Preprocessor Symbol Definition Files

Preprocessor Symbol Definition Files offer a simple way to manage custom #defines in a Unity project for multiple platforms.

Links

- Online Documentation
- GitHub Repository
- Download Unity Package
- Asset Store

Donate ♥

Contents

- Technical Information
- Contact & Legal Information

- Preface
- Preprocessor-Symbol-Definition-File
- Preprocessor-Definition-Settings
- Practical Example (Steamworks.NET)
- Practical Example (Thread Dispatcher)
- Available Symbols
- Install Guide
- Uninstall Guide
- FAQ

Technical Information

• Unity Version:	Unity 2019.1.0f2 or newer
• Api Compatibility Level:	.NET Standard 2.0 or .NET 4.x
• Scripting Backend:	Mono or IL2CPP
• Last Update:	08.09.2021
• Asset Version:	1.0.1

Contact & Legal Information

• Author:	© 2021 Jonathan Lang
• Contact:	johnbaracudagames@gmail.com
• License:	MIT License

Preface

Unity allows Platform dependent compilation. One aspect of this feature are custom #defines, also known as preprocessor symbols or scripting define symbols. These symbols allow you to tell the compiler what should and what should not be compiled. There are two main ways to do this. First by using an #if compiler directive which will tell the compiler to completely ignore aspects of your code, even preventing your code from reaching the IL, and second by using the ConditionalAttribute Class that omits calls to specified code.

This asset offers you an easy way to manage those symbols in your unity project by providing dedicated files called: Preprocessor Symbol Definition Files, that enable you to handle those symbols in an intuitive and intelligible way.

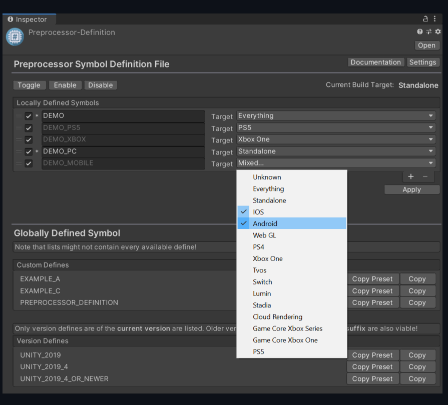
Preprocessor Symbol Definition Files

Preprocessor Symbol Definition Files are dedicated objects that store and manage custom preprocessor symbols.

Create a new file in your **Project** window by selecting the folder where you would like to create the file and then navigate to (menu: Assets > Create > Preprocessor Definition) or (rightclick: Create > Preprocessor Definition).

Add a new symbol by making a new entry in the Locally Defined Symbols list, then press Apply to confirm your changes. Both adding and removing symbols must be applied manually. Pressing Apply forces Unity to refresh the list of script definition symbols and reload the assemblies. Entries contained in the list can be activated and deactivated manually.

The Build Target Group of each symbol can also be set individually. Symbols are only active if their build target group matches the active build target group of the project. Definition files allow you to select multiple build target groups for a symbol simultaneously.



Inspector
Preprocessor-Definition
Open
Preprocessor Symbol Definition File
Documentation
Settings
Toggle
Enable
Disable
Current Build Target: Standalone
Locally Defined Symbols
DEMO
DEMO_PS5
DEMO_XBOX
DEMO_PC
DEMO_MOBILE
Target
Everything
PS5
Xbox One
Standalone
Mixed...
Unknown
Everything
Standalone
IOS
Android
Web GL
PS4
Xbox One
Tvos
Switch
Lumin
Stadia
Cloud Rendering
Game Core Xbox Series
Game Core Xbox One
PS5
Apply
Globally Defined Symbol
Note that lists might not contain every available define!
Custom Defines
EXAMPLE_A
EXAMPLE_C
PREPROCESSOR_DEFINITION
Only version defines are of the current version are listed. Older ver suffix are also viable!
Version Defines
UNITY_2019
UNITY_2019_4
UNITY_2019_4_OR_NEWER
Copy Preset
Copy

Preprocessor Definition Settings

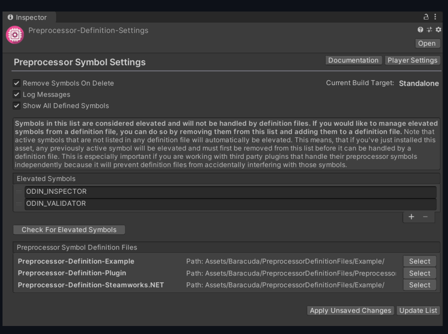
The settings file is located as an asset in your project and acts not only as a configuration cache, but also as a tool to manage global values and as a tool to locate and manage every definition file in your project. Select this file manually: navigate to (menu: Tools > Preprocessor-Symbol-Definition-File > Settings). Please ensure that there is only on instance of this type file at any time.

Remove Symbols on Delete
Removes the content of a Preprocessor Symbol Definition File when it is deleted. If this option is not enabled the symbols of a deleted file will be elevated and must be removed manually.

Log Messages
When enabled, messages will be logged when symbols are removed, added or elevated.

Save And Apply on Load
This option is only available in Unity 2020.2 or newer. When enabled, unsaved changes will be applied when scripts are about to recompile.

Elevated Symbols
Symbols in this list are considered elevated and will not be handled by definition files. If you would like to manage elevated symbols from a definition file, you can do so by removing them from this list and adding them to a definition file. Note that active symbols that are not listed in any definition file will automatically be elevated. This means, that if you've just installed this asset, any previously active symbol will be elevated and must first be removed from this list before it can be handled by a definition file. This is especially important if you are working with third party plugins that handle their preprocessor symbols independently because it will prevent definition files from accidentally interfering with those symbols.



Inspector
Preprocessor-Definition-Settings
Open
Preprocessor Symbol Settings
Documentation
Player Settings
Current Build Target: Standalone
Elevated Symbols
Check For Elevated Symbols
Preprocessor Symbol Definition Files
Select
Apply Unsaved Changes
Update List

☑ Remove Symbols On Delete
☑ Log Messages
☑ Show All Defined Symbols

Symbols in this list are considered elevated and will not be handled by definition files. If you would like to manage elevated symbols from a definition file, you can do so by removing them from this list and adding them to a definition file. Note that active symbols that are not listed in any definition file will automatically be elevated. This means, that if you've just installed this asset, any previously active symbol will be elevated and must first be removed from this list before it can be handled by a definition file. This is especially important if you are working with third party plugins that handle their preprocessor symbols independently because it will prevent definition files from accidentally interfering with those symbols.

ODIN_INSPECTOR
ODIN_VALIDATOR

Preprocessor-Definition-Example	Path: Assets/Baracuda/PreprocessorDefinitionFiles/Example/
Preprocessor-Definition-Plugin	Path: Assets/Baracuda/PreprocessorDefinitionFiles/Preprocesso
Preprocessor-Definition-Steamworks.NET	Path: Assets/Baracuda/PreprocessorDefinitionFiles/Example/

The settings file also displays every definition file, located anywhere in the project and enables each of these files to be selected manually. You can manually check whether the list is complete or if there are some definition files located in the project that are not in the list by pressing **Update List**. Pressing **Apply Unsaved Changes** will check if unsaved changes are present in any of the listed definition files and apply them.

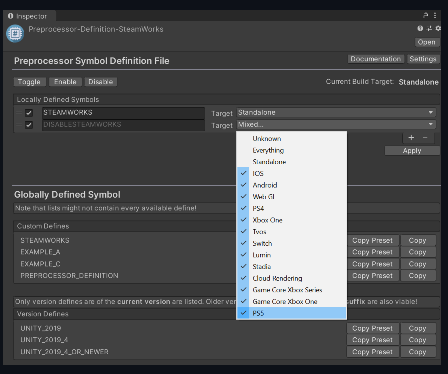
Practical Example : Steamworks.NET

Consider the following: we are creating a multi-platform game whilst using Steamworks.NET. If you've ever worked with Steamworks.NET in a multi-platform project, you know that you must manually define: DISABLESTEAMWORKS for the majority of your target platforms to disable it, which must be repeated every time you add a new platform. You also have to wrap your Steamworks API calls in an #if compiler directive. Although not the most time-consuming process, it is definitely a very important but sometimes unintelligible process. In general, adding and removing custom symbols for many platforms, especially in large projects with a decent amount of custom symbols can be a very delicate task.

```
using System;
using System.Collections.Generic;
using UnityEngine;

public class Player : MonoBehaviour
{
   private void Awake()
   {
#if !DISABLESTEAMWORKS

      // Add steam logic here...

#endif //!DISABLESTEAMWORKS
   }

}
```

If we use this asset instead, we can handle this scenario much more elegantly, simply by creating a Preprocessor Symbol Definition File and adding a new entry to define DISABLESTEAMWORKS. We then select the platforms on which we want the symbol to be active and press apply. Every time we switch platforms, the definition file will automatically check if either needs to define or un-define the symbol.

I've defined another symbol for this example: STEAMWORKS is an optional symbol that will be activated when DISABLESTEAMWORKS is disabled. We can then wrap our Steamworks API calls in a #if STEAMWORKS block instead of an #if !DISABLESTEAMWORKS which will improve the clarity of our code and enable us to individually activate/deactivate our steam code without affecting the internals of the Steamworks API itself.

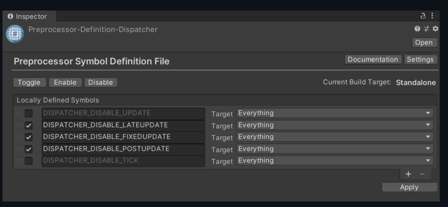


Inspector
Preprocessor-Definition-SteamWorks
Open
Preprocessor Symbol Definition File
Documentation
Settings
Toggle
Enable
Disable
Current Build Target: Standalone
Locally Defined Symbols
STEAMWORKS
DISABLESTEAMWORKS
Target
Standalone
Mixed...
Unknown
Everything
Standalone
IOS
Android
Web GL
PS4
Xbox One
Tvos
Switch
Lumin
Stadia
Cloud Rendering
Game Core Xbox Series
Game Core Xbox One
PS5
Apply
Globally Defined Symbol
Note that lists might not contain every available define!
Custom Defines
STEAMWORKS
EXAMPLE_A
EXAMPLE_C
PREPROCESSOR_DEFINITION
Only version defines are of the current version are listed. Older ver suffix are also viable!
Version Defines
UNITY_2019
UNITY_2019_4
UNITY_2019_4_OR_NEWER
Copy Preset
Copy

Practical Example : Thread Dispatcher

Another practical use case for this asset can be found in one of my other assets: Thread Dispatcher, a tool to pass the execution of a delegate or coroutine from a background thread back to the main thread. Thread Dispatcher uses preprocessor symbols to enable unneeded aspects of its player loop to be completely disabled in order to reduce unnecessary overhead in performance-critical projects. More Details.

Preprocessor Symbol Definition Files can be used by the Thread Dispatcher asset to manage those symbols in a clean, intuitive and intelligible way. Two clicks are all it takes to switch entire modules of the asset on and off. The image below shows a definition file that contains every preprocessor symbol used by the Thread Dispatcher asset. This not only enables the symbols to be switched on and off more easily, but also helps to get a better and faster overview of the current state of the asset. In addition, the symbols are applied across all platforms and therefore do not have to be added or removed again manually every time the platform is changed.



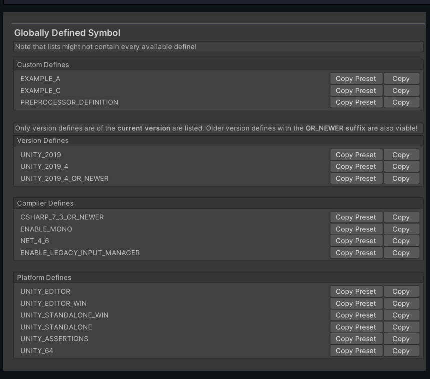
Inspector
Preprocessor-Definition-Dispatcher
Open
Preprocessor Symbol Definition File
Documentation
Settings
Toggle
Enable
Disable
Current Build Target: Standalone
Locally Defined Symbols
Apply

	Symbol		Target
☐	DISPATCHER_DISABLE_UPDATE	Target	Everything
☑	DISPATCHER_DISABLE_LATEUPDATE	Target	Everything
☑	DISPATCHER_DISABLE_FIXEDUPDATE	Target	Everything
☑	DISPATCHER_DISABLE_POSTUPDATE	Target	Everything
☐	DISPATCHER_DISABLE_TICK	Target	Everything

Available Symbols

Both, definition files and the settings file can display multiple categories of defined symbols. The contents of those lists, except for the custom defines, might not contain every available symbol. Every symbol defined by Unity is included in these lists. Every entry offers two quick ways to copy its content to the clipboard. Use Copy for the unchanged symbol and Copy Preset to copy the symbol with the following format:

```
#if SYMBOL

#endif
```



Globally Defined Symbol
Copy Preset
Copy

Note that lists might not contain every available define!

Custom Defines
EXAMPLE_A
EXAMPLE_C
PREPROCESSOR_DEFINITION

Only version defines are of the **current version** are listed. Older version defines with the **OR_NEWER suffix** are also viable!

Version Defines
UNITY_2019
UNITY_2019_4
UNITY_2019_4_OR_NEWER

Compiler Defines
CSHARP_7_3_OR_NEWER
ENABLE_MONO
NET_4_6
ENABLE_LEGACY_INPUT_MANAGER

Platform Defines
UNITY_EDITOR
UNITY_EDITOR_WIN
UNITY_STANDALONE_WIN
UNITY_STANDALONE
UNITY_ASSERTIONS
UNITY_64

Install Guide

Before importing this asset, please make sure that the symbol PPSDF is not active in your project. (This could only happen if you've used this asset previously in your project)

Right now there is a known issue in unity 2020.3.15f2 where unity will not register the assembly-definition-file of this asset correctly when re-importing. This issue can be fixed simply by forcing unity to reload the project (e.g. moving a folder, restarting the project.) I was unable to reproduce this bug in another project. Please let me know if you encounter anything that could be related to this issue.

Uninstall Guide

If you are deleting individual files and Remove Symbols on Delete is enabled, the symbols managed by each individual file will be removed automatically. However, if you are deleting the whole package, individual symbols will not be removed. If you plan to remove this asset from your project you should proceed according to the following points.

1. Select the settings file and scan the whole project for Preprocessor-Definition-Files (press: Validate Files).
2. Enable or disable Remove Symbols on Delete, depending on whether you want to keep or remove this symbols of a file.
3. Select every definition file individually and delete it.
4. Delete the contents of this asset. (default location: Assets / Baracuda / PreprocessorDefinitionFiles)
5. Finally, make sure that the custom define PPSDF is removed.

Be aware that if you delete the folder containing this asset, individual definition files that are located in your project will not be deleted, but will become zombie files that can be resurrected when this asset is re-imported. Not removing the PPSDF symbol could also cause issues when re-importing this asset.

FAQ

I get an error when (re-)importing this asset!
Right now there is a known issue in unity 2020.3.15f2 where unity will not register the assembly-definition-file of this asset correctly when re-importing. This issue can be fixed simply by forcing unity to reload the project (e.g. moving a folder, restarting the project.) I was unable to reproduce this bug in another project. Please let me know if you encounter anything that could be related to this issue.

I found a bug!
Contact me directly via Mail or use my socials. If it's a bug, you know how to fix it, and you have some spare time here is the link to the repository: GitHub.

Why this asset?
Take a look at the Practical Example if you are not sure if this is what you need.

How can I support you?
The best way to support me would be a donation. You can also support me with some exposure on Twitter, GitHub, Itch or a good review on the Asset Store.

I have another question!
Contact me: Mail, Twitter.

© 2021 Jonathan Lang