Clear, organized, and optimized logging for Unity

BACKBONE LOGGER

Version 1.0

By Xavinet

Contenido

1. Introduction

Backbone Logger is a lightweight and powerful logging system for Unity that helps you keep your Console organized and efficient.
As projects grow, the Unity Console often becomes cluttered with hundreds of messages, making it hard to identify what really matters. Backbone Logger solves this by letting you:

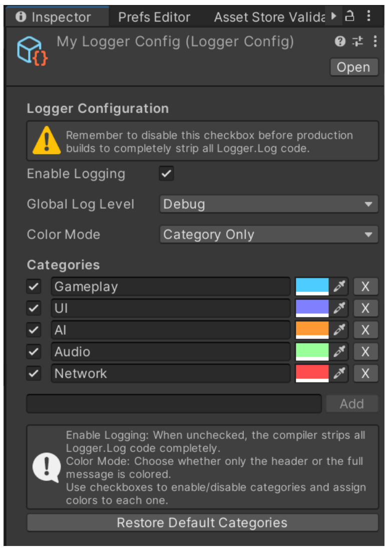
- **Organize logs into categories** (Gameplay, UI, Network, etc.).
- **Filter by severity level** (Debug, Info, Warning, Error, Critical).
- **Use color-coded messages** to quickly spot issues.
- **Strip all logging code** from production builds with a single checkbox for **zero runtime cost**.

With Backbone Logger, you see the **signal through the noise**, keeping your workflow clean and focused.
It's simple to set up, easy to customize, and built to scale with your project.

2. Quick Start

*Goal: Set up Backbone Logger and see your first colored logs in **under 2 minutes**.*

2.1 Create a LoggerConfig



Logger Configuration

Remember to disable this checkbox before production builds to completely strip all Logger.Log code.

Enable Logging	✓
Global Log Level	Debug
Color Mode	Category Only

Categories

✓	Gameplay	🟦
✓	UI	🟪
✓	AI	🟧
✓	Audio	🟩
✓	Network	🟥

Enable Logging: When unchecked, the compiler strips all Logger.Log code completely.
Color Mode: Choose whether only the header or the full message is colored.
Use checkboxes to enable/disable categories and assign colors to each one.

The LoggerConfig asset stores all logger settings: categories, colors, and filtering options.

1. In Unity, right-click in the **Project window** and select:
Create → Backbone → Logging → LoggerConfig
2. Name it, for example: LoggerConfig_Main.
3. By default, it includes common categories:
Gameplay, UI, AI, Audio, Network.

2.2 Initialize the Logger in Your Scene

Backbone Logger must **load its configuration** when the game starts.
There are two ways to do this:

Option 1 — Use LoggerManager (recommended)

1. Create an empty GameObject in your scene (e.g., **Logger**).
2. Add the **LoggerManager** component.
3. Drag your LoggerConfig asset into the **Config** field.

This automatically applies the settings at runtime.

Option 2 — Initialize from Your Own Script

If you need more control, call ApplyConfig() manually:

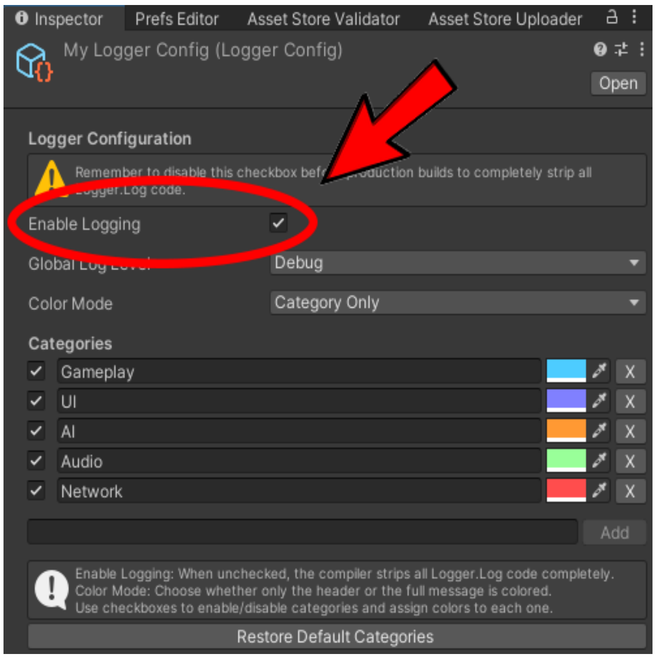
```csharp
using UnityEngine;
using Backbone;

public class CustomLoggerSetup : MonoBehaviour
{
    public LoggerConfig config;

    void Awake()
    {
        if (config != null)
            config.ApplyConfig(); // Activates Backbone Logger
    }
}
```

2.3 Enable Logging Checkbox

The Backbone Logger is disabled by default. Enable it by opening the LoggerConfig object and checking the box, but remember to turn it off before deployment.



Logger Configuration

Remember to disable this checkbox bef[illegible]oduction builds to completely strip all Logger.Log code.

Enable Logging	✓
Global Log Level	Debug
Color Mode	Category Only

Categories

✓	Gameplay	🟦	X
✓	UI	🟪	X
✓	AI	🟧	X
✓	Audio	🟩	X
✓	Network	🟥	X

Enable Logging: When unchecked, the compiler strips all Logger.Log code completely.
Color Mode: Choose whether only the header or the full message is colored.
Use checkboxes to enable/disable categories and assign colors to each one.

2.4 Your First Log

Add a simple log to any script:

```csharp
Logger.Log("Player spawned successfully", LogLevel.Info, "Gameplay");
```

- **Severity default:** If not provided, it defaults to Info.
- **Category default:** If not provided, it defaults to "General".

Example using defaults:

```
Logger.Log("This uses default severity (Info) and category (General)");
```

2.5 Press Play and Verify

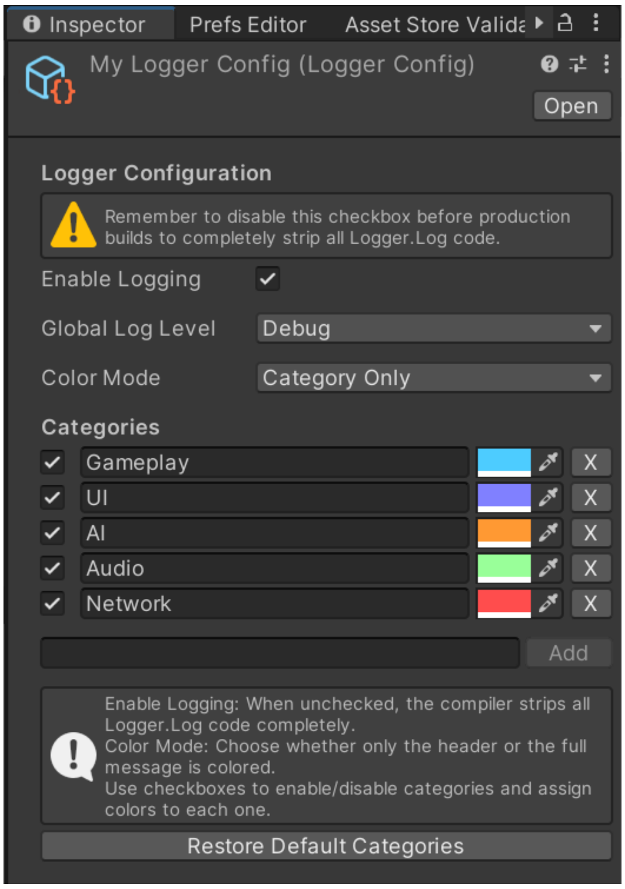
1. Hit **Play** in the Unity Editor.
2. Open the **Console window**:
   - You should see logs grouped by category and colored headers.
   - If no log is shown double-check that Enable Logging checkbox is enabled
   - If colors don't appear, enable **Rich Text** in Console settings.

Backbone Logger is now ready to use!

3. In-Depth Configuration

This section explains how to fully customize Backbone Logger for your project.
Everything is managed through the **LoggerConfig** asset.

3.1 LoggerConfig Inspector Overview



Logger Configuration

Remember to disable this checkbox before production builds to completely strip all Logger.Log code.

Enable Logging ✓

Global Log Level: Debug

Color Mode: Category Only

Categories

✓	Gameplay	🟦
✓	UI	🟪
✓	AI	🟧
✓	Audio	🟩
✓	Network	🟥

Enable Logging: When unchecked, the compiler strips all Logger.Log code completely.
Color Mode: Choose whether only the header or the full message is colored.
Use checkboxes to enable/disable categories and assign colors to each one.

Field	Description
Enable Logging (Compile-Time)	Global checkbox that determines if Logger.Log calls are included in builds. When unchecked, the compiler **strips all log code** completely for zero runtime cost.
Global Log Level	Filters logs by severity. Example: setting this to Warning hides all Debug and Info logs.
Color Mode	Header Only → Colors only [Level] [Category]. Header and Message → Colors the entire log line.
Categories List	Add, rename, enable/disable, and assign colors to each category. Disabled categories won’t show in the Console.

3.2 Log Levels

Each log has a **severity level**, which determines how it’s displayed and filtered.

Level	Use Case
Debug	Internal development details like variable states or AI decisions.
Info	Normal events such as scene loads or checkpoints. *(Default if not specified)*
Warning	Something unexpected happened, but the game can continue.
Error	A problem occurred and should be fixed, but the game still runs.
Critical	A serious failure requiring immediate attention.

3.3 Categories

Categories organize logs by **feature or system**.
Examples: Gameplay, UI, Network, AI.

- Each category can be toggled on/off in the inspector.
- Assign a **unique color** to make logs visually distinct.
- Add custom categories by clicking **Add** in the inspector.
- You may also populate the list with the default categories by pressing the ‘Restore Default Categories’ button**Runtime control:**

```
Logger.SetCategoryActive("AI", false); // Temporarily hide all AI Logs
```

3.4 Compile-Time Stripping

Backbone Logger uses Unity’s **Scripting Define Symbols** to completely remove logging code from builds.

- **Enabled:** Logs are included and filtered at runtime.

- **Disabled:** Calls to Logger.Log() are **removed at compile-time**, resulting in **zero CPU or memory cost**.

This is controlled by the **Enable Logging (Compile-Time)** checkbox at the top of the LoggerConfig inspector.

3.5 Advanced Code Examples

Adding a Custom Category at Runtime

```
Logger.AddCategory("Analytics", true, Color.cyan);
```

Changing the Global Log Level Dynamically

```
 Logger.GlobalLevel = LogLevel.Warning;
// Only Warning, Error, and Critical logs will appear now
```

Minimal Logging Example

```
// Defaults: Severity = Info, Category = "General"
Logger.Log("Simple message with defaults");
```

3.6 Best Practices

- Keep categories **minimal and meaningful**.
- Use **Debug** logs only during development.
- Always **disable** Enable Logging (Compile-Time) before production builds.
- Create separate LoggerConfig assets for **Development**, **QA**, and **Release** setups.