Custom Logger User Manual

Contents

Introduction

Overview

Custom Logger is a Unity development tool that provides a powerful and flexible logging system for Unity projects. It allows developers to centralize and manage console logging through configurable settings, making it easier to track and debug issues in their applications.

Features

- **Configurable Logging**
    - Multiple logging configurations with unique keys
    - Enable/disable logging per configuration
    - Custom color settings per configuration
- **Flexible Logging System**
    - Supports standard Unity log types (Log, Warning, Error)
    - Automatic type identification in logs
    - Color-coded output for better readability
    - Intercepting log handler for seamless integration with Unity's logging system
- **Easy Integration**
    - Simple setup through Unity Package Manager
    - Automatic settings file generation

- Intuitive configuration interface

Requirements

- .NET 4.x Runtime
- ScriptableObject support

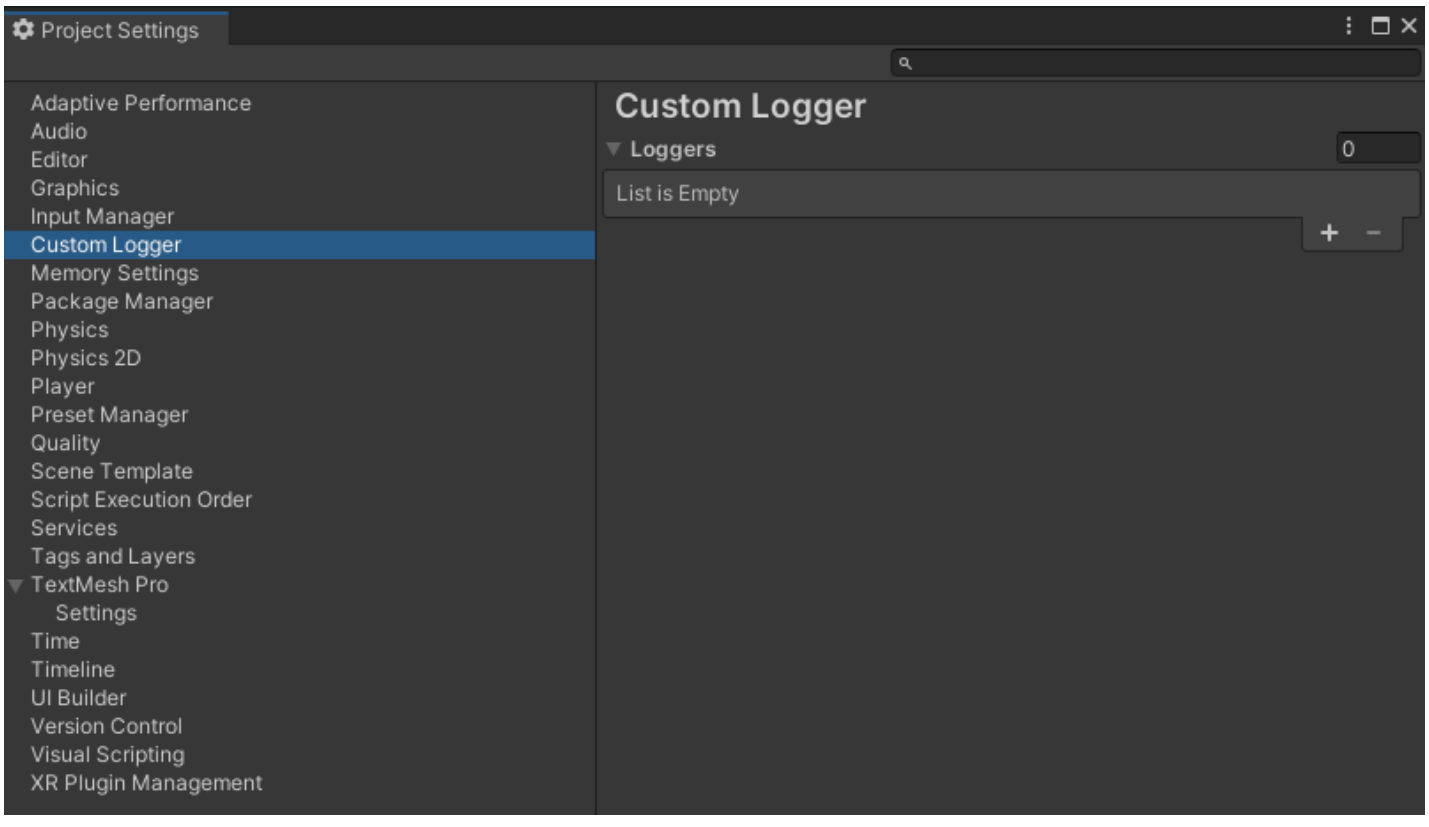
Getting Started

Installation

1. Open Unity Editor
2. Go to Window → Package Manager
3. Click on the "+" button at the top left corner
4. Select "Add package from git URL"
5. Enter: `https://github.com/CRE-Tools/CustomLogger.git`
6. Click "Add"
7. Wait for the package to be installed

Basic Setup

1. Open the Custom Logger Settings through:
   - Edit → Project Settings → Custom Logger
   - File "Assets/LoggerSettings/Loggers.asset" (this file will be created automatically when first opening the settings window at Project Settings)
2. Add a new configuration to the list
   - Assign a Key Name for the configuration respecting the following rules:
     - Always use a unique key for each logging configuration
     - The key must respect format: only letters, no spaces, no reserved strings
   - Configure the desired log color
   - Set the initial state (enabled/disabled)
     - notice that messages will only appear if the configuration for this key is enabled
3. Save the settings by clicking on the "Apply Settings" button
4. Wait for recompile



Quick Start Guide

Here's a quick example of how to use Custom Logger in your code:

Using Custom Logger

- Use namespace PUCPR.CustomLogger

Basic Approach:

```
using UnityEngine;
using PUCPR.CustomLogger;

public class ExampleClass : MonoBehaviour
{
    private void Start()
    {
        // New approach using Unity's built-in Debug methods, basic with always log
        Debug.LogFormat("Game started", CustomLoggerKey.AlwaysLog);
    }
}
```

If you want to use a custom key, you can configure a new key in the settings file.

This will generate a new enum value in the CustomLoggerKey enum.

This enum is used to identify the logging configuration.

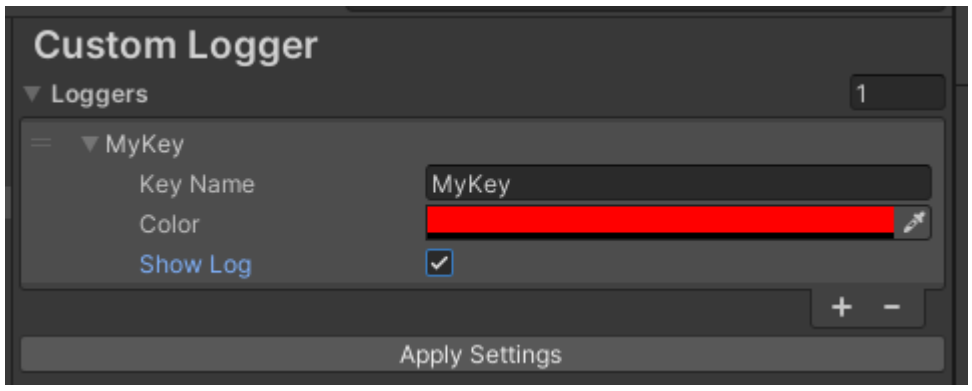
To do this:

1. Open the Custom Logger Settings through:
    - Edit → Project Settings → Custom Logger
    - File "Assets/LoggerSettings/Loggers.asset" (this file will be created automatically when first opening the settings window at Project Settings)
2. Add a new configuration to the list
    - Assign a Key Name for the configuration respecting the following rules:
        - Always use a unique key for each logging configuration
        - The key must respect format: only letters, no spaces, no reserved strings
    - Configure the desired log color
    - Set the initial state (enabled/disabled)
        - notice that messages will only appear if the configuration for this key is enabled
3. Save the settings by clicking on the "Apply Settings" button
4. Wait for recompile

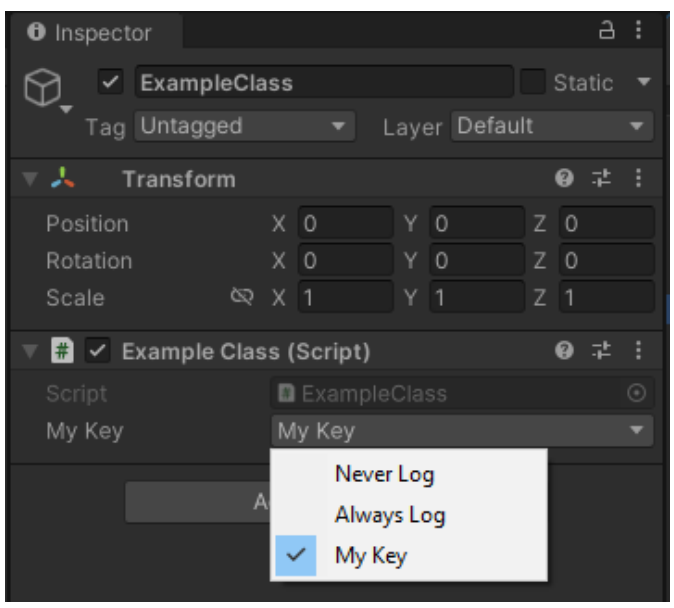
Example:

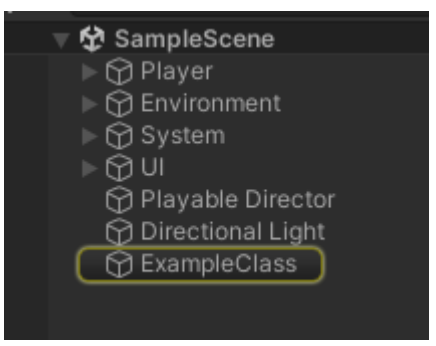
New key created:

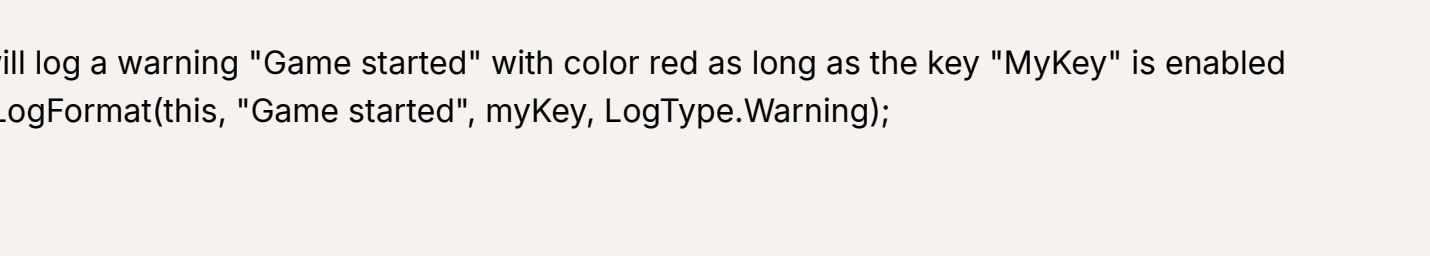
- Key Name: "MyKey"
- Color: Red
- Enabled: True




```
using PUCPR.CustomLogger;

public class ExampleClass : MonoBehaviour
{
    //Selected as CustomLoggerKey.MyKey in the inspector
    [SerializeField] private CustomLoggerKey myKey;

    private void Start()
    {
        //This will log a warning "Game started" with color red as long as the key "MyKey" is enabled
        Debug.LogFormat(this, "Game started", myKey, LogType.Warning);
    }
}
```







Package Structure

Assets

- Assets/LoggerSettings/
- Loggers.asset (this is the settings file where you can configure the logging keys. will be created automatically when first opening the settings window)
  -Package/CustomLogger/
- Documentation~/
  - UserManual.md (this is the user manual)
  - UserManual.pdf (this is the user manual in pdf format)

Scripts

- Package/CustomLogger/Scripts/
- Runtime/
  - CodeGenerator.cs (this is the code generator that generates the CustomLoggerKey enum)
  - CustomLoggerType.cs (this is object type that configurations for keys are stored in the settings file)
  - CustomLoggerKey.cs (auto generated enum with all the keys configured in the settings file)
  - CustomLoggerSettings.cs (this is the settings manager that handles the settings file)
- Editor/
  - EDITOR_Logger.cs (Custom Editor for the settings file)
  - InterceptingLogHandler.cs (Intercepts Unity log messages and applies custom formatting)
  - InterceptingLogInitializer.cs (Initializes the intercepting log handler)

Settings

- `Assets/LoggerSettings/Loggers.asset`: Contains all configurable settings per configuration key including:
  - Enable/Disable key
  - Color settings

Usage

Core Concepts

1. **Logging Configurations**
   - Each configuration is identified by a unique key
   - Configurations can be enabled/disabled in the settings file
   - Each configuration can have its own color settings
   - The intercepting log handler automatically processes all log messages and applies custom formatting to those that have a valid key
2. **Log Types**
   Follow standard log types:
   - Log: Standard information messages
   - Warning: Warning messages
   - Error: Error messages
   - Pass them as additional parameter in the args array (example: Debug.LogFormat(this, "Game started", CustomLoggerKey.AlwaysLog, LogType.Warning);)
3. **Key System**
   - Each enum value represents a different logging configuration
   - Use these keys to organize and filter your logs

Best Practices

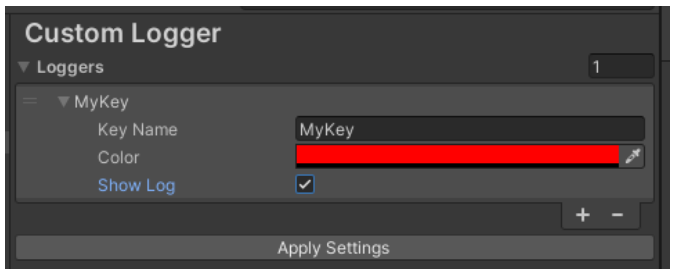
1. **Configuration Management**
   - Use descriptive keys for different logging contexts
   - Keep enabled only the keys that are necessary for the current state of the project
   - Enable/Disable configuration keys in Settings to control which logs are displayed
2. **Logging Strategy**
   - Log important state changes and errors grouped by configuration key
   - Pass CustomLoggerKey as additional parameter in the args array

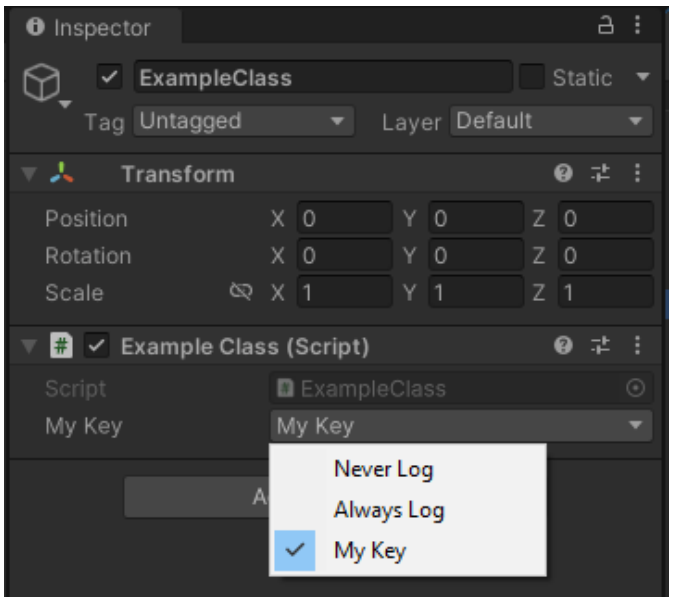
- For log types other than Log (which is default), pass the LogType as additional parameters in the args array
- Pass "this" as the context object for better debugging in the Unity console. This will identify the GameObject source of the log
- Use AlwaysLog for fast access to log messages without any additional configuration (this key is always enabled)
- Use NeverLog to completely disable specific log messages (this key is always disabled)
- For custom keys, ensure they are properly configured in the settings

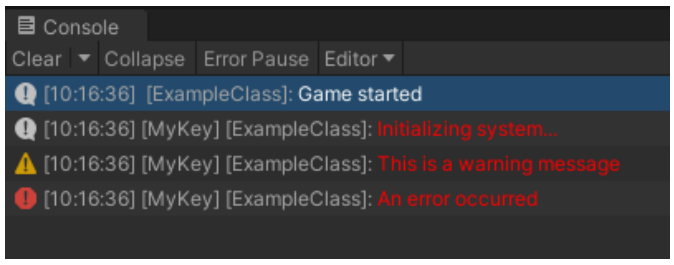
3. **Code Organization**
   - Keep logging configurations organized by feature
   - Use consistent naming conventions for keys
   - Group related logs under the same configuration key
   - Declare key as a field of the class to avoid redeclaring it every time it is used

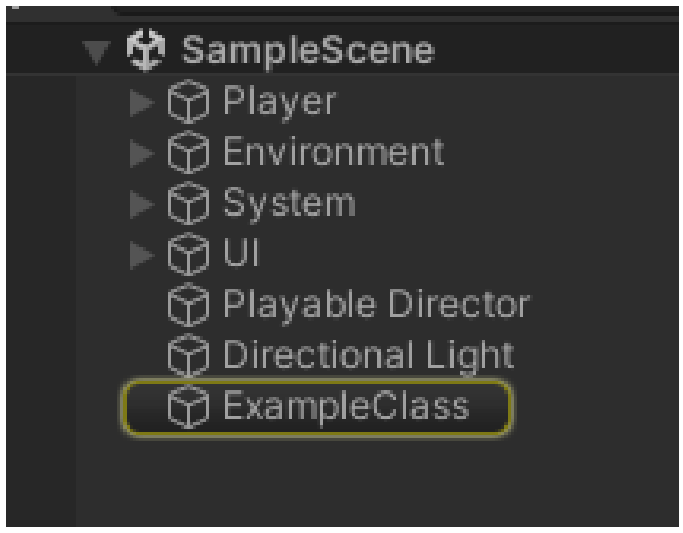
Examples

```
using UnityEngine;
using PUCPR.CustomLogger;

public class ExampleClass : MonoBehaviour
{
    [SerializeField] private CustomLoggerKey myKey;

    private void Start()
    {
        // Simple log with AlwaysLog (always enabled)
        Debug.LogFormat(this, "Game started", CustomLoggerKey.AlwaysLog);

        // Log with custom key (must be configured in settings)
        Debug.LogFormat(this, "Initializing system...", myKey);

        // Warning example
        Debug.LogFormat(this, "This is a warning message", myKey, LogType.Warning);

        // Error example
        Debug.LogFormat(this, "An error occurred", myKey, LogType.Error);
    }
}
```









Troubleshooting

Common Issues

1. **Logs not appearing**
    - Check if the configuration is enabled in settings
    - Verify the correct key is being used
    - Verify that the key is not NeverLog
    - Verify that the log is called by Debug.LogFormat
2. **Color not applying correctly**
    - Check if the color is properly configured in settings
    - Verify the correct key is being used

FAQ

1. **Can I use multiple configurations in one log?**
    - No, each log must belong to a single configuration
    - Use descriptive configurations to organize your logs
    - You can group multiple logs under the same configuration key
2. **How do I change log colors?**
    - Open the Custom Logger settings window
    - Select the configuration
    - Adjust the color settings as needed
3. **How do I enable/disable a log configuration?**
    - Open the settings window
    - Enable/disable the configuration
4. **How do I add a new configuration?**
    - Open the settings window
    - Click on the plus button to add a new configuration on the bottom of the list
    - Enter a unique key name respecting format (only letters, no spaces, no reserved characters)
    - Set the desired log color
    - Set the initial state (enabled/disabled)
    - Click "Apply Settings"
5. **How do I remove a configuration?**
    - Open the settings window

- Select the configuration
- Click on the minus button to remove the configuration from the list
- Click "Apply Settings"