LiveWatch

Debug with full history of changes

https://livewatch.pages.dev/

by JustIngvar

Table of contents

1. Home

1.1 What is LiveWatch?

LiveWatch is a powerful tool for monitoring variables in real time in *Unity*, with ***full*** history of changes.

Get it on the Asset Store

1.2 With LiveWatch, you can:

Track variables with full history

Monitor all changes to your variables, ensuring you can review the entire history of each variable's state over time.

Monitor any type of data

Track any variable type whether it's a built-in Unity type, a collection (`List` , `Dictionary` , etc.), or a custom type.

Flexible display modes

Switch between **Graph mode** for high-level visual overviews, or use **Cell mode** for a more detailed, table-like view.

Advanced search

Dive into the entire history of values using complex search queries connected by boolean logic.

Conditional formatting

Customize value appearances using color formats, including conditional rules based on the values themselves.

Save/Load functionality

Store your variable data as binary files and inspect the recorded values on different devices or projects.

1.3 What is LiveWatch Lite?

LiveWatch Lite is the free basic version of **LiveWatch**, offering limited features compared to the **Pro** (Full) version.

Get it on the Asset Store

Feature	Lite	Pro
Full history of changes	✓	✓
Basic variable types	✓	✓
Cell/Graph modes	✓	✓
Custom types	✗	✓
Advanced search	✗	✓
Conditional formatting	✗	✓
Extra text	✗	✓
Stack traces	✗	✓
Custom buttons	✗	✓
Save/Load to binary	✗	✓
Private members	✗	WIP
Remote debugging	✗	WIP

Not sure where to begin? Jump into the next chapter

Getting Started

2. Getting started

2.1 Installation

If you upgrade from Lite to Full version, make sure to delete *LiveWatchLite* folder.

2.1.1 **Check for dependencies**

Ensure your project is running `Unity 2021.3` or a later version for compatibility. While the tool might work with earlier versions, functionality cannot be guaranteed.

2.1.2 **Download and Import the Package**

Open the Unity Asset Store page for **LiveWatch** and click `Open in Unity` to access it via the `Package Manager`. Or you can manually open the `Window > Package Manager` menu in Unity, search for the asset, and `Download` it from there. Once the asset is downloaded, click `Import` to bring it into your project.

2.1.3 **Initial Setup**

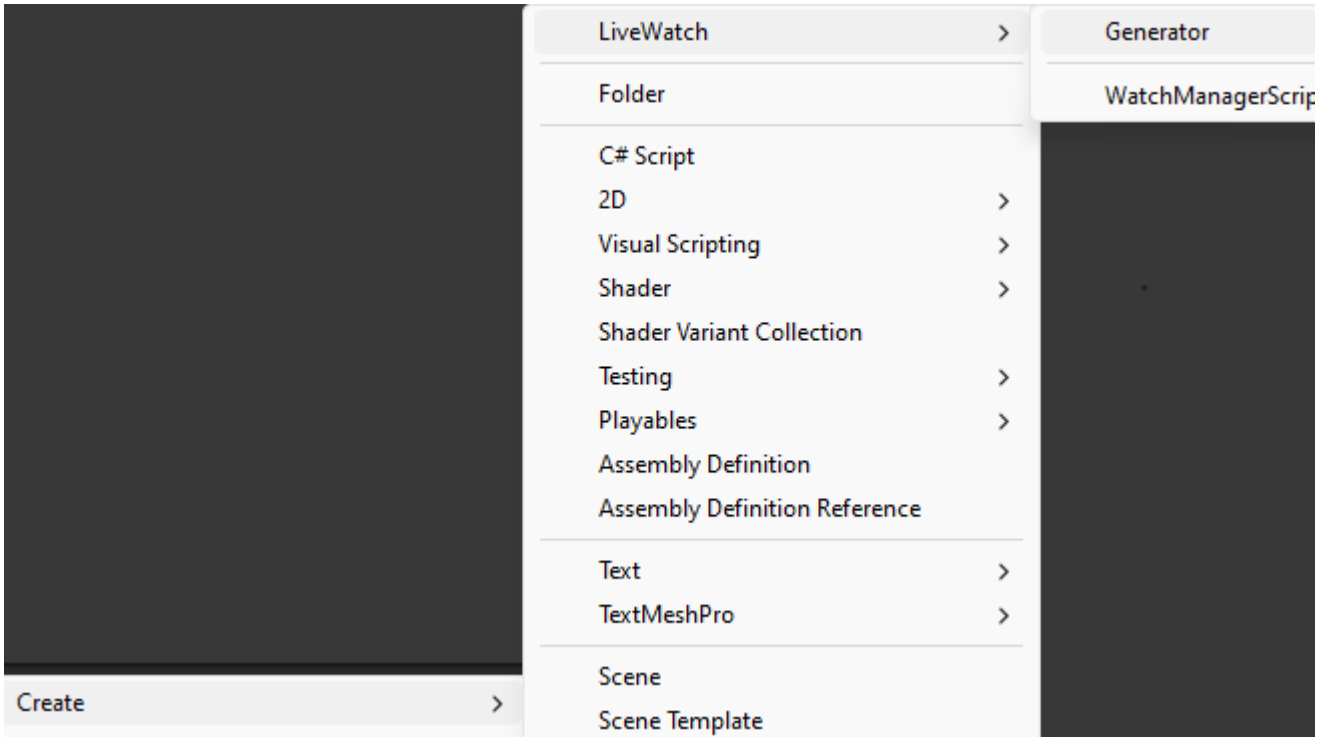
Once imported, the setup is simple. For basic functionality, you can jump right into the Quick Start guide and get started with watching variables.

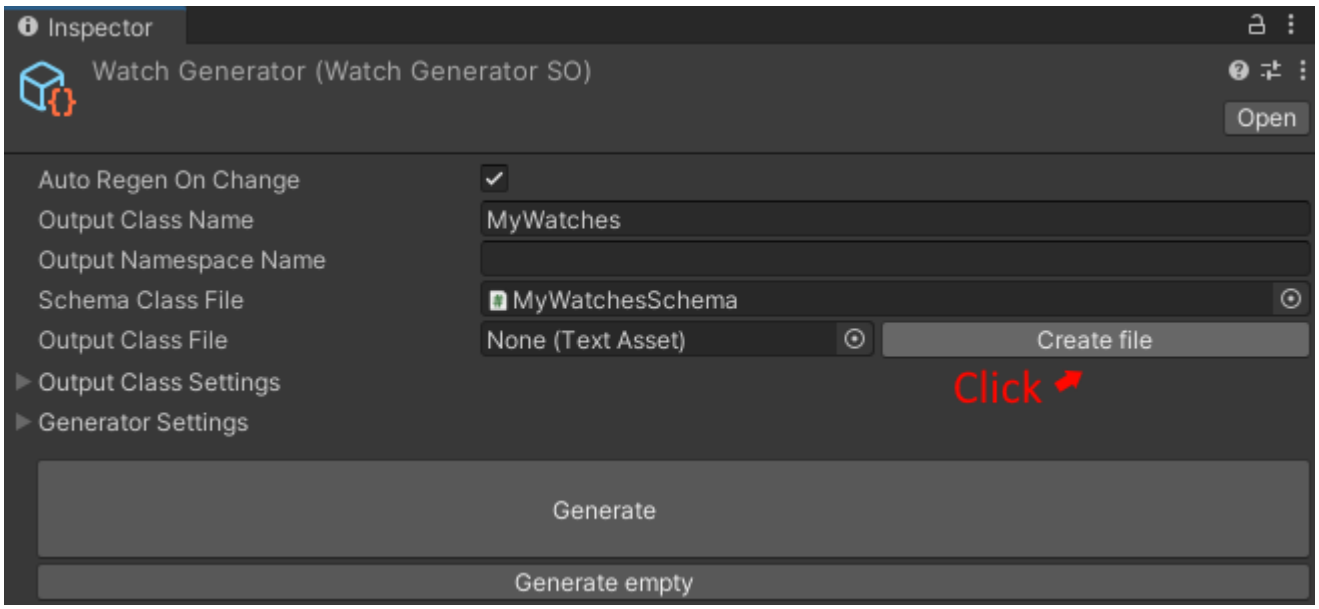
2.2 Quick Start

If you only plan to watch basic type variables (e.g., `string`, `bool`, numeric types like `float` or `int`) or using Lite version, you can skip the first two steps and go directly to Step 3.

2.2.1 **Step 1**: Set up Generator PRO

- In the Project Window, go to **Create > LiveWatch > Generator**.
- Enter a name for the watch class (e.g., `MyWatch`) and optionally specify a namespace.
- Click the **Create File** button in the `Schema Class File` field, and then in the `Output Class File` field to generate the required files.





2.2.2 **Step 2**: Define the variables in Schema

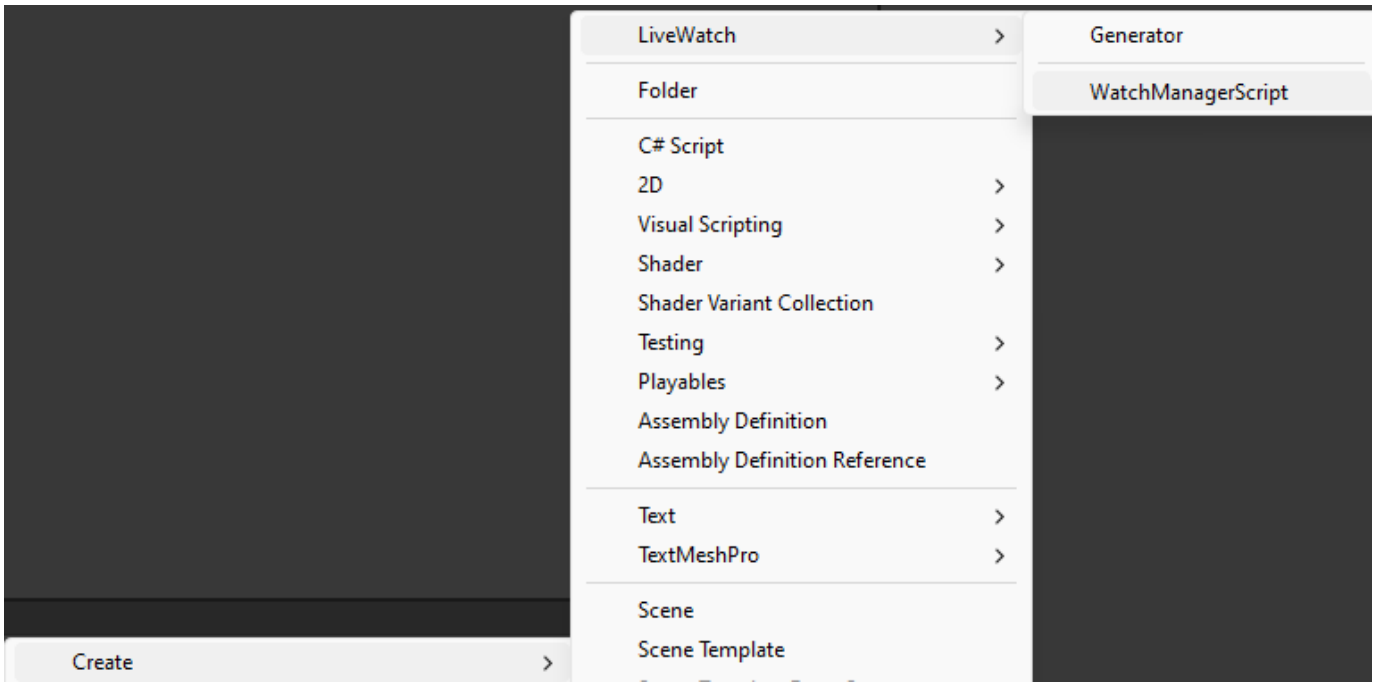
- Open the generated schema class script.
- Inside the `OnGenerate` method, list the variables you want to watch. For base types you don't need to generate them as they are pre-generated by default. For custom types, use the `Generate<T>()` command, where `T` is your custom type (e.g., `Generate<MyCustomType>()`).

```
using ...

namespace Ingvar.LiveWatch.TowerDefenceDemo
{
    public class TD_WatchesSchema : WatchGenerationSchema
    {
        public override void OnGenerate()
        {
            Generate<LevelStateType>();
            Generate<List<MobMain>>();
            Generate<LevelConfig>();
        }
    }
}
```

2.2.3 **Step 3**: Create WatchManager Script

- In the Project Window, go to **Create > LiveWatch > WatchManagerScript**.
- Open the generated script and in the Start() method, add your watches using the following syntax:
  `{YourWatchClassName}.AddOrGet("YourVariableName", () => yourVariableValue);`
  You can place this line anywhere in your project.




```cs
using ...

public class WatchManager : MonoBehaviour
{
    private void Awake()
    {
        Watch.DestroyAll();
    }

    private void Start()
    {
        Watch.GetOrAdd(path: "Frame", valueGetter: () => Time.frameCount).SetAlwaysCollapsable();
        Watch.Push(path: "Log", value: "Hello World!");
    }

    private void LateUpdate()
    {
        Watch.UpdateAll();
    }
}
```

What is `SetAlwaysCollapsable()` ?

`SetAlwaysCollapsable()` hides less important value columns in **Collapse** mode. This will keep your watch window cleaner by hiding frequent or non-essential updates. These values will still be recorded and can be viewed by toggling the **Collapse** button in the watch window.

2.2.4 **Step 4**: Attach WatchManager and ***Watch*** the magic

- Attach WatchManager script to any GameObject in your scene.

• Enter **Play Mode**, and you'll be able to monitor your variables in **Window > LiveWatch**. More about Watch Window

---

Want to Learn More?

If you want to learn more about how to use this tool in a real project, take a look at the `TowerDefenceDemo` provided with this asset. This demo includes practical examples and showcases how you can use **LiveWatch** to track variables and debug in more complex scenarios.

Next chapter: Learning Basics

2.3 Learning Basics

To get a better understanding of how this tool works, let's start with the basic concepts involved:

2.3.1 Watch Window

This is the main interface for interacting with your watched variables. It allows you to inspect variables of any type in real-time using *Cell* or *Graph* mode. You can also search through variable histories, and *Save* or *Load* recorded data for later analysis. The **Watch Window** is your central hub for tracking and managing variable changes efficiently.

More about Watch Window

2.3.2 Watch

The main class used to create, destroy, and update variables in your project. It also provides methods for tracking *basic* variable types, including both *auto* and *manual* tracking modes.

Basic and custom variable types

You can watch both basic types and your own custom types PRO:

- Tracking variables of **basic types** requires no setup — these are readily available through the `Watch` class methods. The basic types supported are: `float`, `double`, `int`, `string`, `bool`, `char`, `decimal`, `long`, `short`, `byte`, `ulong`, `ushort`, and `sbyte`.
- If you need to track **custom types** PRO, you must first generate them using *WatchSchema* and *WatchGenerator*. The generated class will contain the same tracking methods as the `Watch` class, but tailored for your custom types.

Manual and auto tracking mode

There are two ways to expose variables to the Watch window:

- **Auto Mode**: Call `GetOrAdd` with a getter for your variable, and it will be updated automatically on each `UpdateAll` call.
- **Manual Mode**: Explicitly `Push` variable values when needed. This gives you more control but requires careful use to avoid visual inconsistencies.

More about Watch API

2.3.3 WatchReference

An entry point for any operation with a specific watched variable. This is returned from every watch method in the `Watch` class or from your custom-generated class. You'll use it for tasks like pushing new values, adding formatting, or changing priority — everything related to managing a watched variable is done through this entity.

More about WatchReference API

2.3.4 WatchSchema PRO

The base class for every code generator schema. To create your own watchable types, you need to derive from this class, then use the `Generate<T>()` method to add new watchable types. Finally, reference this class file in the **WatchGenerator**.

More about WatchSchema API

2.3.5 **WatchGenerator** 

A `Scriptable Object` designed to simplify the code generation for new watchable types. You define a name for the generated class, provide a reference to the `WatchSchema` where the watchable types are described, and set a reference to the output file. The generated class will contain methods similar to the `Watch` class. By default, the generation process is automatic and triggered by any changes to the linked `WatchSchema` class.

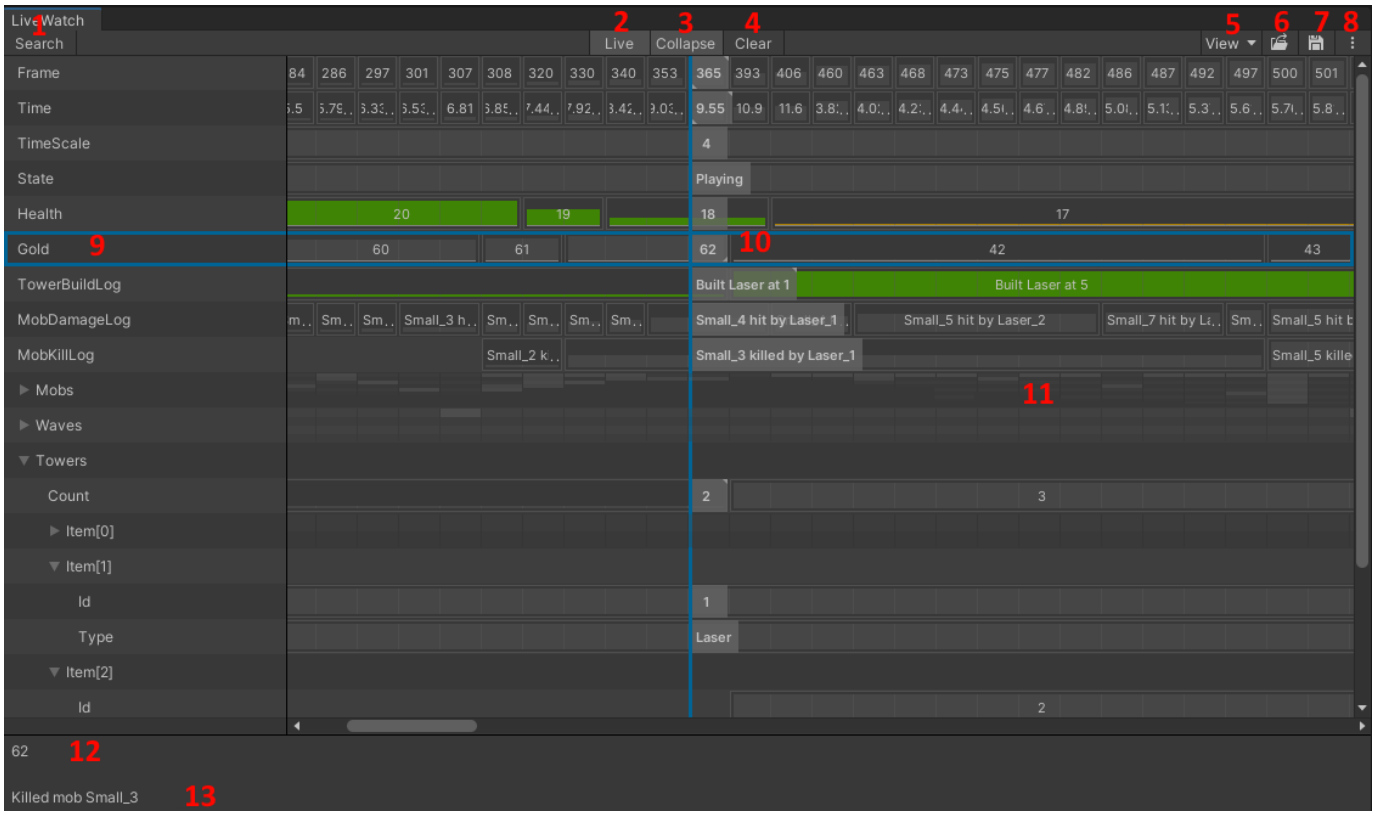
More about WatchGenerator

3. Manual

3.1 Window Overview

3.1.1 **Watch Window**

This is the main window of the tool, where most of its features are accessible. You can find it under **Window/LiveWatch**.
Below are descriptions of the key components:



1. *Search* PRO

Toggles the visibility of the Search Panel.

2. *Live*

Enables or disables data recording, including manual updates. *Enabled by default*.

3. *Collapse*

Hides columns with unchanged values, showing only columns with unique values. *Enabled by default*.

> **How to make variable values Collapsable?**
>
> Use `SetAlwaysCollapsable()` method.

4. *Clear*

Deletes all recorded data for the variables. The variables remain, but all data is lost. Be cautious when using this, as there is no confirmation prompt.

5. *View*

Submenu for adjusting data visualization settings.

6. *Load* PRO

Loads watch data from a previously saved binary file.

7.*Save* PRO

Saves the current data as a binary file to a selected path.

8. *Preferences*

Submenu with various useful commands.

9. *Variable Name*

Displays the name of the watched variable. If the variable has child members (i.e., it's not a basic type), there will be a foldout button on the left of the label.

> **Info**
>
> Note that the column is *resizable* by dragging with the mouse.

10. *Variable Values*

Values are organized into cell segments, where identical values are merged into one long cell. Each cell contains a progress bar that compares the variable's value to others visible on the screen.
Clicking anywhere in the value area will bring up a selection column showing all values from the same iteration. If the selected cell has unique values to the left or right, small guiding triangles appear, indicating whether the neighboring value is larger or smaller (top or bottom triangle).

11. *Child Variables Preview*

If a variable has child members, its values are displayed in a compressed view regardless of their count or depth. Each line in the preview cell represents a specific child ordered from top to bottom. A bright rectangle indicates a changed value in that column. Search results are also shown here.

> **Why does the preview take time to appear?**
>
> This calculation runs in the background on a separate thread to avoid slowing down the interface, but processing time may increase depending on the number and depth of the child variables.

12. *Info Area*

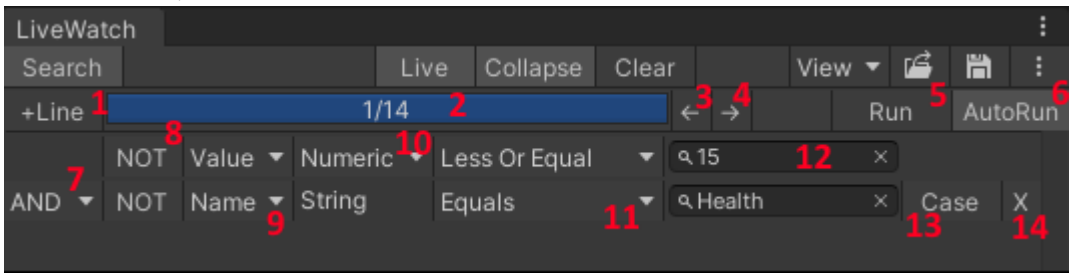
Depending on what is selected, this area will show either the full name or the value of the specific variable, regardless of its length.

13. *Extra Text* PRO

You can attach additional information to each pushed value. For example, in the screenshot, it shows the source of the gold change.

3.1.2 **Search Panel** PRO

The *Search Panel* functionality is more advanced than in a typical Console, allowing for the use of multiple queries linked by boolean operators. Additionally, similar to other areas, the *Search Panel* is resizable.



1. *New Line*

Adds a new query line.

2. *Progress Bar*

Displays the current search progress along with a counter.

3. *Previous Result*

Navigates to the previous search result

> **Why Collapse mode disables after click?**
>
> Jumping to a different search value will disable the **Collapse** feature to prevent skipping potentially collapsed values.

4. *Next Result*

Similar to the previous function, this navigates to the next result. The jump order is *left* to *right* for values, and then from *top* to *bottom* for variables, starting with variable labels if there are such queries.

5. *Run*

Triggers the search. Processed in a separate thread.

6. *AutoRun*

When enabled, the search restarts every time the queries are modified.

7. *Query Connective Operator*

Defines the boolean operator, either `OR` or `AND`.

8. *Inverse*

Inverts the affected query line (e.g., a query of `<= 15` will become `> 15` after inversion).

9. *Target*

Specifies the target for the query line, which can be either a variable *Name* or *Value*.

10. *Target Type*

For variable names, it's always *String*. For values, it includes: *String* (`string` and `char`), *Bool*, *Decimal* (`float` , `double`), *Integer* (`int` , `short` , etc.), and *Numeric* (for both *Decimal* and *Integer*).

11. *Query Line Operator*

Determines the query operator based on target types. For *String*, the options are *Equals* and *Contains*. For numerical values, it includes *Equals*, *Greater*, *GreaterOrEqual*, *Less*, and *LessOrEqual*.

12. *Query*

Represents the query text itself.

13. *Case*

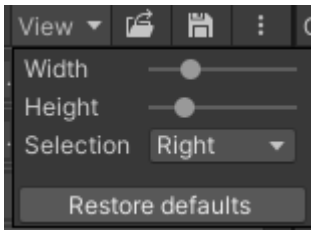
Toggles case sensitivity for *String* queries. For example, after enabling, the previously *positive* query `'Health' Equals 'health'` will become *negative*.

14. *Delete*

Removes the query line. Note that the first line cannot be deleted.

3.1.3 *Cell View Submenu*

This submenu allows you to customize the appearance of the cells.



1. *Width*

Slider for adjusting the width of the cells. If the width drops below a certain level, the window will switch to `Graph` mode.

2. *Height*

Slider for adjusting the height of the cells.

3. *Selection*

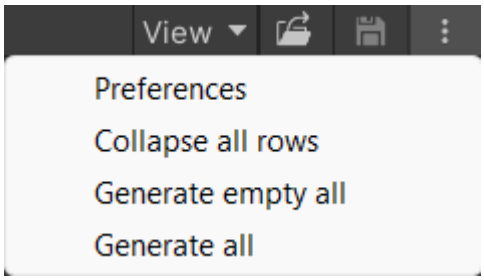
Option for selecting the direction of the selection column: `Right` or `Left` .

4. *Restore Defaults*

Resets the above settings to their default values.

3.1.4 *Preferences Submenu*

This submenu provides access to various settings related to **LiveWatch**.



1. *Preferences*

Link to the Editor `Preferences` Window, where you can adjust **LiveWatch** settings and parameters.

2. *Collapse all rows*

Collapses all variable rows recursively, folding all child elements within each row for a cleaner and more organized view.

3. *Generate empty all* PRO

Calls *Generate empty* on every `Generator` scriptable object present in the project.

4. *Generate all* PRO

Calls *Generate* on every `Generator` scriptable object present in the project.

Next chapter: Window Shortcuts

3.2 Window Shortcuts

This section lists all shortcuts that can be used in the Watch Window. Most of them can be changed in Preferences.

1. `^ Ctrl` + `Mouse Scroll`

   Changes the width of value cells.

2. `⇧ Shift` + `Mouse Scroll`

   Adjusts the height of variables.

3. `⎇ Alt` + `Mouse Scroll`

   Scrolls through values horizontally (*left*/*right*).

4. `Mouse Scroll`

   Scrolls through variables vertically (*up*/*down*).

5. `F`

   Switches the selection direction to *right*/*left*.

6. `Space`

   Expands the currently selected variable (if it has children).

7. `↓ Down`

   Moves to the next variable (below the current one).

8. `↑ Up`

   Moves to the previous variable (above the current one).

9. `← Left`

   Moves to the previous value (to the left of the current one).

10. `→ Right`

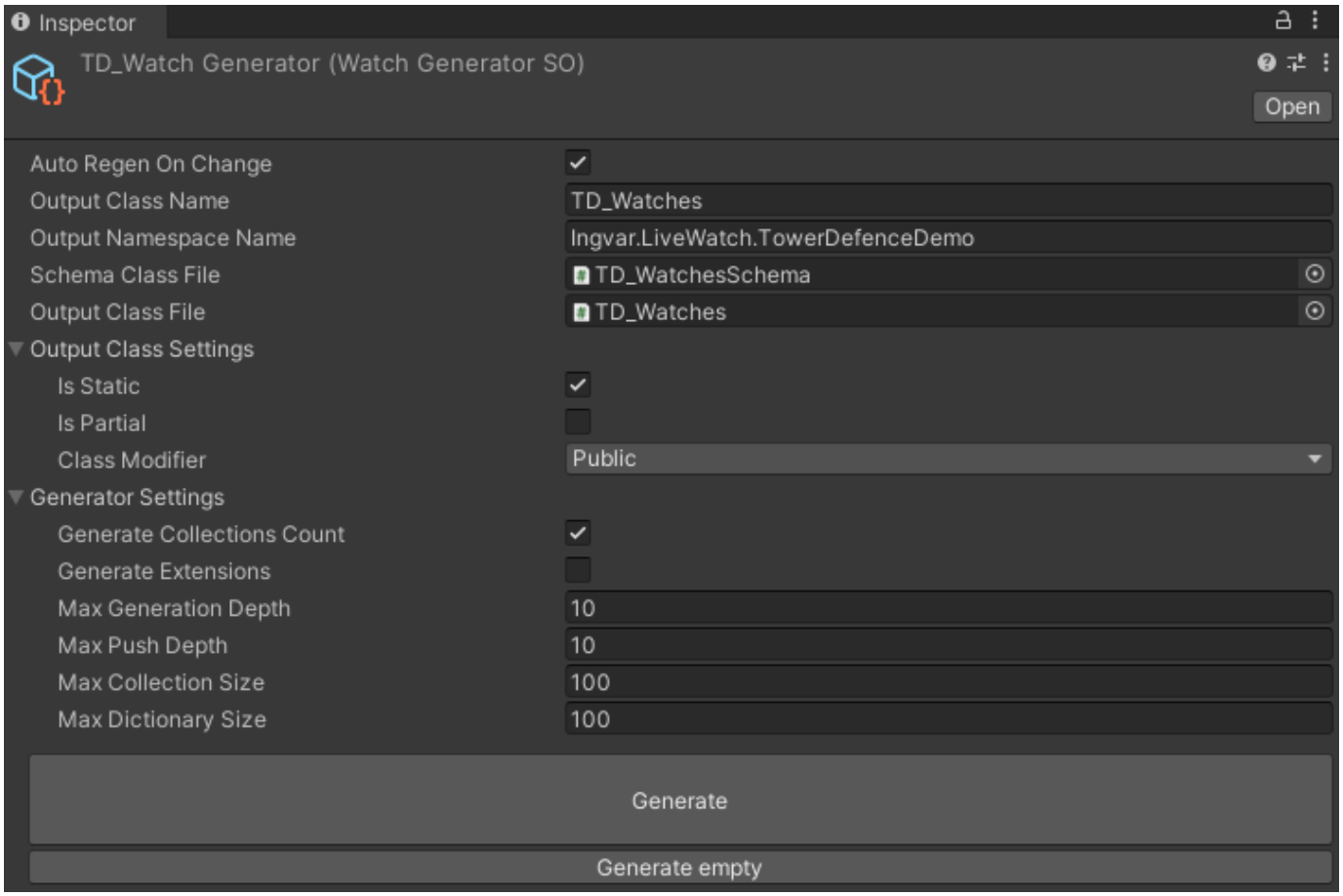
    Moves to the next value (to the right of the current one).

Next chapter: Generator Overview

3.3 Generator Overview

Watch Generator is a `ScriptableObject` that automates and simplifies the code generation process for custom variable types. In most cases, you only need to set it up once, and then it will function autonomously.

It can be created via **Create > LiveWatch > Generator**.



1. *Auto Regen On Change*

Automatically regenerates code after every change in the Schema file.

2. *Output Class Name*

Field for specifying your custom `Watch` class name.

3. *Output Namespace Name*

Field for your custom `Watch` class namespace. If left empty, no namespace will be generated.

4. *Schema Class File*

Reference to your `Schema` script in the project. This is where you can list the types you want to watch.

5. *Output Class File*

Reference to your custom `Watch` class script.

6. *IsStatic*

Indicates whether the generated class should be `static`.

7. *IsPartial*

Indicates whether the generated class should be `partial`.

8. *Class Modifier*

Specifies whether your Watch class will be `public` or `internal`.

9. *Generate Collections Count*

Determines whether `Count` / `Length` fields will be generated for collection types (`Array`, `List`, `Dictionary`, etc.).

10. *Generate Extensions*

Indicates whether extension methods will be generated. By default, this is turned off to reduce the generated file size.

11. *Max Generation Depth*

Defines how deep the generator will dive inside types. Avoid raising this too much as it can significantly affect performance and may lead to cyclic dependency issues.

12. *Max Push Depth*

Defines how deep Push methods will dive into their children. Also not recommended to raise too high for similar reasons.

13. *Max Collection Size*

Defines the maximum capacity for watchable collection (`Array`, `List`, `Stack`, etc.) members. Members exceeding this limit will be discarded. This limit does not affect other members like `Count`.

14. *Max Dictionary Size*

Same as the previous parameter but for *dictionaries*.

15. *Generate*

Triggers the generation process. This can take some time if there are many watchable types in the `Schema`.

16. *Generate Empty*

Also triggers generation, but all generated methods will be empty. This is useful if your project has compilation errors caused by the generated `Watch` class (e.g., after deleting some type's member). To resolve this, you can manually delete affected parts in the generated class or use **Generate Empty**, ensure that the project can be recompiled, and then click **Generate**.

Next chapter: Usage On Device

3.4 Usage on Device

Currently, **LiveWatch** is limited to use within `UnityEditor`. However, you can record data in a build, save it as a binary file, and then open it later in `UnityEditor` for review. To enable this, **WatchSaveCanvas** prefab is available in the root folder of the tool, providing the necessary setup for data exporting. If it doesn't suit your project's needs, feel free to create a custom implementation.

Steps to Use **LiveWatch** in a Build:

3.4.1 1. Add the `WatchSaveCanvas` Prefab

Add `WatchSaveCanvas` to your scene. By default, it can be accessed through a transparent `OpenButton` in the top left corner of the screen. Adjust this as needed or implement a custom access point. Just make sure to call `WatchServices.SaveLoader.Save` when saving data (you can pass `WatchStorageSO.instance.Watches` and your finish callback as optional parameters).

3.4.2 2. Add the `LIVE_WATCH_BUILD` Define

In your project settings, add the `LIVE_WATCH_BUILD` define.

3.4.3 3. Make a Build

Create a build of your project.

3.4.4 4. Launch and Save

Run the build and create a save. If using the default logic, click the top left corner, modify the file name if needed, and click save. The save path will display upon completion, defaulting to the Application.persistentDataPath folder.

3.4.5 5. Transfer and Load in Unity

Locate the `.watch` file, transfer it to your PC, and open it in the Unity editor by clicking `Load`.

4. Reference API

4.1 Watch

`Watch` class serves as the primary entry point for all operations involving watch variables.

4.1.1 Global Management Methods

These methods manage global operations for all variables in the project.

UpdateAll

```
public static void UpdateAll()
```

Updates all variables. Typically called from the `LateUpdate()` method but can be invoked elsewhere if needed.
How does UpdateAll **work?** For each variable, `Push()` is called recursively using the `Func<T> valueGetter` from `GetOrAdd()`. If a variable is created manually without a getter, it should be updated manually via `Push()`.

> **Update order matters**
>
> Manual updates must be done ***BEFORE*** calling `UpdateAll()` to avoid inconsistent visual representation.

> **Ensure Live Is Active**
>
> This method won't function if the Live Toggle is `off` in the **LiveWatch** window.

ClearAll

```
public static void ClearAll()
```

Clears all values for all variables, similar to the Clear button in the **LiveWatch** window.

DestroyAll

```
public static void DestroyAll()
```

Destroys all variables and their data. Usually called in `Awake()` to remove variables from previous playmode sessions. If you want to preserve variables from the previous session, you can skip this call.

4.1.2 Type-Indifferent Methods

These methods interact with variables regardless of type.

GetOrAdd

```
public static WatchReference<T> GetOrAdd<T>(string path)
```

Creates an existing variable or adds a new one of type `T` with name `path`. Note that type `T` must be one of the basic types or a type generated from the Schema.

PushEmpty

```
public static WatchReference<Any> PushEmpty(string path)
```

Pushes an empty value to the variable identified by name `path`, regardless of its type.

> **What is `Any` type?**
>
> `Any` is not a real type, it's a special type for cases when variable type does not matter.

PushFormat PRO

```
public static WatchReference<Any> PushFormat(string path, WatchValueFormat format)
```

Pushes a colored `format` to the variable with name `path`.

PushExtraText PRO

```
public static WatchReference<Any> PushExtraText(string path, string extraText)
```

Pushes additional text `extraText` to the variable with name `path`.

PushStackTrace PRO

```
public static WatchReference<Any> PushStackTrace(string path)
```

Captures and pushes the current stack trace to the variable with name `path`.

4.1.3 Per Type Methods

These methods are designed for concrete types including basic types and generated ones.

GetOrAdd

```
public static WatchReference<T> GetOrAdd(string path, Func<T> valueGetter)
```

Creates or retrieves a variable identified name `path`. The `valueGetter` is a function that is used for automatic updates to the variable during each `UpdateAll` call.

```
public static WatchReference<T> GetOrAdd<V>(WatchReference<V> parent, string path, Func<T> valueGetter)
```

Creates or retrieves a variable with name `path` that is a child of the specified `parent` variable. The `valueGetter` allows for automatic updates.

Setup

```
public static WatchReference<T> Setup(WatchReference<T> watchReference)
```

Prepares the specified variable `watchReference` for its first update. This method is automatically called for all variables created via the aforementioned methods and during the first call to `Push()`.

Push

```
public static WatchReference<T> Push(WatchReference<T> watchReference, T value, int maxRecursionDepth = 10)
```

Pushes the provided `value` into the given `watchReference` variable. The `maxRecursionDepth` controls how deep the push operation will proceed to prevent an infinite recursion loop.

```
public static WatchReference<T> Push(string path, T value, int maxRecursionDepth = 10)
```

Pushes the specified 'value' to the variable identified by name `path`. The `maxRecursionDepth` regulates the depth of the push operation.

PushValue

```
internal static WatchReference<T> PushValue(this WatchReference<T> watchReference, T value, int maxRecursionDepth = 10)
```

An extension method that functions similarly to the standard `Push()` method. It is not generated by default for custom types but can be enabled in the Generator.

> **Note if you want to use extensions**
>
> Extension methods must be called from the same namespace and assembly.

GetOrAddChild

```
internal static WatchReference<T> GetOrAddChild<V>(this WatchReference<V> parent, string path, Func<T> valueGetter)
```

An extension method that operates identically to the usual `GetOrAdd()`. It is not generated by default for custom types but can be enabled in the Generator.

SetupWatch

```
internal static WatchReference<T> SetupWatch(this WatchReference<T> watchReference)
```

An extension method that mirrors the functionality of the standard `Setup()`. It is not generated by default for custom types but can be enabled in the Generator.

4.2 WatchReference

`WatchReference<T>` is a struct type returned by most Watch methods, providing straightforward access to common operations for watch variables.

4.2.1 *ChildCount*

```
public int ChildCount
```

Returns the number of child variables. Returns 0 if there are none.

4.2.2 *GetChildNames*

```
public IEnumerable<string> GetChildNames()
```

Retrieves a collection containing the names of all child variables.

4.2.3 *GetOrAdd*

```
public WatchReference<V> GetOrAdd<V>(string path)
```

Retrieves an existing child variable or adds a new one with the specified name `path` and type `V`. This method is similar to `GetOrAdd()`.

4.2.4 *SetAlwaysCollapsable*

```
public WatchReference<T> SetAlwaysCollapsable()
```

Configures the variable columns to always be *collapsable*. If Collapse button is enabled in the Watch window, values from these variables will behave as if they haven't changed. This is useful for constantly changing values, to prevent cluttering.

4.2.5 *SetSortOrder*

```
public WatchReference<T> SetSortOrder(int value)
```

Assigns a sorting order to the variable, determined by the specified `value`. This functions similarly to scripting execution order in Unity: a higher value places the variable lower in the display order and vice versa. If no sorting order is specified, it defaults to the creation order of the variables, with a *float* value expected between `0` and `1`.

4.2.6 *SetDecimalPlaces*

```
public WatchReference<T> SetDecimalPlaces(int value)
```

Sets the number of decimal places for numeric values. The default is `2`. If applied to a non-numeric variable, it has no effect.

4.2.7 *UpdateOnce*

```
public WatchReference<T> UpdateOnce()
```

Configures the variable to update its value only once, after which all future updates will be ignored. Applies to both auto-updated variables during the first `UpdateAll()` call and manually-updated variables during the first `Push()`.

4.2.8 *PushEmptyValue*

```
public WatchReference<T> PushEmptyValue(bool withRoot = true, int maxRecursionDepth = 10)
```

Pushes an empty value to the variable and all of its children recursively, constrained by `maxRecursionDepth`. If `withRoot` is set to `false`, the empty value will not be pushed to the target variable itself, only to its children.

4.2.9 *SetTitleFormat* PRO

```
public WatchReference<T> SetTitleFormat(WatchTitleFormat format)
```

Overrides `format` for the variable's display area.

4.2.10 *SetDefaultValueFormat* PRO

```
public WatchReference<T> SetDefaultValueFormat(WatchValueFormat format)
```

Overrides the default `format` for the variable's values.

4.2.11 *AddConditionalValueFormat* PRO

```
public WatchReference<T> AddConditionalValueFormat(Func<T, bool> condition, WatchValueFormat format)
```

Defines a rule for formatting values with `format` based on a condition. `condition` is a delegate that takes a value of type `T` and returns whether the condition is met.

> **If multiple conditions are `true` for the same value**
>
> If multiple positive conditions are present, only the last one will take effect.

```
public WatchReference<T> AddConditionalValueFormat(Func<T, bool> condition, Func<T, WatchValueFormat> dynamicFormat)
```

Defines a rule for formatting values, similar to the previous method, but allows for `dynamicFormat` that determines the format based on the value.

4.2.12 *PushValueFormat* PRO

```
public WatchReference<T> PushValueFormat(WatchValueFormat format)
```

Directly pushes value `format` to the variable. Pushing formats functions like pushing values and must be done before calling `Watch.UpdateAll()`.

4.2.13 *PushExtraText* PRO

```
public WatchReference<T> PushExtraText(string extraText)
```

Pushes additional text `extraText` to the variable. Similar to pushing values, this must also be done before `Watch.UpdateAll()` is called.

4.2.14 *SetTraceable* PRO

```
public WatchReference<T> SetTraceable()
```

Marks variable as *Traceable*, automatically capturing the stack trace whenever its value is pushed.

> **Why I don't see the stack trace?**
>
> Note that `SetTraceable()` has no effect on auto-updating variables, as it would only capture the internal `Watch.UpdateAll` call. In such cases, you can manually push the stack trace using `PushStackTrace()` (or it's direct version) whenever you're certain the variable value has been changed.

4.2.15 *PushStackTrace* PRO

```
public WatchReference<T> PushStackTrace()
```

Captures and attaches the current invocation stack trace to the variable. Works regardless of whether `SetTraceable()` is enabled. Like pushing values, this must be called before `Watch.UpdateAll()`.

4.2.16 *SetMinMaxModeAsGlobal*

```
public WatchReference<T> SetMinMaxModeAsGlobal()
```

Sets the variable's min/max calculation mode to *global*, meaning the values min/max will be determined based on the entire history. By default, it's *local*, meaning the min/max is calculated only from the values currently visible on screen.

4.2.17 *SetMinMaxModeAsCustom*

```
public WatchReference<T> SetMinMaxModeAsCustom(double minValue, double maxValue)
```

Sets the variable's min/max calculation mode to *custom*, meaning the min/max values are manually set to the specified `minValue` and `maxValue`. By default, it's *local*, where min/max is calculated only from the values currently visible on screen.

> **What is *min/max mode*?**
>
> Min/max values are calculated for every variable to determine the fill level of the value cell's progress bar. It doesn't affect anything except the displayed cell visuals.

4.2.18 *SetCustomAction*

```
public WatchReference<T> SetCustomAction(string name, Action action)
```

Adds a custom button to the variable info area. `name` defines the button's text, and `action` specifies the function to execute when the button is clicked.

4.2.19 *TrackObjectReference*

```
public WatchReference<T> TrackObjectReference()
```

Enables the variable of any Unity type (like `MonoBehaviour` or `ScriptableObject`) to appear in the variable's info area, making it easy to locate the related object in the scene or project.

4.2.20 *PushObjectReference*

```
public WatchReference<T> PushObjectReference(UnityEngine.Object reference)
```

Pushes an `UnityEngine.Object` reference (e.g., `MonoBehaviour`, `ScriptableObject`) to the variable, allowing it to be displayed in the variable's info area.

4.3 WatchTitleFormat

The `WatchTitleFormat` is a struct used to customize the appearance of a watch variable's label area

4.3.1 *Constructor*

```
public WatchTitleFormat(Color color)
```

This constructor allows you to set the background color `color` for the label area of a watch variable.

4.4 WatchValueFormat

`WatchValueFormat` is a struct used to define the formatting options for a variable's value.

4.4.1 *Constructor*

```
public WatchValueFormat(Color fillColor)
```

This constructor sets the progress fill color `fillColor` for a variable cell.

```
public WatchValueFormat(Color fillColor, Color graphLineColor)
```

This constructor sets the progress fill color `fillColor` for a variable cell and the line color `graphLineColor` used in **Graph** mode.

4.5 WatchSchema

`WatchSchema` class is used by the **Generator** to declare and describe types for generation

4.5.1 *Generate*

```
protected WatchVariableDescriptor Generate(Type type)
```

Adds the specified type `type` to the generator. This method should be called from the `OnGenerate()` method. It returns an instance of `WatchVariableDescriptor`, which can be utilized to specify which fields should be generated or excluded.

```
protected WatchVariableDescriptor Generate<T>()
```

This is the generic version of the `Generate(Type)` method.

4.5.2 *Define*

```
protected WatchVariableDescriptor Define(Type type, bool withInheritors = true)
```

Adds a type `type` definition to the schema. This method functions similarly to `Generate` but does not add the target type to the generator; it only stores the type description. This is useful for inherited classes, as all definitions from the base class are applied to derived classes. `withInheritors` parameter determines whether this definition applies to all inheritors of the type. If set to `false`, only the type itself will be affected. This method should be called from the `OnDefine()` method.

```
protected WatchVariableDescriptor Define<T>(bool withInheritors = true)
```

This is the generic version of the `Define(Type, bool)` method.

4.5.3 *OnDefine*

```
public virtual void OnDefine()
```

This method allows you to override or add new type definitions. By default, `OnDefine()` includes several predefined definitions. For instance, it contains a definition for `MonoBehaviour` to exclude all its members, and definitions for basic Unity types like `Vector3`, which only includes the fields `x`, `y`, and `z`. You can modify this behavior by overriding this method and adding a new `Define` for the target type.

4.5.4 *OnGenerate*

```
public virtual void OnGenerate()
```

In this method, you can declare your generated types. If a target type was previously described by `Define` or if multiple descriptors for the same type exist, a *merging* algorithm will be applied. Merging prioritizes the descriptor of an *inherited* class over the base class. Therefore, if a field is ignored in the base class descriptor but allowed in the derived class descriptor, it will be visible in the generated derived type while remaining hidden in the base class.

4.6 WatchVariableDescriptor

`WatchVariableDescriptor` class allows you to specify which type members to include or exclude in the generated watch variable, along with custom names and additional parameters.

4.6.1 *Reset*

```
public WatchVariableDescriptor Reset()
```

Clears all specified data in the descriptor.

4.6.2 *SetSelfRecursion*

```
public WatchVariableDescriptor SetSelfRecursion(bool allow)
```

Determines whether the type can include members of the same type, useful for types like `Transform` and `Vector3` to ignore recursive members. Self recursion is *disabled* by default.

4.6.3 *SetTypeMask*

```
public WatchVariableDescriptor SetTypeMask(MemberType mask)
```

Defines which member types (`Field` , `Property` , `Method` , or `All`) can be included. By default, it includes fields and properties but excludes methods.

4.6.4 *SetSortOrder*

```
public WatchVariableDescriptor SetSortOrder(string memberNameReal, int sortOrder)
```

Assigns a sorting order to a variable's member `memberNameReal` , determined by the specified `sortOrder` . Works just like `SetSortOrder` in `WatchReference` .

4.6.5 *SetDecimalPlaces*

```
public WatchVariableDescriptor SetDecimalPlaces(string memberNameReal, int decimalPlaces)
```

Sets the number of decimal places for a variable's member `memberNameReal` , determined by the specified `decimalPlaces` . Works just like `SetDecimalPlaces` in `WatchReference` .

4.6.6 *SetAlwaysCollapsable*

```
public WatchVariableDescriptor SetAlwaysCollapsable(string memberNameReal)
```

Sets a variable's member `memberNameReal` values as always collapsable.

4.6.7 *UpdateOnceMember*

```
public WatchVariableDescriptor UpdateOnceMember(string memberNameReal)
```

Configures the member `memberNameReal` to update its value only once, after which all future updates will be ignored. Applies to both auto-updated variables during the first `UpdateAll()` call and manually-updated variables during the first `Push()` .

4.6.8 *RevertUpdateOnceMember*

```
public WatchVariableDescriptor RevertUpdateOnceMember(string memberNameReal)
```

Removes the `UpdateOnce` modifier from the specified member `memberNameReal` , allowing it to update normally again.

4.6.9 *IgnoreMember*

```
public WatchVariableDescriptor IgnoreMember(string memberName)
```

Excludes a member from the generated variable.

4.6.10 *RenameMember*

```
public WatchVariableDescriptor RenameMember(string memberNameReal, string memberNameShown)
```

Changes a member's displayed name in the variable.

4.6.11 *IgnoreMember*

```
public WatchVariableDescriptor IgnoreMember(string memberName)
```

Excludes a member from the generated variable.

4.6.12 *IgnoreMembers*

```
public WatchVariableDescriptor IgnoreMembers(params string[] memberNames)
```

Excludes multiple members.

4.6.13 *AllowMember*

```
public WatchVariableDescriptor AllowMember(string memberName)
```

Re-includes a member that was ignored in a lower priority descriptor.

4.6.14 *AllowMembers*

```
public WatchVariableDescriptor AllowMembers(params string[] memberNames)
```

Re-includes multiple members.

4.6.15 *ShowOnlyMember*

```
public WatchVariableDescriptor ShowOnlyMember(string memberName)
```

Excludes all other members except the specified one.

```
public WatchVariableDescriptor ShowOnlyMember(string childNameReal, string childNameShown)
```

Combines `ShowOnlyMember` and `RenameMember`.

4.6.16 *ShowOnlyMembers*

```
public WatchVariableDescriptor ShowOnlyMembers(params string[] memberNames)
```

Same as `ShowOnlyMember`, but for multiple members.

4.6.17 *IgnoreAllMembersDeclaredInClass*

```
public WatchVariableDescriptor IgnoreAllMembersDeclaredInClass()
```

Ignores all members from the target class, useful for base class definitions.

```
public WatchVariableDescriptor IgnoreAllMembersDeclaredInClass(Type hierarchyClass)
```

Same as `IgnoreAllMembersDeclaredInClass`, but for a specified class in the hierarchy.

4.6.18 *AllowAllMembersDeclaredInClass*

```
public WatchVariableDescriptor AllowAllMembersDeclaredInClass()
```

Restores all previously ignored members from the target class.

```
public WatchVariableDescriptor AllowAllMembersDeclaredInClass(Type hierarchyClass)
```

Same as `AllowAllMembersDeclaredInClass`, but for a specified class in the hierarchy.

4.6.19 *SetObjectReferenceTrackable* PRO

```
public WatchVariableDescriptor SetObjectReferenceTrackable(bool trackForThisType)
```

Enables or disables tracking of Unity types (e.g., `MonoBehaviour`, `ScriptableObject`), allowing them to appear in the variable's info area for easy reference in the scene or project.

4.6.20 *SetObjectReferenceTrackable* PRO

```
public WatchVariableDescriptor SetObjectReferenceTrackable(string memberNameReal)
```

Enables the member `memberNameReal` of any Unity type (like `MonoBehaviour` or `ScriptableObject`) to appear in the variable's info area, making it easy to locate the related object in the scene or project.

4.6.21 *SetObjectReferenceNonTrackable* PRO

```
public WatchVariableDescriptor SetObjectReferenceNonTrackable(string memberNameReal)
```

Disables tracking for the specified member `memberNameReal`, preventing it from appearing in the variable's info area.

4.6.22 *SetTraceable* PRO

```
public WatchVariableDescriptor SetTraceable(string memberNameReal)
```

Sets a variable's member `memberNameReal` values as traceable.

4.6.23 *SetMinMaxModeAsGlobal* PRO

```
public WatchVariableDescriptor SetMinMaxModeAsGlobal(string memberNameReal)
```

Sets a variable's member `memberNameReal` min/max mode as global.

4.6.24 *SetMinMaxModeAsCustom* PRO

```
public WatchVariableDescriptor SetMinMaxModeAsCustom(string memberNameReal, double minValue, double maxValue)
```

Sets a variable's member `memberNameReal` min/max mode as custom.

4.6.25 *SetCollectionModeAsKey* PRO

```
public WatchVariableDescriptor SetCollectionModeAsKey(string keyMemberRealName)
```

Enables collection types (*Array*, *List*, etc.) to be displayed as key-value pairs in editor, similar to dictionary types. The `keyMemberRealName` parameter defines which member of the element will be used as the display key.

5. Support

For any inquiries, please reach out to us.

justingvar.dev@gmail.com