CORTA STUDIOS
EXPERIMENTAL GAMES

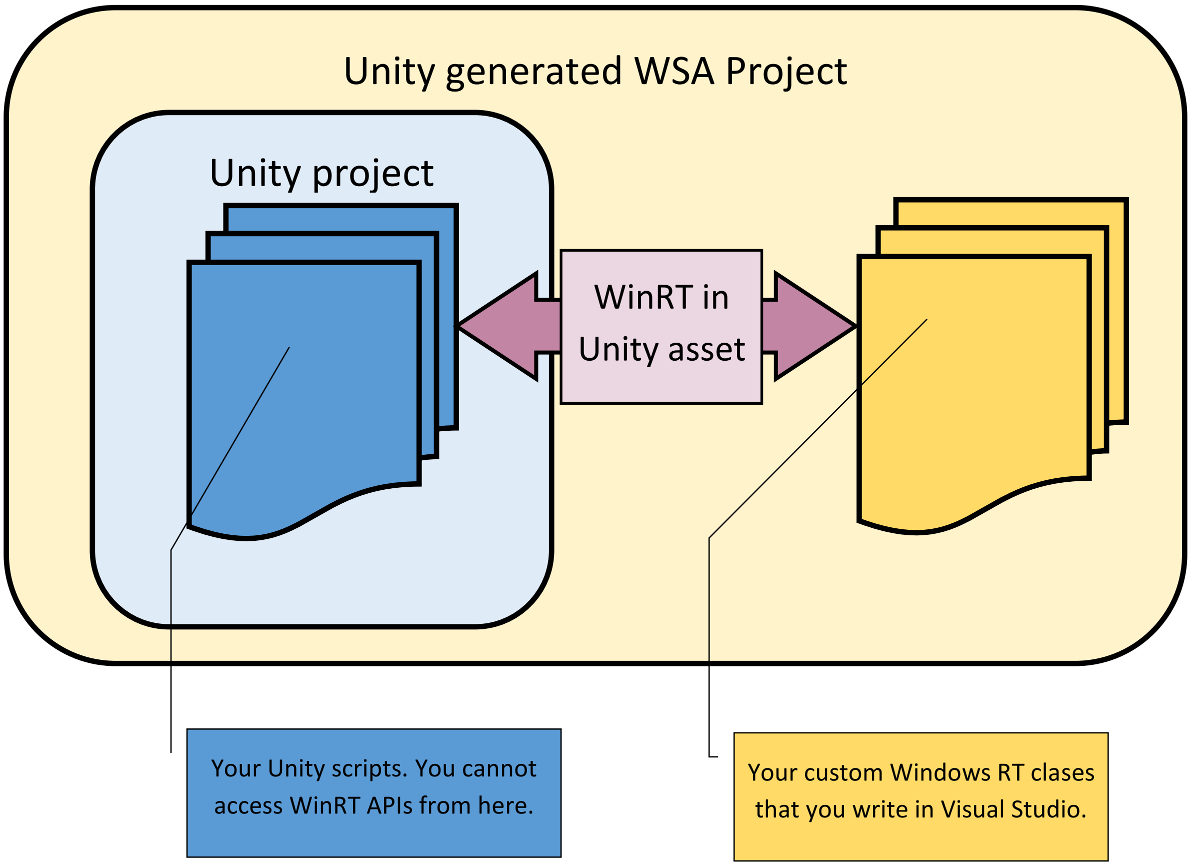
WINRT IN UNITY. DOCUMENTATION

Hi, I'm Salvador Romero. You might remember me from such Unity assets as Dynamic Color Correction or from my games at Corta Studios. Today I'm here to tell you about my new asset WinRT in Unity, and how you can use to access new Windows 8.1 and Windows 10 features such *Cortana*, *speech*, *ink* and more! So, grab a cup of developing beverage, sit comfy on your dev/gaming chair and relax while we walk through this **documentation**.

MOTIVATION

You can invoke WinRT API from Unity when you target Windows Store apps, but because most of those API use modern features of C# not supported by the Mono version used by Unity (like *async* calls) you cannot really use them. You cannot use XAML controls easily either. If you want to access such APIs and controls, you have to write further code in the Visual Studio project generated by Unity.

This asset contains scripts for Unity and a package for your WSA project that stablish a connection between your unity scripts and your WSA classes, so you can use it to call UWP APIS and use XAML control that you define with Visual Studio from and back to your Unity scripts.


Unity generated WSA Project
Unity project
WinRT in Unity asset
Your Unity scripts. You cannot access WinRT APIs from here.
Your custom Windows RT clases that you write in Visual Studio.

HOW TO STABLISH A CONNECTION BETWEEN YOUR UNITY SCRIPTS AND YOUR WSA APP

Or how this asset work under the hood, briefly:

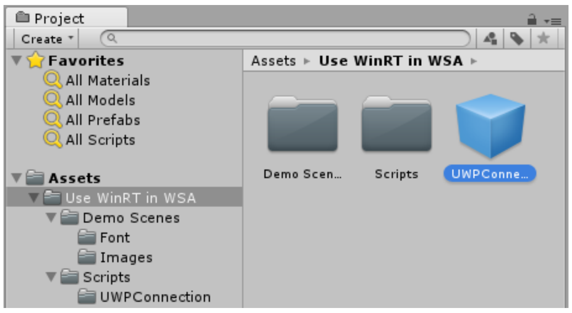
You cannot access the classes that you write in the Visual Studio project from Unity because they are not available when Unity generates the WSA project. But you can raise an event from your Unity scripts and your WSA app code can subscribe to this event. In this way, your WSA classes can listen to your Unity scripts. You can access your Unity scripts from your WSA classes because they are visible at compilation time.

Unity scripts run usually on the app thread, while your xaml controls and page code runs on the UI thread. You need to have this in mind when accessing your classes and scripts as mentioned above. This could be a source of problems if you don't know what you are doing.

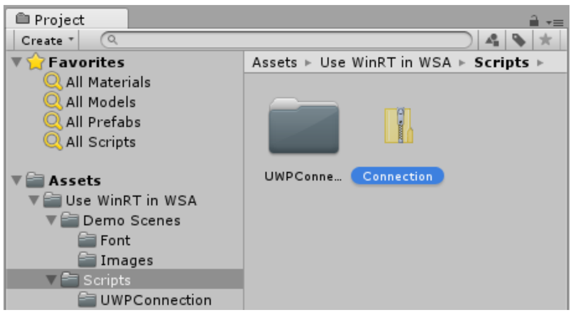
This asset manages all those complications for you and presents you a programming interface that is convenient and scalable. Even if you know how to make this setup, you might find this asset useful.

THE ASSET

This asset consists on some scripts and one prefab that you need to use in Unity, plus a zip files containing three *.cs* files that you need to import in your WSA project generated by Unity.



In the asset root folder, you will find the prefab that you need to use this asset in Unity.



In the script folder, you will find a zip file containing the C# classes for your WSA project.

Do not unzip the zip folder in your Unity project to avoid Unity to compile those files.

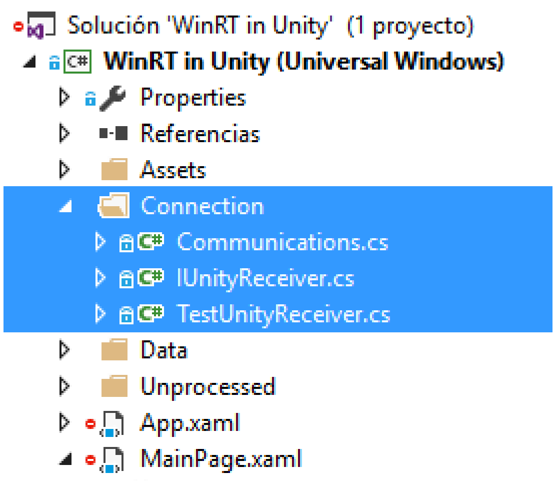
SETTING UP THE ASSET

IN YOUR UNITY PROJECT

Drag and drop the prefab *UWPConnectionManager* **in your first scene**. When the WSA starts up, it will look up for an Unity object with this name to start up the connection, so it's important that you have it in the hierarchy of your first scene, even if you don't plan to use it on your first scene.

IN YOUR WSA PROJECT

Unzip the *.cs* files contained in the zip file *Connection* that you will find on the Scripts folder of the asset. Do not unzip the files inside the Unity project, to avoid Unity to compile those files. Once you have generated the WSA project, open it on Visual Studio and import the *.cs* files in the project. You may want to create a folder with the namespace name to keep the project better organized.



You need then to edit the file *MainPage.xaml.cs*, which you will find in the solution inspector as a child of *MainPage.xaml*. Unfortunately, you need to this manually.

1. On the file headers, add:

```
//WinRT in Unity addition 1
using UWPConnection;
```

2. On the constructor, add the following line. The greyed lines are for reference:

```
public MainPage()
{
            this.InitializeComponent();
            NavigationCacheMode = Windows.UI.Xaml.Navigation.NavigationCacheMode.Required;
            AppCallbacks appCallbacks = AppCallbacks.Instance;
            // Setup scripting bridge
            _bridge = new WinRTBridge.WinRTBridge();
            appCallbacks.SetBridge(_bridge);

         //WinRT in Unity addition
         appCallbacks.Initialized += OnInitialized;
[…]
}
```

3. At the end of the class, add the following function:

```
//WinRT in Unity addition 3
private void OnInitialized()
{
    new Communications(new TestUnityReceiver());
}
```

The class `TestUnityReceiver` comes with the package as an example, but you may want to change it for your own class. Check the section “Receiving messages in your WSA app from Unity” bellow for more information.

USING THE ASSET FOR COMMUNICATION

ENCODING MESSAGES

When using the asset to send messages from your Unity scripts to your WSA app, or vice versa, you will be using one of these functions, respectively (Only the prototypes are shown here. The actual functions are on different classes):

```
//Send messages from Unity to your WSA app
```

```
public void SendToUWP(object arg)

//Send messages from your WSA app to your Unity scripts
public static void SendToUnity(object arg)
```

Those are the only functions for communications, so you must encode your message using the argument of the function: `object arg`. You will receive this object as you pass it (more on the reception of messages below).

For very simple communications need, you can just pass a variable as parameter. You can check the type and act in consequence when you receive it. Example:

```
//In your WSA classes
int i = 5;
Communications.SendToUnity(i);
```

```
//In your Unity scripts
if (arg is int)
    DoSomething(arg as int);
```

If you have several different actions that you need to communicate, you can pass an array of object as argument. The first object can be a string encoding the nature of the communication, and the rest the argument needed. Example:

```
//In your WSA classes
int secondsToClick = 7;
string destination = "http://cortastudios.com/#photon_rush";

object[] args = new object[3];
args[0] = "AddClicked";
args[1] = secondsToClick;
args[2] = destination;

Communications.SendToUnity(args);
```

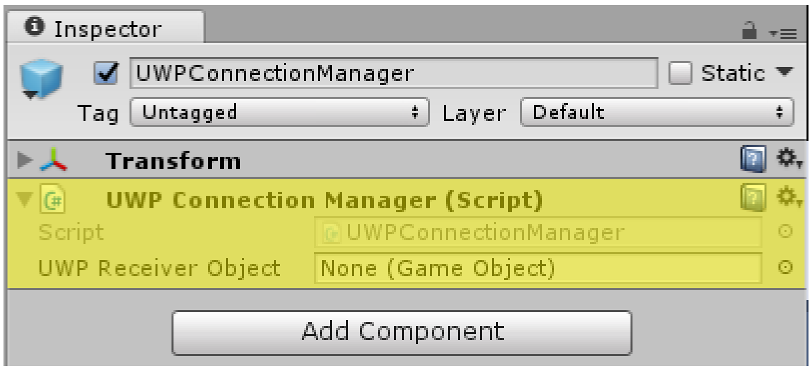
```
//In your Unity scripts
if (arg is object[])
{
    var argArray = arg as object[];
    if (argArray.Length > 0)
    {
        string command = argArray[0] as string;
        switch (command)
        {
             case "AddClicked":
                  if (argArray.Length == 3)
```

```
                        DoSomethingAfterAdClicked(argArray[1] as int, argArray[2]
as string);
                    break;
              case "AnotherCommand":
                    DoAnotherThing();
                    break;
         }
    }
}
```

Experienced developers might want to use reflection or more advance techniques.

SENDING MESSAGES FROM UNITY TO YOUR WSA APP

Since you have created it in your first scene, the prefab *UWPConnectionManager* will exist in all your scenes (It uses the *Don't destroy on load* function). It has a component of the type UWPConnectionManager. Access to it and use its `SendToUWP(object arg)` function.


Inspector
UWPConnectionManager
Static
Tag Untagged
Layer Default
Transform
UWP Connection Manager (Script)
Script UWPConnectionManager
UWP Receiver Object None (Game Object)
Add Component

```
var message = new object();

var connectionObject = GameObject.Find("UWPConnectionManager");
var connection = connectionObject.GetComponent<UWPConnectionManager>();
connection.SendToUWP(message);
```

This code stub is simplified. Remember to add failsafe measures to your code to avoid null pointer exceptions.

SENDING MESSAGES FROM YOUR WSA APP TO UNITY

You need to use the static function `SendToUnity(object arg)` of the class `Communications`. In this case it's easier because both the class and the function are static:

```
var message = new object();
Communications.SendToUnity(message);
```

RECEIVING MESSAGES IN YOUR WSA APP FROM UNITY

You need create a class implementing the interface `IUnityReceiver`. This interface has only one function prototype:

```
void ReceiveFromUnity(object arg);
```

This function will be invoked whenever you send a message from Unity as described above.

You need to create an object of this class when setting up the asset instead of the example class that is indicated in the section “Setting up the asset”. Just change the example class for your own class. For example:

```
//WinRT in Unity addition 3
private void OnInitialized()
{
    //new Communications(new TestUnityReceiver()); //This is the example class
    new Communications(new MyIUnityReceiverClass());
}
```

RECEIVING MESSAGES IN UNITY FROM YOUR WSA APP

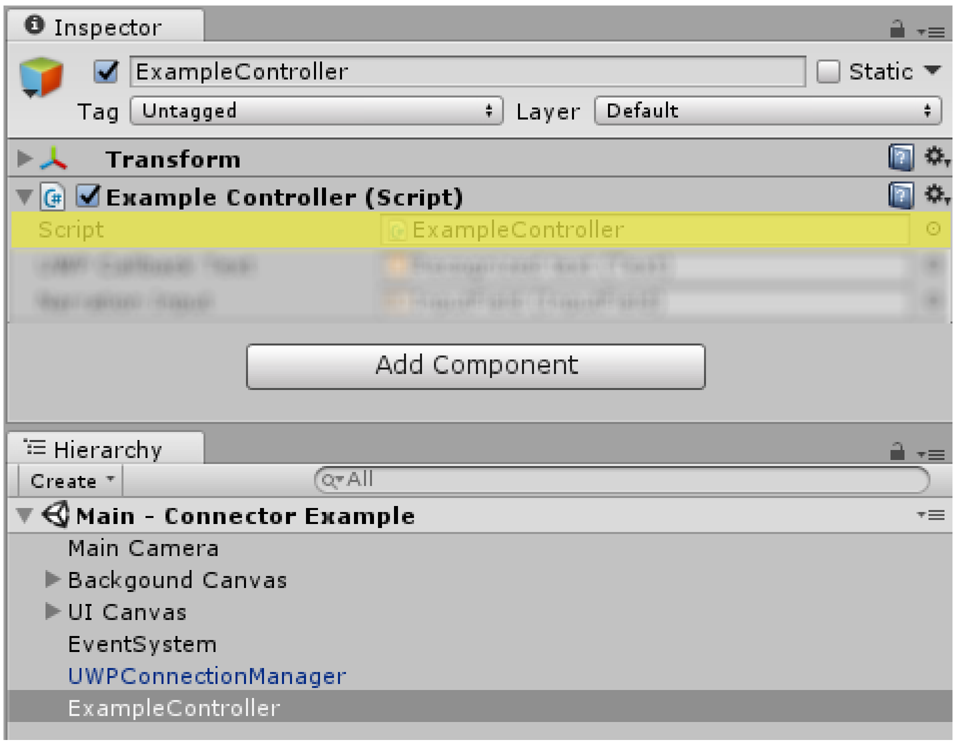
You need to create a MonoBehaviour class implementing the interface `IUWPReceiver`. This interface has only one function prototype:

```
void ReceiveFromUWP(object arg);
```

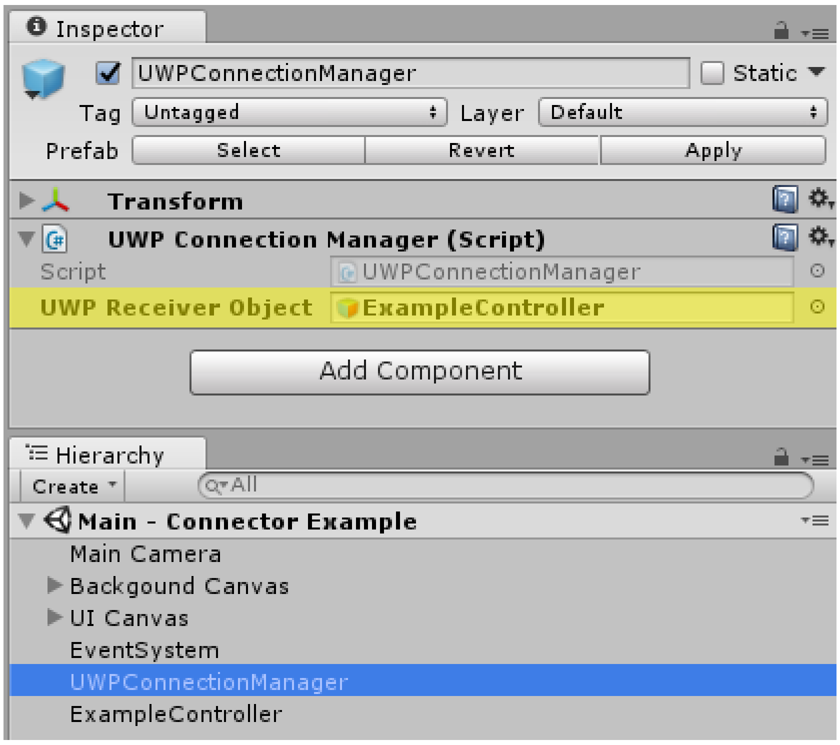
Typically, this implementation will happen on the scene controller, so the header of the controller must look something like this:

```
public class ExampleController : MonoBehaviour, IUWPReceiver
```

Add this script to a game object:


Inspector
ExampleController
Static
Tag Untagged
Layer Default
Transform
Example Controller (Script)
Script ExampleController
Add Component
Hierarchy
Create
All
Main - Connector Example
Main Camera
Backgound Canvas
UI Canvas
EventSystem
UWPConnectionManager
ExampleController

And then, drag this game object to the field **UWP Receiver Object** of the *UWPConnectionManager* prefab:



UWPConnectionManager
Tag: Untagged
Layer: Default
Prefab
UWP Connection Manager (Script)
Script: UWPConnectionManager
UWP Receiver Object: ExampleController

Main - Connector Example
Main Camera
Backgound Canvas
UI Canvas
EventSystem
UWPConnectionManager
ExampleController

When a message is sent from your WSA app, the *UWPConnectionManager* prefab will received it and check if there is an `IUWPReceiver` game object linked as described above. If there is, the function `void ReceiveFromUnity(object arg)` will be invoked, relaying the message to the controller object.

USING THE ASSET IN MULTIPLE SCENES

As explained before, the prefab UWPConnectionManager will exist in all your scenes because it uses the *don't destroy on load* function. You will be able to access it to send message to your WSA app as described above even if you change scene.

However, to receive messages you need to have a game object, typically the game controller, linked to the prefab, as explained in the previous section. When you change scenes, this game controller object is normally destroyed, so this link to the prefab will be broken. What you need to do to overcome this is to drag the prefab into the new scene, and link the new scene game controller to the prefab. The new prefab will link the original prefab with the new scene controller, and then will auto-destroy itself. In this way, there will still be only one prefab running and accessible to send messages to the WSA app, with the current scene controller linked.

THE DEMO PROJECT

To demonstrate this asset, I have put together a demo app that you can download from the Microsoft Store here: https://www.microsoft.com/store/apps/9mxzlr596z4p

The scenes to build this app are included in the asset, so you can learn to set up the asset while building the app yourself.

A step by step tutorial on how to build the app can be found in a document along this one, in a presentation format.

ABOUT

This asset is published under the account of Corta Studios. You can learn more about Corta Studios here: http://cortastudios.com

If you run into any trouble or want to get in contact with me, please write to salvador@cortastudios.com

The asset is available for free, but if you like it, or consider that it has saved you some valuable time, you can help me back by:

- Rating the asset in the Asset Store, or write a review
- Invite me to a coffee, which I will gladly accept because it will boost my moral – and inspire me to share more code. Thank you!!
    - Paypal (https://www.paypal.me/salvadorjesus)
    - Bitcoin (bitcoin:1NCra5L4yDZHarLarAuWfF5Fs9v6x4Acp1)

1NCra5L4yDZHarLarAuWfF5Fs9v6x4Acp1