

# Unity Workflow Toolkit

Final Major Project TDD

# Contents

Document Information	3
Specifications	4
Software & Tools Used	5
Version Control	6-9
Documentation	10-11
Code Conventions	12-13
Copyrighting	14
Version List	15
Build Delivery Method	16-18
Logging	20
Reporting Bugs	21-22
Libraries	23
UML Structure	24-26
Save System	27-29
Performance Budgets	30
Challenges	31
Class Reference	32
Build Log	33
Bibliography	34
Credits	<b>35</b>

# Document Information

Start Date: 11/10/2024

Current Version: 0.8

Last Modified: 20/10/24

## Changelog

Date	Vers	Changes Made

# Specifications

Unity Version: 2022.3.18f1

Operating System: Windows 10 & 11

Memory: 500 MB

Storage: 10 MB

Current Build Size: 520 KB

# Software & Tools Used

## Software

Type	Name	Version
Game Engine	Unity	2022.3.18f1
Code Editor	Rider, Visual Studio Code	2024.1.2
Image Creation	Adobe Photoshop	2024
Documentation	Website, PlantUML	n/a, 7.11.2-u2023.2

## File Formats

Type	Format
Scripts	.cs .meta
Assets	.png
Documentation	.pdf .html .css, .puml
Misc	.unitypackage

# Version Control

## Types

When managing this project, one key area I need to consider is Version Control; it allows me to compare differences in project versions (keeping them preserved) while ensuring the security of the overall project itself. For this project, I am considering two types of version control: **online** and **offline**. Both come with upsides and downsides, so making decisions based on this project is more important than ever; here are some of the **upsides** of Online:

## What's the difference?

### Online

Online version control consists of websites such as Github or Gitlab, which offer a cloud-based repository to which you can upload your project, check file history and collaborate; this makes it suitable for individual developers and development teams.

### Offline

Offline version control consists of keeping your code on your PC or putting it on a separate hard drive/USB Stick; it offers complete control over your code, and you do not need internet access to use or update the files.

# Version Control

## Positives of Online

**Automated Backups:** When saving a file, it automatically gets put into the cloud, ensuring your project's scripts are always up to date.

**Collaboration:** Multiple people at once can simultaneously work on the same project at once

**Global Access:** As long as you have an internet connection, you can keep track of your code

**Tracking:** You can track file history and who did the edit (if working with multiple people) and roll back any issues.

## Negatives of Online

**Internet Access:** if you don't have internet access, these files aren't accessible, and if the internet is slow or unstable, it can affect workflow.

**Security:** The code is dependent on your account not being compromised; for example, if there is a data breach or bug, your code can be leaked; this is even worse if your project is generating monetisation.

**Misuse:** Your code might be written over by someone else's push; this will take a long time to fix

# Version Control

## Positives of Offline

**Internet Access:** you don't need internet access to access your files, making it suitable if you're travelling or there is a power outage - it won't affect your workflow.

**Privacy:** you aren't uploading your code to someone else's servers, letting you not have to worry about data breaches or unauthorised access

**Long-Term Archival:** I can store data indefinitely without worrying about subscription plans, how much storage I have left, etc.

**Faster Access:** As everything is offline, you can access your files relatively quickly

## Negatives of Offline

**Collaboration:** you will have to use USB sticks to transfer files, which will take a very long time and cause conflicts, especially the more people there are

**No Global Access:** You have to transport your USB with you if you want to update any files - which can be tiresome, especially if you forget it

**Data Redundancy:** Keeping track of the most recent versions might get a bit confusing

**Risk of Loss:** You might lose a USB stick, whereas you can't with cloud.



# Version Control

## Which am I Choosing?

I usually use an offline backup, but I have chosen both as this project is essential. However, I will be taking a more staggered approach. Offline backups will be the main focus, but I will use version control for significant changes or additions, as this will give me the security of knowing a USB stick cannot be the sole point of failure. I was given a lovely, large USB stick a few weeks ago that I will use for this project alone.

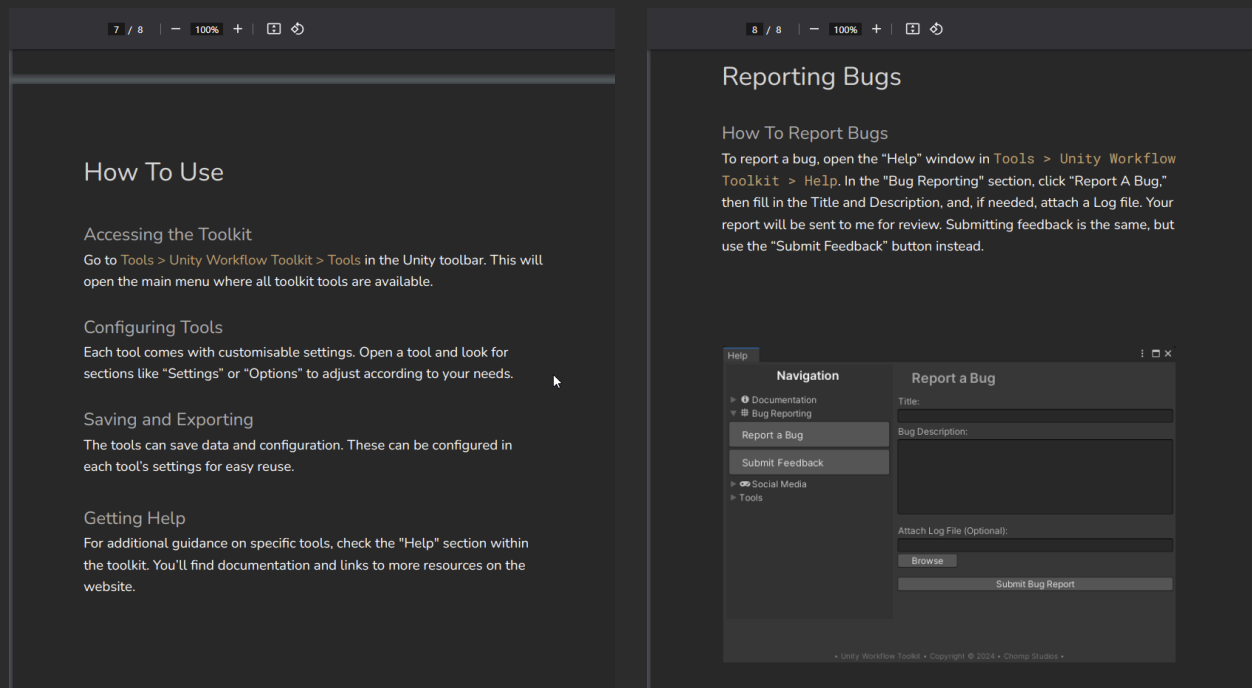
## Why?

I get the benefits (also drawbacks) of both; I can access essential versions anywhere but have my up-to-date versions on a separate USB stick (that is not connected to the pc); this gives me more security than just using Online Version Control, with the peace of mind that my code isn't entirely in the hands of other people's servers.

# Documentation

## Technical Documentation

Technical documentation will be covered here and inside the “Documentation Document” I made in more depth; this refers to how the code works, how and why I have made each bit of code, and why I have done it like that. The documentation document is the document inside of the project files. When people use it, they can read it and understand how to use the product - it is a little barebones right now, but I will improve it constantly.



# Documentation

## Website-Based Documentation

I have made a small website for hosting User-Sided documentation, i.e., how the tools work, A section covering available tools and how they work, how to install the tool to Unity and much more; this approach would be better for the everyday user.

I have purchased a 2-year Domain (wattie.work), and I already have hosting; this will allow me to put the website up for anybody to visit and use.

# Code Conventions

## Brief Description

Code Conventions are fundamental, especially in projects like this; these are primarily **guidelines**, not rules - but they should still be followed as often as possible; these guidelines **help** other programmers or developers should they want to help with this project or even fix a bug in their copy of **Unity Workflow Toolkit**. It is also just good practice to have homogeneous code. That is easy to work with

## Brief Description

C# primarily uses the Resharper or StyleCop code conventions; I will be following ReSharper (the styles, not the software) with a few modifications:

## Naming Rules

- All names should be in English.
- Names should have one meaning, so each variable is unique.
- Brackets should be on the same line (K&R)
- Indentation: 2 Spaces per indentation

```
if (_showSettings) {  
    ShowSettings();  
}
```

```
for (var i = 0; i < _filePaths.Length(); ++i) {  
    File.Move(_filePaths[i], _destinationFileName);  
}
```

# Code Conventions

## Classes & Namespaces

- PascalCase (e.g. `public class PostBuildActions`)

## Variables

- camelCase (e.g. `public bool showBuild`)
- If private, prefixed with an underscore (e.g. `private bool _addFiles`)
- Temporal Variables (only used once or twice) should have short names (`GameManager gm`), whereas long-term variables use a longer name (`private string _exportDirectory`)

## Methods

- PascalCase (e.g. `void DrawHeader(string toolName, string desc)`)
- Private helper methods should be in a separate helper script

## Arrays & Collections

- camelCase and use plural nouns (`List<string> _noteNames`)

## Regions

- Regions are used to separate code (`#region Private Variables`)

## Loops

- Switch loops and for loops have preferences.
- Loops should be negative when using loops that will interact with UI: (`for (var i = _noteNames.Count - 1; i >= 0; i--)`)

# Copyrighting

## Notice

I have decided to include a copyright notice at the top of each script inside of my code (that I have made) - I feel like this is essential as people might not realise that my code is copyrighted, and I do not want it to be redistributed. Here's what it looks like

```
#region Copyright
/*
-----
*   Copyright © 2024 • Chomp Studios • All Rights Reserved
*
-----
*   Author: Wattie King
*
-----
*   Date: 30/10/2024
*
-----
*   Description: Post Build Actions tool code
*
-----
*/
#endregion
```

## Licence

I have decided to use the default Unity EULA License for this project; it prohibits redistribution but allows you to modify the code for your purposes. I feel like this licence is perfect for what I want.

# Version List

## Intentions

These version numbers are being used to track new features, bug fixes or improvements, increasing when these are added. Here is what I Plan to have in each version:

Version no.	Intended addition
v 0.1.0	The first tool (most likely the Script Counter) is to add the UI at the top of unity
v 0.2.0	The second tool (likely Notes) bug fixes any issue with either.
v 0.3.0	The third tool (likely the Asset Deletion tool) fixes any bugs possible, adds logging to the project (including the older tools)
v 0.4.0	Fourth Tool, hopefully, the Unused Asset Remover tool
v 0.5.0	Bug fixes, general code optimisation & refactoring
V 0.6.0	

# Build Delivery Method

## Process

At the end of the night or when specific conditions have been met (like a big update or implementation), I will export the current project to my USB stick as a .unitypackage with a versioned and timestamped name, e.g.

```
Unity Workflow Toolkit (v0.1.0) 11-10-24_21-01.unitypackage
```

I will also export the source code to my USB Stick as a security measure. I could do this automatically.

## Versioning

I will continue to use semantic versioning ([here](#)) (e.g. v0.1.0, v0.5.0, 1.1.4) as this is consistent with other projects and how the tools I have made name the game files. I am using **major.minor.patch** format; a patch will increase the last number, a minor update will improve the middle number, and the major updates (think, release v1.0.0) will update the first number. After going up an integer, I will add a changelog (but this changelog will be worked on throughout the development process)

## Packaging

The packaging will be in a specific format, unitypackage, as this lets you transport all files in a very, very compressed single file. I could do this automatically with something like Unity Build, but this is pricey and not worth it for this sort of project.



# Build Delivery Method

## Quality Assurance

I will test the tools myself until I have gone through the “Testing” phase (go to the Design Document for more information), and then I will get my developers to assist with this.

## Delivery

When in testing, this will be given to friends through cloud services. Still, the release version will only be updated in the marketplace where I sell the project (Unity Asset Store and Itch.io); I am responsible for delivering new builds.

I will aim to have a new version every week that will at least go up 1 version “point” (e.g. v0.1.0 -> v0.2.0)


# Build Delivery Method

## Post-Delivery

I will notify the testers (and “stakeholders”, although there are none right now) when new builds are ready to test and publish the release notes to the website that show additions, removals and bug fixes in a simple four-box scheme - which I call the “4 box key” it makes it easy to identify what is what:

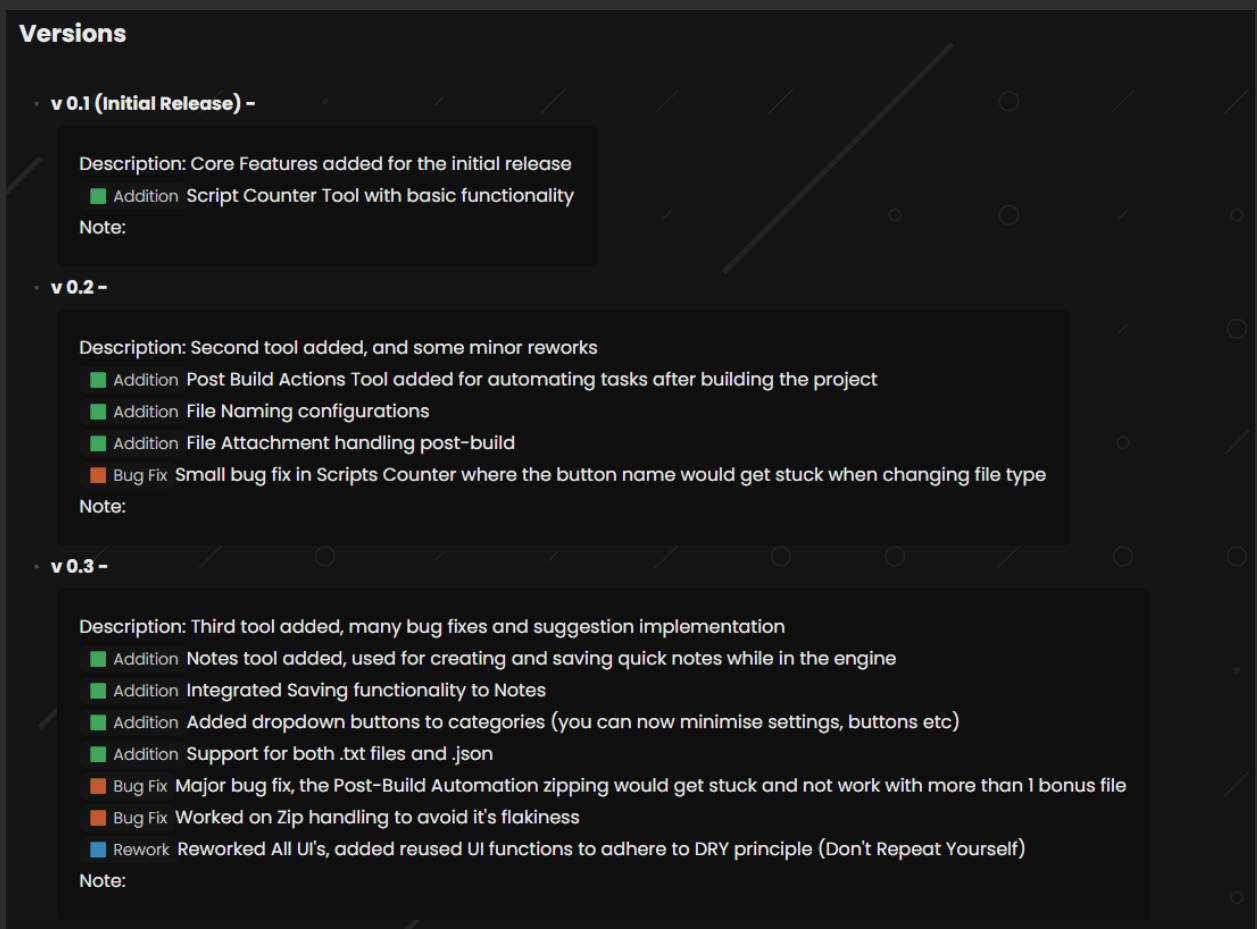
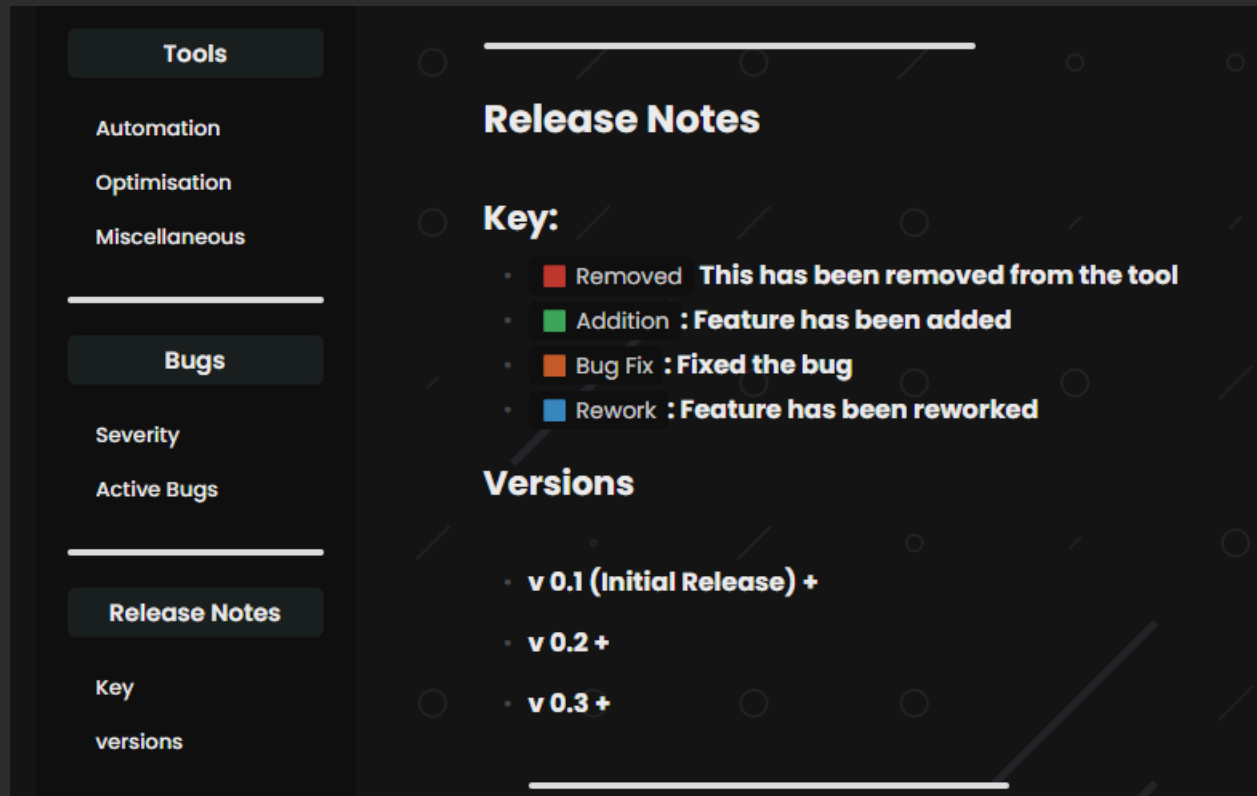
 = Additions

 = Removed

 = Bug Fix

 = Reworked

This has already been implemented on my website, and it is very convenient and easy to understand.



# Logging

## How It Works

Logging is essential to projects, especially this one - it lets users who might have issues upload or send the log file to me so I can look at it and debug it (to fix the bug they are facing). I am using NLog and a customised “front” for it. Important debug information will be sent to a file inside of the user’s Game files; the file path is “[EditorLogs/Workflow Toolkit/uwt.log](#)”, and it can be sent to my Discord Support server that will be made before Release v1.0.0.

## What it looks like

```
2024-10-12 21:54:05.7030 | INFO | [LOGGER] Post Build Actions Tool
Launched | |
Editor.Workflow_Toolkit.Utilities.LoggingElements.LogInfo(LoggingElements.
cs:54)
2024-10-12 21:54:22.3958 | INFO | [LOGGER] Building Game Now, then moving
to zipping. | |
Editor.Workflow_Toolkit.Utilities.LoggingElements.LogInfo(LoggingElements.
cs:54)
2024-10-12 21:54:25.4985 | INFO | [LOGGER] Zipped Requested Files... | |
Editor.Workflow_Toolkit.Utilities.LoggingElements.LogInfo(LoggingElements.
cs:54)
```

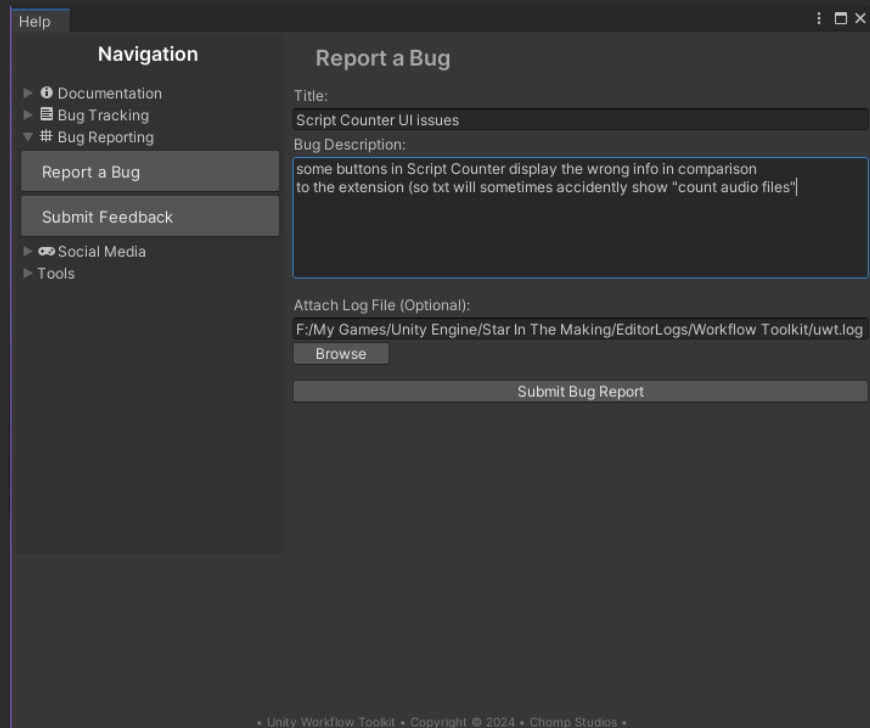
## Two Types

There are two types of logging for this project: one that shows a notification inside of the editor (in the console) and the NLog Logger above; they have different uses - the console log will show less critical information, telling the user to open up the log file to view more critical bugs, crashes etc.

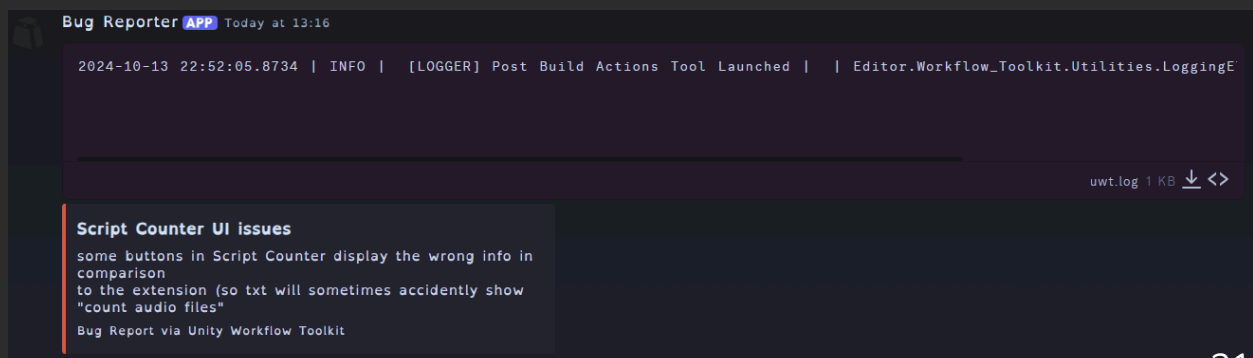
# Reporting Bugs

## How?

I currently have a separate “help” window for users who need help setting up and accessing the tools; it has a bit dropdown called “Report A Bug” You can then attach a title, a description of the bug and an optional log file, this gets sent to a private channel in my discord server. I would prefer a self-hosted server to upload the log files as this is much safer and secure (as somebody can “spam” the webhook), but this is too big for the current scope.



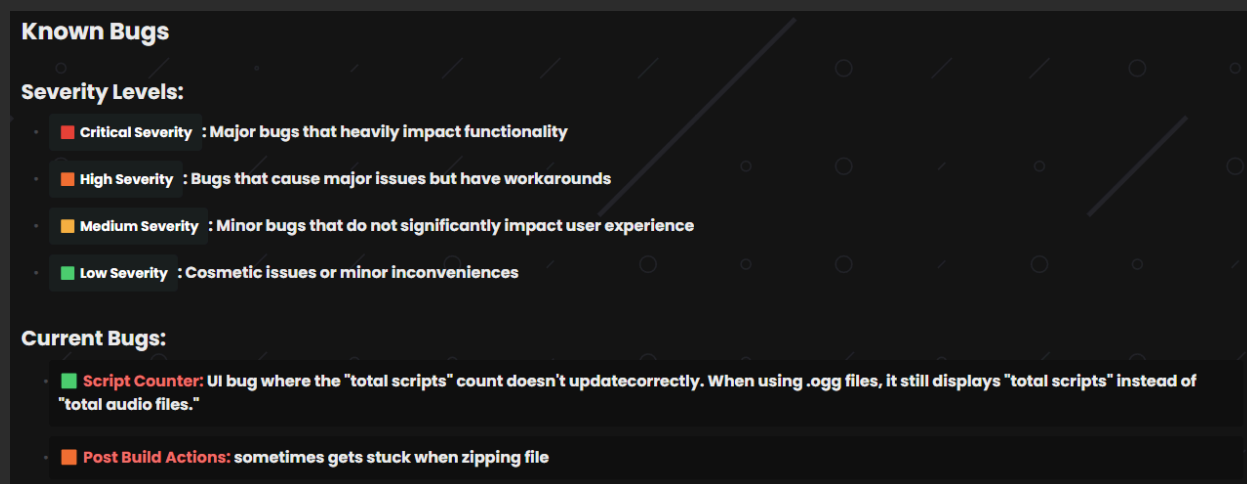
The screenshot shows a 'Help' window with a 'Report a Bug' form. The form has a sidebar with navigation links: Documentation, Bug Tracking, Bug Reporting (selected), Report a Bug, Submit Feedback, Social Media, and Tools. The main form area has fields for Title (Script Counter UI issues), Bug Description (some buttons in Script Counter display the wrong info in comparison to the extension (so txt will sometimes accidentally show "count audio files")), Attach Log File (Optional) (F:/My Games/Unity Engine/Star In The Making/EditorLogs/Workflow Toolkit/uwt.log), and a Submit Bug Report button. The footer of the window reads: Unity Workflow Toolkit • Copyright © 2024 • Champ Studios •



# Reporting Bugs

## What Happens Next?

If it's deemed a bug that needs to be fixed, it will be added to the bug tracker on the Unity Workflow Toolkit website; it will then be selected for severity and fixed when possible (the severity dramatically increases the speed at which this happens).



## When will it get fixed?

If the bug is critical or high, a patch will be released (thus incrementing the final number) to fix the bug, and any medium- or Low-severity bugs will be fixed in the next minor or major update.

# Libraries

## Newtonsoft

Newtonsoft ([version 13.0.3](#)) is a JSON serializer used in this project for... serialising JSON in tools like Notes. Find it [here](#)

## NLog

I am using NLog ([version 5.3.3](#)) to log bugs and issues. It is much better than the default logger, as I can customise it. Find it [here](#)

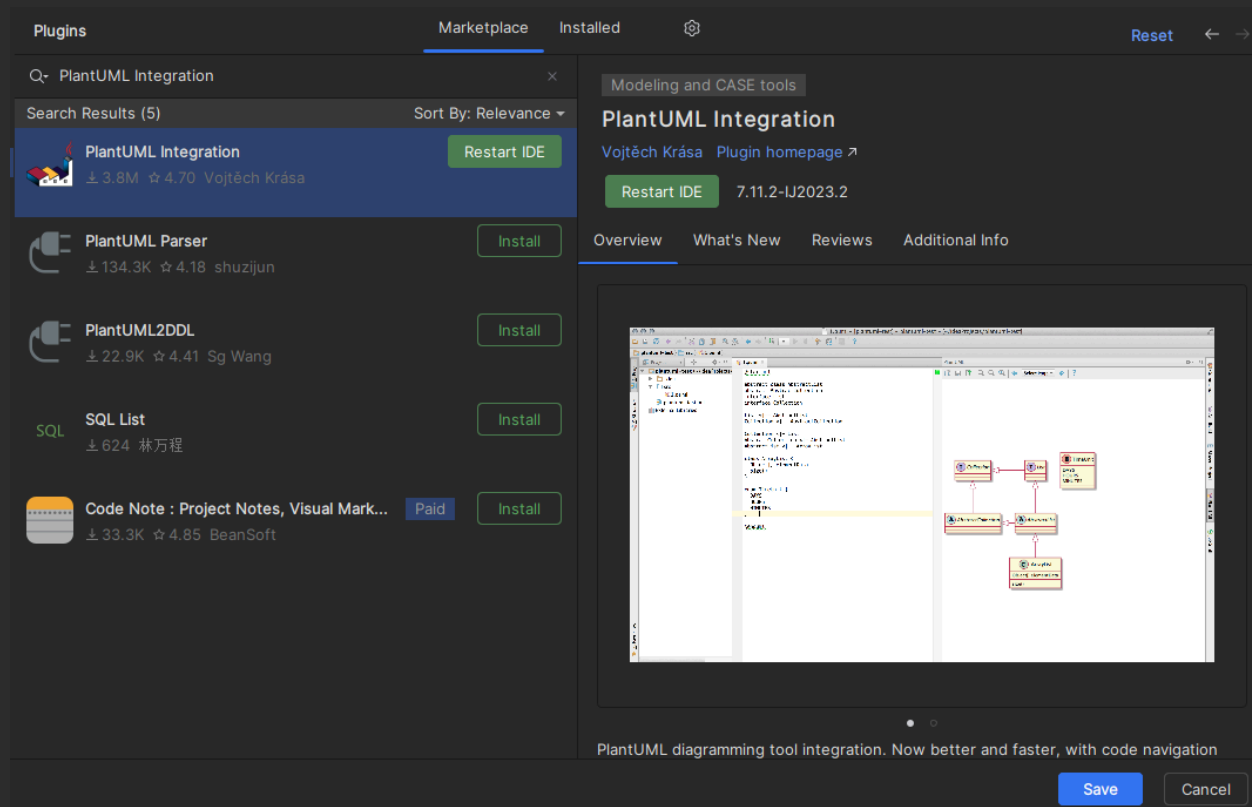
## Licences

Name	Licence Type
NLog	BSD
NewtonSoft.Json	MIT

# UML Structure

## Software Used

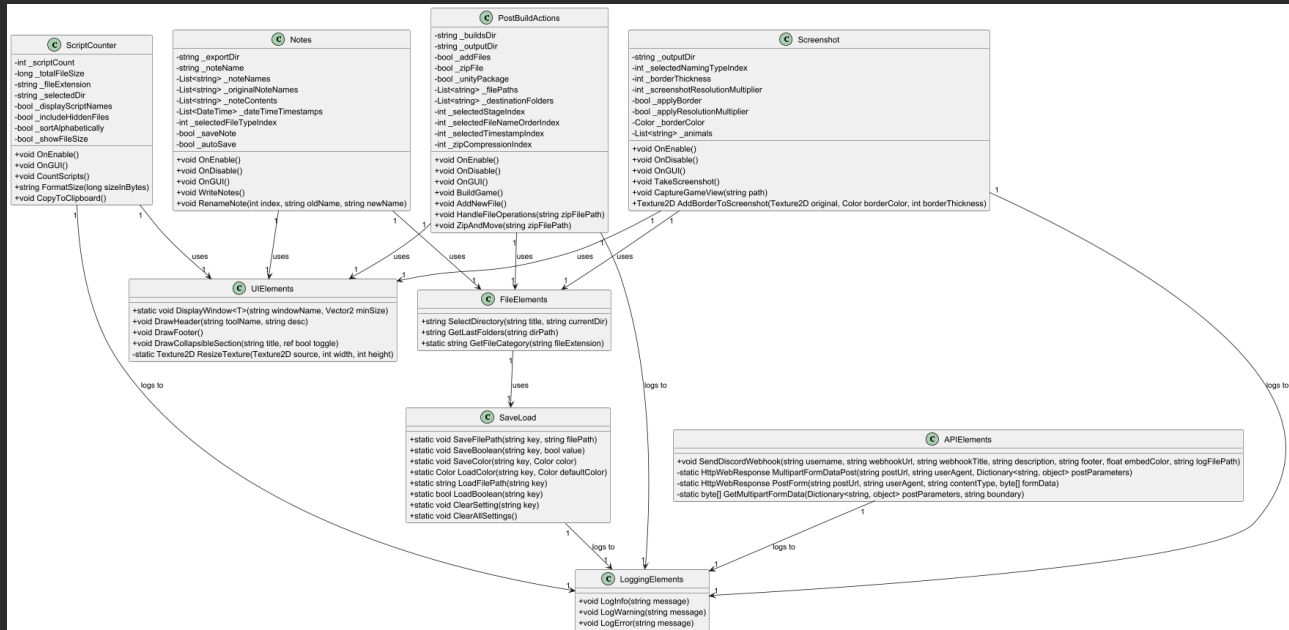
I am using a Rider-Integrated PlantUML plugin for my UML structure; it is very efficient and not challenging to get these images (which I like). This software generates diagrams based on the code I write. It shows how different classes, objects, and strings are connected, giving me a charming visual demonstration. I will be using this to help you understand my code better.



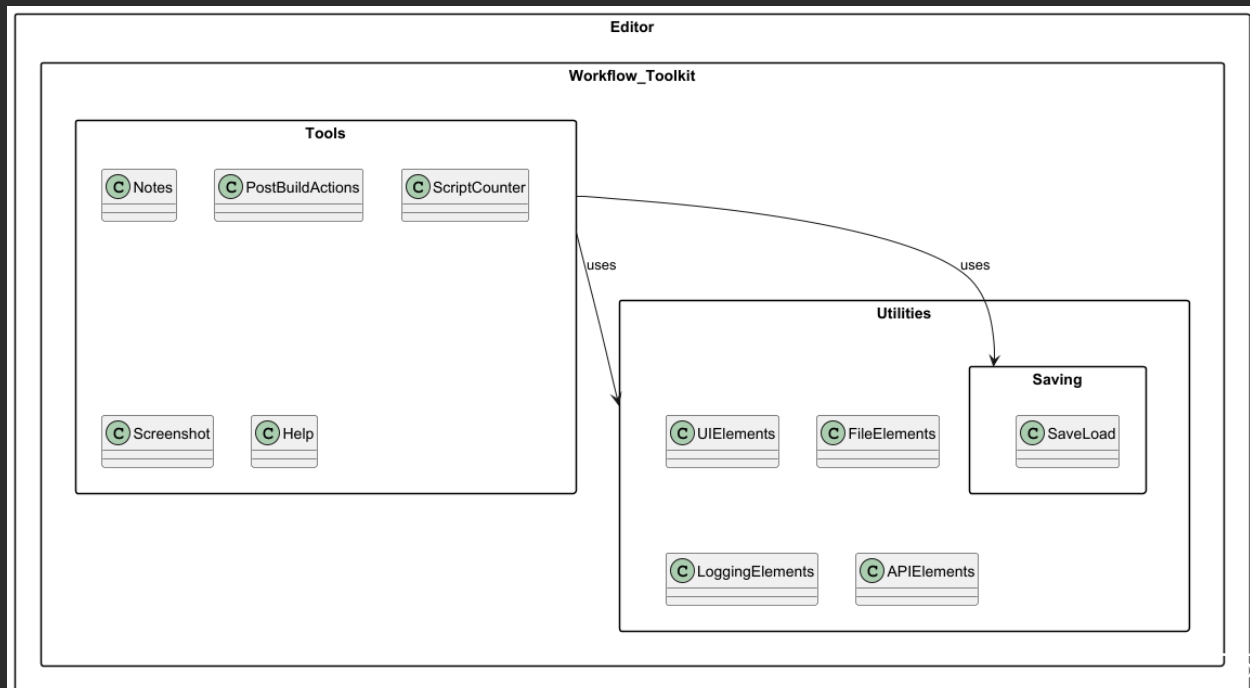


# UML Structure

## Classes

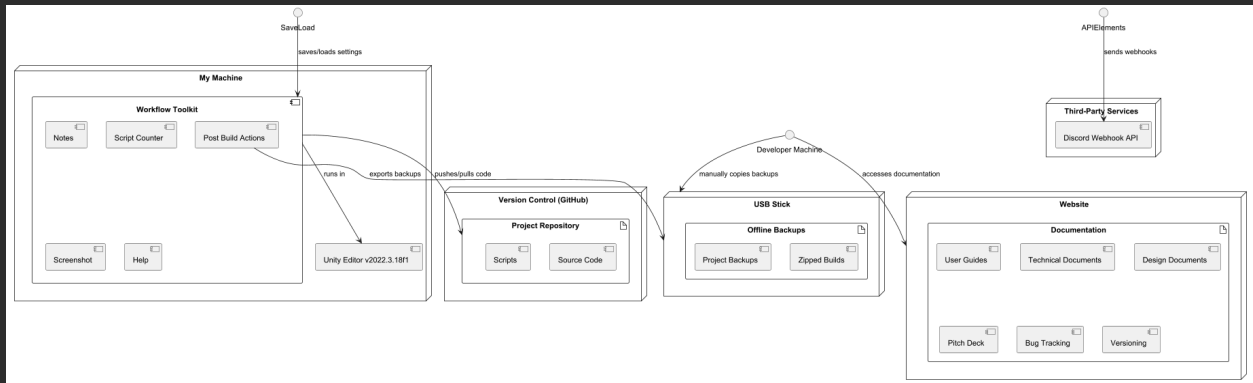


## Packages



# UML Structure

## Deployment



# Save System

## Explanation

I have programmed a saving system, so when you're using the tool (e.g. Post Build Actions), you won't have to attach the same things over and over again - I feel like this works very well - it is my own "front" for Unity's inbuilt "PlayerPrefs" - I would usually use a complete version of my JSON save load. Still, I felt that due to the scope, it would be more effective (and better for different operating systems) if I used the built-in unity one. The file that interacts with saving is called SaveLoad; it houses multiple functions that let me save and load the settings (file paths, booleans, strings, integers and colours).

## How do you save a value?

When you want to save a value, you first call the respective function (Booleans would be `SaveLoad.SaveBoolean(string key, bool value)`, Colour would be `SaveLoad.SaveColor(string key, Color colour)`) inside of the Default Unity's `OnDisable()` method with the necessary information - for example, to save the `Builds Directory` and `File Output Directory` your on `OnDisable()` The process would look like this:

```
private void OnDisable() {  
    Utilities.Saving.SaveLoad.SaveFilePath("PBA_BuildsDir", _buildsDir);  
    Utilities.Saving.SaveLoad.SaveFilePath("PBA_OutputDir", _outputDir);  
  
    PlayerPrefs.Save();  
}
```

# Save System

## How do you Load a value?

Using the same example, to load the `Builds Directory` and `File Output Directory` you had previously saved using `OnDisable()`, you need to make an `OnEnable()` Method or add the following to a preexisting one you already have - you call the `Utilities.Saving.SaveLoad.LoadFilePath()` Method to retrieve the File Path variable and then assigning it to a value, it will look something like this:

```
private void OnEnable() {  
    _buildsDir = Utilities.Saving.SaveLoad.LoadFilePath("PBA_BuildsDir", null);  
    _outputDir = Utilities.Saving.SaveLoad.LoadFilePath("PBA_OutputDir", null);  
}
```

This works the same if you're saving colours, integers, strings or any other variable - if the Loading Function requires a second argument to access (e.g. `LoadFilePath("PBA_BuildsDir", null)`). The Second argument is the default value.

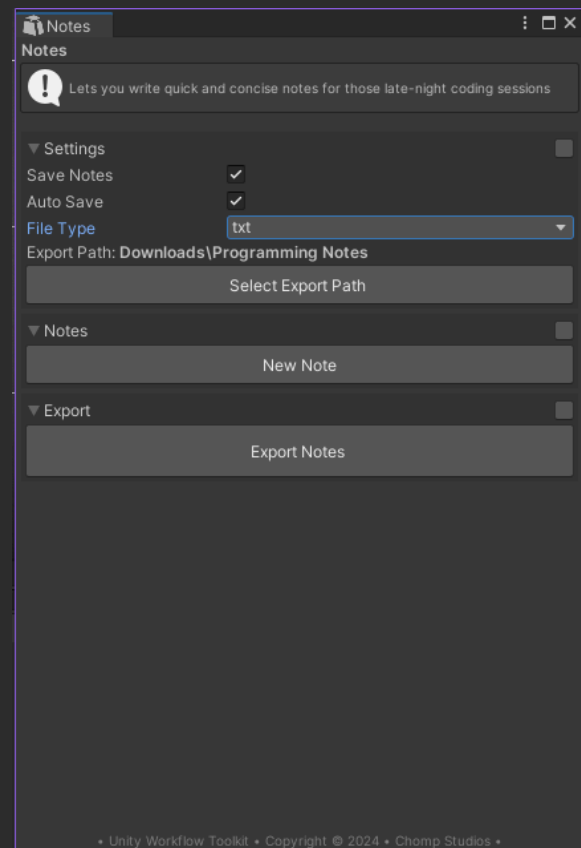
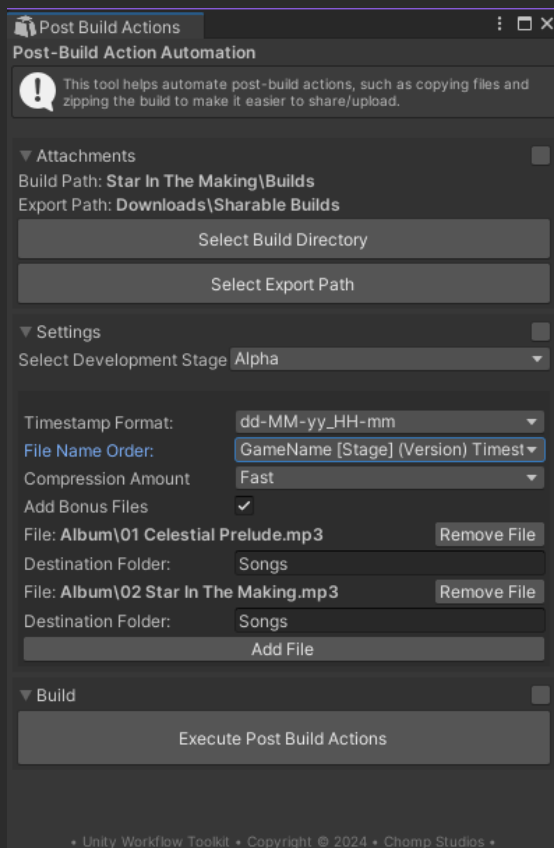
## General Rules

When saving a **key** (e.g. `"PBA_BuildsDir"`), Ensure that the Tool's name is prefixed with an underscore, and then the variable's name is used - this ensures clarity.

# Save System

## Saving In Action

Once following the instructions on the last page, the tools now look like this when opening them, and it works properly and correctly:



## Clearing Settings

To clear your settings, I have made it so you can clear them in the “Help” Window - this is because you might set up a different project or just want something different, and this ensures all are clear - due to this being on top of a unity project, I will individually delete each key, rather than just calling `PlayerPrefs.DeleteAll();`

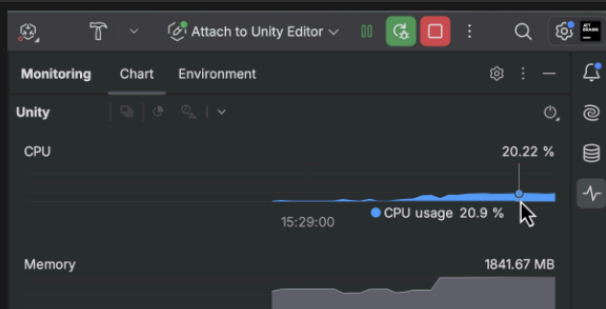
# Performance Budgets

## Profiling

To see how long my Unity tools take to load and work, I have used the built-in Unity Profiler, which lets me see different performance data in charts. While I'm not experienced in this sort of thing, I still like to look at how well or poorly optimised my code is and see where it can be improved.

## How?

While writing this, there was an update to the IDE I use that adds profiling help inside of the code editor itself; it adds support for seeing how much RAM each script in your code uses, so I will be using this to make sure the most minimum amount is being used.



### Unity Game development

Rider 2024.2 adds support for the Unity editor to the recently introduced *Monitoring* tool window, so you will automatically see CPU and memory usage when you attach the debugger to the Unity editor. We've also made some nice updates to shader support, with shader variant keywords now working with compute shaders, and the pass name shown in the shader context switcher.

[Read more ↗](#)

# Challenges

## Explanation

I have faced several Technical Challenges during this project. Some Tools I thought would be easier than others weren't and some projects that I thought might be harder were more straightforward - due to scope and time remaining, I had to cut this project a little short - with four total scripts (But I will continue to develop this project as time goes on; [Post Build Actions](#), [Script Counter](#), [Fancy Screenshots](#) & [Notes](#)). I wanted to make an Unused Asset Remover, but I found this incredibly difficult due to Unity not keeping track of scripts the user has created; it would take much more effort (thus increasing and going past my Scope) to include this now.

# Class Reference

Current Classes: 15

Class Name	Purpose	Type
PostBuildActions.cs	Post Build Actions tool code automates long and annoying builds.	Tool
ScriptsCounter.cs	Scripts Counter tool code counts all kinds of assets in your project.	Tool
Notes.cs	Make notes inside of Unity for late-night coding sessions	Tool
Screenshot.cs	Tool for screenshotting your game easily	Tool
Help.cs	Popup to help the user understand how to use the tool	Tool
FileElements.cs	Controls File Handling functions	Utility
UIElements.cs	Controls UI functions, like drawing headers and footers.	Utility
LoggingElements.cs	Allows customizability of the NLog Library, which houses all things logging.	Utility
APIElements.cs	Helper function for Third-Party APIs	Utility
SaveLoad.cs	Saving and Loading your desired settings	Saving



# Class Reference

Class Name	Purpose	Type
ToolName.cs		Enum
BuildOptionsBuilder.cs		Utility
BuildOptionSettings.cs		Utility
CompressionType.cs		Enum
TooltipElements.cs		Utility

# Build Log

Date	Vers	Changes Made
01/10/24	v 0.1.0	Core Features added for the initial release
03/10/24	v 0.2.0	Second tool added, and some minor reworks.
6/10/24	v 0.3.0	Third tool added, bug fixes and suggestion implementation.
12/10/24	v 0.4.0	Logging and optimisation.
20/10/24	v 0.5.0	Saving and Bug Fixes
31/10/24	v 0.6.0	addition of documentation and reporting
03/11/24	v 0.7.0	Logging and optimisation.
11/11/24	v 0.8.0	Bug Fixing & Feedback Implementation

# Bibliography

# Credits

## Suprep

Lead DevOps Software Engineer gave me tips & tricks for writing a good Technical Design Document and suggested what to add.

## Elise

Redrew the Logo for this tool with proper measurements.

## Jack Hogg - Tester 1

Giving me very influential feedback that has catapulted this project on to another level