

- Developer
- / Documentation
- / Unreal Engine ▾
- / Unreal Engine 5.4 Documentation
- / Making Interactive Experiences
- / Gameplay Framework
- / Actors
- / Spawning and Destroying an Actor
- / Spawning Actors

Spawning Actors

Methods of creating new instances of Actors in gameplay code.



SpawnActor Method

The process of creating a new instance of an **Actor** is known as **spawning**. Spawning of Actors is performed using the `UWorld::SpawnActor()` function. This function creates a new instance of a specified class and returns a pointer to the newly created Actor.

`UWorld::SpawnActor()` may only be used for creating instances of classes which inherit from the Actor class in their hierarchy.

```
1 AActor* UWorld::SpawnActor
2 (
3 UClass* Class,
4 FName InName,
5 FVector const* Location,
6 FRotator const* Rotation,
7 AActor* Template,
8 bool bNoCollisionFail,
9 bool bRemoteOwned,
10 AActor* Owner,
```

```
11 APawn* Instigator,  
12 bool bNoFail,  
13 ULevel* OverrideLevel,  
14 bool bDeferConstruction  
15 )  
16  
17
```

 Copy full snippet

Parameter

Description

Class	A UClass specifying the class of the Actor to be spawned.
InName	(Optional) An FName to assign as the Name of the Actor being spawned. If no value is specified, the name of the spawned Actor will be automatically generated using the form [Class]_[Number].
Location	(Optional) An FVector supplying the initial location to spawn the Actor at.
Rotation	(Optional) An FRotator supplying the initial rotation to spawn the Actor with.
Template	(Optional) An AActor to use as a template when spawning the new Actor. The spawned Actor will be initialized using the property values of the template Actor. If no template Actor is specified, the class default object (CDO) will be used to initialize the spawned Actor.
bNoCollisionFail	(Optional) A bool that determines whether a collision test will be performed when spawning the Actor. If true, no collision test will be performed when spawning the Actor regardless

Parameter

Description

	of the collision settings of the root component or template Actor.
<code>bRemoteOwned</code>	(Optional) <code>bool</code> .
<code>Owner</code>	(Optional) The <code>AActor</code> that owns the spawned Actor.
<code>Instigator</code>	(Optional) The <code>APawn</code> that is responsible for damage done by the spawned Actor.
<code>bNoFail</code>	(Optional) A <code>bool</code> that determines whether spawning will not fail if certain conditions are not met. If <code>true</code> , spawning will not fail because the class being spawned is <code>bStatic=true</code> or because the class of the template Actor is not the same as the class of the Actor being spawned.
<code>OverrideLevel</code>	(Optional) The <code>ULevel</code> to spawn the Actor in, i.e. the <code>Outer</code> of the Actor. If no level is specified, the <code>Outer</code> of the <code>Owner</code> is used. If no <code>Owner</code> is specified, the persistent level is used.
<code>bDeferConstruction</code>	(Optional) A <code>bool</code> that determines whether the construction script will be run. If <code>true</code> , the construction script will not be run on the spawned Actor. Only applicable if the Actor is being spawned from a Blueprint .
Return Value	


Parameter	Description
	<p>The spawned Actor in the form of an <code>AActor</code> pointer. The return value must be cast to convert to the derived type specified by <code>Class</code> parameter.</p>

Usage

```

1  AKAsset* SpawnedActor1 = (AKAsset*) GetWorld()-
    >SpawnActor(AKAsset::StaticClass(), NAME_None, &Location);
2

```

 Copy full snippet

Spawn Function Templates

In order to make spawning Actors more friendly, several function templates are provided for the most common usage patterns. These make creating Actors much simpler as they require a smaller subset of parameters and allow the type of the returned Actor to be specified.

Spawn T Instance, Return T Pointer

This function template spawns an instance of the template class `T` at the same location, and with the same rotation, as the root component of the Actor performing the spawn operation, and returns a pointer to that instance of the same type as the template class, i.e. `T*`. The owning Actor, instigating Pawn, and whether the spawn operation should fail if the spawned Actor encroaches, or would be colliding with another Actor already present in the world can be specified.

```

1  /** Spawns and returns class T, respects default rotation and translation of
    root component. */
2  template< class T >
3  T* SpawnActor

```

```

4 (
5 AActor* Owner=NULL,
6 APawn* Instigator=NULL,
7 bool bNoCollisionFail=false
8 )
9 {
10 return (T*)(GetWorld()->SpawnActor(T::StaticClass(), NAME_None, NULL, NULL,
    NULL, bNoCollisionFail, false, Owner, Instigator));
11 }
12

```

 Copy full snippet

Usage

```

1 MyHUD = SpawnActor<AHUD>(this, Instigator);
2

```

 Copy full snippet

Spawn T Instance with Transform, Return T Pointer

This function template spawns an instance of the template class `T` at the specified `Location` and with the specified `Rotation`, and returns a pointer to that instance of the same type as the template class, i.e. `T*`. In addition to the location and rotation, the owning Actor, instigating Pawn, and whether the spawn operation should fail if the spawned Actor encroaches, or would be colliding with, another Actor already present in the world can be specified.

```

1 /** Spawns and returns class T, forcibly sets world position. */
2 template< class T >
3 T* SpawnActor
4 (
5 FVector const& Location,
6 FRotator const& Rotation,
7 AActor* Owner=NULL,
8 APawn* Instigator=NULL,

```

```

9  bool bNoCollisionFail=false
10 )
11 {
12 return (T*)(GetWorld()->SpawnActor(T::StaticClass(), NAME_None, &Location,
    &Rotation, NULL, bNoCollisionFail, false, Owner, Instigator));
13 }
14

```

 Copy full snippet

Usage

```

1 Controller = SpawnActor<AController>(GetLocation(), GetRotation(), NULL,
    Instigator, true);
2

```

 Copy full snippet

Spawn Class Instance, Return T Pointer

This function template spawns an instance of the specified `Class` at the same location, and with the same rotation, as the root component of the Actor performing the spawn operation, and returns a pointer to that instance cast to the type of the template class, i.e. `T*`. This requires that the specified `Class` be a child of the template class `T`. In addition to the class, the owning Actor, instigating Pawn, and whether the spawn operation should fail if the spawned Actor encroaches, or would be colliding with, another Actor already present in the world can be specified.

```

1 /** Spawns given class and returns class T pointer, respects default
    rotation and translation of root component. */
2 template< class T >
3 T* SpawnActor
4 (
5 UClass* Class,
6 AActor* Owner=NULL,
7 APawn* Instigator=NULL,
8 bool bNoCollisionFail=false

```

```

9  )
10 {
11 return (Class != NULL) ? Cast<T>(GetWorld()->SpawnActor(Class, NAME_None,
    NULL, NULL, NULL, bNoCollisionFail, false, Owner, Instigator)) : NULL;
12 }

```


 Copy full snippet

Usage

```

1 MyHUD = SpawnActor<AHUD>(NewHUDClass, this, Instigator);
2

```

 Copy full snippet

Spawn Class Instance with Transform, Return T Pointer

This function template spawns an instance of the specified `Class` at the specified `Location` and with the specified `Rotation`, and returns a pointer to that instance of the same type as the template class, i.e. `T*`. This requires that the specified `Class` be a child of the template class `T`. In addition to the class, location, and rotation, the owning Actor, instigating Pawn, and whether the spawn operation should fail if the spawned Actor encroaches, or would be colliding with, another Actor already present in the world can be specified.

```

1  /** Spawns given class and returns class T pointer, forcibly sets world
2  position. */
3  template< class T >
4  T* SpawnActor
5  (
6  UClass* Class,
7  FVector const& Location,
8  FRotator const& Rotation,
9  AActor* Owner=NULL,
10 APawn* Instigator=NULL,
11 bool bNoCollisionFail=false

```

```
11 )  
12 {  
13 return (Class != NULL) ? Cast<T>(GetWorld()->SpawnActor(Class, NAME_None,  
    &Location, &Rotation, NULL, bNoCollisionFail, false, Owner, Instigator)) :  
    NULL;  
14 }  
15
```

 Copy full snippet

Usage

APawn* ResultPawn = SpawnActor(DefaultPawnClass, StartLocation, StartRotation, NULL,
Instigator);