

Developer

/ Documentation

/ Unreal Engine ▾

/ Unreal Engine 5.4 Documentation

/ Animating Characters and Objects

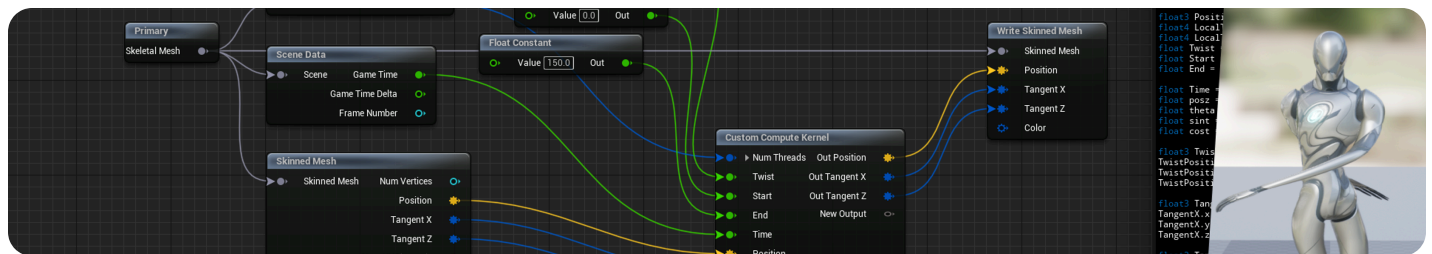
/ Skeletal Mesh Animation System

/ Animation Assets and Features

/ Deformer Graph

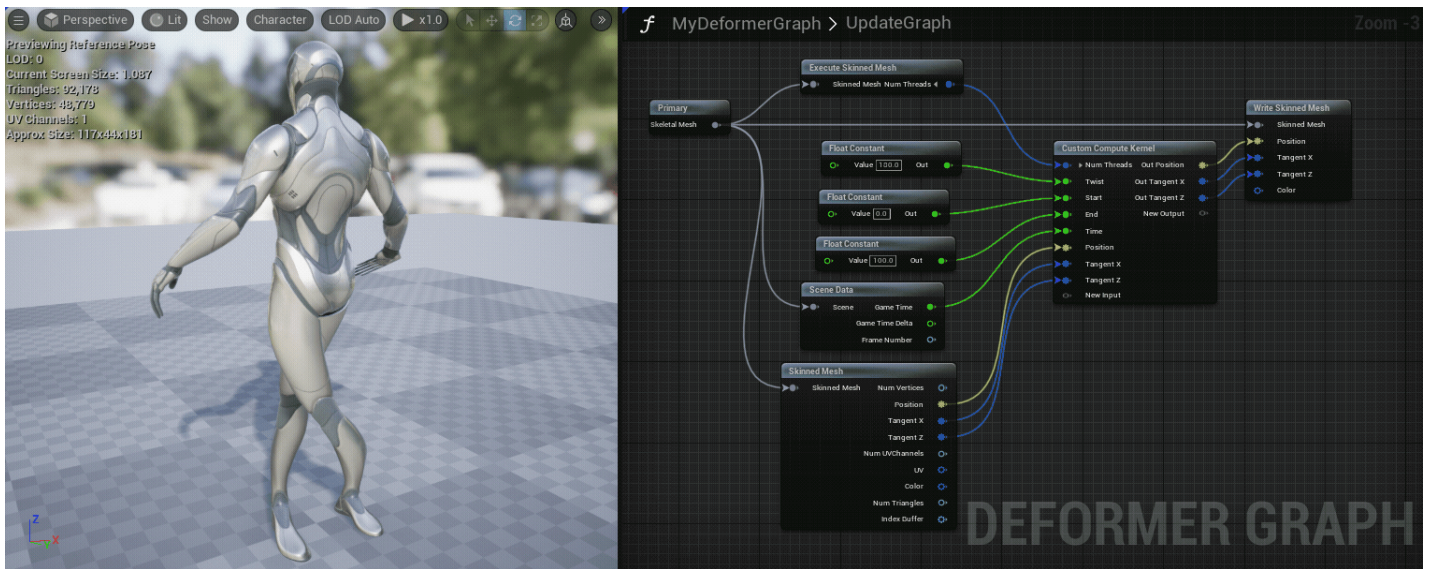
Deformer Graph

Use Deformer Graphs to create and edit custom mesh deformations for skinned characters and objects.



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

Using **Unreal Engine's Deformer Graph** plugin, you can create and edit Deformer Graph assets to perform and customize mesh deformations for any skinned mesh in Unreal Engine. Using Deformation Graphs, you can create and modify logic that adjusts the geometry of the mesh to fine tune mesh deformation behaviors, or create entirely new deformations within Unreal Engine. Most often Deformer Graphs are used to fine-tune realistic skin and fabric behaviors in motion for characters, or for one-off animations that are easier to create using deformation logic than hand-authored animations.



Mesh Deformers come in many forms, such as [Morph Targets](#), [Cloth simulation](#), and even generated models created in external DCC (Digital Content Creation) software such as the [Machine Learning \(ML\) Deformer](#).

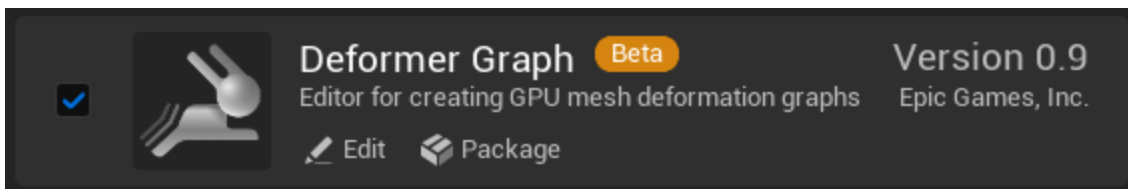
This document provides an overview of the Deformer Graph system in Unreal Engine.

How to Use the Deformer Graph

Here you can read about how to set up and begin using the Deformer Graph in Unreal Engine.

Prerequisites

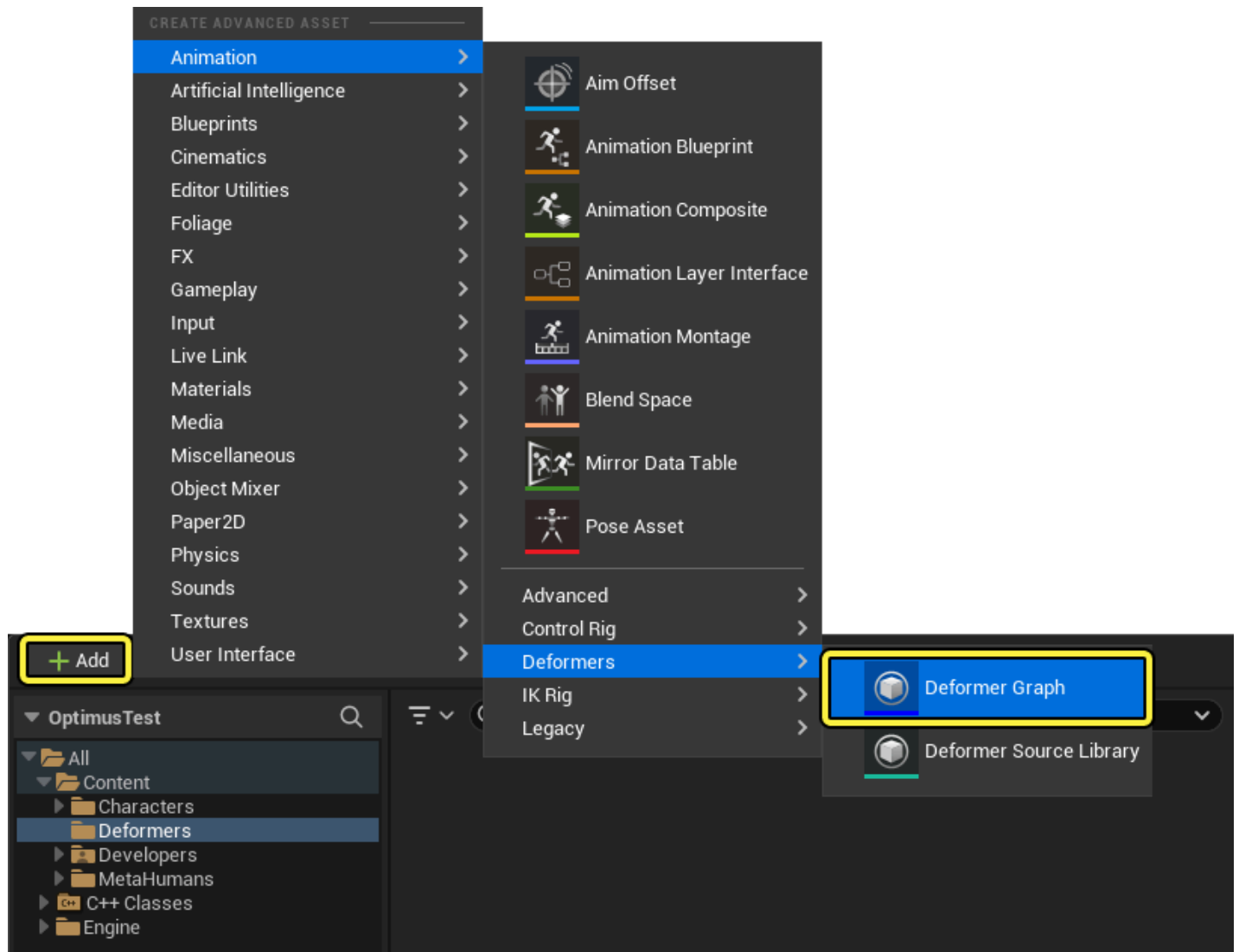
- Enable the **Deformer Graph** [plugin](#). Navigate in the **Menu Bar** to **Edit > Plugins** and locate the **Deformer Graph** in the **Animation** section, or using the **Search Bar**. Enable the plugin and restart the editor.



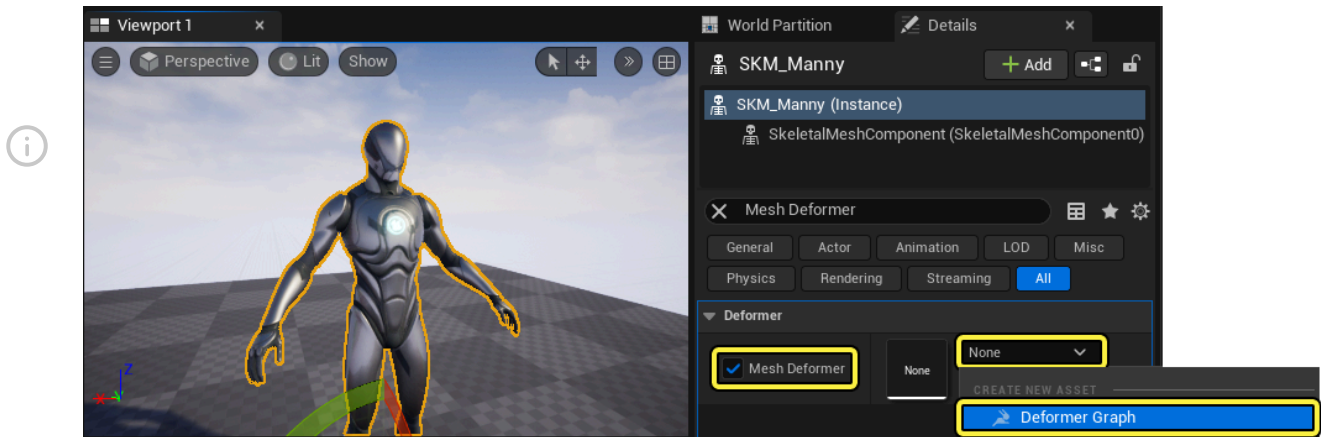
- Your project contains a skinned mesh, which could be a [Skeletal Mesh](#) character, or a [Static Mesh](#) object.

Creating a Deformer Graph Asset

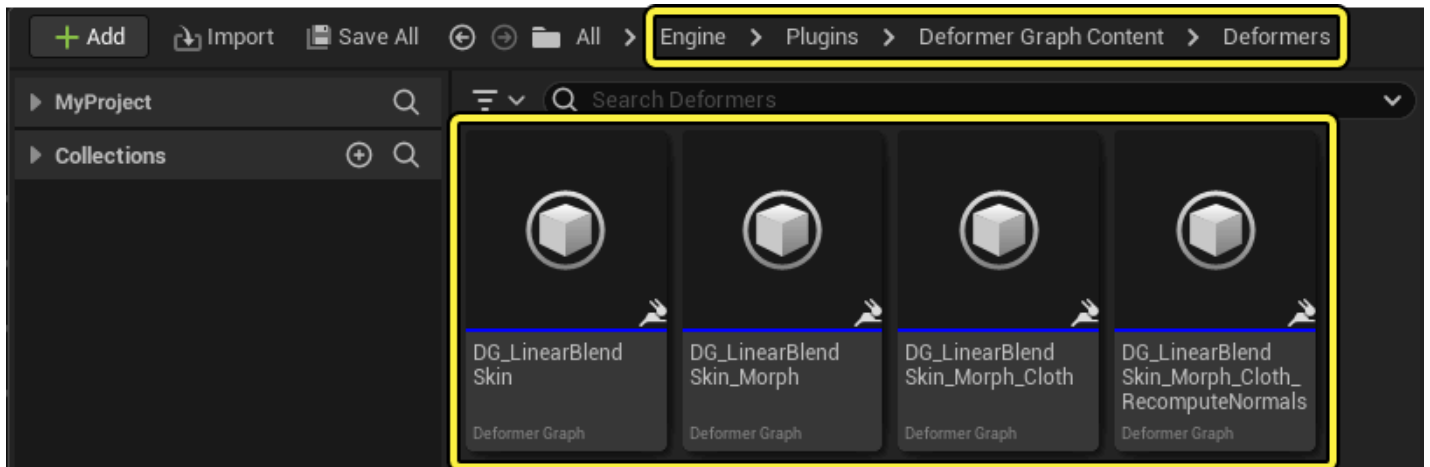
Create a **Deformer Graph** asset by **right-clicking** in the **Content Browser** and navigating to **Animation > Deformers > Deformer Graph**.



With a Mesh selected in the Viewport, you can also navigate in the **Details** panel to the **Mesh Deformer** property, and after **enabling** the Mesh Deformer property, you can create a new Deformer Graph by selecting the **Deformer Graph** option from the selection menu.



The following default Deformer Graph Assets are packaged alongside the Deformer Graph Plugin and are available for you to use or customize when creating custom mesh deformer logic for your characters and objects. These Deformer Graphs can be accessed in the Content Browser by navigating to **Engine > Plugins > Deformer Graph Content > Deformers**.

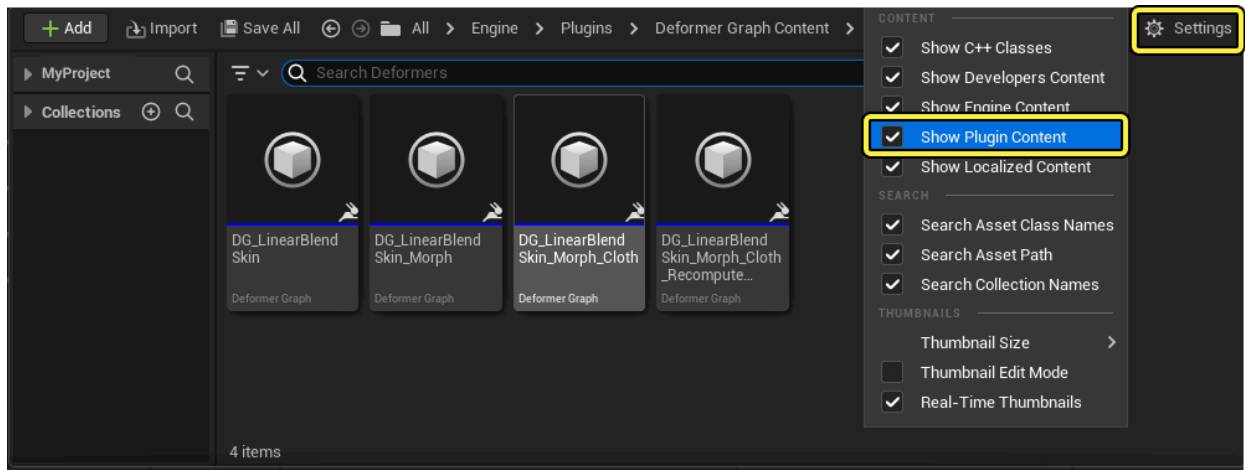


Asset	Description
DG_LinearBlendSkin	This graph applies the standard linear blend skinning pipeline. All skinning features are supported using this graph.

Asset	Description
DG_LinearBlendSkin_Morph	This graph applies the standard linear blend skinning and Morph Target pipelines. All features of those pipelines are supported using this graph.
DG_LinearBlendSkin_Morph_Cloth	This graph applies the standard linear blend skinning, Morph Target and Cloth simulation pipelines. All features of those pipelines are supported using this graph. This is a superset of the DG_LinearBlendSkin and DG_LinearBlendSkin_Morph Deformer Graph, and can be used instead of those without any performance penalty. The simpler Deformer Graph assets, DG_LinearBlendSkin and DG_LinearBlendSkin_Morph , exist individually as reference for those who want to build their own implementations.
DG_LinearBlendSkin_Morph_Cloth_RecomputeNormals	This graph applies the standard linear blend skinning, Morph Target , and Cloth simulation pipelines. It also applies additional passes to recompute vertex normals. This emulates the existing behavior of the GPU Skin Cache with Recompute Tangents settings.

If you do not see the **Engine** folder, or the default Deformer Graphs as available options when selecting a Deformer Graph in a mesh's Details panel, ensure you have **Plugin Content** enabled in the **View Settings** of the **Content Browser**.





For more information about creating and using Deformer Graph assets, refer to the [How to create a custom Deformer Graph](#) guide.

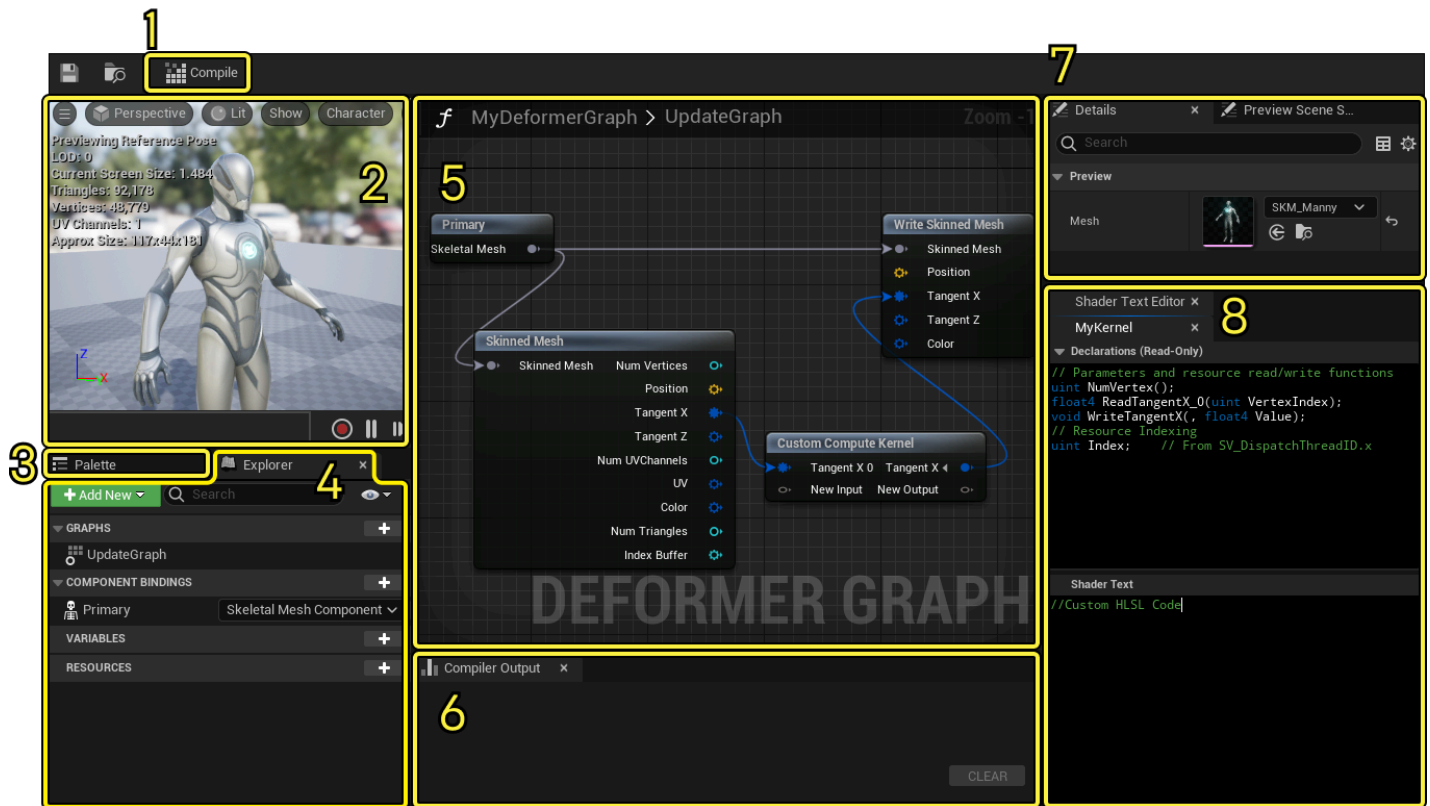
Opening The Deformer Graph

After enabling the **Deformer Graph Plugin**, you can open the **Deformer Graph Editor** by **double-clicking** any Deformer Graph Asset in the **Content Browser** or in a game objects **Details** panel, in the **Mesh Deformer** property.



Deformer Graph Interface

Here is a reference for the Deformer Graph interface you can use when creating and editing custom Deformer Graph assets, to deform meshes in Unreal Engine.



Interface	Description
1. Compile	Compiles the Deformer Graph.
2. Viewport	See the effect of the Deformer Graph logic on the mesh.
3. Palette	A toolbox of unique Deformer Graph Blueprint nodes.
4. Explorer	Add and manage Deformer Graph Subgraphs , Component Bindings , Variables , and resources .
5. Graph Editor	Similar to other Blueprint Editors , manage and edit Deformer Graph logic using a series of unique Deformer Graph nodes .
6. Compile Output	Review compiler logs and information, after successful and unsuccessful compiles of the Deformer Graph.
6. Details	Access component and node specific information and properties.

Interface	Description
7. Shader Text Editor	A text editor used to modify Custom Compute Kernel node programming using HLSL (High-Level Shader Language) .

Palette Panel

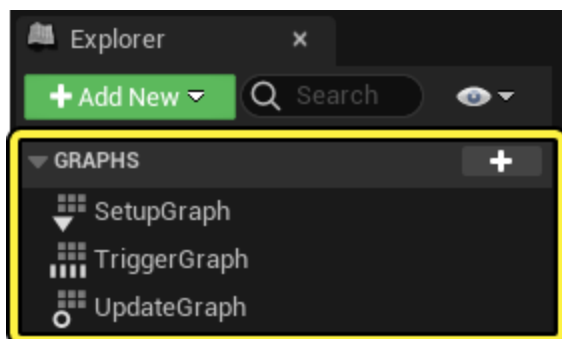
Using the Deformer Graph **Palette**, you can drag and drop custom Deformer Graph nodes and Kernels to reference or use when creating mesh deformation logic.



For a comprehensive list and description of the available components in the Palette panel, refer to the [Palette Component Reference](#) list at the bottom of this document.

Deformer Graph Subgraphs

There are 3 types of **Subgraphs** you can use within a Deformer Graph.

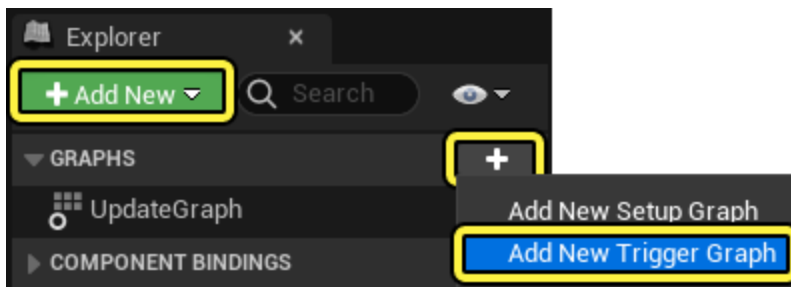


By default, a Deformer Graph only contains an **Update Graph** which is executed once per frame.

You can optionally add a **Setup Graph** that executes when the owning object is **initialized**.

Additionally, you can create and use **Trigger Graphs**, which can be executed by request from a Blueprint or C++ code. When Trigger Graphs are requested, they are queued and executed **after** an initial Setup Graph, if one is present, and **before** the Update Graph every frame.

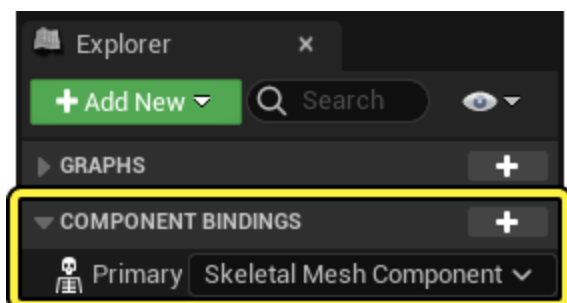
You can add a Subgraph to the Deformer Graph by clicking **Add (+)**, adjacent to the **Graphs** section of the Explorer panel, or by clicking the green **(+) Add New** button at the top of the Explorer panel. After clicking either, you can then choose to create either a **Setup Graph** or a **Trigger Graph** from the drop down menu.



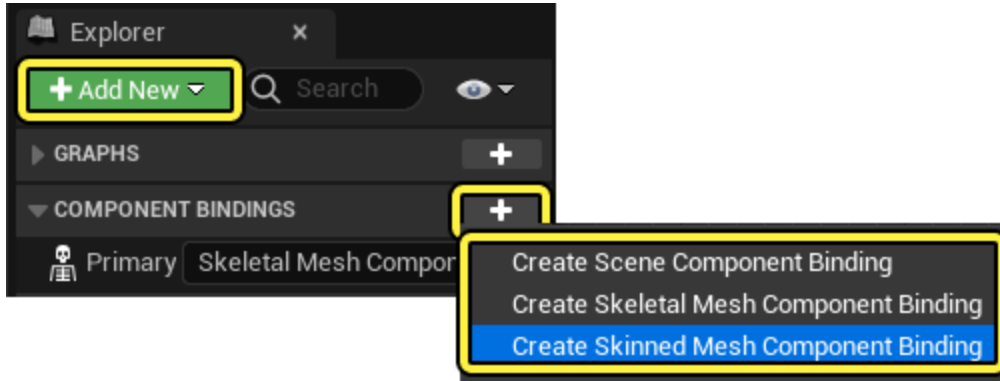
No additional Update Graphs, and only one Setup Graph, can be added to a Deformer Graph. You can add as many Trigger Graphs as your project needs. Trigger Graphs can then be renamed, by **double-clicking** the name of the desired Trigger Graph in the **Explorer** panel or by selecting the Trigger Graph and pressing the **F2** key.

Component Bindings

Component Bindings are references to available components contained in the owning object.



You can create additional component bindings to the Deformer Graph by clicking **Add (+)**, adjacent to the **Component Bindings** section of the **Explorer** panel, or by clicking the green **(+) Add New** button at the top of the **Explorer** panel.

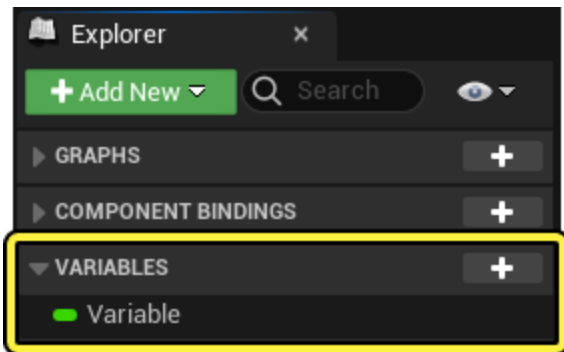


The following are some examples of Component Bindings that you can use when creating Deformer Graph logic:

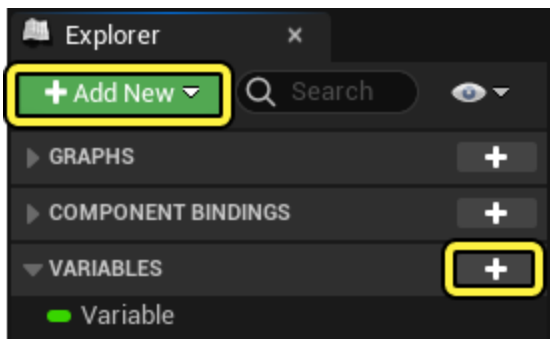
Component Binding	Descriptions
Skeletal Mesh Component	Add a reference to the owning object's Skeletal Mesh component. Using the Skeletal Mesh, many properties and components can be referenced, such as the Mesh Vertices and Triangles , and Scene Time .
Scene Component	<p>A Scene component draws data and information from the scene the character or subject the Deformer Graph is occupying. Game time and other useful information can be drawn from scene components.</p> <div><p>i</p><p>Scene information can also be gathered from a Skeletal Mesh or Skinned Mesh component binding, but there may be a context where using a separate Scene component binding is useful for control or organizational purposes.</p></div>
Skinned Mesh Component	A Skinned Mesh component can be representative of any kind of skinned mesh. This can be an auxiliary object attached to the character, or this can represent a character or object without bones.

Variables

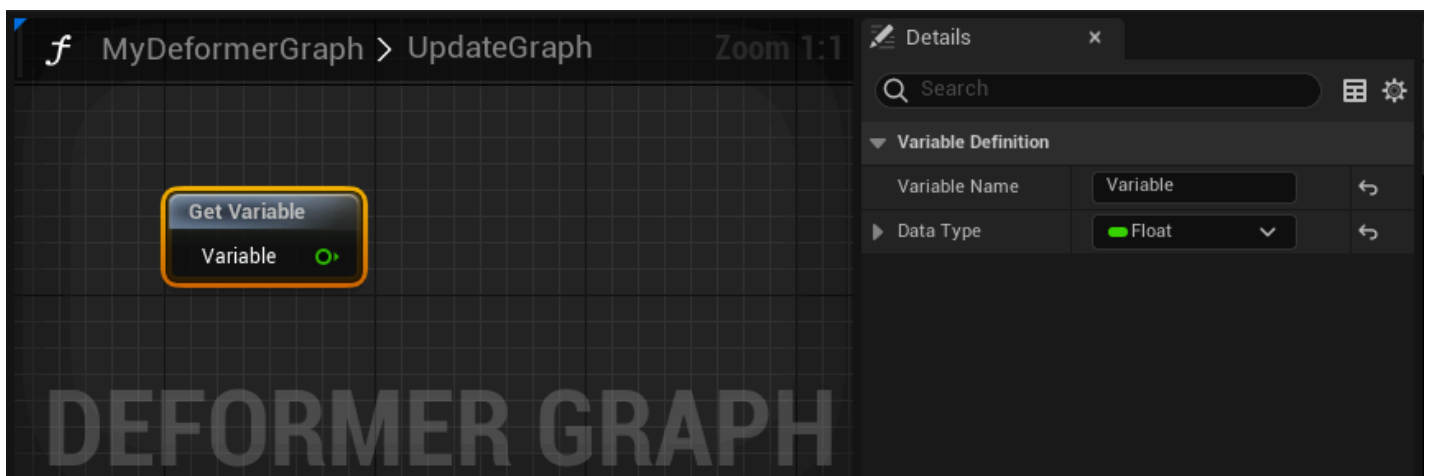
In the **Variables** section of the **Explorer Tab** you can create and manage externally visible variables you can reference to adjust and control logic in the Deformer Graph.



You can create a variable by clicking **Add (+)**, adjacent to the **Variables** section of the **Explorer** panel, or by clicking the green (+) **Add New** button at the top of the **Explorer** panel.



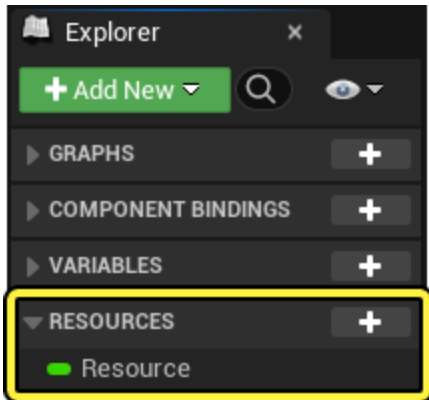
After creating a new variable, you can open its properties in the Details panel, by selecting the variable in the Explorer panel.



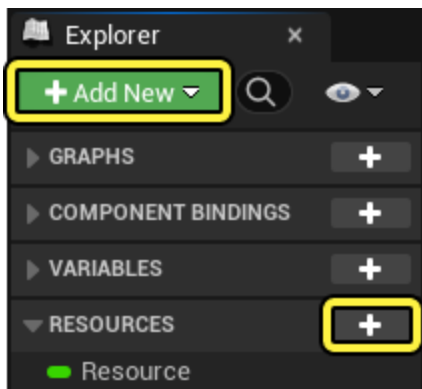
In the Details panel, you can name the variable using the **Variable Name** property, and define its **Data Type**. Variables can be assigned any data type, or component, for use with the other Deformer Graph nodes.

Resources

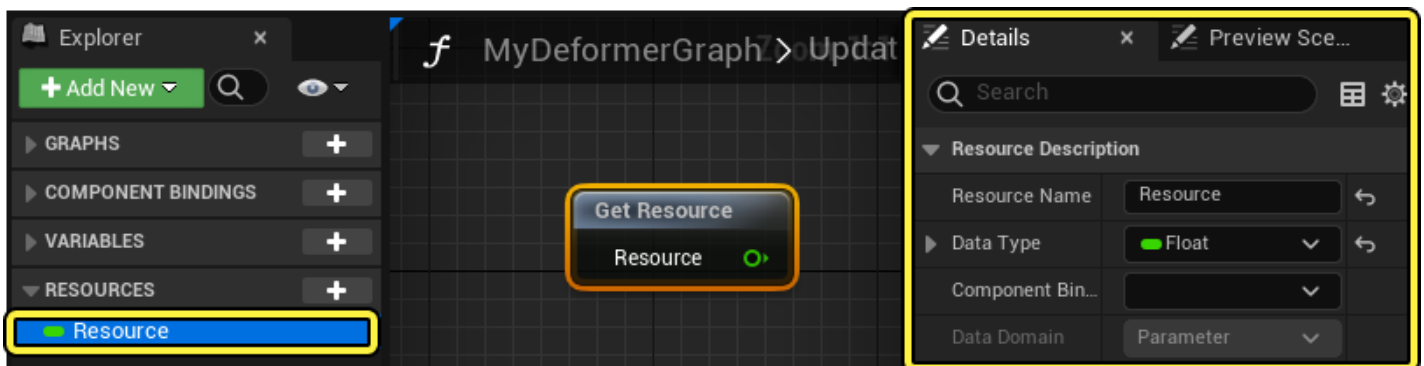
Similar to [Variables](#), you can create and manage **Shader Resources** in the **Resources** section of the Explorer panel. Resources are internal references to values and components that other nodes in the Deformer Graph can reference to perform mesh deformation functions.




You can create a resource by clicking **Add (+)**, adjacent to the **Resources** section of the **Explorer** panel, or by clicking the green **(+) Add New** button at the top of the **Explorer** panel.



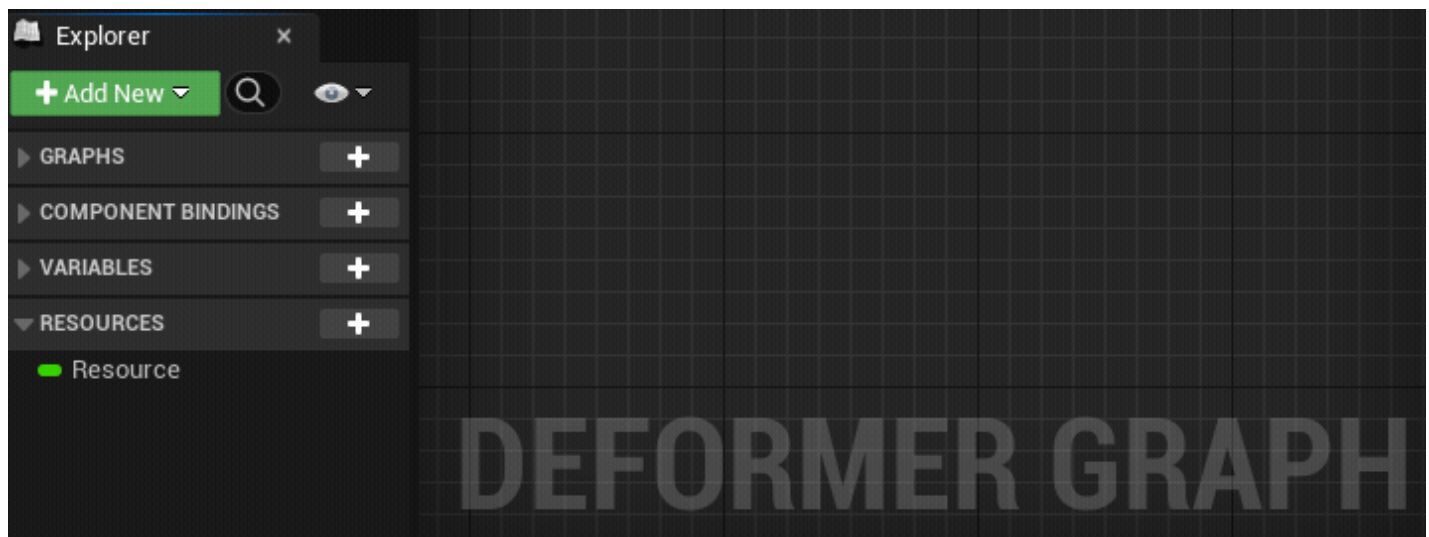
After creating a resource, you can select it in the Explorer tab to open its properties in the Details panel.



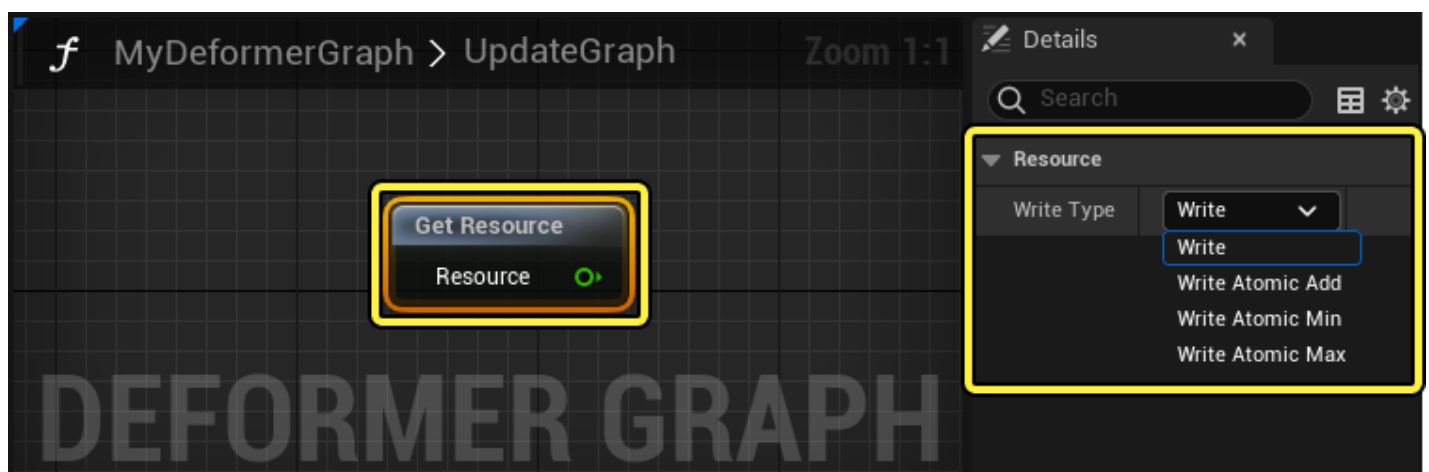
You can adjust the following resource parameters:

Property	Description
Resource Name	Name the resource in the provided field.
Data Type	Set the type of data the resource can store.
Component Binding	Set the component binding the resource is bound to.
Data Domain	<p>Set the data domain for the resource.</p> <p>In the first drop down menu you can select the dimension. You can select from the following dimensions:</p> <ul style="list-style-type: none"> • Triangle: Select all triangles. • Vertex: Select all vertices. • Vertex > Bone: Select groups of vertices by bone. • Vertex > Index0: Select groups of vertices within Index 0. • Vertex > UVChannel: Select groups of Vertices by UVChannel. • Expression...: Input a custom data type using the provided field. <p>When the dimension is set to Triangle or Vertex, you can additionally specify how many data values should be stored per entry. Beginning at x1, for one data point per entry, you can also select from x3, x4, or x8.</p> <div>  <p>Data points must match, not only in their dimension but also in the amount of data values present for each entry. If you are experiencing input/output pin incompatibility, or unresponsive resources in the Defromer Graph logic, ensure your Data Domain properties are set to compatible values.</p> </div>

Unlike variables, you can **set**, as well **reference** resource values and components within the Deformer Graph. After creating a resource, you can **drag-and-drop** the resource into the Deformer Graph Editor, and choose from a **Get** instance, used for referencing the existing value contained in the resource, a **Set** instance, used for setting or defining the resource for use later, or a **Get/Set** instance, that can be used to both set and reference the stored value.



After creating a **Get/Set Resource**, **Get Resource**, or **Set Resource node**, you can modify how the instance of the resource interacts with the **resource buffer**. You can define this behavior by selecting the node in the **Graph** and setting the **Write Type** property in the node's **Details** panel.

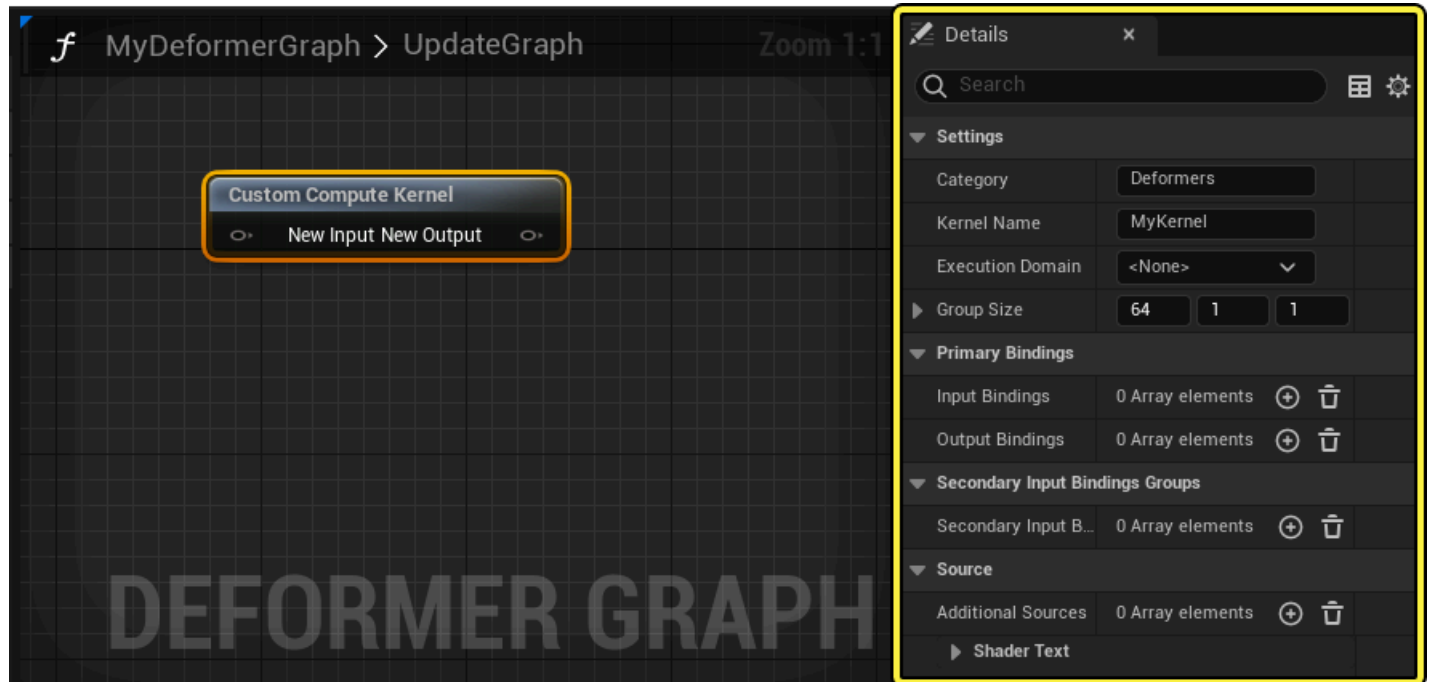


You can choose from the following write type property options:

Option	Description
Write	Writes the value to the resource buffer.
Write Atomic Add	Overwrites the graph value to the value in the resource buffer.
Write Atomic Min	Overwrites the graph Min value to the value in the resource buffer.
Write Atomic Max	Overwrites the graph Max value to the value in the resource buffer.

Details Panel

The **Deformer Graph Details** panel shows information and settings regarding each Deformer Graph node. To see a node's properties, select the node in the Graph Editor, and the Details Panel shows its properties.



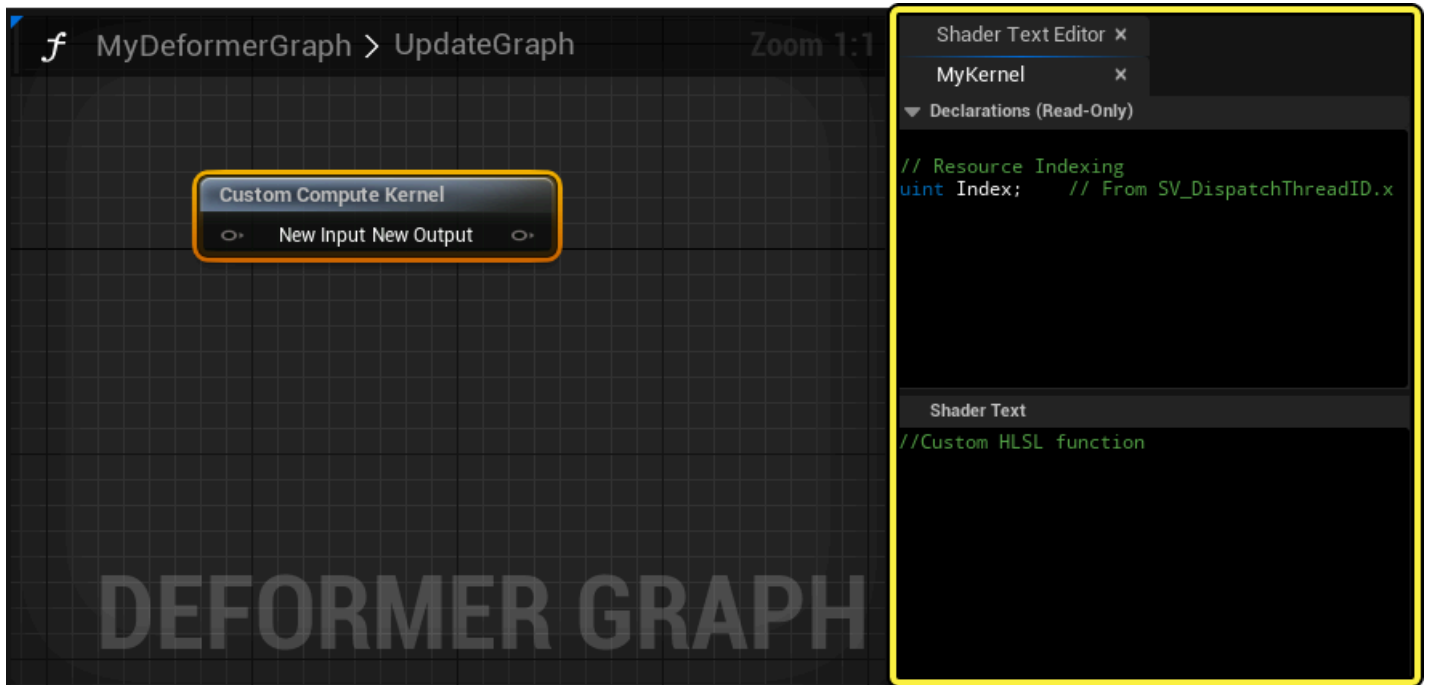
When selecting Kernel Nodes, **Input** and **Output** bindings can be added and modified.



To achieve functional results, Input and Output bindings must match the exact data type they are connected to. Please ensure any Kernel Input or Output binding is set to match the desired connected or extracted value.

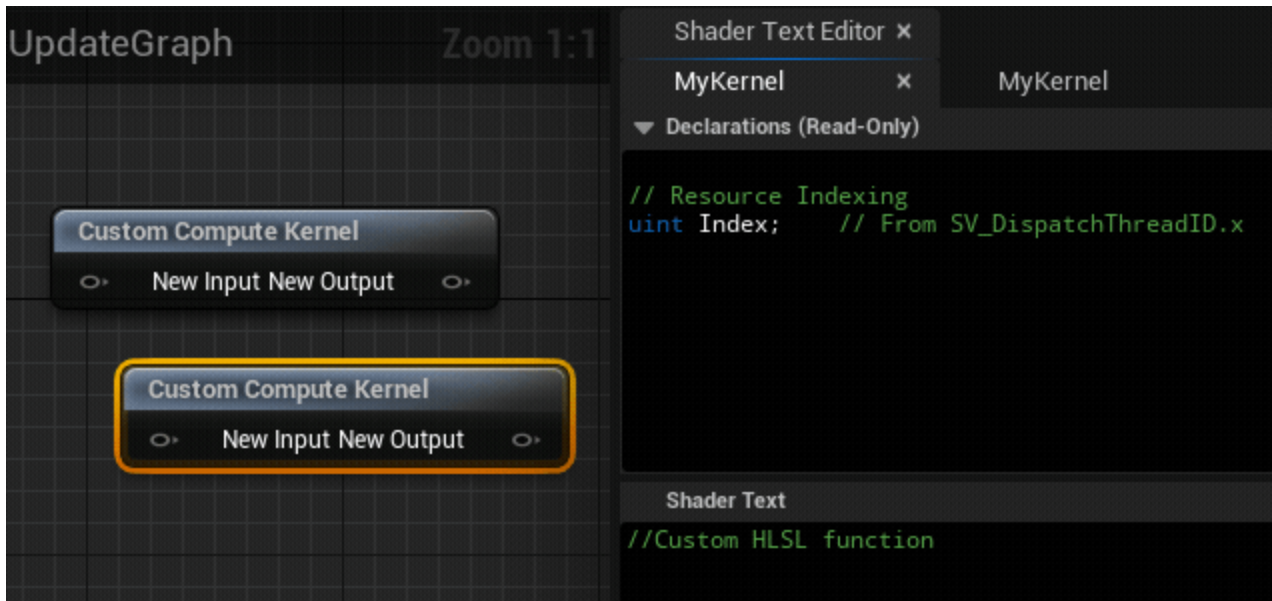
Shader Text Editor

Using the **Shader Text Editor**, you can program **Custom Compute Kernel** nodes to control specific mesh deformation behaviors.



The Shader Text Editor operates in **HLSL** (High-Level Shader Language) to program kernels.

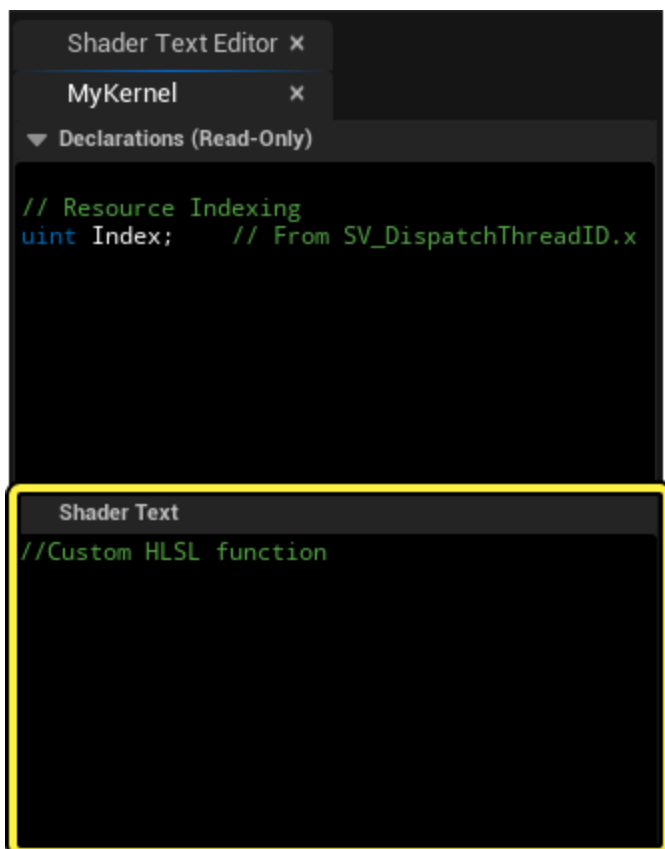
For each Custom Compute Kernel added to a Deformer Graph, a new tab opens within the Shader Text Editor panel.



Within each Custom Compute Kernel tab, you can access the node's **Declarations**, which are read-only, and represent the Graph Editor modification made to the node, such as added Input and Output bindings.



You can also access the node's **Shader Text** where you can write custom HLSL code to create mesh deformations.

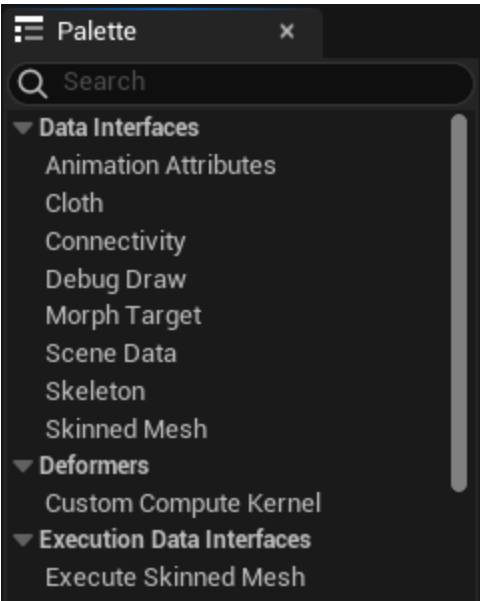


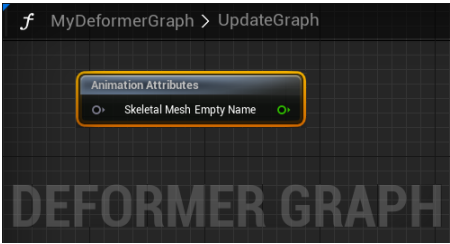
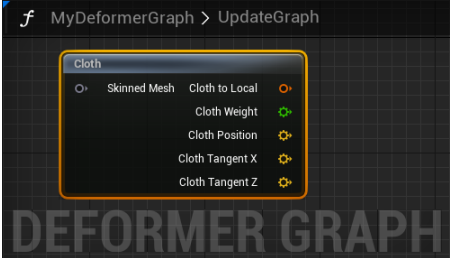
For more information about the HLSL coding standard, refer to the [Microsoft High Level Shader Language reference documentation and programming guide](#).

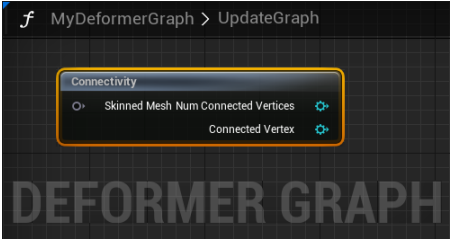
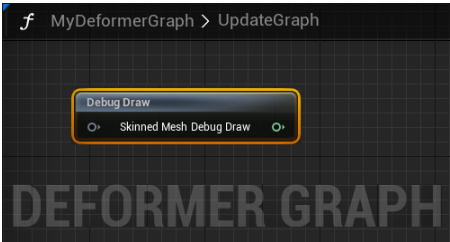
For an example workflow of using the Shader Text Editor, refer to the [How to Create a Deformer Graph](#) guide.

Palette Component Reference

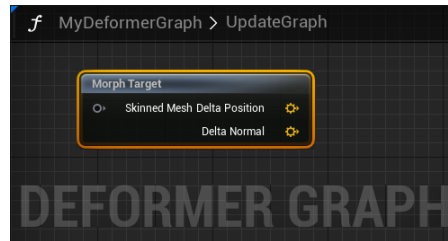
Here you can reference a list of the available components found in the Deformer Graph **Palette** and a description of their function.



Palette Component	Node	Description
Animation Attributes		Add references to Animation Attributes that you can use to trigger or activate Deformer logic.
Cloth		Add and attach references to Cloth simulation models associated with the skeletal mesh. Refer to Cloth Simulation Models for more information.

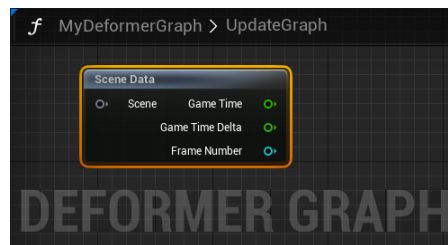
Palette Component	Node	Description
Connectivity		<p>Connect Skinned Mesh components to an existing vertex or vertice data set.</p>
Debug Draw		<p>Uses a connected Skinned Mesh component binding to draw a debug view of the mesh in the Viewport. Debug data is extracted as a string and is accessible using the nodes output pin in the Graph.</p> <p>Options in the Details panel include:</p> <ul style="list-style-type: none"> • Force Enable: Forces the debug draw to occur in the Viewport. • Max Line count: Set the maximum amount of lines between vertices the debug drawing can display. • Max triangle Count: Set the maximum amount of geometry triangles the debug drawing can display. • Max Character Count: Set the maximum number of characters the debug drawing can display. • Font Size: Set the font size for the debug drawing overlay.

Morph Target



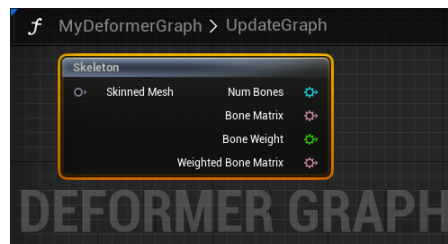
Using a Skinned Mesh component, the [Morph Target](#) node can isolate the **Delta Position** and Delta Normal as Vector 3** values.

Scene Data



Extract scene data you can use to control or trigger Deformer Graph logic. A **Scene component binding** can be created, but you can also use a **Skeletal Mesh component binding** to reference the current scene the skeletal mesh occupies at runtime.

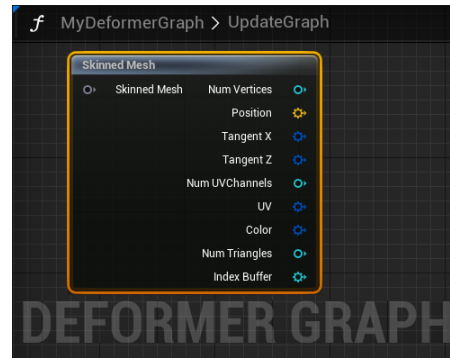
Skeleton



Extract Skeleton data from a Skeletal Mesh component binding. The available outputs include:

- **Num Bones:** Output value is a **UInt** and represents the number of bones in the skinned mesh's skeleton.
- **Bone Matrix:** Output value is a 3×4 matrix of bone positions.
- **Bone Weight:** Output value is the **alpha** weight of bones, in the form of a float.
- **Weighted Bone Matrix:** Output value is a 3×4 matrix of weighted bone positions.

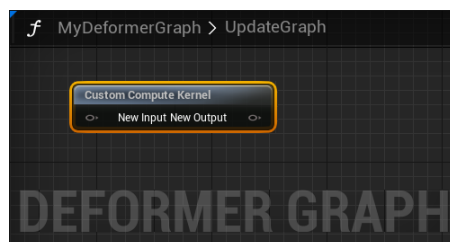
Skinned Mesh



Extract mesh data from any **Skinned Mesh** component binding. The available outputs include:

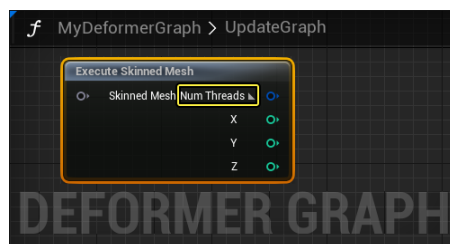
- **Num Vertices:** Output the number of vertices contained in the mesh as an integer.
- **Position:** Output the position of the Vertices in a **Vector 3**.
- **Tangent X:** Output the **tangent** on the **X** axis as a **Vector 2**.
- **Tangent Z:** Output the **tangent** on the **Z** axis as a **Vector 2**.
- **Num UV Channels:** Output the number of UVChannels the mesh possesses as an integer.
- **UV:** Output the mesh vertices by a set UVChannel as a Vector 2 value.
- **Color:** Output the **R, G, B,** and **W** values of the meshes color as a **Vector 4**.
- **Num Triangles:** Output the number of triangles that comprise the mesh's geometry as an integer.
- **Index Buffer:** Output and isolate the mesh's index buffer coordinates from its mesh vertice coordinates, as an array.

Custom Compute Kernel



Create a **Custom Compute Kernel** that can be programmed using **HLSL (High-Level Shader Language)**, to perform custom mesh deformations. For more information on creating and using Custom Compute Kernels, refer to the [How to Create a Custom Deformer Graph](#) documentation.

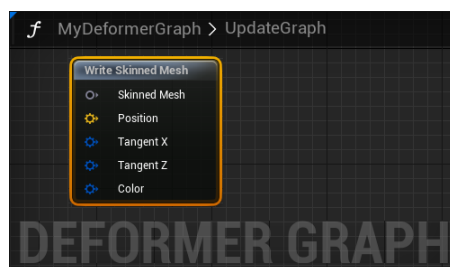
Execute Skinned Mesh



Using a **Skinned Mesh** component as an input, you can execute mesh data threads for use in a **Kernel**. In the Details panel you can set the **Domain** of the function to either run the Kernel with one thread per **vertex**, or one thread per **triangle**.

The **Execute Skinned Mesh** node outputs an **IntVector 3** value that can be connected and used with a **Custom Compute Kernel**. To isolate the output data on only the **X**, **Y**, or **Z** axis, expand the **output** pin using the **triangle** next to the output pin. Each axis has a separate **Integer** output pin.

Write Skinned Mesh

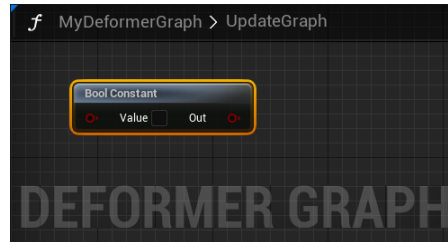


The **Write Skinned Mesh** output node is the final operation of a Deformer Graph, writing any modifications performed to the mesh's **vertices or triangles****.

The **Skinned Mesh** input pin connects to the initial mesh **component binding**. Then, using the Vector 3 **Position** input pin,

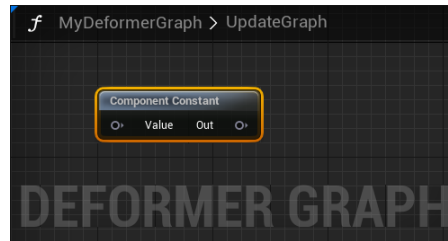
and the Vector 4 **Tangent X**, **Tangent Z**, and **Color** input pins, the new mesh properties can be written to the mesh.

Bool Constant



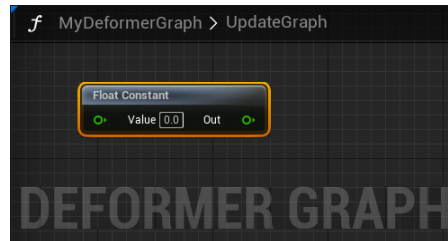
Set a constant **boolean** value. The node can accept a bool variable value, or you can manually toggle the boolean in the graph or in the node's **Details** panel using the **Value** property.

Component Constant



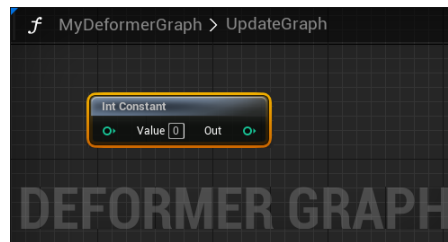
Set a **component constant**, using a component binding. A component constant can be set in the graph, or manually defined using the **Value** property in the node's **Details** panel.

Float Constant



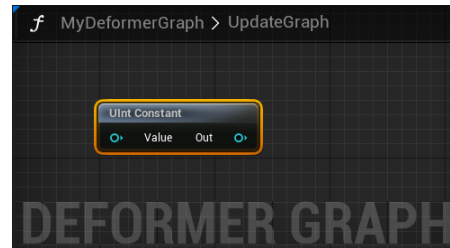
Set a constant **float** value. The node can accept a **float** value from another component, or you can manually set a value in the graph or in the node's **Details** panel.

Int Constant



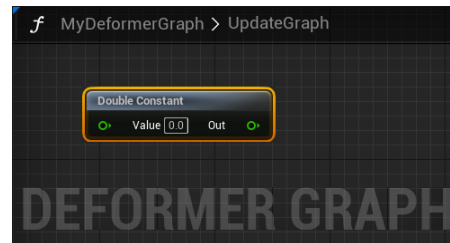
Set a constant **signed integer** value. The node can accept **zero** and **positive** integer values from another component, or you can manually set a value in the graph or in the node's **Details** panel.

UInt Constant



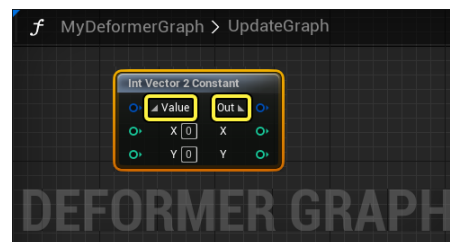
Set a constant **unsigned integer** value. The node can accept **any** integer value from another component, or you can manually set a value in the graph or in the node's **Details** panel.

Double Constant



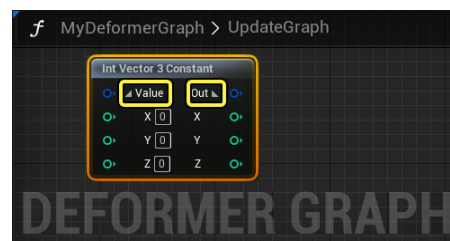
Set a constant **double** value. The node can accept any double value from another component, or you can manually set a value in the graph or in the node's **Details** panel.

Int Vector 2 Constant

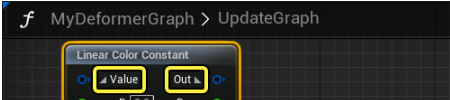


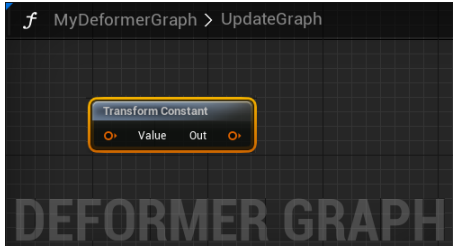
Set a constant **Vector 2** value on the **X** and **Y** axes. The node can accept any vector 2 value from another component, or you can manually set **X** and **Y** integer values in the graph by expanding the input pin using the triangle or in the node's **Details** panel. Using the triangle near the output pin you can also split the vector 2 value into Integer values for the **X** and **Y**.

Int Vector 3 Constant



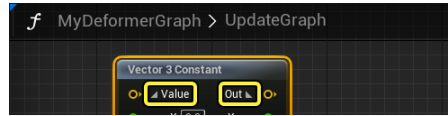
Set a constant **Vector 3** value on the **X**, **Y**, and **Z** axes. The node can accept any Vector 3 value from another component, or you can manually set **X**, **Y**, and **Z** integer values in the graph by expanding the input pin using the triangle or in the node's **Details** panel. Using the triangle near the output pin you can also split the

		Vector 3 value into Integer values for the X , Y , and Z values.
Int Vector 4 Constant		Set a constant Vector 4 value on the X , Y , Z , and W axes. The node can accept any Vector 4 value from another component, or you can manually set X , Y , Z , and W integer values in the graph by expanding the input pin using the triangle or in the node's Details panel. Using the triangle near the output pin you can also split the Vector 4 value into Integer values for the X , Y , Z , and W values.
Linear Color Constant		Set a constant Color value for the R , G , B , and A values stored as a float 4 value. The node can accept any float 4 or Color value from another component, or you can manually set R , G , B , and A float values in the graph by expanding the input pin using the triangle or in the node's Details panel. Using the triangle near the output pin you can also split the Color value into float values for the individual R , G , B , and A values.
Quat Constant		Set a constant Quat or float 4 value on the X , Y , Z , and W axes. The node can accept any Quat or float 4 value from another component, or you can manually set X , Y , Z , and W float values in the graph by expanding the input pin using the triangle or in the

Palette Component	Node	Description
		node's Details panel. Using the triangle near the output pin you can also split the Quat value into separate float values for the individual X , Y , Z , and W values..
Rotator Constant		Set a constant Rotator or float 3 value on the X (Pitch) , Y (Yaw) , and Z (Roll) axes. The node can accept any Rotator or float 3 value from another component, or you can manually set Pitch , Yaw , and Roll values in the graph by expanding the input pin using the triangle or in the node's Details panel. Using the triangle near the output pin you can also split the Rotator value into float values for the Pitch , Yaw , and Roll .
Transform Constant		Set a constant Transform value. The node can accept any transform value from another component, or you can manually set the Location, Rotation, and Scale values on the X , Y , and Z axes in the node's Details panel.
Vector 2 Constant		Set a constant Vector 2 value on the X , Y , and Z axes. The node can accept any Vector 2 value from another component, or you can manually set X , Y , and Z float values in the graph by expanding the input pin using the triangle or in the node's Details panel. Using the triangle near the output pin you can also split the Vector 2

value into float values for the **X**, **Y**, and **Z** axes.

Vector 3 Constant



Set a constant **Vector 3** value on the **X**, **Y**, and **Z** axes. The node can accept any Vector 3 value from another component, or you can manually set **X**, **Y**, and **Z** float values in the graph by expanding the input pin using the triangle or in the node's **Details** panel. Using the triangle near the output pin you can also split the Vector 3 value into float values for the **X**, **Y**, and **Z** axes.

Vector 4 Constant

Set a constant **Vector 4** value on the **X**, **Y**, **Z**, and **W** axes. The node can accept any Vector 4 value from another component, or you can manually set **X**, **Y**, **Z**, and **W** float values in the graph by expanding the input pin using the triangle or in the node's Details panel. Using the triangle near the output pin you can also split the Vector 4 value into float values for the **X**, **Y**, **Z**, and **W** axes.