

Properties

Reference for creating and implementing properties for gameplay classes.



Property Declaration

Properties are declared using standard C++ variable syntax, preceded by the UPROPERTY macro which defines property metadata and variable specifiers.

```
1 UPROPERTY([specifier, specifier, ...], [meta(key=value, key=value, ...)])
2 Type VariableName;
3
```

 Copy full snippet

Core Data Types

Integers

The convention for integral data types is "int" or "uint" followed by the size in bits.

Variable Type	Description
uint8	8-bit unsigned
uint16	16-bit unsigned
uint32	32-bit unsigned
uint64	64-bit unsigned
int8	8-bit signed

Variable Type	Description
int16	16-bit signed
int32	32-bit signed
int64	64-bit signed

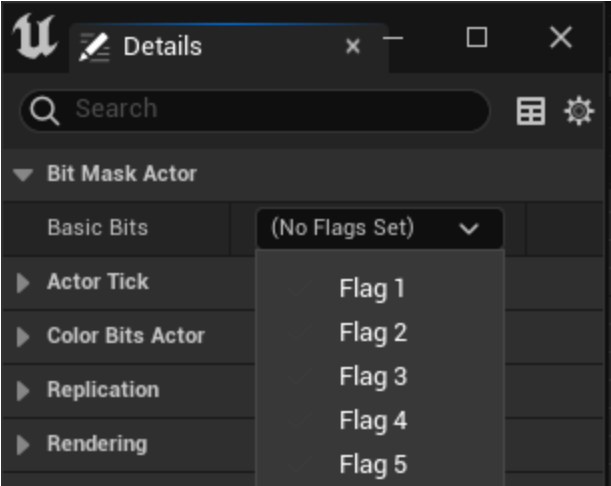
As Bitmasks

Integer properties can now be exposed to the Editor as bitmasks. To mark an integer property as a bitmask, just add "bitmask" to the meta section, as follows:

```
1 /*~ BasicBits appears as a list of generic flags in the editor, instead of an
   integer field. */
2 UPROPERTY(EditAnywhere, Meta = (Bitmask))
3 int32 BasicBits;
4
```

 Copy full snippet

Adding this meta tag will cause the integer to be editable as a drop-down list of generically-named flags ("Flag 1", "Flag 2", "Flag 3", etc.) that can be turned on or off individually.



You can also make integer parameters to Blueprint-callable functions behave as bitmasks, by adding the `Bitmask` meta tag (no value necessary) to a `UPARAM` Specifier for the parameter.

```
1 /*~ You can set MyFunction using a generic list of flags instead of typing in
   an integer value. */
2 UFUNCTION(BlueprintCallable)
3 void MyFunction(UPARAM(meta=(Bitmask)) int32 BasicBitsParam)
4
5
```

 Copy full snippet

In order to customize the bitflags' names, we must first create a UENUM with the "bitflags" meta tag:

```
1 UENUM(Meta = (Bitflags))
2 enum class EColorBits
3 {
4   ECB_Red,
5   ECB_Green,
6   ECB_Blue
7 };
8
```

 Copy full snippet



The supported value range for a bitmask enumerated type is 0 to 31, inclusive. This corresponds to the bits (starting at bit 0) of a 32-bit integer variable. In the example above, bit 0 is `ECB_Red`, bit 1 is `ECB_Green`, and bit 2 is `ECB_Blue`.

As an alternate declaration style, you can use the `ENUM_CLASS_FLAGS` to turn your enumerated type into a bitmask after defining it. In order to use the flag selector in the editor, we must also add the meta field `UseEnumValuesAsMaskValuesInEditor` and set it to `true`. The key difference is that this method uses the mask values directly, instead of the bit numbers. The equivalent enumerated type, made using this method, would look like this:

```
1 UENUM(Meta = (Bitflags, UseEnumValuesAsMaskValuesInEditor = "true"))
2 enum class EColorBits
3 {
4   ECB_Red = 0x01,
5   ECB_Green = 0x02,
6   ECB_Blue = 0x04
7 };
8
9 ENUM_CLASS_FLAGS(EColorBits);
```

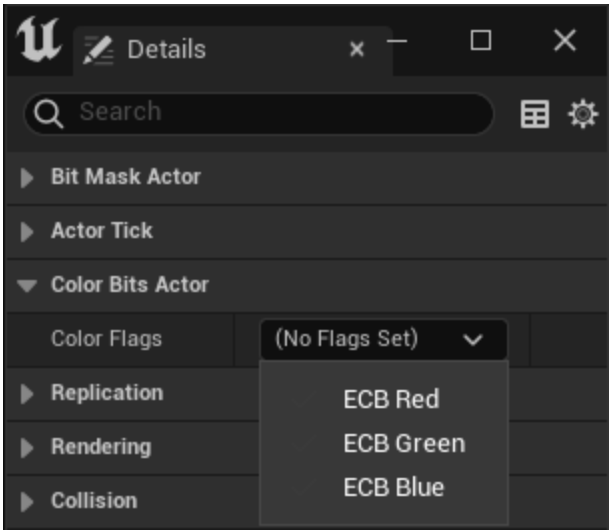
 Copy full snippet

After creating this UENUM, we can reference it with the "BitmaskEnum" meta tag, like this:

```
1 /*~ This property lists flags matching the names of values from EColorBits.
   */
2 UPROPERTY(EditAnywhere, Meta = (Bitmask, BitmaskEnum = "EColorBits"))
3 int32 ColorFlags;
4
```

 Copy full snippet

Following this change, the bitflags listed in the drop-down box will take on the names and values of the enumerated class entries. In the example above, ECB_Red is value 0, meaning it will activate bit 0 (adding 1 to ColorFlags) when checked. ECB_Green corresponds to bit 1 (adding 2 to ColorFlags), and ECB_Blue corresponds to bit 2 (adding 4 to ColorFlags).



Similarly, you can add `BitmaskEnum` and the appropriate enumerated type name to the meta section of your `UPARAM` tag to customize it.

```
1 /*~ MyOtherFunction shows flags named after the values from EColorBits. */
2 UFUNCTION(BlueprintCallable)
3 void MyOtherFunction(UPARAM(meta=(Bitmask, BitmaskEnum = "EColorBits")) int32
  ColorFlagsParam)
4
5
```

 Copy full snippet



While enumerated types can contain more than 32 entries, only the first 32 values will be visible in a bitmask association in the Property Editor UI. Similarly, while explicit-value entries are accepted, entries with explicit values not between 0 and 31 will not be included in the drop-down.

Floating Point Types

Unreal uses the standard C++ floating point types, float, and double.

Boolean Types

Boolean types can be represented either with the C++ bool keyword or as a bitfield.

```
1 uint32 bIsHungry : 1;
2 bool bIsThirsty;
3
```

 Copy full snippet

Strings

Unreal Engine 4 supports three core types of strings.

- FString is a classic "dynamic array of chars" string type.
- FName is a reference to an immutable case-insensitive string in a global string table. It is smaller and more efficient to pass around than an FString, but more difficult to manipulate.
- FText is a more robust string representation designed to handle localization.

For most uses, Unreal relies on the TCHAR type for characters. The TEXT() macro is available to denote TCHAR literals.

```
1 MyDogPtr->DogName = FName(TEXT("Samson Aloysius"));
2
```

 Copy full snippet

For more on the three string types, when to use each one, and how to work with them, see the [String Handling documentation](#).

Property Specifiers

When declaring properties, **Property Specifiers** can be added to the declaration to control how the property behaves with various aspects of the Engine and Editor.

Property Tag	Effect
<code>AdvancedDisplay</code>	The property will be placed in the advanced (dropdown) section of any panel where it appears.
<code>AssetRegistrySearchable</code>	The <code>AssetRegistrySearchable</code> Specifier indicates that this property and its value will be automatically added to the Asset Registry for any Asset class instances containing this as a member variable. It is not legal to use struct properties or parameters.
<code>BlueprintAssignable</code>	Usable with Multicast Delegates only. Exposes the property for assigning in Blueprints.
<code>BlueprintAuthorityOnly</code>	This property must be a Multicast Delegate. In Blueprints, it will only accept events tagged with <code>BlueprintAuthorityOnly</code> .
<code>BlueprintCallable</code>	Multicast Delegates only. Property should be exposed for calling in Blueprint code.
<code>BlueprintGetter=GetterFunctionName</code>	This property specifies a custom accessor function. If this property isn't also tagged with <code>BlueprintSetter</code> or <code>BlueprintReadWrite</code> , then it is implicitly <code>BlueprintReadOnly</code> .

Property Tag

Effect

<code>BlueprintReadOnly</code>	This property can be read by Blueprints, but not modified. This Specifier is incompatible with the <code>BlueprintReadWrite</code> Specifier.
<code>BlueprintReadWrite</code>	This property can be read or written from a Blueprint. This Specifier is incompatible with the <code>BlueprintReadOnly</code> Specifier.
<code>BlueprintSetter=SetterFunctionName</code>	This property has a custom mutator function, and is implicitly tagged with <code>BlueprintReadWrite</code> . Note that the mutator function must be named and part of the same class.
<code>Category="TopCategory\ SubCategory\ . . ."</code>	Specifies the category of the property when displayed in Blueprint editing tools. Define nested categories using the operator.
<code>Config</code>	This property will be made configurable. The current value can be saved to the <code>.ini</code> file associated with the class and will be loaded when created. Cannot be given a value in default properties. Implies <code>BlueprintReadOnly</code> .
<code>DuplicateTransient</code>	Indicates that the property's value should be reset to the class default value during any type of duplication (copy/paste, binary duplication, etc.).
<code>EditAnywhere</code>	Indicates that this property can be edited by property windows, on archetypes and instances. This Specifier is incompatible with any of the "Visible" Specifiers.
<code>EditDefaultsOnly</code>	Indicates that this property can be edited by property windows, but only on archetypes. This Specifier is incompatible with any of the "Visible" Specifiers.
<code>EditFixedSize</code>	Only useful for dynamic arrays. This will prevent the user from changing the length of an array via the Unreal Editor property window.
<code>EditInstanceOnly</code>	Indicates that this property can be edited by property windows, but only on instances, not on archetypes. This Specifier is incompatible with any of the "Visible" Specifiers.
<code>Export</code>	Only useful for Object properties (or arrays of Objects). Indicates that the Object assigned to this property should be exported in its entirety as a

Property Tag

Effect


	subobject block when the Object is copied (such as for copy/paste operations), as opposed to just outputting the Object reference itself.
GlobalConfig	Works just like Config except that you cannot override it in a subclass. Cannot be given a value in default properties. Implies BlueprintReadOnly.
Instanced	Object (UCLASS) properties only. When an instance of this class is created, it will be given a unique copy of the Object assigned to this property in defaults. Used for instanting subobjects defined in class default properties. Implies EditInline and Export.
Interp	Indicates that the value can be driven over time by a Track in Sequencer.
Localized	The value of this property will have a localized value defined. Mostly used for strings. Implies ReadOnly.
Native	Property is native: C++ code is responsible for serializing it and exposing it to Garbage Collection.
NoClear	Prevents this Object reference from being set to none from the editor. Hides the clear (and browse) button in the editor.
NoExport	Only useful for native classes. This property should not be included in the auto-generated class declaration.
NonPIEDuplicateTransient	The property will be reset to the default value during duplication, except if it's being duplicated for a Play In Editor (PIE) session.
NonTransactional	Indicates that changes to this property's value will not be included in the editor's undo/redo history.
NotReplicated	Skip replication. This only applies to struct members and parameters in service request functions.
Replicated	The property should be replicated over the network.

Property Tag	Effect
<code>ReplicatedUsing=FunctionName</code>	The <code>ReplicatedUsing</code> Specifier specifies a callback function which is executed when the property is updated over the network.
<code>RepRetry</code>	Only useful for struct properties. Retry replication of this property if it fails to be fully sent (for example, Object references not yet available to serialize over the network). For simple references, this is the default, but for structs, this is often undesirable due to the bandwidth cost, so it is disabled unless this flag is specified.
<code>SaveGame</code>	This Specifier is a simple way to include fields explicitly for a checkpoint/save system at the property level. The flag should be set on all fields that are intended to be part of a saved game, and then a proxy archiver can be used to read/write it.
<code>SerializeText</code>	Native property should be serialized as text (<code>ImportText</code> , <code>ExportText</code>).
<code>SkipSerialization</code>	This property will not be serialized, but can still be exported to a text format (such as for copy/paste operations).
<code>SimpleDisplay</code>	Visible or editable properties appear in the Details panel and are visible without opening the "Advanced" section.
<code>TextExportTransient</code>	This property will not be exported to a text format (so it cannot, for example, be used in copy/paste operations).
<code>Transient</code>	Property is transient, meaning it will not be saved or loaded. Properties tagged this way will be zero-filled at load time.
<code>VisibleAnywhere</code>	Indicates that this property is visible in all property windows, but cannot be edited. This Specifier is incompatible with the "Edit" Specifiers.
<code>VisibleDefaultsOnly</code>	Indicates that this property is only visible in property windows for archetypes, and cannot be edited. This Specifier is incompatible with any of the "Edit" Specifiers.

Property Tag	Effect
<code>VisibleInstanceOnly</code>	Indicates that this property is only visible in property windows for instances, not for archetypes, and cannot be edited. This Specifier is incompatible with any of the "Edit" Specifiers.

Metadata Specifiers

When declaring classes, interfaces, structs, enums, enum values, functions, or properties, you can add **Metadata Specifiers** to control how they interact with various aspects of the engine and editor. Each type of data structure or member has its own list of Metadata Specifiers.


Metadata only exists in the editor; do not write game logic that accesses metadata.

Property Meta Tag	Effect
<code>AllowAbstract="true/false"</code>	Used for <code>Subclass</code> and <code>SoftClass</code> properties. Indicates whether abstract Class types should be shown in the Class picker.
<code>AllowedClasses="Class1, Class2, .."</code>	Used for <code>FSoftObjectPath</code> properties. Comma delimited list that indicates the Class type(s) of assets to be displayed in the Asset picker.
<code>AllowPreserveRatio</code>	Used for <code>FVector</code> properties. It causes a ratio lock to be added when displaying this property in details panels.
<code>ArrayClamp="ArrayProperty"</code>	Used for integer properties. Clamps the valid values that can be entered in the UI to be between 0 and the length of the array property named.
<code>AssetBundles</code>	Used for <code>SoftObjectPtr</code> or <code>SoftObjectPath</code> properties. List of Bundle names used inside Primary Data Assets to specify which Bundles this reference is part of.
<code>BlueprintBaseOnly</code>	Used for <code>Subclass</code> and <code>SoftClass</code> properties. Indicates whether only Blueprint Classes should be shown in the Class picker.
<code>BlueprintCompilerGeneratedDefaults</code>	Property defaults are generated by the Blueprint compiler and will not be copied when the

Property Meta Tag

Effect

	<code>CopyPropertiesForUnrelatedObjects</code> function is called post-compile.
<code>ClampMin="N"</code>	Used for float and integer properties. Specifies the minimum value <code>N</code> that may be entered for the property.
<code>ClampMax="N"</code>	Used for float and integer properties. Specifies the maximum value <code>N</code> that may be entered for the property.
<code>ConfigHierarchyEditable</code>	This property is serialized to a config (<code>.ini</code>) file, and can be set anywhere in the config hierarchy.
<code>ContentDir</code>	Used by <code>FDirectoryPath</code> properties. Indicates that the path will be picked using the Slate-style directory picker inside the <code>Content</code> folder.
<code>DisplayAfter="PropertyName"</code>	This property will show up in the Blueprint Editor immediately after the property named <code>PropertyName</code> , regardless of its order in source code, as long as both properties are in the same category. If multiple properties have the same <code>DisplayAfter</code> value and the same <code>DisplayPriority</code> value, they will appear after the named property in the order in which they are declared in the header file.
<code>DisplayName="Property Name"</code>	The name to display for this property, instead of the code-generated name.
<code>DisplayPriority="N"</code>	If two properties feature the same <code>DisplayAfter</code> value, or are in the same category and do not have the <code>DisplayAfter</code> Meta Tag, this property will determine their sorting order. The highest-priority value is 1, meaning that a property with a <code>DisplayPriority</code> value of 1 will appear above a property with a <code>DisplayPriority</code> value of 2. If multiple properties have the same <code>DisplayAfter</code> value, they will appear in the order in which they are declared in the header file.
<code>DisplayThumbnail="true"</code>	Indicates that the property is an Asset type and it should display the thumbnail of the selected Asset.

Property Meta Tag

Effect

<div>EditCondition="BooleanPropertyName"</div>	<p>Names a boolean property that is used to indicate whether editing of this property is disabled. Putting "!" before the property name inverts the test.</p> <div><div><div>i</div><div><p>The EditCondition meta tag is no longer limited to a single boolean property. It is now evaluated using a full-fledged expression parser, meaning you can include a full C++ expression.</p></div></div></div>
<div>EditFixedOrder</div>	<p>Keeps the elements of an array from being reordered by dragging.</p>
<div>ExactClass="true"</div>	<p>Used for <code>FSoftObjectPath</code> properties in conjunction with <code>AllowedClasses</code>. Indicates whether only the exact Classes specified in <code>AllowedClasses</code> can be used, or if subclasses are also valid.</p>
<div>ExposeFunctionCategories="Category1, Category2, .."</div>	<p>Specifies a list of categories whose functions should be exposed when building a function list in the Blueprint Editor.</p>
<div>ExposeOnSpawn="true"</div>	<p>Specifies whether the property should be exposed on a Spawn Actor node for this Class type.</p>
<div>FilePathFilter="FileType"</div>	<p>Used by <code>FFilePath</code> properties. Indicates the path filter to display in the file picker. Common values include "uasset" and "umap", but these are not the only possible values.</p>
<div>GetByRef</div>	<p>Makes the "Get" Blueprint Node for this property return a const reference to the property instead of a copy of its value. Only usable with Sparse Class Data, and only when <code>NoGetter</code> is not present.</p>
<div>HideAlphaChannel</div>	<p>Used for <code>FColor</code> and <code>FLinearColor</code> properties. Indicates that the <code>Alpha</code> property should be hidden when displaying the property widget in the details.</p>
<div>HideViewOptions</div>	<p>Used for <code>Subclass</code> and <code>SoftClass</code> properties. Hides the ability to change view options in the</p>

Property Meta Tag

Effect

	Class picker.
<code>InlineEditConditionToggle</code>	Signifies that the boolean property is only displayed inline as an edit condition toggle in other properties, and should not be shown on its own row.
<code>LongPackageName</code>	Used by <code>FDirectoryPath</code> properties. Converts the path to a long package name.
<code>MakeEditWidget</code>	Used for Transform or Rotator properties, or Arrays of Transforms or Rotators. Indicates that the property should be exposed in the viewport as a movable widget.
<code>NoGetter</code>	Causes Blueprint generation not to generate a "get" Node for this property. Only usable with Sparse Class Data.
<code>ScriptName="DisplayName"</code>	The name to use for this clas, property, or function when exporting it to a scripting language. You may include deprecated names as additional semi-colon-separated entries.