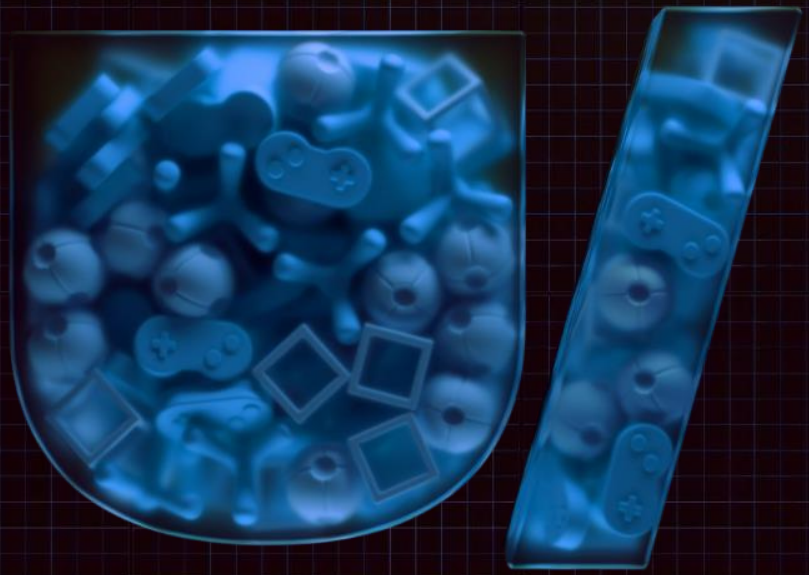
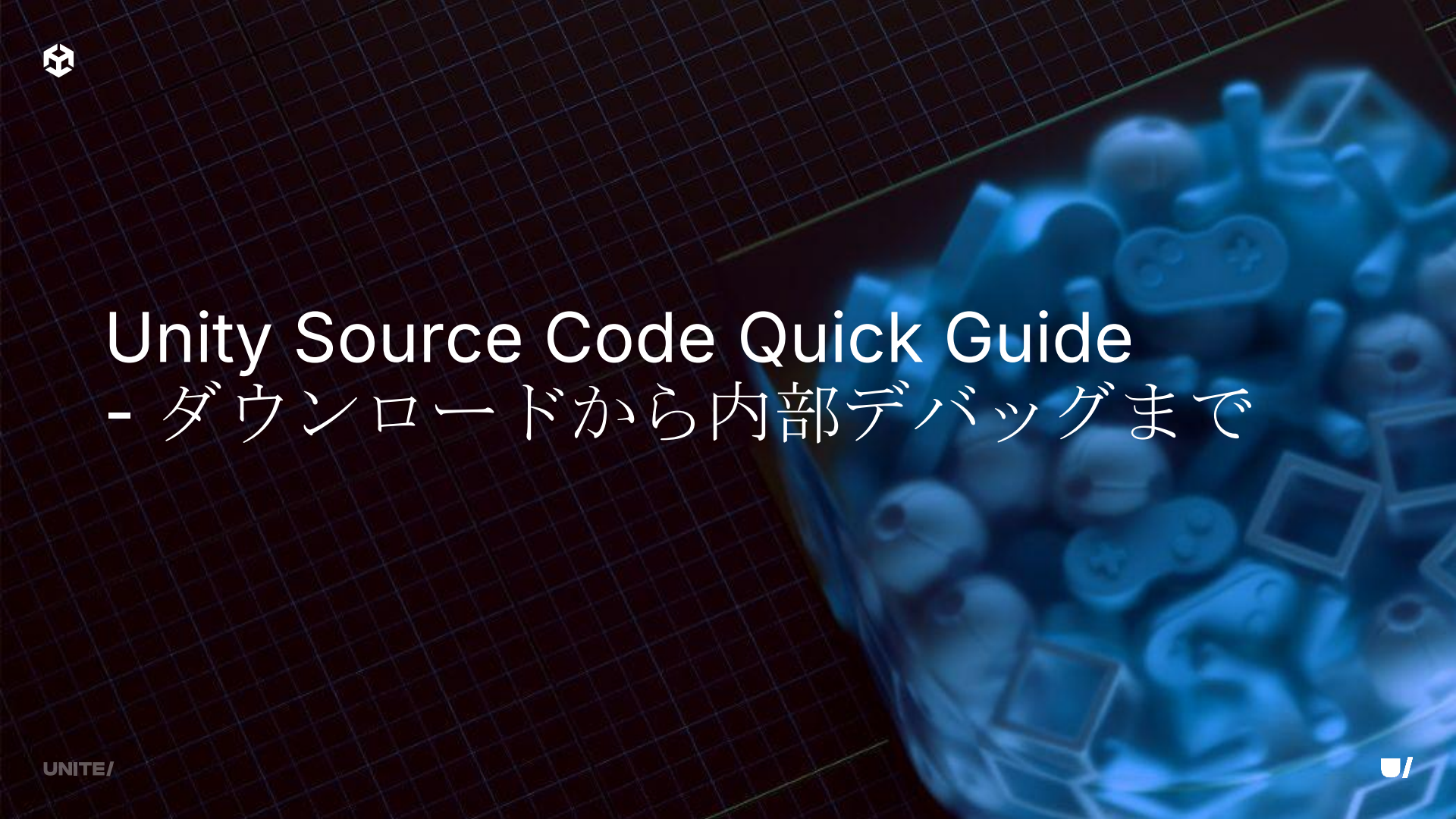



Unity Source Code Quick Guide
- ダウンロードから内部デバッグまで
UNITE/

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？

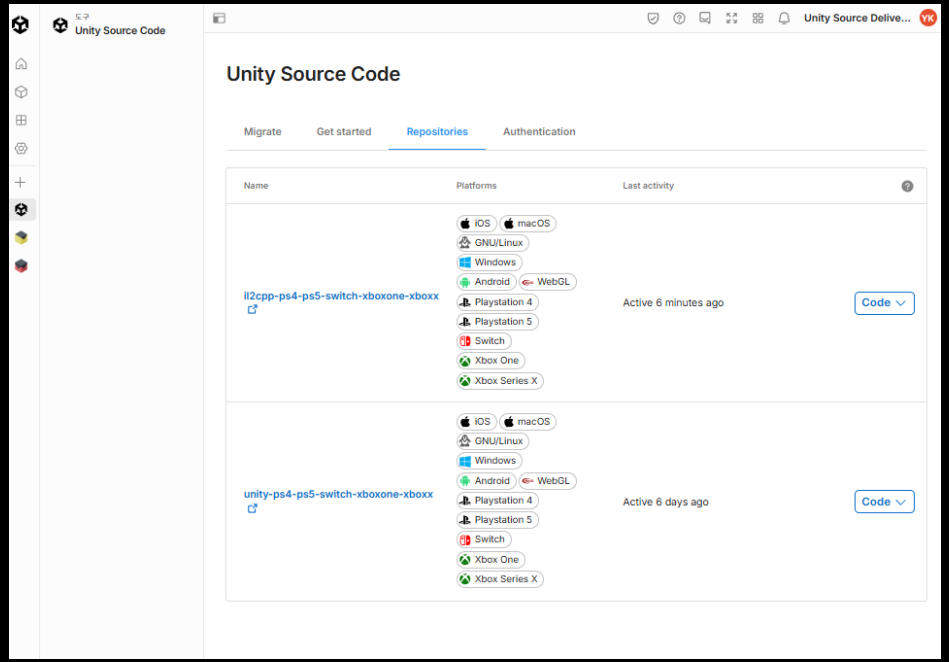
UnityのRuntime/Editorをビルド可能なC++/C#コードにアクセスすることが出来ます。

・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

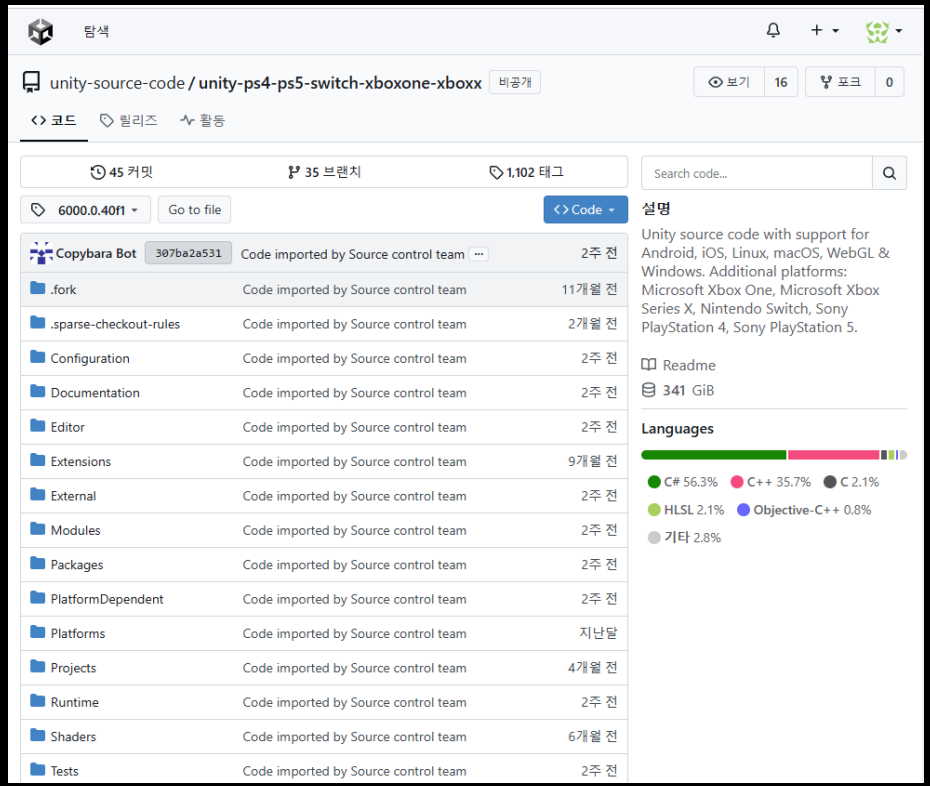
ソースコードへのアクセス

- → 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




```
Unity@DESKTOP-G3OHB4E 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

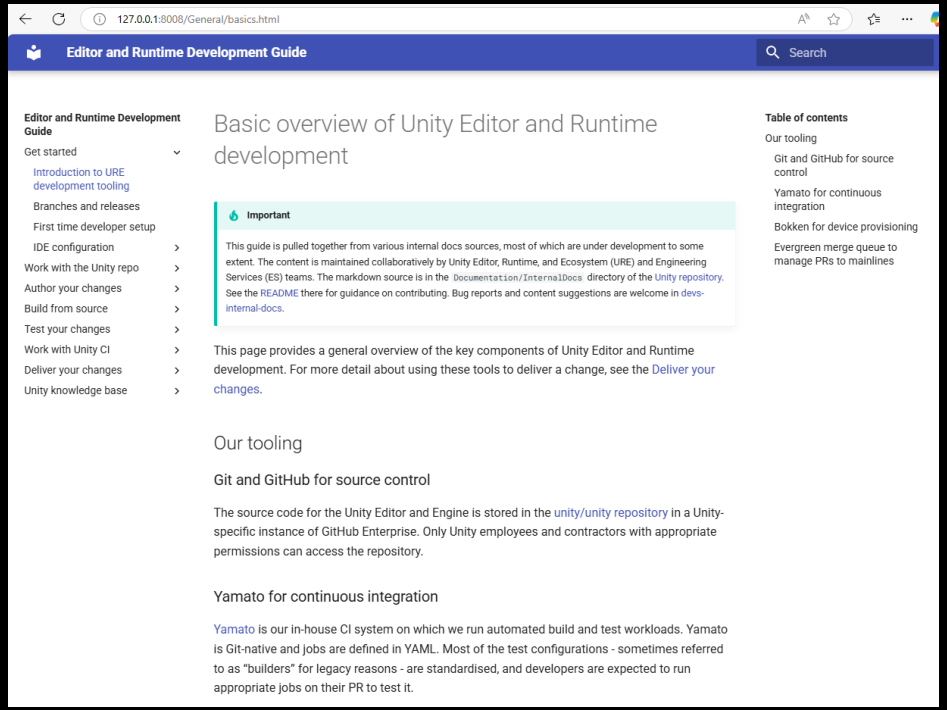
→ まずはコマンドラインから始まります
- ドキュメントを閲覧するのもコマンド
  - Pythonが必要
- ビルド用のコマンド
  - Visual Studio/Xcodeのプロジェクトファイルはビルドのコマンドから生成
  - ビルドもコマンドだけで出来る

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



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
  - 引数がわからない場合、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)

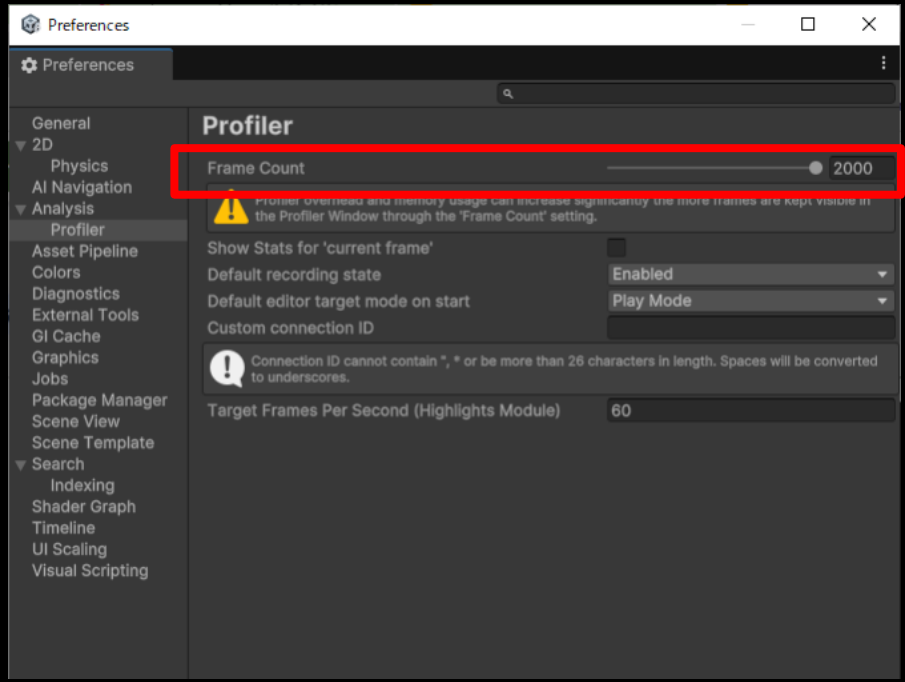
実際にデバッグする (iOS Runtime)

iOS Debug
(デモ動画)

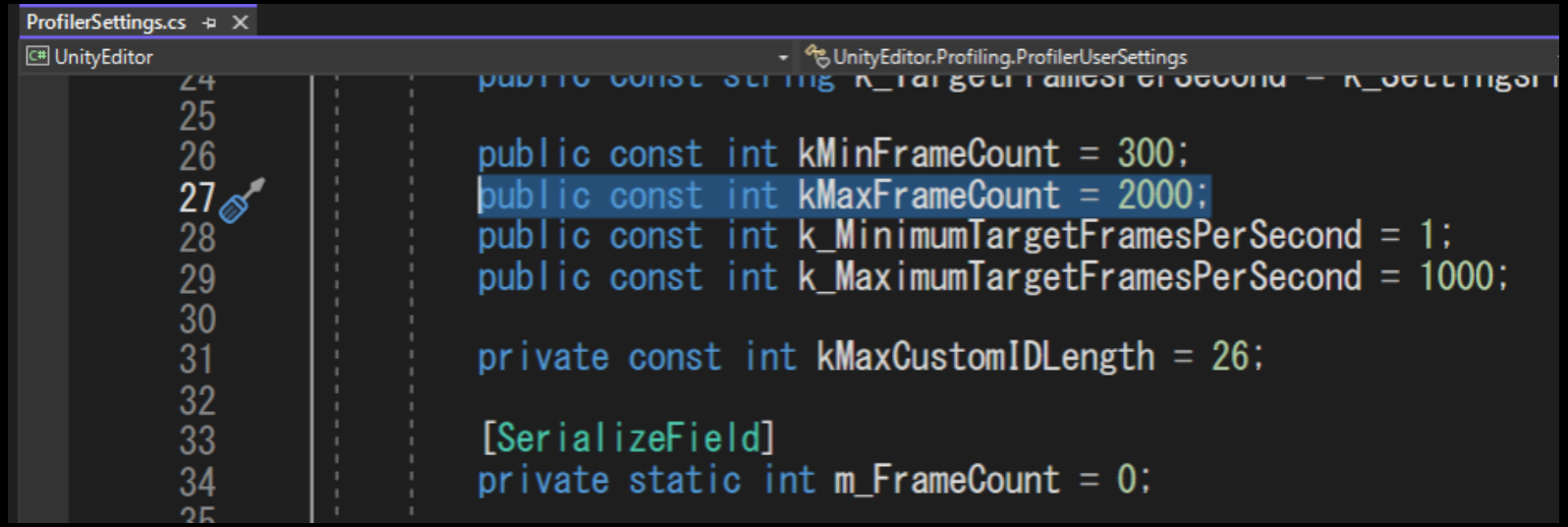
デバッグ・調査のために書き換える - Part1 -

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。

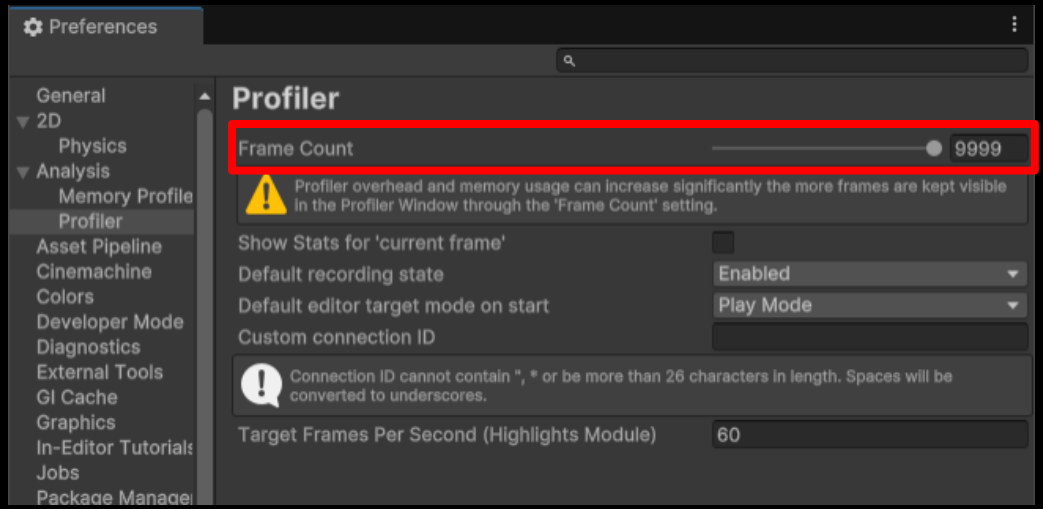


Profilerのキャプチャーフレーム上限



“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

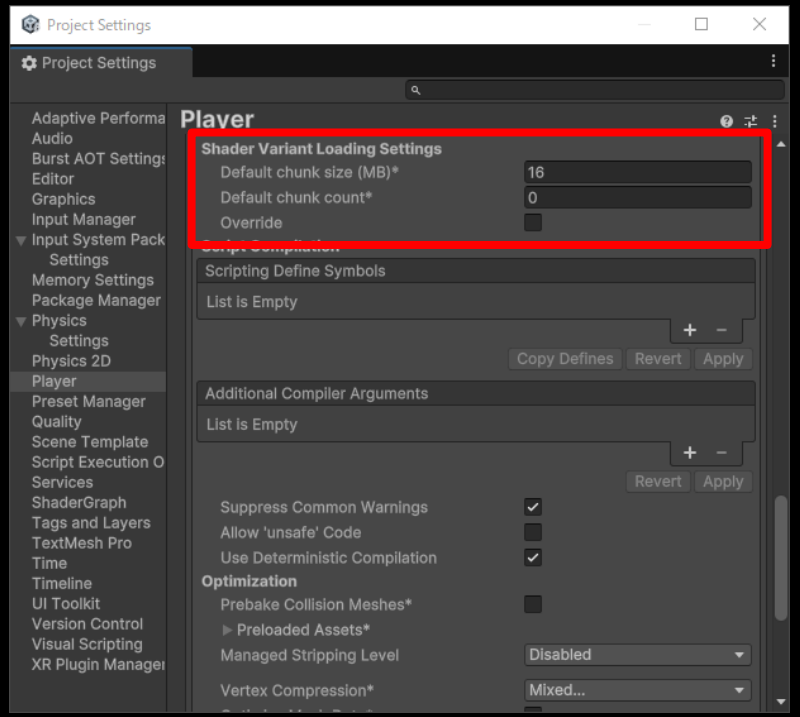


2000から 9999に変更してビルド。これで 9999フレームまでキャプチャ可能に

デバッグ・調査のために書き換える
- Part2 -

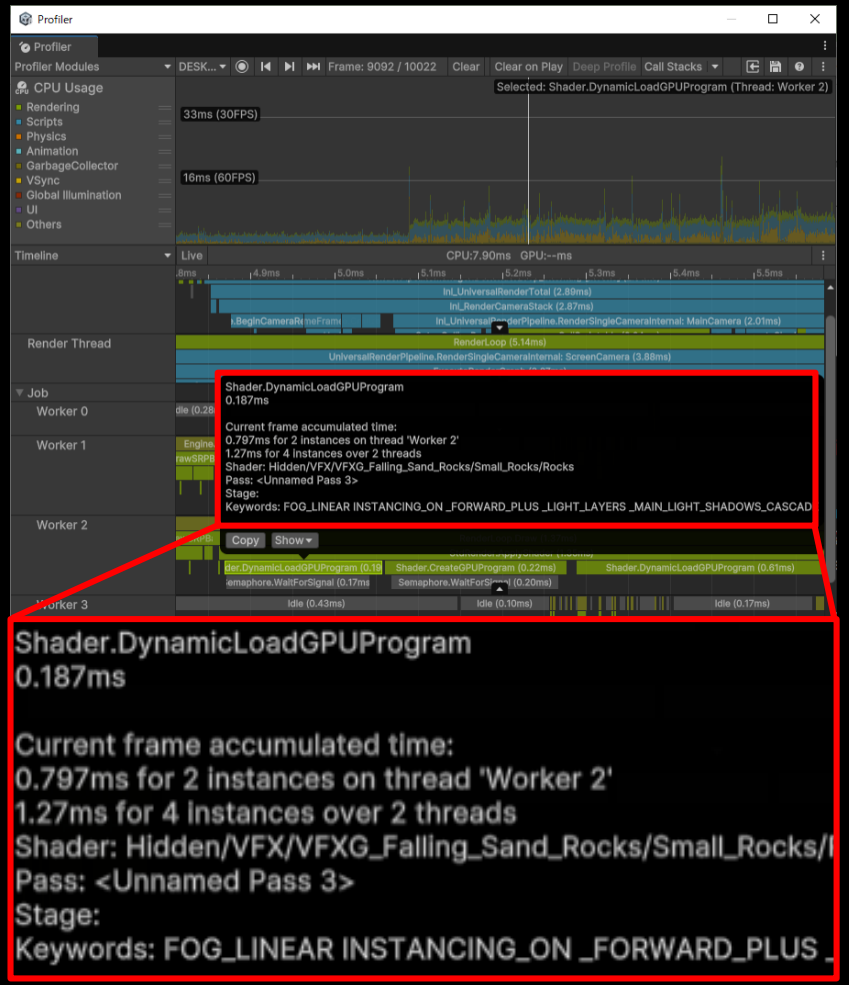
書き換えのシチュエーション（2）

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++)

```
// PROFILER_MAKERマクロを使って、ProfilerMakerをGlobal変数として宣言
PROFILER_MARKER(
       gShaderDynamicLoadGPUProgram,  /* <- アクセスのためのGlobal変数 */
             kProfilerRender,                /* <- カテゴリ */
      "Shader.DynamicLoadGPUProgram", /* <- Profilerの名前 */
      /* 以下がMetadataの型と名前を宣言 */
      core::string, "Shader", core::string, "Pass",
      core::string, "Stage", core::string, "Keywords");


// 実際に計測したい時は PROFILER_AUTOマクロでScopeを自動的に計測
// ConstructorでBegin、DestructerでEndするようなインスタンスを自動で作ってくれる
void SubProgram::Compile(CompilationData& data){
#if ENABLE_PROFILER // 前に処理があるけど‥省略
    // ここから、関数が終わるまでの間を自動で計測します
    PROFILER_AUTO(gShaderCreateGPUProgram,
                  /* 以下メタデータ */
                  data.fromShader->GetScriptClassName(), passName, *stageDesc, variant);
#endif
}
```

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