Things I've stumbled upon while developing for consoles

JACK KNOBEL – SEPTEMBER MELBOURNE UE4 MEETUP

Who Am I?

Current:

Gameplay Programmer at *Beethoven and Dinosaur* working on *Artful Escape*.

Previously:

Programmer & Xbox One Lead* - Bards Tale 4

Gameplay Programmer – Trover Saves the Universe

Getting Started with Console Development

Getting Started with Console Development

- Apply for approval from each console owner
  - Varies from console to console (Switch you have to email an application with a pitch or meet a Nintendo representative)

- For Unreal fill out the form here: https://epicgames.secure.force.com/Forms/FormConsoleAccessRequest
  and checkout the platform portals as they may have a "Automatically Email Epic and say you're approved" button

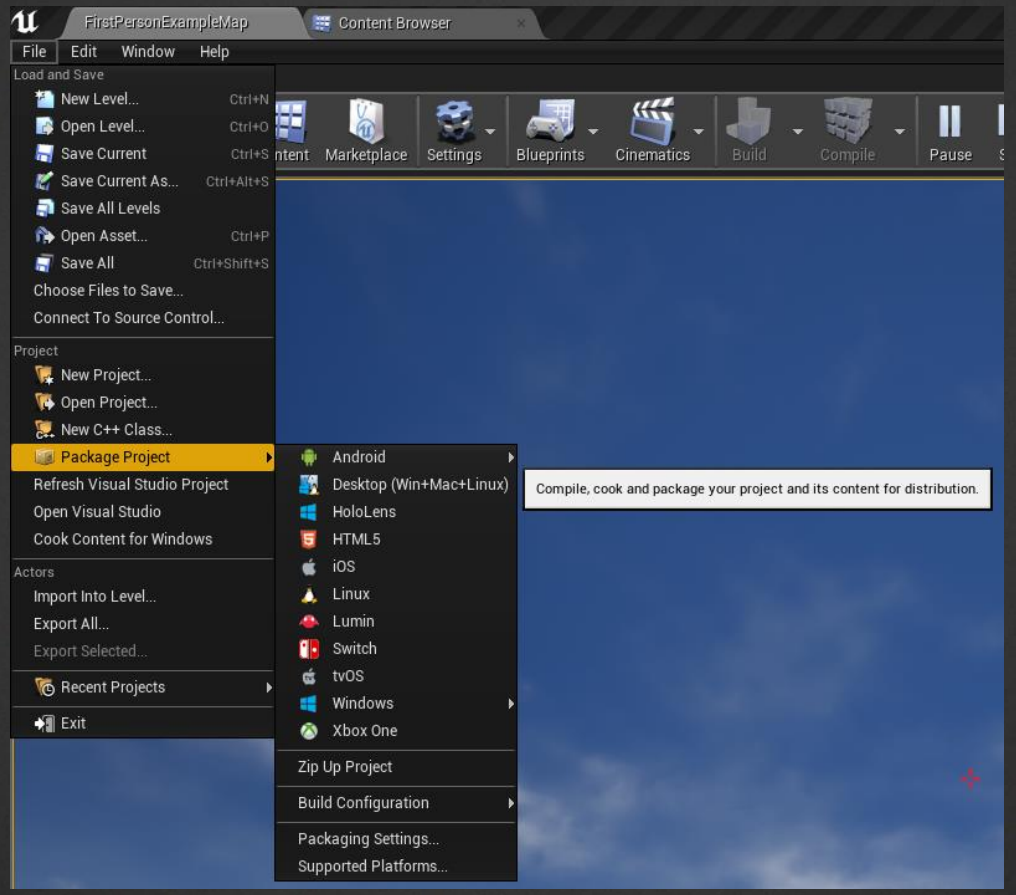
Console Development in Unreal

- Download the source version of UE4 from Github: https://github.com/EpicGames/UnrealEngine
  - Be sure to link your Unreal account with your GitHub account
- Login to the Epic FTP with the details you should have been provided (once approved) and download the Platform API for each platform.
- Follow the instructions in each platforms extension

Packaging


PACKAGING
IN THE EDITOR
PACKAGING
VIA
COMMANDLINE
PACKAGING
FROM
UNREAL FRONTEND
USING BUILDGRAPH
ON A DEDICATED
BUILD MACHINE

Packing: Editor



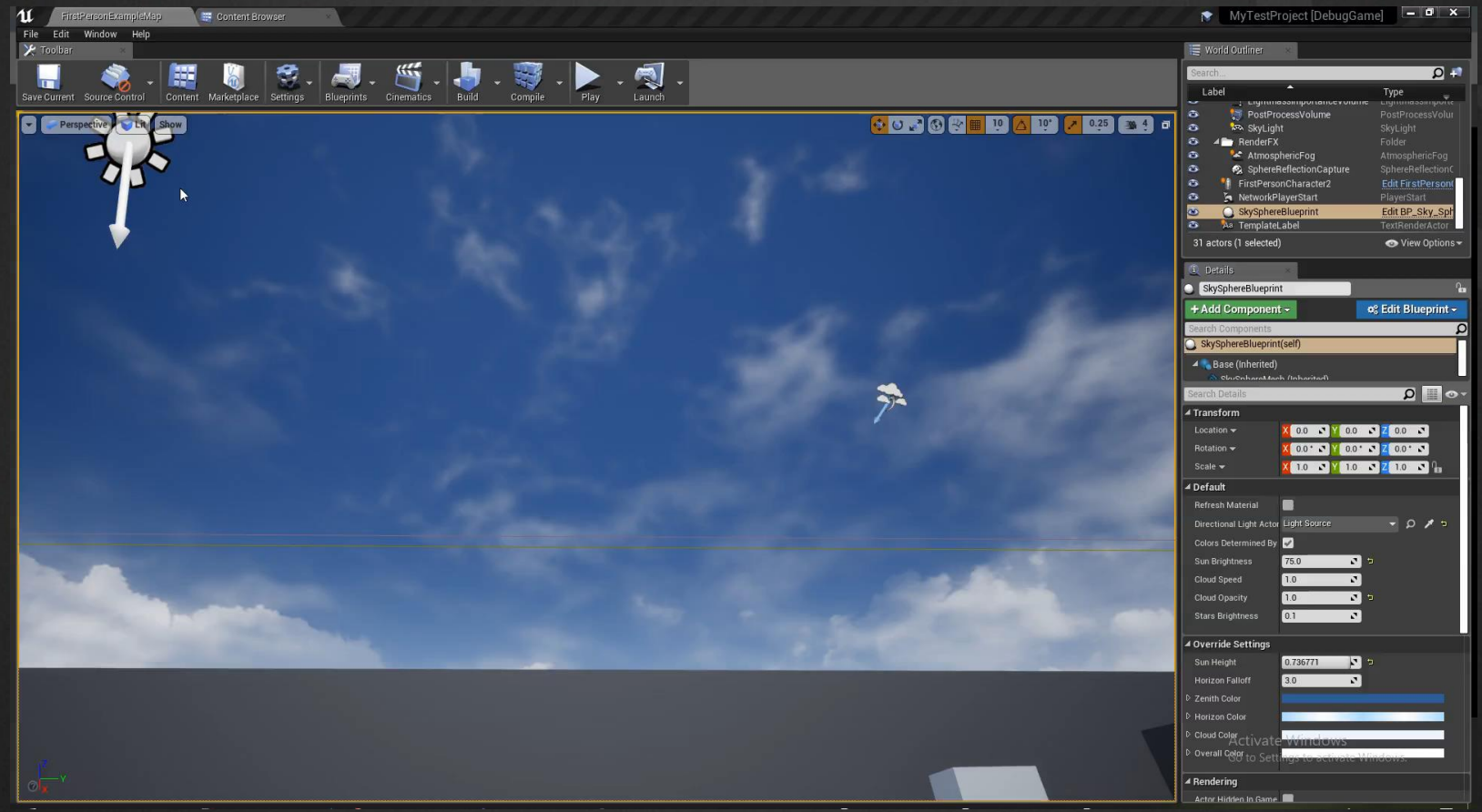
Packaging with the Project Launcher

What's…. the Project Launcher?

YOUR NEW BEST FRIEND!

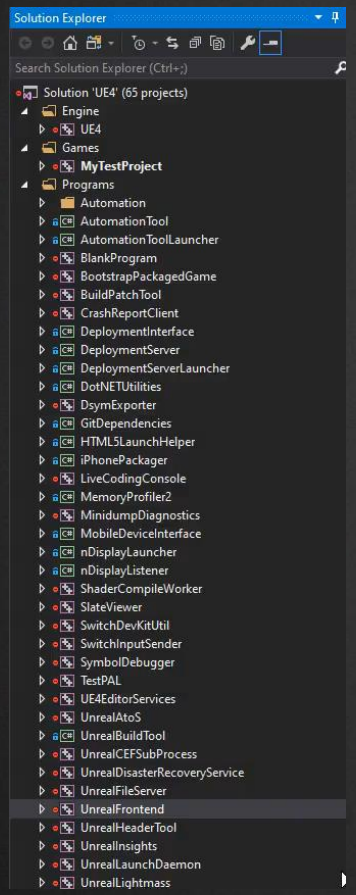
Project Launcher

In Editor



UNREAL ENGINE

Unreal Frontend
Visual Studio



- Exported to /Engine/Binaries/Win64/UnrealFrontend.exe
- Pin it to your start menu

Unreal Frontend

YOUR ONE STOP SHOP FOR ALL THINGS DEVELOPMENT

Packaging: Command line

RunUAT.bat BuildCookRun
-Project=<.UProjectPath>
-Platform=<TargetPlatform> (XboxOne, Win64 etc)
-Config=<BuildConfig> (Development, Shipping etc)
-Build
-Cook
-Stage
-StagingDirectory=<OutputDirectory>

Packaging: Command line

Additional arguments to consider

- Ignore list of maps and cook specific maps
  -Map=Map1;Map2;Map3;

- ***-cmdline=<Arguments>*** to have arguments automatically be applied on package boot
  - Appends the commandline arguments to the UE4Commandline.txt file in the package root

Improving your Iteration Times

Shared Derived Data Cache (DDC)

- Used to store versions of assets in the formats used by UE for your target platforms.
  (Where all your compiled shaders are stored.)

- Configured by modifying the [DerivedDataBackendGraph] section in the DefaultEngine.ini file in your project's config folder

OR

- Setting a windows environment variable UE-SharedDataCachePath to point to the network location

Fill your DDC ahead of time

UE4\Engine\Binaries\Win64\UE4Editor.exe ProjectName -run=DerivedDataCache -fill

HOT TIP

Cooking Tips

REDUCING YOUR TIME TO COOK

Cooking Tips
Settings to improve your cook times

- In DefaultEditor.ini
  [CookSettings]
  MaxMemoryAllowance=<AmountInMb>

- In DefaultEngine.ini
  [DevOptions.Shaders]
  NumUnusedShaderCompilingThreads=<Threads Free>

Arguments to consider passing to the cooker

- -iterative
- -iterativedeploy
- -fastcook

Cooking Tips
Modify the cook your way

- FCookModificationDelegate (GameDelegates.h)
- AssetManager
  - ModifyCook
  - ShouldCookForPlatform
  - And many more
- UObject::NeedsLoadForTargetPlatform
  - Allows for the object to be skipped during cook (Objects can be excluded on a per platform level!)

Shopping for a new cooking PC?

HOT TIP

Cooking ***does not*** use the GPU

Saving and serializing is single threaded, focus on higher single threaded performance rather then more cores

Improving Compile Times

Improving Compile Times
<GameName>.build.cs

- Make sure you're using IWYU
  PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
  - on by default for new projects, might be off for older ones

- bFasterWithoutUnity = true
  - Much faster iteration times
  - Slower rebuild times
  - Will also help verify classes have the right includes*

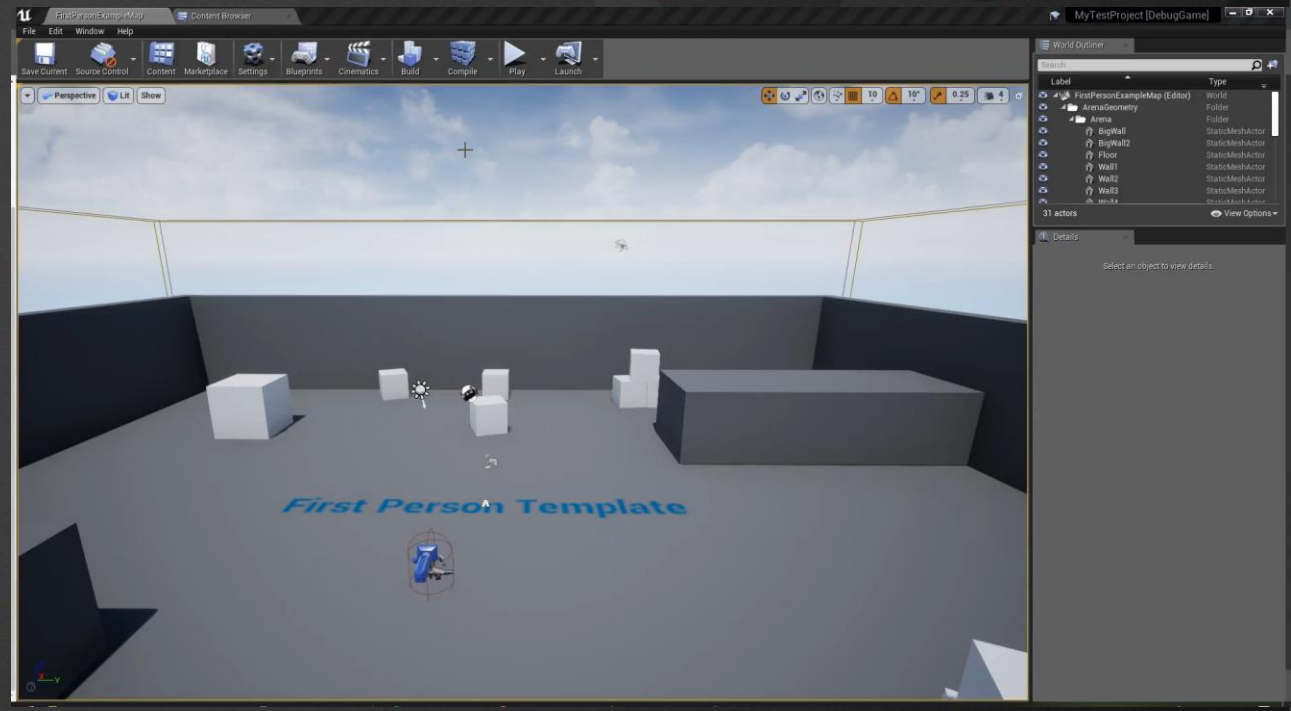
Improving Compile Times
General

- Reduce includes in your header files
  - Use forward declarations
  - Consider using the PIMPL pattern in some scenarios when using monolithic headers
- Consider breaking your game up into more modules/plugins

Improving Compile Times

Live Coding (Live++ in Unreal)

- Like hot reload but not terrible
- Available in 4.22+
- Can update code in running games
  - Includes packaged games!



Platform Specific Improvements

PS4

- Make use of SN-DBS! It's **FREE**
- Allows for distributed compiling of game source code
- Can also do distributed compilation of shaders with a community made patch on UDN

XboxOne

- Network shares should be your best friend for mass deployment
- [Redacted]
- [Redacted]
  - [Redacted]
- Can't dive into specifics because NDA..
  - Talk to me afterwards if you want to know more about XboxOne deployment!

Automation

Automation: BuildGraph

- Script-based build system available in > 4.13
- Very similar to makefiles or Jenkins pipeline scripts
- Written in an XML doc
- Checkout the Unreal Documentation to get started https://docs.unrealengine.com/en-US/Programming/BuildTools/AutomationTool/BuildGraph/index.html

Automation: Gauntlet

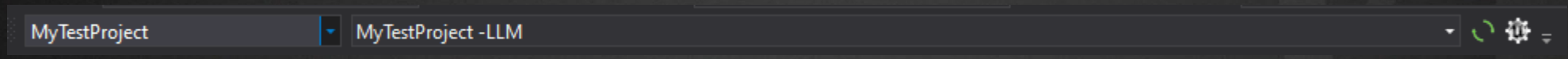
- Added in 4.21
- New framework for running builds and validating results
  - Doesn't require specific Unreal-side code
  - Platform independent script implementation
- Examples available in ActionRPG and EffectsDemo

Launching

PIE vs Standalone vs Packaged

- **PIE:** Play In Editor
- **Standalone:** Runs game exe using editor for content delivery
  - Closer representation of a packaged game then PIE
    - Still has Editor overhead
    - **Does not** use cooked content
    - *WITH_EDITOR* still true in this context
  - *-game* as a start-up arg, in editor or right clicking on UProjectFile->LaunchGame
- **Packaged Game:** … well you've packaged it up

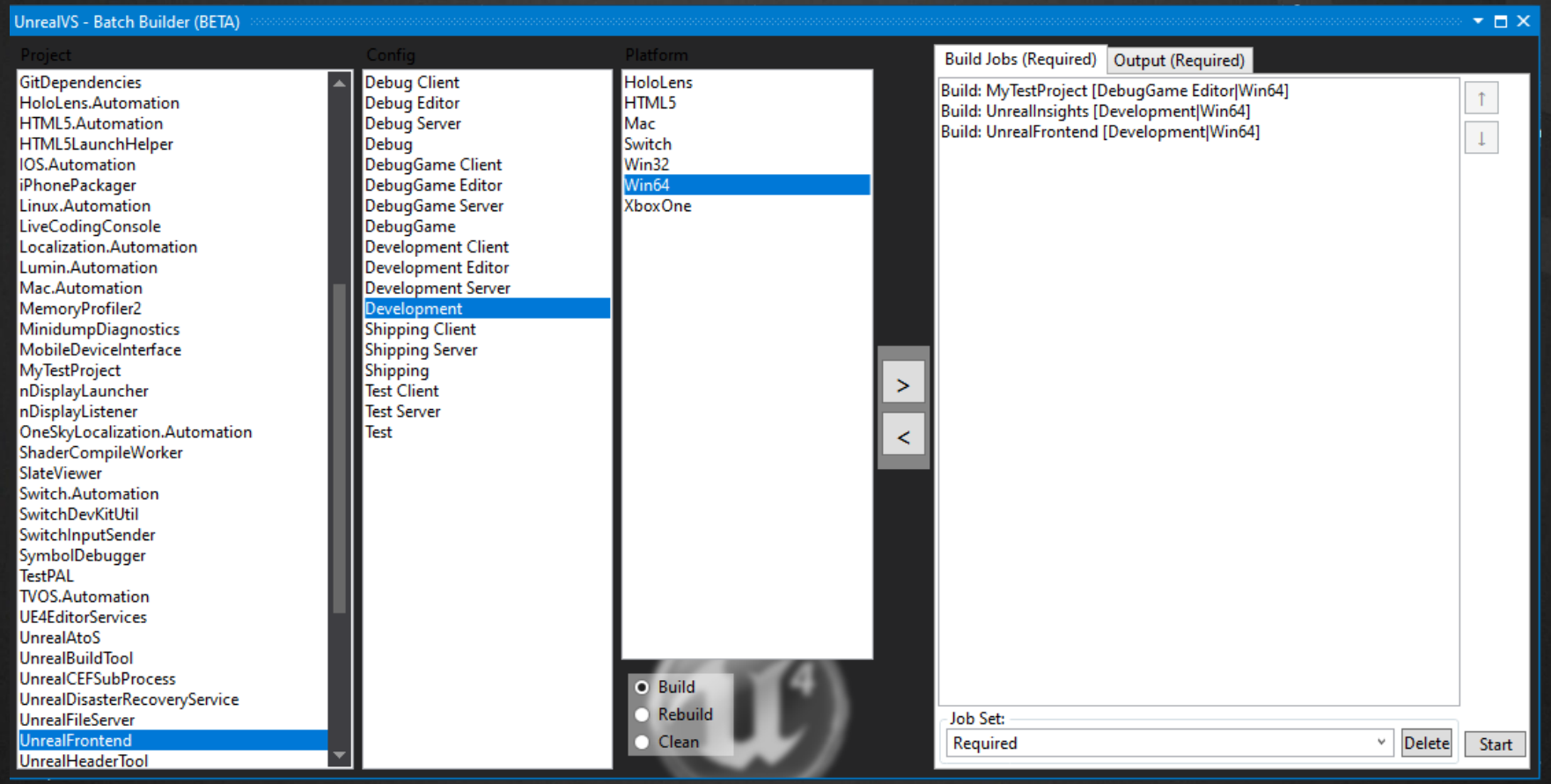
Unreal VS



- Visual Studio Extension made by Epic
- Found in *Engine/Extras/UnrealVS/*
- *Adds a nice batch of QOL improvements*

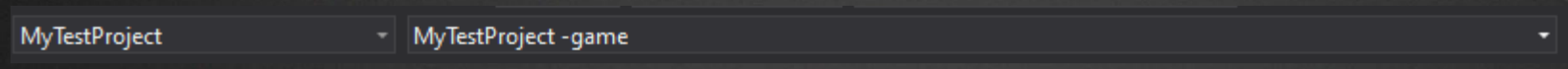
Unreal VS: Features

Batch Compiling

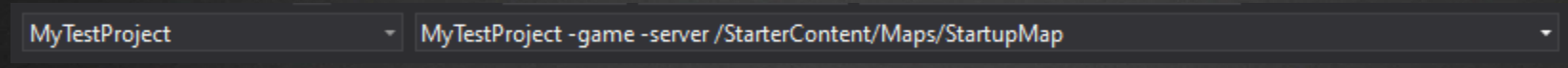


Unreal VS: Features
Start up Arguments

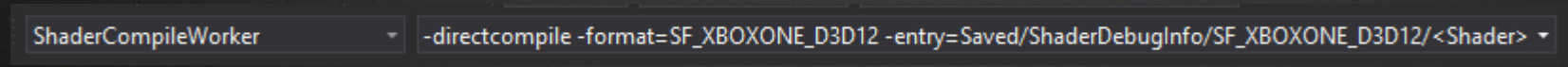
- Launching Standalone



- Passing general game arguments



- Shader Debugging



Unreal VS: Features
Smaller Improvements

- One click Regenerate Project Files

- Makes it easier to debug UE Tools
  - ShaderCompiler
  - UAT/UBT
  - etc

Launching from VS

- Pass -*game* as a start-up param to test Standalone

- Cook by the book, and run VS in **non editor** build config (Debug Game, Development etc)
  - Does not support Shipping
  - Deploys using content in Saved/Cooked
  - Should work on pretty much any platform (in my experience anyway)*

Cook by the… huh?

♫ COOKING BY THE BOOK ♫

Cook on The Fly

- Pros:
  - Useful to quickly jump into the game
  - Don't have to wait on the full game to be packaged
  - Great early on!
- Cons:
  - Doesn't scale well
  - Profiling isn't reliable
  - Expect many a hitch

Cook by the Book

- Pros:
  - All game contents packaged
    - More accurate reflection of the final product
  - Performance is not impacted
  - Easier/faster to debug (imo)
- Cons:
  - Can be slower to build (Definitely will be on your first run)

Analysing Game Performance

Before you Profile

HOT TIP

Profile in a packaged build or *at the very least* in Standalone.
Don't profile in editor!

Turn off frame rate smoothing
Project Settings -> General Settings -> Frame Rate -> Smooth Frame Rate

Tools

- Platform Specific
  - Xbox One – PIX
  - Switch - Dedicated CPU and GPU Profiler
  - Apple - Xcode
- Unreal Tools
  - Unreal Frontend
  - Unreal Insights (new in 4.23)
    - It's really nice

Identifying your Bottlenecks

- ***stat fps***

- **stat unit**

Did you know you can change the colour thresholds of stat unit/game?

HOT TIP

t.TargetFrameTimeThreshold = <ValueInMS>
Stat FPS' values below this will be drawn in green

t.UnacceptableFrameTimeThreshold = <ValueInMS>
Stat FPS' values above this will be drawn as red

Profiling: Unreal Frontend

Profiling: Unreal Frontend
Getting started

- *stat startfile* - To start profiling
- *stat stopfile* - To stop profiling
- Pass *-LoadTimeFile* as a start up param to begin profiling at game start

Profiling: Unreal Frontend

SEEING IT IN ACTION

Profiling: Unreal Insights

NEW YEAR, NEW ~~YOU~~ PROFILER

Profiling: Unreal Insights
Getting started

- CPU Profiling: *-cpuprofilertrace*
- Load Time Profiling: *-loadtimetrace*
- Without these Insights will still be able to see your session but no trace data will exist!

Profiling: Unreal Insights
Additional Notes

- Bug in 4.23 means on some platforms the device won't auto connect to Insights, can be overridden with *-tracehost=[dev_pc_ip]*
  - Hoping to have it fixed in 4.23.1!

- We will be back to Insights shortly….

Render/GPU Thread

Profiling: Render/GPU Thread

- *stat GPU*
  - Displays the cost of each pass on the GPU at a high level
- *stat scenerendering*
- *stat renderthreadcommands*
  - Lists the commands being submitted on the render thread (usually to the RHI thread)
- *stat DumpFrame -Root=<ThreadName>*
  - *-ms=0.1* to filter results

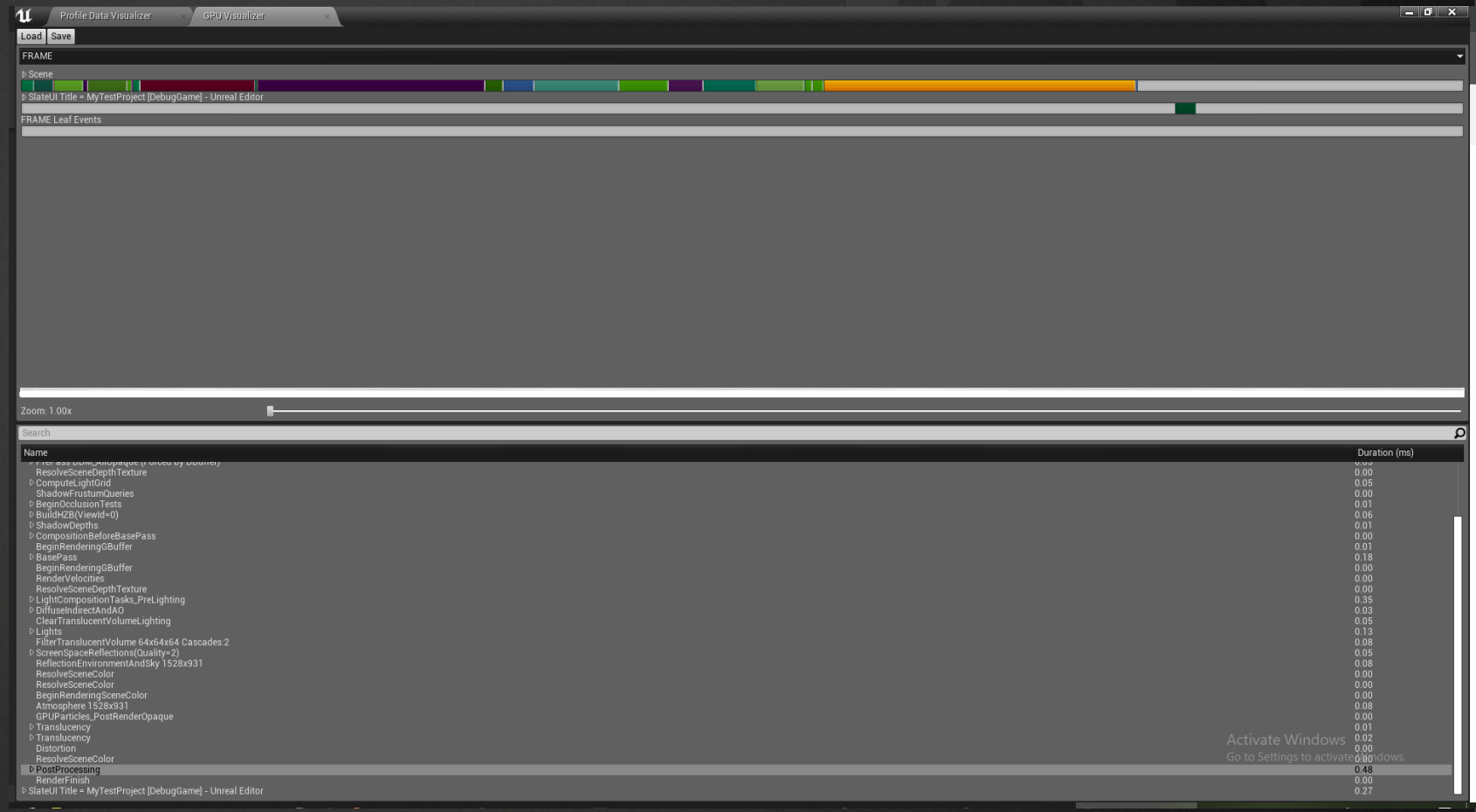
Profiling: Render/GPU Thread

GPU Visualiser

- Shortcut: *cntrl* + *shift* + ,

or

- Commandline: *ProfileGPU*



Profiling: Render/GPU Thread
Insights

- Render Thread Profiling
  - Available with *-cpuprofilertrace*
- GPU Profiling
  - Doesn't exist yet sorry!
  - But development is underway

Game Thread

Profiling: Game Thread

- stat game

Game [STATGROUP_Game]
Cycle counters (flat)

	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
World Tick Time	2	0.48 ms	0.79 ms	0.04 ms	0.10 ms
Tick Time	6	0.38 ms	0.66 ms	0.00 ms	0.00 ms
Char Movement Total	1	0.05 ms	0.15 ms	0.00 ms	0.00 ms
PlayerController Tick	1	0.04 ms	0.25 ms	0.00 ms	0.00 ms
Post Tick Component Update	3	0.03 ms	0.14 ms	0.00 ms	0.01 ms
Blueprint Time	7	0.03 ms	0.03 ms	0.00 ms	0.00 ms
Transform or RenderData	4	0.02 ms	0.12 ms	0.00 ms	0.01 ms
Update Camera Time	2	0.02 ms	0.03 ms	0.02 ms	0.03 ms
TickableGameObjects Time	3	0.02 ms	0.03 ms	0.01 ms	0.02 ms
Queue Ticks	2	0.02 ms	0.10 ms	0.02 ms	0.10 ms
GT Tickable Time	2	0.02 ms	0.02 ms	0.00 ms	0.00 ms
Nav Tick Time	2	0.00 ms	0.01 ms	0.00 ms	0.01 ms
Camera ProcessViewRotation	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Net Tick Time	2	0.00 ms	0.00 ms	0.00 ms	0.00 ms
GC Sweep Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Blueprint Latent Actions	3	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Reset Async Trace Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
TeleportTo Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Finish Async Trace Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Net Broadcast Tick Time	2	0.00 ms	0.00 ms	0.00 ms	0.00 ms
EndScopedMovementUpdate Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Cooldown Dequeuing	2	0.00 ms	0.00 ms	0.00 ms	0.00 ms

Counters	Average	Max	Min
Ticks Queued	24.00	24.00	24.00
TimerManager Heap Size	3.00	3.00	3.00

- *stat dumphitches*
  - Writes to console any hitches that occur with a full stack of what caused it
  - -ms=0.5 to only dump hitches > then the set time

Profiling: Unreal Insights

NEW YEAR, NEW ~~YOU~~ PROFILER

Memory

Profiling: Memory
General

- *stat memory* for general info
- *obj list* for a list of currently loaded objects
  - *-alphasort* orders them in an easier to read fashion
- *memreport –full* for a complete breakdown of memory for your running title

Profiling: Memory
LLM

- Pass *–LLM* as a start up argument
- *stat LLM*
- *stat LLMFull*
- Particularly useful for Consoles

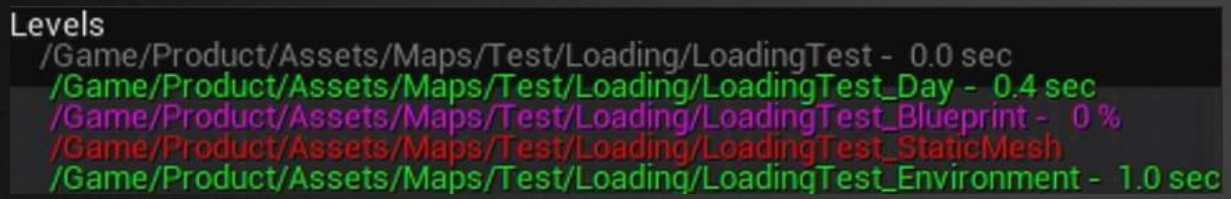
Load Time Performance

Profiling: Load Time Performance

- *stat levels*
- *Loadtimes.dumpreport*
- Via Unreal Insights with *-loadtimetrace*

Profiling: Load Time Performance

stat levels



- Green - Level is loaded and visible.
- Orange - Level is in the process of being made visible.
- Yellow - Level is loaded but not visible.
- Blue - Level is unloaded but still residing in memory, will be cleaned up when garbage collection occurs.
- Red - Level is unloaded.
- Purple - Level is preloading.

Profiling: Load Time Performance

Loadtimes.dumpreport

- Pass –file to dump loadtimes to a file
- Lowtime=0.05 to filter results
- Be sure to enter Loadtimes.reset to clear any previously loaded assets in the history

Profiling: Load Time Performance

Unreal Insights

General Performance Tweaks

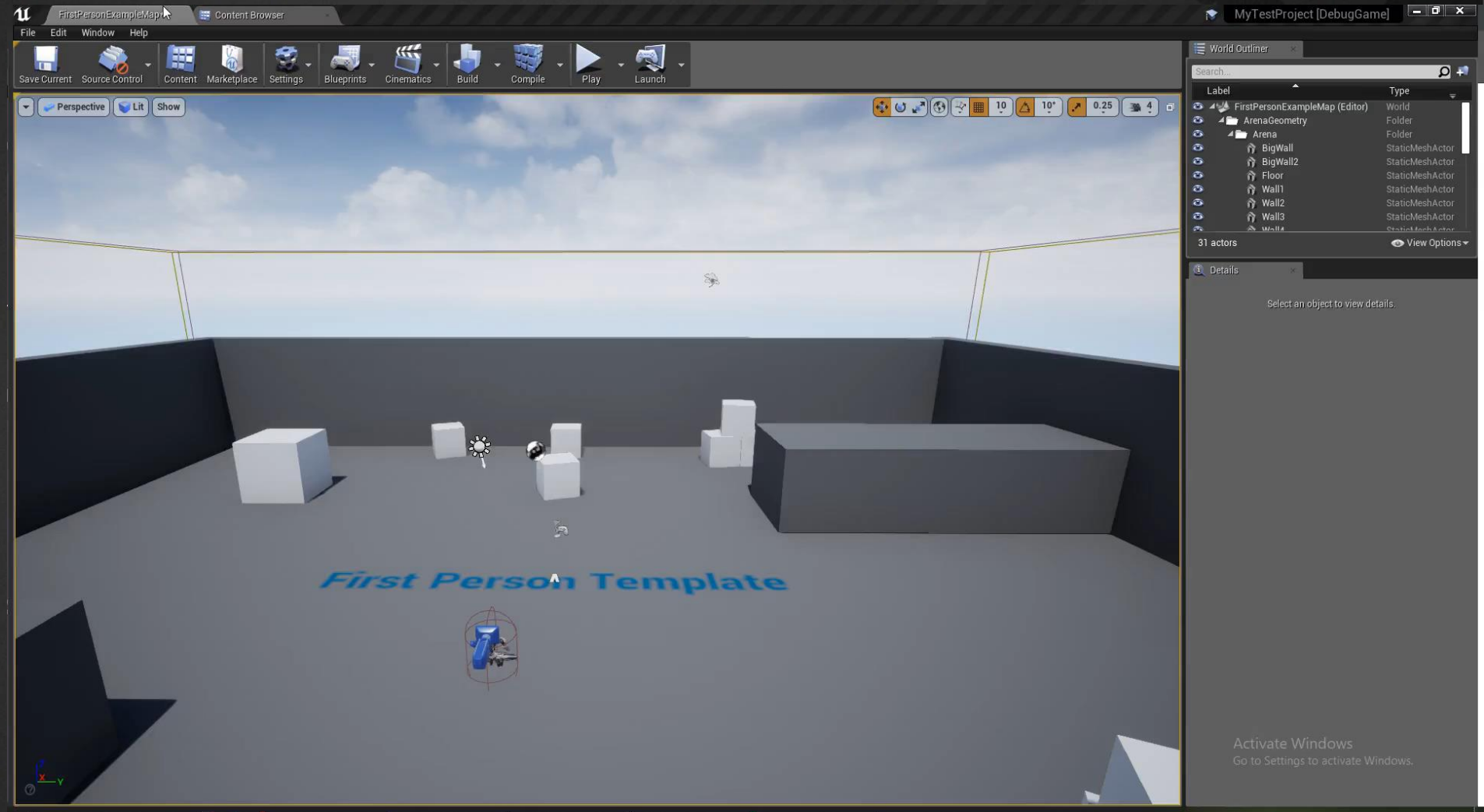
Render/GPU Thread

Device Profiles

- Allow you tweak bulk settings on a per device level
  - Set Texture Group Limits
  - Allows for heavy reduction in on disk size and setting of max streaming limits
  - Set any console variable
- Set Scalability Settings
- Supports parenting so you can chain device profiles
- Stored in *[ProjectRoot]/Config/DefaultDeviceProfiles.ini*
  - Additionally under each platform directory */Config/XboxOne/* etc

Device Profiles

Profile Manager



UNREAL ENGINE

Device Profiles
Adding your own

- What if I want to have a 4k@30 option and a 1080@60 on XboxOne X?
- Create a device profile section in the device profiles ini
- *UDeviceProfileManager* allows you to add your own Device Profiles
  - Set via *SetOverrideDeviceProfile*
  - Can restore the games' default profile via *RestoreDefaultDeviceProfile*

Dynamic Resolution

Dynamic Resolution

- Enabled State Set by r.DynamicRes.OperationMode
  - 0 is disabled
  - 1 enables Dynamic Resolution based on the Game User Settings state (set in C++ or in Blueprint).
  - 2 enables Dynamic Resolution regardless of the Game User Settings state
- Other Configurable values
  - r.DynamicRes.MinScreenPercentage
  - r.DynamicRes.MaxScreenPercentage
  - r.DynamicRes.FrameTimeBudget

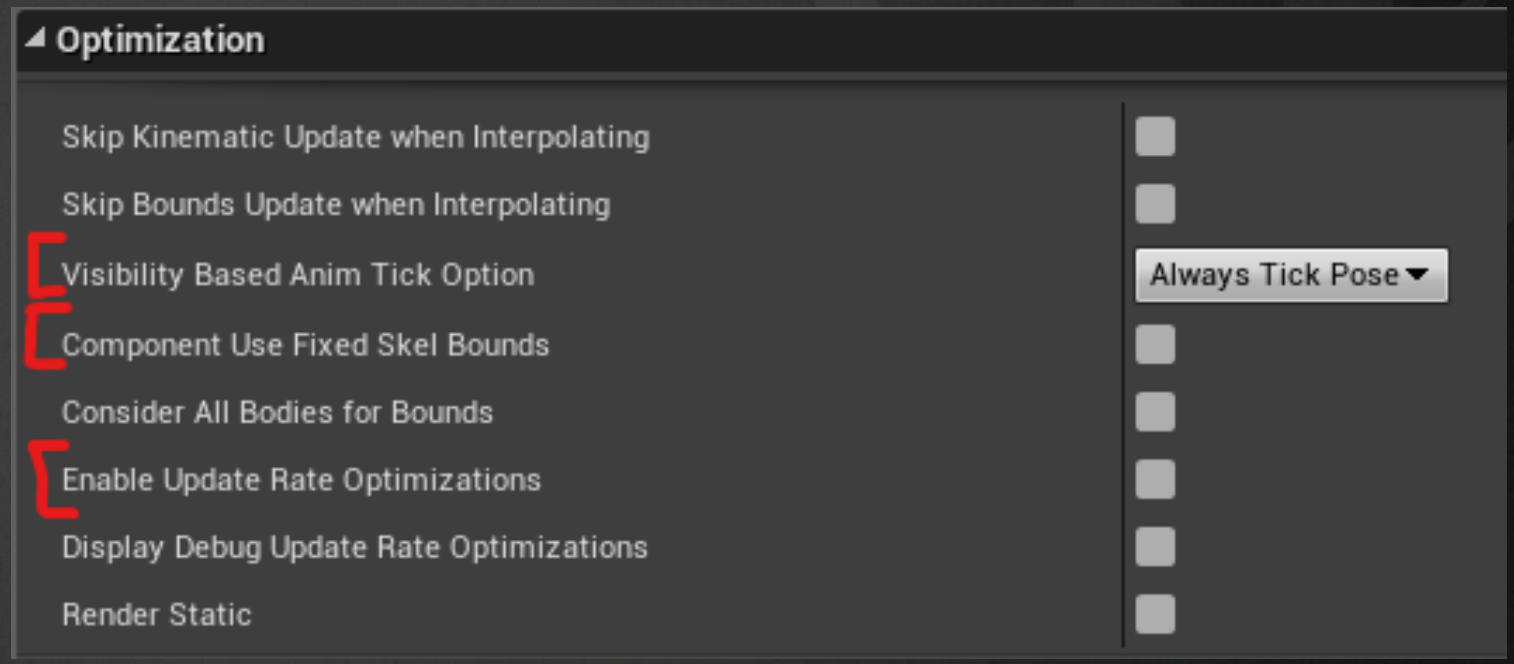
Dynamic Resolution
Monitoring

stat unit

stat unitgraph

Animation Updates

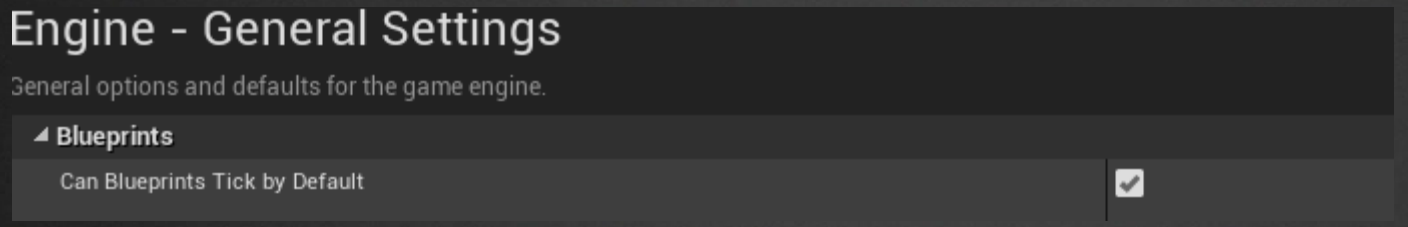
- Modify the update rate
- Consider using fixed skeletal bounds to reduce the computation required to calculate the bounding box.
- Enable Update Rate Optimizations



Game Thread

Performance Tweaks: Game Thread
Reduce Ticking

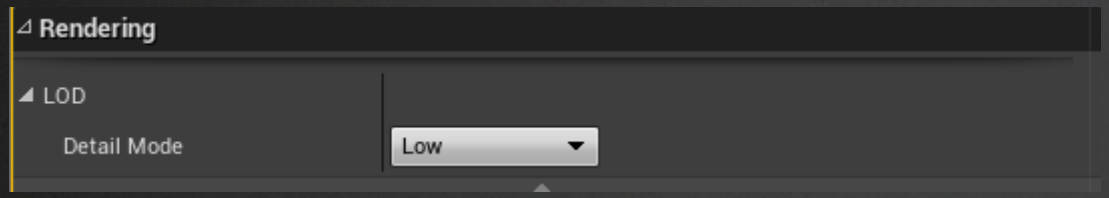
- Consider disabling this Tickbox in your Project Settings



- Avoid ticks as much as possible
  - Timers
  - Event Driven work instead of tick processing

Performance Tweaks: Game Thread
Detail Modes for Components

- All *Scene Components* have a property called DetailMode



- Enables the ability to pick and choose when an object is rendered and additionally even cooked!
  - *r.DetailMode* used in conjunction with Device profiles allows you to scale your world with systems
  - Enabling *r.CookOutUnusedDetailModeComponents* will result in components being skipped during the packaging process.

Performance Tweaks: Game Thread
Significance Manager

- Framework that allows for flexible assessment and prioritization of objects in your game world
- Particle System Components and Particle Emitters already support it!
- Plugin that needs to be enabled
- Docs available here: https://docs.unrealengine.com/en-US/Engine/Performance/SignificanceManager/index.html

Low Latency Frame Syncing
Overview

- In earlier versions of the engine the game thread synced with rendering thread.
- Introduction of the parallel rendering meant the Game Thread and Render thread can get much further ahead then the GPU.
- Also introduced high levels of input latency
- Low Latency Frame Syncing fixes this!

Low Latency Frame Syncing
Configuration

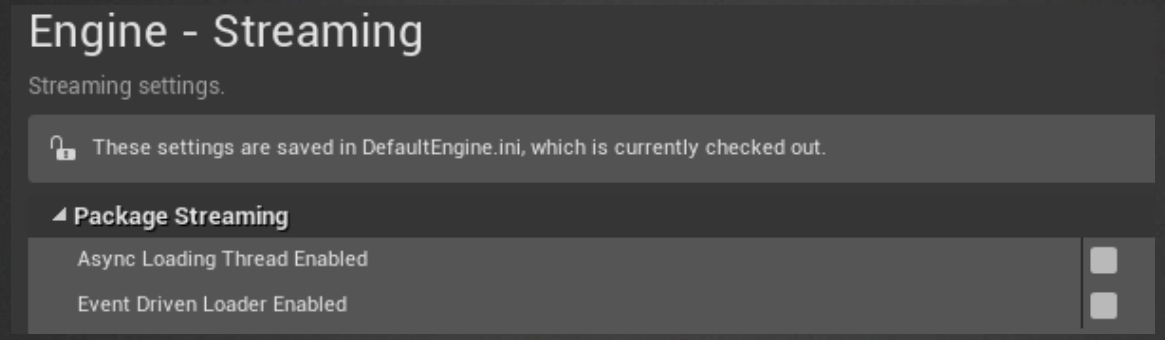
- **r.GTSyncType**
  - 0 = Game thread syncs with rendering thread (old behaviour, and default).
  - 1 = Game thread syncs to the RHI thread (equivalent to UE4 before parallel rendering)
  - 2 = Game thread syncs with the swap chain present +/- an offset in milliseconds.
- **rhi.SyncInterval**
- **rhi.SyncSlackMS**
- **r.OneFrameThreadLag**

Load Time Performance

Performance Tweaks: Load Time

- This is a whole talk on it's own!
  (In fact… I have done a talk on this and if you want the slides afterwards let me know!)

Performance Tweaks: Load Time
Tick Boxes



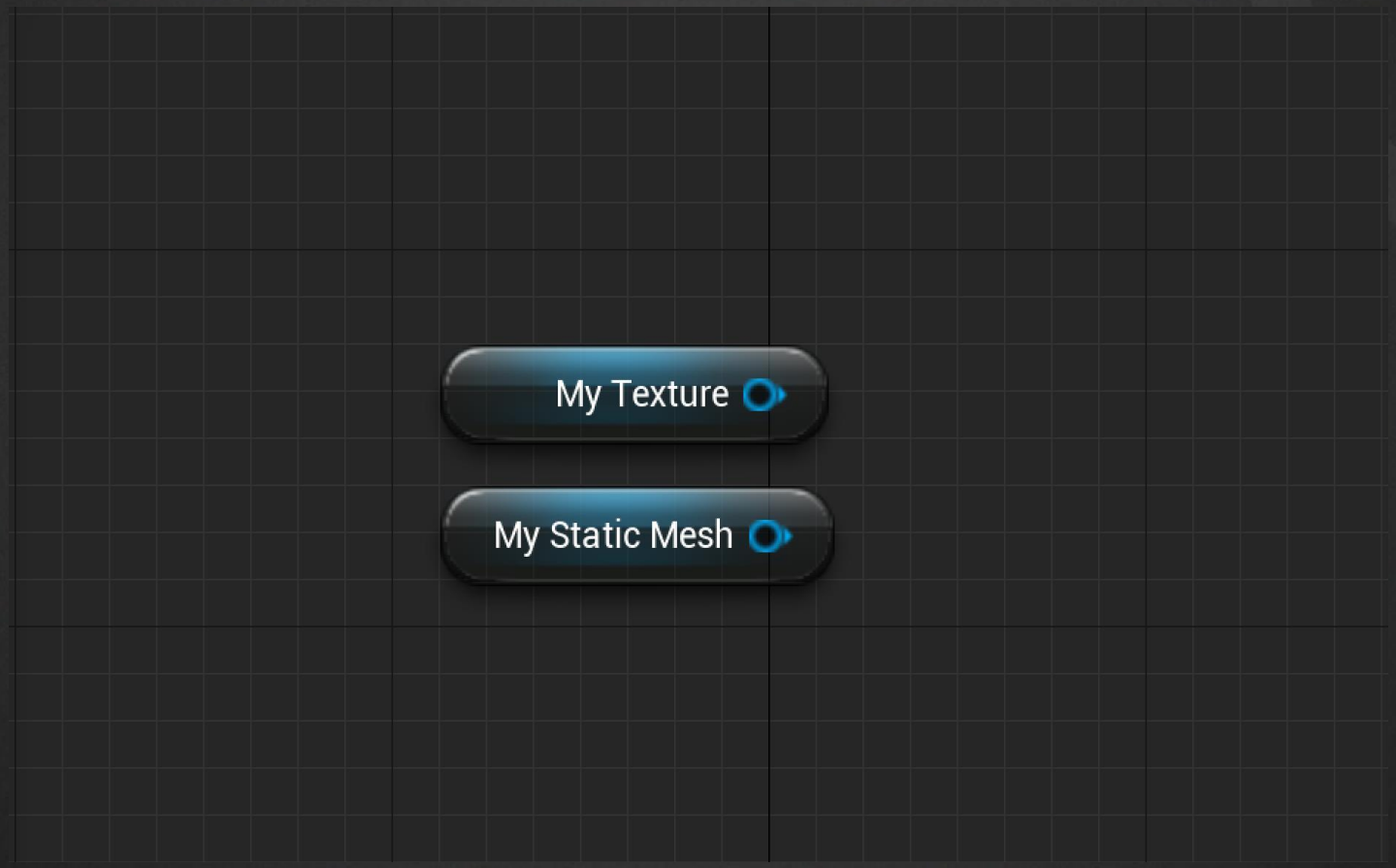
- If you have a new project these should be enabled by default
- Turning on both in my experience has resulted in a ~30-40% improvement in load times

Performance Tweaks: Load Time

References: Object and Class Properties

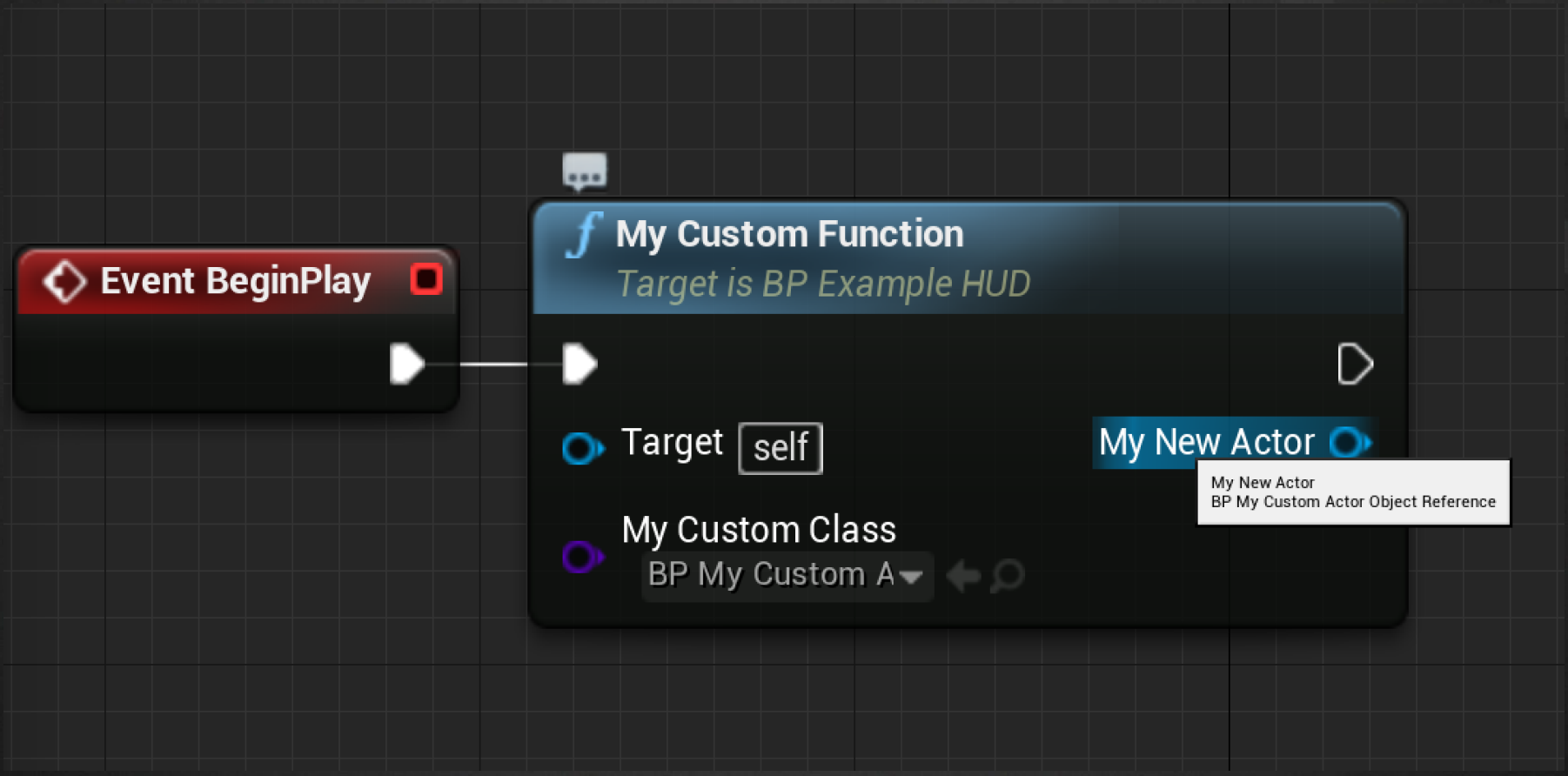
Performance Tweaks: Load Time

References: Asset Refs



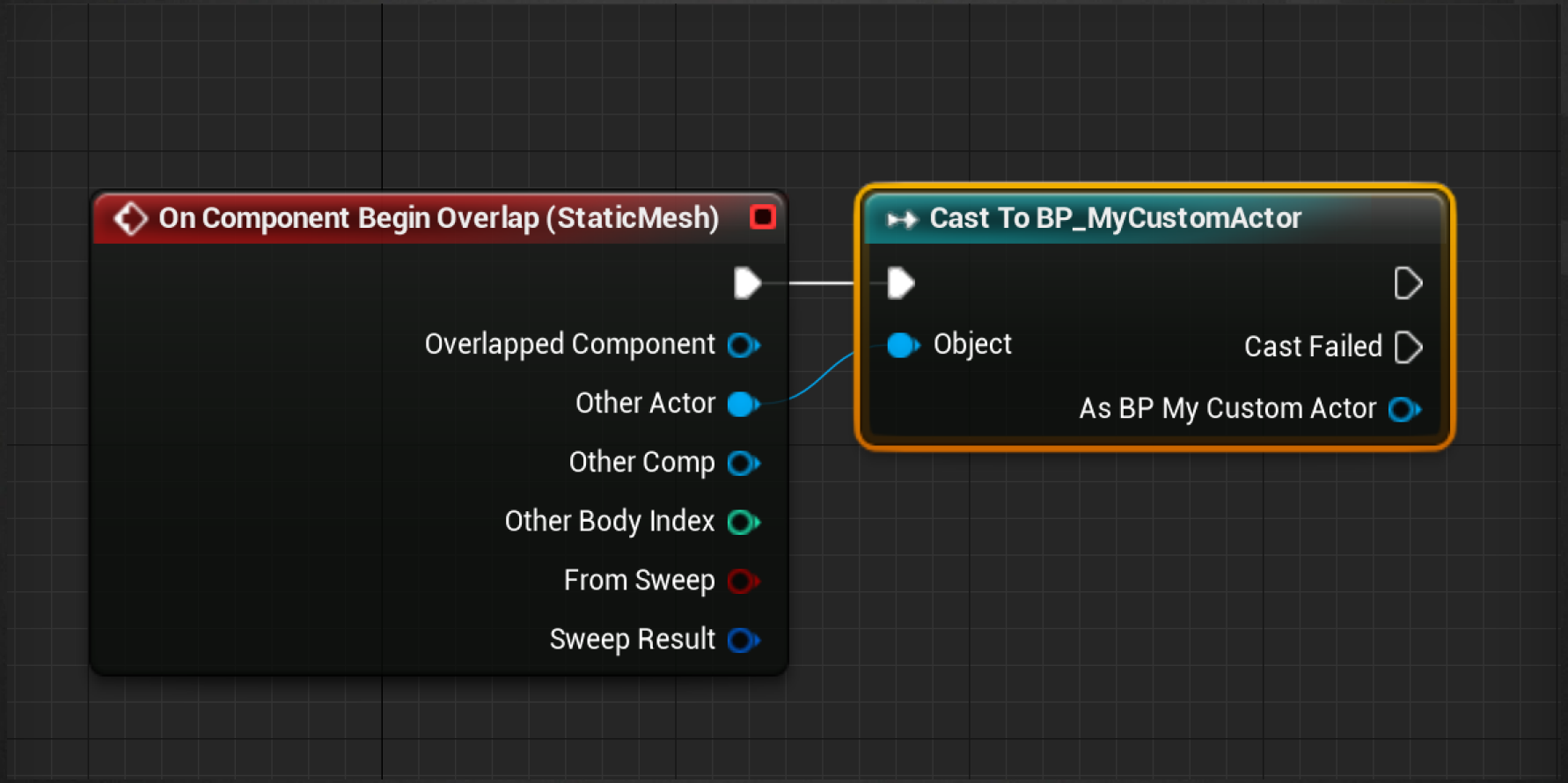
Performance Tweaks: Load Time

References: Function/Event Input and Output Pins



Performance Tweaks: Load Time

References: Cast



Performance Tweaks: Load Time
Ways around it

- Use Interfaces
  - Helps reduce the need for casts
- Decouple your Blueprint code
- Pak & Pak File Compression
  - Not desirable on all platforms (PS4 in particular)
  - Practically required on Xbox One

Performance Tweaks: Load Time
Ways around it

- Soft Object References
  - Soft Class Ptrs (TSoftClassPtr in C++)
  - Soft Object Ptrs (TSoftObjectPtr in C++)
  - Load with Asset Manager or Streamable Manager

- Design your **assets**, not just the game!

Performance Tweaks: Load Time

Aligning your pak file

- Performed by passing *–fileopenlog* as a start up parameter.
- Play through all the majors areas of your game; a full end to end play test.
- Copy the generated file from:
  <Platform>/<YourGame>/Build/ <Platform >/FileOpenOrder/
  to:
  <ProjectDir>/Build/<Platform>/FileOpenOrder/
- Re-package the game
- One of the biggest gains I've seen for HDD performance

Did you know that manually loading an asset synchronously will cause a flush of all currently async loading assets?

HOT TIP

Whelp!

UI Performance

Performance Tweaks: UI

- UI pre-passes run on Game Thread
- I always recommend a UI "Root" widget that is the one central container for 2D widgets
  - Easier to set focus and Navigation
  - Makes it easy to see screen layout in relation to each UI
  - HUD Class makes a great owner for this!
- **Don't** have hard references all the way down the widget hierarchy.
  - Load/Add widgets when required.

Performance Tweaks: UI
General Tips

- Hidden Widgets still have their layout considered
  - Consider using Collapsed where possible

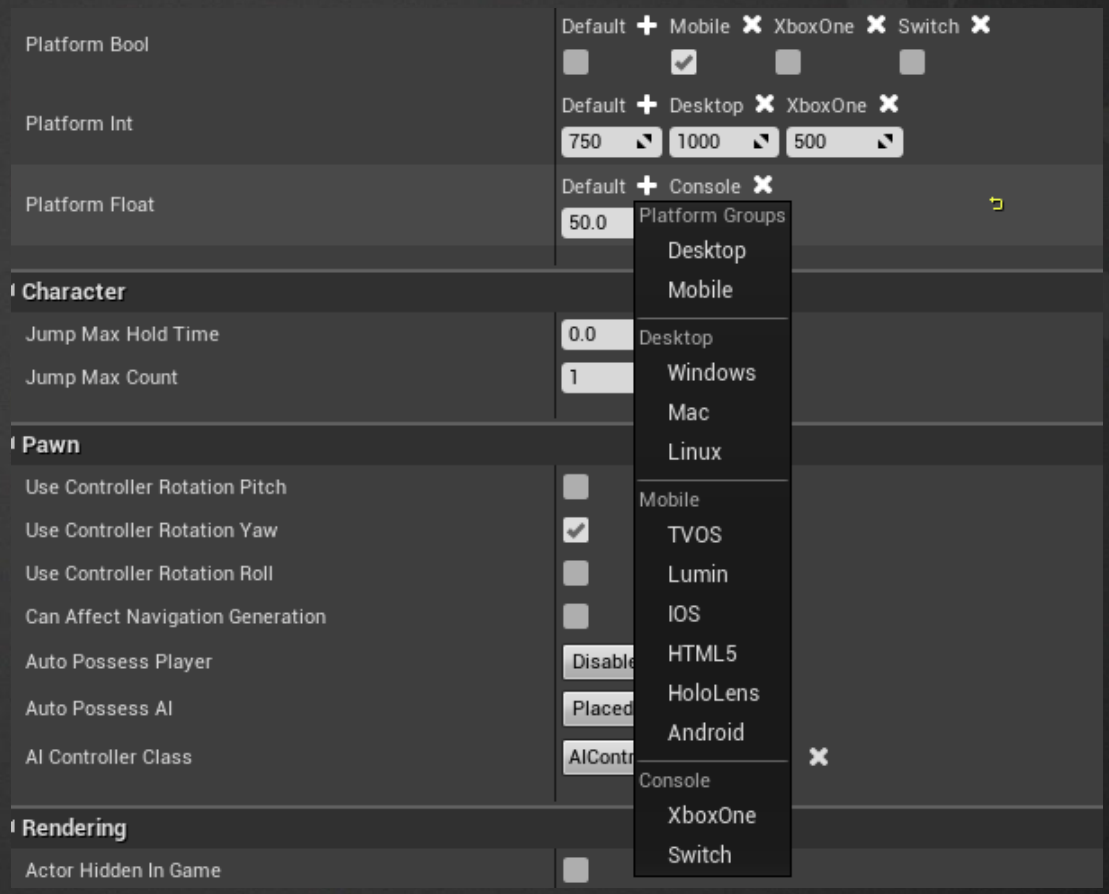
- Font loading can cause hitches
  - Font's by default are set to Lazy Load (Loaded on Demand)
  - Consider using Inline for fonts that are used regularly. Increase in memory but hitch free.

General Platform Development Tips

Per Platform Variables (4.20+)

Variables that allow for per platform and per *group* values

Only available to be used in C++ but can be exposed to Blueprints



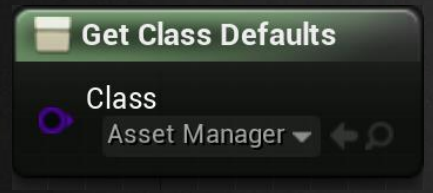
UDeveloperSettings Class

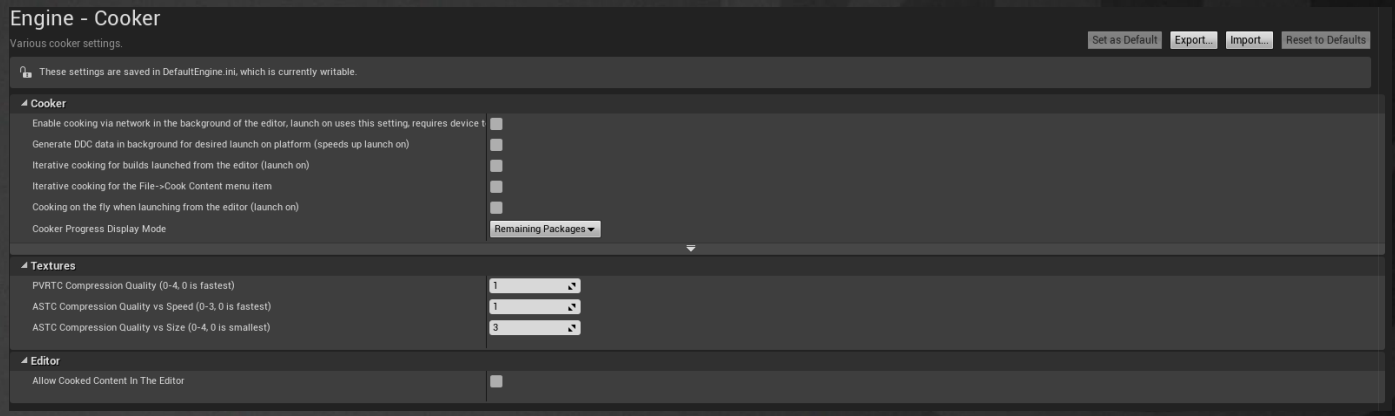
- Global Config Settings

- C++ via:
  GetDefault<MyDeveloperClassName>()

Or

- In blueprint via:






```
UCLASS(config=Engine, defaultconfig, meta=(DisplayName="Cooker"))
class UNREALED_API UCookerSettings : public UDeveloperSettings
{
    GENERATED_UCLASS_BODY()

    virtual void PostInitProperties() override;
    virtual void PostEditChangeProperty(FPropertyChangedEvent& PropertyChangedEvent) override;

public:

    UPROPERTY(GlobalConfig, EditAnywhere, Category = Cooker, meta = (DisplayName = "Enable cooking via network in the background of the editor, launch on uses
    bool bEnableCookOnTheSide;

    UPROPERTY(GlobalConfig, EditAnywhere, Category = Cooker, meta = (DisplayName = "Generate DDC data in background for desired launch on platform (speeds up l
    bool bEnableBuildDDCInBackground;

    /** Enable -iterate for launch on */
    UPROPERTY(GlobalConfig, EditAnywhere, Category = Cooker, meta = (DisplayName = "Iterative cooking for builds launched from the editor (launch on)"))
    bool bIterativeCookingForLaunchOn;

    /** Enable -iterate for launch on */
    UPROPERTY(GlobalConfig, EditAnywhere, Category = Cooker, meta = (DisplayName = "Iterative cooking for the File->Cook Content menu item"))
    bool bIterativeCookingForFileCookContent;

    /** Enable -cookonthefly for launch on */
    UPROPERTY(GlobalConfig, EditAnywhere, Category = Cooker, meta = (DisplayName = "Cooking on the fly when launching from the editor (launch on)"))
    bool bCookOnTheFlyForLaunchOn;
```

Platform Extensions (4.23+)

BRINGING IN A BRIGHTER FUTURE FOR CONSOLE DEVELOPERS ™

Platform Extensions
Why?

- Version Control permissions were complex
  (don't necessarily want all developers accessing console code)
- Every platform was hard coded
- Adding additional platforms usually meant modifying a lot of engine code
- Protecting an unreleased platform was high risk

Platform Extensions
Directory Structure

- Platform Extensions on an Engine Level will live in [UE4Root]/Platforms/[Platform]/…

- Projects can additionally add there own in the same vein via [ProjectRoot]/Platforms/[Platform]/…

Platform Extensions

Build.cs and Target.cs Changes

- UnrealTargetPlatform was previous an enum with all platforms
- Now a class with static members with the same name extendable via a Partial Class

Platform Extensions
HAL Standardization

- Hardware Abstraction Layer (HAL) has been updated to combine platform headers.

- Example:
  - TVOS overrides IOS platform.
  - Rather then having a TVOSPlatformTime.h, a macro is used to include the correct Platform Header e.g.
    #include COMPILED_PLATFORM_HEADER(PlatformTime.h)

Platform Extensions
Ini File Specification

- Has been extended to support "layering"
- Allowing for parenting of config files for further granularity in your config files
- Affects Config ini files (DefaultGame.ini, DefaultEngine.ini) and DeviceProfiles.ini

Platform Extensions

There are more changes so do check the release notes and the information forum post here: https://forums.unrealengine.com/unreal-engine/announcements-and-releases/1617783-attention-platform-changes-ahead

Debugging UFunctions
(incl Blueprint Functions)

Tired of seeing this in your callstack?

[Your C++ Code]
UFunction::Invoke(...)
UObject::ProcessEvent(...)
AActor::ProcessEvent(...)
[Rest of the stack]

HOT TIP

Debugging UFunctions
(incl Blueprint Functions)

In your immediate window in *Visual Studio* try:

Running in Editor:

{,,UE4Editor-Core}::PrintScriptCallstack()

HOT TIP

A Packaged build:

::PrintScriptCallstack()

Now you'll be presented with a nicely formatted output of the callstack include blueprint function names!

RTFM and Release Notes

- Please read the console API manuals before beginning!
- Check the release notes for the engine AS WELL as the updates to each platform API (Available on UDN and Forums)

Check Platform SDK expiration date

- Console SDK's do expire, take note of when they do! (It'll **always** happen close to ship)

- You don't **have** to upgrade engines to get the latest SDK, updating platform SDK's is usually pretty straight forward.

Additional Information

- Joe Conley's excellent talk:
  Adjusting Your Content to Perform on Target Hardware
  https://www.youtube.com/watch?v=Ln8PCZfO18Y

- Unreal Insights | Inside Unreal
  https://www.youtube.com/watch?v=TygjPe9XHTw

Questions?

Getting in Contact

- Discord
  - Unreal Engine Oceania - https://discord.gg/nTcTpf9
  - Unreal Slackers - https://discord.gg/unreal-slackers
  - GDL (Game Dev League) - https://discord.gg/gamedev
- Twitter - @jack_knobel