Rubi Logger
Documentation

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.
Rubi Logger includes 2 types of Loggers:

- `RubiLogger`
    - Optimized but can't add your format log
- `CFRubiLogger`
    - Takes performance but you can set up your Log Format for each output like Console, Screen or File.

I highly recommend using the standard RubiLogger because it's not that expensive like the CF version. The CF version is good if you want to change the log format and use it until your project is in the "Prototype" stage.

Content

Package includes:

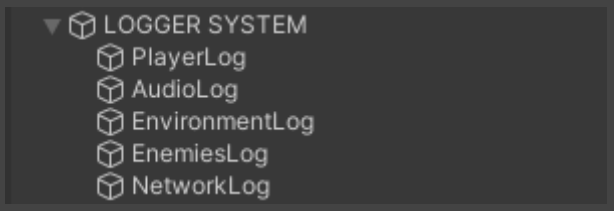
- Four MonoBehaviour Scripts:
    - `RubiLogger` - the main script that handles all logging part
        - `CFRubiLogger` - custom format version of standard rubi logger, that allows users to set up their own log formats.
    - `LoggerDisplay` - additional script that handles logs appears on the GameView screen using UI. In the `Awake` method it searches all `RubiLogger`'s and listens for screen logs.

        - `CFLoggerDisplay` - additional script inherited from `LoggerDisplay` that handles logs made using `CFRubiLogger` appears on the GameView screen using UI.
- One Scriptable Object script:
    - `LogFormat` - allow user create his own Log Format and change order of Log Parts of the log.

- Two Editor Scripts:
    - `RubiLoggerEditor` - to have custom inspector for RubiLogger
    - `LoggerDisplayEditor` - to have custom inspector for DisplayLogger
    - CFRubiLogger

Usage

The Rubi Logger system is primarily used through the RubiLogger component. This component can be added to any GameObject in your scene. We suggest having a list of objects as children of one parent object. Then add to each child a RubiLogger component.



Then add a serialized field in any script you want to log at and assign the logger you want to handle that log in the inspector.

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 displayed in the console. If this is set to false, no logs will be displayed.

Prefix

This setting allows you to specify a prefix for the logs. This prefix will be displayed at the beginning of each log message.

Prefix Color

This setting allows you to specify the color of the prefix. The color can be set using the standard Unity color picker.

Log Level Filter
This setting allows you to filter logs based on their level. Only logs of this level and higher will be displayed.

Buttons
There are also two buttons available in the Logger component in the Unity Inspector:

Auto Prefix
Clicking this button will automatically set the Logger prefix to the name of the GameObject the Logger component is attached to.

Set Default Path
Clicking this button will set the log file path to the default path, which is "Game Logs/log.txt". You can change the default path inside the RubiLogger script.

By adjusting these settings, you can control how and where your logs are displayed.

Logging messages

To log a message, call the Log method on your RubiLogger instance. The Log method takes several parameters:

- **`LogLevel logLevel`**
  The level of the log.

- **`string message`**
  The message to log.

- **`Object sender`**
  The object that is sending the log. This is typically “this”.

- **`LogType logType`**
  The type of the log output. This is set to LogType.Console by default.

- **`bool bypassLogLevelFilter`**
  Whether to bypass the log level filter. If this is true, the message will be logged regardless of the current log level filter. This is set to *false* by default.

Log Levels

The Logger system supports three levels of logs:
- `LogLevel.Info`: Used for general information and debugging.
- `LogLevel.Warning`: Used for warnings that might indicate a problem.

- `LogLevel.Error`: Used for errors that have occurred.

Log Types

The Logger system supports different types of log outputs:

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

Writing Logs to a File

The Logger system can write logs to a file. To enable this, set the logType parameter of the Log method to `LogType.File`, `LogType.ConsoleAndFile`, `LogType.ScreenAndFile`, or `LogType.All`. The path to the log file can be set in the logFilePath property on the `RubiLogger` component.

Conclusion

The `RubiLogger` system provides a powerful and flexible logging solution for Unity. With features like log level filtering, color coding, and multiple output types, it is a significant upgrade over the standard `Debug.Log`.