RUBI LOGGER

Documentation

v.1.0.0

Warning

Notice that this documentation is a PDF. I recommend you to go directly to the Google Doc version of it so you would be sure that you are reading the latest version.
[Google Doc Version]

Introduction

Rubi Logger is a powerful and flexible logging solution for Unity. It is designed to replace the standard `Debug.Log` system, providing additional features and control over your application's logging.

Installation

To install the Logger system, simply import the provided package into your Unity project.

Content

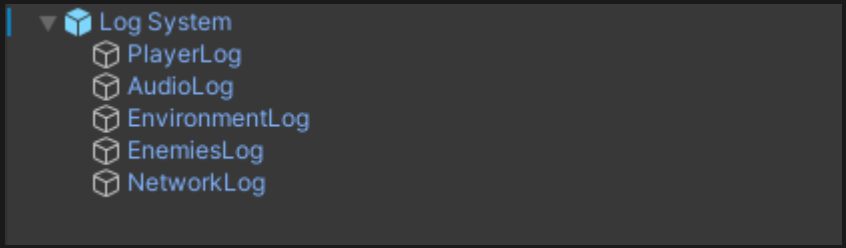
Package includes:

- Two MonoBehaviour Scripts:
  - `RubiLogger` - logger for MonoBehaviours
  - `RubiLoggerStatic` - logger for NonMonoBehaviours
  - `LoggerDisplay` - additional script that handles logs appears on the GameView screen using UI. In the `Awake` method it searches all `RubiLogger`'s and listens for screen logs.
- Two Editor Scripts:
  - `RubiLoggerEditor` - to have custom inspector for RubiLogger
  - `LoggerDisplayEditor` - to have custom inspector for DisplayLogger

- One NON-MonoBehaviour script:
  - RubiConstants - this script holds static class RubiConstants which holds Default Path for your logs and colors for each log level. Script also holds both LogLevel and LogOutput enums.

Quick Start

1. Take a “Log System” prefab from the Prefabs folder and add it in your scene.



2. Parent object is an empty GameObject and it’s there only for clear project structure. Its children are actual RubiLoggers that you can use. Each of this RubiLogger is a category for your logs. It's made to have well structured logs.

3. You can remove loggers you don’t need, rename or add your new logger. To add a new logger you need to create a new empty GameObject as a child of “Log System” GameObject and add a RubiLogger as a component.

4. You can change settings for each RubiLogger.

   a. Show Logs - if you disable this logger will not generate logs anywhere. You are actually disabling it.

   b. Category - it will be used in logs as a prefix. It's a very important part of the RubiLogger. It helps to have clean and understandable logs.

   c. Category Color - used to not only distinguish categories by name, but also by color. Please be sure you don’t have similar colors for two or more categories.

   d. Log Level Filter - you can set this property to Debug, Info, Warn or Error. If a logger will get a log which

level is below Log Filter Level - log won’t be generated. For example Log Filter is set to Info and you are trying to log a Debug Log, you won’t see a log anywhere. It’s made if you want to reduce the amount of logs you generate.

e. Screen Logs - you can disable this if you are not going to show logs on screen. Enable it if you want to use Screen Logs and be sure you added LoggerDisplay in your scene as a child of Canvas.

f. If Screen Logs are disabled you can see a toggle called Show Error When Disabled. You can use it if you don’t want to get error logs when you are trying to show logs on screen when Screen Logs disabled.

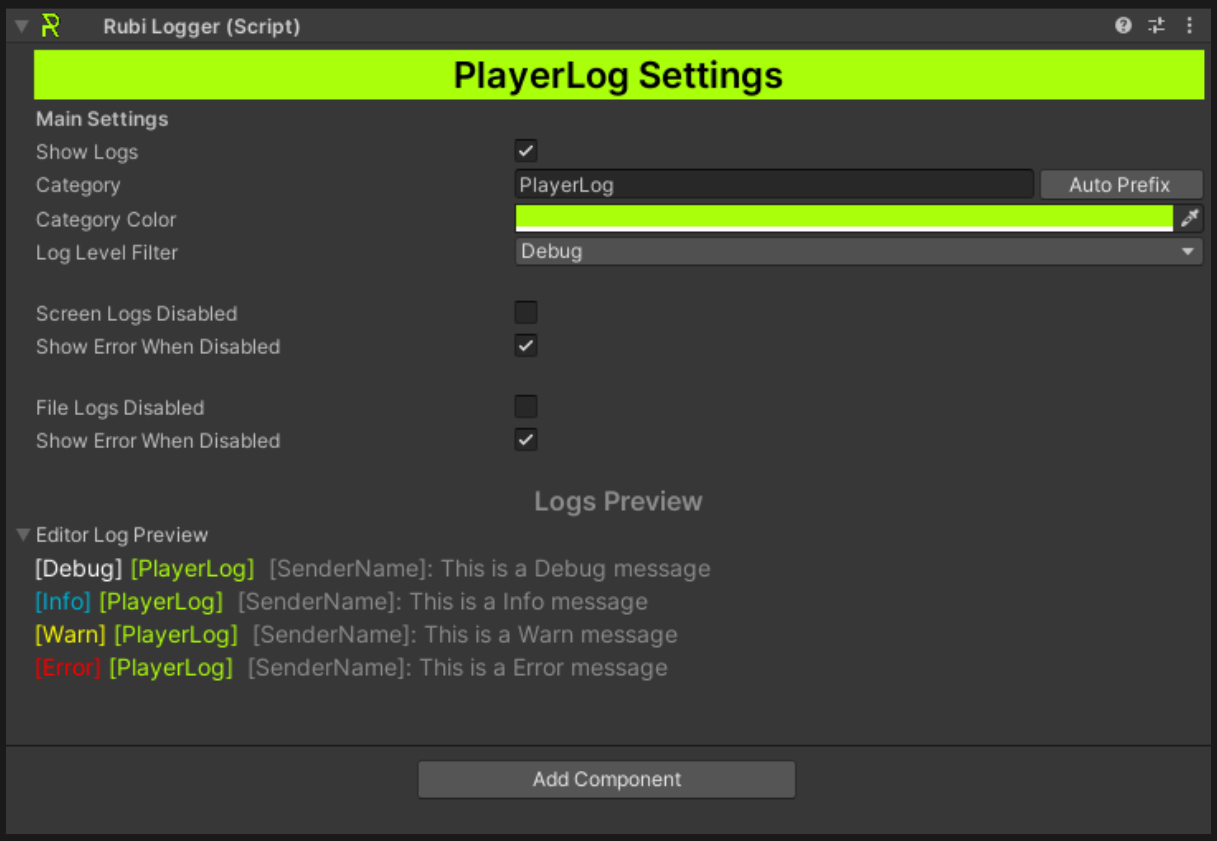
g. Screen Logs Settings are set in LoggerDisplay

h. File Logs - you can disable this if you are not going to write logs in file. Enable it if you want to write logs into File.

i. If File Logs are disabled you can see a toggle called Show Error When Disabled. You can use it if you don’t want to get error logs when you are trying to write logs in a file when File Logs disabled.

j. In File Log Settings you can change the directory where RubiLogger will create a file for logs. Also you can press “Set Default Path” and you will get the default path that is specified in the RubiLogger script.

k. Logs Preview shows you how your logs will look like for the current RubiLogger.



5. When you set up all your RubiLoggers it's time to use it. Add a **`[SerializedField] private RubiLogger rubiLogger`** inside your scripts you want to log from. I attached an example below.

```
7 asset usages
public class Player : MonoBehaviour
{
    [SerializeField] private RubiLogger rubiLogger;  Unchanged
}
```

6. When you want to log anything, call the **`Log()`** method from **`rubiLogger`**. Log has 5 parameters, but requires a minimum of 3 parameters:

    a. LogLevel - it's Debug, Info, Warn or Error.

    b. Message - it's the message you want to log.

c. **Sender** - it's `this` in 99% of cases. It sends information about who sent this log.

d. **LogOutput** - this parameter is LogOutput.Console by default. If you want to show log only in the console you don't need to specify this parameter. But if you want to change the output of the log you can specify the output you need.

e. **ByPassFilterLogLevelFilter** - this parameter is false by default. You can use it to make RubiLogger generate the log regardless of its Log Level Filter.

```
Event function
10      private void Start()
11      {
12          rubiLogger.Log(LogLevel.Info, message: "Player started!", sender: this);
13      }
14  }
```

Console Log

```
Event function  More...
10      private void Start()
11      {
12          rubiLogger.Log(LogLevel.Info, message: "Player started!", sender: this, LogOutput.ConsoleAndFile);
13      }
14  }
```

Console and File Log

7. That's all! But be sure to add Logger Display into your scene as a Canvas child. You can use the Logger Display prefab.

Logger Settings

The system provides a variety of settings that can be adjusted to customize the behavior of the logging system. These settings can be accessed and modified through the RubiLogger component in the Unity Inspector.

Show Logs

This setting controls whether logs should be generated. If this is set to false, no logs will be generated. It's useful when you want to ignore all logs coming to this RubiLogger.

Category Name

This setting allows you to specify a Category prefix for the logs. This prefix will be displayed at the beginning of each log message. It's a very useful feature that can help you to organize your logs using categories. You can press the *“Auto Prefix”* button to set Category Name based on the GameObject’s name this script is attached to.

Category Color

This setting allows you to specify the color of the Category prefix. The color can be set using the standard Unity color picker. It helps you distinguish categories by color.

Log Level Filter

This setting allows you to filter logs based on their level. Only logs of this level and higher will be displayed. It’s useful when you want to ignore logs of a specific level. For example, when you want to disable all non-essential logs like Debug and Info in your release version of a game.

Log File Path

Here you can specify a directory path for your log file. Press *“Set Default Button”* to set the default path specified in the RubiConstants script.

Show Error When Disabled

For example, if you are trying to Log a message into a file and RubiLogger’s File Log is disabled - you will get an error in the console. If you want to hide errors like this - set this property to false.

Logging messages

To log a message, call the **Log()** method on your RubiLogger instance. Method takes several parameters:

1. (enum)LogLevel logLevel
2. object message
3. object sender
4. (enum)LogOutput logOutput
5. bool bypassFilterLevel

Log Levels

The Logger system supports three levels of logs:

- LogLevel.Debug
  Should be used for information that may be needed for diagnosing issues and troubleshooting or when running applications in the test environment for the purpose of making sure everything is running correctly.

- LogLevel.Info
  Should be used for indicating that something happened, the application entered a certain state.

- LogLevel.Warn
  Should be used for indicating that something unexpected happened in the application, a problem, or a situation that might disturb one of the processes. But that doesn't mean that the application failed. The Warn level should be used in situations that are unexpected, but the code can continue the work.

- LogLevel.Error
  Should be used when the application hits an issue preventing one or more functionalities from properly functioning.

Message

Message is object type so you can put anything you could put in Debug.Log. If you want to get maximum results from logs be careful when choosing what message to send as a log.

Sender

You always need to specify who is sending the log. It’s useful because you can double-click the log in the console editor and your IDE will open the script where a log was called.

Also it’s useful because you will see a sender name inside the log. It has the same purpose as a Category. It helps to have clean manageable logs.

Log Outputs

The Logger system supports different types of log outputs:

- `LogOutput.Console`: The log message will be displayed in the console.
- `LogOutput.Screen`: The log message will be displayed on the screen.
- `LogOutput.File`: The log message will be written to a file.
- `LogOutput.ConsoleAndScreen`: The log message will be displayed in the console and on the screen.
- `LogOutput.ConsoleAndFile`: The log message will be displayed in the console and written to a file.
- `LogOutput.ScreenAndFile`: The log message will be displayed on the screen and written to a file.
- `LogOutput.All`: The log message will be displayed in the console, on the screen, and written to a file.

Log Level Filter

The Logger system allows you to filter logs based on their level. This is controlled by the logLevelFilter property on the Logger component. Only logs of this level and higher will be displayed. If you want to log a message regardless on RubiLogger settings - set the last parameter bypassLevelFilter to true.

Non-MonoBehaviour Logging

If you want to log from a non-MonoBehaviour script you can call method Log from RubiLoggerStatic. You should specify Level of log, message and sender name. You can also specify output the same as with RubiLogger but also you can specify category name and category color using UnityEngine.Color. Category name and color are not required and it's set as "NonMono" and gray color by default. File log will be written in a file specified in RubiConstant default path. It also supports screen logs, no need for extra moves, just call log with screen in output but don't forget to have Logger Display in the scene. Non-MonoBehaviour Logging doesn't support filtering and file or screen logs disabling.

Rubi Constants

This file holds LogLevel and LogOutput enums, Default Path and Colors for each LogLevel. These properties were not supposed to be changed but in case you want to add new LogLevel, LogOutput, change Default Path or color of any log level - you can do it here. But I tried to put the most obvious values so I hope you won't need to change anything.

FAQ

- *Can I add more LogLevels?*

Yes, you can. Just open the RubiLogger script and add a new level inside the LogLevel enum. Also set its color inside LogLevelColors using hexadecimal value. If you don't set its color inside the dictionary - you will get the error.

- *Can I make RubiLoggers to write in different files?*

Yes, just specify different files in the Log File Path.

- *Can I change RubiLogger's settings in runtime?*

No, since all properties are private. Changing RubiLogger's settings in runtime is a bad practice. But if you really want to - change needed properties to public.

- I have a Logger Display in the scene but there are no logs appearing on the screen.

Please make sure you enabled screen logs in RubiLogger you are trying to call Log from. Also check RectTransform of LoggerDisplay. If you still have any issues with that, please don't hesitate to contact me - I'm always there to help :)

Contact

Please feel free to contact me if you have any questions.
rubickanov@gmail.com