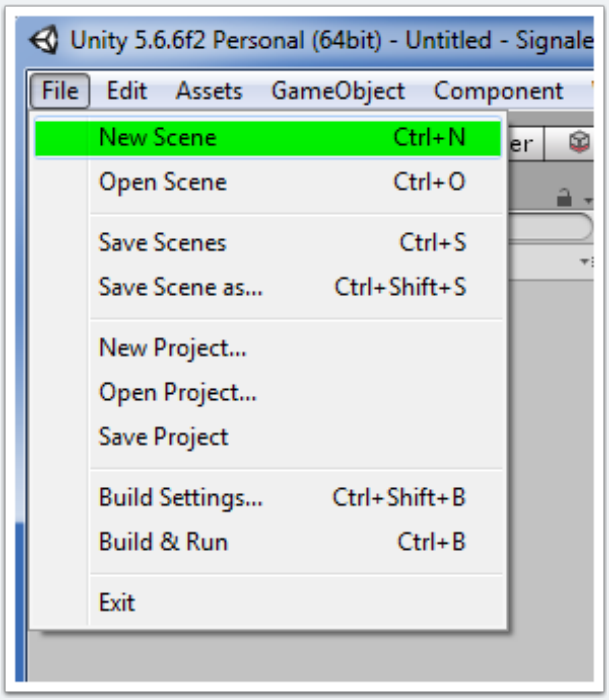
Signaler Quick Start

Quick introduction to setting up and using Signaler in your Unity projects.

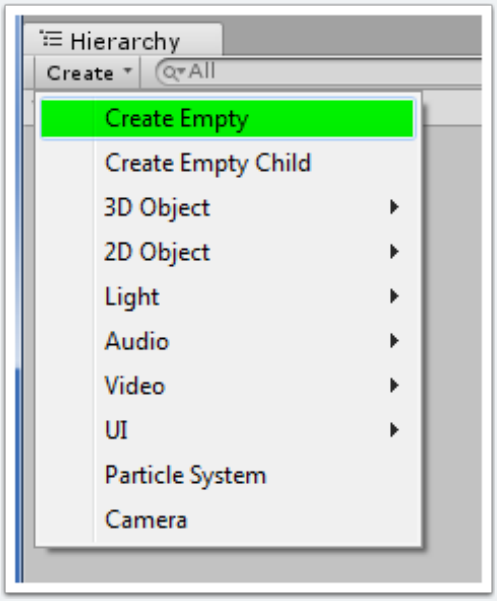
Create a New Scene

From the Unity editor menu, go to File > New Scene. Save the scene (Ctrl + S or File > Save Scenes) as QuickStart.



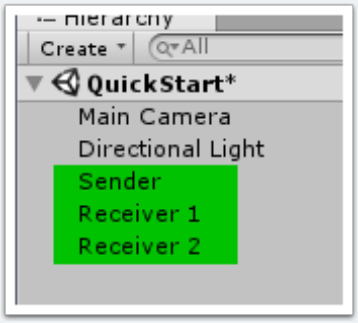
Create Gameobjects

In the scene hierarchy, create three new empty game objects. Click the Create > Create Empty menu item three times.



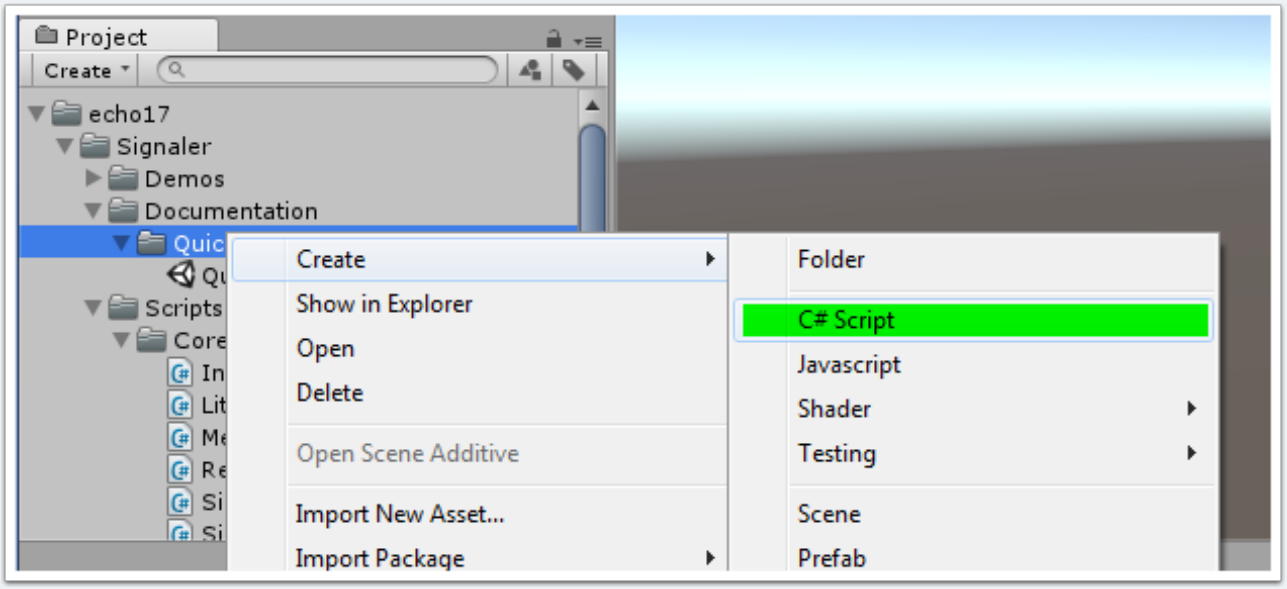
Name Gameobjects

Name the gameobjects just created as Sender, Receiver 1, and Receiver 2.



Create Sender Script

Create a new C# script and name it Sender in the Project window. Right-click the folder where the script should reside and select Create > C# Script.

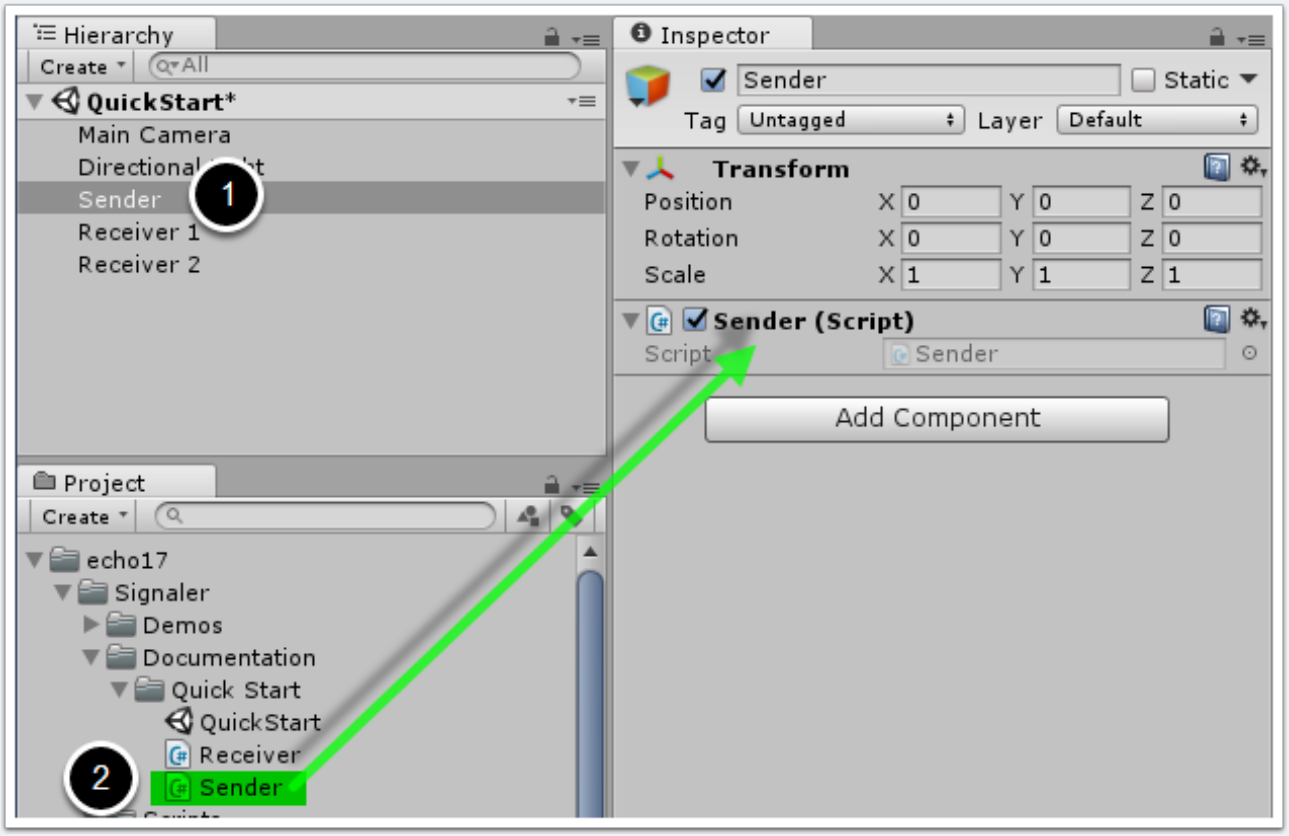


Create Receiver Script

Do the same thing as the above step, but name the file Receiver this time.

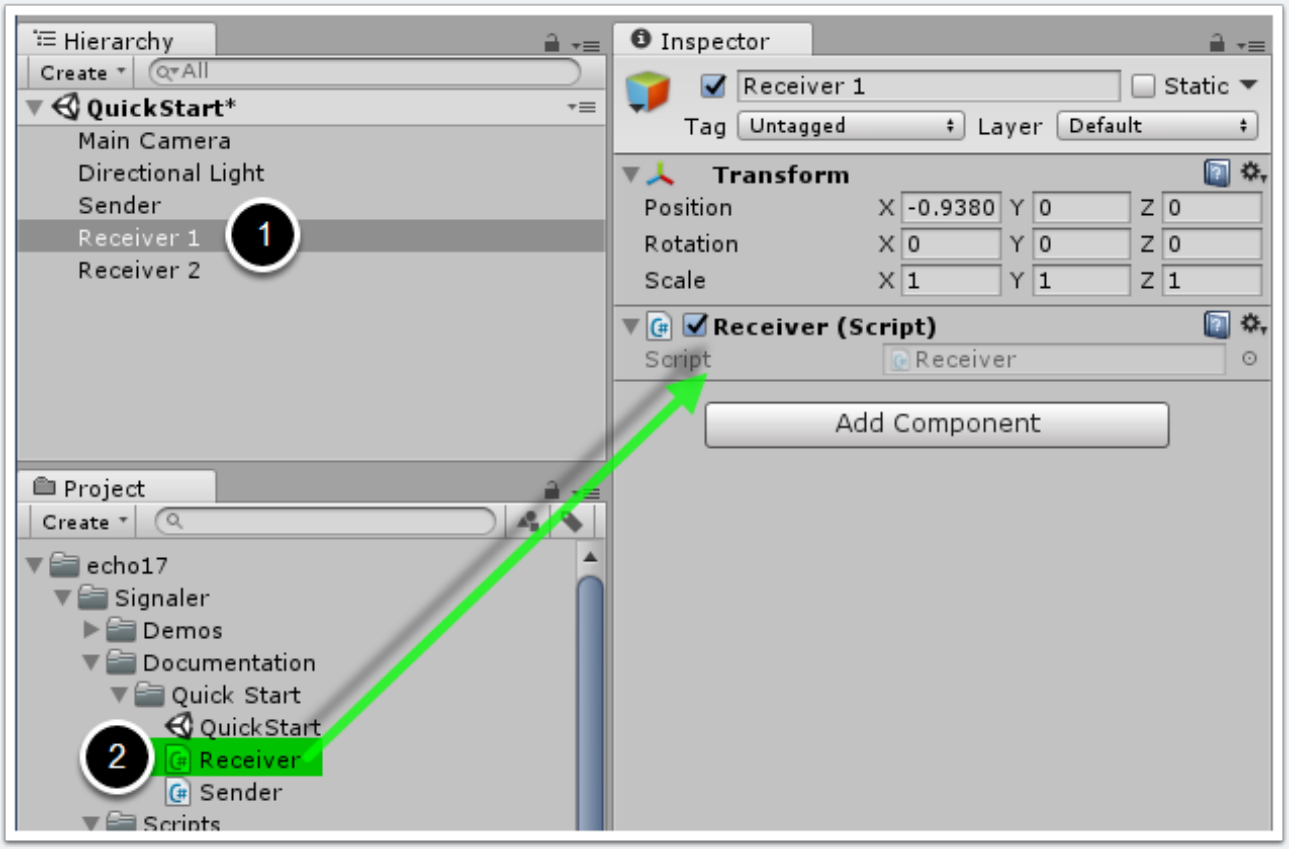
Attach Sender Script to Gameobject

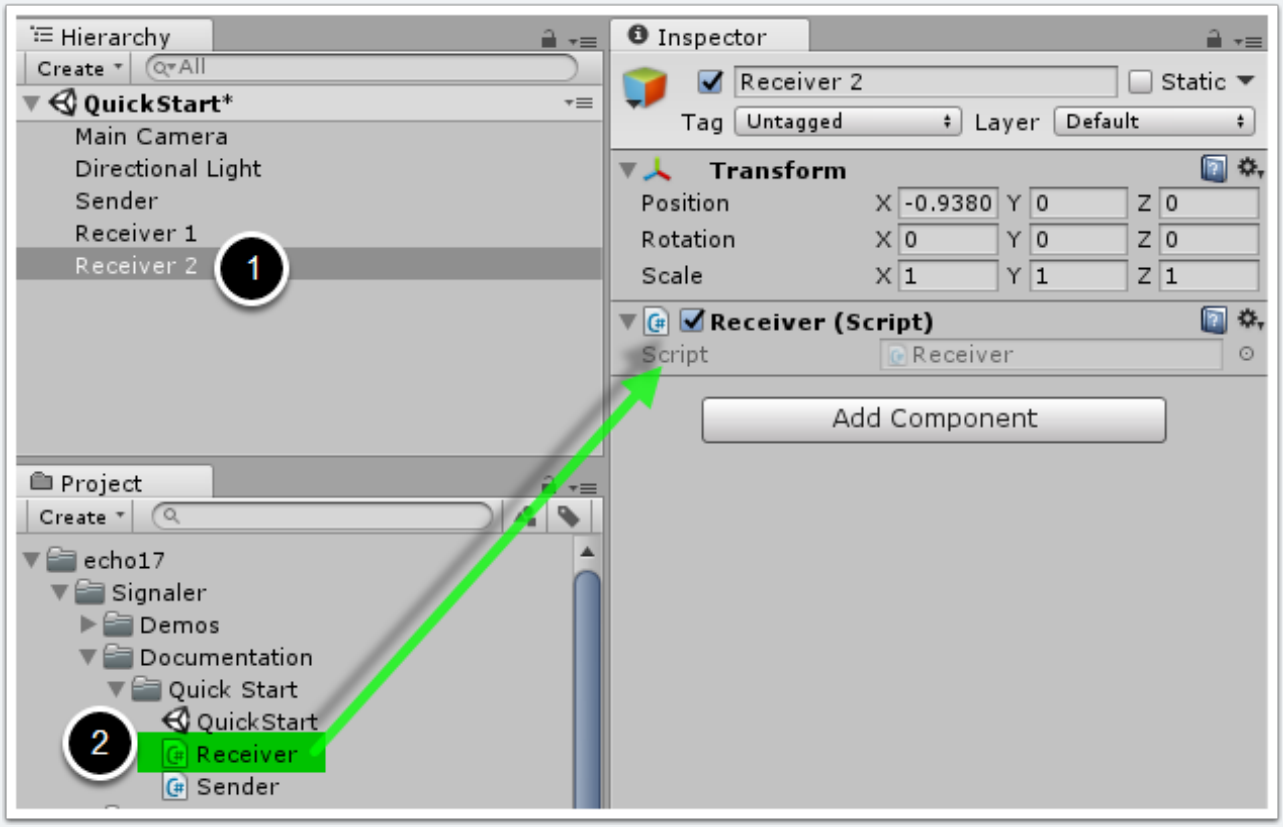
Click on the Sender gameobject in the scene hierarchy. Drag the Sender script from your Project window to the Sender gameobject's inspector to attach the component.



Attach Receiver Script to Gameobject

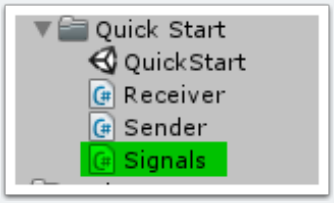
Click on each Receiver gameobject in the scene hierarchy. Drag the Receiver script from your Project window to each Receiver gameobject's inspector to attach the component.





Create Signals Script

Create a new C# script and name it Signals in the Project window.



Code Signals Script

Open Signals script and change the code to the following:

```
public struct MySignal
{
    public string message;
}
```

This is a simple struct that will convey a string message from a sender to all receivers subscribed to it. Note that we can use structs or classes, but since signals are generally immutable (they do not change after they are created), a struct is more efficient and will not cause memory allocations.

Code Sender Script

Open the Sender script and change the code to the following:

```
using UnityEngine;
using echo17.Signaler.Core;

public class Sender : MonoBehaviour, IBroadcaster
{
   void Start ()
   {
        var signal = new MySignal()
        {
            message = "Hello World!"
        };

        Signaler.Instance.Broadcast<MySignal>(this, signal: signal);
```

```
        }
}
```

This sets up a simple class that will broadcast a signal. Note that we need to use the echo17.Signaler.Core library. Also note that the Sender class inherits the IBroadcaster interface. This allows us to pass that this class sent out the broadcast. The Broadcast method is called on the Signaler singleton instance with the MySignal signal type. We pass this sender as the broadcaster and a simple "Hello World" message. There are other options when broadcasting, like what group to send to, but we will leave these at the default in this quick start.

We do the broadcast in the Start method to ensure that other classes have had a chance to subscribe to the signal in their own Awake methods. We could broadcast from elsewhere, like an event or Update method, but for this tutorial we will just use the Start method.

Code Receiver Script

Open the Receiver script and change the code to the following:

```
using UnityEngine;
using echo17.Signaler.Core;

public class Receiver : MonoBehaviour, ISubscriber
{
   void Awake()
   {
        Signaler.Instance.Subscribe<MySignal>(this, OnMySignal);
   }

   private bool OnMySignal(MySignal signal)
   {
```

```
        Debug.Log(name + " received signal. Message = '" + signal.message + "'");

        return true;
    }
}
```

This sets up a simple receiver class that will subscribe to the MySignal signal broadcast, printing the message it finds from the signal to the console. Note that Receiver inherits from the ISubscriber interface to allow us to know what receivers heard the broadcast. The Awake method is used to subscribe to the signal, passing the Signal type, the subscriber, and the method to call when the signal is received. There are other optional parameters, such as the filter and order, that we will not use in this tutorial.

We use the Awake method to ensure that the signal is subscribed to before the broadcast is sent out.

The function OnMySignal handles the signal, debugging the results to the console. We return true to indicate that the reciever did get the broadcast. If we return false here, the broadcaster will think that this particular receiver did not hear the signal. This can be useful if you are filtering in the handling function, but in this tutorial we just allow all receivers to hear the broadcast.

Run Scene

Run the scene and look at the results in the Console.