

Code Manager

Manual



16.12.2021
Version 1.0

Silvan Ott

Content

1	Overview.....	3
2	Requirements	3
3	Compatibility	3
4	Setup.....	3
5	Features.....	3
5.1	Script Templates.....	3
5.1.1	Define script folders	3
5.1.2	Create script templates	4
5.1.3	Create new script.....	4
5.1.4	Placeholders	5
5.2	Clean Code.....	6
5.2.1	Create Regexes	6
5.2.2	Test Regex Window	7
5.2.3	Create Clean Code Rules	9
5.2.4	Select Clean Code Rules	16
5.2.5	Clean Code Console.....	17
5.3	Code Inspector	18
5.3.1	Start an inspection.....	18
5.3.2	Settings	19
5.3.3	Read.....	19
5.3.4	Write.....	19
5.3.5	Additional information	20
5.4	Code Manager Settings	20
6	Additional information	20
6.1	Regex	20
6.2	Contact	20

1 Overview

The Code Manager has two main features. The First Feature is 'Script Templates'. With Script Templates you can create generic templates from which you can create scripts in the before defined folder structure. With the second main feature 'Clean Code' you can create clean code rules and then scan the folders for your clean code rule violations. You can create clean code rules with regexes to find either unwanted- or undocumented code or define coding guidelines.

2 Requirements

There are no requirements for Code Manager, but it is recommended to have a basic understanding of regexes. Otherwise, there are some links to learn the basics of regex at the end of the document.

3 Compatibility

Code Manager was developed in Unity 2020.3.3f1 on Windows 10. Older Unity versions might work but were not verified.

4 Setup

Either import the Unity package in your project or download the tool from GitHub: <https://github.com/Morchul/CodeManager>. After downloading, place the folder under Assets/Plugins/ in your project.

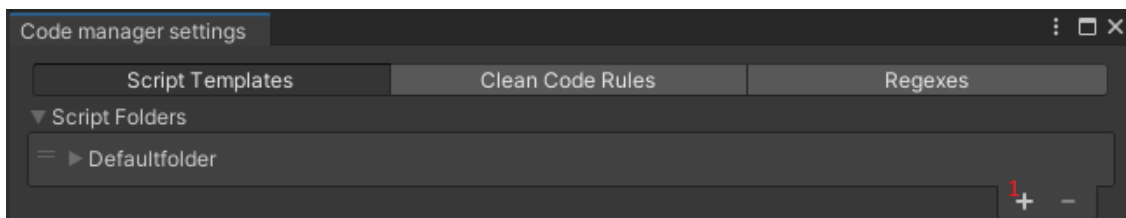
5 Features

5.1 Script Templates

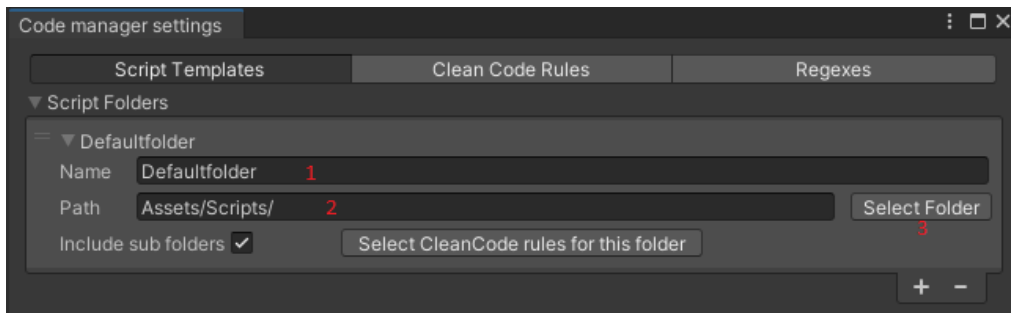
With Script Templates you create templates which can contain placeholders. In the settings you can give every placeholder a value. You then can create a new script from one of these templates and during script creation the placeholders will be replaced through your defined values. With this you can use the same templates in multiple projects, and just need to adjust some placeholder values. Script Templates also lets you define script folders. During script creation you can choose one of the script folders to create the script in there or input the creation path manually.

5.1.1 Define script folders

1. Open the Code Manager settings under: Code Manager => Settings.
2. Open the Script Folders list and add a new item by pressing the "+" (1) button.



3. Expand the new Item.
 - In the Name field (1) you can give your script folder a name. (This does not have to match the real folder and is just for identification)
 - In the Path field (2) you now can either write the path to the folder or select the folder in the explorer by pressing the Select Folder (3) button. The Folder must be under Assets/ and at the end of the path there must be a "/" so the path points into the folder.

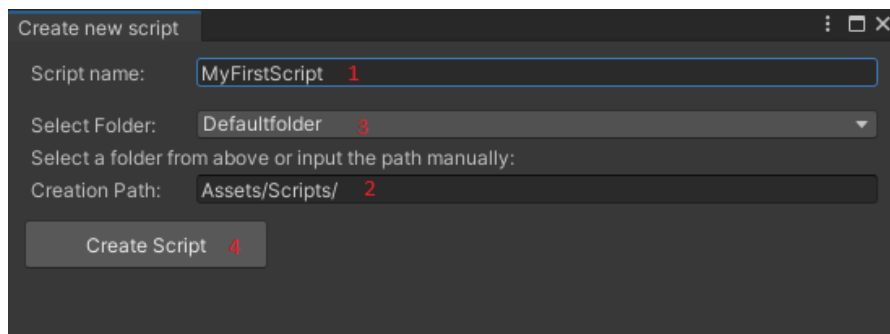


5.1.2 Create script templates

1. Click the menu: Code Manager => Script Templates => New script template.
2. The explorer will open. Enter the name of the new script template and press Save.
 - The script template must be under the folder Assets/ScriptTemplates/
 - The template must be a .txt
 - You do not have to use the menu to create a template. You can create a .txt by your own and simply put it somewhere under Assets/ScriptTemplates/

5.1.3 Create new script

1. Open the Script Create window under: Code Manager => Script Template => New script or with the shortcut Ctrl + T
2. Here you can see all templates you have created. Select the template from which you want to create your script.
3. A new window will open where you can input the script name (1) and the path where the script should be created (2). You can either select one of the script folders (3) which will set the path or write the path manually. The folders to select are the script folders created in the settings.
4. Press the Create Script (4) button to create the new script.

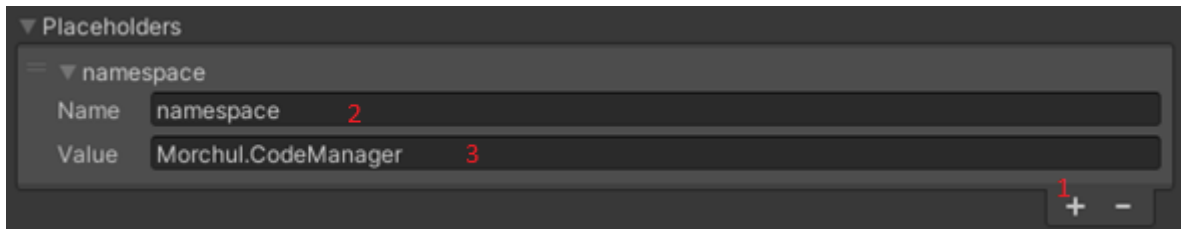


You can also open the Script create window by right clicking on a folder in your Project and navigate to the menu: Create => Script from template. The path will then automatically be filled with the path of the folder where you right clicked.

5.1.4 Placeholders

To create a new placeholder, open the Code Manager settings under: Code Manager => Settings. Open the Placeholders list and add a new item (1).

Write the name of the placeholder in the Name field (2) and which value the placeholder should have after creation in the Value field (3)



To use a placeholder in the template, write the Name of the placeholder surrounded by "%". After you create a new script from this template the placeholders will be replaced.

```
using UnityEngine;
using System.Collections;

namespace %namespace%
{
    public class %ScriptName% : MonoBehaviour
    {
        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
    }
}
```

```
using UnityEngine;
using System.Collections;

namespace Morchul.CodeManager
{
    public class MyFirstScript : MonoBehaviour
    {
        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
    }
}
```

How you can see there is already a placeholder in the default template: "%ScriptName%" this is a default placeholder and will be replaced through the name of the script.

It is in your own responsibility that a placeholder name never occurs twice in the settings. If so one placeholder will just be ignored.

List of default placeholders	
Placeholder name	Value
ScriptName	The name of the script
TemplateName	The name of the template

5.2 Clean Code

Clean Code is a tool to keep your scripts without any unwanted- or undocumented code and let you define coding guidelines. All this with Regexes. These three types (unwanted code, code documentation and code guideline) are called clean code rules and work the following:

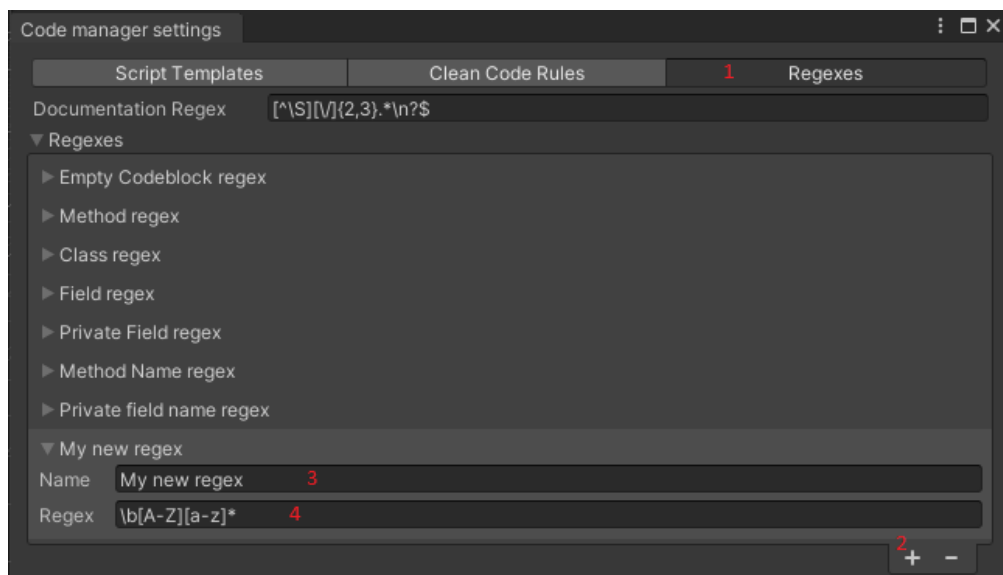
- **Unwanted Code:** You can define a regex for code you do not want. If this regex matches anything, it will be displayed to you in the clean code console as clean code violation.
- **Code Documentation:** You can define a regex for code you want to have documented. If this regex matches with something, the code before this match will be tested with a Documentation regex. This documentation regex should check if there is a documentation. If there is not any, a clean code violation will be displayed in the clean code console.
- **Code Guidelines:** You define a search regex which has at least one group name in it. If this search regex matches with something the value of the group specified by the group name will be tested with the match regex, which you can also define by yourself. If this test does not match a clean code violation will be displayed in the clean code console.

There is a clean code console in which all clean code violations will be displayed. By clicking on them you will be directed to the place where the clean code violation occurred.

For each created script folder, you can choose between all clean code rules you created. The folder will then be scanned for violations of all the chosen clean code rules.

5.2.1 Create Regexes

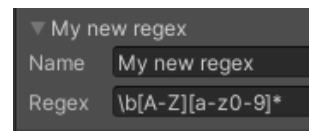
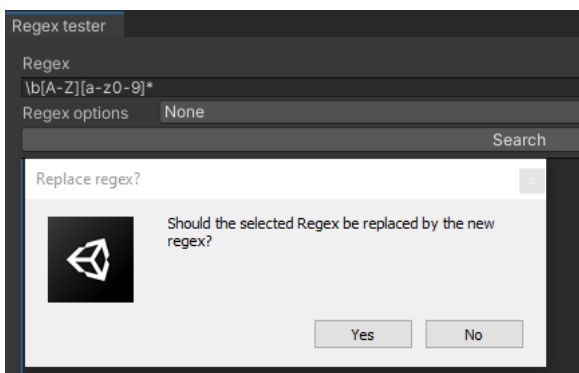
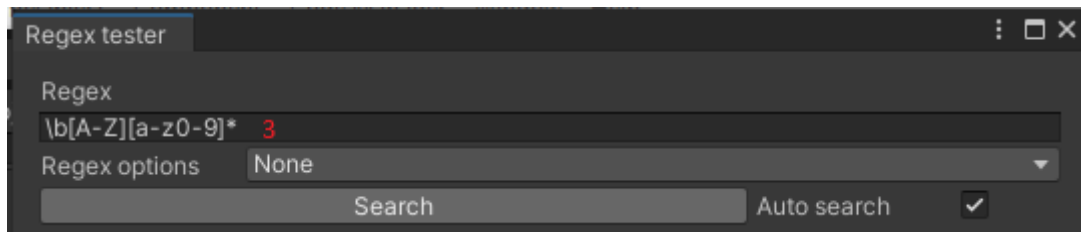
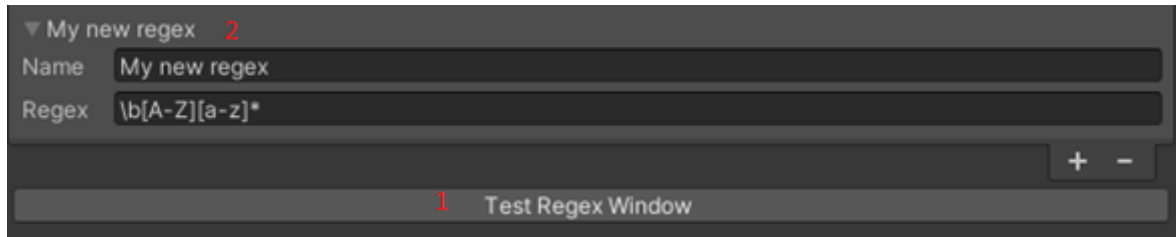
1. Open the Code Manager settings under: Code Manager => Settings.
2. Change to the Regexes Tab (1), open the Regexes list, and add a new item (2)
3. Enter the name of the regex in the Name field (3) and the regex in the Regex field (4). If the Regex is not valid a message will appear beneath the Regex field.



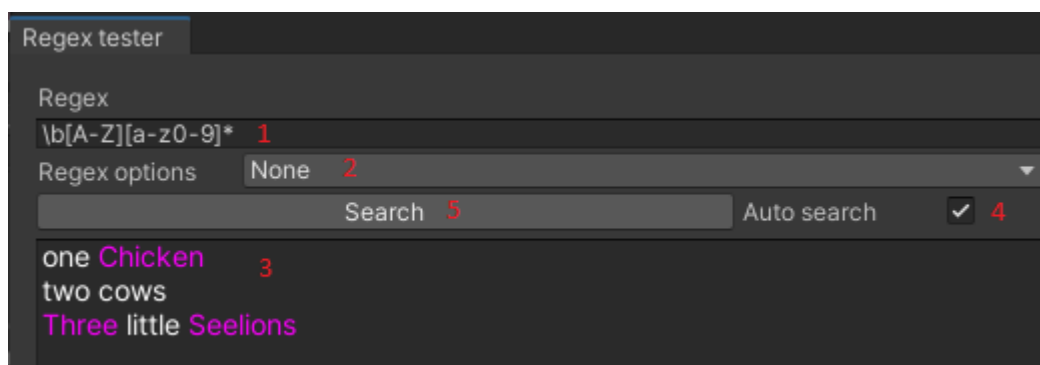
5.2.2 Test Regex Window

With the Test Regex Window, you can check your regexes and update them easily.

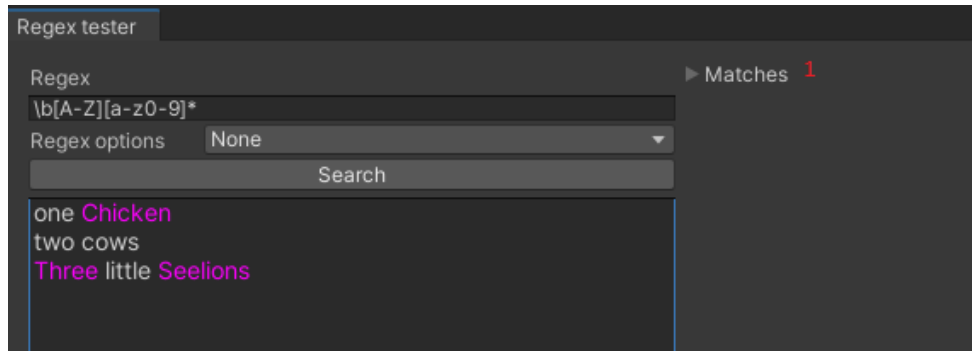
To open the Test Regex Window, press the “Test Regex Window” (1) Button under the Regexes List. It is important to check which regex in the Regexes list is selected before you press the button. The selected Regex (2) will be automatically filled in the Regex field (3) when the window opens. Also, if you close the window and you made any changes to the regex, the Test Regex Window will ask if you want to replace the regex with the new one. If you press yes, the selected regex will be replaced by the new one.



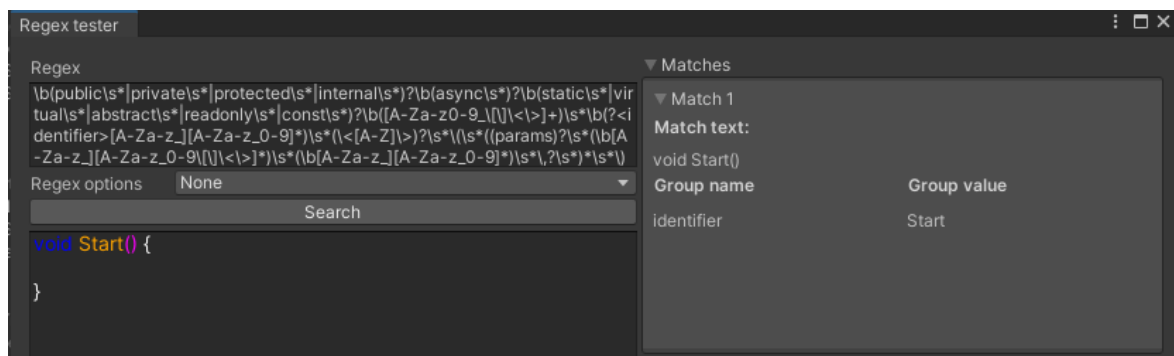
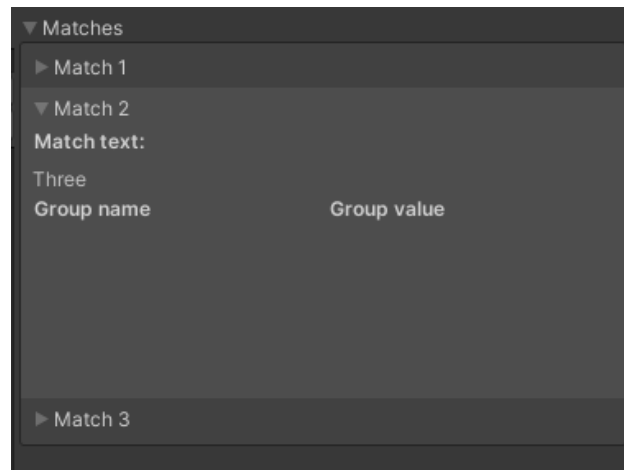
In the Test Regex Window, you can input a regex (1), select regex options (2) and input some text (3). A search will automatically start if something has changed, and the Auto search (4) is set. Else you must press the Search (5) button.



If a match was found, you will see a Matches List appearing on the right side (1). The text will also take some color where the match was found.



If you expand the Matches list, you will find every match that was found in the text. Expand a match to see which text exactly matched and which groups were found if groups were specified. If groups were specified, you also see a difference in the color highlighting.

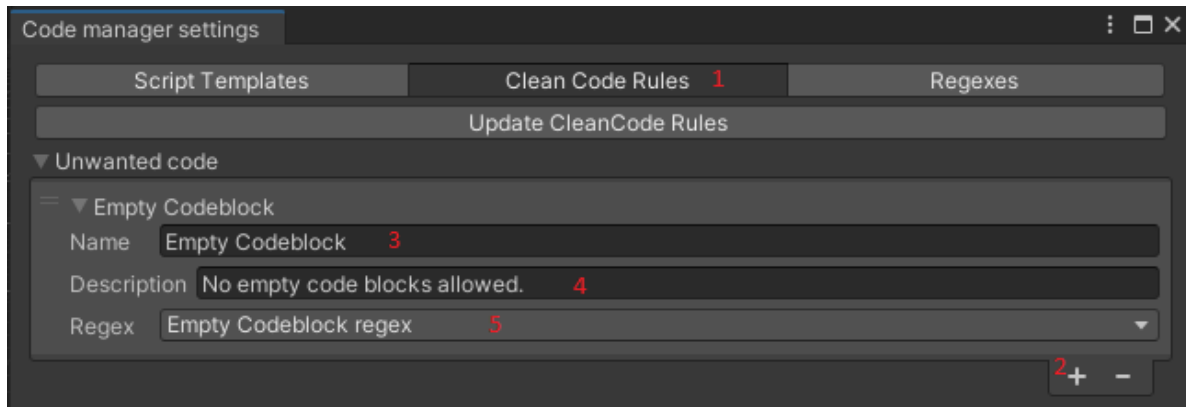


5.2.3 Create Clean Code Rules

There are three different types of clean code rules. Unwanted code, code documentation and code guideline.

Unwanted Code:

1. Open the Code Manager Settings under: Code Manager => Settings.
2. Select the Clean Code Rules tab (1), open the Unwanted Code list and add a new item (2)
3. Enter the name of the unwanted code rule in the Name field (3). Enter a description in the Description field to explain the rule (4). Finally, select a Regex you have created which matches the unwanted code (5).



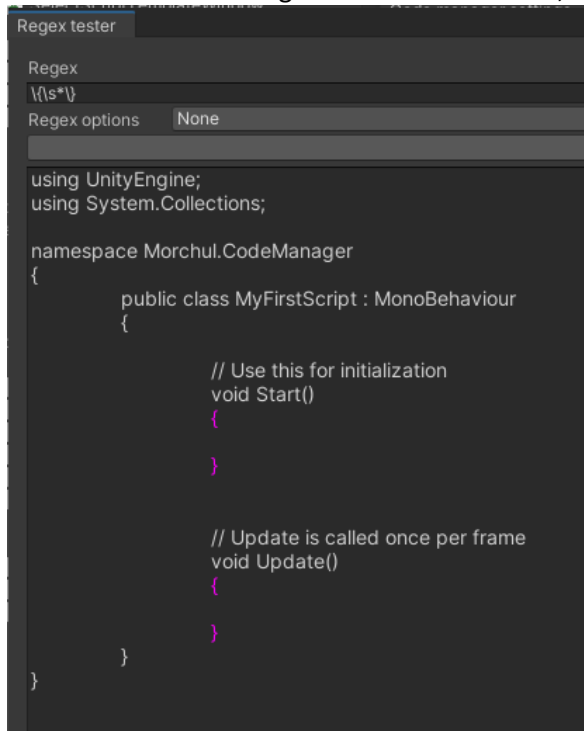
Every time the regex of the unwanted code will find a match in a script a clean code violation will be created.

Unwanted Code Example:

Unwanted Code	
Name	Empty Codeblock
Description	No empty code blocks allowed
Regex	Empty Codeblock regex

Regex	
Name	Empty Codeblock regex
Regex	<code>\{\s*\}</code>

The Unwanted code regex found two matches, meaning two clean code violations will be created.



Regex tester

Regex
`\{\s*\}`

Regex options

```
using UnityEngine;
using System.Collections;

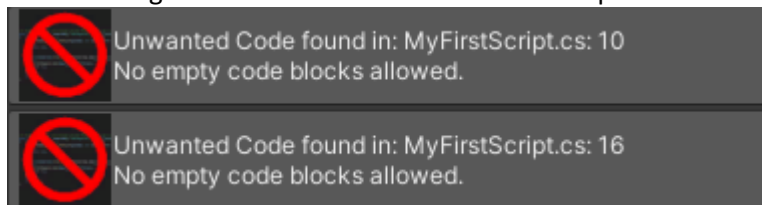
namespace Morchul.CodeManager
{
    public class MyFirstScript : MonoBehaviour
    {
        // Use this for initialization
        void Start()
        {

        }

        // Update is called once per frame
        void Update()
        {

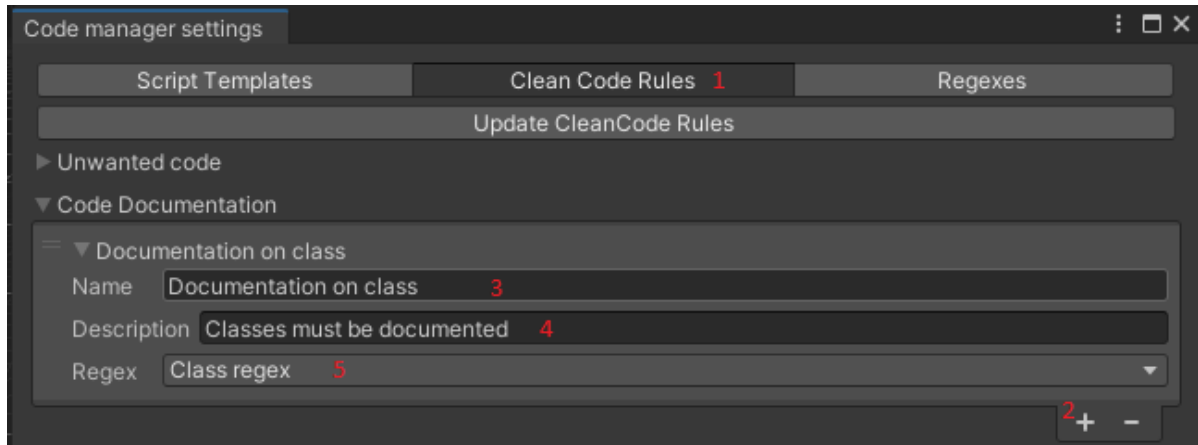
        }
    }
}
```

The resulting Clean Code violation with the description defined in the Unwanted Code:

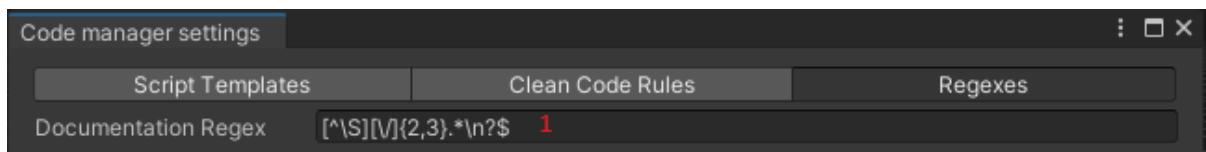


Code Documentation:

1. Open the Code Manager Settings under: Code Manager => Settings.
2. Select the Clean Code Rules tab (1), open the Code Documentation list and add an item (2)
3. Enter the name of the code documentation rule in the Name field (3). Enter a description in the Description field to explain the rule (4). Finally, select a Regex you have created which matches the code which should be documented (5).



Every time the code documentation regex will find a match, the code before this match will be tested with the Documentation Regex, found at the top of the Regexes Tab (1). If the code before does not match the Documentation Regex a clean code violation will be created.



The default Documentation Regex matches a documentation if the last line before the documented code starts with two or more “/”.

```

/// <summary>
/// Simple documentation
/// </summary>
public class MyFirstScript : MonoBehaviour
{
//Simple documentation
public class MyFirstScript : MonoBehaviour
{

```

Code Documentation Example:

Code Documentation	
Name	Documentation on class
Description	Classes must be documented
Regex	Class Regex

Regex	
Name	Class Regex
Regex	.*class.*\b(?<identifier>[A-Za-z_][A-Za-z_0-9]*)

Documentation Regex	[^\S][\V]{2,3}.*\n?\$
----------------------------	-----------------------

The Code documentation regex found a match:

The screenshot shows a 'Regex tester' window. The 'Regex' field contains the pattern `.*class.*\b(?<identifier>[A-Za-z_][A-Za-z_0-9]*)`. The 'Regex options' are set to 'None'. A 'Search' button is visible. Below the input fields, a code snippet is displayed with the following content:

```
using UnityEngine;
using System.Collections;

namespace Morchul.CodeManager
{
    public class MyFirstScript : MonoBehaviour
    {
        // Use this for initialization
        void Start() {

        }

        // Update is called once per frame
        void Update()
        {

        }
    }
}
```

Now the code before will be tested with the documentation regex

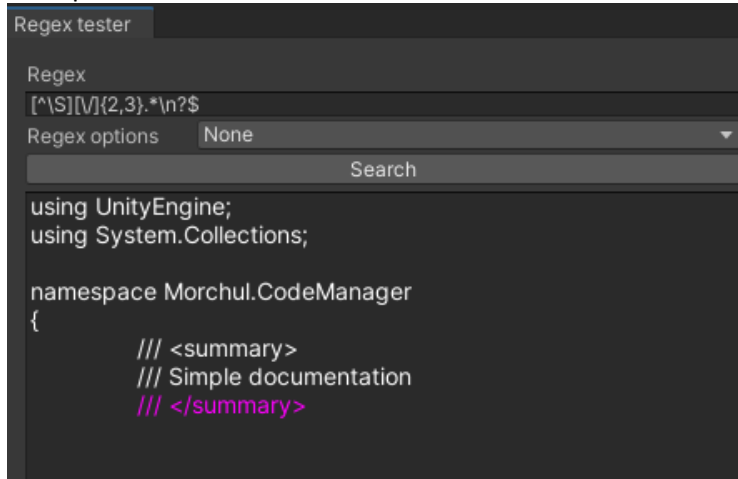
The screenshot shows the same 'Regex tester' window, but the 'Regex' field now contains the pattern `[^\S][\V]{2,3}.*\n?$`. The 'Regex options' remain 'None'. The same code snippet as in the previous screenshot is shown below the input fields.

No match was found so there is no documentation and a clean code violation will be created

The screenshot shows a code editor with a dark background. A message box is overlaid on the code, displaying the following text:

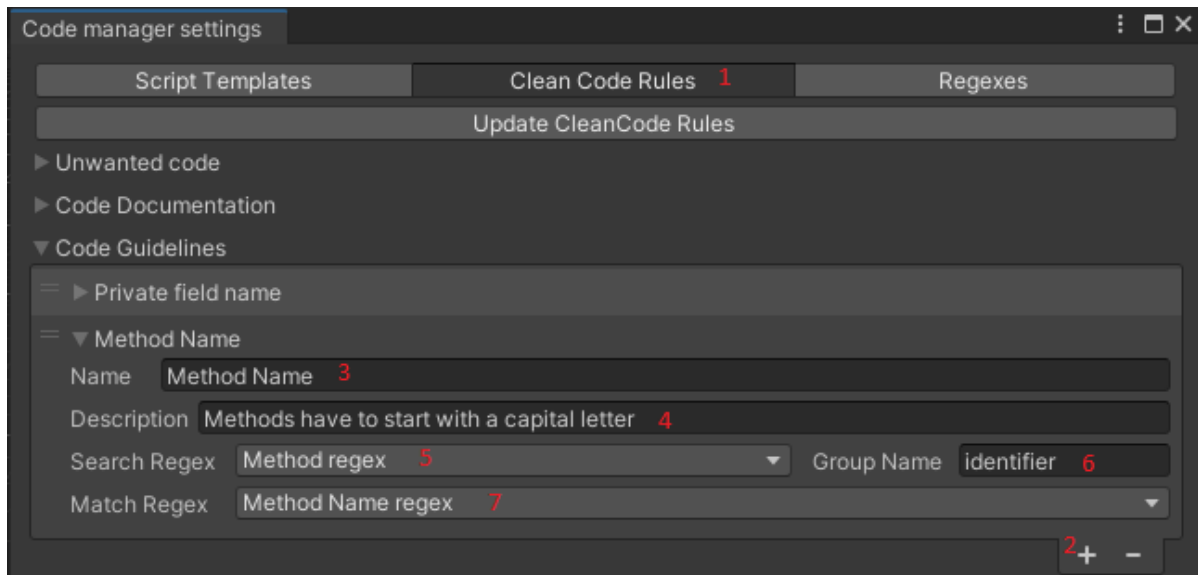
```
/// Summary:
/// Documentation
/// Summary:
Code not documented in: MyFirstScript.cs: 6
Classes must be documented
```

Example of documented code



Code Guideline:

1. Open the Code Manager Settings under: Code Manager => Settings.
2. Select the Clean Code Rules tab (1), open the Code Guideline list and add a new item (2)
3. Enter the name of the code guideline rule in the Name field (3). Enter a description in the Description field to explain the rule (4). Select a search Regex you have created which matches the code where the code guideline rule applies (5). This regex should contain a group name. Write the group name you want to check for in your code guideline rule into the Group Name field (6). Finally, select the Match Regex (7) with which the group value will be tested.



Code guideline rules work as follow: First, all will be searched with the Search Regex. If the Search Regex finds a match the value of the defined group will be tested with the Match Regex and if there is no match a clean code violation will be created.

A short example: The Search Regex would match any method declaration. With the group you then can select only the name of the method and with the Match Regex you check if the method name corresponds to your code guideline rule.

Code Guideline Example:

Code Guideline	
Name	Method Name
Description	Methods have to start with a capital letter
Search Regex	Method regex
Group Name	identifier
Match Regex	Method Name regex

Regex	
Name	Method Regex
Regex	<code>\b(public\s* private\s* protected\s* internal\s*)?\b(async\s*)?\b(static\s* virtual\s* abstract\s* readonly\s* const\s*)?\b([A-Za-z0-9_\[\]\<\>]+\s*\b(?<identifier>[A-Za-z_][A-Za-z_0-9]*)\s*(\<[A-Z]\>)?\s*(\s*((params)?\s*(\b[A-Za-z_][A-Za-z_0-9_\[\]\<\>]*)\s*(\b[A-Za-z_][A-Za-z_0-9]*)\s*\,\s*)*\s*\)</code>

Regex	
Name	Method Name Regex
Regex	<code>\b[A-Z][a-zA-Z_0-9]*</code>

The Search Regex found a match

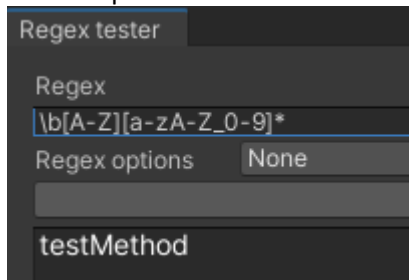
The screenshot shows a 'Regex tester' window. On the left, the 'Regex' field contains a complex pattern, and the 'Regex options' are set to 'None'. Below this is a 'Search' button. The main area displays a C# code snippet with a method named 'TestMethod'. On the right, the 'Matches' panel shows 'Match 1' with the 'Match text' being 'void TestMethod()'. Below this, a table lists the 'Group name' and 'Group value'. The 'identifier' group is highlighted with a red box, and its value is 'TestMethod'.

As we can see on the right-hand side, the value of our group identifier is "TestMethod". Now this value will be tested with the Match Regex.

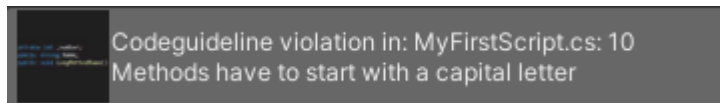
The screenshot shows a 'Regex tester' window. The 'Regex' field contains the pattern '\b[A-Z][a-zA-Z_0-9]*'. The 'Regex options' are set to 'None'. Below this is a 'Search' button. The main area displays the text 'TestMethod' in a pink font, indicating a successful match.

There is one match, so the value corresponds to the code guideline.

An example if there was no match:



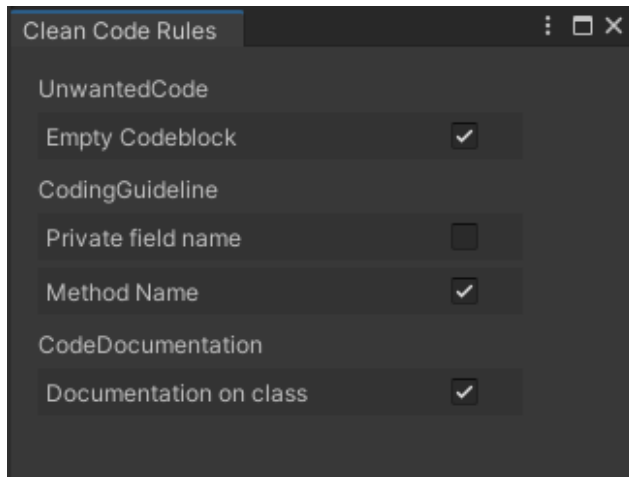
A clean code violation would be created:



5.2.4 Select Clean Code Rules

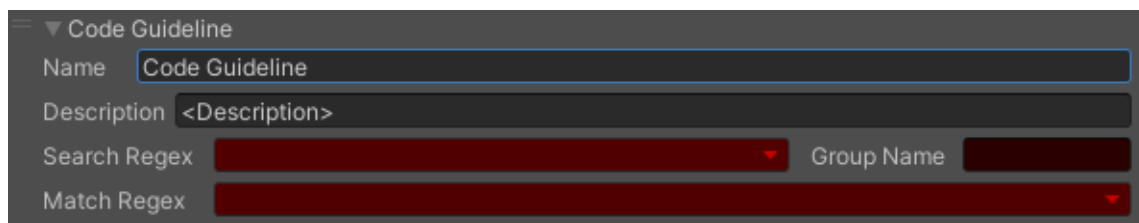
Now, after the creation of Clean Code rules, you must choose for which clean code rules the scripts inside the folder should be scanned. You do this on each script folder individually.

1. Open Script Templates settings under: Code Manager => Settings.
2. Expand the Script Folders list and the folder you want to edit.
3. Press the Button "Select CleanCode rules for this folder"
4. Select all Clean Code Rules for which you want to scan in this folder. If you have selected Include sub folders the scans will also be done in sub folders for the selected rules.

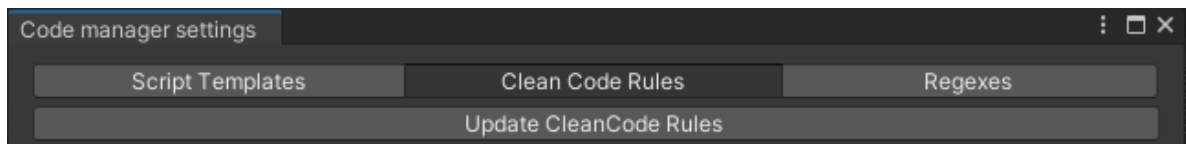


If you do not see your Clean Code rule to select, there are two possible reasons:

First, your Clean Code rule is not valid. This is indicated by the red fields in your clean code rule:



Second, your Clean Code rules are not updated. The Clean Code rules will be updated if you close the Code manger settings window or if you press the "Update CleanCode Rules" button at the top in the Clean Code Rules tab.

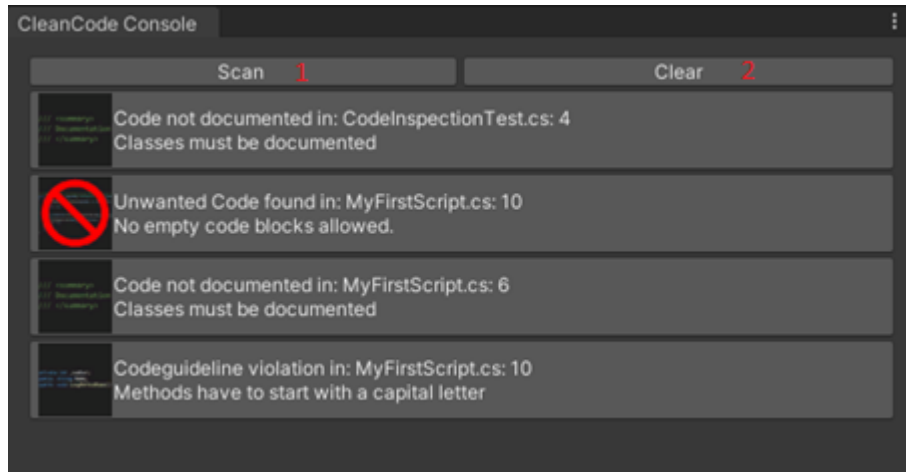


5.2.5 Clean Code Console

You can open the Clean Code Console under: Code Manager => Clean Code => Console. Or Ctrl + M
The console has two Buttons: Scan (1) to scan all folders for Clean Code violations and Clear (2) to clear the console.

All clean code violations will be displayed in the console. A clean code violation contains the type of the violation, the file name and line index where the violation occurred and the description of the rule which was broken.

By clicking on a clean code violation, the script will be opened at the location of the clean code violation.



5.3 Code Inspector

The Code Manager contains a tool to find and replace parts of an either plain text or text in a file. This tool can be accessed by code in your own project. The main classes are CodeInspector, CodeInspection and CodePiece.

With the CodeInspector you can start an inspection of plain text or a file. The CodeInspection is the instance which represents the inspection. If you then search with the CodeInspection for text, the code will be split in several CodePieces which can be edited individually. If the changes are made the CodeInspection will write the new changed code into the file or will change the internal plain text.

5.3.1 Start an inspection

To inspect a file or plain text you need the CodeInspector. There are two static methods to start an inspection and each of them returns a CodeInspection instance.

```
CodeInspection textInspection = CodeInspector.InspectText("PlainText");  
CodeInspection fileInspection = CodeInspector.InspectFile("Path/To/File.txt", InspectionMode.READ);
```

To inspect plain text, set the text as parameter in the InspectText method. To inspect a file, you must input the file path as first parameter. As second parameter you must choose an InspectionMode. There are two inspection modes: READ and READ_WRITE. You can have several READ inspections active on one file. But if you want to have a READ_WRITE inspection you must remove all other inspections from this file. You can either get write permission by setting the right mode directly as parameter or later with the method:

```
CodeInspector.StopFileInspection(fileInspection);
```

To stop inspecting a file so another inspection can get write permission you must call:

```
CodeInspector.StopFileInspection(fileInspection);
```

Methods which can only be used with write permission:

- AddFirst
- AddLast
- AddBefore
- AddAfter

Also, if you change the content of a CodePiece the changes will be discarded when you are not in READ_WRITE mode.

Plain text inspections always have READ_WRITE permission. Every inspection has its own copy of the string, and the string will not be shared even if you pass the same one.

5.3.2 Settings

The settings of each CodeInspection can be accessed through the Settings attribute. To change the settings simply assign new CodeInspectionSettings to the Settings attribute:

```
fileInspection.Settings = new CodeInspectionSettings()
{
    AddLineIndex = true,
    RegexOptions = System.Text.RegularExpressions.RegexOptions.None,
    RegexTimeout = System.TimeSpan.Zero
};
```

AddLineIndex	If this is set to true, the line count of each CodePiece generated from this code inspection will be calculated. Additionally, this allows you to calculate the line index of the CodePiece.
RegexOptions	Options used by the regex class in the code inspection
RegexTimeout	A timeout for the search action

5.3.3 Read

If you have created an inspection you can search after a certain code piece by regex.

Read methods are:

```
fileInspection.GetEverything(out LinkedListNode<CodePiece> wholeText);
fileInspection.Find("Regex", out LinkedListNode<CodePiece> foundCodePiece);
fileInspection.FindAll("Regex", out LinkedListNode<CodePiece>[] foundCodePieces);
```

Every one of the read methods is returning a Boolean which is true if something was found.

5.3.4 Write

To make any changes you first must ensure that your code inspection is in READ_WRITE mode. If so, you have several methods to add new code pieces:

```
LinkedListNode<CodePiece> newCodePiece1 = codeInspection.AddFirst("New Code add the beginning");
LinkedListNode<CodePiece> newCodePiece2 = codeInspection.AddLast("New Code add the end");
LinkedListNode<CodePiece> newCodePiece3 = codeInspection.AddBefore(foundCodePiece, "New Code before foundCodePiece");
LinkedListNode<CodePiece> newCodePiece4 = codeInspection.AddAfter(foundCodePiece, "New Code after foundCodePiece");
```

If you want to change a CodePiece you can edit the Code attribute:

```
foundCodePiece.Value.Code = "New changed code";
```

After you did all the changes, you can get the current text with the CreateCurrentCode method. With this method nothing will be changed inside the file or plain text. It has more the purpose of a preview.

```
Debug.Log("Current code: " + codeInspection.CreateCurrentCode());
```

To commit every change and write it to the file call Commit. To discard every change, call Cancel:

```
if (codeInspection.Commit())
{
    //Changes written successful
}
codeInspection.Cancel();
```

To access the current file code or plain text you can use the CompleteCode attribute.

This attribute will always be updated by a Cancel or Commit call and represents the current file content or the actual state of the plain text.

```
Debug.Log("Current Code: " + codeInspection.CompleteCode);
```

If you inspect a plain text you can set the CompleteCode with SetEverything method.

```
textInspection.SetEverything("New Code");
```

5.3.5 Additional information

If AddLineIndex is set to true in the settings, you can get the line index of a code piece from the code inspection:

```
int lineIndex = codeInspection.GetLineIndex(foundCodePiece);
```

There are some examples under:

Plugins/Morchul/CodeManager/Examples/CodeInspectorExamples.cs

5.4 Code Manager Settings

The Code Manager settings object is a ScriptableObject and can be shared by simply copy and paste. The Code Manager settings object must be named: CodeManagerSettings.asset and must be placed in the folder: Assets/Plugins/Morchul/CodeManager/Resources.

6 Additional information

6.1 Regex

Regexes are an important part of Code Manager. It is important to have some basic knowledge about them. Here are some links to learn regex and what they are capable of:

https://en.wikipedia.org/wiki/Regular_expression (Definition of Regex)

<https://docs.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.regex?view=net-5.0>

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference>

<https://www.softwaretestinghelp.com/csharp-regex-tutorial/>

<https://regex101.com/> (Online Regex Tester)

6.2 Contact

Email: morchul.dev@gmail.com