UE4 C++ Utilities

Niklas Smedberg
Senior Engine Programmer, Epic Games

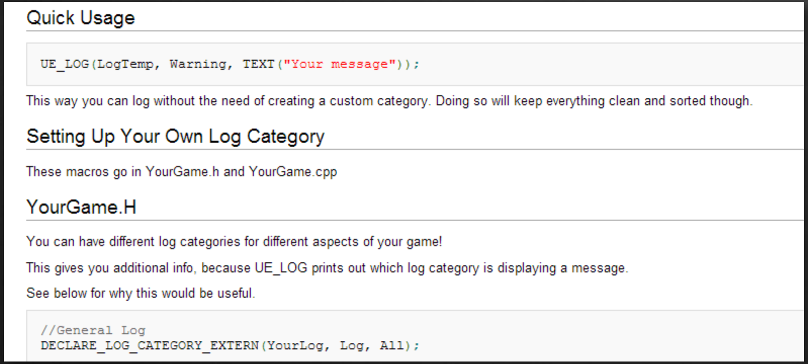
Content

- Logging
- File Management
- Containers
- Strings
- Localization
- Math

```
// Copyright 1998-2014 Epic Games, Inc. All Rights Reserved.

#include "StrategyProjectile.generated.h"

UCLASS(Blueprintable)
class AStrategyProjectile : public AActor, public IStrategyTeamInterface
{
	GENERATED_UCLASS_BODY()

	/** movement component */
	UPROPERTY(VisibleDefaultsOnly, Category=Projectile)
	TSubobjectPtr<UProjectileMovementComponent> MovementComp

	/** collisions */
	UPROPERTY(VisibleDefaultsOnly, Category=Projectile)
	TSubobjectPtr<USphereComponent> CollisionComp
```

Logging Overview

- Saved to $(ProjectFolder)/Saved/Logs/$(ProjectName).log
  - E.g. D:\Dev\UE4\Soul\Saved\Logs\Soul.log
  - Previous log files are backed up and renamed
- Wiki URL:
  - https://wiki.unrealengine.com/Logs,_Printing_Messages_To_Yourself_During_Runtime



Quick Usage

```
UE_LOG(LogTemp, Warning, TEXT("Your message"));
```

This way you can log without the need of creating a custom category. Doing so will keep everything clean and sorted though.

Setting Up Your Own Log Category

These macros go in YourGame.h and YourGame.cpp

YourGame.H

You can have different log categories for different aspects of your game!

This gives you additional info, because UE_LOG prints out which log category is displaying a message.

See below for why this would be useful.

```
//General Log
DECLARE_LOG_CATEGORY_EXTERN(YourLog, Log, All);
```

Logging

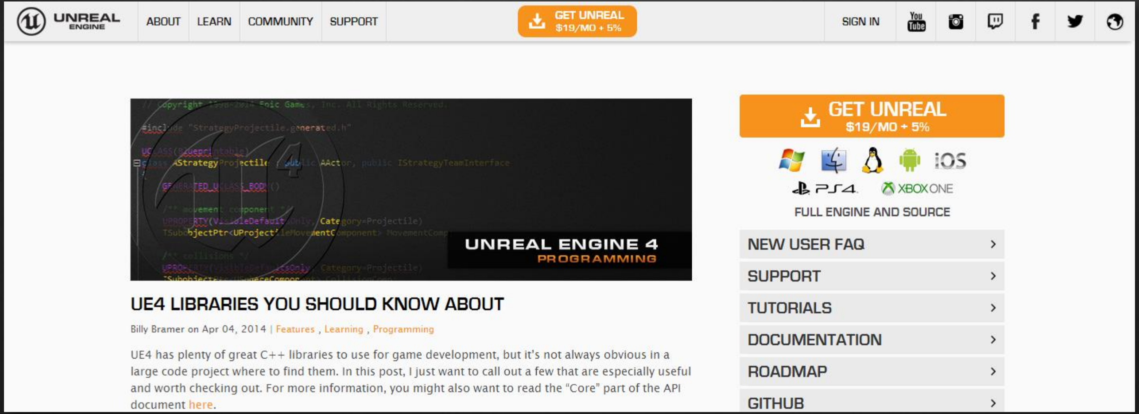
- Buffered logging
  - Logs from other threads are buffered and gathered up on the Main thread
  - Fully featured: Categorized, Warning/Error levels, Filtered, Printf-style
  - UE_LOG(Category, Level, FormatString, ...);
- Immediate logging
  - For platform-specific debugging purposes
  - Immediately outputs to attached debugger log window
  - FPlatformMisc::LowLevelOutputDebugString(Text);
  - FPlatformMisc::LowLevelOutputDebugStringf(FormatString, ...);

File Management

- Accessed via FPlatformFileManager singleton
  - FPlatformFileManager::Get()
- File system interface is IPlatformFile
  - FPlatformFileManager::Get().GetPlatformFile()
  - Returns a platform-specific singleton
    - E.g. FAndroidPlatformFile
  - Can also be used for virtual file systems
    - E.g. FPakPlatformFile
  - Example functions: OpenRead(), FileExists(), CopyFile(), etc
- File interface is iFileHandle
  - Example functions: Seek(), Read(), Write()

UE4 Containers

- Source folder:
  - Engine\Source\Runtime\Core\Public\Containers
- Blog URL:
  - https://www.unrealengine.com/blog/ue4-libraries-you-should-know-about



UE4 LIBRARIES YOU SHOULD KNOW ABOUT

Billy Bramer on Apr 04, 2014 | Features , Learning , Programming

UE4 has plenty of great C++ libraries to use for game development, but it's not always obvious in a large code project where to find them. In this post, I just want to call out a few that are especially useful and worth checking out. For more information, you might also want to read the "Core" part of the API document here.

UE4 Containers: TArray

- Dynamic array
- One of the most popular classes in the UE4 code base
  - Find-in-files found 25,000+ references
- Can be declared UPROPERTY and displayed in editor property windows
- Can be replicated over network
- Example: TArray::RemoveAtSwap()
  - Removes N elements by overwriting from the end
  - Does not preserve order, but is fast
- There is also a TStaticArray

UE4 Containers: TSet

- Similar to the C++ STL *set* class
- Common set operations: Intersect(), Union(), Difference()
- Other examples: Contains(), Add(), Remove(), iteration
- Implemented by hash table
- For new types, overload this function for your new type:

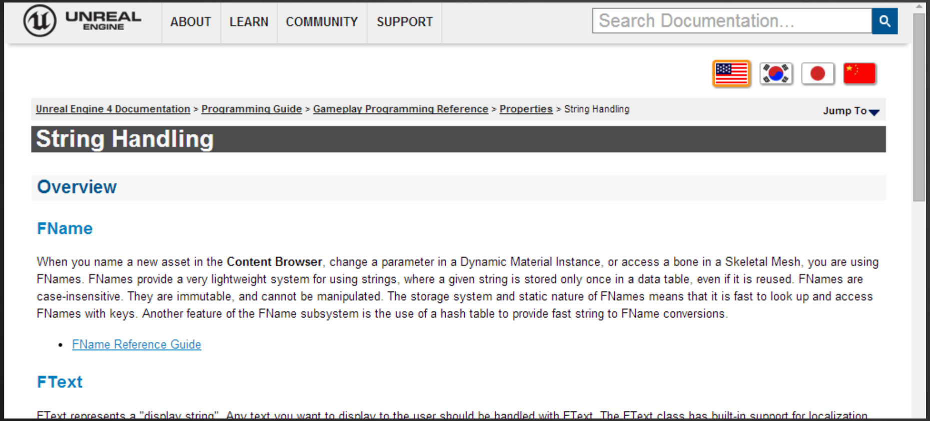
```
uint32 GetTypeHash( const MyType& MyObject );
```

UE4 Containers: TMap

- Similar to the C++ STL *map* class, or a *dictionary* class
- Very popular in the UE4 code base
  - Find-in-files found 3,000+ references
- Stores key-value pairs
- Implemented by using TArray and TSet
- Fast add, remove and look-up
- But beware of cache misses in inner loops

UE4 Strings

- Three main types: FString, FText and FName
- Documentation:
  - https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/StringHandling/index.html



Unreal Engine 4 Documentation > Programming Guide > Gameplay Programming Reference > Properties > String Handling

String Handling

Overview

FName

When you name a new asset in the **Content Browser**, change a parameter in a Dynamic Material Instance, or access a bone in a Skeletal Mesh, you are using FNames. FNames provide a very lightweight system for using strings, where a given string is stored only once in a data table, even if it is reused. FNames are case-insensitive. They are immutable, and cannot be manipulated. The storage system and static nature of FNames means that it is fast to look up and access FNames with keys. Another feature of the FName subsystem is the use of a hash table to provide fast string to FName conversions.

- FName Reference Guide

FText

FText represents a "display string". Any text you want to display to the user should be handled with FText. The FText class has built-in support for localization

Strings: FString

- Implemented using TArray<TCHAR>
- Used for dynamic string manipulation
- Examples of nice functions:
  - FString FString::Printf(FormatString, …);
  - int32 FString::Compare(OtherString, CaseSensitivityMode);
  - int32 FString::ParseIntoArray(StringArray, Delimiter, bIgnoreEmpty);
- String literals in UE4
  - Wrapped in TEXT("…") macro for cross-platform Unicode usage
  - Example: UE_LOG(LogEngine, Log, TEXT("Initializing Engine…"));

Strings: FText

- Builds on top of FString, but is immutable
- Intended for text displayed to user
  - E.g. text in Slate UI
- Takes localization into account
  - E.g. string comparison rules, numbers, dates, times, text formatting

Strings: FName

- Immutable, very lightweight storage
- Used for all object and asset names in UE4
- Basically just two integers
  - Index into a global name table
- Not case-sensitive

Localization

- FText TestHUDText = NSLOCTEXT("Your Namespace", "Your Key", "Your Text");
  - Namespace: When a word has different meanings, e.g. “chest”
  - Key: Unique identifier, e.g. “HUD_UserNameLabel”
  - Text: Default text if no localization is available
- UE4 parses all source code to find all uses of NSLOCTEXT
  - Done by running a commandlet called *“GatherText”*
  - E.g. UE4Editor-Cmd.exe -run=GatherText -config=...\Config\Localization\Engine.ini
  - Generates JSON files that can be translated directly or via third-party apps like OneSky
- Documentation:
  - https://www.unrealengine.com/blog/creating-a-localization-ready-game-in-ue4-part-1-text

Math

- FMath
  - Derives from FPlatformMath and adds cross-platform math functions
  - FPlatformMath is a typedef to a platform-specific class
    - typedef FWindowsPlatformMath FPlatformMath;
    - Shared base class is FGenericPlatformMath
  - Use it like a namespace
    - It just has public static member functions (mostly inlined)
- Examples:
  - if (FMath::IsNearlyZero(Value)) { … }
  - float GoodValue = FMath::Clamp(InputValue, 0.0f, 1.0f);
  - float WorldDistance = Fmath::PointDistToLine(Point, Line, Origin);
  - if (FMath::IsNaN(ResultFromComplexCalculations)) { … }

UE4 C++ Questions?

Documentation, Tutorials and Help at:

• AnswerHub:	http://answers.unrealengine.com
• Engine Documentation:	http://docs.unrealengine.com
• Official Forums:	http://forums.unrealengine.com
• Community Wiki:	http://wiki.unrealengine.com
• YouTube Videos:	http://www.youtube.com/user/UnrealDevelopmentKit
• Community IRC:	#unrealengine on FreeNode

Unreal Engine 4 Roadmap

- Imgtfy.com/?q=Unreal+engine+Trello+