

# FlexCompiler

---

*For the best experience, you may check the latest online version of the doc here:*

<http://compiler.flexflew.com>

FlexCompiler is a Unity3D editor extension intended as a handy tool for both asset publishers and game developers.

It's a simple GUI wrapper for csharp compilers(mcs, gmcs, smcs, csc, roslyn, etc.) to turn source codes to assemblies(dlls) right in Unity editor.

It also features a basic obfuscation functionality(currently experimental).

[VIEW ON YOUTUBE >](#)

## Introduction

---

### What for?

- For publishers: to close source
- For developers: to reduce project re-compiling time

### Features

- Simple workflow: drag - drop - compile
- Xml Documentation generating
- Assembly information configuring
- Compilers switching
- Preprocessor directives parsing
- Build arguments customizing
- Compiler tasks importing/exporting
- Chained/dependency tasks support
- Obfuscation support

### Compilers

The following compilers are supported by default:

- [Mono family](#)(recommended): mcs, gmcs, smcs
- [.Net Framework](#)(Windows only): csc(.Net35 targeted)

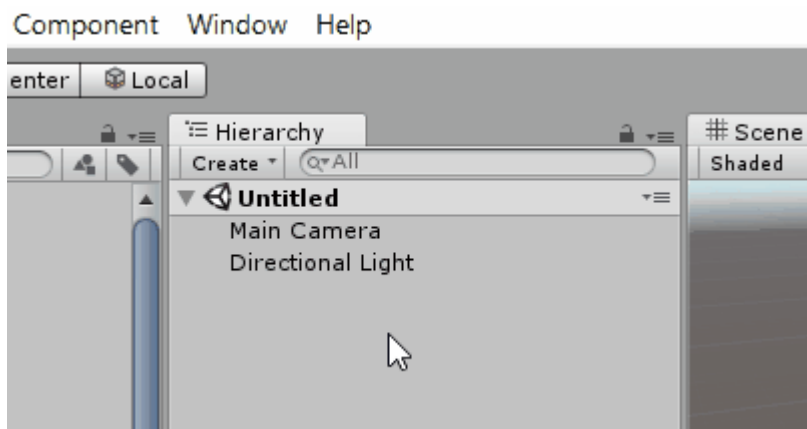
## Getting Started

---

### Import And Launch FlexCompiler

Import FlexCompiler into your project.

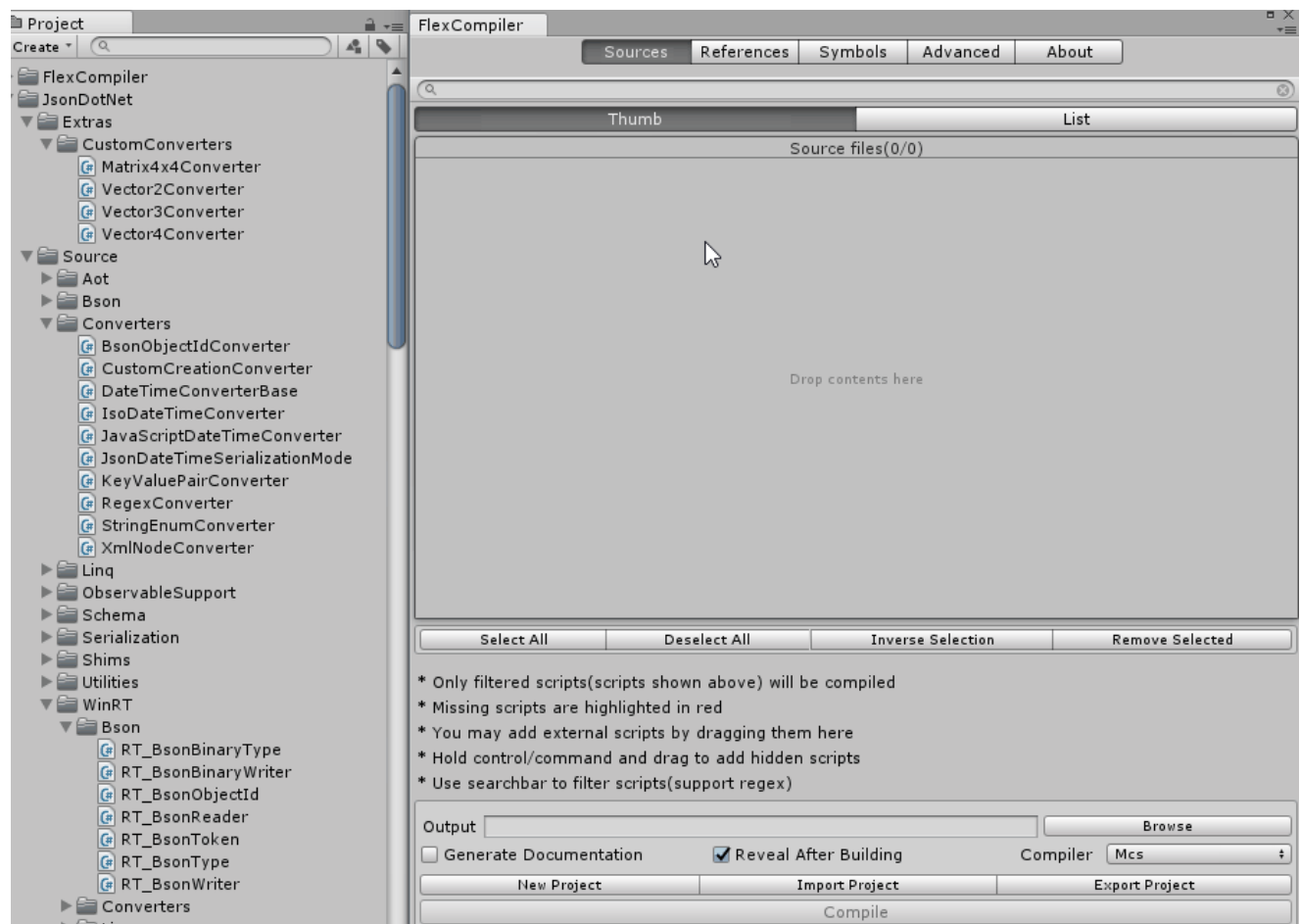
Open FlexCompiler Editor window at `Window/FlexCompiler` or by pressing `F9`.



## Add Source Scripts

Make sure you're on **Sources** tab.

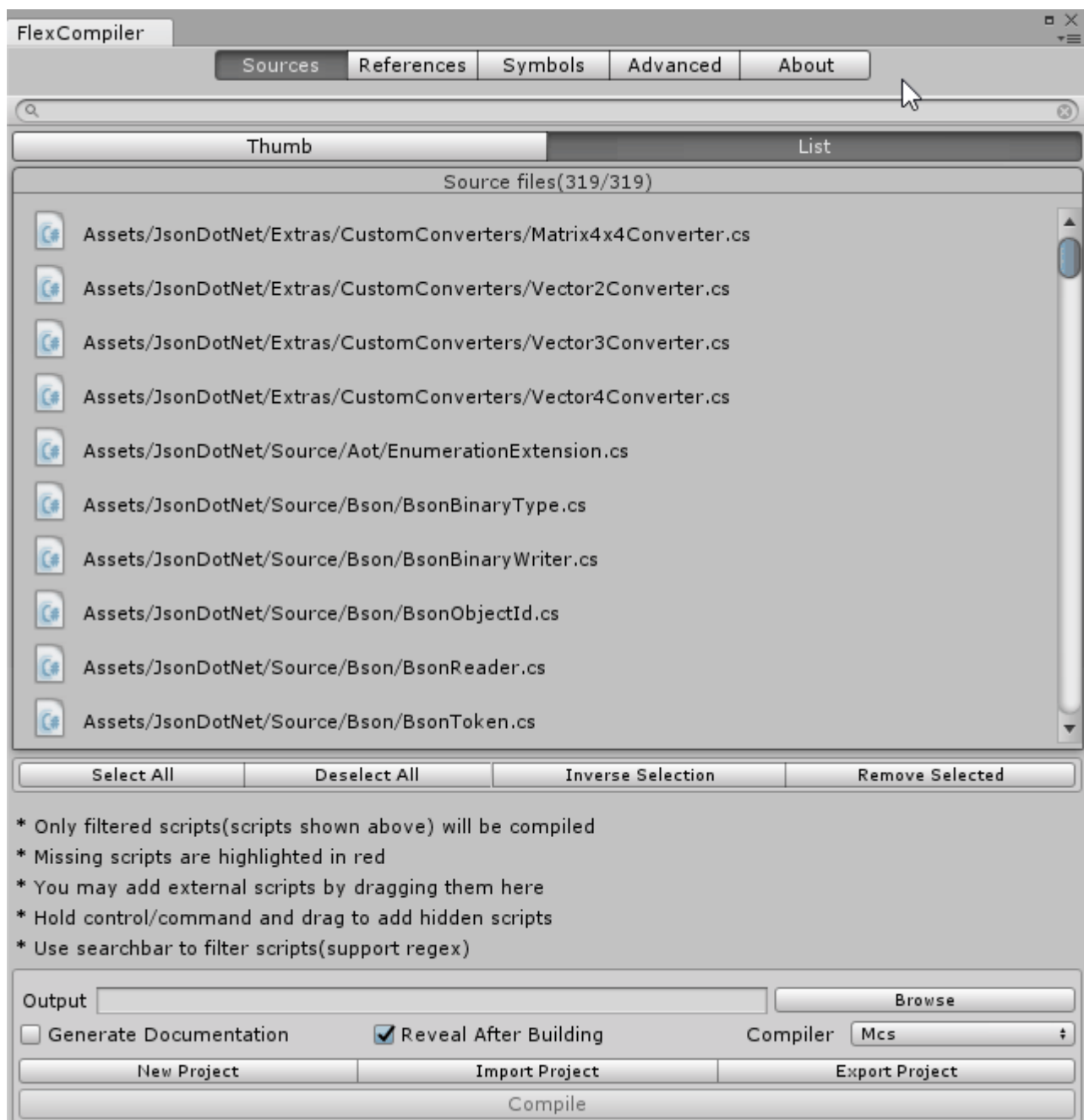
Drag and drop sources(folders or scripts) right to source panel.



## Refine Results(Optional)

Use filter to remove unwanted sources(don't forget to reset filter after that).

**Important:** Only sources visible in the source panel will be compiled. So that you can also use filter to keep wanted sources instead of removing unwanted.



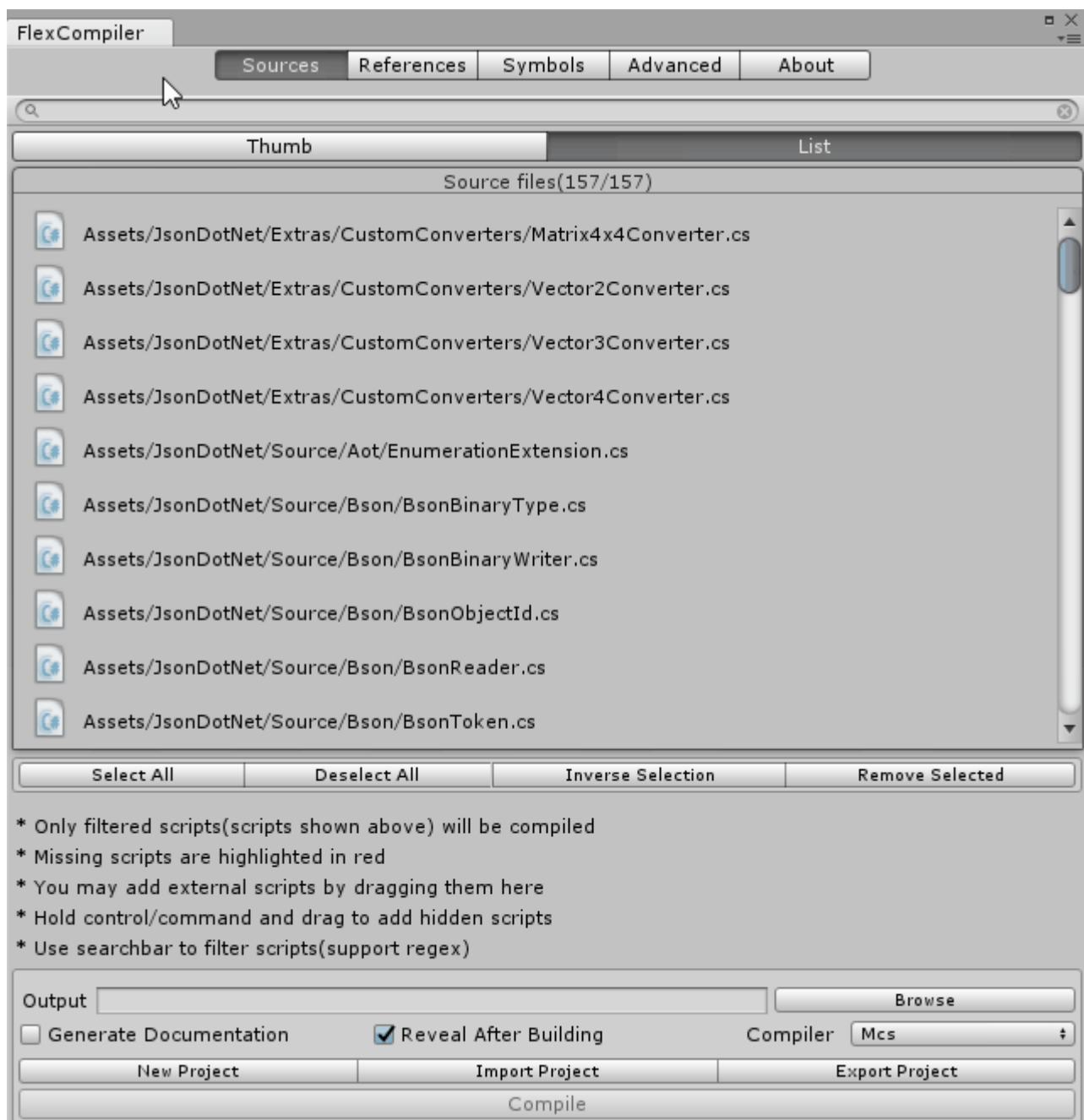
## Set References

Switch to **References** tab.

Possible references are automatically listed here.

Check required items. If you are not clear about what references are required, you may just check all.

You can also drag and drop external assemblies(dlls) here as references.



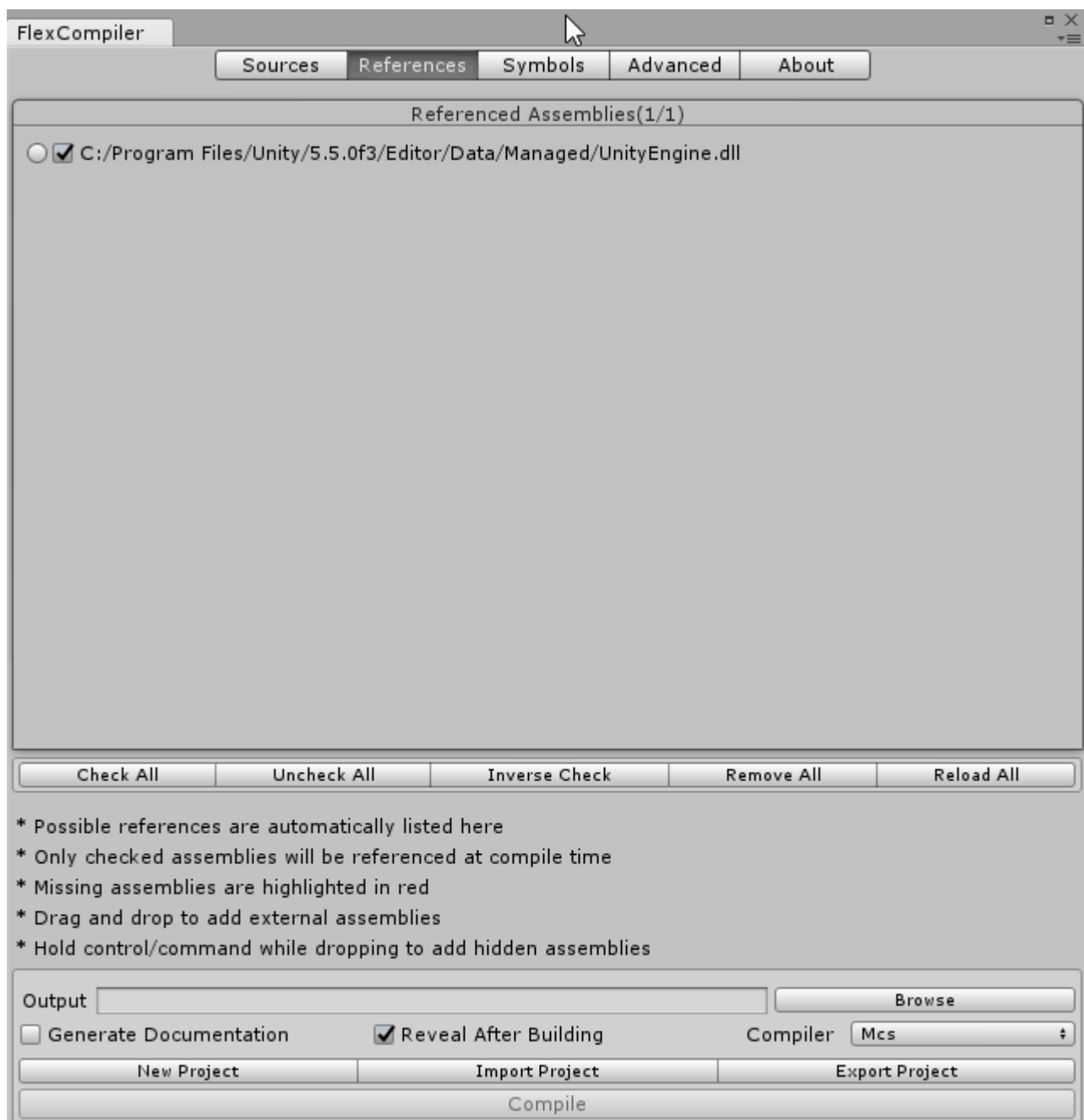
## Set [Preprocessor Directives](#)(Optional)

Use linebreak or semicolon( ; ) to separate directives.

Use # to comment out a line.

**Important:** If you have platform directives e.g. UNITY\_ANDROID , UNITY\_IOS in source scripts, you should make sure the right platform directive is listed in the symbols panel.









If not seeing the right one, you need to added it manually, or switch platform and reload symbols by clicking Default button on symbols panel.

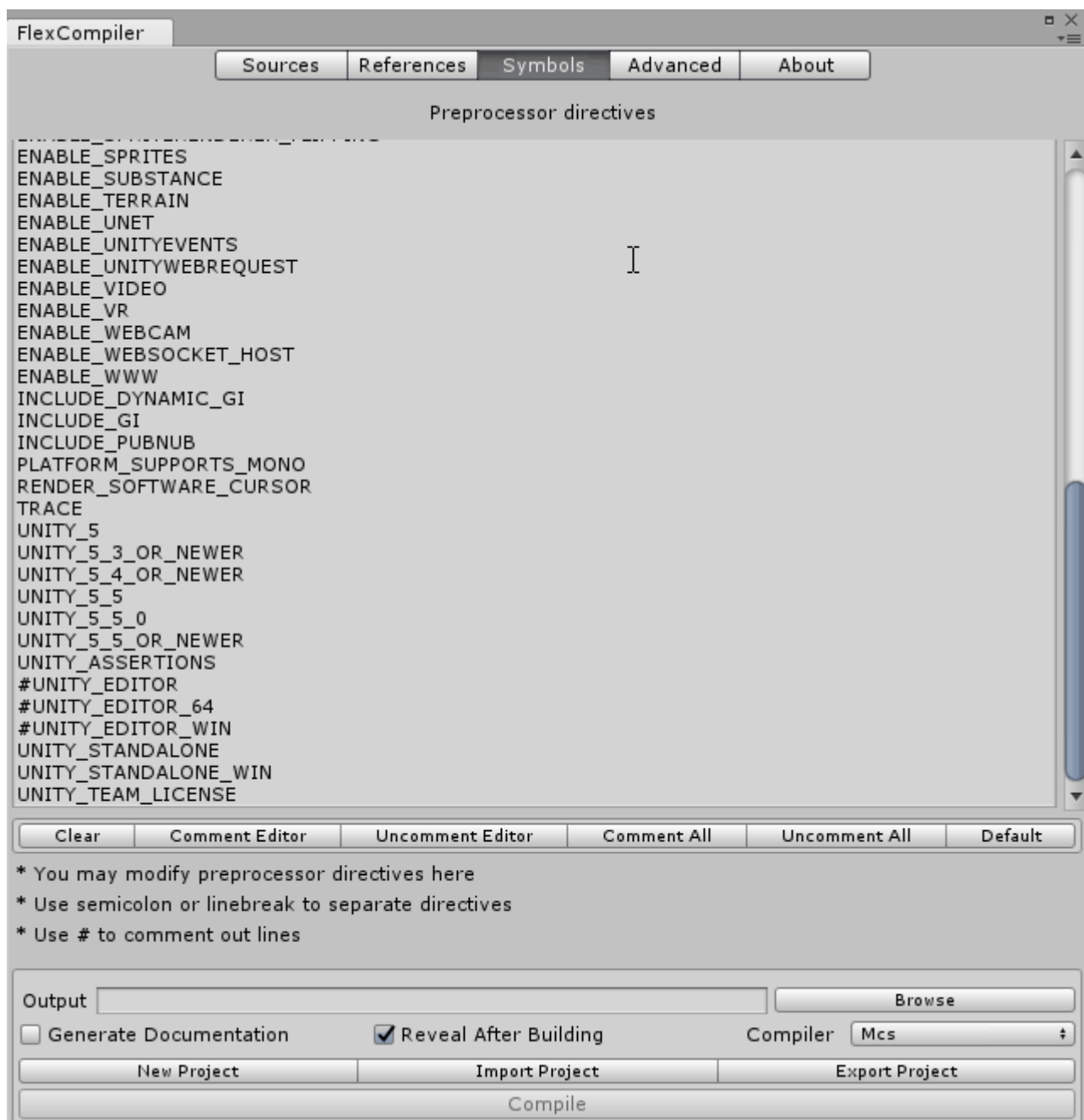


See [Unity Platform Dependent Compilation](#)

## Set Assembly Information(*Optional*)

Switch to **Advanced** tab. Toggle on **Assembly Info** panel, check **Enabled** , set required fields.  
Right click on an assembly to check its information

Name	Date modified	Type
 Assets	1/8/2017 01:46	File folder
 Library	1/8/2017 13:04	File folder
 ProjectSettings	1/7/2017 03:05	File folder
 Temp	1/8/2017 13:04	File folder
 .flexcompiler.fpj	1/8/2017 13:01	FPJ File
 New Unity Project.fpj	1/8/2017 11:43	FPJ File
 sample.dll	1/7/2017 05:11	Application extensic
 sample.xml	1/7/2017 05:11	URL:vscode

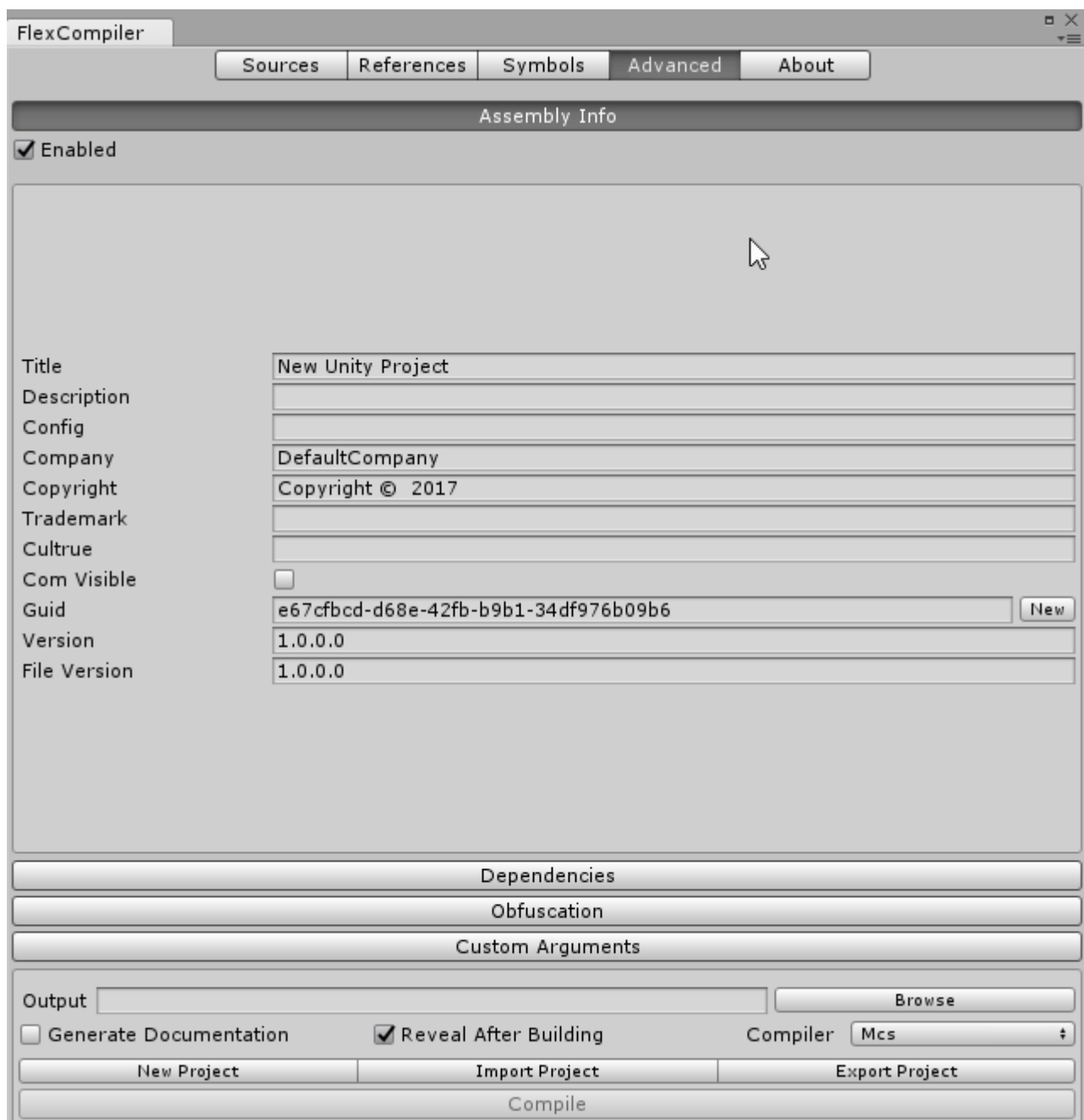


## Enable Obfuscation(Optional)

Switch to **Advanced** tab.

Toggle on **Obfuscation** panel, check **Enabled**.

**Obfuscation is currently experimental! Use at your own risk.**



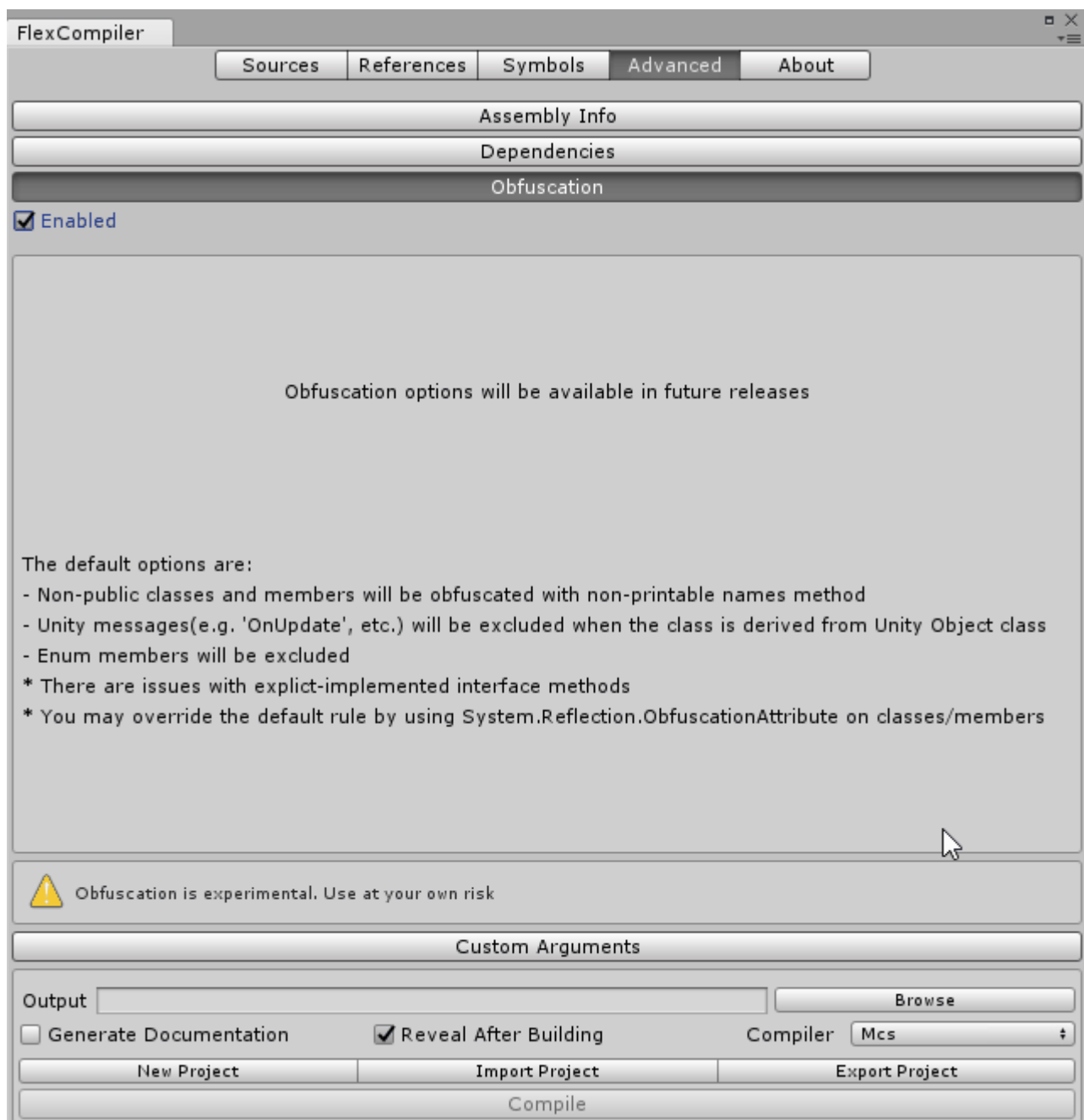
## Set Output

Click **Browse** on the bottom panel and set output path.

You may check **Generate Documentation** to enable [XML Documentation](#) generating.

*XML Documentation provides IDE intellisense suggestions. You can also generate html api docs from it with thirdparty tools like [Sandcastle](#).*





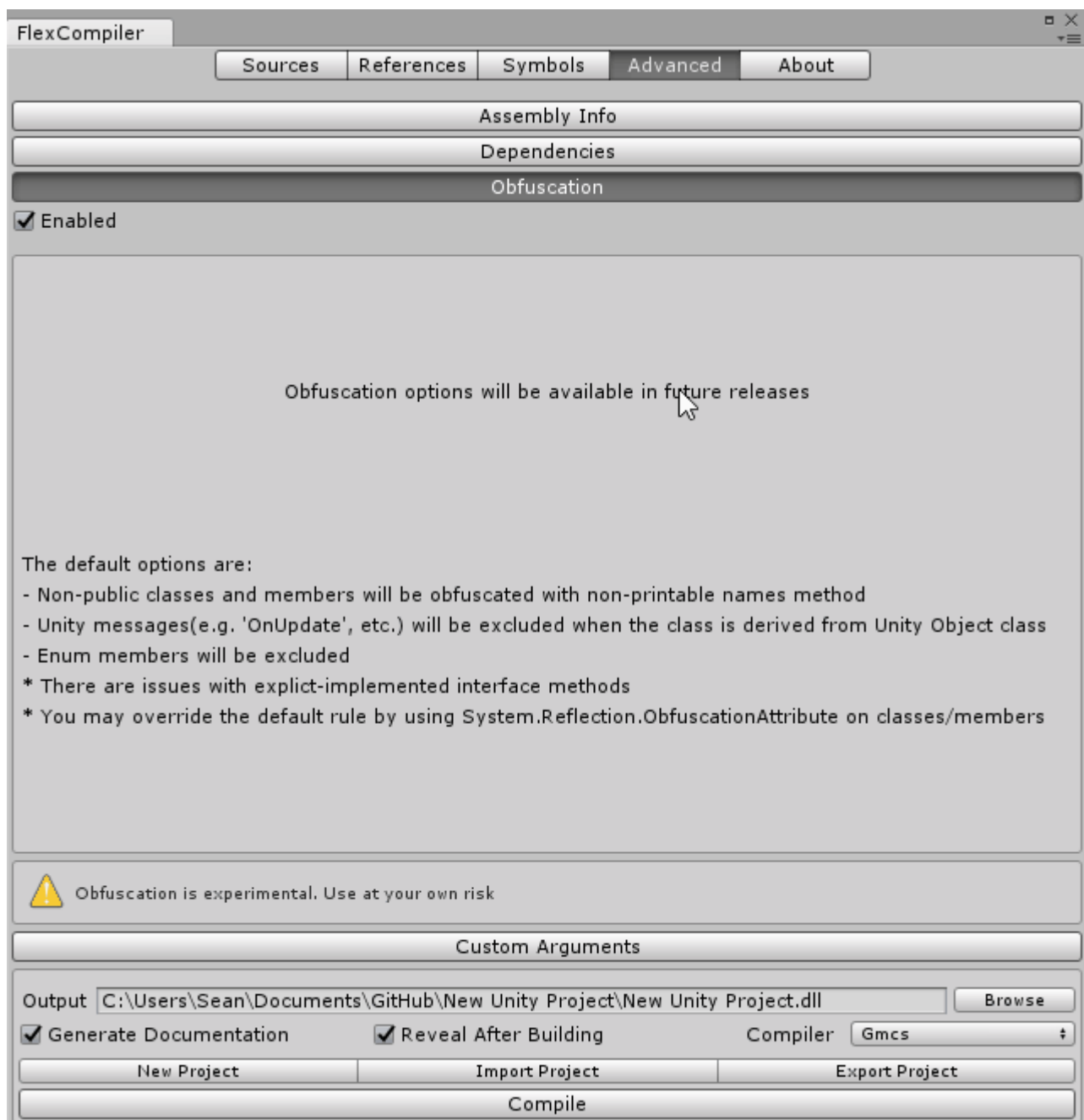
## Compile

When ready, click **Compile** at the bottom.

You can check **Reveal After Building**.

If everything's right, The output dll will be revealed in Finder/FileExplorer and the console will say *Building Success*.

If the building fails, you should check the console for errors.



## Tips

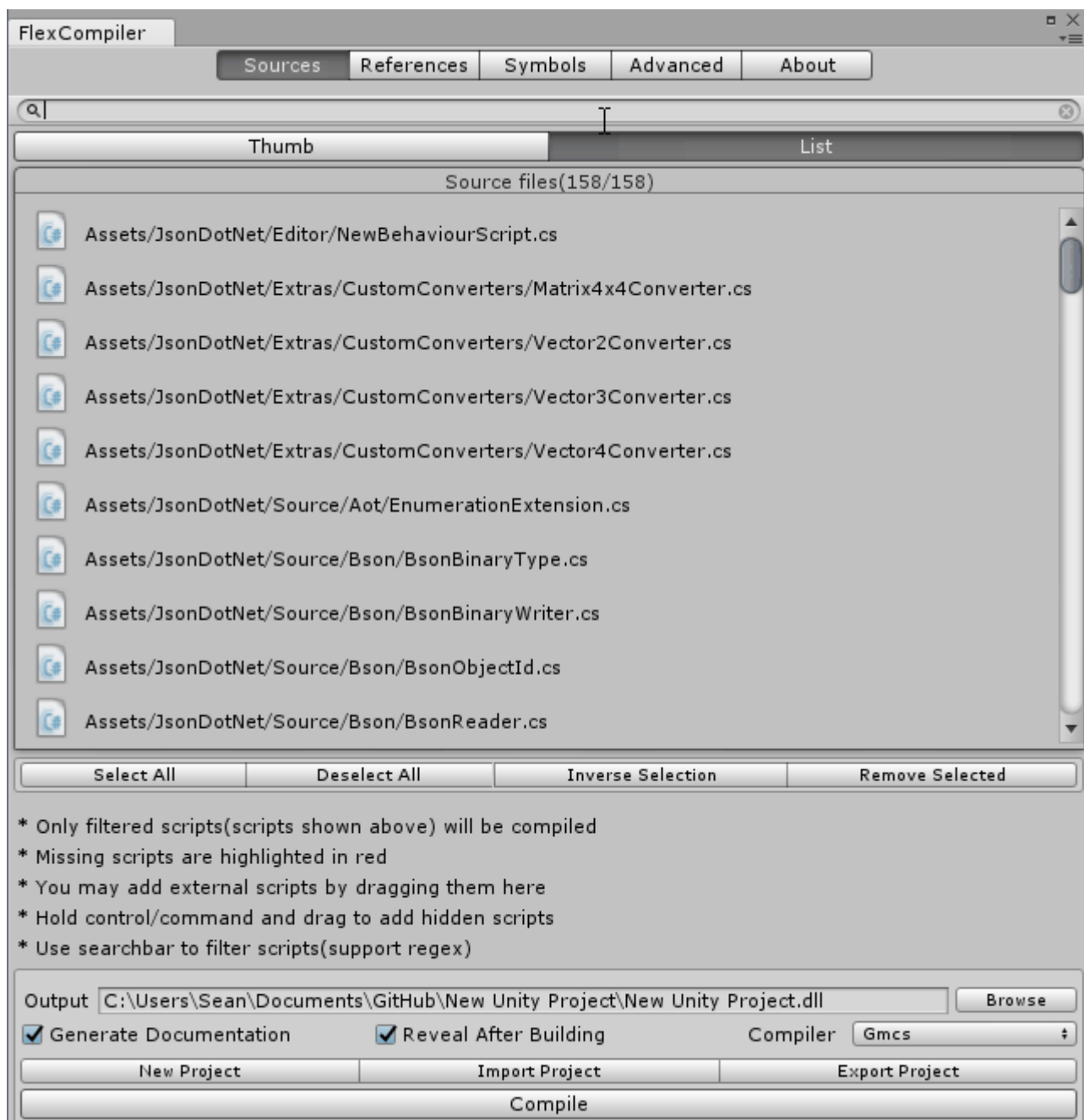
### How to make non-editor builds?

First you should exclude all editor scripts from source panel by removing them or filtering them out.

It's recommended to put editor scripts in subfolders named *Editor*.

To exclude *Editor* folders by filtering, enter `^(?!/Editor/).*` in searchbar.

To do the contrary, enter `/Editor/`.



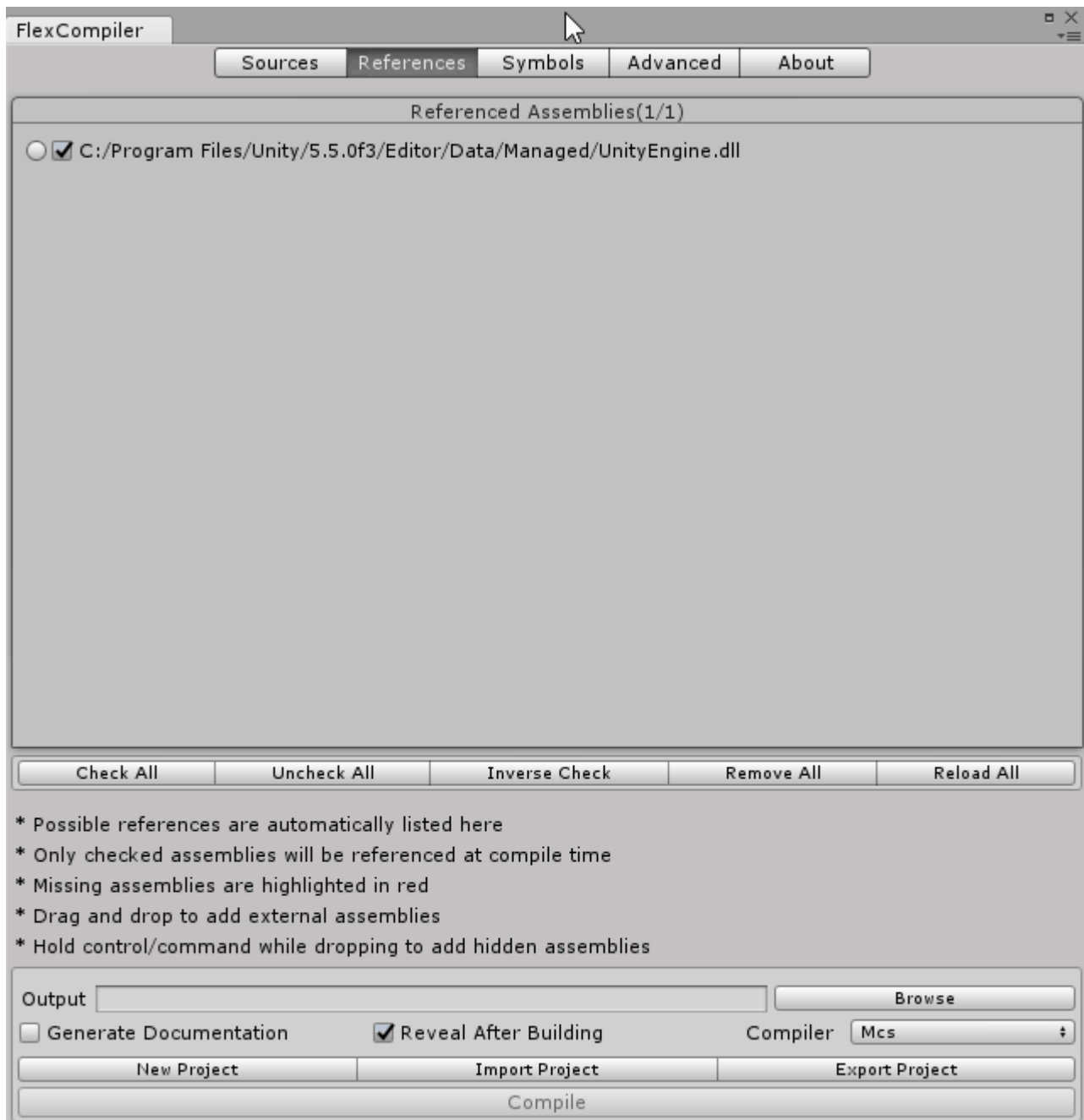
Switch to **References** panel, uncheck **UnityEditor.dll** and any editor-related assemblies.

Next, you need to remove or comment out editor directives:

**UNITY\_EDITOR**, **UNITY\_EDITOR\_64**, **UNITY\_EDITOR\_WIN** in symbols panel.

This will prevent codes wrapped with them from being compiled.

```
#if UNITY_EDITOR
    public void Do()
    {
        //...
    }
#endif
```



## How to make dependency builds?

Assume that you need two separate assemblies(dlls) generated from the same source codes: `Tween.dll`, `TweenEditor.dll`.

`Tween.dll` can work on its own. But `TweenEditor.dll` cannot, i.e. it requires `Tween.dll`.

This is what called **Dependency**.

The non-editor build is a project, which we called Project-A. The editor build is Project-B. Project-B depends on Project-A.

To setup the build process, first we create Project-A.

Follow the [Non-editor build guide](#) to strip-off editor codes.

Then we export Project-A by clicking `Export Project` at the bottom.

Next, click `New Project`, setup Project-B. This time we only keeps editor scripts.

**Important:** If one source file is used both by editor and non-editor builds, i.e. you have

`UNITY_EDITOR`, `UNITY_EDITOR_64`, `UNITY_EDITOR_WIN` in it, you should separate the codes with `#if...#else`

directives.

Otherwise some parts will be compiled duplicate times, which leads to assembly loading errors.

When Project-B is setup correctly, switch to `Advanced` tab, toggle on `Dependencies`.

Drag and drop the exported Project-A here.

Now you can just compile Project-B without a pain. Project-A will be outputted where you set it. And Project-B will reference outputted A automatically.

**Important:** Dependency projects will not be compiled again if the output result already exists.

## Trouble Shooting

---

error CS0234: The type or namespace name `Xml` does not exist in the namespace `System`. Are you missing `'System.Xml'` assembly reference?

**Solution:** Switch compilers to `gmcs`, `smcs` or `csc` other than `mcs`.

## Online Docs

---

You may check the latest online version of the doc here:

<http://compiler.flexflew.com>