::IsMouseOnView ( )

Returns the result of the last query. NB: you must have issued and fetched a query prior to this

6.4.2.13 bool UCoherentBaseComponent::IsReadyForBindings ( ) const

Indicates if functions can be bound to the View

6.4.2.14 bool UCoherentBaseComponent::IsReadyForScripting ( ) const

Indicates if scripts can be executed via View->TriggerEvent()

6.4.2.15 void UCoherentBaseComponent::IssueMouseOnUIQuery (float *normX*, float *normY*)

Issues a query that will determine if the coordinates supplied are on an element in the View

6.4.2.16 bool UCoherentBaseComponent::IsTransparent ( ) const

Gets is the View is transparent

6.4.2.17 bool UCoherentBaseComponent::IsViewOnDemand ( ) const

Gets is the View is on-demand

6.4.2.18 void UCoherentBaseComponent::Load (const FString & *path*)

Requests a new URL to be loaded in the View

6.4.2.19 void UCoherentBaseComponent::Redraw ( ) const

Requests a View to completely re-draw itself

6.4.2.20 void UCoherentBaseComponent::Reload (bool *ignoreCache*)

Requests the View to reload the current URL

6.4.2.21 void UCoherentBaseComponent::Resize (int32 *resizeWidth*, int32 *resizeHeight*)

Resizes the View

6.4.2.22 void UCoherentBaseComponent::SetClickThroughAlphaThreshold (float *threshold*)

Click-through queries check the alpha under the mouse. This value defines the threshold when a pixel is categorized on the UI or on the Game

6.4.2.23 void UCoherentBaseComponent::SetTargetFramerate (int32 *target*)

Sets the maximum FPS that the View will try to achieve. It will never exceed the FPS of the game though.

6.4.2.24 void UCoherentBaseComponent::TriggerJSEvent (const FString & *name,* UCoherentUIJSEvent * *eventData*) const

Triggers an event in JavaScript

6.4.3 Member Data Documentation

6.4.3.1 FUIBindingsReleasedSignature UCoherentBaseComponent::BindingsReleased

The bindings have been released

6.4.3.2 bool UCoherentBaseComponent::bReceiveInput

Indicates whether input is forwarded to this view

6.4.3.3 FUICursorChanged UCoherentBaseComponent::CursorChanged

Called when the page requests that the cursor be changed

6.4.3.4 FUIOnError UCoherentBaseComponent::ErrorOccured

Called when an error happens in the View

6.4.3.5 FUIFailLoad UCoherentBaseComponent::FailLoad

Called when the requested page has failed its load for some reason

6.4.3.6 FUIFinishLoadSignature UCoherentBaseComponent::FinishLoad

Called when the requested page has been loaded completely

6.4.3.7 bool UCoherentBaseComponent::ForceSoftwareRendering

Forces the View to render on the CPU only

6.4.3.8 bool UCoherentBaseComponent::IsOnDemand

On-demand views will synchronize their frames with the game

6.4.3.9 FUIOnJavaScriptEvent UCoherentBaseComponent::JavaScriptEvent

Called an event for this View is triggered in JavaScript

6.4.3.10 FUIJavaScriptMesssage UCoherentBaseComponent::JavaScriptMessage

Called when JavaScript emits a message

6.4.3.11 int32 UCoherentBaseComponent::MaxFPS

Indicates the maximal FPS that the View will try to achieve. It will never exceed the FPS of the game though.

6.4.3.12 FUIOnNavigate UCoherentBaseComponent::OnNavigate

Called when the page moves to a new URL

6.4.3.13 FUIReadyForBindingsSignature UCoherentBaseComponent::ReadyForBindings

When fired, the View is ready for binding events. Any event bound prior to this will be ignored

6.4.3.14 FUIScriptMessage UCoherentBaseComponent::ScriptMessage

Called when JavaScript emits a message

6.4.3.15 bool UCoherentBaseComponent::SupportClickThrough

Views with this property set can make UI click-through queries

6.4.3.16 bool UCoherentBaseComponent::Transparent

Indicates whether the View is transparent or composed on white

6.4.3.17 FUIOnViewCreated UCoherentBaseComponent::UIOnViewCreated

Called when the View object is created and ready for usage

6.4.3.18 FUIScriptingReady UCoherentBaseComponent::UIScriptingReady

Called when the View is ready to accept events

6.4.3.19 FString UCoherentBaseComponent::URL

The URL to load when the View is created

6.4.3.20 bool UCoherentBaseComponent::UseSharedMemory

Forces the View to use shared memory

6.5 UCoherentMediaRequestHandler Class Reference

`#include <CoherentMediaRequestHandler.h>`

Inherits UObject.

Public Member Functions

- void SelectAudioStream (int32 Index)
- void SelectVideoStream (int32 Index)

Public Attributes

- FUIMediaStreamRequest AudioStreamRequest
- FUIMediaStreamRequest VideoStreamRequest

6.5.1 Detailed Description

Handler for user media requests

Exposes events for choosing input audio and video stream.

6.5.2 Member Function Documentation

6.5.2.1 void UCoherentMediaRequestHandler::SelectAudioStream (int32 *Index*)

Choose an audio stream

Parameters

index	the index in the array given to AudioStreamRequest of the device to use.

6.5.2.2 void UCoherentMediaRequestHandler::SelectVideoStream (int32 *Index*)

Choose a video stream

Parameters

index	the index in the array given to VideoStreamRequest of the device to use.

6.5.3 Member Data Documentation

6.5.3.1 FUIMediaStreamRequest UCoherentMediaRequestHandler::AudioStreamRequest

Fired when an audio stream was requests with the list of audio devices

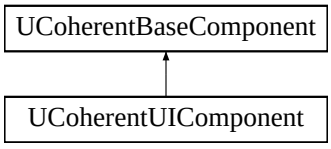
6.5.3.2 FUIMediaStreamRequest UCoherentMediaRequestHandler::VideoStreamRequest

Fired when an audio stream was requests with the list of video devices

6.6 UCoherentUIComponent Class Reference

`#include <CoherentUIComponent.h>`

Inheritance diagram for UCoherentUIComponent:



Public Member Functions

- bool EnsureMeshData ()

Additional Inherited Members

6.6.1 Detailed Description

Component that can be added to actors in the world that have UI textures on them

6.6.2 Member Function Documentation

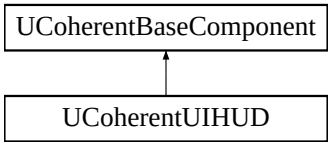
6.6.2.1 bool UCoherentUIComponent::EnsureMeshData ( )

Ensure that MeshData is loaded

6.7 UCoherentUIHUD Class Reference

`#include <CoherentUIHUD.h>`

Inheritance diagram for UCoherentUIHUD:



Additional Inherited Members

6.7.1 Detailed Description

Component that has to attached to HUD actors. It is used to have Coherent UI Views (HTML5 pages) as HUDs.

6.8 UCoherentUIJSEvent Class Reference

`#include <CoherentUIJSEvent.h>`

Inherits UObject.

Public Member Functions

- void AddInt32 (int32 integer)
- void AddUInt32 (uint32 integer)
- void AddString (const FString &str)
- void AddFloat (float fl)
- void AddBool (bool b)
- void AddObject (UObject *object)
- void Clear ()

6.8.1 Detailed Description

The class represents an event triggered by the game that will be processed in the JavaScript of the View. Use this class to pass parameters to JavaScript.

6.8.2 Member Function Documentation

6.8.2.1 void UCoherentUIJSEvent::AddBool (bool *b*)

Adds a Boolean parameter to the event

6.8.2.2 void UCoherentUIJSEvent::AddFloat (float *fl*)

Adds a Float parameter to the event

6.8.2.3 void UCoherentUIJSEvent::AddInt32 (int32 *integer*)

Adds an Integer parameter to the event

6.8.2.4 void UCoherentUIJSEvent::AddObject (UObject * *object*)

Adds an Object parameter to the event

6.8.2.5 void UCoherentUIJSEvent::AddString (const FString & *str*)

Adds a String parameter to the event

6.8.2.6 void UCoherentUIJSEvent::AddUInt32 (uint32 *integer*)

Adds an Unsigned Integer parameter to the event

6.8.2.7 void UCoherentUIJSEvent::Clear ( )

Clears the internal state of the event

6.9 UCoherentUIJSPayload Class Reference

`#include <CoherentUIJSPayload.h>`

Inherits UObject.

Public Member Functions

- FString GetString (int32 index)
- int32 GetInt32 (int32 index)
- uint32 GetUInt32 (int32 index)
- bool GetBool (int32 index)
- float GetNumber (int32 index)
- void ReadObject (int32 index, UObject *object)

Public Attributes

- FString EventName

6.9.1 Detailed Description

The class represents data passed by JavaScript to the application. Use this class to unpach parameters passed from JavaScript to the application.

6.9.2 Member Function Documentation

6.9.2.1 bool UCoherentUIJSPayload::GetBool (int32 *index*)

Gets a Boolean parameter passed from JavaScript

Parameters

index	the index of the parameter

6.9.2.2 int32 UCoherentUIJSPayload::GetInt32 (int32 *index*)

Gets an Integer parameter passed from JavaScript

Parameters

index	the index of the parameter

6.9.2.3 float UCoherentUIJSPayload::GetNumber (int32 *index*)

Gets a Float parameter passed from JavaScript

Parameters

index	the index of the parameter

6.9.2.4 FString UCoherentUIJSPayload::GetString (int32 *index*)

Gets a String parameter passed from JavaScript

Parameters

index	the index of the parameter

6.9.2.5 uint32 UCoherentUIJSPayload::GetUInt32 (int32 *index*)

Gets an Unsigned Integer parameter passed from JavaScript

Parameters

index	the index of the parameter

6.9.2.6 void UCoherentUIJSPayload::ReadObject (int32 *index*, UObject * *object*)

Reads an Object passed from JavaScript and updates the properties of the UObject passed. All properties that have the same name and compatible type will be updated.

Parameters

index	the index of the parameter
object	the object whose properties to update

6.9.3 Member Data Documentation

6.9.3.1 FString UCoherentUIJSPayload::EventName

The name of the event triggered from JavaScript

6.10 UCoherentUILiveViewComponent Class Reference

`#include <CoherentUILiveViewComponent.h>`

Inherits UActorComponent.

Public Member Functions

- void QueueUpdateLiveViewOnRenderThread ()

Public Attributes

- FString LinkName
- bool UpdateEveryFrame
- UTexture * Texture
- bool UpdateWhenPaused

6.10.1 Detailed Description

Coherent UI Live View Component

The class exposes all functionality needed for live views usage in Coherent UI. The component is complementary to the base Coherent UI Component.

6.10.2 Member Function Documentation

6.10.2.1 void UCoherentUILiveViewComponent::QueueUpdateLiveViewOnRenderThread ( )

Manual update of the live view texture

6.10.3 Member Data Documentation

6.10.3.1 FString UCoherentUILiveViewComponent::LinkName

The unique identifier of the live view link

6.10.3.2 UTexture* UCoherentUILiveViewComponent::Texture

The texture whose data will be transferred to Coherent UI

6.10.3.3 bool UCoherentUILiveViewComponent::UpdateEveryFrame

Indicates whether the live view will be updated every frame

6.10.3.4 bool UCoherentUILiveViewComponent::UpdateWhenPaused

```
If false, the live view will not be updated when the game is paused. The update only works
```

with the global setting of the Coherent UI System UpdateWhenPaused set to true

6.11 UCoherentUISettings Class Reference

```
#include <CoherentUISettings.h>
```

Inherits UObject.

6.11.1 Detailed Description

Implements the settings for the Coherent UI plugin.

Index