Coherent UI for Unity3D

1.3.2

Contents

Chapter 1

Introduction

This guide describes the features of the Coherent UI integration in Unity3D. A basic understanding of the Unity3D engine, as well as HTML/JavaScript is assumed. Having basic knowledge of the C++ API and design of Coherent UI may be advantageous. To familiarize yourself with Coherent UI, please read the main documentation file or visit `Coherent UI` website.

1.1 Brief overview of Coherent UI

Coherent UI is a modern user interface middleware solution that allows you to integrate HTML pages built with CSS and JavaScript in your game. The communication between your game and the HTML engine is done through the UI System component. Each HTML page is called a *View*. The *View* component allows you to perform operations on the page, such as resizing, navigating to a different URL, sending input events, executing custom JavaScript code and so on. You can create a view through the CreateView method of the UI System component. It requires you to supply some initialization parameters, such as width, height, initial URL, etc. It also requires an instance of a *View Listener*. *Coherent UI* is highly asynchronous by design, meaning that when you change the URL of a View, for example, the function will return immediately and you will receive a notification when the URL has actually changed. The `ViewListener` is the class that receives such notifications for a specific view - when the URL was changed, the page you're trying to open requires authentication details, etc.

1.2 Differences between Desktop and Mobile version

Coherent UI can be divided conceptually in two libraries - *Coherent UI Desktop* (for Windows & MacOSX) and *Coherent UI Mobile* (for iOS). Due to platform limitations the two have a different subset of features. Namely the Mobile version has some limitations while the Desktop version is fully featured.

Mobile limitations on iOS include:

- You are only able to create 2D views on-top of your game for HUDs or in-game browsers. Views splatted on 3D surfaces in the game world are not supported due to platform limitations on iOS.
- Input management must be implemented through minor changes in the HTML & JS and is not pixel perfect but HTML element-based.
- Bound objects are currently missing from the binding
- No on-demand views and frame-rate control

Other that that the API has been kept 100% compatible between the Desktop & Mobile versions. The Core binding, Resource management and View management are the same. *Coherent UI Mobile* supports both device builds and simulator ones.

Chapter 2

Installation

Coherent UI for Unity3D is distributed in a **unitypackage** file. You can import this package in your project by either double-clicking on it, or by importing it through Unity in the *Project* window.

After importing the package, you have to **install** the assets provided. This is done by the *Assets* → *Coherent UI* → *Install Coherent UI* menu entry.

2.1 Prerequisites

To run *Coherent UI* on Windows you need to install the *DirectX End-User Runtimes (June 2010)*. You can download the runtimes from this URL http://www.microsoft.com/en-us/download/details.aspx?id=8109 .

Warning

Not having the runtimes installed will cause missing DLL errors.

Coherent UI Mobile supports iOS 5.1 and above.

2.2 Package contents and structure

The package has the following structure:

- *CoherentUI* - contains the UI debugger, samples, and documentation. Due to specifics of the Asset Store publishing tools, it also contains all the other Coherent UI assets, which need to be installed before usage. This is done by clicking the *Assets* → *Coherent UI* → *Install Coherent UI* menu entry. Coherent UI∗ menu entry.

The following are initially placed in the *CoherentUI* folder and moved by the install script:

- *Editor* - contains editor classes for displaying the properties of *Coherent UI* components plus a post-build step class. These classes provide utility functionality.
- *Plugins* - contains the Coherent UI libraries. These are automatically copied when building.
- *Standard Assets/Scripts/CoherentUI* - contains the Coherent UI integration classes. You'll find a Detail folder inside which contains classes that are internal for the implementation and are used by the "public" classes. The interface you should be using is outside the Detail folder.
- *StreamingAssets* - contains assets that need to be copied as-is in the build directory. These include the Coherent UI host process executable, the libraries it needs, locales and UI resources.

2.3 Usage of the package

After importing the *CoherentUI.unitypackage* in your project, the two main scripts you'll be using are *Standard Assets/Scripts/CoherentUI/CoherentUISystem* and *Standard Assets/Scripts/CoherentUI/CoherentUIView*. The easiest way to use Coherent UI is to drag the `CoherentUIView` component onto an object and hitting Play - that's it! Everything will be up and running. Actually, you can do the same for most of the usage scenarios - just drag the component and then configure it in the inspector. Here's a bit more detail about the two scripts.

The first script, `CoherentUISystem`, defines initialization parameters of the Coherent UI System and should be placed no more than once in your project. The UI system is meant to be initialized in the first scene and live throughout the game's lifetime. You need to add this component to your scene only if you need custom initialization of the Coherent UI System. For the most cases, using only CoherentUIViews is enough - they will automatically create an instance of the Coherent UI System for you with reasonable default parameters. Check CoherentUISystem Lifetime for details.

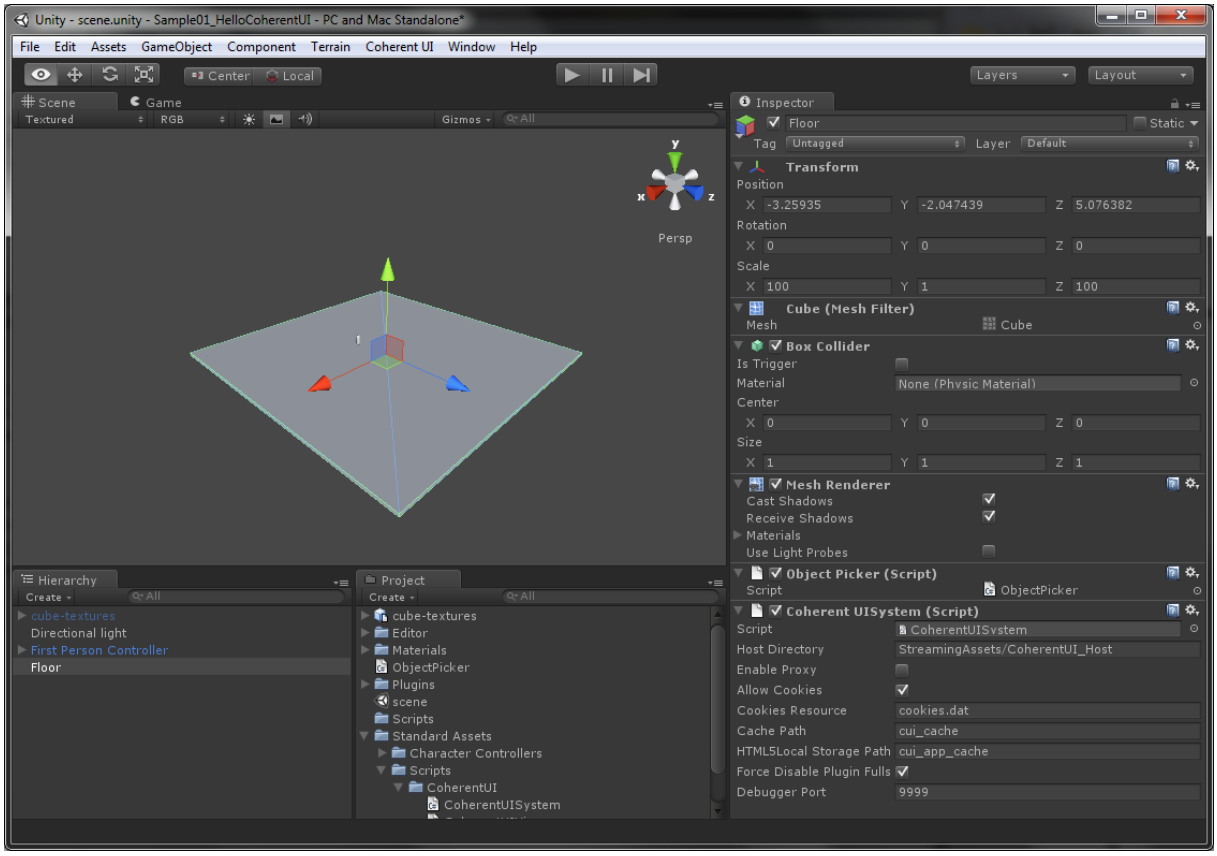

Figure 2.1: CoherentUISystem properties

The second script, `CoherentUIView`, will represent a single HTML page. This is the component that renders your CSS and JavaScript animations and makes your game alive. This component can be placed on any object that is renderable and serves as its material. When placing it on an object all the needed components are automatically created, hidden from you, and the rendered output is bound to the mainTexture of a new material that is created at runtime. This material is set as the gameObject's renderer material so that you see the page rendered on your object.

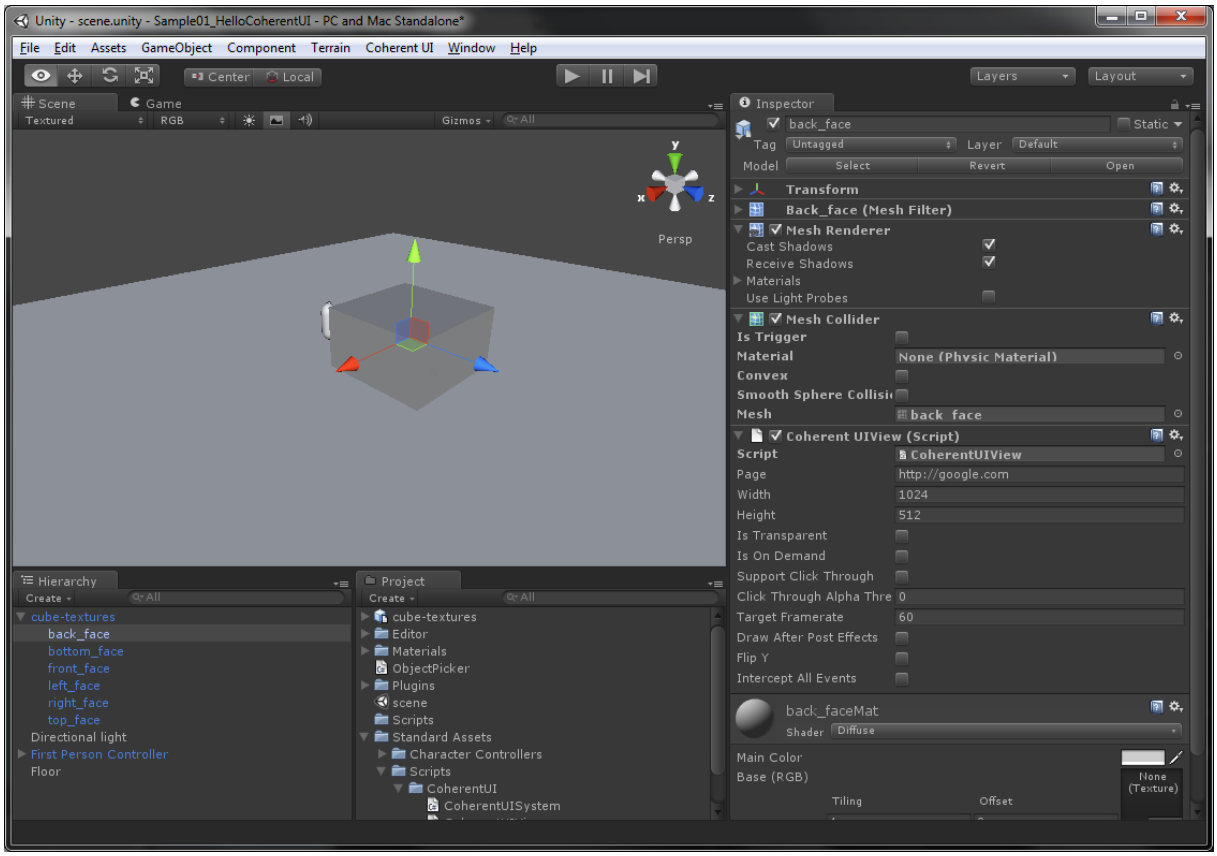

Figure 2.2: CoherentUIView properties

You can also place views on cameras. When doing so, you get your page rendered on the whole viewport of the camera. This way you can easily add a heads-up display for your game. Don't forget to mark your HUD as transparent and make your HTML page transparent! There's even an option for drawing the HUD after the post-effects if you like (The option is available for non-HUDs, too, but it results in a no-op because it doesn't make sense).

2.4 Switching between build targets

Coherent UI Mobile has different properties and a restricted subset of features compared to the desktop versions. To switch between the desktop features and the mobile ones you must switch the current active target in Unity. This is done by clicking on File->Build Settings selecting the target platform and clicking "Switch Platform". After doing so the properties of the Views will change to reflect their capabilities specific to the selected platform. Even if the current target is iOS and hence you'll build with *Coherent UI Mobile* you can still preview your game in the Editor. The preview will use the desktop version of *Coherent UI*. Keep in mind that in this case you might see some differences compared to the the mobile device after the build as the desktop version has more features. You should check often how your UI behaves and looks on the device itself (or the simulator).

Chapter 3

Samples

Note

Some samples show Desktop-specific features while others focus on Mobile targets

The samples provide a starting point for you. They are located in `Assets/CoherentUI/Samples` and the required scripts are already configured.

Note

Make sure you have installed the Coherent UI package using the `Assets` → `Coherent UI` → `Install Coherent UI` menu entry before trying to run them.

The samples are based on a simple scene we've set up for you. The scene consists of a light, a floor, a cube, and a FPS controller so you can move around. There are a few key points in the base sample you should be aware of.

- The FPS controller's `MouseLook` script has been modified a bit - you can toggle if it's active using the **Alt** key. This was done for convenience so you can navigate over the page, projected on the cube easily, without looking around. The CharacterMotor script has also been modified for the same reason.
- The cube's faces all have `MeshCollider` components. They are needed by Unity to produce texture coordinates in the raycast hit info. The texture coordinates, in turn, are needed to calculate the position of the hit point on the Coherent UI View.
- The scenes that have interactive Coherent UI surfaces placed on objects use the *Click-to-focus* feature of the Coherent UI View. It allows the user to forward input to the view when she clicks on it and stop forwarding it when she clicks outside.

Note

In the case where the mesh collider does not coincide with the actual geometry of the object, the raycast and coordinate calculation must take place in the user code. In the samples we're only using simple geometry that is the same as the mesh collider so no further work is needed.

3.1 Hello, Coherent UI

Just hit play - and you'll see google. To stop looking around when you move the mouse, press the **Alt** key. You can toggle that any time. Now feel free to search for anything you like :). You can change the dimensions of the View or the Page URL while playing - try changing it to something interesting, for example `http://hexgl.bkcore.com/` (also change the resolution to 1280x720 to be more natural).

You can also add a *CoherentUIView* component to one of the other sides of the cube. To do so, locate the *CoherentUIView* script under **Standard Assets/Scripts/Coherent UI/CoherentUIView** in the *Project* window. In the Hierarchy window, expand the cube-textures object and drag the *CoherentUIView* script on the back-face sub-object in cube-textures.

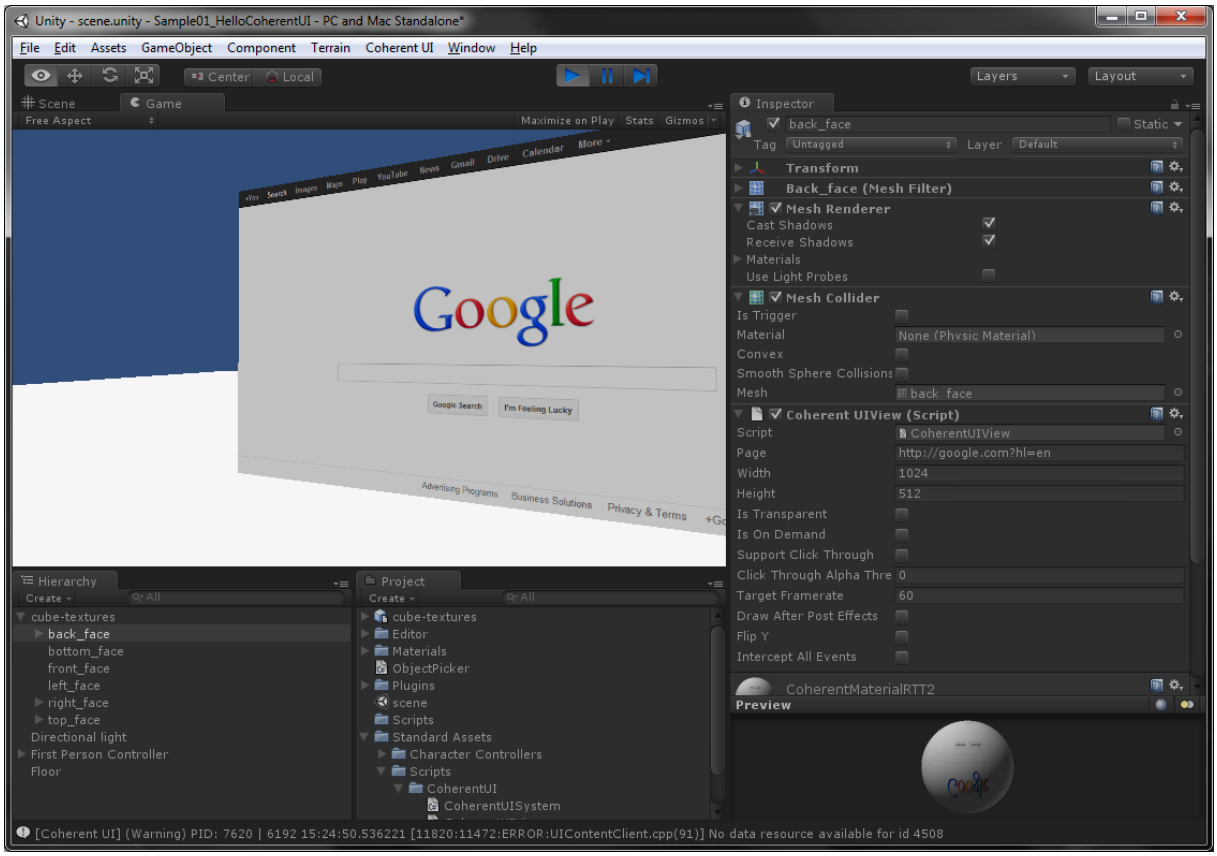

Figure 3.1: CoherentUIView on a texture

You can also try pages that show dialog boxes, such as

- http://www.w3schools.com/js/tryit.asp?filename=tryjs_alert
- http://www.w3schools.com/js/tryit.asp?filename=tryjs_confirm
- http://www.w3schools.com/js/tryit.asp?filename=tryjs_prompt
- http://www.httpwatch.com/httpgallery/authentication/ (scroll down and click Display Image)

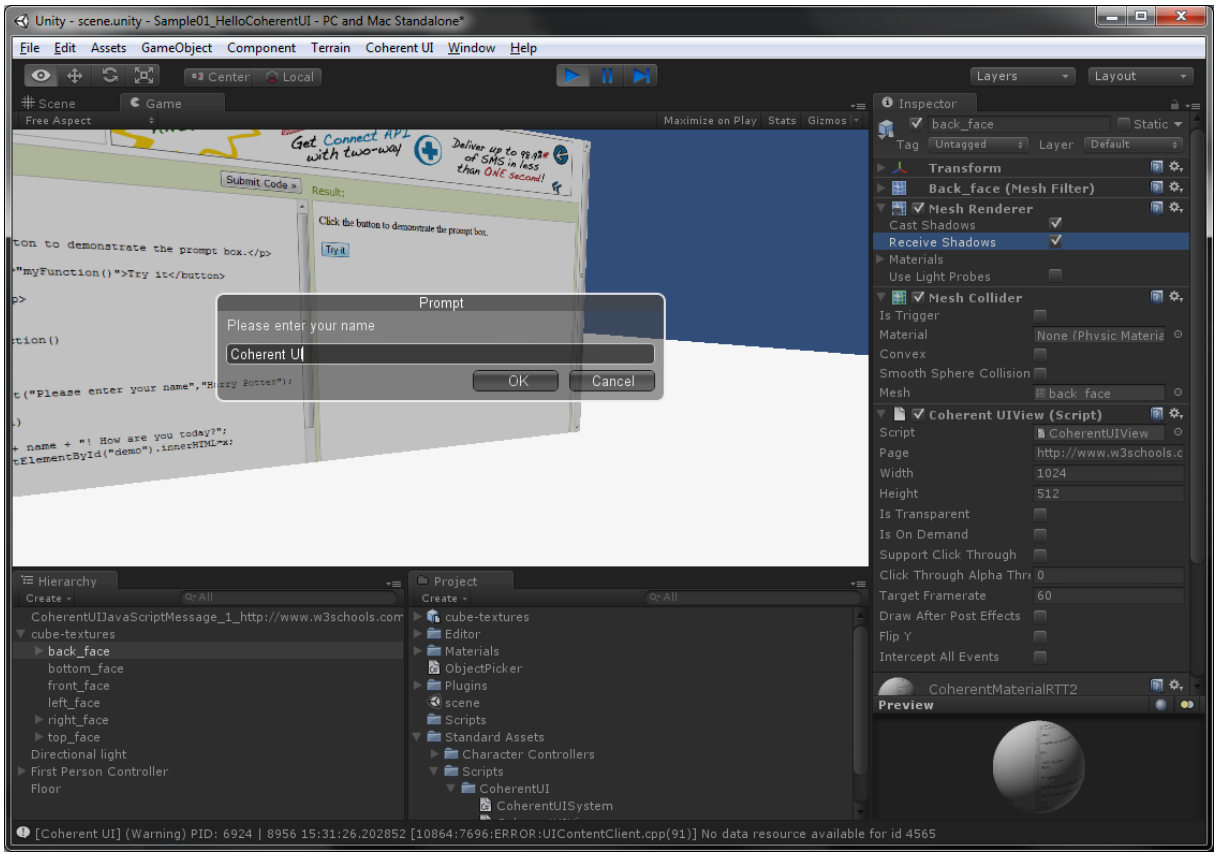

Figure 3.2: CoherentUIView showing a prompt

3.2 Facebook integration

This sample uses a facebook application that shows some of your photos in a rotating circle. You'll see how to customize the `View` and `ViewListener` behaviour to suit your needs.

Note

If you're having trouble displaying the local resources, make sure you have selected the *UIResources* folder located in the root of the project using the *Edit* → *Project Settings* → *Coherent UI* → *Select UI folder* menu in the editor. This folder will be used for resources addressed with `coui://` links.

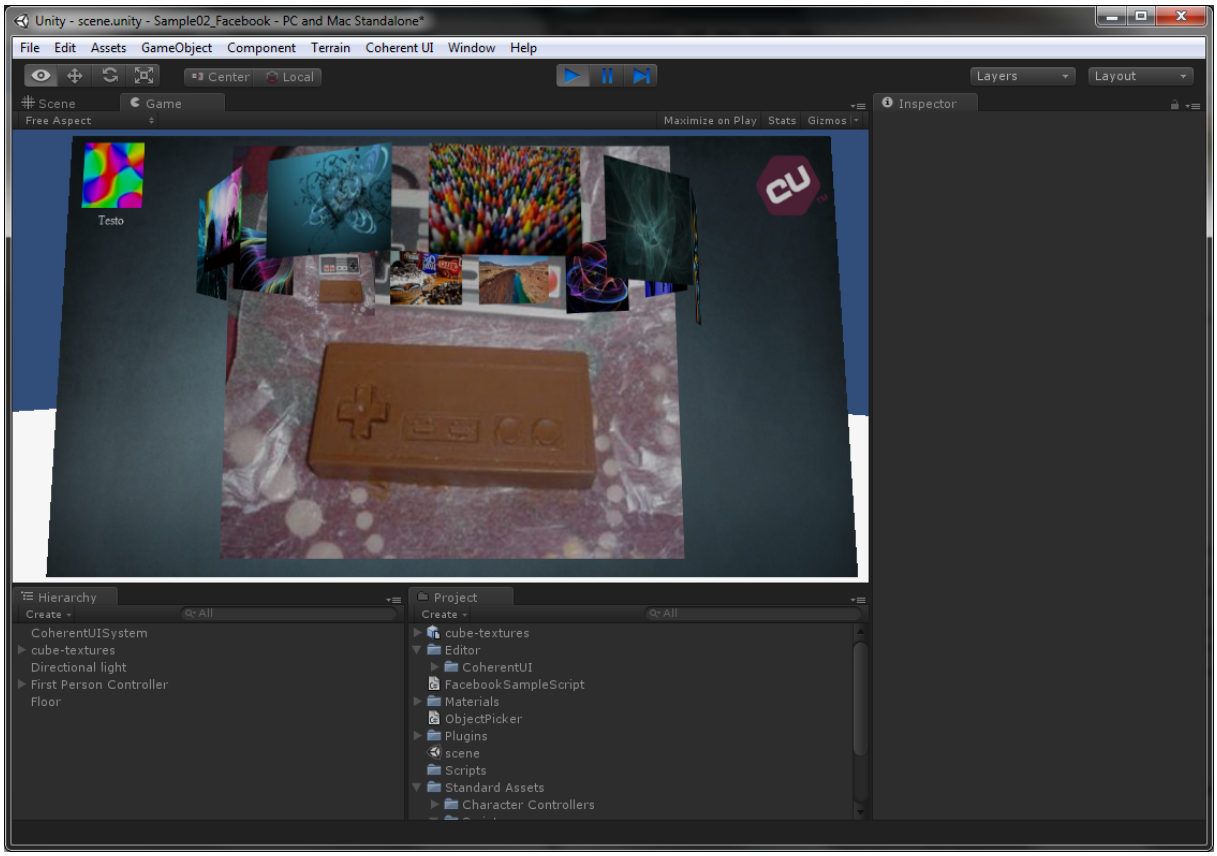

Figure 3.3: CoherentUIView Facebook integration sample

You will find a new script - `FacebookSampleScript`. We need to modify the default behaviour of `UnityViewListener` for this sample. The key points are:

- In its `Start` method the `FacebookSampleScript` obtains the `CoherentUIView` component in the game object.
- In the `Start` method the script also attaches to the `CoherentUIView.OnViewCreated` event to enable intercepting URL requests and to the `UnityViewListener.URLRequest` event to redirect the Facebook login URL to the local HTML page.

This is the easiest way to handle events provided by the `UnityViewListener`. Another option is to derive the `UnityViewListener` and `CoherentUIView` but that is unnecessary.

The `FacebookSampleScript.OnURLRequestHandler` checks if the requested URL is the one that was registered in the *Facebook Application settings* (this might be any made up URL). If that's the case, we redirect it to the local page that uses our Facebook application. You will find the page we're redirecting to in the sample project folder under *FacebookSample/facebook.html*.

Now that we've explained what the class does, the only thing left is to hit play. You can press **Alt** to stop moving/looking so you can browse in the Facebook application.

3.3 Menu And HUD

This sample shows a game menu and HUD built with *Coherent UI*.

Note

If you're having trouble displaying the local resources, make sure you have selected the *UIResources* folder located in the root of the project using the *Edit* → *Project Settings* → *Coherent UI* → *Select UI folder* menu in the editor. This folder will be used for resources addressed with `coui://` links.

Double-click the *Menu* scene in the Project window (located in *Scenes*) to make it active, if it isn't already. In the *Menu* scene we have a `MenuScript` component attached to the camera. `MenuScript` does three things:

- sends the mouse position to the view every `Update()`
- registers the handlers for clicking on a menu button
- loads the game when the "New Game" button is clicked

Note

You have to add the *game* scene to the build settings of your project so Unity3D can load it.

In the game, we have another component for controlling the view - `HUD`, this time implemented in *UnityScript*. Each frame the component updates the compass orientation, based on its transformation.

In addition to updating the compass, the HUD component takes care of disabling the `CharacterMotor` component when the focus is on a *Click-to-focus* view. The HUD script attaches to `CoherentUISystem::OnViewFocused` event and changes the `canControl` property of the selected `CharacterMotor` whenever a *Click-to-focus* view gains or loses focus. This allows typing in the focused view without moving the character. Here is the HUD component in code:

```
#if !UNITY_3_5 || !UNITY_3_4
// This preprocessor condition always evaluates to true.
// The #if block is needed for successful compilation of this script before installation.
// After installing, the CoherentUIView script is moved to Standard Assets and is compiled first,
// so it can be referenced by other scripts of different language, in this case HUD.js.

// This preprocessor block should contain all the public fields of the real script so they can be assigned.
public var characterMotor : CharacterMotor;
#else
#pragma strict
#if !UNITY_IPHONE || UNITY_EDITOR
import Coherent.UI.Binding; // to use View.TriggerEvent with extra arguments

private var View : Coherent.UI.View;
private var CurrentDirection : float;

// CharacterMotor component to be disabled when a Click-To-Focus view has gained focus
public var characterMotor : CharacterMotor;


function Start () {
    var viewComponent = GetComponent(typeof(CoherentUIView)) as CoherentUIView;

    viewComponent.OnViewCreated += this.ViewCreated;
    viewComponent.OnViewDestroyed += this.ViewDestroyed;

    CurrentDirection = 0;

    var uiSystem = Component.FindObjectOfType(typeof(CoherentUISystem)) as CoherentUISystem;
    // get notified when a Click-to-focus view gains or loses focus
    uiSystem.OnViewFocused += this.ViewFocused;
}

function ViewCreated(view : Coherent.UI.View) {
    View = view;
    var viewComponent = GetComponent(typeof(CoherentUIView)) as CoherentUIView;
    Debug.LogWarning(String.Format("View {0} created", viewComponent.Page));
}

function ViewDestroyed() {
    View = null;
}

function ViewFocused(focused : boolean) {
    if (characterMotor) {
        // enable or disable the character movements
        characterMotor.canControl = !focused;
```

```
    }
}

function Update () {
    if (View != null)
    {
        var direction = this.transform.rotation.eulerAngles.y;
        if (Mathf.Abs(direction - CurrentDirection) > 2)
        {
            View.TriggerEvent("SetAbsoluteCompassRotation", direction);
            CurrentDirection = direction;
        }
    }
}
#endif

#endif
```

In the game scene we have one more component - `ObjectPicker`, that takes care of directing the input to the correct `CoherentUIView`. First it checks whether the mouse is on a HUD element and if it is not, then raycasts and checks whether the mouse is over the browser window. Make sure the "Support Click Through" checkbox is ticked on the HUD `CoherentUIView` - this allows you to detect whether the mouse is over a transparent area in the View. If the property is left unchecked the input will never reach any objects in the world, because the system will think that the mouse is always over the HUD.

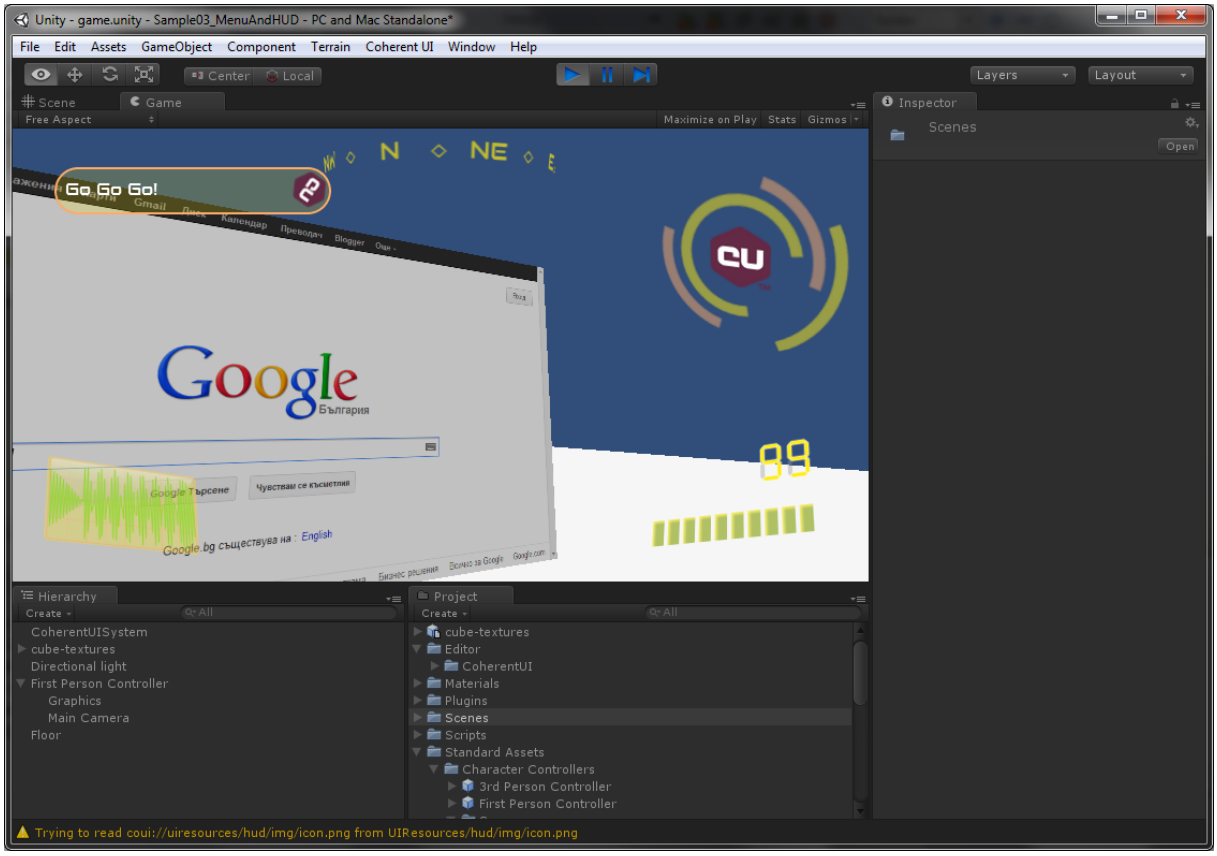

Figure 3.4: CoherentUIView HUD sample in action

3.4 Binding Demo

This sample will demonstrate communication between Unity3D and JavaScript used by the HTML page loaded in a Coherent UI View.

Note

If you're having trouble displaying the local resources, make sure you have selected the *UIResources* folder located in the root of the project using the *Edit* → *Project Settings* → *Coherent UI* → *Select UI folder* menu in the editor. This folder will be used for resources addressed with `coui://` links.

The sample shows a simple options dialog. The options are passed from JavaScript to Unity3D and back to exercise the binding. This walkthrough will give you a brief overview of the Coherent UI Binding fundamentals but be sure to check out the Binding section in this guide, as well as the **Binding for .NET** section in the general reference document.

Note

This sample guide is written for C# Unity3D scripts

First, when passing custom object types between Unity3D and JavaScript, you must first inform Coherent UI about the data in the type. You can do that using the `CoherentType` attribute:

```
using Coherent.UI.Binding;

// all properties / fields for Options will be visible to Coherent UI
[CoherentType(PropertyBindingFlags.All)]
public struct GameOptions
{
    public string Backend;
    public uint Width;
    public uint Height;

    public string Username
    {
        get {
            return System.Security.Principal.WindowsIdentity.GetCurrent().Name.ToString();
        }
    }

    // rename the NetPort property to NetworkPort
    [CoherentProperty("NetworkPort")]
    public uint NetPort { get; set; }
}
```

Now the `GameOptions` structure will correspond to a JavaScript object having the same properties. Note that you can rename a property using the `CoherentProperty` attribute. In this case, the "NetPort" property will correspond to "NetworkPork" in JavaScript.

After exposing the properties, we need to register some event handlers. This can be done in two ways. The first way to bind event handlers is to add a handler for the `ReadyForBindings` event of the `UnityViewListener` (manual binding). The second way (.NET only) is to add a `CoherentMethod` attribute to the method you want to bind (automatic binding). See the CoherentMethod section in this guide for more details.

Note

The sample provides two scripts for binding - ManualBinding.cs and AutomaticBinding.cs. Make sure that only one of those scripts is active when exploring the sample.

We'll explore the manual binding first. Start by registering a handler for `ReadyForBindings`:

```
m_View = GetComponent<CoherentUIView>();
m_View.Listener.ReadyForBindings += HandleReadyForBindings;
```

The handler would look like this:

```
void HandleReadyForBindings (int frameId, string path, bool isMainFrame)
{
    if (isMainFrame)
    {
        // bind ApplyOptions to "ApplyOptions" in JavaScript
        m_View.View.BindCall("ApplyOptions", (Action<GameOptions>)this.ApplyOptions);
        m_View.View.BindCall("GetLatency", (Func<int>)this.GetNetworkLatency);
        m_View.View.BindCall("GetGameTime", (Func<int>)this.GetGameTime);
```

```
        m_View.View.BindCall("GetMath", (Func<BoundObject>)(() => {
            return BoundObject.BindMethods(new MyMath());
        }));


        // triggered by the view when it has loaded
        m_View.View.RegisterForEvent("ViewReady", (Action)this.ViewReady);
    }
}
```

Now, when JavaScript calls `engine.call("ApplyOptions", options)`, Unity3D will execute its handler - namely the `(Action<GameOptions>)this.ApplyOptions` method registered above.

The `options` structure passed as a parameter looks like this:

```
function onApplyButton(){
    var options = {};
    options.__Type = "GameOptions";
    options.Backend = $("#backend").text();
    options.Width = Number($("#gameWidth").val());
    options.Height = Number($("#gameHeight").val());
    options.Username = $("#user").text();
    options.NetworkPort = Number($("#netPort").val());

    // This will call the C++ engine code with the just created structure. It'll be correctly populated
    engine.call("ApplyOptions", options);
}
```

Note that there is one "internal" property - **__Type**. This property defines the mapped type and is essential for correct behavior of the binding provided by Coherent UI.

The "ApplyOptions" handler just bounces the options back to JavaScript:

```
public void ApplyOptions(GameOptions options)
{
    m_View.View.TriggerEvent("gameConsole:Trace", options);
}
```

`gameConsole:Trace` will dump the `options` object in the console.

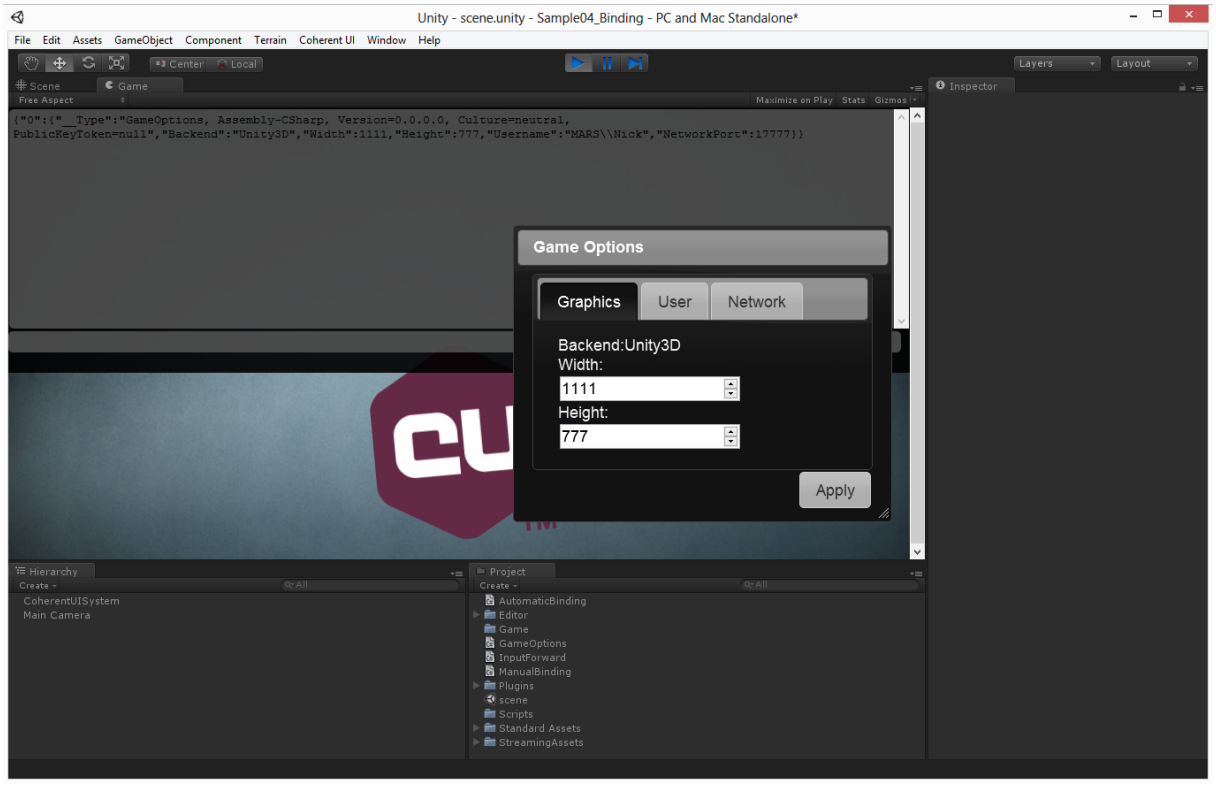

Figure 3.5: Binding sample in action

The automatic binding saves us the first two steps - instead of registering a handler in the `UnityViewListener` callback you can decorate a method with the `CoherentMethod` attribute:

```
[Coherent.UI.CoherentMethod("ApplyOptions", false)]
public void ApplyOptions(GameOptions options)
{
    m_View.View.TriggerEvent("gameConsole:Trace", options);
}
```

The first argument of the attribute is the JavaScript function that we're binding. The second defines whether the function is an *event* (opposed to a *call* - *calls* are single-cast only).

Returning a value to JavaScript

Calls can also return a value. Let's inspect the "GetLatency" binding:

```
// By default, the second argument of CoherentMethod is false
[Coherent.UI.CoherentMethod("GetLatency")]
public int GetNetworkLatency()
{
    // not actual latency :)
    return (int)UnityEngine.Random.Range(0, 1000);
}
```

Since the execution of JavaScript is **not** synchronous you will receive the result in a callback. Coherent UI provides a promise/future pattern for convenience. This is the "GetLatency" JavaScript call in all its glory:

```
function getLatency() {
    engine.call("GetLatency").then(function() {
        $("#latency").text(arguments[0]);
    });
}
```

As you can see from the script, the value returned by Unity3D will be passed to the callback as an argument.

Method Binding

This sample also shows how to use binding of .Net methods. For detailed explanation see "Exposing Methods" in the "Binding for .Net" chapter in the API documenation.

The sample exposes an instance of `MyMath` to JavaScript.

```
class MyMath
{
    public double Sum(double[] numbers)
    {
        return numbers.Sum();
    }

    public double Average(double[] numbers)
    {
        return numbers.Average();
    }
}
```

When the game is up and running, we request the `MyMath` instance, store it as a global object and register two console commands that use it.

```
engine.on('Ready', function () {
    //get the exposed MyMath object
    engine.call('GetMath').then(function (math) {

        // store reference to the object
        window.MyMath = math;

        // register commands for the console
        engine.trigger('gameConsole:AddCommand', 'sum', 'computes the sum of its arguments', function (line
      ) {
```

```
            var numbers = line.split(/\s+/).map(Number);
            numbers.splice(0, 1);
            window.MyMath.Sum(numbers).then(function (sum) {
                engine.trigger('gameConsole:Trace', 'The sum of', numbers, 'is', sum);
            });
        });

        engine.trigger('gameConsole:AddCommand', 'avg', 'computes the average of its arguments', function (
    line) {
            var numbers = line.split(/\s+/).map(Number);
            numbers.splice(0, 1);
            window.MyMath.Average(numbers).then(function (avg) {
                engine.trigger('gameConsole:Trace', 'The average of', numbers, 'is', avg);
            });
        });
    });
});
```

To try out the commands press ~ to open the game console and type `sum 40 2`.

Events

When you need multiple handlers for a JavaScript function you need to register an *event*. This can be done by the `Coherent.UI.View.RegisterForEvent` method for manual binding

```
m_View.View.RegisterForEvent("ViewReady", (Action)this.ViewReady);
```

and passing `true` as a second parameter of the `CoherentMethod` attribute for automatic binding.

```
[Coherent.UI.CoherentMethod("ViewReady", true)]
public void ViewReady()
{
    // show the options
    m_View.View.TriggerEvent("OpenOptions", m_GameOptions);
}
```

Bear in mind that *events* **cannot** return a value.

If you want to handle an event in JavaScript, you need to register a handler using the `engine.on` call.

```
engine.on('OpenOptions', function (options) {
    // Open an options dialog
});
```

An event can be triggered by JavaScript by using

```
engine.trigger("EventName", eventArgs);
```

or by Unity3D using

```
m_View.View.TriggerEvent("EventName", eventArgs);
```

where `m_View` is `CoherentUIView`.

This concludes the binding demo walkthrough.

3.5 Archive resource

This sample demonstrates reading resources from an archive through a custom file handler.

The scene is just a simple camera that has an attached `CoherentUIView` component and another one that forwards input to the view. The important component for this sample is *CustomFileHandlerScript*. It sets the `CoherentUISystem` factory function object in its `Awake` method so the system is initialized with the custom handler.

```
void Awake()
{
    CoherentUISystem.FileHandlerFactoryFunc = () => { return new CustomFileHandler("UIResources", "
      ArchiveResource.tar"); };
}
```

In fact, that's all the component has to do - set the `CoherentUISystem.FileHandlerFactoryFunc` prior to any `Start` method being called. The returned `CustomFileHandler` is the class that does the actual reading/writing. The methods the user needs to implement are:

```
public override void ReadFile (string url, Coherent.UI.ResourceResponse response)
{
    ...
}

public override void WriteFile (string url, Coherent.UI.ResourceData resource)
{
    ...
}
```

They will be called for all *coui://* links and for the cookies (if enabled).

Note

When reading/writing cookies, you may receive a URL that has a `file://` scheme.

The input URL can be interpreted in any way you see fit for your handler. In the example we try to open the link *coui://UIResources/mainmenu/menu.html*. The *mainmenu/menu.html* is actually compressed in an archive. First we verify that the host part of the URL (*UIResources*) is what we expect for compressed resources and then proceed to search the archive for the resource.

Since **Coherent UI** supports asynchronous reading/writing of resources, when done with the I/O operations, you must use the `ResourceResponse` or `ResourceData` object to signal the outcome. You can do that by using the `SignalSuccess` and `SignalFailure` methods.

Note

At the moment the data you read/write must be converted to a native buffer before usage. This is done with the `System.Runtime.InteropServices.Marshal` class. Here's a short snippet that reads a file and then copies the managed buffer in an unmanaged buffer provided by the `ResourceResponse` object:

```
public override void ReadFile (string url, Coherent.UI.ResourceResponse response)
{
    byte[] bytes = File.ReadAllBytes(url);

    IntPtr buffer = response.GetBuffer((uint)bytes.Length);
    if (buffer == IntPtr.Zero)
    {
        response.SignalFailure();
        return;
    }

    Marshal.Copy(bytes, 0, buffer, bytes.Length);

    response.SignalSuccess();
}
```

Make sure you check out the code in *CustomFileHandlerScript* for a complete example.

3.6 iOS Demo

This sample demonstrates how you can control what part of the input is forwarded to Unity3D by using JavaScript. The sample starts with an overlay that has 3 buttons on the left side, area for testing input forwarding on the right, and a red box in the world. The buttons enable/disable the input forwarding for touches on the right side of the overlay and the third one bumps the box upwards. When you touch the box a little force is applied and it should move forward as if you pushed it. If you touch the box in the right area of the overlay the force is applied only if input forwarding is enabled.

There are also buttons for toggling the overlay and mouse look controller in the top left corner for convenience.

Warning

The HTML code for the sample is not designed for small displays so it may look out of proportion on a phone or a phone simulator. You can try lowering the font size in the accompanying *css* file.

Note

The Coherent UI View used has its input state set to "Transparent". If you set it to "Take all" input is consumed before reaching Unity3D and if you set it to "Take none" it is sent directly to Unity3D.

Warning

Unfortunately, at the moment you can't see the actual behavior of the sample in the Unity3D Editor. Input is handled differently for the standalone and mobile versions so it recommended that you test your project either on a mobile device or an emulator.

Code-wise speaking, when you make a touch on a "Transparent" Coherent UI View, the `engine.checkClickThrough` function is called. You can check its code in `coherent.js`. Basically it obtains the DOM element from the touch coordinates and begins a series of checks. First, if the element is the *document body* or has the **coui-noinput** class, input is directly sent to the client (Unity3D). Otherwise, the element is checked for having a **coui-inputcallback** class. If it does, the element's `couiInputCallback` is invoked which determines whether the input is forwarded or consumed; if it doesn't have such class, input is consumed.

The sample enables/disables the input forwarding on the right area by removing/adding the **coui-noinput** class to the corresponding DOM element. In this sample, only the DOM element for the right area has the class **coui-inputcallback** and its `couiInputCallback` function is set when the engine has signaled it's ready. The function itself doesn't do anything special, it just always `returns true`, meaning that the input is consumed in JavaScript and never sent to the client (Unity3D).

You can also bump the box upwards which is a demonstration for the binding for iOS. It's very much the same as binding for .NET/Unity3D standalone so we'll not go in detail here.

Summing up:

- if you want the input forwarded to the client when touching an element, simply add the **coui-noinput** class to the element.
- if you want an element to consume input, ensure that it does **not** have the **coui-noinput**.
- if you want to have custom processing over an element, ensure that it does **not** have the **coui- noinput**, add a **coui-inputcallback**, and add a function `couiInputCallback` which ultimately returns true if you want to consume the input or false if you want to forward it to the client.

Note that for obtaining the element below the touch point we're currently using `document.elementFromPoint`. In the sample, the right area is represented by a `<div>` and there's some text inside it as a child element. Only the `<div>` has the `coui-inputcallback` class. If you touch the text its element will be checked for the `coui-inputcallback` class and since it doesn't have one input will be consumed. Since we want to apply the logic for all touches inside the area this presents a problem. One solution is to add the `coui-inputcallback` class to all child elements and set their `couiInputCallback` functions to the same variable. Another solution is to use the **pointer-events** CSS property on the children elements, e.g.:

```
#menu-right > *
{
    pointer-events: none;
}
```

This is how the sample solves the problem. Note that this is an experimental CSS property and prevents elements from being the target of pointer events. This is fine in the sample but may have adverse effects in your code so use it with caution.

Chapter 4

Programmer's API

The Unity API is an extension of the normal .Net API, and it is contained in the main API Reference documentation.

4.1 CoherentUISystem properties for Desktop

- **Main Camera** - The main camera in the scene. This property is used only for Click to focus Views.
- **Host Directory** - Path to the directory where the Coherent UI executable resides. This path can be relative to the resource path of the game (i.e. the *Assets* folder of the Unity3D project or the <YourGameName>_Data folder of the built game). By default it is *StreamingAssets/CoherentUI*.
- **Enable proxy** - Enables proxy support by autodetecting the system settings. This detection is usually very slow and this setting should be enabled only when the user is behind a proxy and you're accessing the Internet.
- **Allow cookies** - Enables support for cookies.
- **Cookies resource** - A file that will be used for reading and writing cookies when they are enabled.
- **Cache path** - Path for saving cached data. Leave null for in-memory caching only.
- **HTML5 Local storage path** - Path for saving HTML5 page local storage data. Leave null to forbid local storage.
- **Force disable plugin fullscreen** - Disables fullscreen mode for all plugins (e.g. Flash, Silverlight, etc.)
- **Disable web security** - Disable same origin policy. Use with caution.
- **Debugger port** - The port that will be opened for the debugger to connect and debug your interface. A value of -1 means disabled.

The `CoherentUISystem` component also provides a static factory function object (`FileHandlerFactoryFunc`) that the user can customize in order to make use of her own `FileHandler`. See the Custom file handler section for a detailed explanation.

4.2 CoherentUIView properties for Desktop

- **Page** - The URL to be loaded.
- **Width** - The width of the Coherent UI View.
- **Height** - The height of the Coherent UI View.
- **Is Transparent** - Defines if the View supports transparency.

- **Support Click-through** - *Available for transparent Views*. Enables support for queries whether the cursor is over a transparent pixel. A transparent pixel is considered one that has an alpha value below or equal to the *click-through alpha threshold*.
- **Click-through alpha threshold** - A value in the range [0-1] inclusive that determines whether a pixel is transparent; A pixel is transparent if its alpha value is below or equal to the threshold.
- **Click to focus** - When enabled, the View takes the input focus when clicked and releases it when you click outside the View. See Click to focus Views
- **Is On Demand** (**Coherent UI Pro only**) - Provides perfect synchronization of the game frames and Coherent UI frames. Use this when you need to synchronize the game and the interface, e.g. when displaying name tags over the players. Using standard views may introduce a delay of a few frames.
- **Target Framerate** (**Coherent UI Pro only**) - Sets the target framerate for the View. The view will never exceed this framerate.
- **Draw after post-effects** - Defines whether the View is drawn before or after the post-effects. Available for Views attached to cameras.
- **Flip Y** - Flips the drawn image vertically.
- **Intercept All Events** - When enabled, any event triggered in *JavaScript* on the `engine` object is forwarded to the game object containing the View.
- **Enable Binding Attribute** - Enables the usage of the Coherent Method attribute

4.3 CoherentUIView properties for Mobile

- **Page** - The URL to be loaded.
- **Width** - The width of the Coherent UI View in pixels.
- **Height** - The height of the Coherent UI View in pixels.
- **XPos** - The X of the Coherent UI View position relative to the upper-left corner of the device screen.
- **YPos** - The Y of the Coherent UI View position relative to the upper-left corner of the device screen.
- **Show** - Controls if to show the View
- **Input state** - Controls how the view handles user input (touches and gestures). The behavior only influences events that happen in the bounds of the View. If the user touches outside a view the event goes normally to the game.
  - Transparent - the user controls which input events to be talen by the UI and which to pass to the game. This is explained in the Mobile input management section (Input forwarding - Mobile)
  - Take all - the View takes all the inpurt - nothing is passed to the game
  - Take none - the View takes no input - everything goes to the game
- **Flip Y** - Flips the drawn image vertically.
- **Is Transparent** - Defines if the View supports transparency.
- **Support Click-through** - *Available for transparent Views*. Enables transparent input.
- **Intercept All Events** - When enabled, any event triggered in *JavaScript* on the `engine` object is forwarded to the game object containing the View.
- **Enable Binding Attribute** - Enables the usage of the Coherent Method attribute

Chapter 5

Important points

5.1 Binding

Binding *C#*, *UnityScript* and *Boo* handlers to JavaScript callbacks is the same as binding for the .Net platform. You have to register handlers when the `UnityViewListener`'s `ReadyForBindings` event is fired. You can do that by using either `Coherent.UI.View.BindCall` (for single-cast delegates) and `Coherent.UI.View.RegisterForEvent` (when you have multiple subscribers for the event). For more details see the general reference documentation, chapter **Binding for .Net**.

```
private void RegisterBindings(int frame, string url, bool isMain)
{
    if (isMain)
    {
        var view = ViewComponent.View;
        if (view != null)
        {
            // When engine.call('NewGame') is executed in JavaScript,
            // the this.NewGame method will be called as well
            view.BindCall("NewGame", (System.Action)this.NewGame);
        }
    }
}
```

The `Coherent.UI.View` can be obtained using the `View` property of the `CoherentUIView` component.

To take advantage of the *Unity3D* component and message system each CoherentUIView has the `InterceptAllEvents` property. If intercepting of all events is enabled, any event triggered in *JavaScript* on the `engine` object is forwarded to the game object containing the view. This is done by using `SendMessage` with `Coherent.UI.Binding.Value[]` containing all the event arguments.

```
engine.trigger('Event', 42);
// will execute SendMessage('Event', [42])
```

Note

Using this method of handling triggered events has some additional overhead over the direct .Net `Coherent.UI.View.RegisterForEvent` method.

5.2 CoherentMethod attribute for .NET scripts

Instead of handling the `ReadyForBindings` event and doing `BindCall` or `RegisterForEvent` by yourself, you can use the `CoherentMethod` attribute to decorate methods in components.

Warning

This attribute only works if the `CoherentUIView`'s `Enable Binding Attribute` is set to *true*. By default it is **false**.

The decorated methods will be automatically bound to the `View` owned by the `CoherentUIView` component in the Game Object. If the Game Object has no `CoherentUIView` component, the attribute has no effect. The `CoherentMethod` attribute has a string property for the JavaScript event name, and an optional boolean parameter that specifies whether the method is a *call* or *event* handler (*calls* can have only a single handler, while *events* may have many). Here's an example component using the attribute:

```
public class BindingComponent : MonoBehaviour {

    [Coherent.UI.CoherentMethod("NewGame")]
    void MyCallHandler()
    {
        Debug.Log("MyCallHandler called in response to engine.call('NewGame')");
    }

    [Coherent.UI.CoherentMethod("EnemySpotted", true)]
    void MyEventHandler()
    {
        Debug.Log("MyEventHandler called in response to engine.trigger('EnemySpotted')");
    }
}
```

See the Binding Sample for a complete example.

Warning

* Binding methods using the `CoherentMethod` attribute is easier than doing it manually in `ReadyForBindings`, but presents possible performance penalties during game startup. When the `CoherentUIView` component is created, it searches all the other components in the host Game Object for methods marked with `CoherentMethod` using reflection. This can be a costly operation and to prevent undesirable slowdowns during startup the `Enable Binding Attribute` property for each `CoherentUIView` is set to **false** by default.
* The `CoherentMethod` attribute currently does **NOT** support dynamically added components. Methods decorated with the attribute are only bound when the `CoherentUIView` component is created, which is usually when the Game Object it is part of is created.

JavaScript and Unity3D

Consult the *Binding for .NET* chapter in the general reference document. Check the Binding Sample and its walkthrough in this guide for an example.

Briefly, Unity3D can call JavaScript using *events*; JavaScript can call Unity3D using *events* or *calls*.

- Events

Events can be called by JavaScript

```
engine.trigger("MyEvent", args);
```

or Unity3D

```
coherentUIView.View.TriggerEvent("MyEvent", args);
```

The "MyEvent" will be handled by any registered method in JavaScript

```
engine.on("MyEvent", function() {...});
```

or Unity3D

```
coherentUIView.View.RegisterForEvent("MyEvent", handlerMethod);
```

- Calls

Calls, unlike events, can have only one handler. The can also return values. To execute a *call* from JavaScript use

```
engine.call("MyCall", args);
```

It will be handled by a method registered using

```
coherentUIView.View.BindCall("MyCall", handlerMethod);
```

5.3 Namespaces

Coherent UI classes are placed in the Coherent.UI namespace for Desktop and Coherent.UI.Mobile for the Mobile version. You can check the Coherent UI files - for instance CoherentUIView and take a look at the beginning of the file at how the namespaces are imported depending on the Unity platform targeted. Exceptions to that rule are classes that cannot be in a namespace because Unity doesn't allow it, such as components that derive from MonoBehaviour.

5.4 Subclassing CoherentUIView and UnityViewListener

The default CoherentUIView component and the UnityViewListener provide the most common functionality and are usually enough for normal usage of *Coherent UI*. If you need custom behavior, you need to subclass them.

The class you derive from UnityViewListener would usually subscribe to various events that aren't handled by default. It is recommended not to override the OnViewCreated callbacks since the UnityViewListener class contains important logic that you would have to implement yourself otherwise.

The class you derive from CoherentUIView would only need to create an instance of your custom View Listener. This can be done by copying the Update method of CoherentUIView and editing it appropriately.

Note that when subclassing CoherentUIView you will no longer be able to view or edit the properties in the Inspector. That's because we're using C# properties in our component instead of fields and they are not automatically shown. To show the properties of a given C# script we need to make a new Editor script (inside the Editor folder of your Assets) that shows the properties for a specific type. We've already done that for `CoherentUIView`, but you'll have to do it yourself for derived classes. The script contents should be the following:

```
using UnityEngine;
using System.Collections;
using UnityEditor;

[CustomEditor(typeof(<YourType>))]
public class <YourType>ViewEditor : Editor {

    private <YourType> m_Target;
    private CoherentPropertyField[] m_Fields;

    public void OnEnable() {
        m_Target = target as <YourType>;
        m_Fields = CoherentExposeProperties.GetProperties(m_Target);
    }

    public override void OnInspectorGUI() {
        if(m_Target == null)
            return;
        this.DrawDefaultInspector();
        CoherentExposeProperties.Expose(m_Fields);
    }
}
```

Just replace <YourType> with the actual name of your class.

Note

In most cases, subclassing is unnecessary. See Facebook Sample for an example how to subscribe for `UnityViewListener` events.

5.5 Coherent UI system lifetime

Since initialization of the CoherentUISystem component is a costly operation, it is designed to be done once in the first scene of your game. The component itself has the same lifetime as the application. Since Unity tears down the state of the game when you load a new scene, the component is marked not be destroyed using the DontDestroyOnLoad() function. This makes it persist through scenes and is available using the Object.FindObjectOfType function. Getting the system can be done with the following line of code:

```
var uiSystem = Object.FindObjectOfType(typeof(CoherentUISystem)) as CoherentUISystem;
```

5.6 Customizing initialization of the Coherent UI System

When using only `CoherentUIView` components, the Coherent UI System will be automatically initialized using the default parameters. These parameters define global system settings such as whether cookies are enabled, local storage and cache paths, debugger port and others. Check the `CoherentUISystem` component for a full list.

The Coherent UI System can be initialized with parameters other than the defaults in the following ways. Either drag the `CoherentUISystem` component to any object and edit the properties in the Inspector window, or edit the *CoherentUISystem.cs* script located in *Standard Assets/Scripts/CoherentUI* to fit your needs.

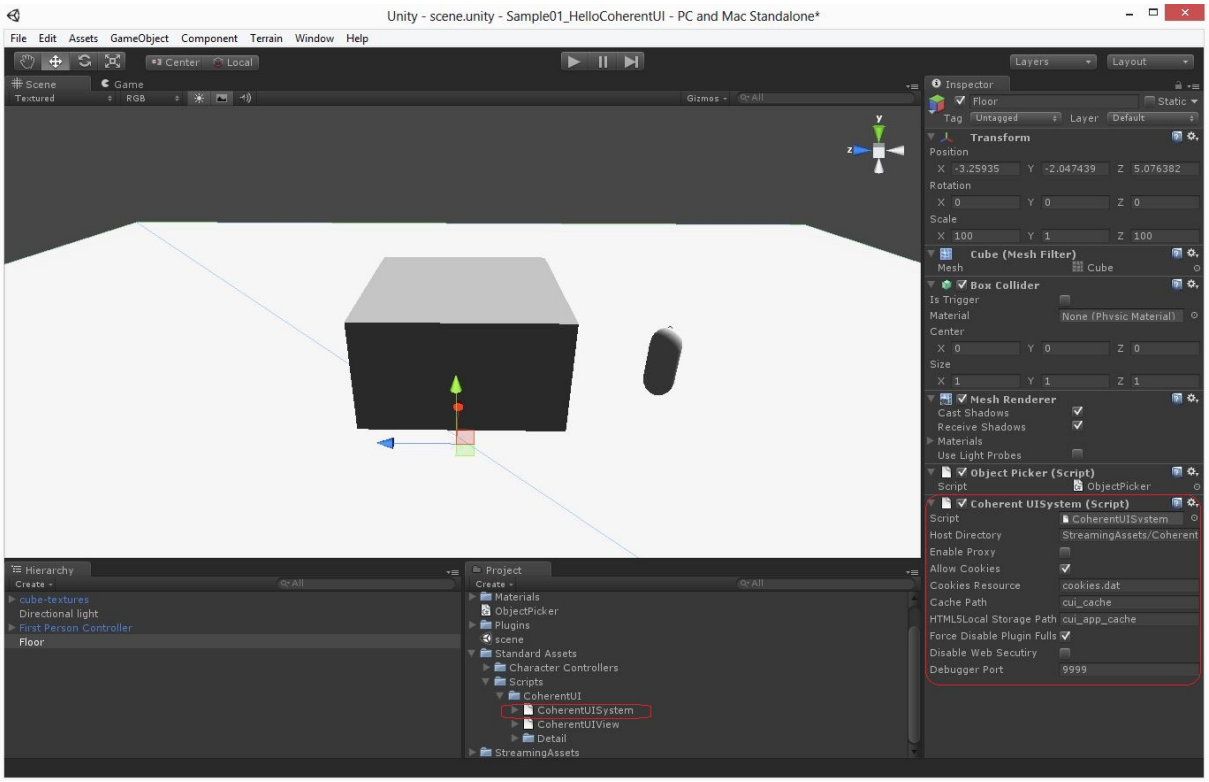

Figure 5.1: Custom CoherentUISystem component

The `CoherentUISystem` component is designed to be only one in the whole game. Adding more than one `CoherentUISystem`s to your level will result in undefined behavior.

5.7 Update cycle

In a standard C++ or .NET application you need to poll Coherent UI for surfaces each frame using the Update and FetchSurfaces API calls. In our Unity integration, this is all hidden from you and you don't have to worry about it. The CoherentUIViewRenderer component issues rendering events which are handled by our library. All that's left for you is to drag a CoherentUIView component on an object!

5.8 Input forwarding - Desktop

A `CoherentUIView` requires focus to receive user input. Usually you'd want to forward input to a single view, but for flexibility **Coherent UI** supports multiple focused views. A `CoherentUIView`'s focus is controlled by the `ReceivesInput` property. To avoid confusion with multiple views, the property is set to **false** by default, meaning no input will be forwarded to **Coherent UI** unless you explicitly set it in your Game Object. It is **NOT** exposed in the Inspector, as it designed to be modified in code only.

Note

The forwarding happens in the `CoherentUISystem`'s `LateUpdate` method, allowing you to do all your logic for input focus management in the `Update` methods of your components.

5.9 Input forwarding - Mobile

On iOS all *Coherent UI* Views are composited in 2D on-top of your Unity3D application content. When the user start a touch event or perfroms a gesture there is a mechanism that decides if that event should be handled by the UI or the application. It works like this: in the properties of the View the user can select one of three input modes for a every View - "Take All", "Take None", "Transparent". Keep in mind that all those modifiers are in effect only for events that happend are in the bounds of the View. If the user touches outside a particular View the event is always handled by the game.

- "Take All" specifies that all events are handled by the View and nothing goes to the game. It is usable if you want to have for instance an in-game browser. All touches on it should be handled by itself and not influence the game.
- "Take None" specifies that the View passes all input to the game. This is usable if you need to just show some non-interactive views or disable their input completely in some situation.
- "Transparent" specifies that the input is either handled by the View or the game - usable for HUDs, Menus etc. For Views with "Transparent" input mode the user is the one in charge of deciding if an element on the page is interactive - hence should receive input or is 'input transparent'.

  Note

  Input "Transparent" views work correctly **ONLY** if you have included *coherent.js* in your HTML page. Upon touch within a View, *Coherent UI Mobile* inspects the touched element:

- if the element has the CSS class *coui-noinput* it passes input to the game. The element does not accept input.
- if the element has the CSS class *coui-inputcallback*, a method called *couiInputCallback(x, y)* is called on the element with the coordinates of the event. It should return "true" if the user wants the element to handle the input and "false" if the game should handle it. This allows for custom fine-grained control in JavaScript on which elements are interactive.
- if the class *coui-noinput* and *coui-inputcallback* are missing from the element, it is assumed that it is interactable and takes the input.

To summarize: If a View has an Input State set to "Transparent" all elements are by default interactive and take input. You can mark elements with the CSS class *coui-noinput* to make them transparent to input. If you need more advanced logic when deciding if an element is interactive or not you can decorate it with *coui-inputcallback* and implement a method *couiInputCallback(x, y)* on it.

5.9.1 Custom file handler

The `CoherentUISystem` component makes use of a static factory function object (`FileHandlerFactoryFunc`) to create the `FileHandler` object that is used reading URLs with the *coui* scheme. The default function returns a handler that reads resources from the path set by *Edit* → *Project Settings* → *Coherent UI* → *Select UI folder* for the Editor and in the Data folder for built games.

The factory function object is public and can be customized. The `FileHandler` it returns is passed to the UI initialization routine in the `Start` method of the `CoherentUISystem` component. That means the user should set the factory function prior to the invocation of the `Start` method of the components - e.g. in the `Awake` method.

Note

You can check the execution order of event function in Unity3D `on this page`

This is an example usage of a custom file handler:

```
public class CustomFileHandlerScript : MonoBehaviour {

    class CustomFileHandler : Coherent.UI.FileHandler
    {
        public override void ReadFile (string url, Coherent.UI.ResourceResponse response)
        {
            // Implementation here
        }

        public override void WriteFile (string url, Coherent.UI.ResourceData resource)
        {
            // Implementation here
        }
    }

    //------------------------------------------------------------------------------

    void Awake()
    {
        CoherentUISystem.FileHandlerFactoryFunc = () => { return new CustomFileHandler(); };
    }
}
```

See Archive resource demo for an example.

5.9.2 UI Resources

Files for Coherent UI are by default selected from the folder set by *Edit* → *Project Settings* → *Coherent UI* → *Select UI folder*. Resources found there will be used by the editor and will automatically be copied in your game upon build.

The selected UI resources folder is per-user so that different developers working on the game can have their UI folders wherever they want on their machine. You can also set a per-project folder for the UI resources. This is done by extending the CoherentPostProcessor class by setting a static const setting named ProjectUIResources:

```
public partial class CoherentPostProcessor {
    public static string ProjectUIResources = "relative/path/to/ui/resources";
}
```

The per-project path must be relative to the folder of the project and the extension class should live under the 'Editor' folder in Unity. This feature is very handy also if you build your game on machines that can't start Unity and you use the command line. The per-user setting overrides the per-project one so that developers can still put their resources wherever they want.

5.9.3 Click-to-focus Views - Desktop only

When the `ClickToFocus` property is enabled on a View, it will automatically take **all** the input focus when you click on it and lose it when you click somewhere else. When focused, all mouse and keyboard input will be forwarded to the View. "Click-to-focus" views have their `ReceivesInput` property managed by the `CoherentUISystem` and you should **NOT** set it manually. If you do so, you'll receive a warning message in Unity3D and the input forwarding behavior will be unexpected.

Warning

There is no way to prevent the standard Unity character controller scripts from moving the character, even when the input event is *used*. You have to manually disable your character controller when you want the user to type in in a view and stands still. For an explanation see the Menu And HUD sample.

Note

"Click-to-focus" views perform raycasts to obtain the object in the 3D world below the cursor. For a raycast to report the correct texture coordinates of the hitpoint, you need to set up a *Mesh Collider* on the objects with Coherent UI Views placed.

"Click-to-focus" Views are useful in cases when you want keyboard input forwarded to a View regardless of the mouse position, e.g. input fields.

Note

To function properly, "Click-to-focus" views need to know which camera is the main camera in the scene. For simple scenes, this can be obtained from the `Camera.main` property in Unity3D. This is what Coherent UI assumes is the main camera, and obtains it in the `MonoBehaviour.OnEnable` callback, which is executed when a scene is loaded. This is done only if there is no currently set camera for the `m_MainCamera` field so it does not interfere with custom user code. For complex scenes with multiple cameras, however, it is up to you to set the public `m_MainCamera` field to the appropriate camera (also visible in the Inspector window).

Which `CoherentUIView`s receive input is up to your gameplay needs. Here we'll walk you through a simple script that you'll see used in the samples - it forwards input to the closest view under the cursor. First, it sets the `ReceivesInput` property to *all* views to **false**. Then it queries the `CoherentUIView` attached to the main camera (if any) whether the mouse is over a solid or transparent pixel (make sure to set the `SupportClickThrough` property of the HUD view to **true** to support this operation). If the mouse is over a solid pixel, then the HUD is focused and receives input. Otherwise, a raycast is generated that finds the object under the cursor. If that object has a `CoherentUIView` component, that's what gets the focus.

Here's the script itself:

To summarize, you can apply any logic you like for input forwarding - e.g. forward input to objects in the view frustum, HUD only, etc. Coherent UI supports multiple focused views. View focus can be modified using the `ReceivesInput` property of `CoherentUIView` which is controlled only by the script code.

You can also mark views as "Click to focus" which makes them take all the focus when clicking them (and lose focus when clicking somewhere else). You should take care not to set the `ReceivesInput` property on "Click to focus"-enabled Views as it is automatically managed. Setting the `ReceivesInput` property on such views manually will result in unexpected behavior and will produce a warning message.

5.10 Hit testing - Desktop only

Note

For and explanation about input forwarding for Mobile check Input Forwarding Mobile

Forwarding input to Views attached to the camera is straight-forward - you only have to mark your view as an input receiver using the ReceivesInput property of CoherentUIView. The mouse position will be obtained from the Input.mousePosition property.

If you want to forward an input event to a View that's attached on an object in the 3D world, you'll have to do a bit more work. You'll have to use a raycast to find the object below the cursor and then transform the texture coordinates of the hit point into the space of the Coherent UI View. Note that Unity provides texture coordinates only when the object has a Mesh Collider component attached. The coordinates must be transformed from [0, 1] to [view.Width, view.Height]. This can usually be done simply by multiplying the coordinates by the dimensions of the View (which are available as properties). Then, you have to set the resulting coordinates to the CoherentUIView component using the SetMousePosition method. Check Input Forwarding for an example script that forwards input to the view on the object that is currently below the cursor. Note that in the samples the `MeshCollider` component has the same geometry as the renderable mesh. This may not always be true and in such cases you would have to make a transformation of the coordinates that works for you.

For a sample how hit testing works see the Menu and HUD sample.

5.11 Logging

Coherent UI logs are automatically redirected to the Unity console (or game log for built games) using the Debug.Log method. You can control the minimum severity of the Coherent UI logs when initializing the Coherent UI System.