Fast/Live Script Reload

Tool will allow you to iterate quicker on your code. You simply go into play mode, make a change to any file and it'll be compiled on the fly and hot-reloaded in your running play-mode session.

Getting started

1. Import
2. Welcome screen will open - it contains all needed info to get started as well as support links and configuration. `You can always get back to this screen via 'Window -> Fast/Live Script Reload -> Start Screen'`
3. Go to Launch Demo -> Basic Example window
4. Follow instructions listed there

> During startup asset will check if Unity Auto Refresh is enabled and offer to adjust it in order to work properly. Depending on your workflow you want it in 'EnabledOutsidePlaymode' or 'Disabled' - this is to ensure Unity will not trigger script compilation on changes and will let Fast Script Reload to work.

> Sometimes Unity can be quite stubborn and try to auto-reload scripts even with Auto-Reload turned off, if you can still see standard 'Reloading script assemblies' progress bar on change please go to: 'Window -> Fast Script Reload -> Start Screen -> Reload -> Force prevent assembly reload during playmode'. This way tool will lock reload in code when you enter playmode.

```
Example scene 'Point' material should automatically detect URP or surface shader
If it shows pink, please adjust by picking shader manually:
1) URP: 'Shader Graphs/Point URP'
2) Surface: 'Graph/Point Surface'
```

On-Device Hot-Reload - Live Script reload

There's an addon to this tool - Live Script Reload - that'll allow you to use same functionality over the network in device build, eg:

- android (including VR headsets like Quest 2)
- standalone windows

Find more details here

Live Script Reload is using Fast Script Reload as a base asset - documentation is combined, if you don't use Live Script Reload you can skip any sections in this document prefixed with [Live-Reload]

How Unity Hot Reload functionality is made? Technical Approach Breakdown

If you'd like to know how tool works from technical point of view - have a look at blog series which gets deeper into the subject:

1) How to build Hot Reload For Unity

2) Hot Reload - on device, (eg Android / Windows Exe)

Reporting Compilation Errors

I've put lots of effort to test various code patterns in various codebases. Still - it's possible you'll find some instances where code would not compile, it's easiest to:

1. Look at compiler error and compare with generated source code, usually it'll be very obvious why issue is occuring
2. Refactor problematic part (look at limitations section as they'll explain how)
3. Let me know via support email and I'll get it fixed

Executing custom code on hot reload

Custom code can be executed on hot reload by adding a method to changed script.

You can see example by adjusting code in 'Graph.cs' file.

```
void OnScriptHotReload()
{
    //do whatever you want to do with access to instance via 'this'
}
```

```
static void OnScriptHotReloadNoInstance()
{
   //do whatever you want to do without instance
   //useful if you've added brand new type
   // or want to simply execute some code without |any instance created.
   //Like reload scene, call test function etc
}
```

EXPERIMENTAL Adding New Fields

Asset has an experimental support for adding new fields at runtime which will also render in Editor and allow you to tweak values - same as with normal fields.

To enable, please: `Window -> Fast Script Reload -> Start Window -> New Fields -> Enable experimental added field support`.

> As this is an experimental feature please expect it to break more often! It'd be great help if you could report any issues via Discord / email.

New Fields - specific limitations

- outside classes can not call new fields added at runtime
- new fields will only show in editor if they were already used at least once

    - eg if you've added a variable into a method, on first call that variable will be initialized and will start showing in editor

New Fields - performance

For new fields to work your code will be adjusted. Instead of calling fields directly your code will call into a method that retrieves value from dynamic dictionary.

Due to that there'll be some overhead with:

- looking up proper object and new-field value in dictionary
- initializing values
- use of dynamic type, which will introduce some additional casts

> All that shouldn't really add too much overhead on dev-machine - you may lose few FPS though.

User Defined Script Overrides

For asset to hot reload your changes it needs to make some adjustments to code. This could cause some issues as described in limitations section.

I continuously work on mitigating limitations. When they happen you can help yourself quickly by creating user defined script override.

Those are simply overrides on a per-file / method basis - when specified their contents will be used for final source code. Allowing you to simply fix any issues. This is especially helpful with larger files.

Replacing methods

You can replace existing methods by specifying their full signature in correct type - contents will be then replaced.

> If no match for the method signature is found it'll be ignored

For example, let's look at limitation with assigning Singleton to Instance using 'this' keyword

> That limitation is already solved although it'll serve as a clean example illustrating the feature

Following code:

```
public class MySingleton: MonoBehaviour {
    public static MySingleton Instance;

    void Start() {
        Instance = this;
    }

    void SomeOtherMethod() {
        //some other logic
    }
}
```

Would be rewritten to:

```
public class MySingleton__Patched_: MonoBehaviour { //FSR: name change to include
__Patched_ postfix to prevent name clashes
    public static MySingleton Instance;

    void Start() {
        //assigning 'this' (now of type MySingleton__Patched_) to
        //'Instance' of type MySingleton will now show compilation error due to
type mismach
        Instance = this;
    }

    void SomeOtherMethod() {
        //some other logic
    }
}
```

In that case you can create a user script rewrite override, to do so:

1. In Unity project panel right-click on 'MySingleton.cs'
2. Select `Fast Script Reload -> Add \ Open User Script Rewrite Override`
3. Asset will create override file for you with some template already in (there's also description in top comment how to use)

Override:

```
public class MySingleton__Patched_: MonoBehaviour {
    void Start() { //override will target class specified above and only defined
methods, in that case Start()
        Instance = (MySingleton)(object)this;   //contents will be changed as they
are visible here, casting to object and then to expected type will correct the
issue
    }
}
```

Adding types

When your code is recompiled it lands in new assembly. This can cause some issues, for example if your class is using `internal` interface - after recompile it won't be able to access that interface.

Following code will fail to compile when changing MyClass as it won't be able to access IInterface which is `internal`:

```
//Defined in file MyClass.cs
public class MyClass: IInterface {

}
```

```
//Defined in other file, IInterface.cs
interface IInterface { //no access modifier for interfaces will infer 'internal' -
only available to assembly it's defined in.

}
```

In that case you can create a user script rewrite override, to do so:

1. In Unity project panel right-click on 'MyClass.cs'
2. Select `Fast Script Reload -> Add \ Open User Script Rewrite Override`
3. Asset will create override file for you with some template already in (there's also description in top comment how to use)

And put following in the override file

```
interface IInterface {
    // add any definitions as needed
}
```

This will then be simply added to newly compiled file and interface will be accessible.

```
//now both in same assembly/file when hot reloaded
public class MyClass: IInterface {

}


interface IInterface {

}
```

Managing existing User Script Rewrite Overrides

You can see all your overrides by:

1. Right clicking on file in Unity project panel
2. Clicking `Fast Script Reload -> Show User Script Rewrite Overrides`

This will take you to tool settings window. You can also access it via:

1. Clicking `Window -> Fast Script Reload -> Start Screen`
2. Selecting `User Script Rewrite Overrides` side tab

EXPERIMENTAL In-editor Hot Reload (outside of playmode)

Asset has an experimental support for hot reload outside of playmode. However with limitations as they are now (eg not being able to add new methods) - it is intended for some specific use cases (like iterating on editor scripts).

> **Feature is not intended as a replacement for Unity compile / reload mechanism**

To enable, please: `Window -> Fast Script Reload -> Start Window -> Editor Hot-Reload -> Enable Hot-Reload outside of play mode`.

> As this is an experimental feature please expect it to break more often! It'd be great help if you could report any issues via Discord / email.

Debugging

Debugging is fully supported although breakpoints in your original file won't be hit.

Once change is compiled, you'll get an option to open generated file [via clickable link in console window] in which you can set breakpoints.

Tool can also auto-open generated files for you on change for simpler access, you can find option via 'Window -> Fast Script Reload -> Start Screen -> Debugging -> Auto open generated source file for debugging'

> Debugging with Rider for Unity 2019 and 2020 is having some issues, you can only open debuggable files with auto-open feature. Clicking a file in console causes subtle static-variables reload (not full domain reload) that'll break your play-session.

Adding Function Breakpoint

If for whatever reason debugger breakpoint is not hit you can try setting Function Breakpoint in your IDE.

For type name you want to include `<OriginalTypeName>__Patched_`, the `__Patched_` postfix is auto-added by asset to prevent name clash. Function name remains unchanged.

Production Build

For Fast Script Reload asset code will be excluded from any builds.

For Live Script Reload you should exclude it from final production build, do that via:

- 'Window -> Fast Script Reload -> Welcome Screen -> Build -> Enable Hot Reload For Build' - untick

When building via File -> Build Settings - you'll also see Live Script Reload status under 'Build' button. You can click 'Adjust' button which will take you to build page for asset. `This is designed to make sure you don't accidentally build tool into release althoguh best approach would be to ensure your release process takes care of that.`

Options

You can access Welcome Screen / Options via 'Window -> Fast/Live Script Reload -> Start Screen' - it contains useful information as well as options.

```
Context menus will be prefixed with used version, either 'Fast Script Reload' or
'Live Script Reload'
```

Auto Hot-Reload

By default tool will pick changes made to any file in playmode. You can add exclusions to that behaviour, more on that later.

You can also manually manage reload, to do so:

1. Un-tick 'Enable auto Hot-Reload for changed files' in Options -> Reload page
2. Click Window -> Fast Script Reload -> Force Reload to trigger
3. or call `FastScriptReloadManager.TriggerReloadForChangedFiles()` method from code

> You can also use Editor -> Hotkeys to bind Force Reload to specific key.

[Live-Reload] Hot-Reload over Network

With on-device build, your code changes will be distributed over the network in real-time.

By default running application will send a broadcast and try to discover editor running the tool.

Broadcast is initiated from device where build is running on (not from editor) - this means device running editor needs to allow the connection.

[Live-Reload] Troubleshooting network issues

If for whatever reason reload over network doesn't work, please:

1. go to 'Window -> Live Script Reload -> Options/Network'
2. make sure port used is not already used by any other application
3. make sure your Firewall allows connections on that port
4. If you think broadcast doesn't work in your network it's best to specify IP Address explicitly (tick 'Force specific UP address for clients to receive Hot-Reload updates' and add IP)
    - this will allow client (build on device) connect directly to specified address

[Live-Reload] Connected Client

In playmode, message will be logged when clients connects. Also Options/Network will display connected client, eg Android phone could be identified as:

```
SM-100232 connected from 192.189.168.68:12548
```

Only 1 client can be connected at any time.

[Live-Reload] Testing with Editor

By default, editor will reflect any changes you made without using network. If you want to force editor to behave as networked client:

1. Press play

2. Find DontDestroyOnLoadObject 'NetworkedAssemblyChangesLoader' -
3. tick 'IsDebug'
4. tick 'Editor Acts as Remote Client'
5. enable NetworkedAssemblyChangesLoader component

Managing file exclusions

Files can be excluded from auto-compilation.

via 'Project' context menu

1. Right click on any *.cs file
2. Click Fast Script Reload
3. Add Hot-Reload Exclusion

You can remove exclusion from same menu

via Exclusions page

To view all exclusions:

1. Right click on any *.cs file
2. Click Fast Script Reload
3. Click Show Exclusions

via class attribute

You can also add `[PreventHotReload]` attribute to a class to prevent hot reload for that class.

Batch script changes and reload every N seconds

Script will batch all your playmode changes and Hot-Reload them in bulk every 3 seconds - you can change duration from 'Reload' options page.

Disable added/removed fields check

By default if you add / remove fields, tool will not redirect method calls for recompiled class. This is to ensure there are no issues as that is generally not supported.

Some assets however will use IL weaving to adjust your classes (eg Mirror) as a post compile step. In that case it's quite likely hot-reload will still work.

Managing reference exclusions

Asset will reference all .dll files that original code is referencing. In some cases that causes compilation error (eg 'Type XYZ is defined in both assembly a.dll and b.dll). You can use those options to exclude specific references from being added.

> 'Start Screen -> Exclude References (Advanced) -> adjust as needed'.

Performance

Your app performance won't be affected in any meaningful way. Biggest bit is additional memory used for your re-compiled code. Won't be visible unless you make 100s of changes in same play-session.

File Watchers Performance Overhead

In some cases watching for file changes is causing significant performance overhead. This is down to the Unity FileWatcher which I'm unable to change or provide suitable replacement for. If you're experiencing this issue please go to `Window -> Fast Script Reload -> File Watcher (Advanced Setup)` and narrow down watchers to specific path where you're working in. You can watch multiple folders in this manner.

LIMITATIONS -please make sure to read those

There are some limitation due to the approach taken to Hot-Reload your scripts. I've tried to minimise the impact to standard dev-workflow as much as possible.

In some cases however you may need to use workarounds as described below.

> In most cases you'll be able to use User Defined Script Overrides to overcome limitations and make hot reload code compilable.

Generic methods and classes won't be Hot-Reloaded

Unfortunately generics will not be Hot-Reloaded, to workaround you'd need to move code to non-generic class / method.

Tool will try to change non-generic methods in those files and will simply skip generic ones.

Note - you can still Hot-Reload for class implementations that derive from generic base class but are not generic themselves, eg.

```
class SingletonImplementation: SingletonBase<SomeConcreteType> {
   public void SomeSpecificFunctionality() {
      //you can change code here and it'll be Hot-Reloaded as type itself is not
generic
   }

   public void GenericMethod<T>(T arg) {
      //changes here won't be Hot-Reloaded as method is generic
   }
}

class SingletonBase<T> where T: new() {
   public T Instance;

   public void Init() {
      Instance = new T(); //if you change this code it won't be Hot-Reloaded as
it's in generic type
   }
}
```

Adding new fields

Experimental support with 1.3, minor limitations remaining:

- outside classes can not call new fields added at runtime
- new fields will only show in editor if they were already used at least once

> You need to opt in via start screen -> 'Window -> Fast Script Reload -> Start Screen -> New Fields -> enable'!

Passing `this` reference to method that expect concrete class implementation

`By default experimental setting 'Enable method calls with 'this' as argument fix' is turned on in options, and should fix 'this' calls/assignment issue. If you see issues with that please turn setting off and get in touch via support email.

Unless experimental setting is on - it'll throw compilation error `The best overloaded method match for xxx has some invalid arguments` - this is due to the fact that changed code is technically different type. The code will need to be adjusted to depend on some abstraction instead (before hot-reload).

This code would cause the above error.

```
public class EnemyController: MonoBehaviour {
    EnemyManager m_EnemyManager;

    void Start()
    {
        //calling 'this' causes issues as after hot-reload the type of
EnemyController will change to 'EnemyController__Patched_'
        m_EnemyManager.RegisterEnemy(this);
    }
}

public class EnemyManager : MonoBehaviour {
    public void RegisterEnemy(EnemyController enemy) { //RegisterEnemy method
expects parameter of concrete type (EnemyController)
        //impementation
    }
}
```

It could be changed to support Hot-Reload in following way:

1. Don't depend on concrete implementations, instead use interfaces/abstraction

```
public class EnemyController: MonoBehaviour, IRegistrableEnemy {
    EnemyManager m_EnemyManager;

    void Start()
    {
```

```
        //calling this causes issues as after hot-reload the type of
EnemyController will change
        m_EnemyManager.RegisterEnemy(this);
    }
}

public class EnemyManager : MonoBehaviour {
    public void RegisterEnemy(IRegistrableEnemy enemy) { //Using interface will go
around error
        //impementation
    }
}

public interface IRegistrableEnemy
{
    //implementation
}
```

2. Adjust method param to have common base class

```
public class EnemyManager : MonoBehaviour {
    public void RegisterEnemy(MonoBehaviour enemy) { //Using common MonoBehaviour
will go around error
        //impementation
    }
}
```

Assigning `this` to a field references

Similar as above, this could cause some trouble although 'Enable method calls with 'this' as argument fix' setting will fix most of the issues.

Especially visible with singletons. eg.

```
public class MySingleton: MonoBehaviour {
    public static MySingleton Instance;

    void Start() {
        Instance = this;
    }
}
```

Calling internal class members from changed code

> You can use User Defined Script Overrides to overcome this limitation

Technically, once your changed code is compiled it'll be in a separate assembly. As a result this changed code won't be able to access internal classes from assembly it originated from.

Extensive use of nested classed / structs

> You can use User Defined Script Overrides to overcome this limitation

If your code-base contains lots of nested classes - you may see more compilation errors.

Easy workaround is to move those nested classes away so they are top-level.

Creating new public methods

Hot-reload for new methods will only work with private methods (only called by changed code).

Adding new references

When you're trying to reference new code in play-mode session that'll fail if assembly is not yet referencing that (most often happens when using AsmDefs that are not yet referencing each other)

Changing class that uses extension and passes itself as a reference

> You can use User Defined Script Overrides to overcome this limitation

Changing class that uses extension method and passes itself as a reference will create compiler error.

Generally that shouldn't be an issue, extension methods are primarily used as a syntatic sugar to extend a class that you do not have access to. You shouldn't need to create extension methods for types you own (instead those are generally instance methods or base class methods).

Given example:

```
public class ExtensionMethodTest
{
   public string Name;

   void Update()
   {
      this.PrintName();
   }
}

//separate extension file
public static ExtensionMethodTestExtensions
{
   public static void PrintName(this ExtensionMethodTest obj)
   {
      Debug.Log(obj.Name);
   }
}
```

When changing `ExtensionMethodTest` you'll get compile error. Workaround would be to include method call in your type, eg:

```
public class ExtensionMethodTest {
   public string Name;

   void Update() {
      this.PrintName();
   }

   private void PrintName() {
      Debug.Log(Name);
   }
}
```

Arguably that's what should be done in the first place.

Adjusting classes that use extension methods without passing itself as a reference - will work correctly. eg:

```
public class ObjectFromExternalAssembly()
{
   //included just to illustrate example, that'd be in compiled assembly
   //that you can't change and use extension method approach

   public string Name;
}

public class ExtensionMethodTester
{
   void Update()
   {
      var t = new ExtensionMethodTest();
      t.PrintName()
   }
}

//separate extension file
public static ObjectFromExternalAssemblyExtensions
{
   public static void PrintName(this ObjectFromExternalAssembly obj)
   {
      Debug.Log(obj.Name);
   }
}
```

Changing class that implements internal interface can trigger compilation error

> You can use User Defined Script Overrides to overcome this limitation

If class is implementing interface that's defined in different file as internal (default for no access modifier) - then changes to that class will fail to compile.

eg.

```
//file IInterface.cs
interface IInterface { //default interface access-modifier is 'internal'
   //declaration
}

//file ClassImplementingIInterface.cs
class ClassImplementingIInterface: IInterface {
   //changing this class will cause CS0122 'IInterface' is inaccessible due to
it's protection level
}
```

> Quick workaround is to declare that interface as public

Changing class that accesses `private protected` members

> You can use User Defined Script Overrides to overcome this limitation

With C# 7.2 `private protected` access modifier was introduced. It works as `protected` access modifier in a sense that inherited classes can access it but. Addition of `private` also limits it to same assembly. Your changes are technically compiled into separate assembly and at the moment trying to access `private protected` in changed code will produce compiler error.

> Easiest workaround for now is to declare those `private protected` members as `protected`.

Limited debugger support for Rider when using Unity 2019 and 2020

Once breakpoint has been hit it'll stop asset from hot-reloading in that play-session. Newer Unity versions are supporting debugging.

No IL2CPP support

Asset runs based on specific .NET functionality, IL2CPP builds will not be supported. Although as this is development workflow aid you can build your APK with Mono backend (android) and change later.

Partial classes

Partial classes are not yet supported

FAQ

How is this asset different than Live Script Reload?

Fast Script Reload

- hot reload in editor

Live Script Reload

- same hot reload in editor as FSR (includes FSR)

- **AND hot reload directly in builds / on-device**
- you don't have to download them both separately, LSR is also 1 package import
- standalone extension priced at $35, or if you've already bought Fast Script Reload it's $5 upgrade

Editor makes full reload on any change in playmode

Unity Editor has an option to auto recompile changes. **For tool to work properly you want to have that either disabled or enabled only outside of playmode.**

You can adjusted auto-reload at any time via `Edit -> Preferences -> Asset Pipeline -> Auto Refresh`.

Tool will also offer to disable auto-refresh on startup.

It's possible to set auto-refresh to enabled but only outside of playmode. Depending on editor version used this can be found in:

- `Edit -> Preferences -> General -> Script Changes While Playing -> Recompile After Finished Playing`
- or `Edit -> Preferences -> Asset Pipeline -> Auto Refresh -> Enabled Outside Playmode`

VS Code shows console shows errors

VS Code proj file (csproj) generate with NetFramework version 4.7.1. One of the plugin DLLs is targeting version 4.8. Even though VS Code does not compile that code (Unity does) - it'll still show error as it didn't load the library for code-completion.

It's not a plugin issue as such, other dlls will have same troubles. Best solution I've found is to force csproj files to be generated with 4.8 version instead. This can be achieved with following editor script

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

public class VisualStudioProjectGenerationPostProcess : AssetPostprocessor
{
    private static void OnGeneratedCSProjectFiles()
        {
            Debug.Log("OnGeneratedCSProjectFiles");
            var dir = Directory.GetCurrentDirectory();
            var files = Directory.GetFiles(dir, "*.csproj");
            foreach (var file in files)
                ChangeTargetFrameworkInfProjectFiles(file);
        }

    static void ChangeTargetFrameworkInfProjectFiles(string file)
    {
        var text = File.ReadAllText(file);
        var find = "TargetFrameworkVersion>v4.7.1</TargetFrameworkVersion";
```

```
            var replace = "TargetFrameworkVersion>v4.8</TargetFrameworkVersion";

            if (text.IndexOf(find) != -1)
            {
                text = Regex.Replace(text, find, replace);
                File.WriteAllText(file, text);
            }
        }

    }
```

> As a one-off you may also need to go to Edit -> Preferences -> External Tools -> click 'Regenerate project files' buttons

When importing I'm getting error: 'Unable to update following assemblies: (...)/ImmersiveVRTools.Common.Runtime.dll'

This happens occasionally, especially on upgrade between versions. It's harmless error that'll go away on play mode.

When upgrading between versions, eg 1.1 to 1.2 example scene cubes are pink

This is down to reimporting 'Point' prefab. Right now plugin will make sure it's using correct shader eg. URP / Built-in but only on initial import.

To fix please go to `FastScripReload\Examples\Point\Point.prefab` and search for 'Point' shader.

Roadmap

- ~~add Mac/Linux support~~ (added with 1.1)
- ~~add debugger support for hot-reloaded scripts~~ (added with 1.2)
- ~~allow to add new fields (adjustable in Editor)~~ (added with 1.3)
- better compiler support to work around limitations