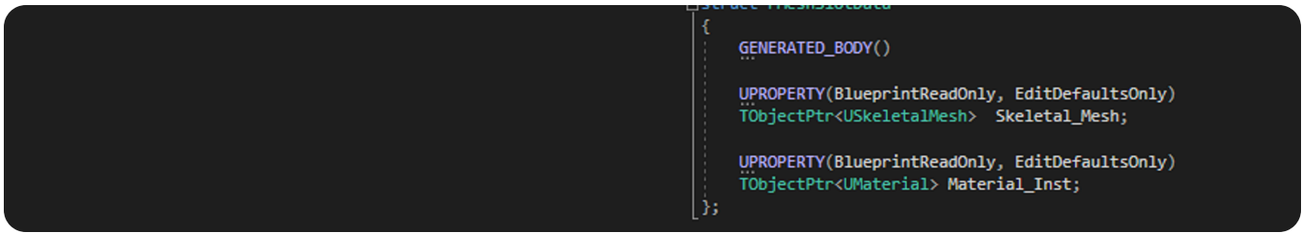
Structs

Reference to creating and implementing structs for gameplay classes.



A **struct** is a data structure that helps you organize and manipulate its member properties. Unreal Engine's reflection system recognizes structs as a `UStruct`, but they are not part of the UObject ecosystem, and cannot be used inside of UClasses.

- A `UStruct` is faster to create than a `UObject` with the same data layout.
- UStruct supports UProperty, but are not managed by the Garbage Collection system and cannot provide the functionality of a UFunction.

Implement a UStruct

To make a struct into a `UStruct`, follow the steps below:

1. Open the **header (.h)** file where you want to define your struct.
2. To define your C++ struct, put the `USTRUCT` macro above the struct's definition.
3. Include the `GENERATED_BODY()` macro as the first line of the definition.

The result should look like the following example:

```
USTRUCT([Specifier, Specifier, ...])
struct FStructName
{
GENERATED_BODY()
};
```

Copy full snippet

> You can tag the struct's member variables with `UPROPERTY` to make them visible to the Unreal Reflection System and Blueprint Scripting. See the list of UProperty Specifiers to learn how the property can behave in various Modules of the Engine and Editor.

Struct Specifiers

Struct Specifiers provide metadata that controls how your structs behave with various aspects of the Engine and Editor.

Struct Specifier	Effect
`Atomic`	Indicates that this struct should always be serialized as a single unit. No auto-generated code will be created for this class. The header is only provided to parse metadata from.
`BlueprintType`	Exposes this struct as a type that can be used for variables in Blueprints.
`NoExport`	No auto-generated code will be created for this class. The header is only provided for parsing metadata.

Best Practices & Tips

Below are some helpful tips to remember when you use `UStruct`:

1. `UStruct` can use Unreal Engine's smart pointer and garbage collection systems to prevent garbage collection from removing `UObjects`.
2. Structs are best used for simple data types. For more complicated interactions in your project, you might want to make a `UObject` or `AActor` subclass instead.
3. `UStructs` **ARE NOT** considered for replication. However, `UProperty` variables **ARE** considered for replication.
4. Unreal Engine can automatically create Make and Break functions for Structs.
    a. Make appears for any `UStruct` with the `BlueprintType` tag.
    b. Break appears if you have at least one `BlueprintReadOnly` or `BlueprintReadWrite` property in the UStruct.
    c. The pure node that Break creates provides one output pin for each property tagged as `BlueprintReadOnly` or `BlueprintReadWrite`.