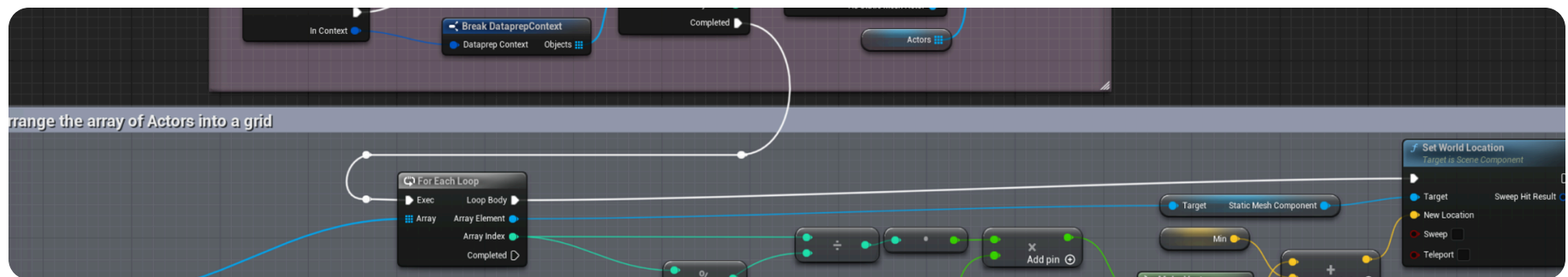


- Developer
- / Documentation
- / Unreal Engine ▾
- / Unreal Engine 5.4 Documentation
- / Working with Content
- / Datasmith
- / Dataprep Import Customization
- / Creating Custom Dataprep Blocks

# Creating Custom Dataprep Blocks

How to create your own custom filters and operators for the Dataprep system in Blueprint.



The [Visual Dataprep](#) system comes with dozens of ready-made filters and operations to handle many of the most common needs for preparing 3D data for real-time visualization. However, no pre-set list of filters and operations could exhaustively support every operation you might need to carry out. If you need your Dataprep graph to do something that is available in the Unreal Editor's scripting system, but that none of the ready-made Dataprep blocks will do for you, you can create your own custom filter or operator to do exactly what you need.

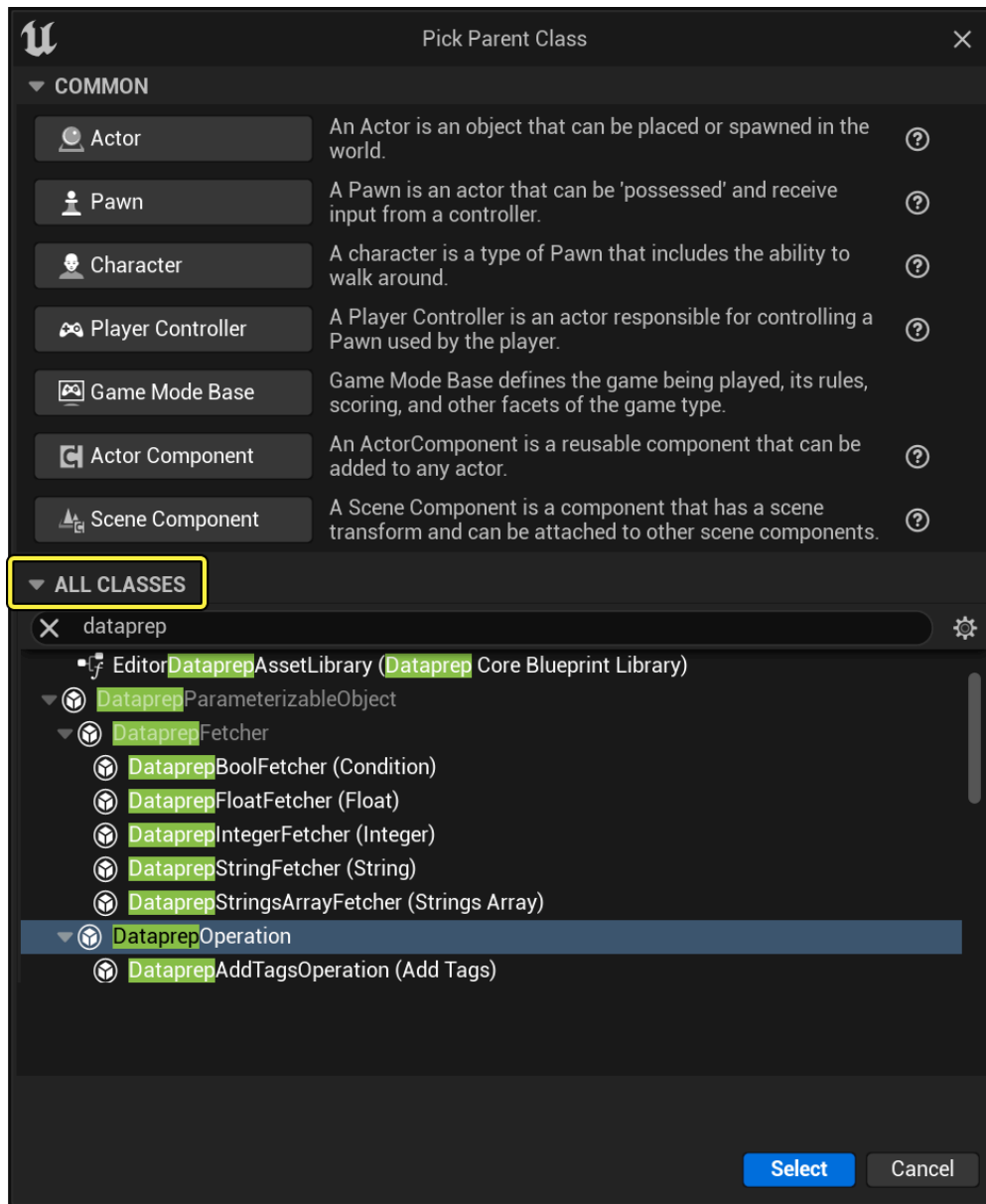
This page describes how to create new filters and operators for your Dataprep graphs.

To use the Dataprep operations in Blueprints, enable the **Dataprep Editor** plugin. For more information on plugins, see [Working with Plugins](#).

# How to Create a Custom Dataprep Block

The overall process is the same for all different types of Dataprep blocks.

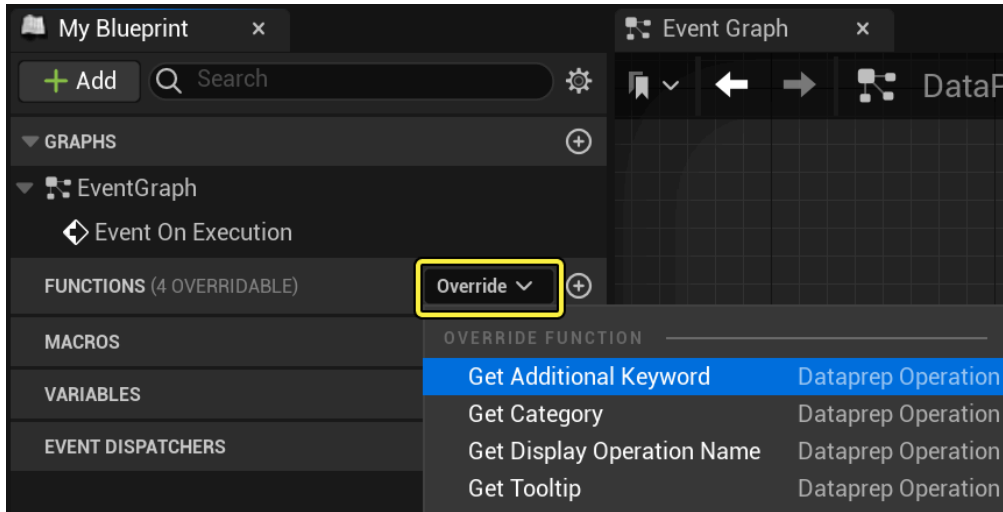
1. Create a new Blueprint class by right-clicking in the **Content Browser** and selecting **Blueprint Class**. For details, see [Creating Blueprint Classes](#).
2. In the **Pick Parent Class** dialog, expand the **All Classes** list at the bottom. Choose the base class that corresponds to the type of block you want to create, and click **Select**. See the table of [Base Classes](#) below for the list of Dataprep parent classes you can choose from.



*Click for full image.*

3. Rename your new class in the **Content Browser** if desired.
4. Double-click your new class to open it in the Blueprint Editor.

5. Override the functions that your class inherits from its parent class. By overriding functions, you can customize the behavior of the dataprep block when the graph gets executed. To override a function, hover over the **Functions** category in the **My Blueprint** panel. Click the **Override** drop-down that appears, then choose the function you want to override.



6. **Compile** and **Save** your Blueprint class.

The next time you open a Dataprep Asset, you should see your new node added to the palette to the left of the graph editor.

## Base Classes

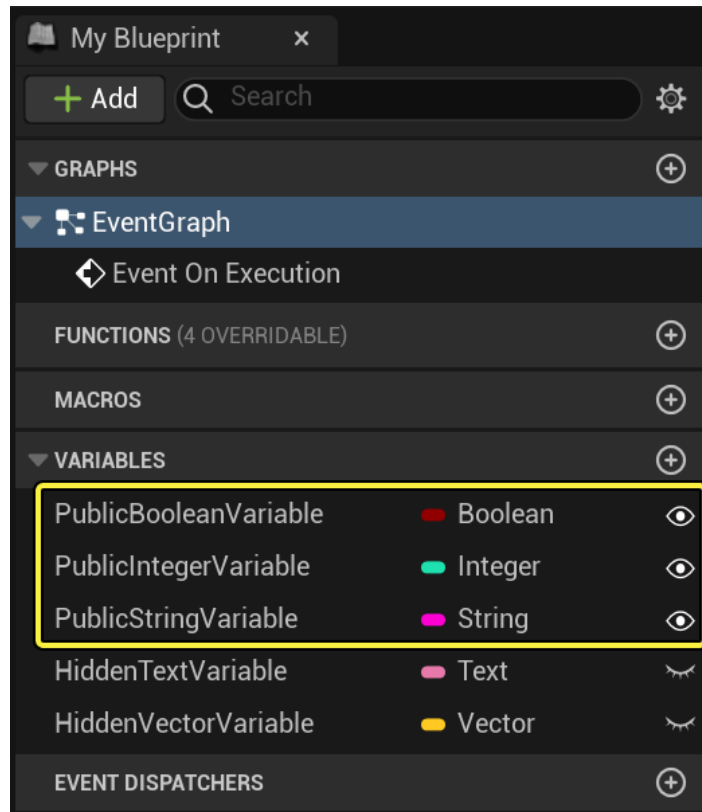
Use one of the following base classes as the parent class for your custom Dataprep block:

Class	Description
<code>DataprepBoolFetcher</code>	Use this class if you want to implement a filter that selects objects if it determines that a certain condition is true.
<code>DataprepFloatFetcher</code>	Use this class if you want to implement a filter that selects objects based on a floating-point numeric property: that is, a number that can have a decimal component.
<code>DataprepIntegerFetcher</code>	Use this class if you want to implement a filter that selects objects based on an integral numeric property: that is, a whole number with no decimal component.
<code>DataprepStringFetcher</code>	Use this class if you want to implement a filter that selects objects based on a string property.
<code>DataprepOperation</code>	Use this class if you want to implement an operator that makes modifications to Actors or Assets.
<code>DataprepEditingOperation</code>	Use this class if you want to implement an operator that makes modifications to Actors or Assets, and that may add or remove Actors and Assets.

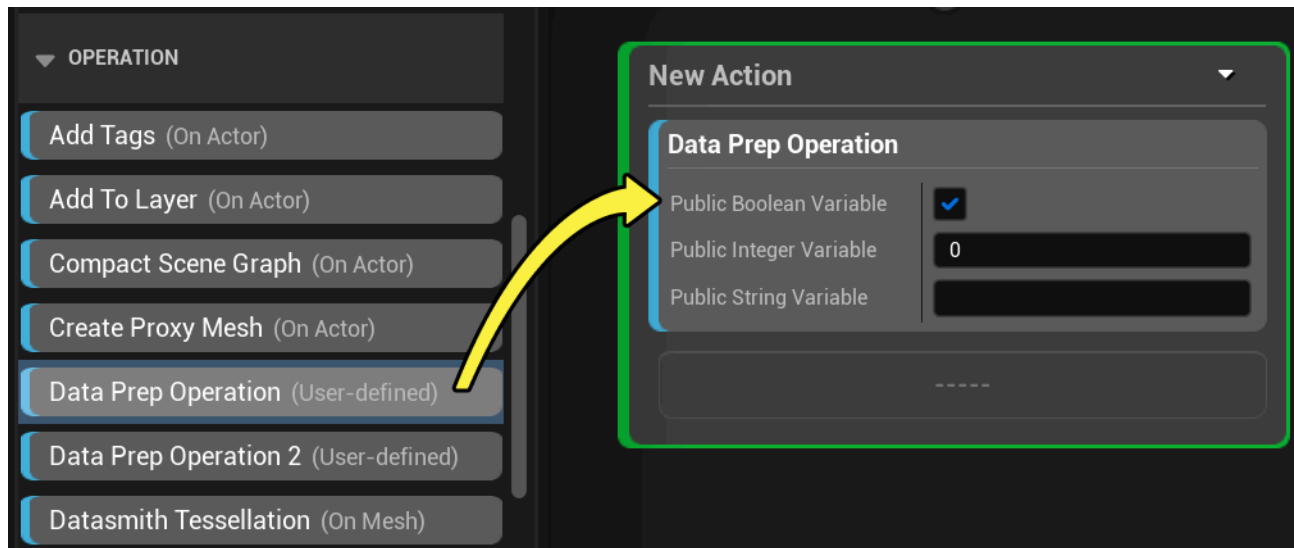
## Variables and Settings

Any variables that you make publicly visible in your Blueprint class will be exposed as customizable settings in the Dataprep editor for every block of this type. You can use this mechanism to offer Dataprep graph designers the ability to customize the way your function operates.

For example, this operator class contains some publicly visible variables, marked with the open eye icons:



When this type of block is used in a Dataprep graph, those publicly visible variables are exposed as customizable settings:



Note in the example above that the **HiddenTextVariable** and **HiddenVectorVariable** are not exposed as settings on the Dataprep block, because they are not marked as public in the Blueprint class. Note also that the settings in the block respect the default settings you set for the variables in the Blueprint class.

## Creating a Custom Filter

When you create a custom filter, you can override any or all of the following functions:

Function	Description
<div>Fetch</div>	<p>When you execute a Dataprep graph, and it treats a filter block of this type, it calls this function for each object in the current context.</p> <p>This function is expected to evaluate the object passed in to the Fetch function to produce an item of information of a specific type: a Boolean value, integer, float, or string. The filter node is responsible for conducting the logic to decide whether the item of information you return satisfies the logical condition</p>

Function	Description
	<p>expressed in the graph settings: for example, whether an integer you return is greater than or less than a value set by the user in the block, whether a string you return matches or is contained within the string set by the user in the block, and so on.</p> <p>This function should also return a Boolean value to indicate whether it was successfully able to calculate an output value for the current object. If you return a <code>false</code> value in this Boolean, the filter will exclude the current object.</p>
<code>Get Additional Keyword</code>	The Dataprep Editor calls this function to get a list of keywords for this block. If the user enters one of the keywords returned by your function into the search filter at the top of the palette, the editor will include this block in the list of matching items, even if the user's keyword does not occur in the name of the block.
<code>Get Display Fetcher Name</code>	The Dataprep Editor calls this function to get the display name that it should show in the palette for this block.
<code>Get Node Display Fetcher Name</code>	The Dataprep Editor calls this function to get the display name that it should show in the actual Dataprep graph for every block of this type.
<code>Get Tooltip Text</code>	The Dataprep Editor calls this function to get the text it should show in the tooltip when the user hovers the mouse over the name of this block in the palette, or over a block of this type in the Dataprep graph.






You must override the `Fetch` function in order for your filter to have any effect. The other functions are optional, but strongly recommended to control the way your block appears in the Dataprep Editor palette.

# Creating a Custom Operation

When you create a custom operation, you can override any or all of the following functions:

Function	Description
<code>Get Additional Keyword</code>	The Dataprep Editor calls this function to get a list of keywords for this block. If the user enters one of the keywords returned by your function into the search filter at the top of the palette, the editor will include this block in the list of matching items, even if the user's keyword does not occur in the name of the block.
<code>Get Category</code>	<p>The Dataprep Editor calls this function to determine what subcategory this block should be placed under in the palette, under the <b>Operations</b> list.</p> <div><p>If you don't override this function, your operation block will be listed under the <b>User-defined</b> category.</p></div>
<code>Get Display Operation Name</code>	The Dataprep Editor calls this function to get the display name that it should show in the palette for this block, and the display name that it should show in the actual Dataprep graph for every block of this type.

Function	Description
<code>Get Tooltip</code>	The Dataprep Editor calls this function to get the text it should show in the tooltip when the user hovers the mouse over the name of this block in the palette, or over a block of this type in the Dataprep graph.
<code>On Execution</code>	<p>When you execute a Dataprep graph, and it treats an operation block of this type, it calls this function to carry out the modifications on the imported Assets and Actors. By creating a graph of Blueprint nodes downstream from this event, you can control what your operation does to the list of Assets and Actors that are passed in to it by the previous block in the Action.</p> <p>See the following sections for details on how to implement your graph.</p>

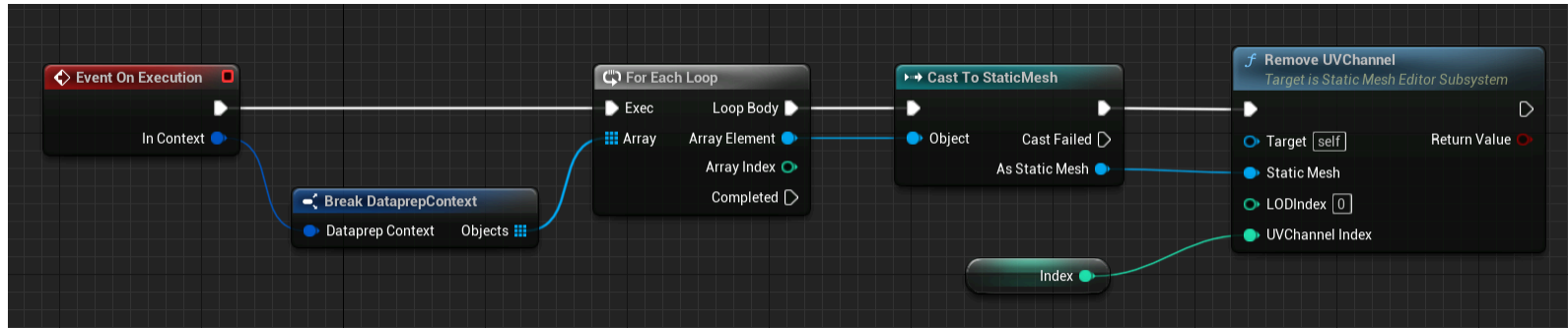


You must override the `Event On Execution` event in order for your operation to have any effect. The functions are all optional, but strongly recommended to control the way your block appears in the Dataprep Editor palette.

## The Dataprep Context

When the Dataprep system triggers the **On Execution** event for an operation block while executing a Dataprep graph, it passes in a *context* object for you to use in your Blueprint graph. This is essentially a list of all the Actors and Assets that your operator can modify. If your operation block is above any filter blocks in its Dataprep Action, this context object will contain all the Assets and Actors that you've imported into the Dataprep world. Otherwise, the context object will contain all the Actors and Assets that have been chosen by the filters above the operation block in the current Dataprep Action.

You can use the **Break DataprepContext** node to get an array of objects from the context. You can then modify all the objects in the context, or you can narrow down the objects to ones that match a specific type.



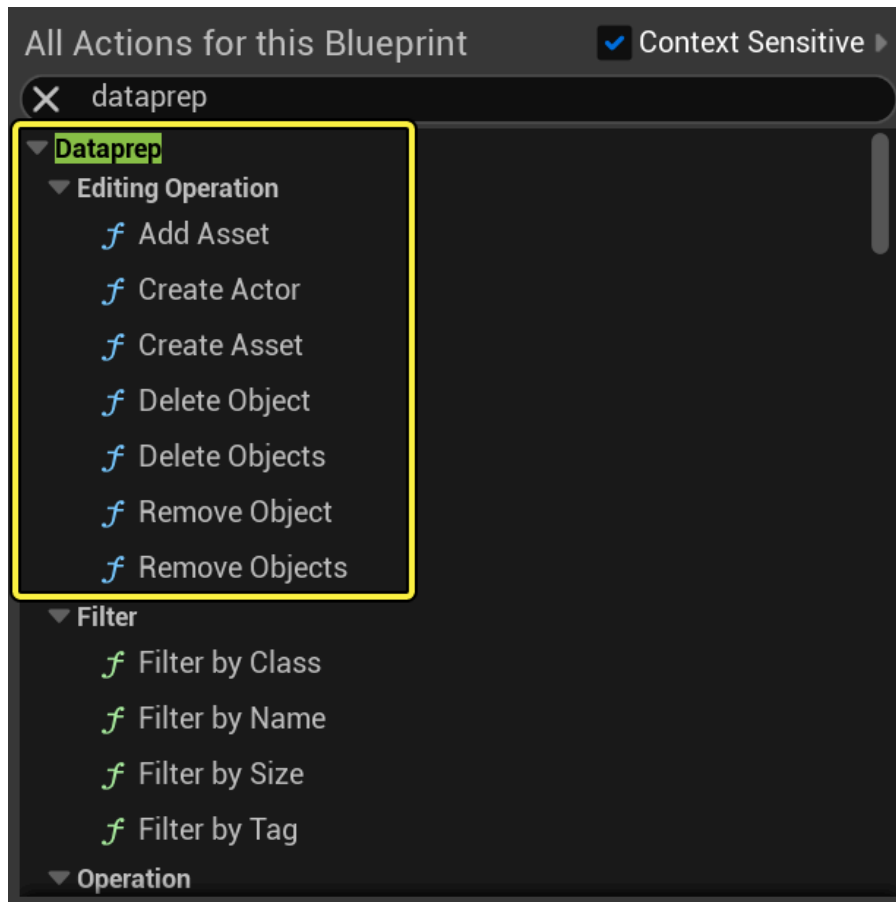
[Click for full image.](#)

For example, the operation above modifies a setting that is only present on Static Mesh Assets, but not on any other kinds of objects that might be contained in the Dataprep context (like Actors, Texture Assets, or Material Assets). So, it loops through the array of objects in the context, casting each to a Static Mesh. It then makes the modification only if the cast succeeds.

## Operations and Editing Operations

There are two parent classes for operator blocks: `DataprepOperation` and `DataprepEditingOperation`. Which one you should use depends on the editing operation you need to carry out. If you only need to modify objects that are already contained in the Dataprep context, you can use the `DataprepOperation` as your parent class. If you need to create new objects—for example, to create Assets in the Content Browser, to spawn new Actors in the Level, or to modify the list of list of objects in the Dataprep context—you must use `DataprepEditingOperation` as your base class.

When you derive your class from `DataprepEditingOperation`, you will have access to additional specialized functions in the Blueprint Editor, under the **Dataprep > Editing Operations** category:



For example, you can use **Add Asset** to add an Asset to the Dataprep context so that it will be included in the Dataprep context that is passed on to other blocks in the same Dataprep Action.



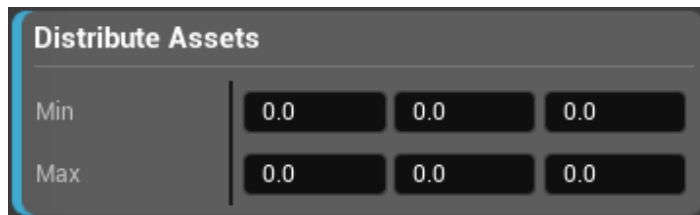
Always use the functions in the **Dataprep > Editing Operations** category to carry out actions whenever possible, instead of other nodes that may serve a similar purpose in normal circumstances. The temporary world that the Dataprep system uses to maintain all your imported data is unique; you can only modify it through these functions.

For example, always use the **Dataprep > Editing Operations > Create Actor** node to spawn a new Actor instead of the typical **Editor Scripting > Level Utility > Spawn Actor from Class** or **Spawn Actor from Object**.

# Operation Examples

## Distributing Static Mesh Actors

The following implementation of the `On Execution` event finds all Static Mesh Actors in the context, and redistributes them in space into a grid. The minima and maxima of the grid are taken from variables named **Min** and **Max**. These are marked as publicly editable, so they show up as a setting on the block in the Dataprep graph. However, the **Actors** variable, which is used to store the array of Static Mesh Actors in the scene, is not marked as editable. Therefore, it does not show up in the block on the Dataprep graph.

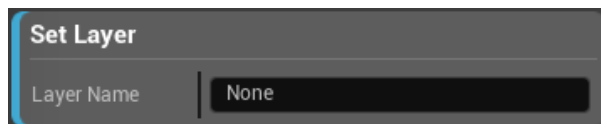


Distribute Assets			
Min	0.0	0.0	0.0
Max	0.0	0.0	0.0

 Copy code

## Adding Objects to a Layer

The following implementation of the `On Execution` event finds Actors in the context and adds them to a specified Layer in the Editor.



Set Layer	
Layer Name	None

 Copy code