

Developer

/ Documentation

/ Unreal Engine ▾

/ Unreal Engine 5.4 Documentation

/ Testing and Optimizing Your Content

/ Significance Manager

Significance Manager

Adjust performance in project-specific ways with the Significance Manager



Meeting performance targets for a shipping game generally includes reducing scene complexity in order to meet the target resolution or frame rate. Level-of-detail systems for geometry, animation, and even audio are commonly used, but there are some cases where these distance-based, per-Actor methods are not sufficient. This is especially true in the case of multiplayer games with high numbers of players or AI-controlled characters that can converge in a single area.

The **Significance Manager** provides a centralized framework that supports the ability to write flexible, project-specific code for evaluating and prioritizing Objects relative to one another. Using this evaluation, Objects can modify their

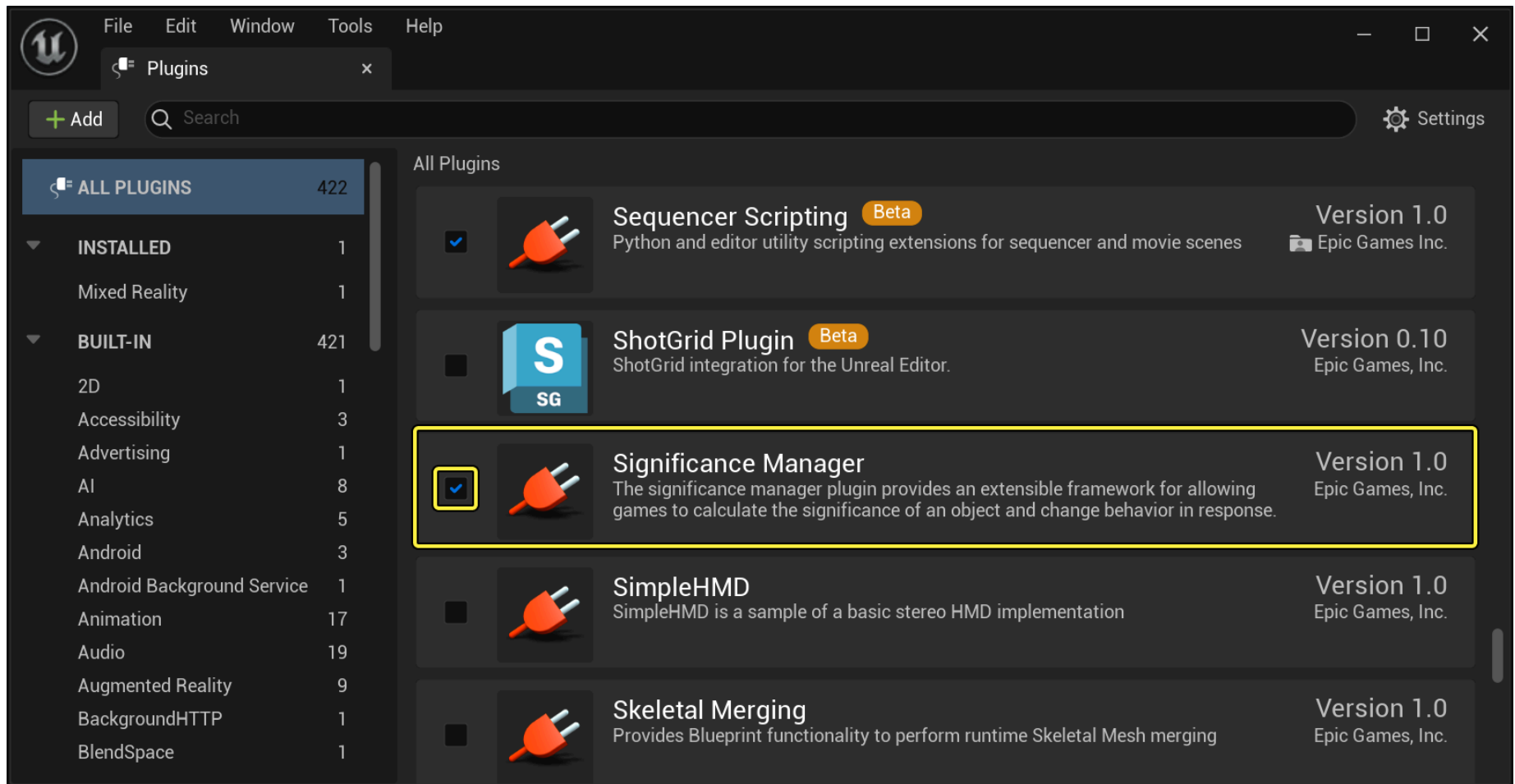
behavior by doing things like shutting off **Components** like **Particle Emitters**, or by running complex AI code less frequently.



The Significance Manager itself does not actually improve performance; rather, it provides a system that can be overridden and customized to suit your project's specific needs.

Setup

Because the Significance Manager exists within a Plugin, it must be enabled in the **Edit > Plugins** menu, and its Module must be added to your project's `*.Build.cs` file.



The Significance Manager is located in the Programming section of the Plugins menu.



After enabling the Significance Manager Plugin, you may need to restart the Engine.

With the Plugin enabled, add "SignificanceManager" to `PublicDependencyModuleNames` in your project's `*.Build.cs` file. The following example line comes from the "Basic C++" project template, modified to use the Significance Manager:

```
PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore", "Sign
```

 Copy full snippet

Significance Manager Base Functionality

The Significance Manager Plugin contains a single class, `USignificanceManager`, which acts as an extensible framework for evaluating the "significance" of managed Objects. These Objects can then adjust their behavior in custom ways to decrease their impact on performance based on their significance values. The specific behaviors that will result in improved performance are custom-defined by the Objects themselves in game code. For example, an Actor that plays a subtle audio cue or particle effect might opt not to do so in the event that it has a low significance value. A more advanced use case might involve grouping similar Actors together and enacting a per-Actor-type budget. One possible use for this would be to ensure that Player-controlled Pawns always run at high detail when they're close to the camera, and then compensating for cases where several players are clustered around the camera by limiting the number of AI-controlled Pawns running at high detail accordingly.

RegisterObject / UnregisterObject

Objects can be registered with the Significance Manager and will be grouped together with other registered Objects based on a user-designated name. The registration process includes the ability for users to specify the functions used to evaluate the significance of the Object, and the optional function that will be run after the evaluation has been made. During registration, the Object's initial significance will be calculated with the Transforms used in the most recent call to the Significance Manager's Update function, if possible. This also provides an opportunity to perform higher-level processing, such as building internal data structures based on the list of known, registered Objects (perhaps different

lists for different types), which can be helpful if your game implements category-based budgets for different types of Objects.

GetSignificance / QuerySignificance

These functions report the cached significance value of an Object. If the Object is not registered with the Significance Manager, that value will be zero. The `QuerySignificance` function, unlike `GetSignificance`, will also indicate that the Object is not registered by returning `false`.

Update

This function takes an array of Transforms and evaluates each managed Object for significance based on each Transform, using the significance function associated with the Object. The final result will be the highest value returned (lowest if `bSortSignificanceAscending` is set to `true`). This function can be overridden to suit the needs of the game, for example, by implementing new pre- or post-processing steps to the system. After evaluating an Object's significance, its Post Significance Function will be called, if one has been specified. This function will be called immediately if the Object's Post Significance Type is Concurrent. If its type is Sequential, it will be called in order, from most significant to least, with all other managed Objects using Sequential post-updates. If no Transforms are supplied, the significance value will be zero.

Significance-evaluation and post-significance-evaluation functions run in parallel, which adds the requirement that these functions be thread-safe. Post-significance-evaluation functions can avoid this requirement by running sequentially (see the `FPostSignificanceFunction` section below for details).

The `Update` function does not run automatically. In most cases, developers will want to call it every frame, and only once per frame. A good place to call it might be in an overridden version of `UGameViewportClient`, as demonstrated in the

following code:

```
1 #include "MyGameViewportClient.h"
2 #include "SignificanceManager.h"
3 #include "Kismet/GameplayStatics.h"
4 void UMyGameViewportClient::Tick(float DeltaTime)
5 {
6     // Call the superclass' Tick function.
7     Super::Tick(DeltaTime);
8     // Ensure that we have a valid World and Significance Manager instance.
9     if (UWorld* World = GetWorld())
10    {
11        if (USignificanceManager* SignificanceManager = FSignificanceManagerModule::Get(World))
12        {
13            // Update once per frame, using only Player 0's world transform.
14            if (APawn *PlayerPawn = UGameplayStatics::GetPlayerPawn(World, 0))
15            {
16                // The Significance Manager uses an ArrayView. Construct a one-element Array to hold the Transform.
17                TArray<FTransform> TransformArray;
18                TransformArray.Add(PlayerPawn->GetTransform());
19                // Update the Significance Manager with our one-element Array passed in through an ArrayView.
20                SignificanceManager->Update(TArrayView<FTransform>(TransformArray));
21            }
22        }
23    }
24 }
25
```

 Copy full snippet

Project-Side Functionality

The Significance Manager only provides the framework for determining the significance of an Object, leaving the actual calculation to be defined by the project. When you register an Object with the Significance Manager, you also register functions matching the following types:

- `FSignificanceFunction`
- `FPostSignificanceFunction`

These functions will be called on the Object during Significance Manager updates.

FSignificanceFunction


This is the primary evaluation function that you must write in order to use the Significance Manager. It takes an Object parameter and a single Transform, and calculates the significance of the Object, which it returns as a `float`. During the Significance Manager's update process, this function will be called once for each Transform that was passed in. The final result will be determined by the Significance Manager's Update function; by default, it will be the highest value. Each registered Object is required to be associated with a function of type `FSignificanceFunction` when it is registered.

FPostSignificanceFunction

A function of this type will be supplied with the Object itself, its old significance value, its new significance value (unless the Object is being unregistered, in which case this value is one), and a `bool` indicating whether the Object is currently being unregistered. Unlike the significance-evaluation function, this has no return value. It is provided as a way for the game to handle changes to the Object's significance or place in the overall order of managed Objects. The Significance Manager will call this function based on how the Object was registered, as follows:

Post Significance Type

Behavior

None	Function is expected to be null. No post-significance-evaluation callback.
Concurrent	Function is expected not to be null, and will be called immediately upon evaluating the Object's significance. Functions called this way must be thread-safe, as they will run in parallel.
Sequential	<p>Function is expected not to be null, and will be called in sorted order with other Sequential Objects after all are evaluated for significance.</p> <div> The requirement for thread-safe code is waived in this case.</div>