Sparse Class Data

The Sparse Class Data system eliminates wasted memory from redundant properties.

Learn to use this **Beta** feature, but use caution when shipping with it.

The **Sparse Class Data** system saves memory by eliminating redundant data on commonly-used **Actor** types. While developing a game, Blueprint-exposed properties provide a great way for designers to iterate on Actor behavior. However, by the time the game ships, many of these properties may effectively be constants if their values do not vary across Actor instances or change during gameplay. With Sparse Class Data, you can transfer those properties into a single shared struct instance, keeping only one copy of the property in memory, while retaining the ability for designers to access its value in Blueprint graphs and edit it in the class defaults. To determine if a property is a good candidate for Sparse Class Data, use the following three-prong test. The property is a good candidate if:

1. It is a member of an Actor class that has many instances in-game at the same time, meaning that a significant amount of memory is tied up in redundant copies.
2. It does not change on placed Actor instances. In other words, the property does not need the `EditInstanceOnly` or `EditAnywhere` UProperty Specifiers, because no instance of the Actor overrides or changes its base value.

3. C++ code does not modify it. Any C++ code that directly accesses the variable must be replaced with calls to an accessor function.

> Implementing the Sparse Class Data feature requires native (C++) code. Any Blueprint-declared variables must move to C++ code to become eligible for this process.

Implementation Example

Once you have decided to use Sparse Class Data for one of your classes, you must identify the candidate properties. Any property tagged with `EditAnywhere`, `EditInstanceOnly`, or `BlueprintReadWrite` is not a candidate for Sparse Class Data. Similarly, any property that changes in native C++ code is ineligible to participate in the Sparse Class Data system. This is because the property could have a different value on one Actor instance from another, either through per-instance editing in the Level Editor, or by Blueprint Scripting or native code altering its value on a single Actor instance during a game session. The following example class contains several properties, some of which are candidates for Sparse Class Data:

```
1  // The properties in this class can all be changed in the editor, but on a
   per-class basis, not a per-instance basis.
2  UCLASS(BlueprintType)
3  class AMyActor : public AActor
4  {
5  GENERATED_UCLASS_BODY()
6
7  public:
8  // This property can be changed in the editor, but only on a per-class
   basis.
9  // Blueprint graphs cannot access or change it.
10 // It is a potential candidate for Sparse Class Data.
11 UPROPERTY(EditDefaultsOnly)
12 float MyFloatProperty;
13
14 // This property cannot be changed in the editor.
15 // Blueprint graphs can access it, but cannot change it.
16 // It is a potential candidate for Sparse Class Data.
17 UPROPERTY(BlueprintReadOnly)
```

```
int32 MyIntProperty;

// This property can be edited on a per-class basis in the editor.
// Blueprint graphs can access it, but cannot change it.
// It is a potential candidate for Sparse Class Data.
UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
FName MyNameProperty;

// This property can be edited on placed MyActor instances.
// It is not a potential candidate for Sparse Class Data.
UPROPERTY(EditAnywhere)
FVector MyVectorProperty;

// This property can be changed in Blueprint graphs.
// It is not a potential candidate for Sparse Class Data.
UPROPERTY(BlueprintReadWrite)
FVector2D MyVector2DProperty;
};

```

Copy full snippet

After identifying candidate variables, create a struct to contain them, and mark the struct with the `BlueprintType` UStruct Specifier. Each property in the struct must include the `EditDefaultsOnly` Specifier.

```
USTRUCT(BlueprintType)
struct FMySparseClassData
{
GENERATED_BODY()

FMySparseClassData()
: MyFloatProperty(0.f)
, MyIntProperty(0)
, MyNameProperty(NAME_None)
{ }

// You can set this property's default value in the editor.
// Blueprint graphs cannot access it.
UPROPERTY(EditDefaultsOnly)
float MyFloatProperty;
```

```
16
17 // This property's value will be set in C++ code.
18 // You can access it (but not change it) in Blueprint graphs.
19 UPROPERTY(EditDefaultsOnly, BlueprintReadOnly)
20 int32 MyIntProperty;
21
22 // You can set this property's default value in the editor.
23 // You can access it (but not change it) in Blueprint graphs.
24 // "GetByRef" means that Blueprint graphs access a const ref instead of a
   copy.
25 UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, meta=(GetByRef))
26 FName MyNameProperty;
27 };
28
```

Copy full snippet

> Unless you specify categories for the properties in your Sparse Class Data struct, they will receive the default name. In the case of `FMySparseClassData`, this would appear as "My Sparse Class Data" in the editor.

The original class needs some modification to use this struct. There are three specific changes that the process requires:

- Tell the class to use this struct as its Sparse Class Data type.
- Add `#if WITH_EDITORONLY_DATA` precompiler directive blocks around the properties that are moving to the new struct, and mark them with `_DEPRECATED` suffixes in the original class. Additionally, remove all UProperty Specifiers and set the properties to `private` access. Do not change other portions of your code to use the `_DEPRECATED` names; those lines will be replaced with accessor calls into the Sparse Class Data struct.
- In editor builds (`#if WITH_EDITOR`), override the `MoveDataToSparseClassDataStruct` function. This function will preserve the existing data values by performing a one-time copy from the original class to the Sparse Class Data struct.

After making these changes, your class should look like this:

```
1 UCLASS(BlueprintType, SparseClassDataTypes = MySparseClassData)
2 class AMyActor : public AActor
3 {
```

```
4  GENERATED_BODY()
5
6  #if WITH_EDITOR
7  public:
8  // ~ This function transfers existing data into FMySparseClassData.
9  virtual void MoveDataToSparseClassDataStruct() const override;
10 #endif // WITH_EDITOR
11
12 #if WITH_EDITORONLY_DATA
13 //~ These properties are moving out to the FMySparseClassData struct:
14 private:
15 // This property can be changed in the editor, but only on a per-class
   basis.
16 // Blueprint graphs cannot access or change it.
17 // It is a potential candidate for Sparse Class Data.
18 UPROPERTY()
19 float MyFloatProperty_DEPRECATED;
20
21 // This property cannot be changed in the editor.
22 // Blueprint graphs can access it, but cannot change it.
23 // It is a potential candidate for Sparse Class Data.
24 UPROPERTY()
25 int32 MyIntProperty_DEPRECATED;
26
27 // This property can be edited on a per-class basis in the editor.
28 // Blueprint graphs can access it, but cannot change it.
29 // It is a potential candidate for Sparse Class Data.
30 UPROPERTY()
31 FName MyNameProperty_DEPRECATED;
32 #endif // WITH_EDITORONLY_DATA
33
34 //~ The remaining properties can change on a per-instance basis and
35 //~ are therefore not involved in the Sparse Class Data implementation.
36 public:
37 // This property can be edited on placed MyActor instances.
38 // It is not a potential candidate for Sparse Class Data.
39 UPROPERTY(EditAnywhere)
40 FVector MyVectorProperty;
41
42 // This property can be changed in Blueprint graphs.
43 // It is not a potential candidate for Sparse Class Data.
44 UPROPERTY(BlueprintReadWrite)
```

```
45 FVector2D MyVector2DProperty;
46 };
47
```

Copy full snippet

The following function copies the existing values of all shared properties:

```
1  #if WITH_EDITOR
2  void AMyActor::MoveDataToSparseClassDataStruct() const
3  {
4  // make sure we don't overwrite the sparse data if it has been saved already
5  UBlueprintGeneratedClass* BPClass = Cast<UBlueprintGeneratedClass>
   (GetClass());
6  if (BPClass == nullptr || BPClass->bIsSparseClassDataSerializable == true)
7  {
8  return;
9  }
10
11 Super::MoveDataToSparseClassDataStruct();
12
13 #if WITH_EDITORONLY_DATA
14 // Unreal Header Tool (UHT) will create GetMySparseClassData automatically.
15 FMySparseClassData* SparseClassData = GetMySparseClassData();
16
17 // Modify these lines to include all Sparse Class Data properties.
18 SparseClassData->MyFloatProperty = MyFloatProperty_DEPRECATED;
19 SparseClassData->MyIntProperty = MyIntProperty_DEPRECATED;
20 SparseClassData->MyNameProperty = MyNameProperty_DEPRECATED;
21 #endif // WITH_EDITORONLY_DATA
22 }
23 #endif // WITH_EDITOR
```

Copy full snippet

You may need to include `Engine/BlueprintGeneratedClass.h` to compile the function above.

At this point, Sparse Class Data is in place. Users editing or accessing the affected properties in the editor will not notice any difference in behavior, and memory usage in shipping builds

will be reduced. If the properties are referenced in C++ code, replace any attempts to access the variable with a call to a getter function. For example, anywhere that the code used to access the `MyFloatProperty` variable should now call `GetMyFloatProperty` instead. UHT will automatically generate this function for you. If you have a non-trivial getter function and need to keep its behaviors, the `NoGetter` UProperty Metadata Specifier will instruct UHT not to generate its own getter functions. For variables that you want to access by const reference instead of by value, use the `GetByRef` UProperty Metadata Specifier.