Modding support

From Unity 2021.1, you can load additional Burst compiled libraries, which provide a way to create modifications that use Burst compiled code.

Burst only provides a method to load additional libraries, and doesn't provide any tooling to create mods. You need a copy of the Unity Editor to compile the additional libraries.

This section gives an example approach to modding with Burst and is a proof of concept.

Supported uses

You can use this function in Play mode (or Standalone Players) only.

Make sure you load the libraries as soon as possible, and before the first Burst-compiled use of a C# method. Unity unloads any Burst libraries that `BurstRuntime.LoadAdditionalLibraries` loads when you exit Play mode in the Editor or quit a Standalone Player.

Example modding system

NOTE
This example is limited in scope. You should have knowledge of assemblies and asmdefs to follow this example.

This example declares an interface that the mods abide by:

```
using UnityEngine;

public interface PluginModule
{
    void Startup(GameObject gameObject);
    void Update(GameObject gameObject);
}
```

You can use this interface to create new classes which follow these specifications and ship it separate to your application. Passing a single `GameObject` along limits the state that the plug-ins can affect.

Modding manager

The following is an example of a modding manager:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Reflection;
using UnityEngine;
using Unity.Burst;

public class PluginManager : MonoBehaviour
{
    public bool modsEnabled;
    public GameObject objectForPlugins;

    List<PluginModule> plugins;

    void Start()
    {
        plugins = new List<PluginModule>();

        // If mods are disabled, early out - this allows us to disable mods, enter Play Mode, exit Play Mode
        //and be sure that the managed assemblies have been unloaded (assuming DomainReload occurs)
        if (!modsEnabled)
            return;

        var folder = Path.GetFullPath(Path.Combine(Application.dataPath, "..", "Mods"));
        if (Directory.Exists(folder))
        {
            var mods = Directory.GetDirectories(folder);

            foreach (var mod in mods)
            {
                var modName = Path.GetFileName(mod);
                var monoAssembly = Path.Combine(mod, $"{modName}_managed.dll");

                if (File.Exists(monoAssembly))
                {
                    var managedPlugin = Assembly.LoadFile(monoAssembly);

                    var pluginModule = managedPlugin.GetType("MyPluginModule");

                    var plugin = Activator.CreateInstance(pluginModule) as PluginModule;

                    plugins.Add(plugin);
                }

                var burstedAssembly = Path.Combine(mod, $"{modName}_win_x86_64.dll");      // Burst dll (assuming windows 64bi
t)
                if (File.Exists(burstedAssembly))
                {
                    BurstRuntime.LoadAdditionalLibrary(burstedAssembly);
                }
            }
        }

        foreach (var plugin in plugins)
        {
            plugin.Startup(objectForPlugins);
        }
    }

    // Update is called once per frame
    void Update()
    {
        foreach (var plugin in plugins)
        {
            plugin.Update(objectForPlugins);
        }
    }
}
```

This code scans the `Mods` folder, and for each folder it finds within, it attempts to load both a managed dll and a Burst-compiled dll. It does this by adding them to an internal list that it can then iterate on and call the respective interface functions.

The names of the files are arbitrary: refer to Simple Create Mod Menu Button, which is the code that generated those files.

Because this code loads the managed assemblies into the current domain, you need a domain reload to unload those before you can overwrite them. Unity automatically unloads the Burst dll files automatically unloaded when you exit Play mode. This is why a Boolean to disable the modding system is included, for testing in the Editor.

A mod that uses Burst

Create a separate Unity project for this to use the project to produce the mod.

The following script attaches to a UI Canvas that contains a text component called **Main UI Label** and changes the text when the mod is used. The text is either **Plugin Updated : Bursted** or **Plugin Updated : Not Bursted**. You will see the **Plugin Updated : Bursted** by default, but if you comment out the lines that load the Burst library in the PluginManager above, then the Burst compiled code doesn't load and the message changes appropriately.

```
using Unity.Burst;
using Unity.Collections;
using Unity.Jobs;
using UnityEngine;
using UnityEngine.UI;

public class MyPluginModule : PluginModule
{
    Text textComponent;

    public void Startup(GameObject gameObject)
    {
        var childTextComponents = gameObject.GetComponentsInChildren<Text>();
        textComponent = null;
        foreach (var child in childTextComponents)
        {
            if (child.name == "Main UI Label")
            {
                textComponent = child;
            }
        }

        if (textComponent==null)
        {
            Debug.LogError("something went wrong and i couldn't find the UI component i wanted to modify");
        }
    }

    public void Update(GameObject gameObject)
    {
        if (textComponent != null)
        {
            var t = new CheckBurstedJob { flag = new NativeArray<int>(1, Allocator.TempJob, NativeArrayOptions.UninitializedMe
mory) };

            t.Run();

            if (t.flag[0] == 0)
                textComponent.text = "Plugin Updated : Not Bursted";
            else
                textComponent.text = "Plugin Updated : Bursted";

            t.flag.Dispose();
        }
    }

    [BurstCompile]
    struct CheckBurstedJob : IJob
    {
        public NativeArray<int> flag;

        [BurstDiscard]
        void CheckBurst()
        {
            flag[0] = 0;
        }

        public void Execute()
        {
            flag[0] = 1;
            CheckBurst();
        }
    }

}
```

Put the above script in a folder along with an assembly definition file with an assembly name of `TestMod_Managed` so the next script can locate the managed part.

Simple Create Mod Menu button

The following script adds a menu button. When you use the menu button, it builds a Standalone Player, then copies the C# managed dll and the `lib_burst_generated` .dll into a chosen `Mod` folder. This example assumes you are using Windows.

```
using UnityEditor;
using System.IO;
using UnityEngine;

public class ScriptBatch
{
    [MenuItem("Modding/Build X64 Mod (Example)")]
    public static void BuildGame()
    {
        string modName = "TestMod";

        string projectFolder = Path.Combine(Application.dataPath, "..");
        string buildFolder = Path.Combine(projectFolder, "PluginTemp");

        // Get filename.
        string path = EditorUtility.SaveFolderPanel("Choose Final Mod Location", "", "");

        FileUtil.DeleteFileOrDirectory(buildFolder);
        Directory.CreateDirectory(buildFolder);

        // Build player.
        var report = BuildPipeline.BuildPlayer(new[] { "Assets/Scenes/SampleScene.unity" }, Path.Combine(buildFolder, $"{modNa
me}.exe"), BuildTarget.StandaloneWindows64, BuildOptions.Development);

        if (report.summary.result == UnityEditor.Build.Reporting.BuildResult.Succeeded)
        {
            // Copy Managed library
            var managedDest = Path.Combine(path, $"{modName}_Managed.dll");
            var managedSrc = Path.Combine(buildFolder, $"{modName}_Data/Managed/{modName}_Managed.dll");
            FileUtil.DeleteFileOrDirectory(managedDest);
            if (!File.Exists(managedDest))  // Managed side not unloaded
                FileUtil.CopyFileOrDirectory(managedSrc, managedDest);
            else
                Debug.LogWarning($"Couldn't update managed dll, {managedDest} is it currently in use?");

            // Copy Burst library
            var burstedDest = Path.Combine(path, $"{modName}_win_x86_64.dll");
            var burstedSrc = Path.Combine(buildFolder, $"{modName}_Data/Plugins/x86_64/lib_burst_generated.dll");
            FileUtil.DeleteFileOrDirectory(burstedDest);
            if (!File.Exists(burstedDest))
                FileUtil.CopyFileOrDirectory(burstedSrc, burstedDest);
            else
                Debug.LogWarning($"Couldn't update bursted dll, {burstedDest} is it currently in use?");
        }
    }
}
```

Additional resources

- Import and configure plug-ins (https://docs.unity3d.com/Manual/PluginInspector.html)

Did you find this page useful? Please give it a rating:

Report a problem on this page