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

```
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]

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
}
```

Options

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

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

```
Options can aslo be accessed via 'Edit -> Preferences -> Fast/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.

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.`

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.

Breakpoints in hot-reloaded scripts won't be hit, sorry!

- only for the scripts you changed, others will work

- with how quick it compiles and reloads you may not even need a debugger

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
   }
}
```

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;
    }
}
```

Extensive use of nested classed / structs

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 fields

Adding new fields is not supported in play mode. You can however simply create local variable and later quickly refactor that out.

eg. for a simple class that moves position by some vector on every update

Initial class before play mode entered

```
public class SimpleTransformMover: MonoBehaviour {
   void Update() {
        transform.position += new Vector3(1, 0, 0);
    }
}
```

Changes in playmode

```
public class SimpleTransformMover: MonoBehaviour {
   //public Vector3 _moveBy = new Vector3(1, 0, 0); //1) do not introduce fields
in play mode

   void Update() {
```

```
        var _moveBy = new Vector3(1, 0, 0); //2) instead declare variable in
method scope
        // (optionally with instance scope name-convention)

        // transform.position += new Vector3(1, 0, 0); //original code - now will
use variable
        transform.position += _moveBy; //3) changed code - uses local variable

        4) iterate as needed and after play mode simply refactor added variables
as fields
    }
}
```

Tool will show error if you try to add/remove fields and won't perform Hot-Reload.

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

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);
   }
}
```

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.

FAQ

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`

Roadmap

- ~~add Mac/Linux support~~ (added with 1.1)
- add debugger support for hot-reloaded scripts
- better compiler support to work around limitations