UNITE / 2025

Saving Development Time with Shared Unity Plugins and SDKs

Reusable Technology

Bruno Cicanci

Lead Software Engineer

Agenda

→ SDK development

→ Unity plugins

→ Test, Publish, Support

SDK development

Why?

- → Add new features or functionalities
- → Expand or customize libraries
- → Focus on what's needed for each project
- → More control over the source code
- → Share standardized solutions across projects

Mobile SDKs

Android library

Android, Amazon Fire

Apple framework

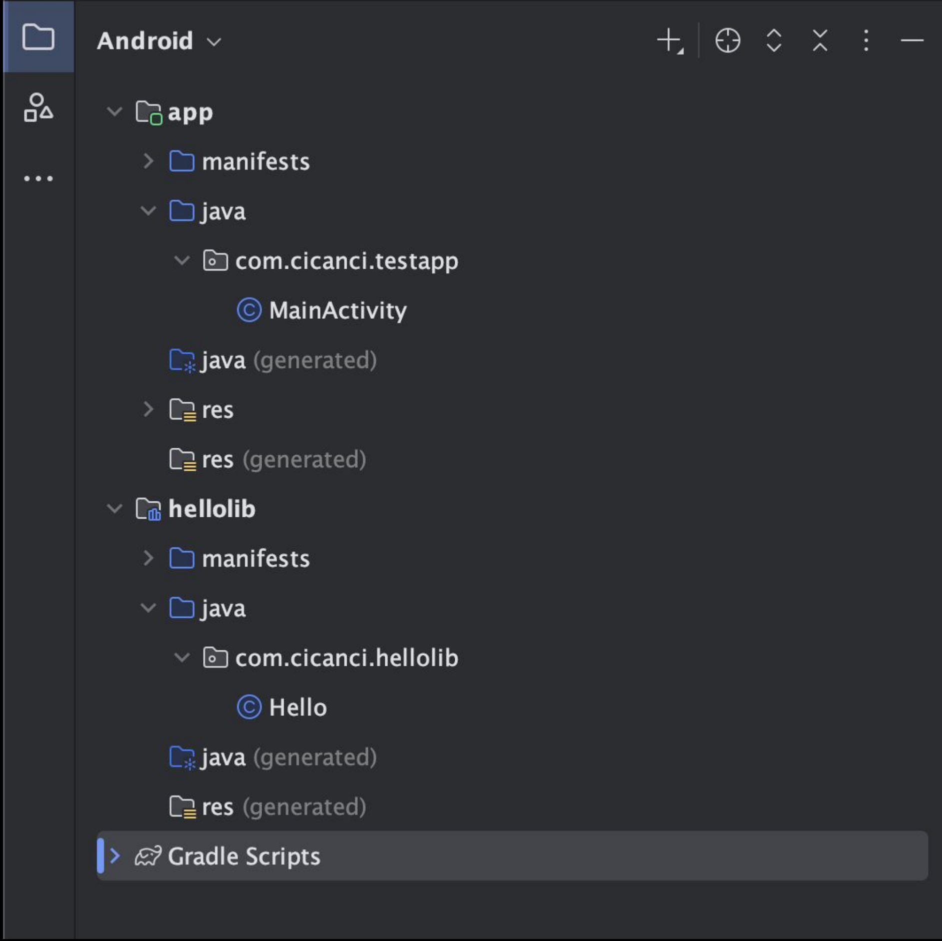
iOS, iOS simulator, macOS, tvOS

Android library

→ Access to OS APIs and many third-party libraries

→ Can use Java, Kotlin, C, C++

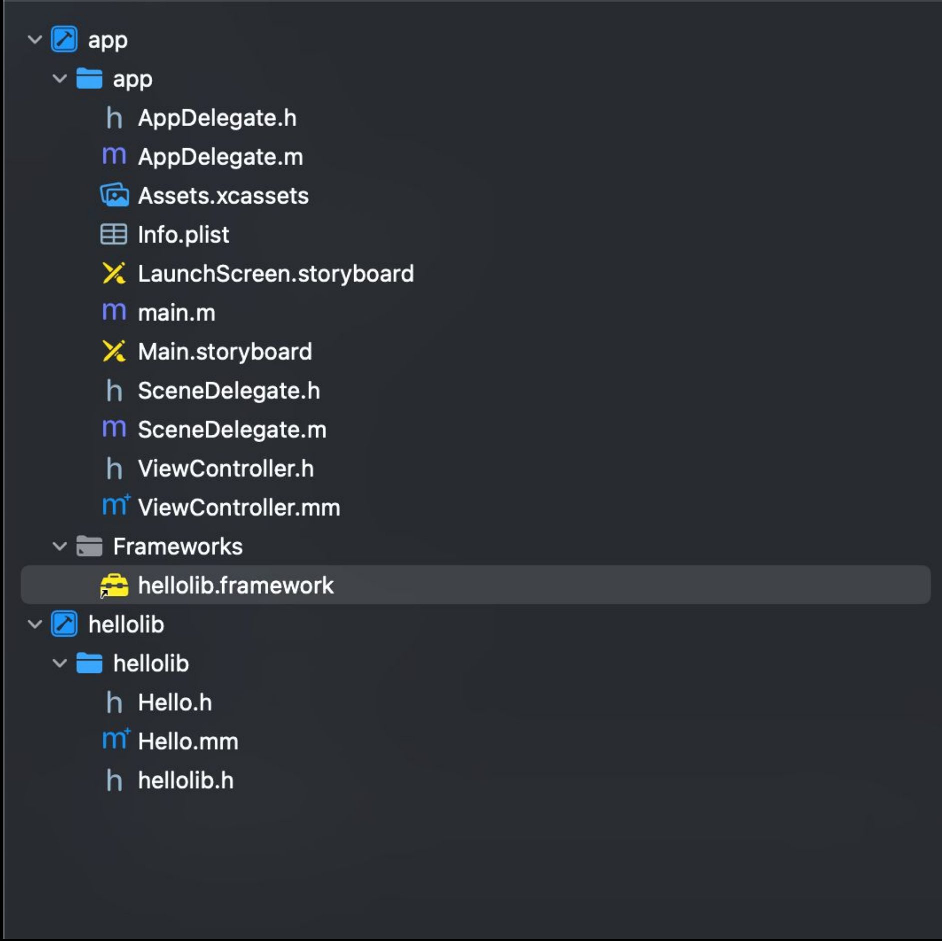
→ Shared as an AAR file


Android

```
app
  manifests
  java
    com.cicanci.testapp
      MainActivity
  java (generated)
  res
  res (generated)
hellolib
  manifests
  java
    com.cicanci.hellolib
      Hello
  java (generated)
  res (generated)
Gradle Scripts
```

Apple framework

- → Access to OS APIs and many frameworks
- → Can use Objective-C, Swift, C, C++
- → Shared as an XCFramework or bundle
- → Libraries can be Dynamic or Static



```
app
  app
    AppDelegate.h
    AppDelegate.m
    Assets.xcassets
    Info.plist
    LaunchScreen.storyboard
    main.m
    Main.storyboard
    SceneDelegate.h
    SceneDelegate.m
    ViewController.h
    ViewController.mm
  Frameworks
    hellolib.framework
hellolib
  hellolib
    Hello.h
    Hello.mm
    hellolib.h
```

Potential issues

→ Xcode upgrades might break compatibility - don't rush upgrades

→ Latest Java SDK and Gradle plugin versions can also cause build issues

Best practices

- → Use dependencies that are compatible with your minimum requirements
- → Always have a simple test app in your project
- → Expose only the APIs that are really required
- → Automate the build process
- → Use unit tests in the IDE

Unity plugins

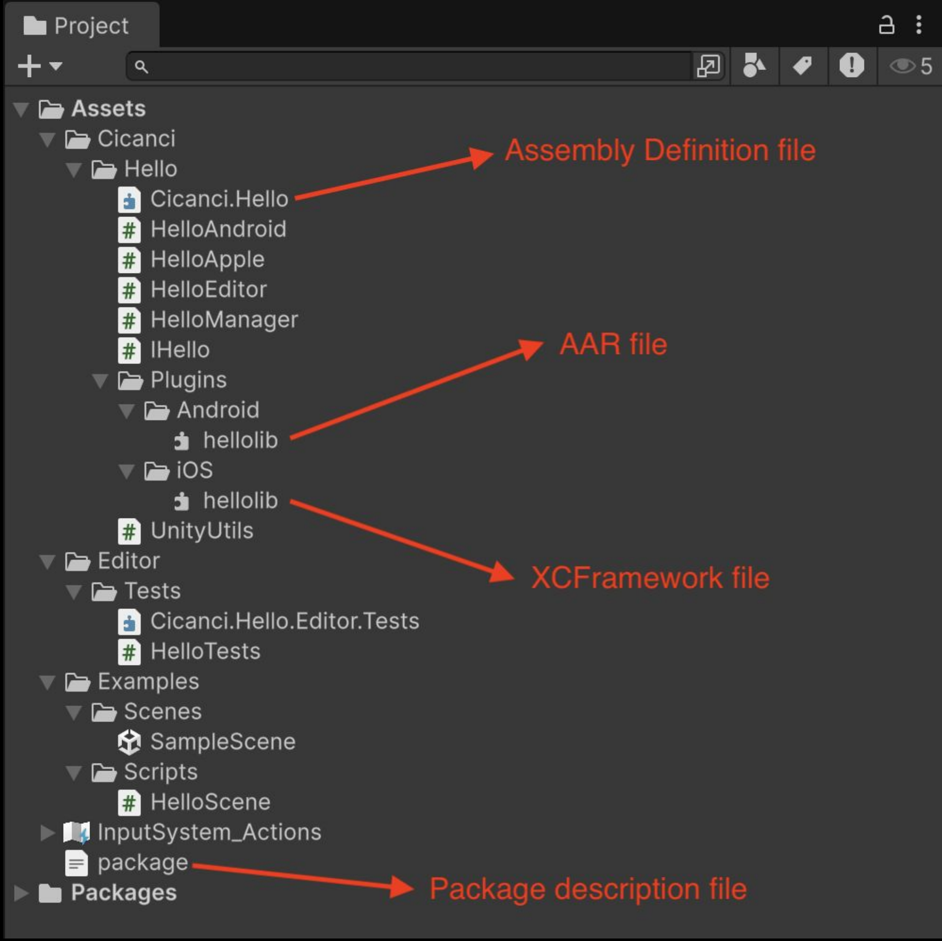
Project structure

→ Keep the plugin separated

→ Use Editor tests and examples

→ Pre-/post-process scripts and metas for configurations and dependencies

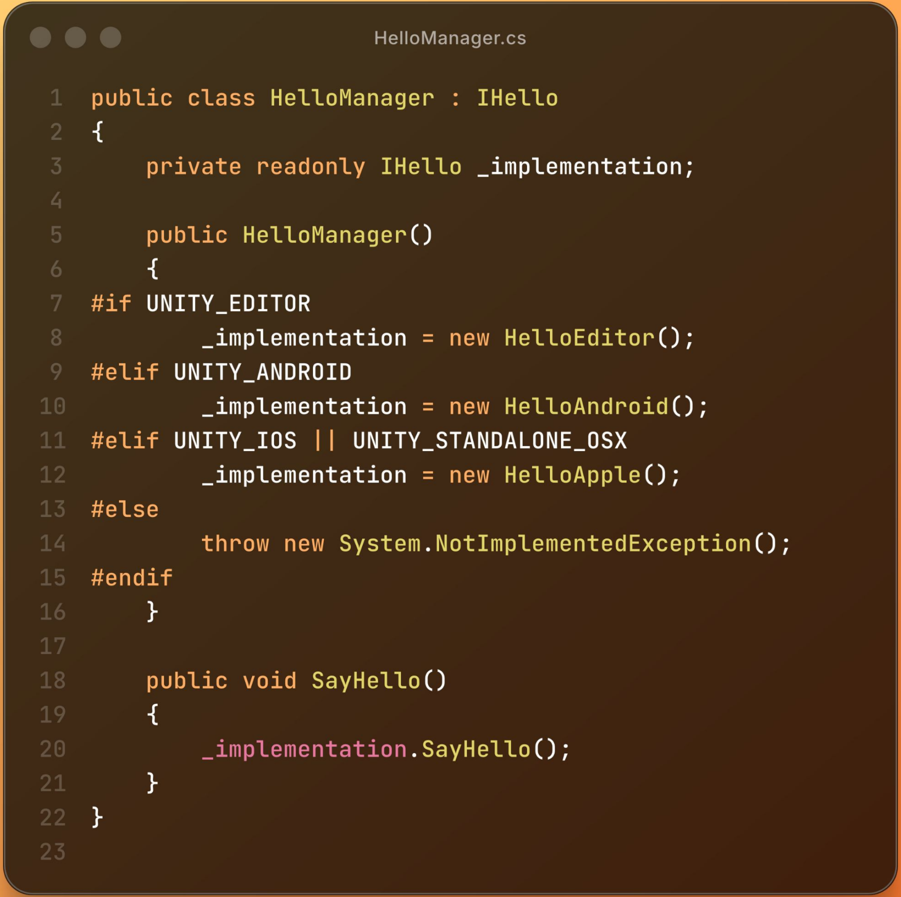
→ `Package.json` file for the package



```
Assets
  Cicanci
    Hello
      Cicanci.Hello
      HelloAndroid
      HelloApple
      HelloEditor
      HelloManager
      IHello
      Plugins
        Android
          hellolib
        iOS
          hellolib
      UnityUtils
    Editor
      Tests
        Cicanci.Hello.Editor.Tests
        HelloTests
    Examples
      Scenes
        SampleScene
      Scripts
        HelloScene
    InputSystem_Actions
    package
Packages
```

Common APIs

- → Have a single entry point to the plugin
- → Implement only the supported platforms
- → Avoid including logic that could be implemented inside the SDKs



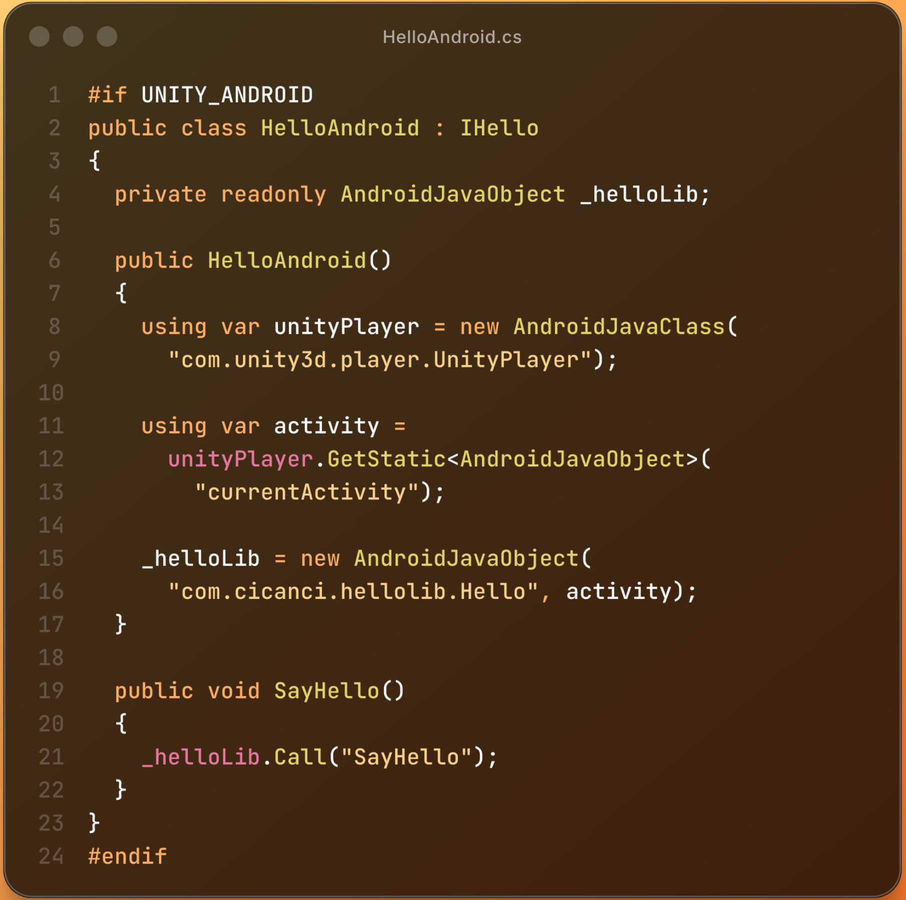
HelloManager.cs

```csharp
public class HelloManager : IHello
{
    private readonly IHello _implementation;

    public HelloManager()
    {
#if UNITY_EDITOR
        _implementation = new HelloEditor();
#elif UNITY_ANDROID
        _implementation = new HelloAndroid();
#elif UNITY_IOS || UNITY_STANDALONE_OSX
        _implementation = new HelloApple();
#else
        throw new System.NotImplementedException();
#endif
    }

    public void SayHello()
    {
        _implementation.SayHello();
    }
}

```

Android implementation

→ Use the main activity from Unity

→ UI related APIs and callbacks must be executed in the UI thread



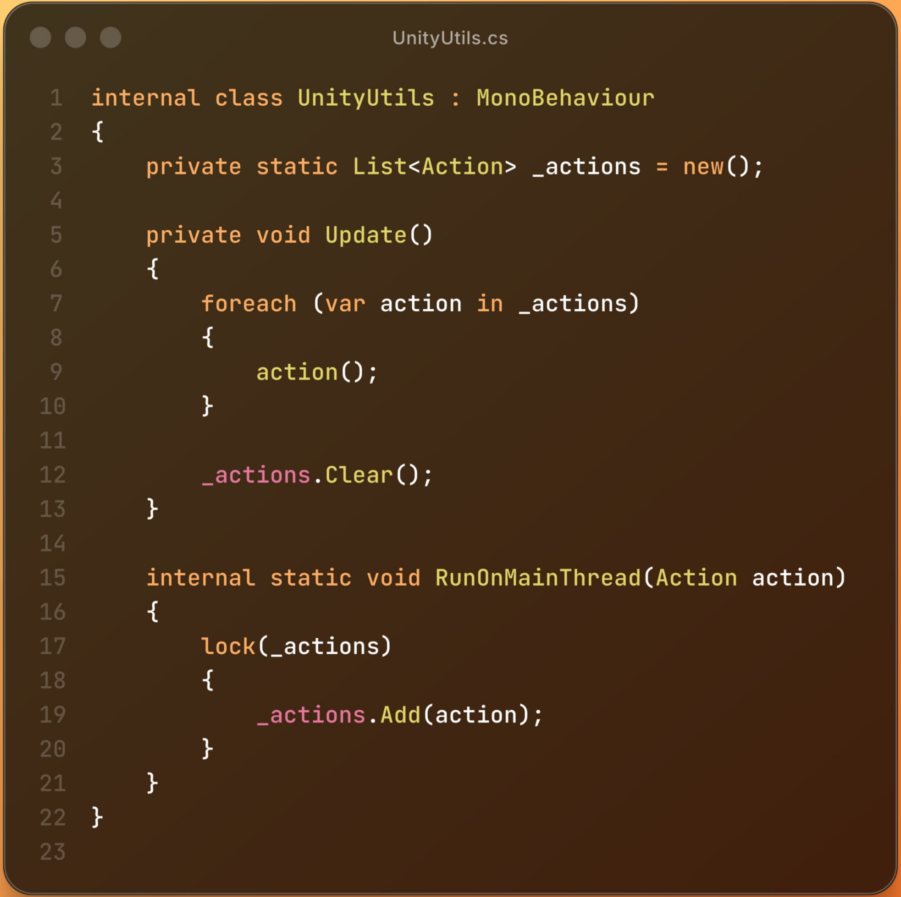
HelloAndroid.cs

```
#if UNITY_ANDROID
public class HelloAndroid : IHello
{
  private readonly AndroidJavaObject _helloLib;

  public HelloAndroid()
  {
    using var unityPlayer = new AndroidJavaClass(
      "com.unity3d.player.UnityPlayer");

    using var activity =
      unityPlayer.GetStatic<AndroidJavaObject>(
        "currentActivity");

    _helloLib = new AndroidJavaObject(
      "com.cicanci.hellolib.Hello", activity);
  }

  public void SayHello()
  {
    _helloLib.Call("SayHello");
  }
}
#endif
```

Running on main thread

→ DontDestroyOnLoad

→ Ensures actions run on the main thread


UnityUtils.cs

```csharp
internal class UnityUtils : MonoBehaviour
{
    private static List<Action> _actions = new();

    private void Update()
    {
        foreach (var action in _actions)
        {
            action();
        }

        _actions.Clear();
    }

    internal static void RunOnMainThread(Action action)
    {
        lock(_actions)
        {
            _actions.Add(action);
        }
    }
}

```

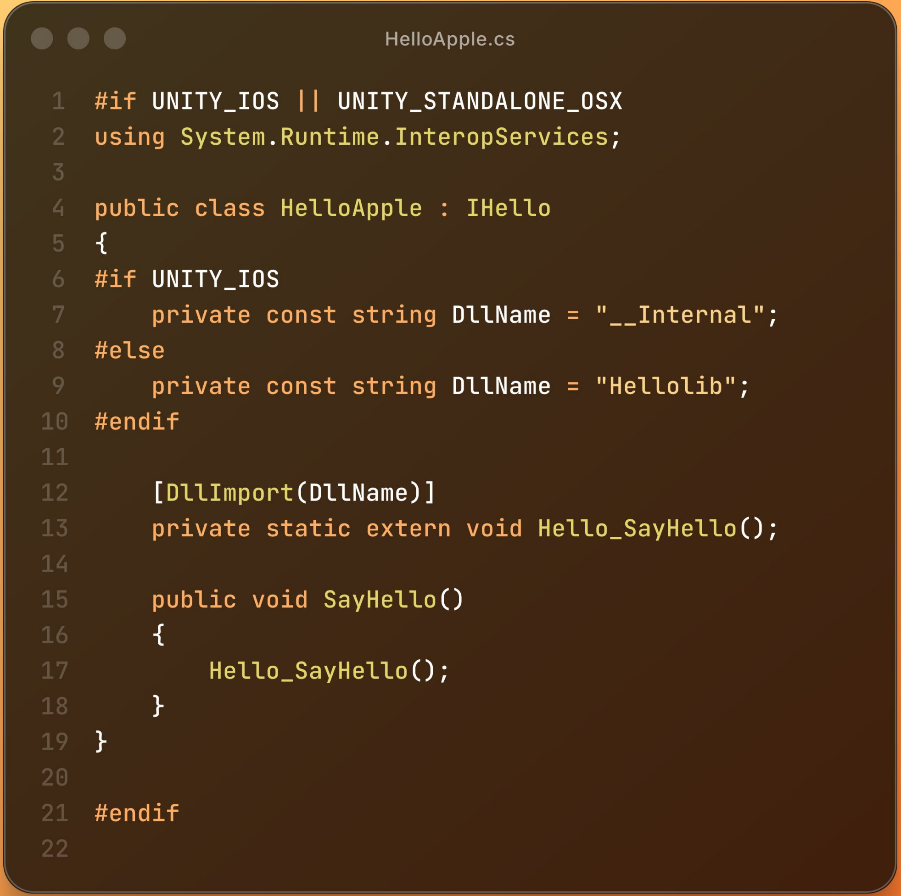
Apple implementation

→ iOS code can often be reused for macOS

  — `DLLImport` names may change when using bundles

→ Apple code requires extern methods

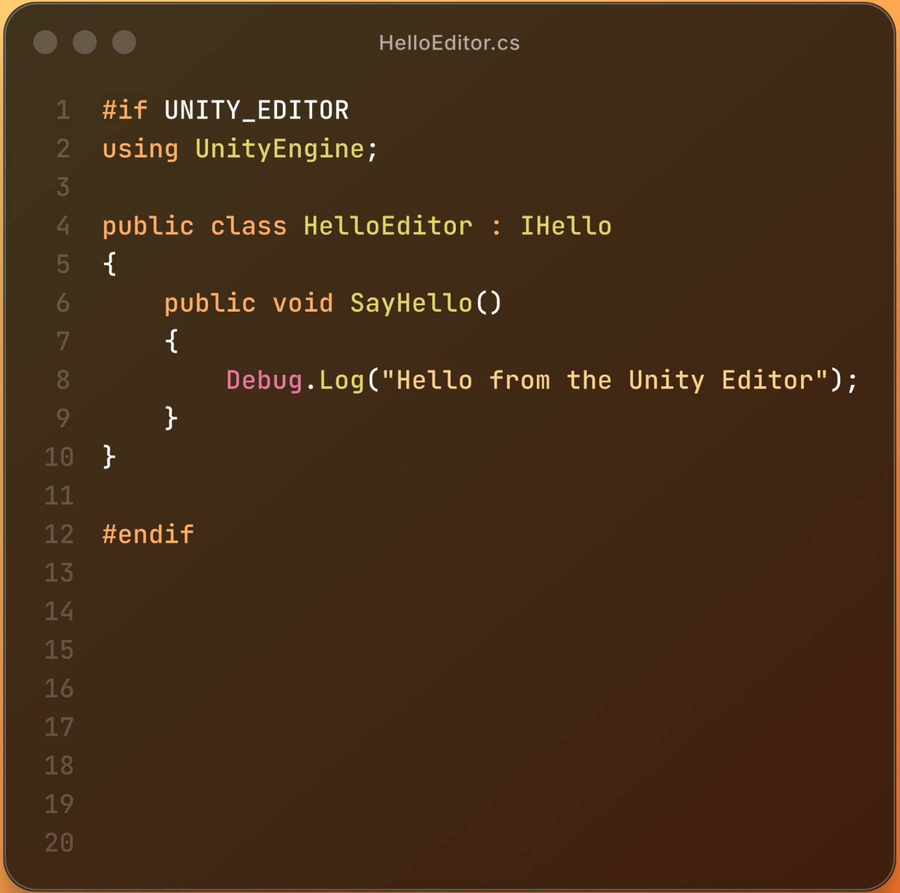
→ UI related APIs and callbacks must be executed in the UI thread


HelloApple.cs

```
#if UNITY_IOS || UNITY_STANDALONE_OSX
using System.Runtime.InteropServices;

public class HelloApple : IHello
{
#if UNITY_IOS
    private const string DllName = "__Internal";
#else
    private const string DllName = "Hellolib";
#endif

    [DllImport(DllName)]
    private static extern void Hello_SayHello();

    public void SayHello()
    {
        Hello_SayHello();
    }
}

#endif
```

Editor implementation

- → Always include a dummy implementation in the Editor
- → Consider adding success and error simulations

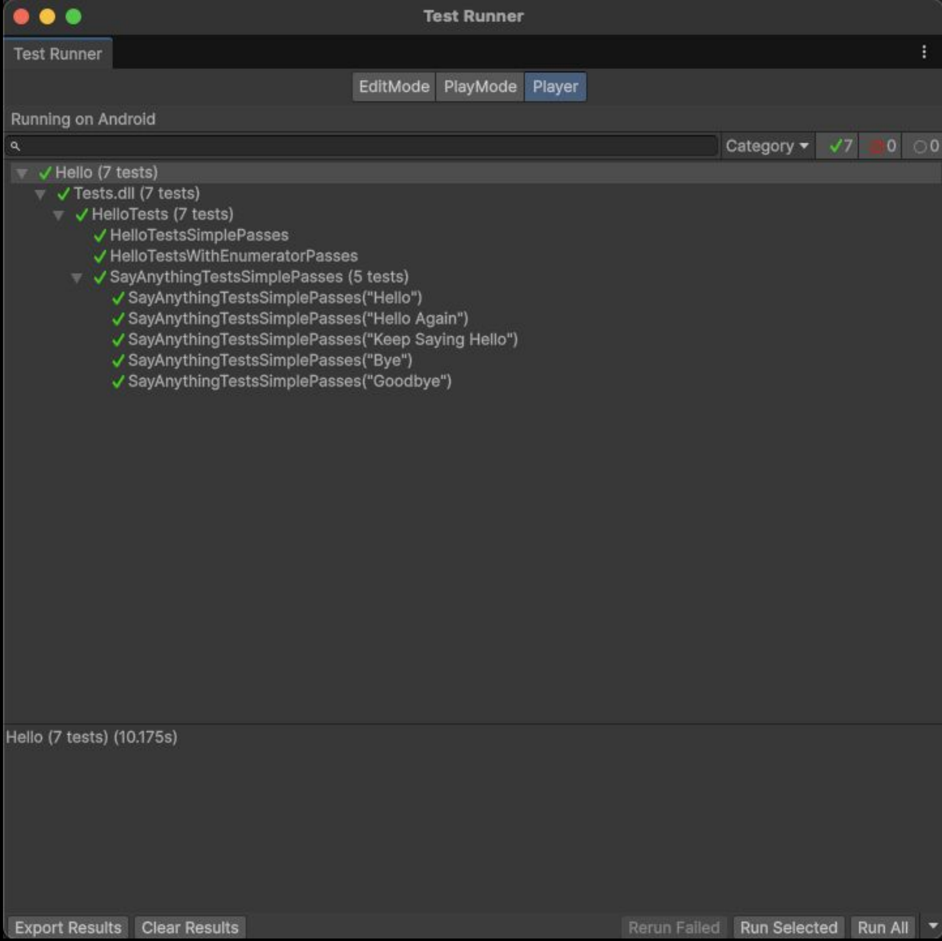

HelloEditor.cs

```
#if UNITY_EDITOR
using UnityEngine;

public class HelloEditor : IHello
{
    public void SayHello()
    {
        Debug.Log("Hello from the Unity Editor");
    }
}

#endif
```

Unit tests

→ Create Player mode tests that can run on connected devices or simulators

→ Each platform must be enabled in the Assembly Definition file



Running on Android

```
✓ Hello (7 tests)
  ✓ Tests.dll (7 tests)
    ✓ HelloTests (7 tests)
      ✓ HelloTestsSimplePasses
      ✓ HelloTestsWithEnumeratorPasses
      ✓ SayAnythingTestsSimplePasses (5 tests)
        ✓ SayAnythingTestsSimplePasses("Hello")
        ✓ SayAnythingTestsSimplePasses("Hello Again")
        ✓ SayAnythingTestsSimplePasses("Keep Saying Hello")
        ✓ SayAnythingTestsSimplePasses("Bye")
        ✓ SayAnythingTestsSimplePasses("Goodbye")
```

Hello (7 tests) (10.175s)

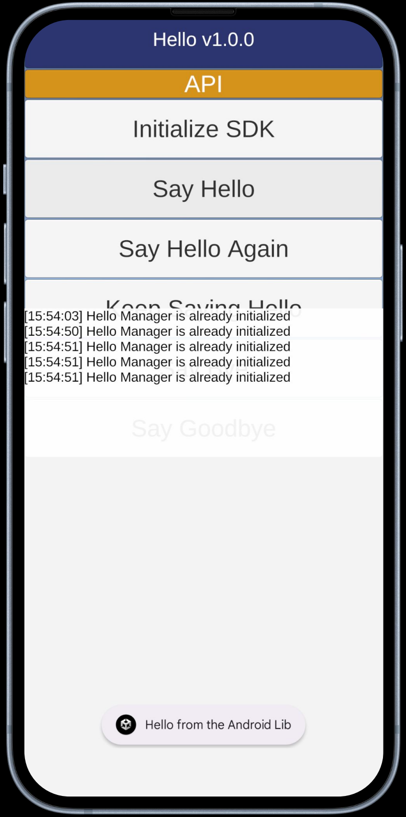
Test scene

Simple test scene to test the APIs

→ Include minimal UI, focused on each API call

→ Keep it simple yet professional

→ Use it to test different Android and iOS versions



```
[15:54:03] Hello Manager is already initialized
[15:54:50] Hello Manager is already initialized
[15:54:51] Hello Manager is already initialized
[15:54:51] Hello Manager is already initialized
[15:54:51] Hello Manager is already initialized
```

Potential issues

→ Dynamic library needs to be embedded and signed on UnityFramework target in Xcode

→ Any SDK dependency must also be included in the package

→ Alternatively, you can use Google's External Dependency Manager

Best practices

→ Include libraries in the Plugins folder (remember to set up metas)

→ Develop the plugin in the minimum supported Unity version

→ Test scene to call each API

→ Simple C# interface to consume libraries, avoid too much logic

→ Always ready to build and run

What about C++?

→ Common layer that can consume both Android and Apple libraries

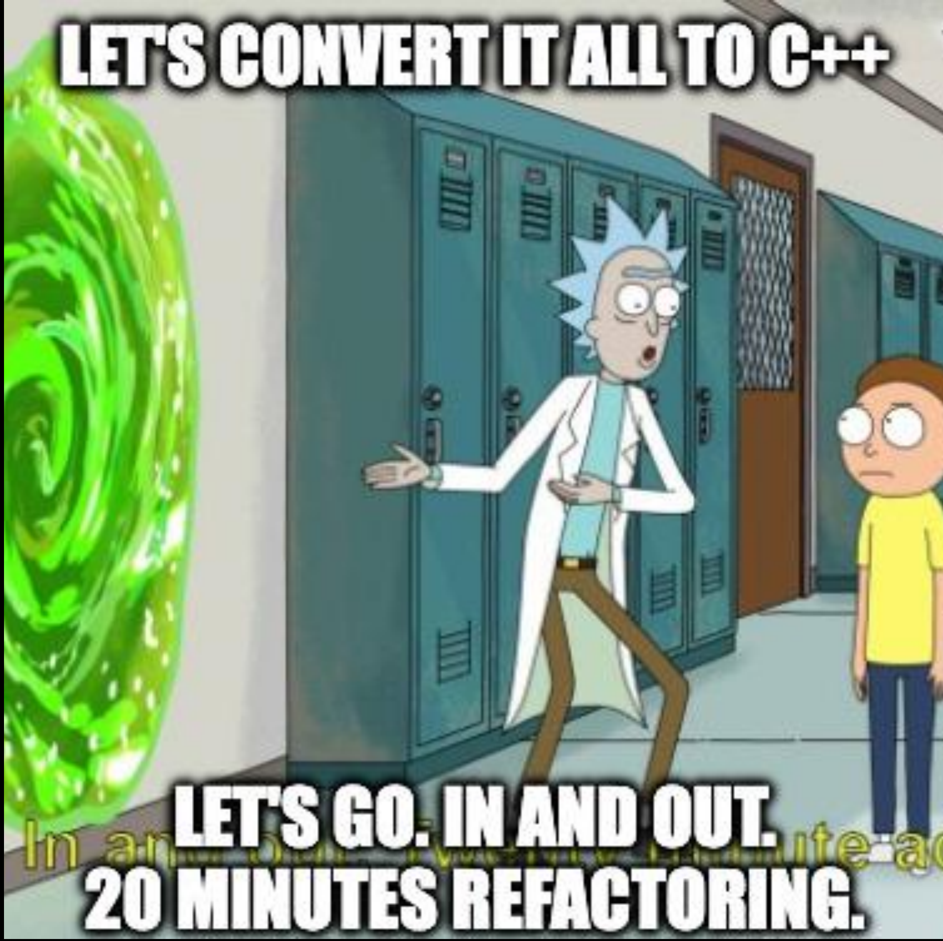
→ Can generate C# code through tools such as SWIG

Is it worth?

→ Can slow down development and debugging

→ Adds a lot of complexity to the project for just an interface

→ Crashes are more difficult to investigate



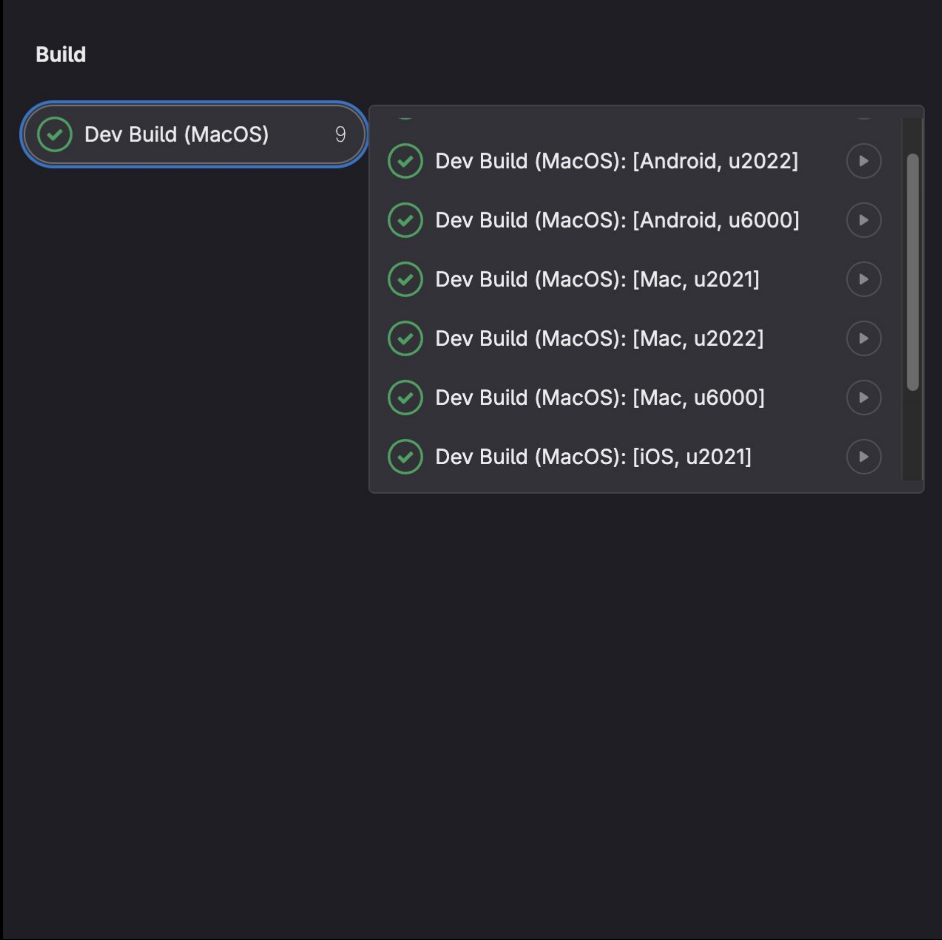
Test, Publish, Support

CI/CD

→ Ensure APIs are valid on each change

→ Build for all supported platforms

→ Run basic tests on each commit and full tests on each merge



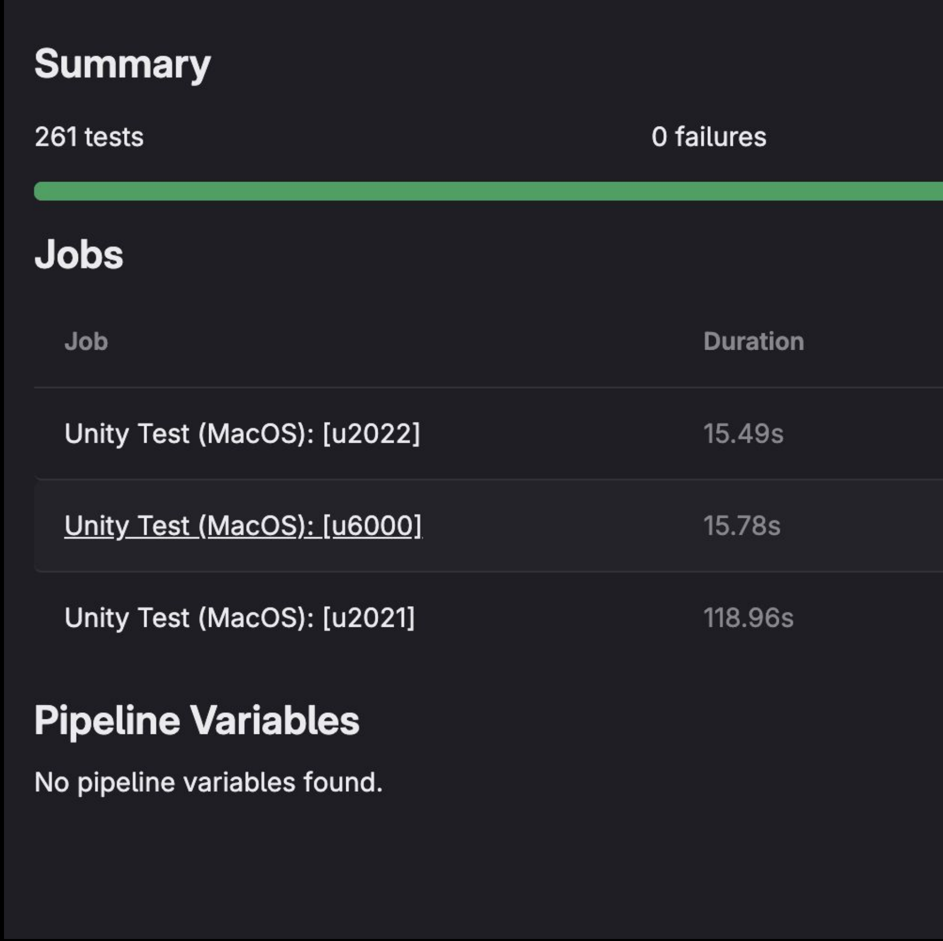
Build

Dev Build (MacOS) 9

Dev Build (MacOS): [Android, u2022]
Dev Build (MacOS): [Android, u6000]
Dev Build (MacOS): [Mac, u2021]
Dev Build (MacOS): [Mac, u2022]
Dev Build (MacOS): [Mac, u6000]
Dev Build (MacOS): [iOS, u2021]

Automation tests

- → Unit tests on different Unity versions
- → Run builds on devices or simulators to test OS-specific features



Summary

261 tests   0 failures

Jobs

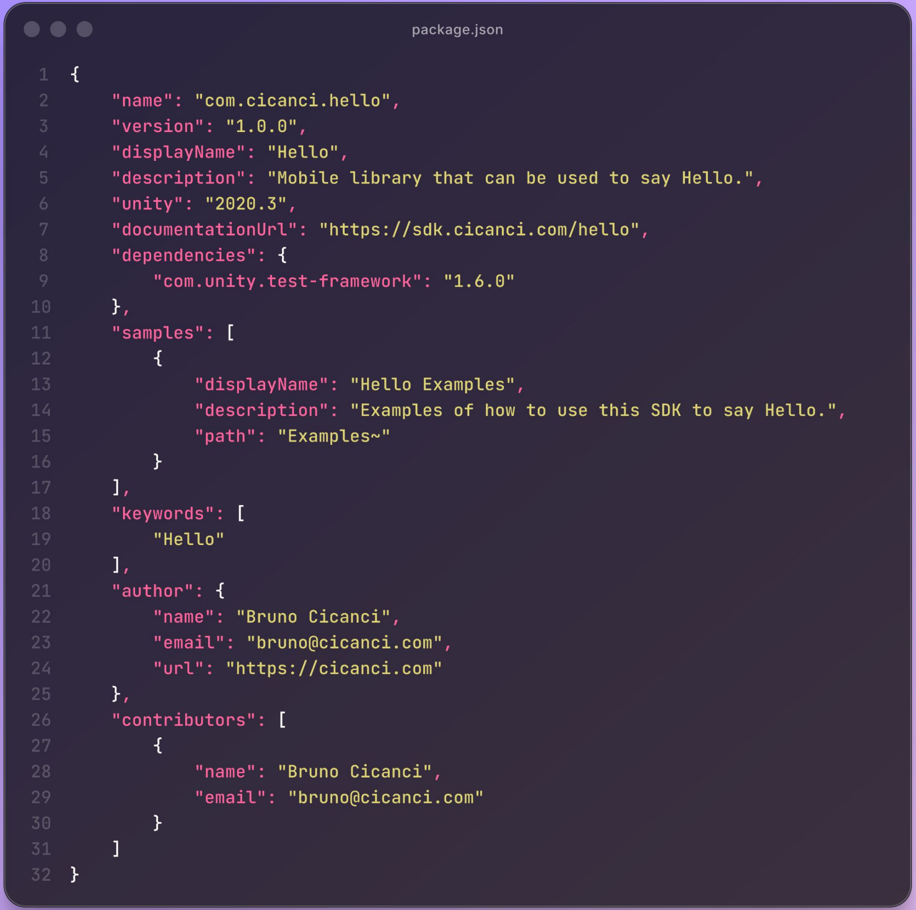
Job	Duration
Unity Test (MacOS): [u2022]	15.49s
Unity Test (MacOS): [u6000]	15.78s
Unity Test (MacOS): [u2021]	118.96s

Pipeline Variables

No pipeline variables found.

NPM package

- → Set up your plugin for Unity
- → Keep the version and dependencies updated
  - — Use the `x.y.z-preview.N` versioning pattern for preview releases
- → You can use hidden folders (~) to include more content



package.json

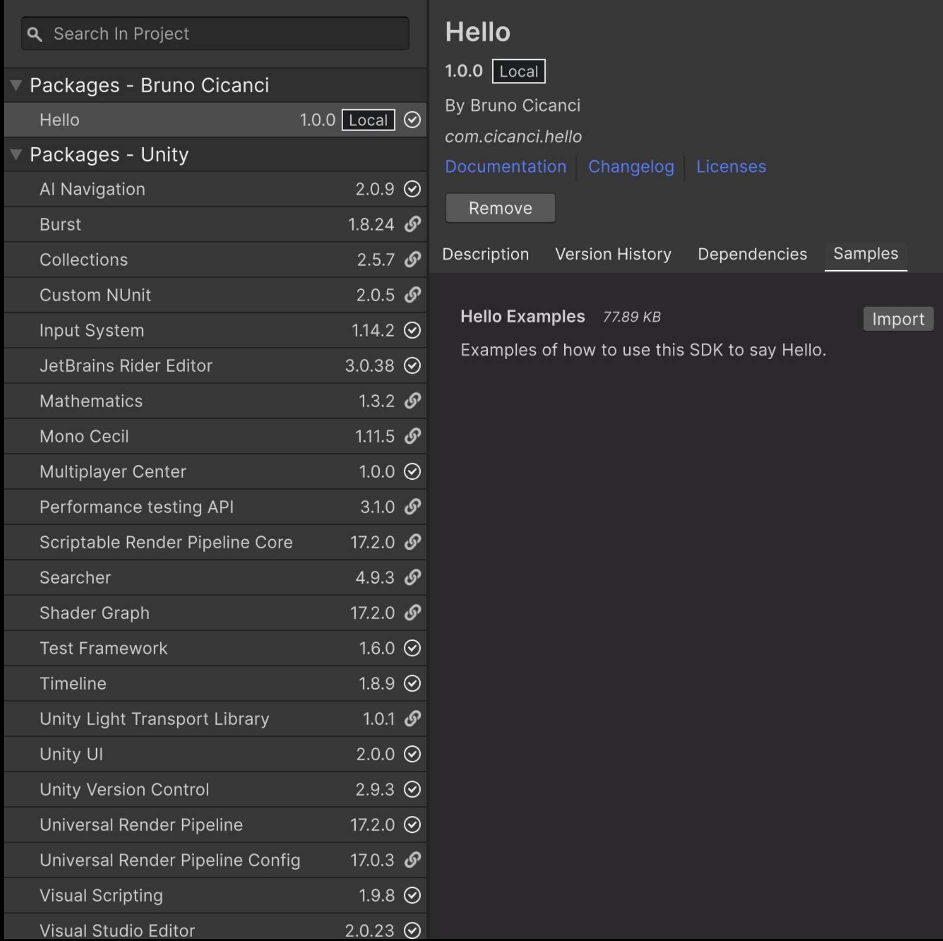
```json
{
    "name": "com.cicanci.hello",
    "version": "1.0.0",
    "displayName": "Hello",
    "description": "Mobile library that can be used to say Hello.",
    "unity": "2020.3",
    "documentationUrl": "https://sdk.cicanci.com/hello",
    "dependencies": {
        "com.unity.test-framework": "1.6.0"
    },
    "samples": [
        {
            "displayName": "Hello Examples",
            "description": "Examples of how to use this SDK to say Hello.",
            "path": "Examples~"
        }
    ],
    "keywords": [
        "Hello"
    ],
    "author": {
        "name": "Bruno Cicanci",
        "email": "bruno@cicanci.com",
        "url": "https://cicanci.com"
    },
    "contributors": [
        {
            "name": "Bruno Cicanci",
            "email": "bruno@cicanci.com"
        }
    ]
}
```

Unity package manager

→ Scoped registries

→ Git repositories

→ Local packages



Search In Project

Packages - Bruno Cicanci

Hello	1.0.0 Local	✓

Packages - Unity

AI Navigation	2.0.9	✓
Burst	1.8.24	🔗
Collections	2.5.7	🔗
Custom NUnit	2.0.5	🔗
Input System	1.14.2	✓
JetBrains Rider Editor	3.0.38	✓
Mathematics	1.3.2	🔗
Mono Cecil	1.11.5	🔗
Multiplayer Center	1.0.0	✓
Performance testing API	3.1.0	🔗
Scriptable Render Pipeline Core	17.2.0	🔗
Searcher	4.9.3	🔗
Shader Graph	17.2.0	🔗
Test Framework	1.6.0	✓
Timeline	1.8.9	✓
Unity Light Transport Library	1.0.1	🔗
Unity UI	2.0.0	✓
Unity Version Control	2.9.3	✓
Universal Render Pipeline	17.2.0	✓
Universal Render Pipeline Config	17.0.3	🔗
Visual Scripting	1.9.8	✓
Visual Studio Editor	2.0.23	✓

Hello

1.0.0 Local

By Bruno Cicanci

com.cicanci.hello

Documentation | Changelog | Licenses

Remove

Description Version History Dependencies Samples

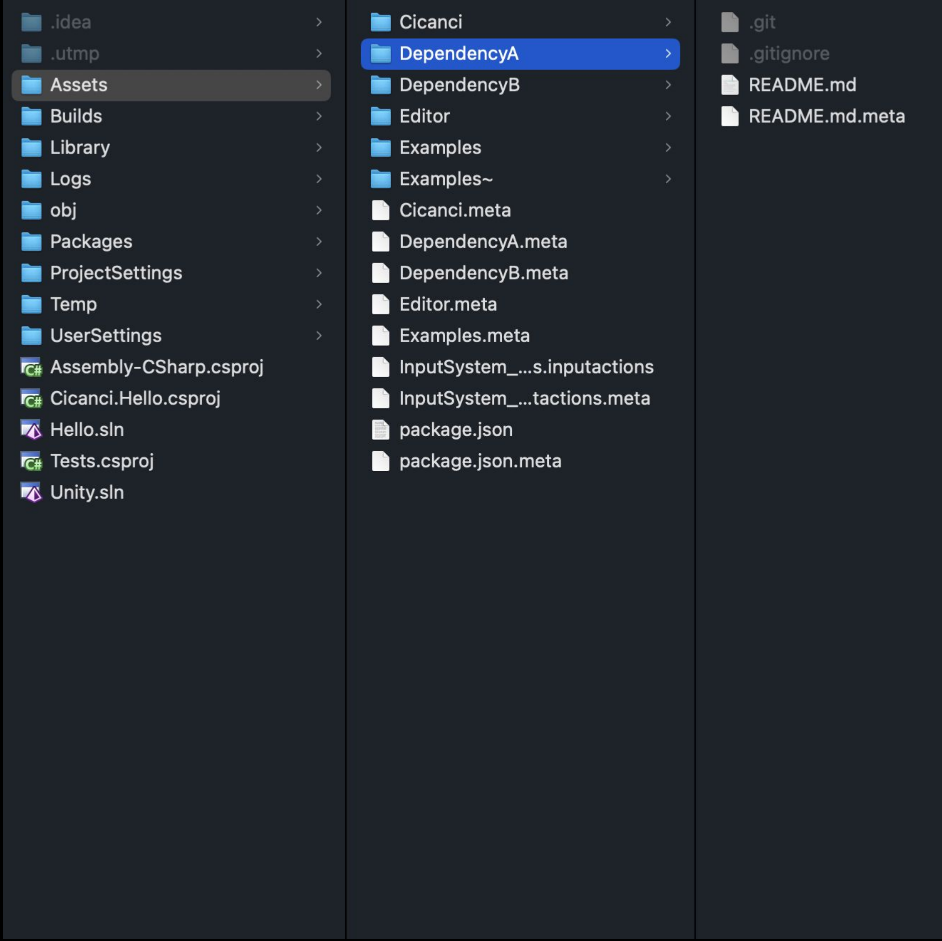
Hello Examples *77.89 KB* Import

Examples of how to use this SDK to say Hello.

Using git submodules

→ Using internal dependencies makes iteration easier and faster

→ You can test local changes without committing to the repository or publishing to NPM



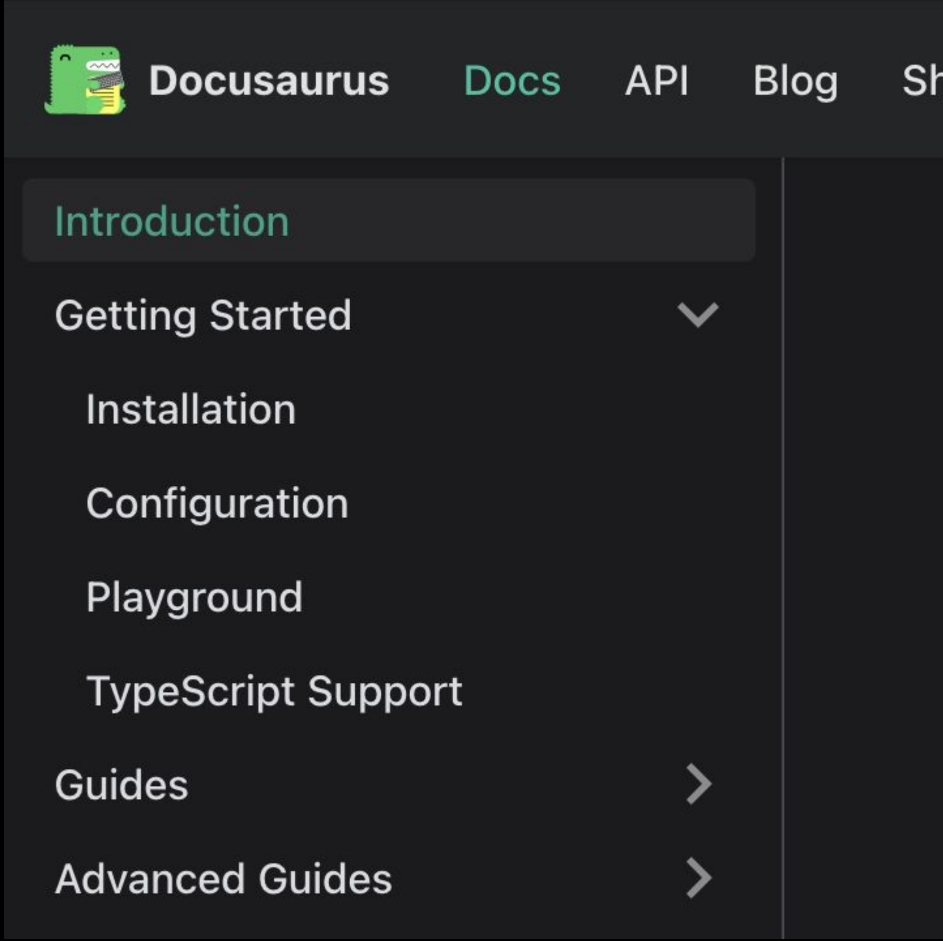
Prioritize support over new features

→ When developing and maintaining SDKs and plugins, support is the priority

→ A stable pipeline can ensure support is easy and fast to provide

Documentation

- → Important as the SDK itself
- → Keep it updated
- → Include a change log
- → A roadmap helps show what's coming up next



Potential issues

→ Make sure you can patch older versions and make a release

→ CI/CD requires time - don't release something half-baked

→ Avoid pushing multiple commits to trigger multiple pipelines

Best practices

- → CI/CD must be stable - fix issues as soon as they appear
- → Keep an eye out for improvements, especially build time
- → You can run multiple jobs on the same machine, but might not be faster
- → Make sure to test your project on all major Unity versions supported
- → You can also test each mobile library project

Wrap-up

- → Keep logic in the SDKs
- → Simple Unity plugins
- → Always ready to build and run
- → Automate everything you can
- → Support comes first

Thank you