

Blueprint Function Libraries

Information about Blueprint Function Libraries for C++ in Unreal Engine.



Often during the development of your project, you may discover the need for a set of functions that make development easier. These functions typically are stateless and reused across a variety of gameplay code. This need may present itself when building with Blueprints. In a previous section of this manual, you saw how to expose your game objects' functions and properties to Blueprints. However, if you want to expose shared utility functions, you don't necessarily want to tie them to a specific gameplay object type. For those cases, we have **Blueprint Function Libraries**.

Blueprint Function Libraries are a collection of static functions that provide utility functionality not tied to a particular gameplay object. These libraries can be grouped into logical function sets, e.g. AI Blueprint Library, or contain utility functions that provide access to many different functional areas, e.g. System Blueprint Library.

Creating a Blueprint Function Library is very similar to exposing functions to Blueprints using the `UFUNCTION()` macro. Instead of deriving from an Actor or directly from `UObject` all Blueprint Libraries inherit from `UBlueprintFunctionLibrary`. They should also contain only static methods. The code below is a snippet from the Analytics Blueprint Library, showing how to setup your library class.

```

1 UCLASS()
2 class UAnalyticsBlueprintLibrary :
3 public UBlueprintFunctionLibrary
4 {
5 GENERATED_UCLASS_BODY()
6 /** Starts an analytics session without any custom attributes specified */
7 UFUNCTION(BlueprintCallable, Category="Analytics")
8 static bool StartSession();
9

```

 Copy full snippet

As you can see in the example above, a Blueprint Function Library is indirectly a `UObject` derived and therefore requires the standard `UCLASS()` and `GENERATED_UCLASS_BODY()` macros. It decorates the functions that are to be callable from Blueprints with the `UFUNCTION()` macro. Functions in a Blueprint Function Library can be designated as `BlueprintCallable` or `BlueprintPure` depending on whether the calls have side effects or not.



See `Engine/Plugins/Runtime/Analytics/AnalyticsBlueprintLibrary` for the full source code.

Below is the implementation of the `StartSession()` function:

```

1 bool UAnalyticsBlueprintLibrary::StartSession()
2 {
3 TSharedPtr<IAalyticsProvider> Provider =
4   FAnalytics::Get().GetDefaultConfiguredProvider();
5 if (Provider.IsValid())
6 {
7   return Provider->StartSession();
8 }
9 else
10 {
11   UE_LOG(LogAnalyticsBPLib, Warning, TEXT("StartSession: Failed to get the
12     default analytics provider. Double check your [Analytics] configuration in
13     your INI"));
14 }
15 return false;


```

```
13 }  
14
```

 Copy full snippet

The implementation above interacts with a non-UObject derived singleton object. This is a good way to expose 3rd party library functions to Blueprints or to interact with C++ classes that don't have `UObject` support. The code below is an example of a Blueprint Function Library method that performs some common work to find the AIController for a controlled actor:

```
1  AAIController* UAIBlueprintHelperLibrary::GetAIController(AActor*  
   ControlledActor)  
2  {  
3  APawn* AsPawn = Cast<APawn>(ControlledActor);  
4  if (AsPawn != nullptr)  
5  {  
6  return Cast<AAIController>(AsPawn->GetController());  
7  }  
8  return Cast<AAIController>(ControlledActor);  
9  }  
10
```

 Copy full snippet

This function takes what would be multiple Blueprint nodes and rolls that into a single node. Of course, you could have made a function in Blueprints for this, but if it's called often the C++ version will yield better performance.