FUSE SDK
UNITY WRAPPER
START GUIDE

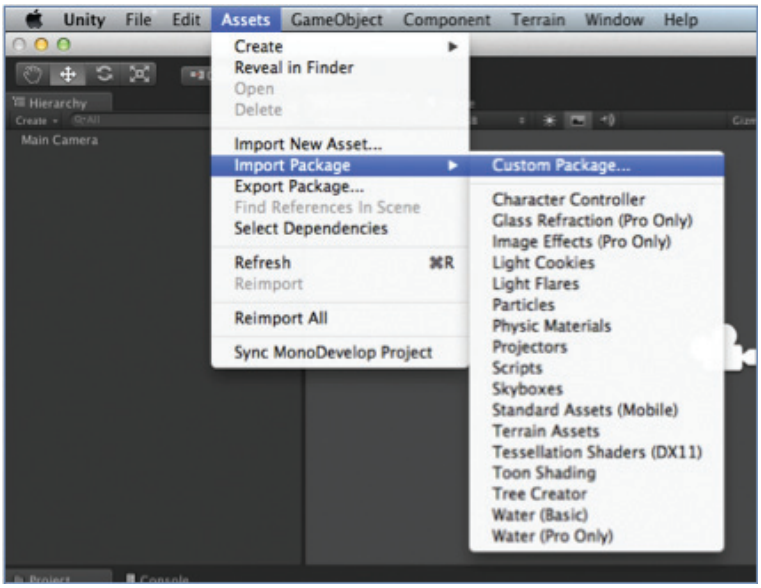
For iOS and Android Applications

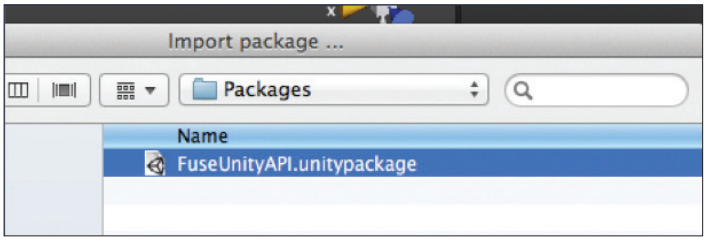
STEP ONE:

Download the Unity wrapper package from github: **https://github.com/fusepowered**

STEP TWO:

In your Unity project, import the package:





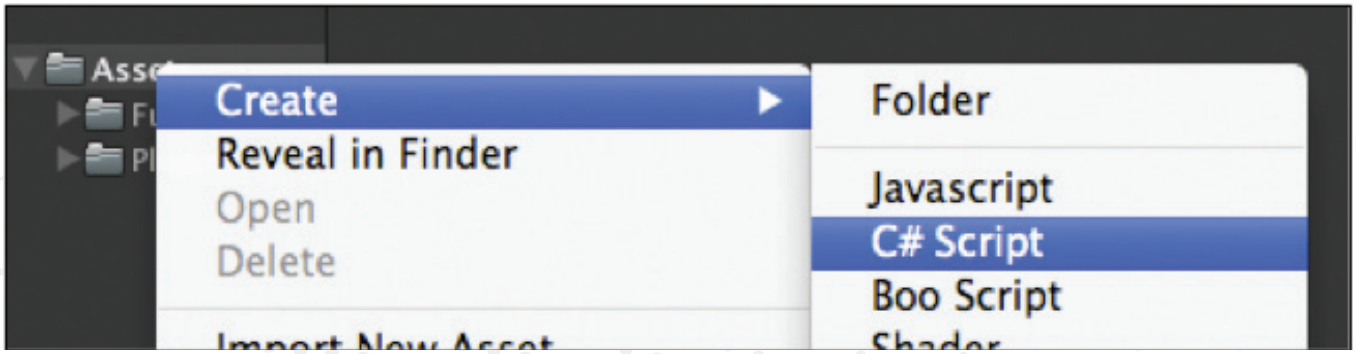
NOTE!

you can also drag-and-drop the contents of the "Assets" folder from the wrapper directory to your assets directory for your project.

STEP THREE:

Create a script to call the API functions.

For this example we'll use a C# script:



STEP FOUR:

The first step is to add a "Start Session" call:

```
using UnityEngine;
using System.Collections;

public class Fuseboxx : MonoBehaviour {

    private string logOutput;

    // Use this for initialization
    void Start () {
        RegisterActions();
        FuseAPI.StartSession ("<APP API KEY GOES HERE>");
    }
```

NOTE!

Fuseboxx™ won't show sessions for your app in real-time, as they are aggregated on the hour. However you can check to see if communication between your app and Fuseboxx™ is working by going to the "Your API Keys" page (Admin -> Integrate SDK -> Your API Keys) and seeing if the SDK version is being reported.

STEP FIVE:

To call an ad, first we register some actions and create some functions:

```
private void RegisterActions()
{
    FuseAPI.SessionStartReceived += sessionStarted;
    FuseAPI.AdAvailabilityResponse += adAvailabilityResponse;
    FuseAPI.AdWillClose += adWillClose;
}

private void sessionStarted()
{
    FuseAPI.CheckAdAvailable();
}

private void adAvailabilityResponse (int isAdAvailable, int hasError)
{
    if (isAdAvailable == 1)
    {
        FuseAPI.ShowAd();
    }
}

private void adWillClose()
{
    // Do something...
}
```

In this example the SessionStartReceived callback will trigger CheckAdAvailable, and this function will both check whether a Fuse Ad is available to be served, and if it is, will also automatically cache the ad.

The AdAvailabilityResponse callback will tell you if that ad is available or if there's an error. This example shows using the returned value of "1" (meaning an ad is available) to trigger showing the ad.

You can pick up the closing of an ad with the AdWillClose callback, and execute further code if you choose to.

STEP SIX:

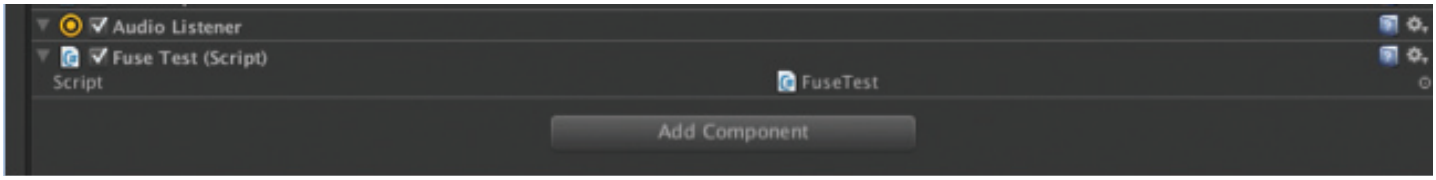
Finally, we complete the script by unregistering the actions:

```
private void UnRegisterActions()
{
    FuseAPI.SessionStartReceived -= sessionStarted;
    FuseAPI.AdAvailabilityResponse -= adAvailabilityResponse;
    FuseAPI.AdWillClose -= adWillClose;
}

void OnDestroy()
{
    UnRegisterActions();
}
```

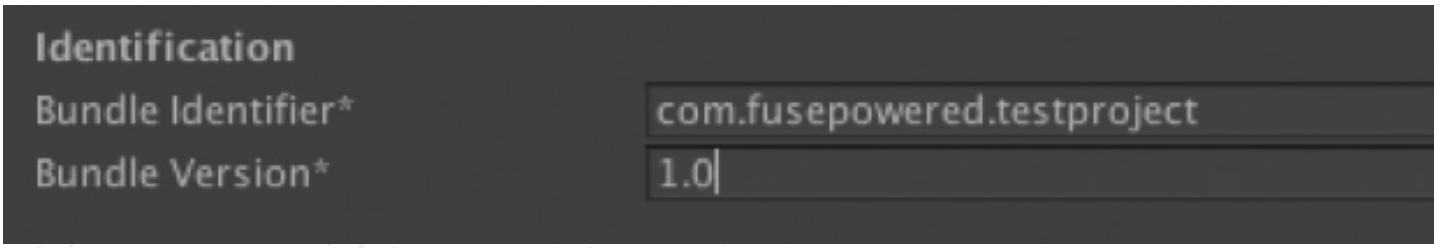
STEP SEVEN:

Make sure that the C# script is attached as a component within your scene.

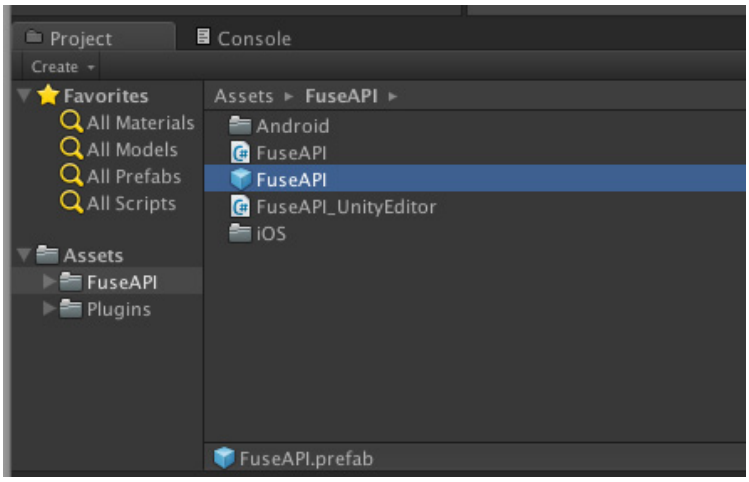


STEP EIGHT:

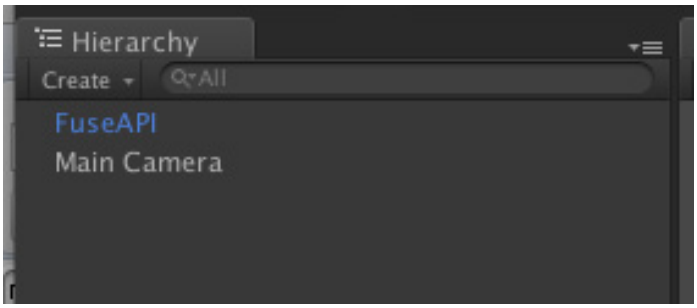
To test on a device, you'll first need to update the Bundle Identifier to match your profile (in Unity: Edit -> Project Settings -> Player)



STEP NINE:



Next drag the FuseAPI prefab to the hierarchy.



NOTE!

For both iOS and Android platforms, the Fuse Unity wrapper will automatically collect push notification tokens.

STEP TEN:

Next, choose "Build & Run" from the File menu in Unity. Make sure you've set the build platform properly first! (File -> Build Settings...)

NOTE!

If you are using Prime31 plugins for your in-app store, Fuseboxx can automatically track your user's in-app purchases. Simply un-comment the first line in the C# script:

For iOS: #define USING_PRIME31_IOS (in Assets/FuseAPI/FuseAPI_Prime31StoreKit.cs)

For Android: #define USING_PRIME31_ANDROID (in Assets/FuseAPI/FuseAPI_Prime31_IAB.cs)

Also, you can register the PurchaseVerification action to know if a purchase has been successfully verified (iOS only).

This is the complete test script, including log output to the device screen so that you can see what's happening:

```
using UnityEngine;
using System.Collections;

public class Fuseboxx : MonoBehaviour {

	private string logOutput;

	// Use this for initialization
	void Start () {
		RegisterActions();
		FuseAPI.StartSession (“<APP API KEY GOES HERE>”);
	}

	private void RegisterActions()
	{
		FuseAPI.SessionStartReceived += sessionStarted;
		FuseAPI.AdAvailabilityResponse += adAvailabilityResponse;
		FuseAPI.AdWillClose += adWillClose;
		logOutput += “Actions registered \n”;
	}

	private void sessionStarted()
	{
		logOutput += “Session started \n”;
		FuseAPI.CheckAdAvailable();
	}

	private void adAvailabilityResponse (int isAdAvailable, int hasError)
	{
		if (isAdAvailable == 1)
		{
			logOutput += “Ad availability response: isAdAvailable = “ +
isAdAvailable + “, hasError = “+hasError+”\n”;
			FuseAPI.ShowAd();
		}
	}

	private void adWillClose()
	{
		// Do something...
		logOutput += “AdWillClose \n”;
	}

	private void UnRegisterActions()
	{
		FuseAPI.SessionStartReceived -= sessionStarted;
		FuseAPI.AdAvailabilityResponse -= adAvailabilityResponse;
		FuseAPI.AdWillClose -= adWillClose;
	}

	void OnDestroy()
	{
		UnRegisterActions();
	}

	void OnGUI()
	{
		GUI.Label(new Rect(100,100,200,200), logOutput);
	}
}
```