Unreal Engine 4 C++ Cheat Sheet

(C) J. Böhmer, March 2018

Version 1.0

1 Reflection System

1.1 UPROPERTY()

BlueprintAssignable	Multicast Delegates only. Exposes property for assigning in Blueprints
BlueprintCallable	Multicast Delegates only. Property property for calling in Blueprints
BlueprintReadOnly	This property can be read by blueprints, but not modified
BlueprintReadWrite	This property can be read or written from a blueprint
Category	Specifies the category of the property within the Editor. Supports sub-categories separated by "|"
EditAnywhere	ndicates that this property can be edited via property windows, archetypes and instances within the Editor
EditDefaultsOnly	Indicates that this property can be edited by property windows, but only on archetypes. This operator is incompatible with the Visible* specifiers
EditFixedSize	Indicates that elements of an array can be modified in Editor, but its size cannot be changed
EditInstanceOnly	Indicates that this property can be edited by property windows, but only on instances, not on archetypes
Transient	Property is transient: shouldn't be saved, zero-filled at load time
VisibleAnywhere	Indicates that this property is visible in property windows, but cannot be edited at all
VisibleDefaultsOnly	Indicates that this property is only visible in property windows for archetypes, and cannot be edited
VisibleInstanceOnly	Indicates that this property is only visible in property windows for instances, not for archetypes, and cannot be edited
AdvancedDisplay	Moves the property into the Advanced dropdown in the Details panel within the Editor
EditCondition (Meta)	The property can only be edited in Editor if the specified bool Property is true. Use ! to invert logic (so you can only edit property if bool is false).

```
UPROPERTY(EditAnywhere, Category="Category|Sub")
bool BoolProperty;
UPROPERTY(BlueprintReadOnly, AdvancedDisplay)
TSubclassOf<UStaticMesh> AdvancedMeshClass;
UPROPERTY(meta=(EditCondition="BoolProperty"))
uint16 ConditionalInt;
```

1.2 UCLASS()

Abstract	An class that is marked as abstract can not be placed or instanced during runtime. This is especially useful for classes, that does not provide functionality on their own and must be inherited and modified for meaningful usage.
Blueprintable	Classes marked with this attribute can be used as a base class for creating Blueprints. On Default this is deactivated. The attribute is inherited by child classes, use NotBlueprintable on childs to disable this.
BlueprintType	Classes with this attribute can be used as variable type in Blueprints.
Placeable	Classes marked as Placable, can be created and placed in a level, UI Scene or Blueprint via the Editor. The flag is inherited by all child classes, use NotPlacable on child to disable this.

```
UCLASS(Blueprintable)
class MyClass : public UObject {
//Class code ...
}
```

1.3 UFUNCTION()

BlueprintAuthorityOnly	This function will not execute from Blueprint code if running on something without network authority
BlueprintCosmetic	This function is cosmetic-only and will not run on dedicated servers
Blueprint-ImplementableEvent	This function is designed to be overriden by a blueprint. Dont provide a body for this function in C++.
BlueprintNativeEvent	This function is designed to be overriden by a blueprint, but also has a native implementation. Provide a body named [Function-Name]_Implementation
BlueprintPure	This function has no side effects on the object. Useful for "Get" functions. Implies Blueprint-Callable
BlueprintCallable	This function can be called from Blueprints and/or C++.
Category	Specifies the category of the function within the Editor. Supports sub-categories separated by "|"
Exec	This function is callable from the Console CLI.

```
UFUNCTION(Exec)
void ConsoleCommand(float param);
UFUNCTION(BlueprintPure)
static FRotator MakeRotator(flat f);
UFUNCTION(BlueprintImplementableEvent)
void ImportantEvent(int param);
```

2 Classes and Functions

2.1 Base Gameplay Classes

- **UObject:** The base class, all classes, that should be used within C++ must extend. The name of child classes should start with U (e.g. UMyNewClass).
- **AActor:** Actor is the base class for all objects, that can be placed in a level. An Actor can has various Components. Child classes should start with A (e.g AMyNewActor).
- **APawn:** The base class, for all actors, that should be controled by players or AI.
- **ACharacter:** Characters are Pawn, which has a mesh collision and movement logic. They represent physical characters in the game world and can use CharacterMovementComponent for walking, flying, jumping and swiming logic.
- **UActorComponent:** The base class for all actor components. Components defines some reusable behavior, that can be added to different actors.
- **USceneComponent:** An Actor Component, which has a transform (position and rotation) and support for attachements.
- **UPrimitiveComponent:** A SceneComponent which can show some kind of geometry, usable for rendering and/or collision. Examples for this type are *StaticMeshComponent*, *SkeletalMeshComponent*, or the *ShapeComponent*s.

2.2 Datastructures and Helpers

- **TArray:** The mostly used container in UE4. The objects in it have a well-defined order, and functions are provided to create, get, modify or sort the elements. Similar to C++'s std::vector. You can iterate over the element like this:

```
for (AActor* Actor : ActorArray) {
Actor->SomeFunc(); }
```

- **TMap:** A container, where every element has a key (of any type), via which you identify every element. Similar to std::map
- **TSet:** A (fast) container to store unique elements without order. Similar to C++'s std::Set
- **TSubclassOf:** When you define a UProperty with the type TSubclassOf<UMyObject>, the editor allows you only to select classes, which are derived from UMyObject.
- **FName:** FNames provide a fast possibility to reference to things via a name. FNames are case-insensitive and can not be manipulated (they are immutable).
- **FText:** FText represents a string that can be displayed to user. It has a built in system for localization (so FTexts can be translated) and are immutable.
- **FString:** FString is the only class that allows manipulation. FStrings can be searched modified and compared, but this makes FStrings less performant than FText or FName.

2.3 Useful Functions

- **UE_LOG():** This functions allows to print message to the UE Log or the Output Log in the Editor. You can set a category (you can use LogTemp for temporal usage) and verbosity (like Error, Warning or Display). If you want to output a variable, you can use printf syntax. Usage Example:

```
//Print Test to console
UE_LOG(LogTemp, Warning, TEXT("Test"));
//Print the value of int n and a string
UE_LOG(LogTemp, Display, TEXT("n=%d"), n);
UE_LOG(LogTemp, Error, TEXT("%s"), MyString);
```

- **AddOnScreenDebugMessage():** If you want to print a debug message directly to the screen you can use AddOnScreenDebugMessage() from GEngine. You can specify a key, displaying time and display color. A message overrides an older message with the same key. Usage example:

```
GEngine->AddOnScreenDebugMessage(-1, 5.f,
     FColor::Red, TEXT("5 second Message"));
//Use FString, if you want to print vars
GEngine->AddOnScreenDebugMessage(-1, 5.f,
     FColor::Red,
     FString::Printf(TEXT("x: %f, y: %f"), x, y));
```

- **NewObject():** NewObject() creates a new UObject with the specific type. Objects created using NewObject() are not visible to the Editor, if you need that, use CreateDefaultSubObject() instead. Usage example:

```
auto RT = NewObject<UTextureRenderTarget2D>();
```

- **CreateDefaultSubobject():** This function creates a new named UObject with the specific type in the context of the current actor. Created objects are visible to the Editor, but this function can only be used in constructor. Usage example:

```
auto Mesh = CreateDefaultSubobject
    <UStaticMeshComponent>(FName("Mesh"));
```

- **LoadObject():** This function loads an objects from a specific asset. Usage example:

```
auto Mesh = LoadObject<UStaticMesh>(nullptr,
    TEXT("StaticMesh'/Asset/Path/Mesh.Mesh'");
```

- **Cast():** Casts an object to the given type. Returns nullptr if the object is not castable to this type. The object that should be casted, must be based on UObject, to work properly. Usage example:

```
AActor* Actor = Cast<AActor>(Other);
if(Actor != nullptr) {
/* do something */ }
```