



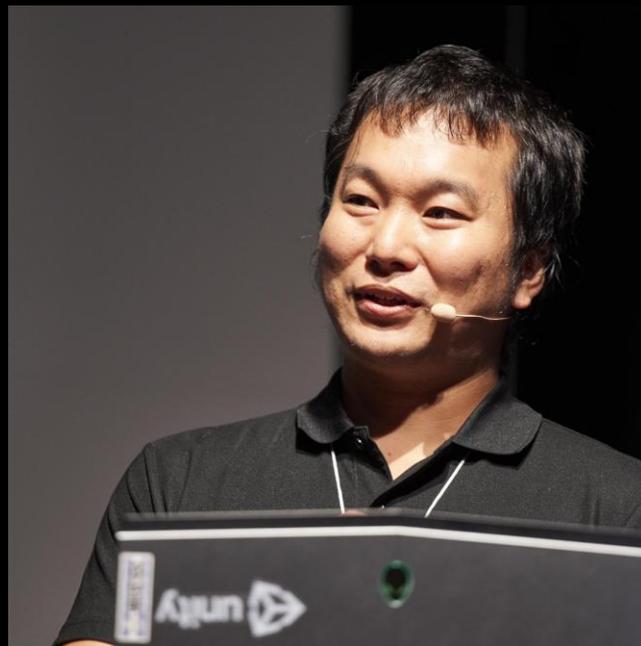
# Unity Source Code Quick Guide

## - ダウンロードから内部デバッグまで



# 黒河 優介

Partner Engineer





# Agenda

- ソースコードについて
- ビルドする方法
- 実際にデバッグする
  - Windows Editor
  - Windows RUntime
  - Mac Editor
  - iOS
  - Android
- デバッグ・調査のために書き換える



# ソースコードについて



## ソースコードで出来る事？

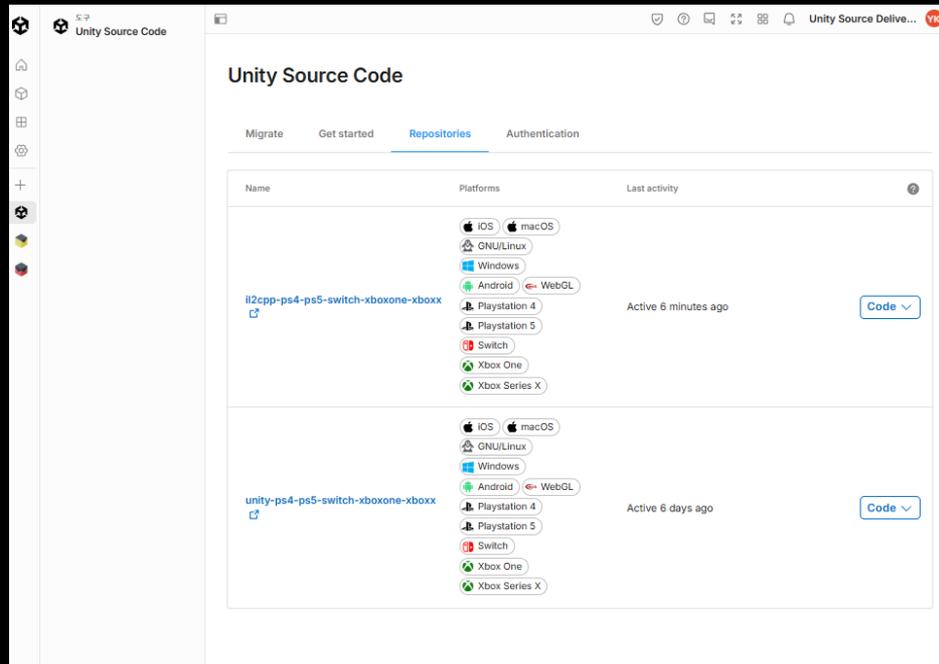
UnityのRuntime/Editorをビルド可能なC++/C#コードにアクセスすることが出来ます。

- ・ 開示されていない処理の詳細について知ることが出来る
  - バージョンアップグレード時に発生する異なる挙動について調査可能
  - > ツール制作にも活かすことが可能
  - エラーログの文字列でGrepして、原因について調査可能
  - Profilerの重い箇所を文字列で該当処理の詳細が調査可能
- ・ Unityそのものをデバッグする
  - Visual Studio / Xcodeで、BreakPointを利用したりしてデバッグする
  - 手元のプロジェクトデータからビルドされたものを使ってデバッグする事が出来ます。
  - ソースコードにログ処理やProfiler処理を追加しながらデバッグする



## ソースコードへのアクセス

- Unity Enterpriseライセンスが必要
  - Enterpriseライセンスの後、個別のアクティベートが必要
  - 通常はコンソール機のソースはアクセス不可
- Unity Dashボードよりアクセス可能
- 利用はあくまで閲覧・デバッグ用
  - 改変したUnityで製品リリースするには別途契約が必要
- gitクライアントでアクセス可能





## gitのインターフェース

- Unityダッシュボード上でSSHキーを登録する事でgitクライアントでのダウンロードが可能
  - ed25519のSSHキーが利用可能
- Unityダッシュボードからリンクされた形でGitのWeb UIインターフェースへアクセス可能
- リリースされたバージョン毎にTagがある

The screenshot shows a Git repository interface for 'unity-source-code'. The repository is public and has 45 commits, 35 branches, and 1,102 tags. The current commit is 6000.0.40f1. The file tree shows a directory structure with folders like .fork, .sparse-checkout-rules, Configuration, Documentation, Editor, Extensions, External, Modules, Packages, PlatformDependent, Platforms, Projects, Runtime, Shaders, and Tests. The commit history shows that all files were imported by Source control team. The right sidebar shows the repository name, description, Readme, size (341 GiB), and language statistics.

File	Author	Commit	Time
Copybara Bot	307ba2a531	Code imported by Source control team	2주 전
.fork	Code imported by Source control team		11개월 전
.sparse-checkout-rules	Code imported by Source control team		2개월 전
Configuration	Code imported by Source control team		2주 전
Documentation	Code imported by Source control team		2주 전
Editor	Code imported by Source control team		2주 전
Extensions	Code imported by Source control team		9개월 전
External	Code imported by Source control team		2주 전
Modules	Code imported by Source control team		2주 전
Packages	Code imported by Source control team		2주 전
PlatformDependent	Code imported by Source control team		2주 전
Platforms	Code imported by Source control team		지난달
Projects	Code imported by Source control team		4개월 전
Runtime	Code imported by Source control team		2주 전
Shaders	Code imported by Source control team		6개월 전
Tests	Code imported by Source control team		2주 전

**Languages**

- C# 56.3%
- C++ 35.7%
- C 2.1%
- HLSL 2.1%
- Objective-C++ 0.8%
- 기타 2.8%

```
Unity@DESKTOP-G30HB4E MINGW64 /d
$ ssh-keygen -t ed25519 -C "test@example.com" -f ./id_ed25519
```





# ビルドする方法



## ビルドに必要なソフトウェア

### 共通

- git & git LFS(ダウンロード用)

### Windows

- Visual Studio C++
- Perl ( Strawberry Perl 5.28+)
- Windows SDK

### Mac OS

- Perl
- XCode
- mac OS SDK

(ソフトウェアのバージョンはビルドしたい  
Unityのバージョンによって異なる)

Unity version	Visual Studio	MSVC	Windows 10 SDK
6000.0	2022 17.9.3	14.39.33519	10.0.20348.0
2023.2	2022 17.4.2	14.34.31933	10.0.20348.0
2022.3	2022 17.4.2 + 2019 16.8.1	14.28.29333 (from VS2019)	10.0.20348.0
2022.2	2019 16.8.1	14.28.29333	10.0.20348.0
2022.1	2019 16.8.1	14.28.29333	10.0.20348.0
2021.3	2019 16.8.1	14.28.29333	10.0.18362.0
2021.2	2019 16.8.1	14.28.29333	10.0.18362.0
2021.1	2019 16.5.4	14.25	10.0.18362.0
2020.3	2019 16.5.4	14.25	10.0.18362.0
2020.2	2019 16.5.4	14.25	10.0.18362.0
2020.1	2019 16.2	14.22	10.0.17134.0
2019.4	2017 15.9	14.16	10.0.17134.0

リポジトリ  
Documentation/InternalDocs/docs/BuildSystem/SetupWindows.md  
より



## まずはコマンドラインから

- まずはコマンドラインから始まります
  - ドキュメントを閲覧するのもコマンド
    - Pythonが必要
  - ビルド用のコマンド
    - Visual Studio/Xcodeのプロジェクトファイルはビルドのコマンドから生成
    - ビルドもコマンドだけで出来る



## ドキュメント閲覧のコマンド

以下のコマンドを実行すると、ブラウザが立ち上がりドキュメントを閲覧できます  
(Pythonが必要)

### Windows

- `Documentation/InternalDoc/view.cmd`

### Mac

- `sh Documentation/InternalDoc/view.sh`

127.0.0.1:8006/General/basics.html

# Editor and Runtime Development Guide

## Basic overview of Unity Editor and Runtime development

**Important**

This guide is pulled together from various internal docs sources, most of which are under development to some extent. The content is maintained collaboratively by Unity Editor, Runtime, and Ecosystem (LRE) and Engineering Services (ES) teams. The markdown source is in the `Documentation/InternalDocs` directory of the [Unity repository](#). See the [README](#) there for guidance on contributing. Bug reports and content suggestions are welcome in [devs-internal-docs](#).

This page provides a general overview of the key components of Unity Editor and Runtime development. For more detail about using these tools to deliver a change, see the [Deliver your changes](#).

### Our tooling

#### Git and GitHub for source control

The source code for the Unity Editor and Engine is stored in the [unity/unity repository](#) in a Unity-specific instance of GitHub Enterprise. Only Unity employees and contractors with appropriate permissions can access the repository.

#### Yamato for continuous integration

[Yamato](#) is our in-house CI system in which we run automated build and test workloads. Yamato is Git-native and Jobs are defined in YAML. Most of the test configurations - sometimes referred to as "builders" for legacy reasons - are standardised, and developers are expected to run appropriate jobs on their PR to test it.

#### Table of contents

- Our tooling
  - [Git and GitHub for source control](#)
  - [Yamato for continuous integration](#)
  - [Bokken for device provisioning](#)
  - [Evergreen merge queue to manage PRs to mainlines](#)





## ビルドコマンド

ビルドのコマンドは大きく二種類

- `perl build.pl`
  - 旧来の方式
  - プロンプトでビルドターゲットが選択可能
- `jam` コマンド
  - 基本はコチラを推奨
  - "`jam WinEditor`" のように引数でビルド対象を設定
  - 引数がない場合、引数なしで実行する事で候補が出力される
  - "`jam WinEditor --help`" のようにプラットフォーム+"--help"でプラットフォームごとの更なるオプションを表示



## Visual Studio / Xcodeプロジェクトもビルドで生成

- jam ProjectFiles
  - VisualStudioやXcodeなどのプロジェクトファイル生成するコマンド
- Projects以下にVisualStudioやXcodeのプロジェクトが生成される
- 生成されたプロジェクトファイルを利用することで、デバッグが可能



## よく使うjamのビルドコマンド(1)

- jam WinEditor
  - WindowsのUnityEditorをビルド (Debugビルド)
- jam WinEditor -sCONFIG=release
  - WindowsのUnityEditorをビルド (Releaseビルド)
- jam MacEditor
  - MacのUnityEditorをビルド (x86)
- jam MacEditorArm64
  - MacのUnityEditorをビルド (Apple Silicon)



## よく使う**jam**のビルドコマンド**(2)**

- jam WinPlayer
  - Windows Standalone Player (Mono only)
- jam AndroidPlayer
  - Android Support ( Mono only)
- jam AndroidPlayer64IL2CPP
  - Android Support( IL2CPP Arm8)
- jam iOSPlayer
  - iOS Support



ここまでの流れを  
ビデオで

ソースコードへのアクセスとダウンロード



# 実際にデバッグする (Windows Editor)

WindowsEditorデバッグ  
(デモ動画)



# 実際にデバッグする (Windows Runtime)

WindowsRuntimeデバッグ  
(デモ動画)



# 実際にデバッグする (Mac Editor)

MacEditor Debug  
(デモ動画)



# 実際にデバッグする (Android Runtime)

Androidでのデバッグ  
(デモ動画)



# 実際にデバッグする (iOS Runtime)

# iOS Debug (デモ動画)

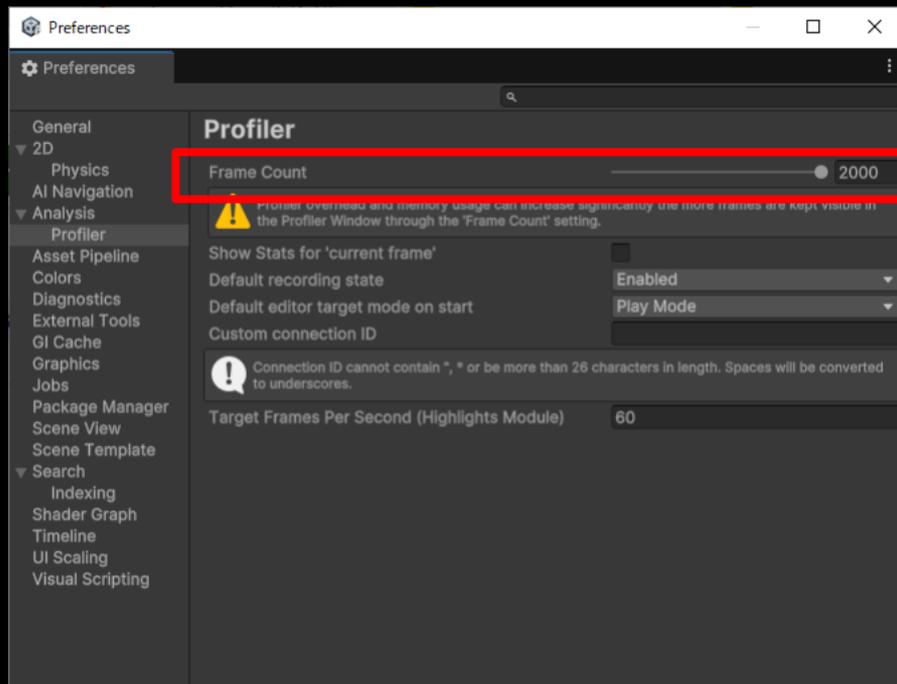


デバッグ・調査のために  
書き換える  
- Part1 -



## 書き換えのシチュエーション (1)

Profilerのキャプチャーフレーム上限を2000以上にしたい。





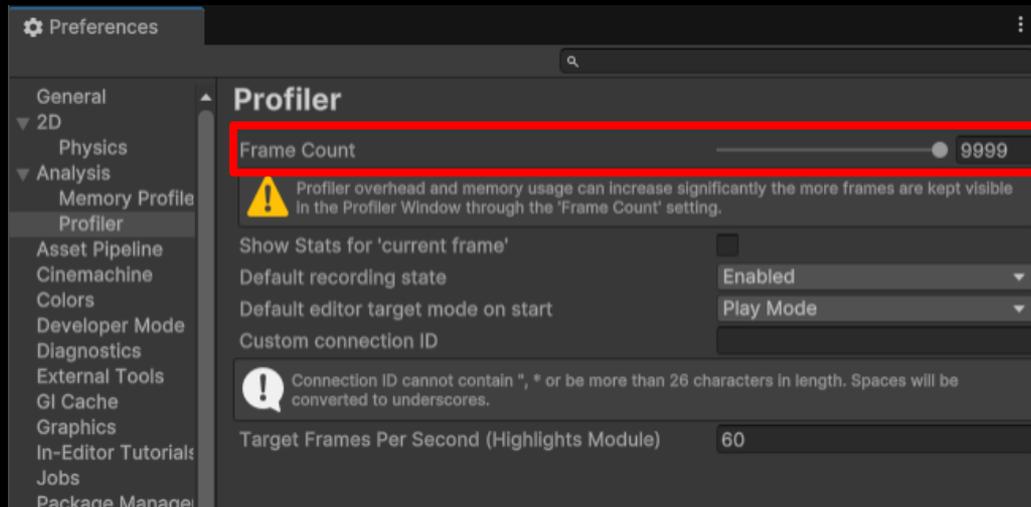
## Profilerのキャプチャーフレーム上限

```
ProfilerSettings.cs  X
UnityEditor  UnityEditor.Profiling.ProfilerUserSettings
24 public const string k_TargetFramesPerSecond = k_Settings
25
26 public const int kMinFrameCount = 300;
27 public const int kMaxFrameCount = 2000;
28 public const int k_MinimumTargetFramesPerSecond = 1;
29 public const int k_MaximumTargetFramesPerSecond = 1000;
30
31 private const int kMaxCustomIDLength = 26;
32
33 [SerializeField]
34 private static int m_FrameCount = 0;
35
```

“Modules/ProfilerEditor/Public/ProfilerSettings.cs”  
の kMaxFrameCount の値を書き換えてビルド。



## Profilerのキャプチャーフレーム上限



2000から 9999に変更してビルド。これで 9999フレームまでキャプチャ可能に

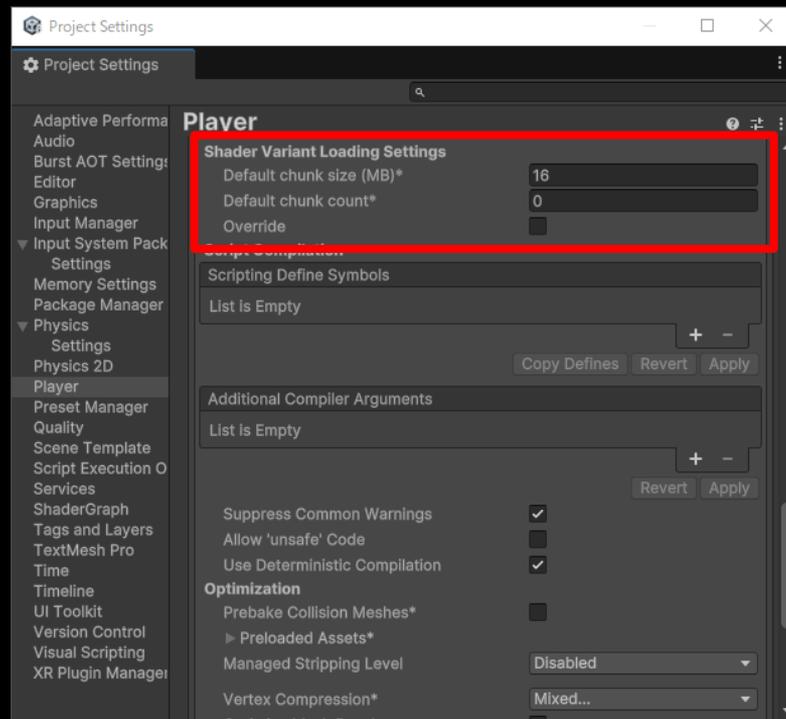


デバッグ・調査のために  
書き換える  
- Part2 -



## 書き換えのシチュエーション (2)

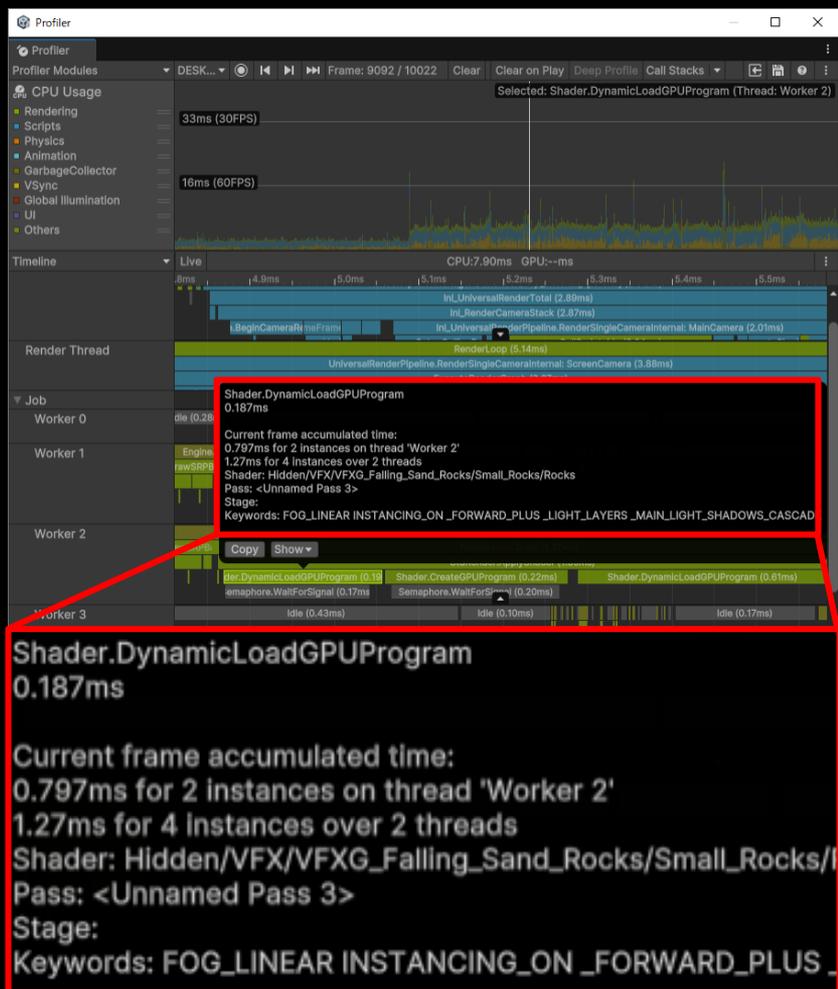
ProjectSettings.SetDefaultShaderChunkCount の効果を、  
もっと詳しく知りたい  
Chunkの展開処理や破棄処理をProfilerで可視化したい





# SetDefaultShaderChunkCountを簡単に説明

- ShaderBinaryDataはChunkに分割されてシリアルライズされる
- それぞれのChunkはLz4圧縮されている
- Shaderがロードされたとき、全てのChunkは圧縮された状態でメモリ上にロードされる
  - SetDefaultShaderChunkCountが0以外の時
- 必要に応じてLz4圧縮されたChunkを解凍して、メモリ上に展開される
  - ProfilerのShader.DynamicLoadGPUProgramにカウントされる
- ChunkCount数が指定された以上になると、展開後のメモリは破棄される





## 今回、書き換えたい事はつまりこういう事

背景：

デフォルト状態では、"Shader.DynamicLoadGPUProgram"で、Chunk展開リクエストが発生したことがわかるが、具体的に何個目のChunkが展開されたかわからない。

Chunkの展開処理だけ知りたいChunkを捨てたかわからない。

今回は以下の二つの処理を追加する

- ・ Chunk展開した処理をProfilerに載せる
- ・ Chunkを捨てた処理もProfilerに載せる



## Code example

### Profilerの処理の書き方@EngineCode(C++)

```
1 // PROFILER_MAKERマクロを使って、ProfilerMakerをGlobal変数として宣言
2 PROFILER_MARKER(
3     gShaderDynamicLoadGPUProgram, /* ← アクセスのためのGlobal変数 */
4     kProfilerRender,              /* ← カテゴリ */
5     "Shader.DynamicLoadGPUProgram", /* ← Profilerの名前 */
6     /* 以下がMetadataの型と名前を宣言 */
7     core::string, "Shader", core::string, "Pass",
8     core::string, "Stage", core::string, "Keywords");
9
10
11 // 実際に計測したい時は PROFILER_AUTOマクロでScopeを自動的に計測
12 // ConstructorでBegin、DestructorでEndするようなインスタンスを自動で作ってくれる
13 void SubProgram::Compile(CompilationData& data){
14 #if ENABLE_PROFILER // 前に処理があるけど省略
15     // ここから、関数が終わるまでの間を自動で計測します
16     PROFILER_AUTO(gShaderCreateGPUProgram,
17                 /* 以下メタデータ */
18                 data.fromShader->GetScriptClassName(), passName, *stageDesc, variant);
19 #endif
20 }
21
22
23
24
25
```



# Let's Demo

DynamicShaderLoadに詳細情報を追加するカスタム  
( デモ動画 )



# まとめ

- UnitySourceCodeはEnterpriseライセンスのお客様はアクセス可能
- 読む以外にも、デバッグを通して内部処理の知識を深める事が出来る
- 実プロジェクトデータを使ってエンジンのデバッグも可能



Thank you

