Metadata Specifiers

Metadata keywords used when declaring UClasses, UFunctions, UProperties, UEnums, and UInterfaces to specify how they behave with various aspects of Unreal Engine and the editor

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.

To add Metadata Specifiers, use the word `meta`, followed by a list of the specifiers and, if appropriate, their values, to your `UCLASS`, `UENUM`, `UINTERFACE`, `USTRUCT`, `UFUNCTION`, or `UPROPERTY` macro, as follows:

```
{UCLASS/UENUM/UINTERFACE/USTRUCT/UFUNCTION/UPROPERTY}(SpecifierX, meta=(MetaTag1
```

Copy full snippet

To add Metadata Specifiers to a specific value within an enumerated type, add the `UMETA` tag to the value itself before the separating comma (if one exists). It should look like:

```
UENUM()
enum class EMyEnum : uint8
{
// DefaultValue Tooltip
DefaultValue = 0 UMETA(MetaTag1="Value1", MetaTag2, ..),

// ValueWithoutMetaSpecifiers Tooltip
ValueWithoutMetaSpecifiers,

// ValueWithMetaSpecifiers Tooltip
```

```
ValueWithMetaSpecifiers UMETA((MetaTag1="Value1", MetaTag2, ..),

// FinalValue Tooltip
FinalValue (MetaTag1="Value1", MetaTag2, ..)
};
```

Copy full snippet

Class Metadata Specifiers

Classes can use the following Metatag Specifiers:

Class Meta Tag	Effect
`BlueprintSpawnableComponent`	If present, the component Class can be spawned by a Blueprint.
`BlueprintThreadSafe`	Only valid on Blueprint function libraries. This specifier marks the functions in this class as callable on non-game threads in animation Blueprints.
`ChildCannotTick`	Used for Actor and Component classes. If the native class cannot tick, Blueprint-generated classes based on this Actor or Component can never tick, even if `bCanBlueprintsTickByDefault` is true.
`ChildCanTick`	Used for Actor and Component classes. If the native class cannot tick, Blueprint-generated classes based on this Actor or Component can have the `bCanEverTick` flag overridden, even if `bCanBlueprintsTickByDefault` is false.
`DeprecatedNode`	For behavior tree nodes, indicates that the class is deprecated and will display a warning when compiled.
`DeprecationMessage="Message Text"`	Deprecated classes with this metadata will include this text with the standard deprecation warning that Blueprint Scripts generate during compilation.
`DisplayName="Blueprint Node Name"`	The name of this node in a Blueprint Script will be replaced with the value provided here, instead of the code-generated name.
`DontUseGenericSpawnObject`	Do not spawn an Object of the class using Generic Create Object node in Blueprint Scripts;

Class Meta Tag	Effect
	this specifier applies only to Blueprint-type classes that are neither Actors nor Actor Components.
`ExposedAsyncProxy`	Expose a proxy Object of this class in Async Task nodes.
`IgnoreCategoryKeywordsInSubclasses`	Used to make the first subclass of a class ignore all inherited `ShowCategories` and `HideCategories` Specifiers.
`IsBlueprintBase="true/false"`	States that this class is (or is not) an acceptable base class for creating Blueprints, similar to the `Blueprintable` or `NotBlueprintable` Specifiers.
`KismetHideOverrides="Event1, Event2, .."`	List of Blueprint events that are not allowed to be overridden.
`ProhibitedInterfaces="Interface1, Interface2, .."`	Lists Interfaces that are not compatible with the class.
`RestrictedToClasses="Class1, Class2, .."`	Blueprint function library classes can use this to restrict usage to the classes named in the list.
`ShortToolTip="Short tooltip"`	A short tooltip that is used in some contexts where the full tooltip might be overwhelming, such as the Parent Class Picker dialog.
`ShowWorldContextPin`	Indicates that Blueprint nodes placed in graphs owned by this class must show their World context pins, even if they are normally hidden, because Objects of this class cannot be used as World context.
`UsesHierarchy`	Indicates the class uses hierarchical data. Used to instantiate hierarchical editing features in Details panels.
`ToolTip="Hand-written tooltip"`	Overrides the automatically generated tooltip from code comments.
`ScriptName="DisplayName"`	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.

Enum Metadata Specifiers

Enumerated types can use the following Metadata Specifiers:

Enumerated Type Meta Tag	Effect
`Bitflags`	Indicates that this enumerated type can be used as flags by integer `UPROPERTY` variables that are set up with the `Bitmask` Metadata Specifier.
`Experimental`	Labels this type as experimental and unsupported.
`ToolTip="Hand-written tooltip"`	Overrides the automatically generated tooltip from code comments.

Individual values within an enumerated type have their own Metadata Specifiers. These differ slightly from other Metadata Specifiers in that they use top-level keyword `UMETA`, and are specified after the value they modify, rather than before.

Enumerated Value UMeta Tag	**Effect**
`DisplayName="Enumerated Value Name"`	This value's name will be the text provided here, rather than the code-generated name.
`Hidden`	This value will not appear in the Editor.
`ToolTip="Hand-written tooltip."`	Overrides the automatically generated tooltip from code comments.

Interface Metadata Specifiers

Interfaces can use the following Metatag Specifier:

Ensure that Blueprint events are only allowed in implementable interfaces. Internal only functions allowed Ensure that if this interface contains Blueprint callable functions that are not Blueprint defined, that it must be implemented natively

Interface Meta Tag	Effect
`CannotImplementInterfaceInBlueprint`	This interface may not contain `BlueprintImplementableEvent` or `BlueprintNativeEvent` functions, other than internal-only functions. If it contains Blueprint-callable functions that are not blueprint-defined, those functions must be implemented in native code.

Struct Metadata Specifiers

Structs can use the following Metatag Specifiers:

Struct Meta Tag	Effect
`HasNativeBreak="Module.Class.Function"`	Indicates that this struct has a custom Break Struct node. The module, class, and function name must be provided.
`HasNativeMake="Module.Class.Function"`	Indicates that this struct has a custom Make Struct node. The module, class, and function name must be provided.
`HiddenByDefault`	Pins in Make Struct and Break Struct nodes are hidden by default.
`ShortToolTip="Short tooltip"`	A short tooltip that is used in some contexts where the full tooltip might be overwhelming, such as the Parent Class Picker dialog.
`ToolTip="Hand-written tooltip`	Overrides the automatically generated tooltip from code comments.

Function Metadata Specifiers

Function Meta Tag	Effect
`AdvancedDisplay="Parameter1, Parameter2, .."`	The comma-separated list of parameters will show up as advanced pins (requiring UI expansion).
`AdvancedDisplay=N`	Replace `N` with a number, and all parameters after the Nth will show up as advanced pins (requiring UI expansion). For example, 'AdvancedDisplay=2' will mark all but the first two parameters as advanced).
`ArrayParm="Parameter1, Parameter2, .."`	Indicates that a `BlueprintCallable` function should use a Call Array Function node and that the listed parameters should be treated as wild card array properties.
`ArrayTypeDependentParams="Parameter"`	When `ArrayParm` is used, this specifier indicates one parameter which will determine the types of all parameters in the `ArrayParm` list.

Function Meta Tag	Effect
`AutoCreateRefTerm="Parameter1, Parameter2, .."`	The listed parameters, although passed by reference, will have an automatically created default if their pins are left disconnected. This is a convenience feature for Blueprints, often used on array pins.
`BlueprintAutocast`	Used only by static `BlueprintPure` functions from a Blueprint function library. A cast node will be automatically added for the return type and the type of the first parameter of the function.
`BlueprintInternalUseOnly`	This function is an internal implementation detail, used to implement another function or node. It is never directly exposed in a Blueprint graph.
`BlueprintProtected`	This function can only be called on the owning Object in a Blueprint. It cannot be called on another instance.
`CallableWithoutWorldContext`	Used for `BlueprintCallable` functions that have a `WorldContext` pin to indicate that the function can be called even if its Class does not implement the `GetWorld` function.
`CommutativeAssociativeBinaryOperator`	Indicates that a `BlueprintCallable` function should use the Commutative Associative Binary node. This node lacks pin names, but features an **Add Pin** button that creates additional input pins.
`CompactNodeTitle="Name"`	Indicates that a `BlueprintCallable` function should display in the compact display mode, and provides the name to display in that mode.
`CustomStructureParam="Parameter1, Parameter2, .."`	The listed parameters are all treated as wildcards. This specifier requires the `UFUNCTION`-level specifier, `CustomThunk`, which will require the user to provide a custom `exec` function. In this function, the parameter types can be checked and the appropriate function calls can be made based on those parameter types. The base `UFUNCTION` should never be called, and should assert or log an error if it is. To declare a custom `exec` function, use the syntax `DECLARE_FUNCTION(execMyFunctionName)`

Function Meta Tag	Effect
	where `MyFunctionName` is the name of the original function.
`DefaultToSelf`	For `BlueprintCallable` functions, this indicates that the Object property's named default value should be the self context of the node.
`DeprecatedFunction`	Any Blueprint references to this function will cause compilation warnings telling the user that the function is deprecated. You can add to the deprecation warning message (for example, to provide instructions on replacing the deprecated function) using the `DeprecationMessage` metadata specifier.
`DeprecationMessage`="Message Text"	If the function is deprecated, this message will be added to the standard deprecation warning when trying to compile a Blueprint that uses it.
`DeterminesOutputType="Parameter"`	The return type of the function will dynamically change to match the input that is connected to the named parameter pin. The parameter should be a templated type like `TSubClassOf<X>` or `TSoftObjectPtr<X>`, where the function's original return type is `X*` or a container with `X*` as the value type, such as `TArray<X*>`.
`DevelopmentOnly`	Functions marked as `DevelopmentOnly` will only run in Development mode. This is useful for functionality like debug output, which is expected not to exist in shipped products.
`DisplayName="Blueprint Node Name"`	The name of this node in a Blueprint will be replaced with the value provided here, instead of the code-generated name.
`ExpandEnumAsExecs="Parameter"`	For `BlueprintCallable` functions, this indicates that one input execution pin should be created for each entry in the `enum` used by the parameter. The parameter must be of an enumerated type that has the `UENUM` tag.
`ForceAsFunction`	Change a `BlueprintImplementableEvent` with no return value from an event into a function.
`HidePin="Parameter"`	For `BlueprintCallable` functions, this indicates that the parameter pin should be hidden from the

Function Meta Tag	Effect
	user's view. Only one pin per function can be hidden in this manner.
`HideSelfPin`	Hides the "self" pin, which indicates the object on which the function is being called. The "self" pin is automatically hidden on `BlueprintPure` functions that are compatible with the calling Blueprint's Class. Functions that use the `HideSelfPin` Meta Tag frequently also use the `DefaultToSelf` Specifier.
`InternalUseParam="Parameter"`	Similar to `HidePin`, this hides the named parameter's pin from the user's view, and can only be used for one parameter per function.
`KeyWords="Set Of Keywords"`	Specifies a set of keywords that can be used when searching for this function, such as when placing a node to call the function in a Blueprint Graph.
`Latent`	Indicates a latent action. Latent actions have one parameter of type `FLatentActionInfo`, and this parameter is named by the `LatentInfo` specifier.
`LatentInfo="Parameter"`	For Latent `BlueprintCallable` functions, indicates which parameter is the LatentInfo parameter.
`MaterialParameterCollectionFunction`	For `BlueprintCallable` functions, indicates that the material override node should be used.
`NativeBreakFunc`	For `BlueprintCallable` functions, indicates that the function should be displayed the same way as a standard Break Struct node.
`NotBlueprintThreadSafe`	Only valid in Blueprint function libraries. This function will be treated as an exception to the owning Class's general `BlueprintThreadSafe` metadata.
`ShortToolTip="Short tooltip"`	A short tooltip that is used in some contexts where the full tooltip might be overwhelming, such as the Parent Class Picker dialog.
`ToolTip="Hand-written tooltip`	Overrides the automatically generated tooltip from code comments.

Function Meta Tag	Effect
`UnsafeDuringActorConstruction`	This function is not safe to call during Actor construction.
`WorldContext="Parameter"`	Used by `BlueprintCallable` functions to indicate which parameter determines the World in which the operation takes place.
`ScriptName="DisplayName"`	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.

Property Metadata Specifiers

Property Meta Tag	Effect
`AllowAbstract="true/false"`	Used for `Subclass` and `SoftClass` properties. Indicates whether abstract Class types should be shown in the Class picker.
`AllowedClasses="Class1, Class2, .."`	Used for `FSoftObjectPath` properties. Comma delimited list that indicates the Class type(s) of assets to be displayed in the Asset picker.
`AllowPreserveRatio`	Used for `FVector` properties. It causes a ratio lock to be added when displaying this property in details panels.
`ArrayClamp="ArrayProperty"`	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.
`AssetBundles`	Used for `SoftObjectPtr` or `SoftObjectPath` properties. List of Bundle names used inside Primary Data Assets to specify which Bundles this reference is part of.
`BlueprintBaseOnly`	Used for `Subclass` and `SoftClass` properties. Indicates whether only Blueprint Classes should be shown in the Class picker.

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

Property Meta Tag	Effect
`DisplayThumbnail="true"`	Indicates that the property is an Asset type and it should display the thumbnail of the selected Asset.
`EditCondition="BooleanPropertyName"`	Names a boolean property that is used to indicate whether editing of this property is disabled. Putting "!" before the property name inverts the test. 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.
`EditFixedOrder`	Keeps the elements of an array from being reordered by dragging.
`ExactClass="true"`	Used for `FSoftObjectPath` properties in conjunction with `AllowedClasses`. Indicates whether only the exact Classes specified in `AllowedClasses` can be used, or if subclasses are also valid.
`ExposeFunctionCategories="Category1, Category2, .."`	Specifies a list of categories whose functions should be exposed when building a function list in the Blueprint Editor.
`ExposeOnSpawn="true"`	Specifies whether the property should be exposed on a Spawn Actor node for this Class type.
`FilePathFilter="FileType"`	Used by `FFilePath` 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.
`GetByRef`	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 `NoGetter` is not present.
`HideAlphaChannel`	Used for `FColor` and `FLinearColor` properties. Indicates that the `Alpha` property

Property Meta Tag	Effect
	should be hidden when displaying the property widget in the details.
`HideViewOptions`	Used for `Subclass` and `SoftClass` properties. Hides the ability to change view options in the Class picker.
`InlineEditConditionToggle`	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.
`LongPackageName`	Used by `FDirectoryPath` properties. Converts the path to a long package name.
`MakeEditWidget`	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.
`NoGetter`	Causes Blueprint generation not to generate a "get" Node for this property. Only usable with Sparse Class Data.
`ScriptName="DisplayName"`	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.