Unity Composition Manual

PiRho Soft

Animation Player

Animation Player is a MonoBehaviour that utilizes Unity's Playables API to play simple AnimationClips on a GameObject. This is best used on objects that do not need the complexity and overhead of full fledged AnimatorControllers.

See the *"FinishDoor"* object in the *"Maze2"* scene and the *"MazeJewel"* Instruction Graph in the Maze project for an example usage.

Audio Player

Audio Player is a MonoBehaviour that utilizes Unity's Playables API to play simple AudioClips on a GameObject.

See the *"Maze1"* scene and the *"MazeKey"* Instruction Graph in the Maze project for an example usage.

Axis Input

Axis Input is a MonoBehaviour that will set a value on an input axis when an object is clicked. This can be used in conjunction with the InputHelper to create virtual axes for things like on screen buttons in a mobile game that move a character directionally or to trigger actions.

See the *"MazeUI"* scene in the Maze project for an example usage.

> Because this uses Unity's IPointerHandler interface the object must have either a Graphic or a Collider and the Canvas or Camera must have a GraphicRaycaster or PhysicsRaycaster respectively.

Fields
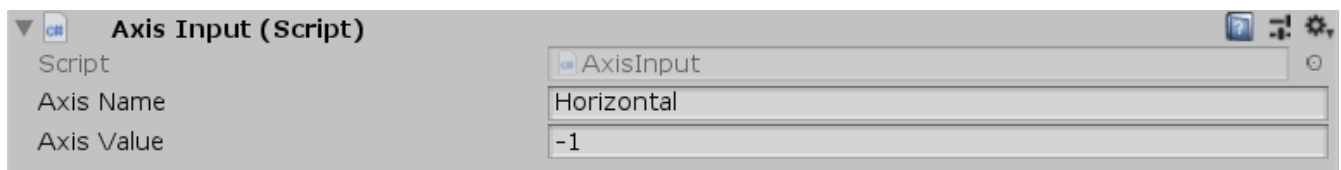
Name	Description
Axis Name	The name of the input axis to set when clicked
Axis Value	The value to set the input axis to when clicked

Bar Binding

Bar Binding is a Variable Binding that will set the fill amount and blend color of a sibling Image based on bindings to two int or float variables. If *AmountVariable* or *TotalVariable* is invalid or does not return an int or float, then the image will be disabled. If speed is greater than 0.0f then changes will be animated to between values. Custom color gradients can also be set depending on the fill amount. Bar Bindings are useful for things like health and progress bars.

See Variable Bindings for more information.
See the *"Battle"* scene in the Battle project for an example usage.

> ℹ️ Make sure the *Image Type* property on Image is set to *Filled* for the Bar Binding to work properly.

Fields
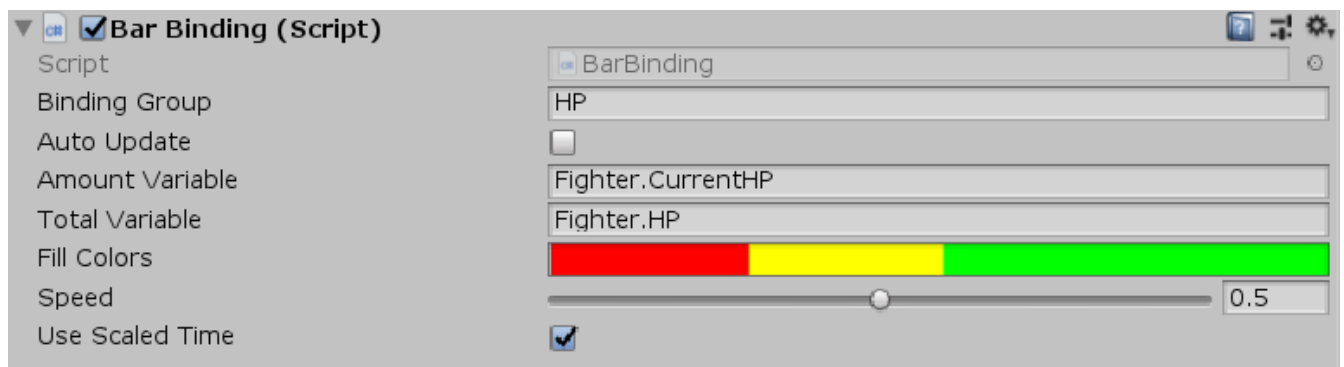
Name	Description
Amount Variable	The VariableReference that represents the fractional amount of the fill
Total Variable	The VariableReference that represents the total amount of the fill
Fill Colors	The Gradient to base the blend color on
Speed	The speed at which to animate changes (in percentage of the total per second)
Use Scaled Time	Whether to base the speed off of scaled time or real time

Binding Root

Binding Root is a MonoBehaviour that acts as a root object for all child Variable Bindings to bind data on. Binding Roots have a *Value* property that child bindings access via the *Value Name* property. *Value* can be set from a derived class such as Object Binding Root or through a Set Binding Node. When accessing variables on a Binding Root, the search will cascade upward to the next Binding Root in the hierarchy with the the default variable store on the Composition Manager as the base. Selection Controls and Menus use Binding Roots on each of their child Menu Items to group each item's data.

See Binding Roots for more information on binding roots.
See the "Card" prefab in the CardGame project for an example usage.

Fields
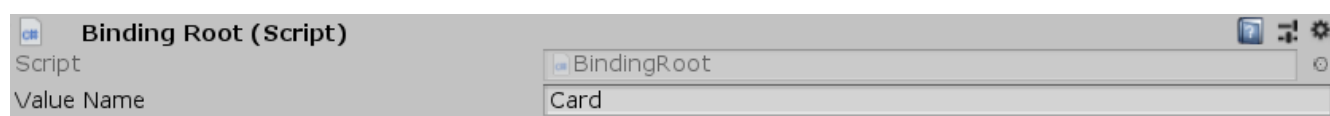
Name	Description
Value Name	The name of the variable that child Variable Bindings can use in order to access this Binding Root's *Value*.

Branch Node

A Branch Node is an Instruction Graph Node that will branch to any number of different nodes based on the the string evaluation of an Expression. This can be thought of similar to a switch statement in C#. Create a Branch Node in the **Create › Control Flow › Branch** menu of the Instruction Graph Window.

See Control Flow for more information.
See the node named *"Branch on Action"* on the *"Battle"* Instruction Graph in the Battle project for an example usage.

Fields
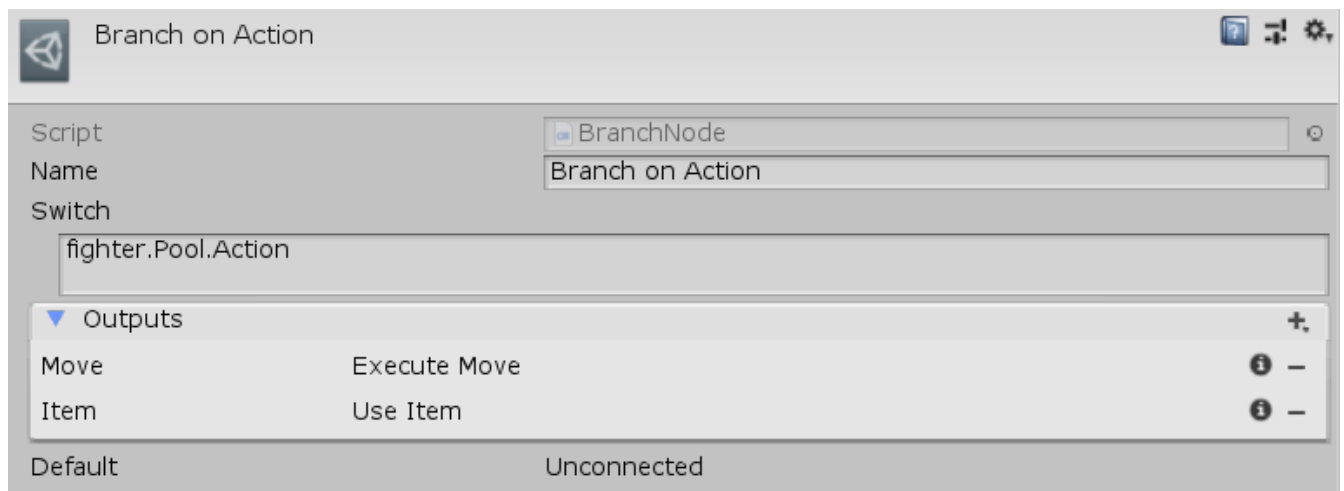
Name	Description
Switch	The Expression to run to determine the node to branch to
Outputs	The dictionary of names to nodes to branch to based on *Switch*
Default	The default node to go to if the value of *Switch* is not in the *Outputs* dictionary

Break Node

A Break Node is an Instruction Graph Node that will force the most recent ILoopNode to stop running. ILoopNode is implemented by Loop Node and Iterate Node. Create a Break Node in the **Create › Control Flow › Break** menu of the Instruction Graph Window.

See Control Flow for more information.
See the *"BoardLoop"* Instruction Graph in the BoardGame project for an example usage.

Button Graph Trigger

Button Graph Trigger is an Instruction Trigger that will run its Instruction Graph when the the specified *Button* is pressed.

See Graphs for more information on instruction graphs.
See the *"LootLevel"* scene in the Loot project for an example usage.

Fields
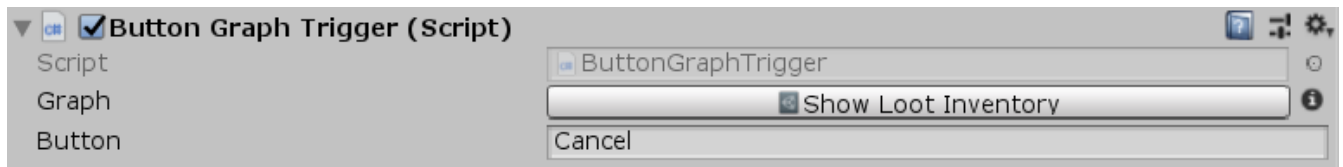
Name	Description
Button	The name of the button that will trigger the Instruction Graph

Button Input

Button Input is a MonoBehaviour that will set an input button to be pressed the object is clicked on. This is useful for things like on screen buttons in a mobile game to trigger actions.

See the *"Calculator"* scene in the Calculator project for an example usage.

> ℹ️ Because this uses Unity's IPointerHandler interface the object must have either a Graphic or a Collider and the Canvas or Camera must have a GraphicRaycaster or PhysicsRaycaster respectively.

Fields

Button Input (Script)		
Script	ButtonInput	
Button Name	Equals	

Name	Description
Button Name	The name of the button to be pressed when the object is clicked

Click Graph Trigger

Click Graph Trigger is an Instruction Trigger that will run its Instruction Graph when it is clicked on.

See Graphs for more information on instruction graphs.
See the *"Card"* prefab in the CardGame project for an example usage.

> **ℹ** Because this uses Unity's IPointerHandler interface the object must have either a Graphic or a Collider and the Canvas or Camera must have a GraphicRaycaster or PhysicsRaycaster respectively.

Collision Graph Trigger

Collision Graph Trigger is an Instruction Trigger that will run its Instruction Graph on collisions. Collision Graph Trigger works in conjunction with a Collision Notifier which will call the respective Enter and Exit functions on the Collision Graph Trigger.

See Graphs for more information on instruction graphs.
See the *"Maze1"* scene in the Maze project for an example usage.

Fields
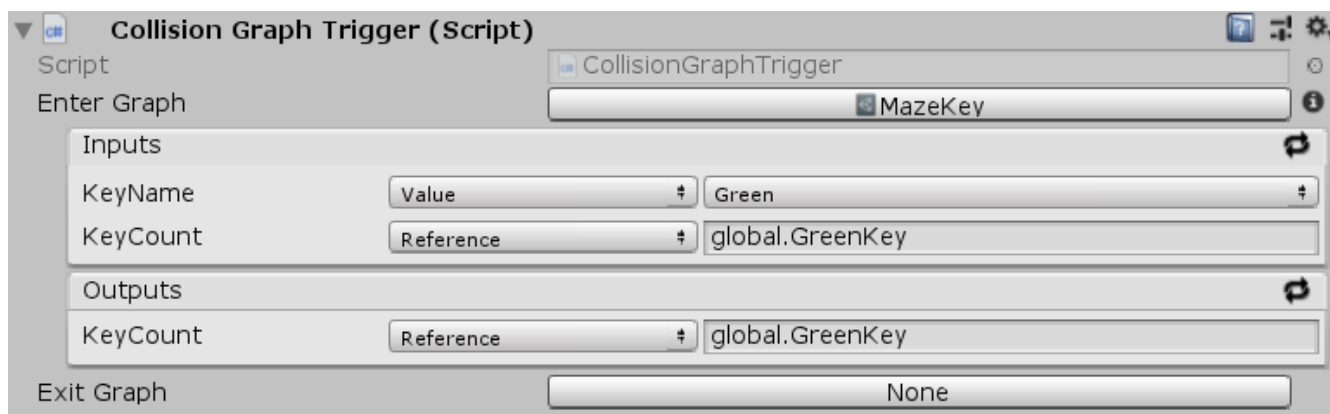
Name	Description
Enter Graph	The Instruction Graph to run when this object is collided with.
Exit Graph	The Instruction Graph to run when this object is exited.

Collision Notifier

A Collision Notifier is a MonoBehaviour used to trigger events during collisions. It works in conjunction with an ICollisionTrigger which can be implemented to receive an `Enter()` and an `Exit()` call when this object starts and stops colliding with the trigger. The Collision Graph Trigger is an example implementation of an ICollisionTrigger that can interact with a Collision Notifier.

See the *"Player" script and the "Maze1"* scene in the Maze project for an example usage.

Command

A Command is a Asset that can be created to give access to custom Expression logic that can be reused and called from other Expressions. This is useful when you have more complex logic that will be calculated in multiple locations. Commands can unify that logic so that it doesn't have to be written multiple times and changing it once changes it for all instances that use it. Create a Command through the **Create › PiRho Soft › Command** menu in the project view.

> ❗ Because a Command is a "Resource" it must be placed in a folder called "Commands" as a direct subfolder of a "Resources" folder so that it can be loaded and accessed on demand.

See Commands for more information.
See the "Apply Damage" Expression Node of the "BattleScratch" Instruction Graph in the Battle project for an example usage.

Fields

Name	Description
Name	The string name of the Command that is used in an Expression to call this command
Parameters	The list of Variables that should be passed to this Command - these can be accessed by name in *Expression*
Expression	The Expression that this command will run when it is called

Comment Node

A Comment Node is an Instruction Graph Node that is used for debugging purposes. It is inert during runtime and is purely used to display custom info about how a graph is functioning. Create a Comment Node in the **Create › Debug › Comment** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

Fields

Name	Description
Comment	The string value of the comment

Composition Manager

The Composition Manager is a MonoBehaviour that manages the entire composition system. Because Composition Manager is a GlobalBehaviour, it is created automatically the first time it is accessed so it does not need to be added to any objects in a scene. Composition Manager maintains a "global" IVariableStore that can be accessed from any VariableReference or Expression using the "global" accessor. It also maintains the "scene" IVariableStore, which gives access to GameObjects in the scene by name using the "scene" accessor. Instruction Graphs can be manually run from script through the Composition Manager using the Run() method.

Conditional Node

A Conditional Node is an Instruction Graph Node that branches the progression of an Instruction Graph based on the `true` or `false` evaluation of the Expression, *Condition*. Create a Conditional Node in the **Create › Control Flow › Conditional** menu of the Instruction Graph Window.

See Control Flow for more information.
See the *"HasKey"* node in "MazeDoor" Instruction Graph in the Maze project for an example usage.

Fields

Name	Description
Condition	The Expression to evaluate to determine the progression of the graph

Create Game Object Node

A Create Game Object Node is an Instruction Graph Node that will spawn a Prefab at the given Name, Position, and Rotation relative to the world, another object, or as a child object. The created object can optionally be stored in a given VariableReference so that it can be accessed later. Create a Create Game Object Node in the **Create › Object Manipulation › Create Game Object** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the "Create Highlight" node on the "BoardTakeTurn" Instruction Graph in the BoardGame project for an example usage.

Fields

Name	Description
Prefab	The prefab to create
Object Name	The name of the new object.
Object Variable	The VariableReference to store the created object in
Positioning	The ObjectPositioning to create the object at
Object	If *Positioning* is Relative, the object to position the created object relative to
Parent	If *Positioning* is Child, the object to make the parent of the created object
Position	The position of the new object - can be a Vector3 value or a VariableReference
Rotation	The rotation of the new object - can be a Vector3 value or a VariableReference, stored as euler angles

Create Scriptable Object Node

A Create Scriptable Object Node is an Instruction Graph Node that will create a new instnace of a ScriptableObject and store it in the given VariableReference. Create a Create Scriptable Object Node in the **Create › Object Manipulation › Create Scriptable Object** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

Fields
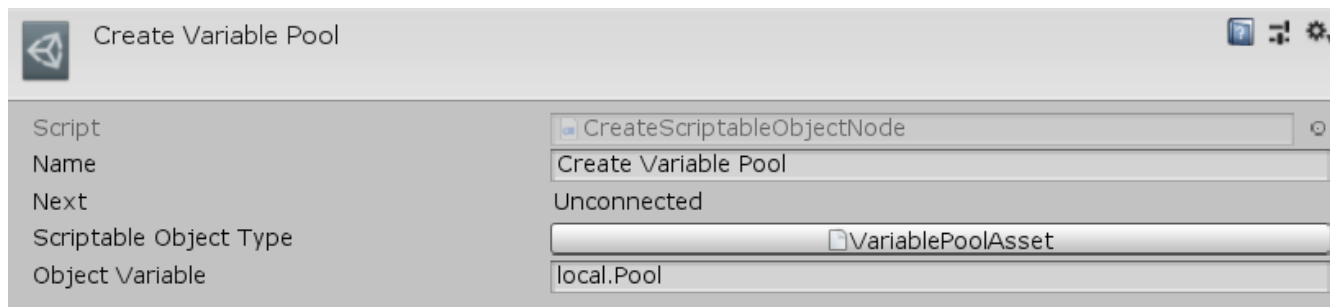
Name	Description
Scriptable Object Type	The type of the ScriptableObject to create
Object Variable	The VariableReference to store the created object in

Cutoff Transition

Cutoff Transition is an abstract implementation of a Transition that provides a custom Shader with an interface to fade, distort, and dissolve the screen image over time using an input texture. Each RGB color component of the input texture is sampled by the Shader and used to determine the output. The R and G channels determine the direction of the distortion original image (distortion) as the x and y offset respectively. Values less than 128 will be negative offset and values greater than 128 will be positve offset. The B component of the image determines the dissolve of the of the image with higher values being cutoff later in the transition. The Color property determines the color to fade to as time goes on or when the B component is cutoff.

See the Fade Transition or Dissolve Transition classes for example implementations of a Cutoff transition.

Destroy Object Node

A Destroy Object Node is an Instruction Graph Node that will destroy any Object that has been previously created, either by a Create Game Object Node, a Create Scriptable Object Node, or that is loaded in a scene. Create a Destroy Object Node in the **Create › Object Manipulation › Destroy Object** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the "Destroy Highlight" node on the "BoardTakeTurn" Instruction Graph in the BoardGame project for an example usage.

Fields

Name	Description
Target	The VariableReference to the object to be destroyed

Disable Object Node

A Disable Object Node is an Instruction Graph Node that will disable the given Object, *Target*. Create a Disable Objects Node in the **Create › Object Manipulation › Disable Object** menu of the Instruction Graph Window.

> ℹ️ *Target* must be a GameObject, Behaviour, or Renderer as these are the only Object, types that Unity allows to be enabled and disabled.

See Graphs for more information on instruction graphs.
See the "Disable Player" node on the "MazeStart" Instruction Graph in the Maze project for an example usage.

Fields

Disable Player	🔲 ⇥ ⚙
Script	◼ DisableObjectNode ⊙
Name	Disable Player
Next	Unconnected
Target	player

Name	Description
Target	A VariableReference to the Object to disable

Dissolve Transition

Dissolve is an implementation of a Cutoff Transition Transition that will fade the screen based on an input Texture over the duration of the Transition. The blue component of the texture will be read as the value of the cutoff. Higher values will take longer to dissolve - values of 0 will be *Color* right from the beginning of the Transition, and values of 255 will not dissolve to *Color* until the end of the Transition. If *Texture* is null, a Texture will automatically be generated with perlin noise to create a smooth looking transition. Create a Dissolve through the **Create › PiRho Soft › Transitions › Dissolve** menu in the project view.

Fields
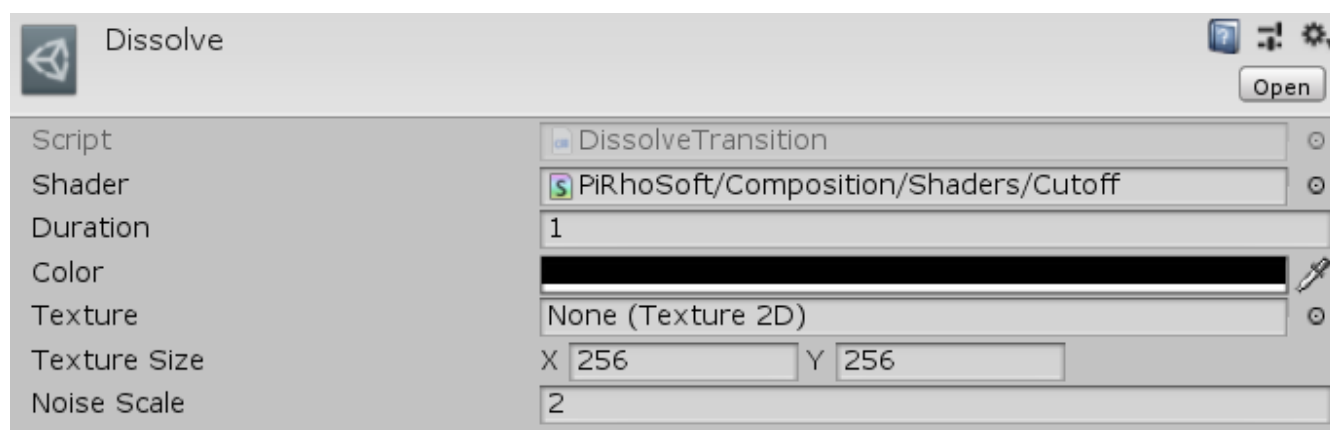
Name	Description
Color	The Color of the screen to dissolve to
Texture	The input Texture that gives the pattern of the dissolve
Texture Size	If *Texture* is null, the size of the Texture to generate
Noise Scale	If *Texture* is null, the scale value of the perlin noise generated as the input Texture

Enable Binding

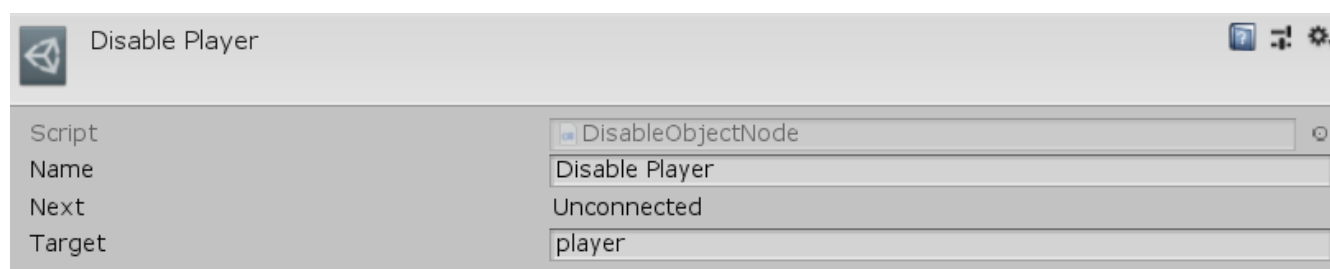Enable Binding is a Variable Binding that will enable or disable an Object based on the evaluation of an Expression. *Object* must be a GameObject, Behaviour, or Renderer as these are the only Object, types that Unity allows to be enabled and disabled. If *Condition* is invalid or does not return a bool, then the object will be disabled.

> If *Object* is set to this binding or the GameObject that has this binding, then the *AutoUpdate* flag will not function as this binding will not receive UpdateBinding() calls from the Composition Manager

See Variable Bindings for more information.
See the *"MazeUi"* scene in the Maze project for an example usage.

Fields
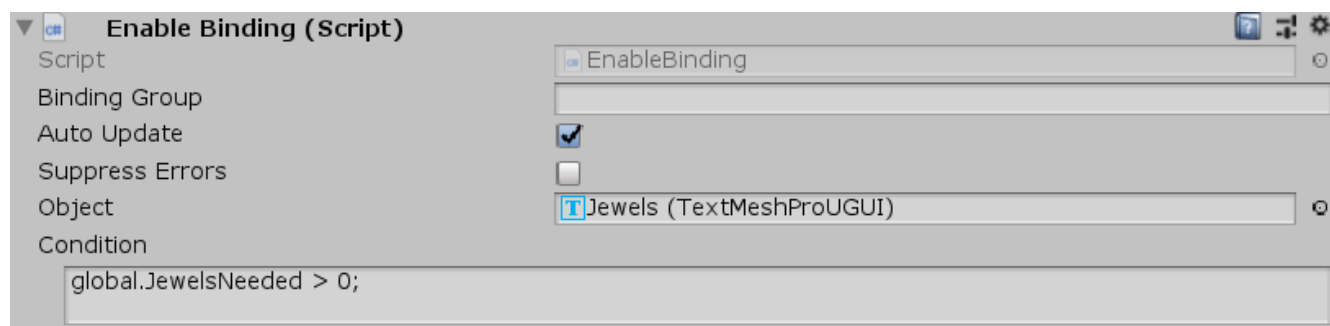
Name	Description
Object	The GameObject, Behaviour, or Renderer to enable or disable based on *Condition*
Condition	The Expression that determines whether *Object* will be enabled or disabled

Enable Graph Trigger

Enable Graph Trigger is an Instruction Trigger that will run its Instruction Graph when this object becomes enabled (in its `OnEnable()` message).

See Graphs for more information on instruction graphs.
See the "Gate" objects in the *"Maze3"* scene of the Maze project for an example usage.

Enable Object Node

An Enable Object Node is an Instruction Graph Node that will enable the given Object, *Target*. Create an Enable Object Node in the **Create › Object Manipulation › Disable Object** menu of the Instruction Graph Window.

> ℹ️ *Target* must be a GameObject, Behaviour, or Renderer as these are the only Object, types that Unity allows to be enabled and disabled.

See Graphs for more information on instruction graphs.

See the "Enable Player" node on the "MazeStart" Instruction Graph in the Maze project for an example usage.

Fields

Name	Description
Target	A VariableReference to the Object to enable

Exit Node

An Exit Node is an Instruction Graph Node that will force the currently running branch of the graph to exit and stop running. Create an Exit Node in the **Create › Control Flow › Exit** menu of the Instruction Graph Window.

See Control Flow for more information.
See the *"Battle"* Instruction Graph in the Battle project for an example usage.

Expression Binding

Expression Binding is a String Binding that will run an Expression and bind its output as the displayed string. If *Expression* evaluates to a float or an int, then customized Formatting can be applied. If *Expression* is invalid, then the text component will be disabled.

See Variable Bindings for more information.
See Expressions for more information.
See the *"MazeUi"* scene in the Maze project for an example usage.

Fields
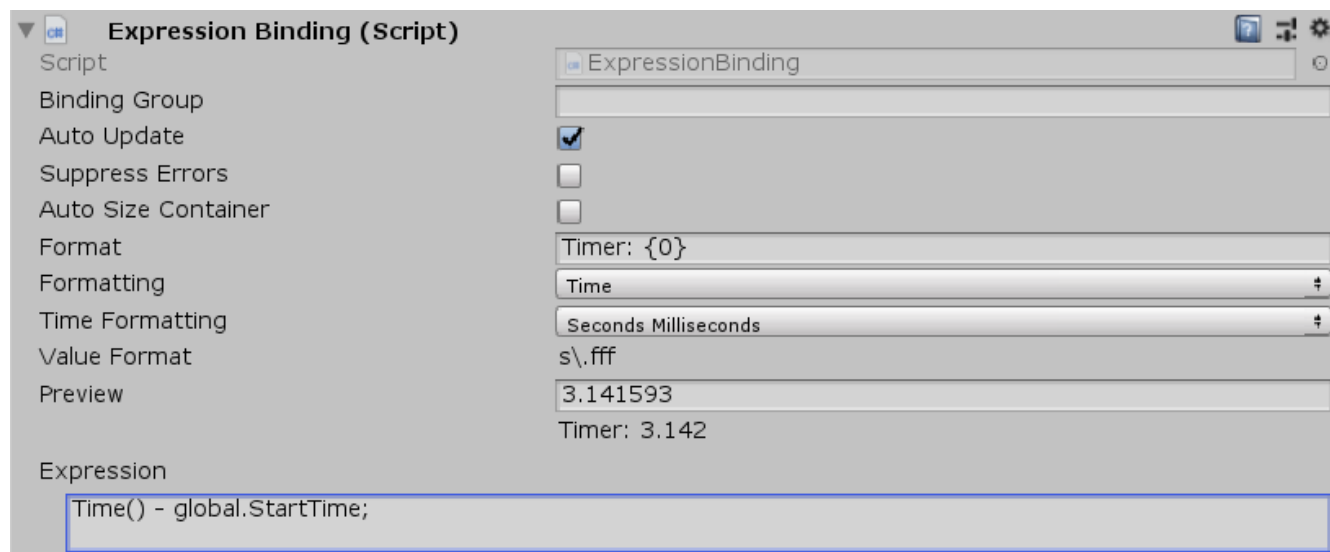
Name	Description
Formatting	The Formatting settings for the text if *Expression* is an int or a float
Expression	The Expression to evaluate to determine the text

Expression Node

An Expression Node is an Instruction Graph Node that runs an Expression. Create an Expression Node in the **Create › Composition › Expression** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See Expressions for more information.
See the *"Apply Damage"* node in "BattleScratch" Instruction Graph in the Battle project for an example usage.

Fields
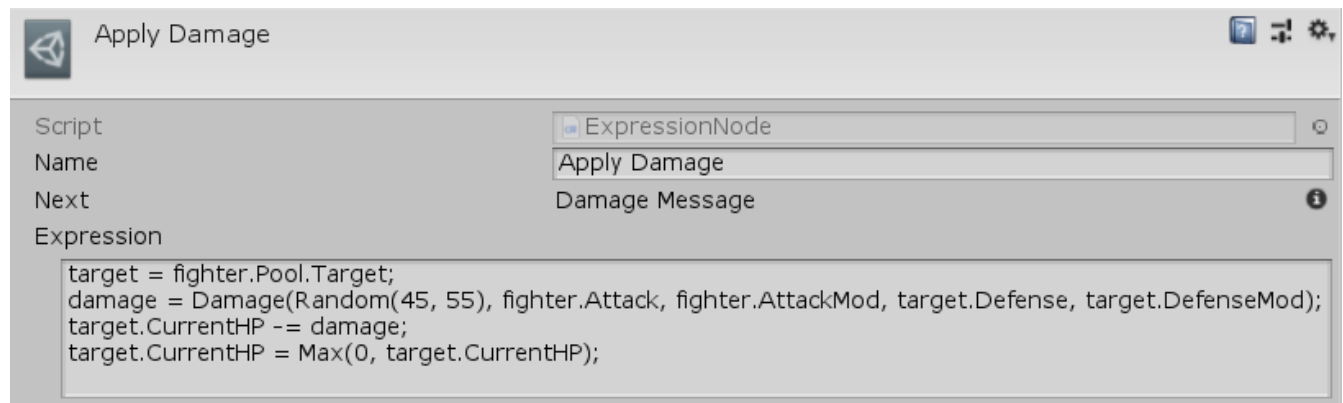
Name	Description
Expression	The Expression to run

Fade Transition

Fade is an implementation of a Cutoff Transition that will fade the screen to a static Color over the duration of the Transition. Create a Fade Transition through the **Create › PiRho Soft › Transitions › Fade** menu in the project view.

Fields

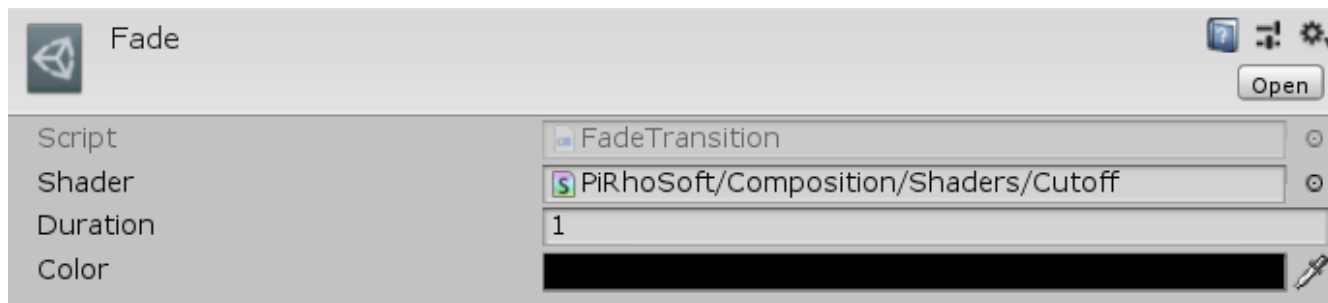| Name	Description
Color	The Color to fade to

Focus Binding Root

Focus Binding Root is a Binding Root that references a Menu in the scene and uses that Menu's currently focused Menu Item as the binding variables for the child Variable Bindings.

See Binding Roots for more information on binding roots.
See Menus for more information on menus.
See the *"Description"* object in the *"Shop"* scene of the Shop project for an example usage.

Fields
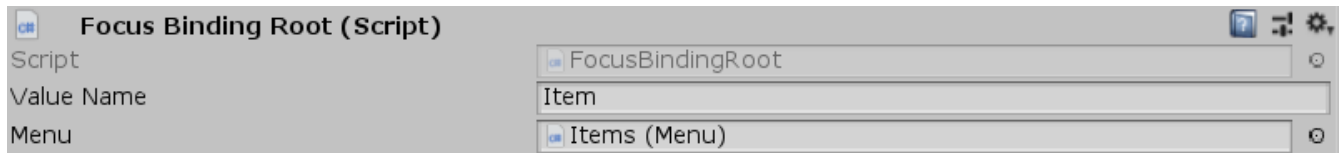
Name	Description
Menu	The Menu whose focused item to use for binding variables

Graph Trigger Binding

Graph Trigger Binding is a Variable Binding that runs an Instruction Graph when a given VariablesReference changes. *Graph* will always run the first time `UpdateBindings()` is called and it will not run again if it is already running.

See Graphs for more information on instruction graphs.
See Variable Bindings for more information on variable bindings.
See the *"Maze3"* scene in the Maze project for an example usage.

Fields
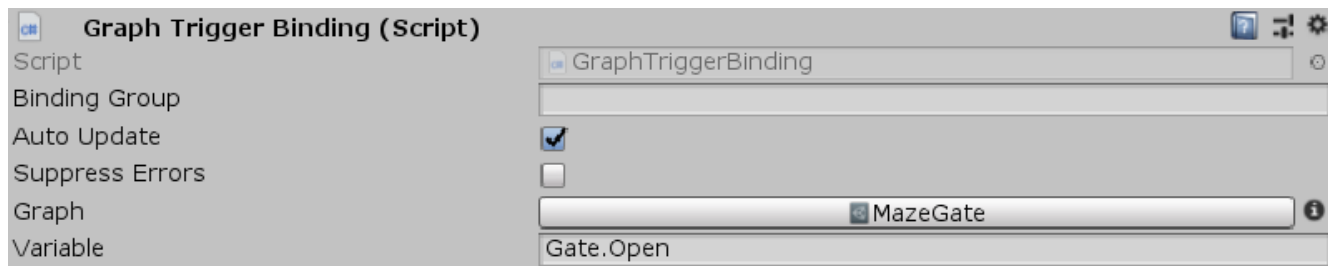
Name	Description
Graph	The Instruction Graph to run when *Variable* changes
Variable	The VariablesReference to watch for changes in

Hide Control Node

A Hide Control Node is an Instruction Graph Node that will deactivate an Interface Control. Create a Hide Control Node in the **Create › Interface › Hide Control** menu of the Instruction Graph Window.

See the Graphs topic for more information on instruction graphs.
See the Controls topic for more information on interface controls.
See the "Hide Timer" node on the "MazeEnd" Instruction Graph in the Maze project for an example usage.

Fields

Name	Description
Control	A VariableReference to the Interface Control to hide

Image Binding

Image Binding is a Variable Binding that will set the Sprite of a sibling Image based on the given
[reference/variable-reference.html/VariableReference]. If *Variable* is invalid, then the image will be
disabled.

See Variable Bindings for more information.
See the *"LootItemDisplay"* prefab in the Loot project for an example usage.

Fields
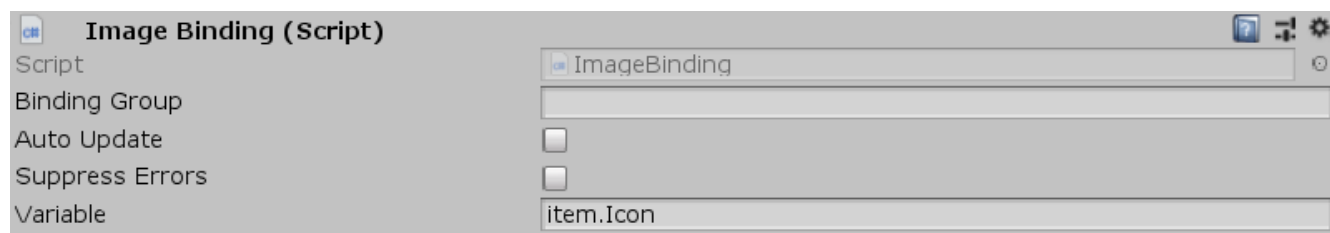
Name	Description
Variable	A VariableReference to the Sprite to display

Image Color Binding

Image Color Binding is a Variable Binding that will set the blend color of a sibling Image based on the given VariableReference. If *Variable* is invalid, then the image will be disabled.

See Variable Bindings for more information.

Fields
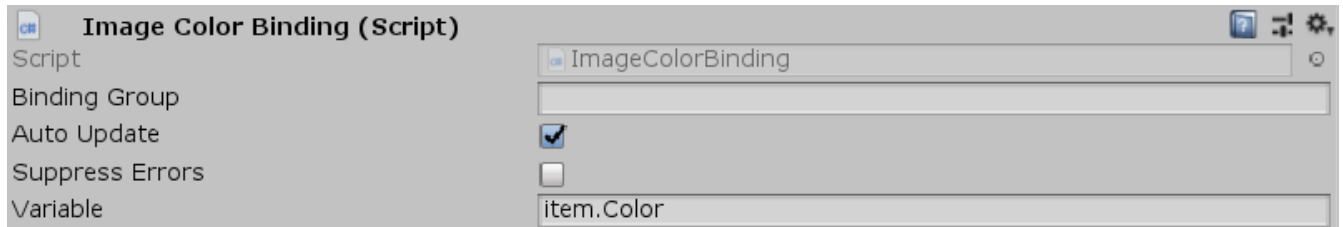
Name	Description
Variable	A VariableReference to the Color to use as the blend color

Input Node

An Input Node is an Instruction Graph Node that will wait to continue to the next node until an input is received. Any number of buttons, keys, or axes can be added to listen for input from, each specifying their own node to branch to. Each input specifies whether it is an `Axis`, `Button`, or `Key`. An `Axis` specifies the value that input axis must reach before triggering. Create a Branch Node in the **Create › Interface › Input** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Calculator"* Instruction Graph in the Calculator project for an example usage.

Fields

Input

Script	InputNode		
Name	Input		

▼ Buttons   +

= Type	Button		−
Name	0		
On Selected	Zero		🛈
= Type	Button		−
Name	1		
On Selected	One		🛈
= Type	Button		−
Name	2		
On Selected	Two		🛈
= Type	Button		−
Name	3		
On Selected	Three		🛈
= Type	Button		−
Name	4		
On Selected	Four		🛈
= Type	Button		−
Name	5		
On Selected	Five		🛈
= Type	Button		−
Name	6		
On Selected	Six		🛈
= Type	Button		−
Name	7		
On Selected	Seven		🛈
= Type	Button		−
Name	8		
On Selected	Eight		🛈
= Type	Button		−
Name	9		
On Selected	Nine		🛈
= Type	Button		−
Name	Plus		
On Selected	Plus		🛈
= Type	Button		−
Name	Minus		
On Selected	Minus		🛈
= Type	Button		−
Name	Divide		
On Selected	Divide		🛈
= Type	Button		−
Name	Multiply		
On Selected	Multiply		🛈
= Type	Button		−
Name	Equals		
On Selected	Equals		🛈
= Type	Button		−
Name	Clear		
On Selected	Initialize		🛈

Name	Description
Buttons	The list of button infos to listen for
Type	The ButtonType to listen for - `Axis`, `Button`, or `Key`
Name	If `Axis` or `Button`, the name of the corresponding input to listen for
Value	If `Axis`, the magintude the axis must reach before being triggered
Key	If `Key`, the KeyCode to listen for

Instruction

Instruction is the base Asset from which all Instruction Graph's derive. The Instruction class defines the VariableDefinitions of the Context and Inputs and Outputs used inside the Instruction Graph.

See Graphs for more information on instruction graphs.

Instruction Graph

An abstract Asset that is the main component of the composition system. Create graphs and use the Instruction Graph Window to build Nodes that execute actions, and create connections to traverse through each node sequentially, similar to a flow chart. An Instruction Graph can have Input Values that are passed in from the calling object and Output Values that it returns to the calling object. All graphs have a *Context* [reference/variable-value.html] which is usually set to the object that originally ran the graph. Edit the definitions for the Definitions for the *Context, Inputs,* and *Outputs* in the inspector for an Instruction Graph, and each of them will automatically be cast to the correct type in script. Built in Instruction Graph types are the Simple Graph, Scoped Graph, and the Mockup Graph (used for prototyping).

See Graphs for more information.

Fields

Name	Description
Context Name	The string name of the context Value to be accessed in Expressions and VariableReferences

Name	Description
Context Definition	The VariableDefinition to cast the context Value to when the Instruction is run
Inputs	The list of VariableDefinitions that describes the inputs into the Instruction
Outputs	The list of VariableDefinitions that describes the outputs from the Instruction
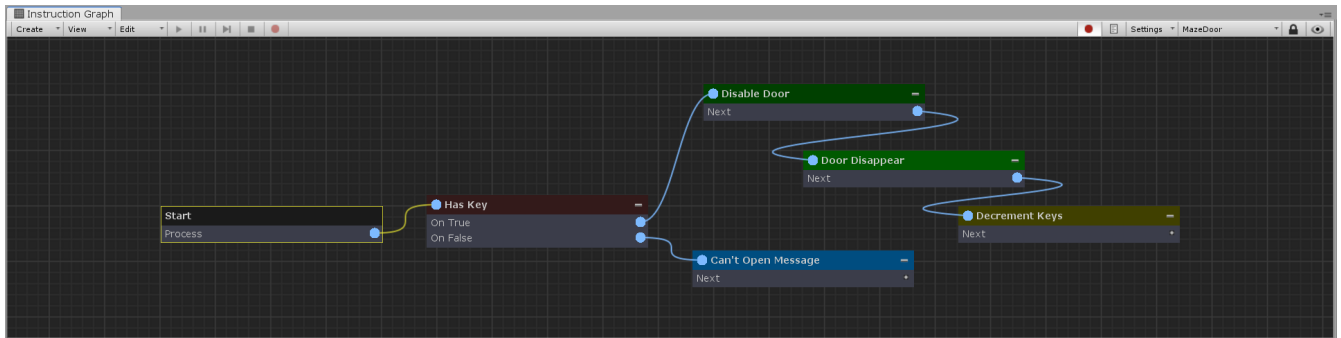
Instruction Graph Node

An abstract Asset that is the main component of an Instruction Graph. This base class has no functionality itself appart from helper methods that may be called from derived classes (see Reference). To implement custom functionality for a node, simply derive and implement the `Run()` method.

See Graphs for more information on instruction graphs.

Fields

Name	Description
Name	A friendly name used to identify this node

Instruction Node

An Instruction Node is an Instruction Graph Node that executes a different Instruction Graph. Intruction Nodes can pass in a new *Context* object to the graph and optionally *WaitForCompletion* of the called graph before moving on to the next node. Create an Instruction Node in the **Create › Composition › Instruction** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Take Turn"* node on the *"BoardLoop"* Instruction Graph in the BoardGame project for an example usage.

Fields

Name	Description
Source	The InstructionSource that specifies whether to run the Instruction Graph from *Instruction* or *Reference*
Instruction	If *Source* is `Value`, the actual Graph to execute
Reference	If *Source* is `Reference`, a VariableReference to the Graph to execute
Context	A VariableValueSource to the context object to be passed to the Graph
Wait For Completion	Whether to wait until the Graph is finished running before moving on to the next node

Instruction Trigger

Instruction Trigger is a MonoBehaviour that stores an Instruction Graph for derived classes or other objects to run manually. Instruction Trigger itself will never trigger *Graph* on its own, only from derived classes when their desired conditions are met and the Run() method is called. Provided implementations of the Instruction Trigger are Start Graph Trigger, Enable Graph Trigger, Click Graph Trigger, and Button Graph Trigger.

See Graphs for more information on instruction graphs.
See the *"Shop"* scene and the *"Player"* script in the Shop project for an example usage.

Fields

Name	Description
Graph	The Instruction Graph to run for this Instruction Trigger

Interface Control

Interface Control is a MonoBehaviour to be attached to UI objects so that they can be manually shown and hidden with the Show Control Node and the Hide Control Node. When loaded an Interface Control always starts inactive until `Activate()` is called. In addition, an Interface Control can maintain a list of GameObjects that are activated and deactivated along with this control.

See the Interface Controls topic for more information.
See the *"Timer"* object in the *"MazeUI"* scene in the Maze project for an example usage.

Fields

Name	Description
Dependent Objects	The list of GameObjects activated and deactivated along with this control
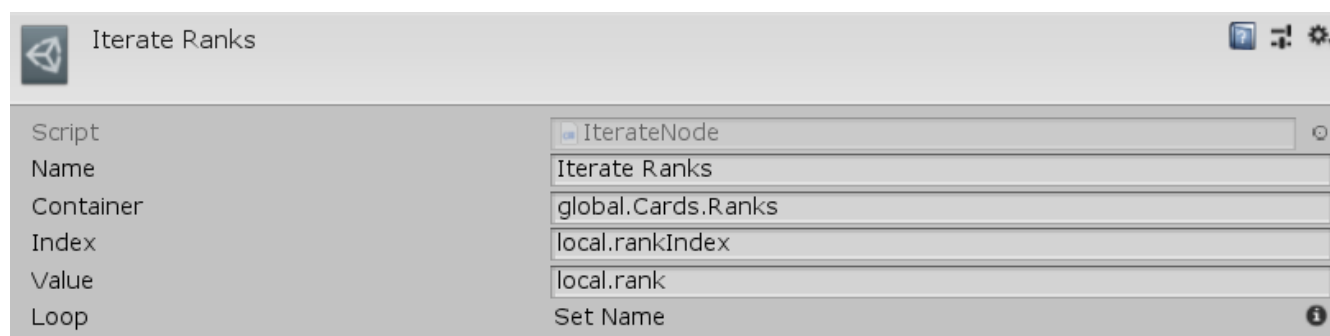
Iterate Node

An Iterate Node is an Instruction Graph Node that implements ILoopNode. It repeatedly runs the next nodes in the graph for each item in the VariableReference, *Container*. *Container* can store either an IVariableStore, or an IVariableList. Each iteration through the loop will store the index of the item in the VariableReference, *Index,* and the value of the item in the VariableReference, *Value.* Create an Iterate Node in the **Create › Control Flow › Iterate** menu of the Instruction Graph Window.

See Control Flow for more information.
See the *"Iterate Ranks"* node on the *"CardLoad"* Instruction Graph in the CardGame project for an example usage.

Fields

Iterate Ranks	
Script	IterateNode
Name	Iterate Ranks
Container	global.Cards.Ranks
Index	local.rankIndex
Value	local.rank
Loop	Set Name

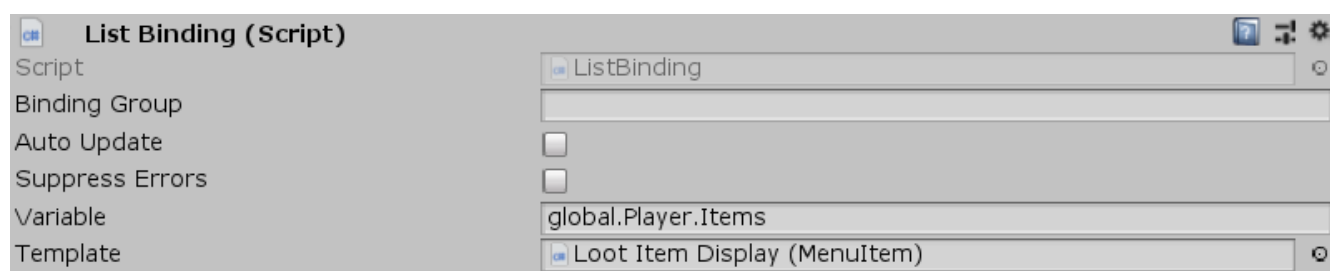| Name	Description
Container	A VariableReference to the IVariableStore or IVariableList to be iterated
Index	A VariableReference to store the index of the item being iterated over
Value	A VariableReference to store the value of the item being iterated over

List Binding

List Binding is a Variable Binding that will create child GameObjects based on *Template* for each Value in the VariableReference, *Variable*. *Variable* must store either an IVariableStore, or an IVariableList. Each *Template* item is instantiated as a Binding Root with its *Value* set to the corresponding item in the store or list. A List Binding is best used in conjunction with a Menu to populate it with Menu Items.

See the Variable Bindings topic for more information.
See the Menus and Selections topic for more information.
See the *"ItemsMenu"* and *"EquipmentMenu"* objects in the *"LootMenu"* scene of the Loot project for an example usage.

Fields

List Binding (Script)	
Script	ListBinding
Binding Group	
Auto Update	☐
Suppress Errors	☐
Variable	global.Player.Items
Template	Loot Item Display (MenuItem)

Name	Description
Variable	A VariableReference to the IVariableStore or IVariableList that child objects should be created for
Template	The prefab (must contain a Binding Root) to use for instantiated child objects

Load Scene Node

A Load Scene Node is an Instruction Graph Node that will load a new Scene. The scene to be loaded can based on a SceneReference, string name, build index, or a VariableReference to a string name or build index. If *WaitForCompletion* is specified, the graph will not move to the next node until the scene has fully completed loading. Create a Load Scene Node in the **Create › Sequencing › Load Scene** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Load UI"* node on the *"MazeStart"* Instruction Graph in the Maze project for an example usage.

Fields
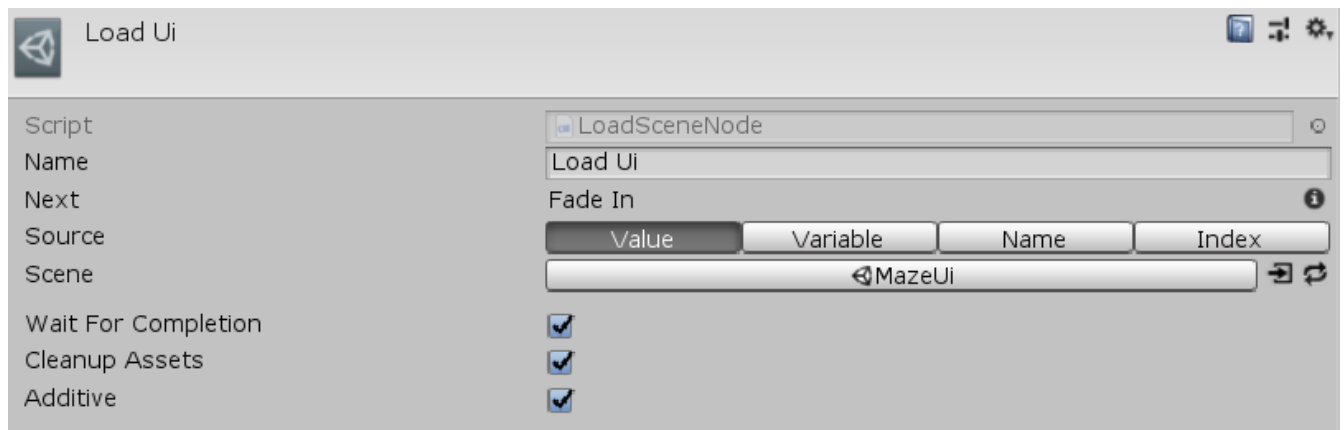
Name	Description
Source	The SourceType to load the scene based off of
Scene	If Value, the SceneReference to load
Scene Variable	If Variable, the VariableReference to a string name or build index of the sceen to load
Scene Name	If Name, the string name of the scene to load
Scene Index	If Index, the build index of the scene to load
Wait For Completion	Whether to wait until the scene has completed loading before moving to the next node
Cleanup Assets	Whether to call UnloadUnusedAssets() after the new scene is loaded
Additive	Whether to load the scene as Additive or Single

Log Node

A Log Node is an Instruction Graph Node that is used for debugging purposes. It logs a Message when the node is executed. Create a Log Node in the **Create › Debug › Log** menu of the Instruction Graph Window.

Fields

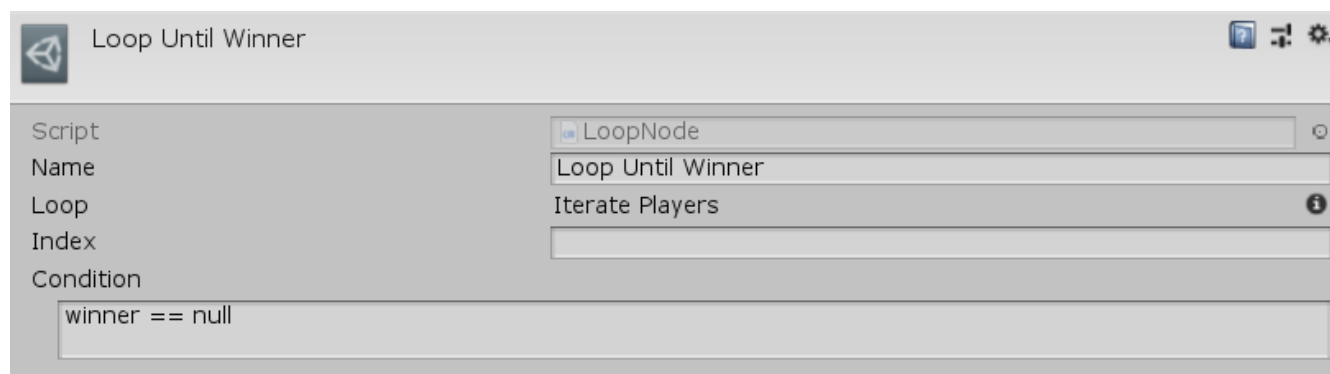| Name	Description
Message	The Message

Loop Node

An Iterate Node is an Instruction Graph Node that implements ILoopNode. It repeatedly runs the next nodes in the graph until the evaluation of the Expression, *Condition* is `false`. Each iteration through the loop will store the index of the item in the VariableReference, *Index*. It can be thought of like `while` loop in script. Create an Iterate Node in the **Create › Control Flow › Iterate** menu of the Instruction Graph Window.

See Control Flow for more information.
See the *"Loop Until Winner"* node on the *"BoardLoop"* Instruction Graph in the BoardGame project for an example usage.

Fields

Name	Description
Index	A VariableReference to store the number of times the loop has run
Condition	The Expression to evaluate to determine if the loop should continue

Material Animation

Material Animation is a MonoBehaviour that will animate a float property named *"_Progress"* on the Material of a sibling Renderer from 0 to 1. If *AutoAdvance* is set then when the behaviour is enabled, the *"_Progress"* property of the Material will increment every frame over the time, *Duration*. If *AutoAdvance* is not set then *Progress* can manually be set either from script or through an AnimationClip and the Material will automatically be updated.

> ℹ️ | The Shader must have a float property named *"_Progress"*

See the *"Scratch"* prefab in the Battle project for an example usage.

Fields
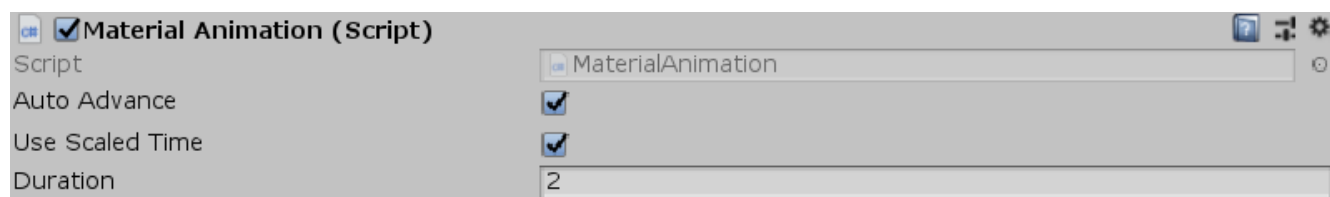
Name	Description
Auto Advance	Whether to automatically start the animation when the behaviour becomes enabled
Progress	If *AutoAdvance* is `false`, the value to set the *"_Progress"* property on the Material to
Use Scaled Time	If *AutoAdvance* is `true`, whether to use DeltaTime or UnscaledDeltaTime to advance *Progress*
Duration	If *AutoAdvance* is `true`, the amount of time (in seconds) the animation will take to occur

Menu

A Menu is a MonoBehaviour that provides an interface for adding, removing, selecting, and focusing child Menu Items. On its own a Menu is mostly just a container for its child MenuItems and requires either a List Binding or a Selection Control to populate it, and a Menu Input to control selection, and focusing. See the *"MazeUi"* scene in the Maze project for an example usage.

See the Menus and Selections topic for more information.
See the *"EquimentMenu"* and *"ItemsMenu"* objects in the *"LootMenu"* scene of the Loot project for an example usage.

Menu Input

Menu Input is a MonoBehaviour that should be attached as a sibling of a Menu to handle the behaviour of input, focusing, selecting, and scrolling, through Menu Items.

See the Menus and Selections topic for more information.
See the *"EquimentMenu"* and *"ItemsMenu"* objects in the *"LootMenu"* scene of the Loot project for an example usage.

Fields
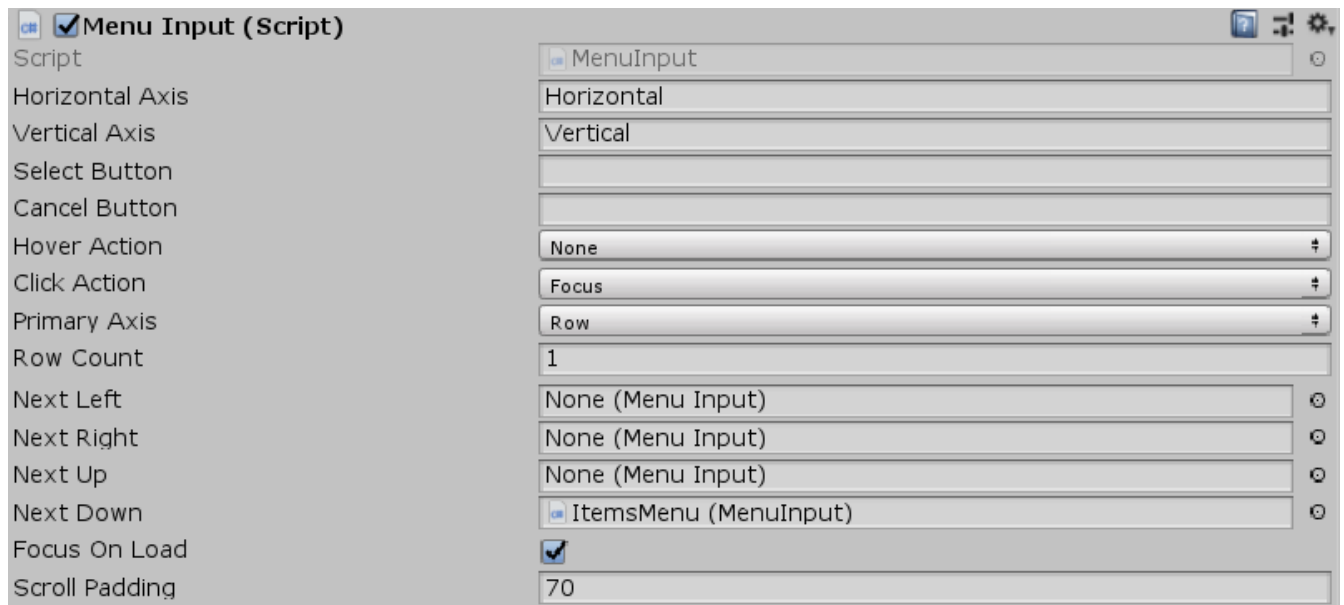
Name	Description
Horizontal Axis	The name of the input axis to use for horizontal movement
Vertical Axis	The name of the input axis to use for vertical movement
Select Button	The input button to use to select the currently focused Menu Item
Cancel Button	The input button to use to cancel the Menu or Selection
Hover Action	The Action to take when a Menu Item is hovered over
Click Action	The Action to take when a Menu Item is clicked on
Primary Axis	Whether the Menu Items are arranged by Row or by Column
Row Count	The number of rows in the menu
Column Count	The number of columns in the menu
Next Left	The menu to transfer input to when moving past the left side of this menu
Next Right	The menu to transfer input to when moving past the right side of this menu
Next Up	The menu to transfer input to when moving past the top of this menu
Next Down	The menu to transfer input to when moving past the bottom of this menu

Name	Description
Focus On Load	Whether to enable input for this menu when it first loads
Scroll Padding	The distance between the edge of the scroll viewport and the Menu Item when focusing on a new a new item

Menu Item

Menu Item is a Binding Root that is used for objects that can be selected as part of a Menu. In addition to the Binding Root's *ValueName* field which exposes the Value that the Menu Item was created from, it exposes the *ItemName* field which gives access to the data of the Menu Item itself. Value's that can be accessed by child Variable Bindings through *ItemName* are *Index, Column, Row, Label*, and *Focused*.

See the Menus and Selections topic for more information.
See the Bindings topic for more information on variable bindings.
See the *"LootItemDisplay"* prefab in the Loot project for an example usage.

Fields
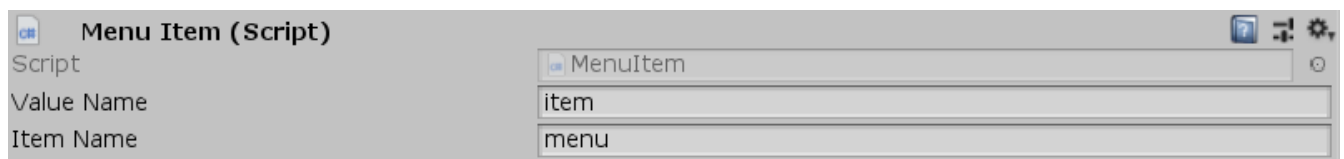
Name	Description
Item Name	The string name used to access this Menu Item's properties by child Variable Bindings
Index	The index of this Menu Item in the menu
Column	The column this Menu Item occupies in its menu
Row	The row this Menu Item occupies in its menu
Label	The string name used to identify the Menu Item
Focused	Whether this Menu Item is currently focused

Message Binding

Message Binding is a String Binding that binding will display a Message on a sibling TextMeshPro component, using the this binding's Binding Root as the variables to resolve the Message.

See the Variable Bindings topic for more information.
See the Messages topic for more information.
See the *"Timer"* object in *"MazeUi"* scene of the Maze project for an example usage.

Fields
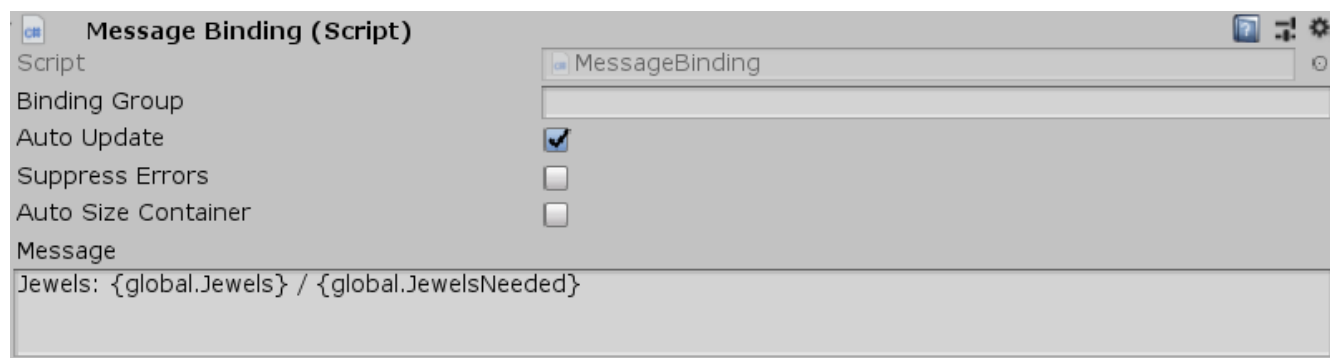
Name	Description
Message	The Message to display

Message Control

A Message Control is an Interface Control that shows a string of text on a TextMeshPro component when promted to. A Message Control is usually activated, shown, and deactivated by a Message Node. Use a Message Input behaviour to manually advance the text.

See the Messages topic for more information.
See the *"Message"* object in *"Board"* scene of the BoardGame project for an example usage.

Fields
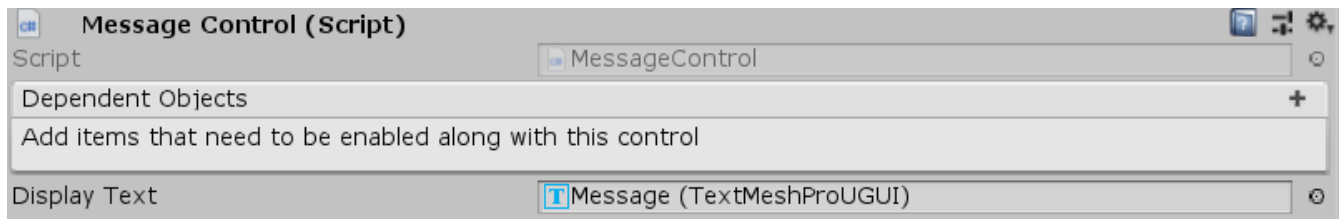
Name	Description
Display Text	The TextMeshPro component to display the Message on

Message Input

Message Input is a MonoBehaviour that should be attached as a sibling of a Message Control to allow the user to advance the text with a button press.

See the Messages topic for more information.
See the *"MazeUi"* scene of the Maze project for an example usage.

Fields
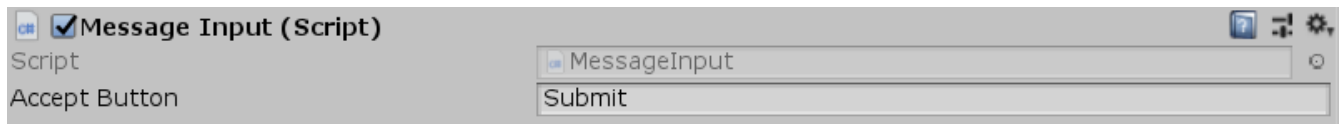
Name	Description
Accept Button	The input button that will advance the text of the Message Control

Message Node
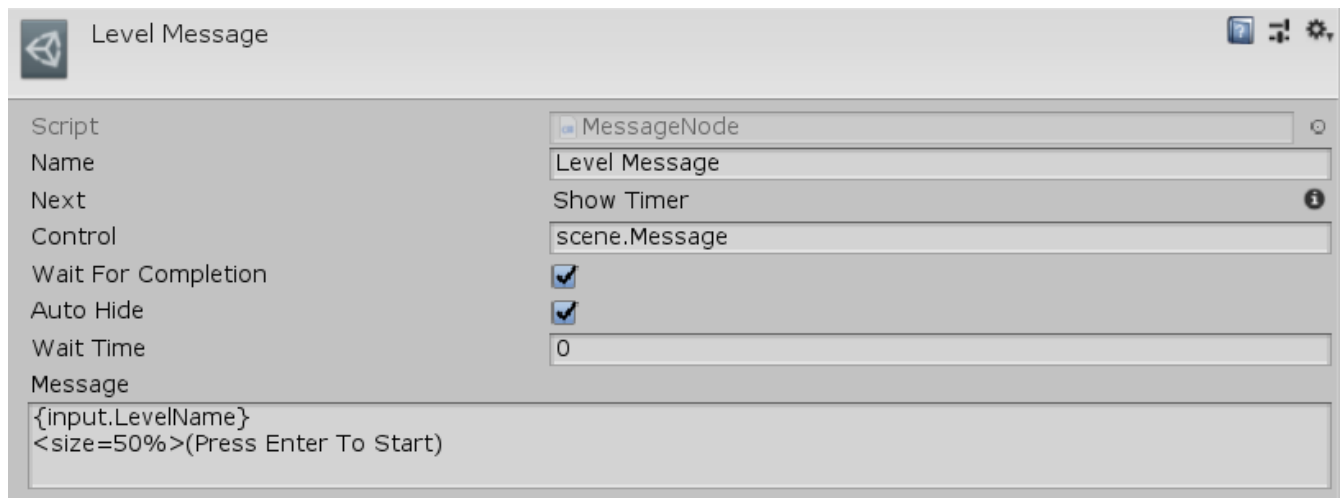
A Message Node is an Instruction Graph Node that will tell a Message Control to show a Message. Showing a Message will automatically activate the Message Control and will hide it when the Message is complete if *AutoHide* is `true`. If *AutoHide* is `false` then a Hide Control Node must be used to deactivate it. Create a Message Node in the **Create › Interface › Message** menu of the Instruction Graph Window.

See the Graphs topic for more information on instruction graphs.
See the Messages topic for more information on messages.
See the *"Shop"* Instruction Graph in the Shop project for an example usage.

Fields

Name	Description
Control	A VariableReference to the Message Control that should display *Message*
Wait For Completion	Whether to wait for *Message* to finish being displayed before moving on to the next node
Auto Hide	Whether to deactivate *Control* when *Message* is finished being displayed
Wait Time	If *AutoHide* is `true`, the amount of time (in seconds) to wait before deactivating *Control*
Message	The Message to display>>

Mockup Graph

A Mockup Graph is an implementation of an Instruction Graph usually used for storyboarding and prototyping in conjunction with Mockup Nodes. A Mockup Graph can be customized to have any number of entry points and are useful to create as a visual template that is turned into a real Instruction Graph later.

See Graphs for more information on instruction graphs.

Fields

Name	Description
Entry Points	The list of starting Nodes that this graph enters into

Mockup Node

A Mockup Node is an Instruction Graph Node usually used in conjunction with a Mockup Graph for the purpose of storyboarding and prototyping. A Mockup Node and have any number of output *Connections* to other nodes. Create a Mockup Node in the **Create › Debug › Mockup** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

Fields
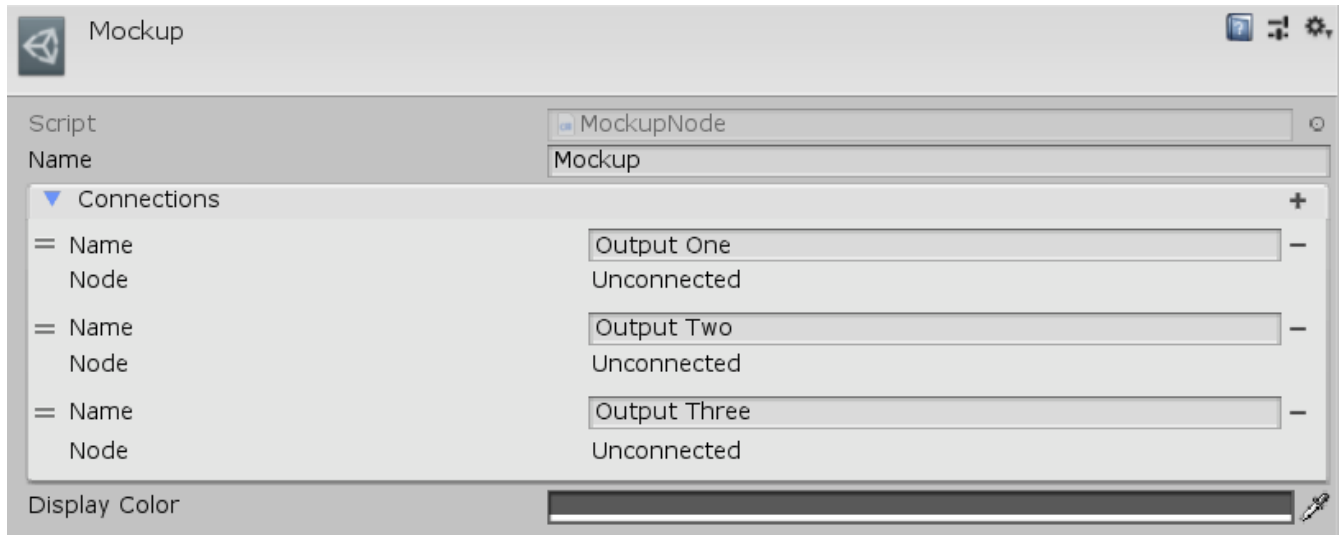
Name	Description
Connections	The list out outputs to other nodes that this node has
Display Color	The Color to display this node as in the Instruction Graph Window
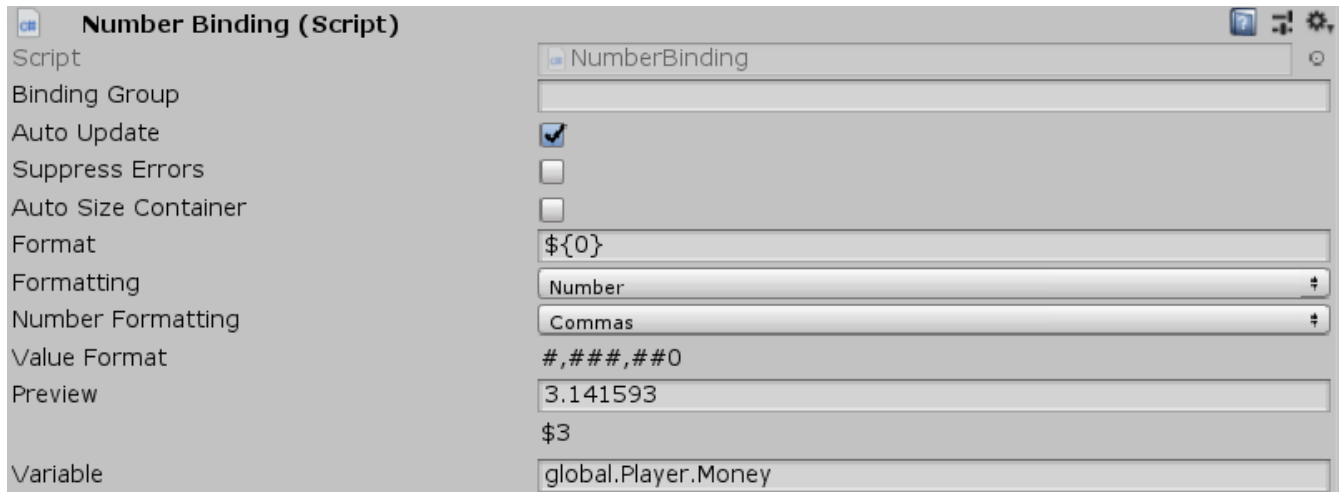
Number Binding

Number Binding is a String Binding that will bind text based on a VariableReference to an int or float with customized Formatting applied. If *Variable* is invalid or does not return an int or float, then the text component will be disabled.

See Variable Bindings for more information.
See the *"Amount"* object in the *"Shop"* scene of the Shop project for an example usage.

Fields

Name	Description
Format	The Formatting to display the number with
Variable	The VariableReference to retrieve the number from

Object Binding Root

Object Binding Root is a Binding Root that references a specific Object and uses it as the binding variables for the child Variable Bindings.

See Binding Roots for more information on binding roots.
See the *"TriangleDisplay"* and *"HexagonDisplay"* objects in the *"Battle"* scene of the Battle project for an example usage.

Fields
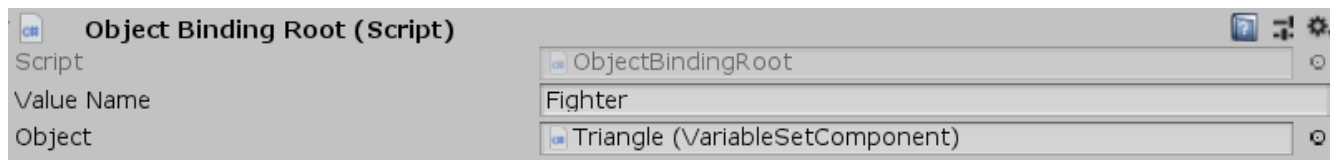
Name	Description
Object	The Object to use as a binding for child Variable Bindings

Pixelate Transition

Pixelate Transition is a Transition that will gradually pixelate the screen over the duration of the Transition. Create a Pixelate through the **Create › PiRho Soft › Transitions › Pixelate** menu in the project view.

Fields

Name	Description
Max Amount	The maximum amount of pixelation

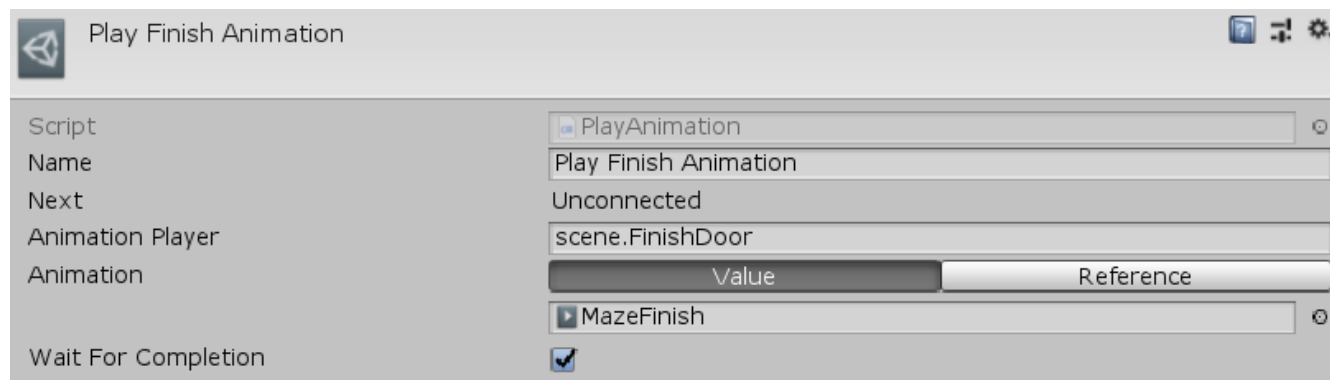# Play Animation Node

Play Animation Node is an Instruction Graph Node that tells an Animation Player to play an AnimationClip. Create a Play Animation Node in the **Create › Animation › Play Animation** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Play Finish Animation"* node on the *"MazeJewel"* Instruction Graph of the Maze project for an example usage.

Fields

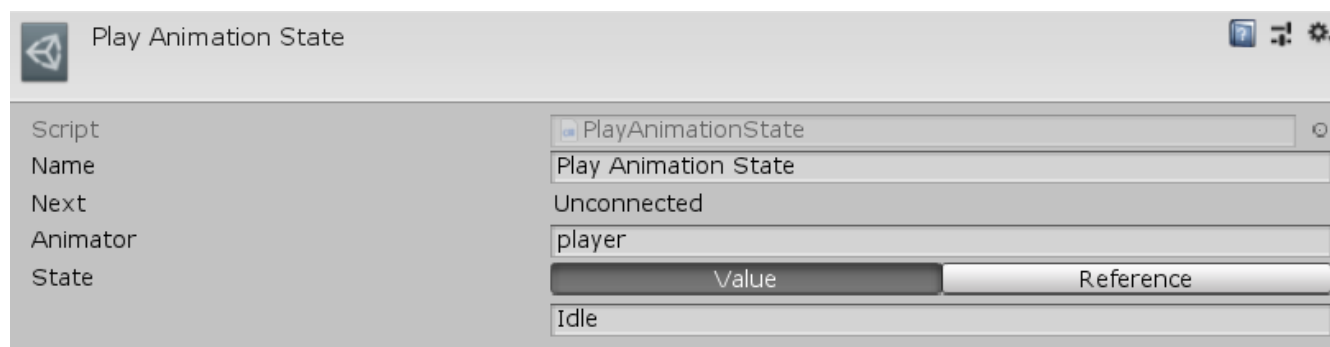| Name	Description
Animation Player	A VariableReference to the Animation Player to play the clip on
Animation	The AnimationClipSource of the AnimationClip to play
Wait For Completion	Whether to wait for *Animation* to finish playing before moving on to the next node

Play Animation State Node

Play Animation State Node is an Instruction Graph Node that tells an Animator to Play() a state. Create a Play Animation State Node in the **Create › Animation › Play Animation State** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

Fields

Name	Description
Animator	A VariableReference to the Animator to play the state on
State	A StringVariableSource to the name of the state to play

Play Effect Node

A Play Effect Node is an Instruction Graph Node that will spawn a prefab at the given Name, Position, and Rotation relative to the world, another object, or as a child object. The created effect can optionally be stored in a given VariableReference so that it can be accessed later. This differs from a standard Create Game Object Node in that a Play Effect Node can *WaitForCompletion* of the effect and destroy it when it is finished. The created effect object is queried for ParticleSystems and other components that implement ICompletionNotifier to determine when the effect is finished.

Create a Play Effect Node in the **Create › Animation › Play Effect** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Play Jewel Sparkle"* node on the *"MazeJewel"* Instruction Graph in the Maze project for an example usage.

Fields

Name	Description
Effect	The prefab of the effect object to create
Effect Name	The name of the new object
Effect Variable	The VariableReference to store the created object in
Positioning	The ObjectPositioning to create the object at
Object	If *Positioning* is `Relative`, the object to position the created object relative to
Parent	If *Positioning* is `Child`, the object to make the parent of the created object

Name	Description
Position	The position of the effect - can be a Vector3 value or a VariableReference
Rotation	The rotation of the effect - can be a Vector3 value or a VariableReference, stored as euler angles
Wait For Completion	Whether to wait until the effect has finished playing before moving on to the next node
Destroy On Complete	Whether to destroy the created effect when it has finished playing

Play Sound Node

Play Sound Node is an Instruction Graph Node that tells an Audio Player to play an AudioClip. Create a Play Sound Node in the **Create › Animation › Play Sound** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Play Pickup Sound"* node on the *"MazeKey"* Instruction Graph of the Maze project for an example usage.

Fields

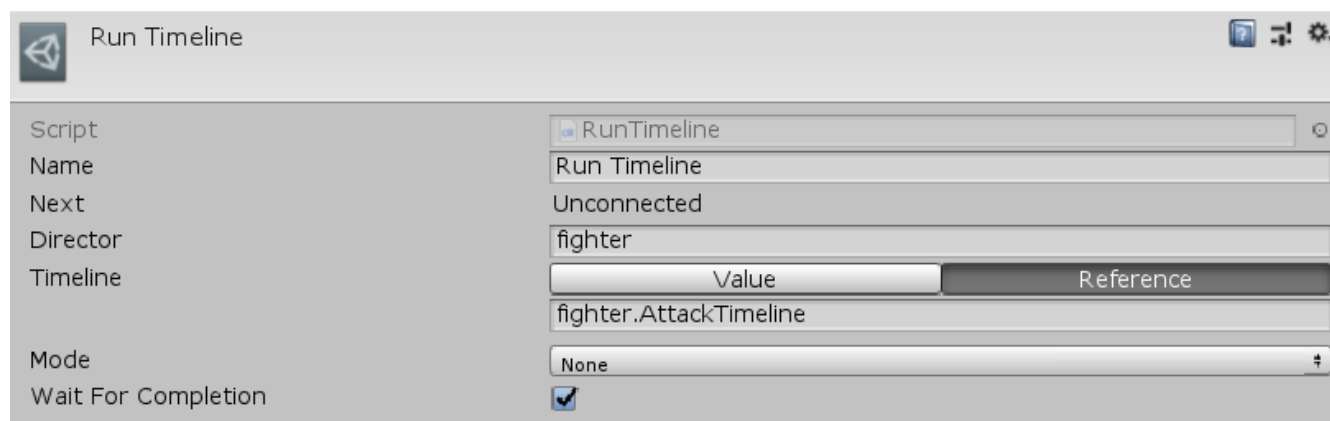| Name	Description
Audio Player	A VariableReference to the Audio Player to play the sound on
Sound	The AudioClipSource of the AudioClip to play
Volume	The FloatVariableSource of the volume to play the sound at
Wait For Completion	Whether to wait for the *Sound* to finish playing before moving on to the next node

Play Timeline Node

Play Timeline Node is an Instruction Graph Node that tells a PlayableDirector to Play() a Timeline. Create a Play Timeline Node in the **Create › Animation › Play Timeline** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

Fields

Name	Description
Director	A VariableReference to the PlayableDirector to run the Timeline on
Timeline	The TimelineVariableSource to run
Mode	The WrapMode to use
Wait For Completion	Whether to wait for the Timeline to finish before moving on to the next node

Play Transition Node

A Play Transition Node is an Instruction Graph Node that tells the Transition Manager to begin a Transition. If the *AutoFinish* field is set then the Transition will complete automatically once its phase is completed, otherwise, either a new Transition needs to be started or a Stop Transition Node needs to be used. A standard example for when *AutoFinish* should be `false` would be to maintain the blank screen after fading out while loading the next scene then fading back in when the loading is done. Create a Play Transition Node in the **Create › Sequencing › Play Transition** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"MazeStart"* Instruction Graph in the Maze project for an example usage.

Fields

Name	Description
Transition	The TransitionVariableSource of the Transition to play
Phase	The TransitionPhase to play
Auto Finish	Whether or not the Transition should end when it is completed or maintain its visual state until a new one is started
Wait For Completion	Whether to wait until the Transition is finished before moving on to the next node

Reset Tag Node

Reset Tag Node is an Instruction Graph Node that calls `ResetTag()` on an object that implements IVariableReset. Usually this used to reinitialize variables on a VariableSetComponent. Create a Reset Tag Node in the **Create › Composition › Reset Tag** menu of the Instruction Graph Window.

See Variables for more information.
See Graphs for more information on instruction graphs.
See the *"Reset Stats"* node on the *"Battle"* Instruction Graph of the Battle project for an example usage.

Fields

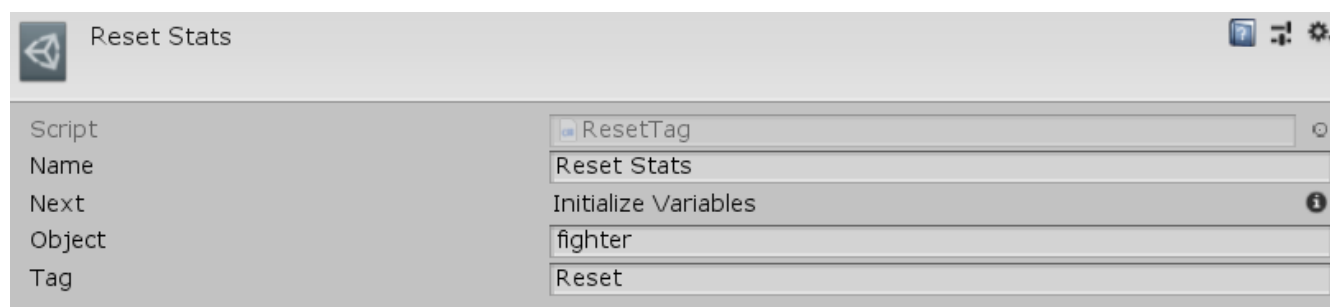| Name	Description
Object	A VariableReference to the IVariableReset object to call `ResetTag()` on
Tag	The string name of the tag to reset

Reset Variables Node

Reset Variables Node is an Instruction Graph Node that calls `ResetVariables()` on an object that implements IVariableReset. Usually this used to reinitialize specific variables on a VariableSetComponent. Create a Reset Variables Node in the **Create › Composition › Reset Variables** menu of the Instruction Graph Window.

See Variables for more information.
See Graphs for more information on instruction graphs.

Fields
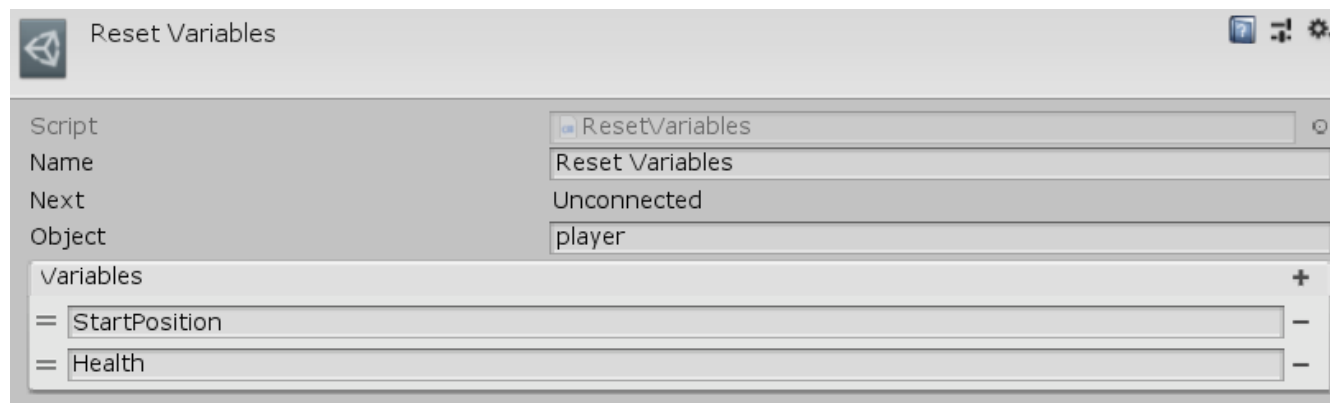
Name	Description
Object	A VariableReference to the IVariableReset object to call `ResetVariables()` on
Variables	The list of string names of the variables to reset

Scoped Graph

A Scoped Graph is an implementation of Instruction Graph with three sequential entry points for Nodes, *"Enter"*, *"Process"*, and *"Exit"*. This is useful for organization of nodes that should have parity of actions when they are starting and finishing (such as disabling and reenabling objects).

See Graphs for more information on instruction graphs.
See the *"MazeStart"* Instruction Graph in the Maze project for an example usage.

Selection Control

A Selection Control is an Interface Control that creates and shows a list of child Menu Items on a sibling Menu to be selected from when promted. A Selection Control usually is activated, shown, and deactivated by a Selection Node. A Selection Control works with a Menu Input which will respond to input for changing focus and selecting an item.

See the Menus and Selections topic for more information on selections.
See the *"Selection"* object in *"Shop"* scene of the Shop project for an example usage.

Selection Node

A Selection Node is an Instruction Graph Node that will tell a Selection Control to show a list of selections created via MenuItemTemplates. Showing a selection will automatically activate the Selection Control and will hide it once a selection has been made if *AutoHide* is `true`. If *AutoHide* is `false` then a Hide Control Node must be used to deactivate it. When a selection is made the selected item and index will be assigned to variables specified by *SelectedItem* and *SelectedIndex*. The graph will then branch to the corresponding node of the selected item. Create a Selection Node in the **Create › Interface › Selection** menu of the Instruction Graph Window.

See the Menus and Selections for more information on selections.
See the Graphs for more information on instruction graphs.
See the *"Shop"* Instruction Graph in the Shop project for an example usage.

Fields

Name	Description
Control	A VariableReference to the Selection Control that should display *Items*
Selected Item	A VariableReference to store the Value of the selected item in
Selected Index	A VariableReference to store the index of the selected item in

Name	Description
Is Selection Required	Whether or not the selection can be canceled (no item is selected)
Auto Hide	Whether to deactivate the *Control* once the selection has been made
Items	The list of Selection Items that will appear in the selection

Sequence Node

A Sequnce Node is a Instruction Graph Node that implements ISequenceNode. It will run each of its child nodes in sequential order. This is useful for organizational purposes or to continue the execution of a graph after a node that does not have an end connection (like an Iterate Node). Create a Sequnece Node in the **Create › Control Flow › Sequence** menu of the Instruction Graph Window.

See Control Flow for more information.
See the *"Battle"* Instruction Graph in the Battle project for an example usage.

Fields
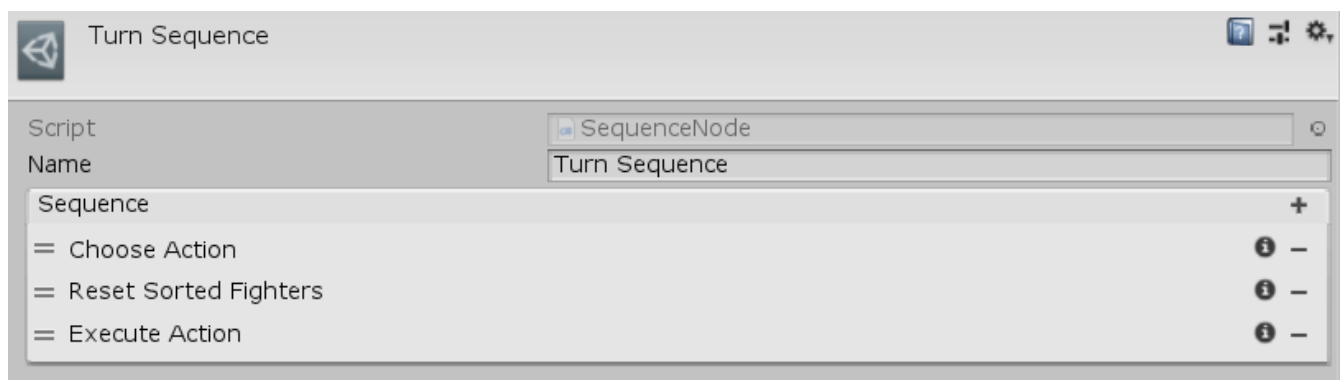
Name	Description
Sequence	The list of Nodes to execute in sequential order

Set Animation Parameter Node

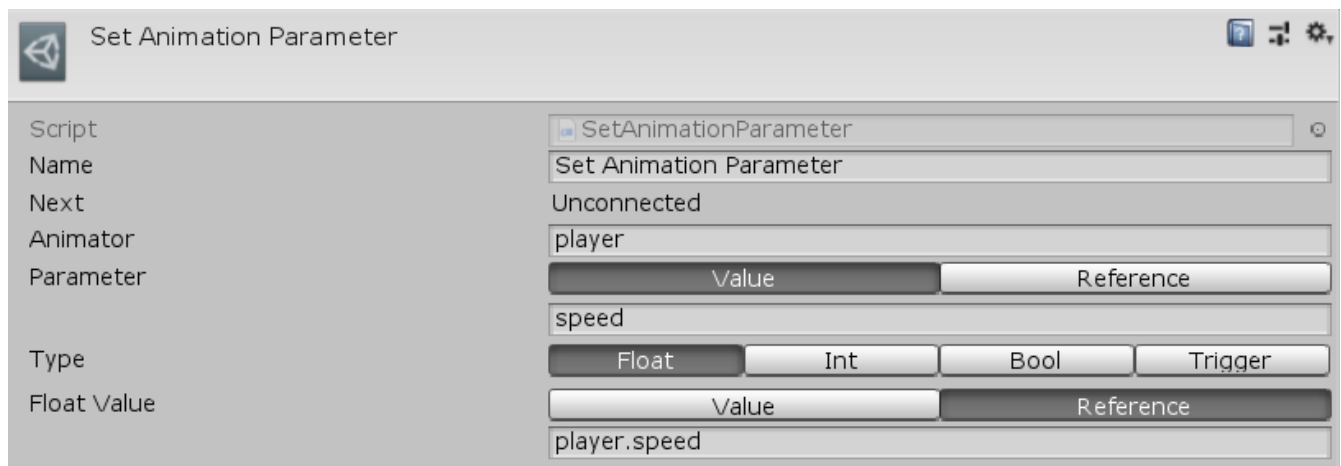Set Animation Parameter Node is an Instruction Graph Node that sets the value of an AnimationParameter on an Animator. Create a Set Animation Parameter Node in the **Create ›** **Animation › Set Animation Parameter** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

Fields

Name	Description
Parameter	A StringVariableSource to the name of the parameter to set
Type	The Type of parameter to set
Animator	A VariableReference to the Animator to set the parameter on
Bool Value	If *Type* is Bool, a BoolVariableSource to the value of the bool to set
Int Value	If *Type* is Int, a IntVariableSource to the value of the int to set
Float Value	If *Type* is Float, a FloatVariableSource to the value of the float to set

Set Binding Node

Set Binding Node is an Instruction Graph Node that sets the Value of a Binding Root to the resolved VariableReference, *Binding*. Create a Set Binding Node in the **Create › Interface › Set Binding** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See Variable Bindings for more information on variable bindings.
See the *"Set Card Binding"* node on the *"CardLoad"* Instruction Graph in the CardGame project for an example usage.

Fields

Name	Description
Object	The VariableReference to the Binding Root to set the binding on
Binding	The VariableReference to set as the binding

Show Control Node

A Show Control Node is an Instruction Graph Node that will activate an Interface Control. Create a Show Control Node in the **Create › Interface › Show Control** menu of the Instruction Graph Window.

See the Graphs for more information on instruction graphs.
See the Controls topic for more information on interface controls.
See the "Show Timer" node on the "MazeStart" Instruction Graph in the Maze project for an example usage.

Fields
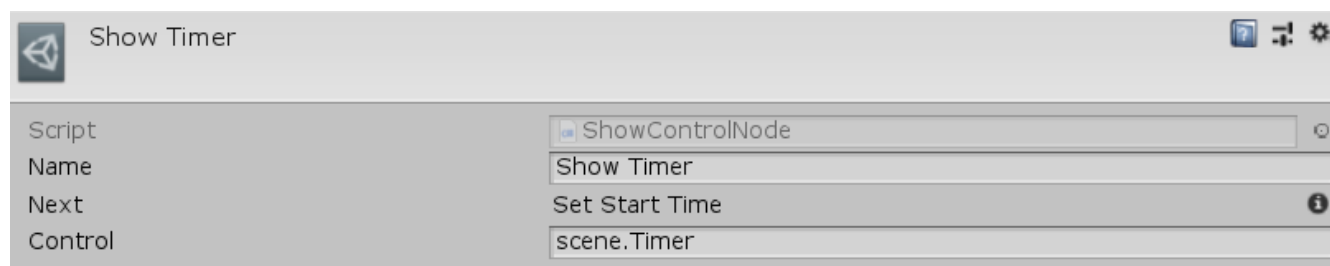
Name	Description
Control	A VariableReference to the Interface Control to show

Shuffle Node

A Shuffle Node is an Instruction Graph Node that will shuffle an IVariableList. Create a Shuffle Node in the **Create › Composition › Shuffle** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Shuffle Cards"* node on the *"CardShuffle"* Instruction Graph in the CardGame project for an example usage.

Fields
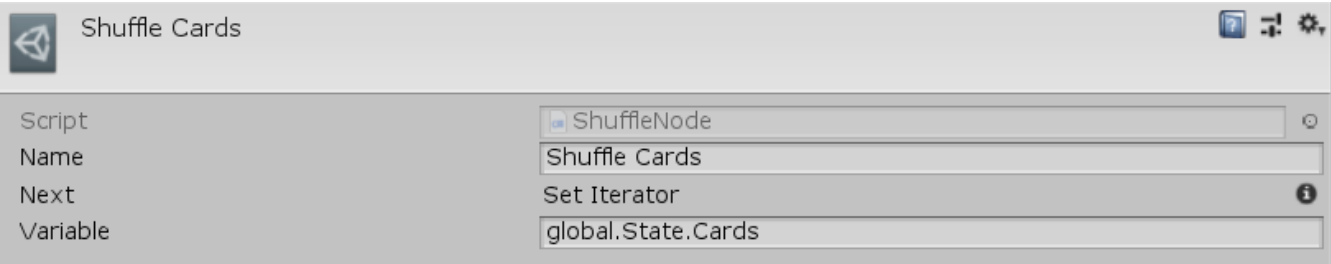
Name	Description
Variable	A VariableReference to the IVariableList to shuffle

Simple Graph

A Simple Graph is the most basic implementation of an Instruction Graph and should be sufficient for most use cases. A Simple Graph has a single entry point *Process* with no other special behaviour.

See Graphs for more information on instruction graphs.

Sort Node

A Shuffle Node is an Instruction Graph Node that will shuffle the given VariableList, *List*. If each Value in *List* is an IVariableStore, then they can be sorted by Values on that IVariableStore (similar to *System.Linq*'s `SortBy().ThenBy()` methods). Create a Sort Node in the **Create › Composition › Sort** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Sort Fighters"* node on the *"Battle"* Instruction Graph in the Battle project for an example usage.

Fields

Name	Description
List	A VariableReference to the VariableList to sort
Sort By Property	Whether to sort each Value in *List* by itself or by properties on it (if it is an IVariableStore)
Sort Conditions	If *SortByProperty* is `true`, then a list of VariableReferences on each item in *List* to sort by sequentially

Sprite Binding

Sprite Binding is a Variable Binding that will set the Sprite of a sibling SpriteRenderer based on the given [reference/variable-reference.html/VariableReference]. If *Variable* is invalid, then the renderer will be disabled.

See Variable Bindings for more information.
See the *"Card"* prefab in the CardGame project for an example usage.

Fields
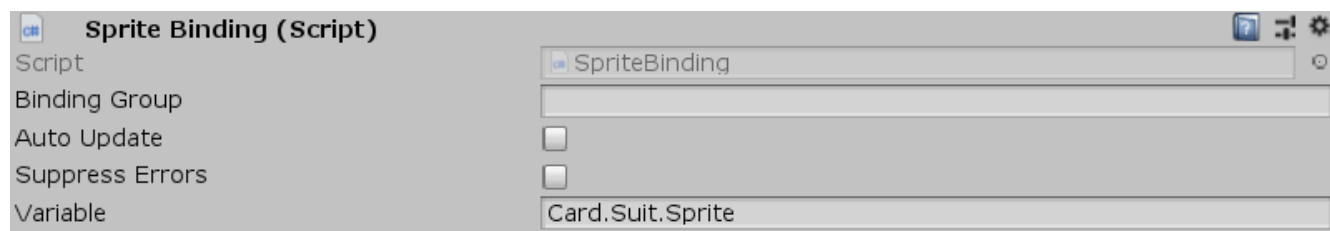
Name	Description
Variable	A VariableReference to the Sprite to display

Sprite Color Binding

Sprite Color Binding is a Variable Binding that will set the blend color of a sibling SpriteRenderer based on the given VariableReference. If *Variable* is invalid, then the renderer will be disabled.

See Variable Bindings for more information.
See the *"Card"* prefab in the CardGame project for an example usage.

Fields
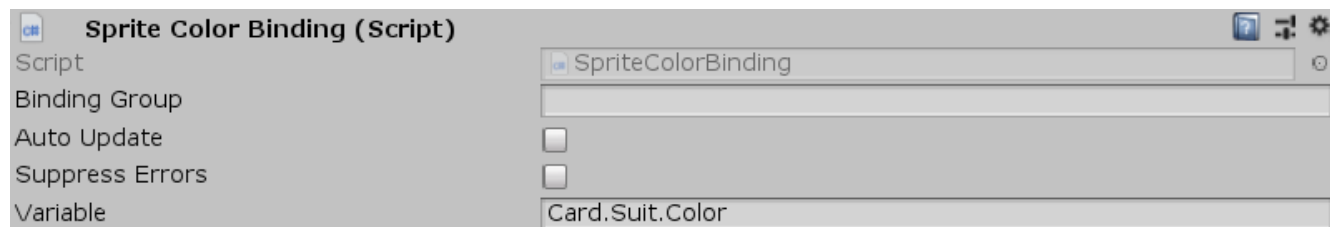
Name	Description
Variable	A VariableReference to the Color to use as the blend color

Start Graph Trigger

Start Graph Trigger is an Instruction Trigger that will run its Instruction Graph when this object is loaded (in its `Start()` message).

See Graphs for more information on instruction graphs.
See the "Player" object in the *"Maze1"* scene of the Maze project for an example usage.

Stop Transition Node

A Stop Transition Node is an Instruction Graph Node that is used to manually end the current Transition. Generally ending a Transition happens with the *AutoFinish* property on a Play Transition Node however, in the case that a Transition needs to be ended manually this node is available. Create a Stop Transition Node in the **Create › Sequencing › Stop Transition** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

String Binding

String Binding is an abstract Variable Binding that should be derived from in order to bind string data for display on a sibling TextMeshPro component. The TextMeshPro container can be optionally set to fit the size of the bound text. Example implementations of a String Binding are Expression Binding, Message Binding, Number Binding, and Text Binding.

See Variable Bindings for more information.

Fields

Name	Description
Auto Size Container	Whether to size the TextMeshPro container to fit the bound text.

Text Binding

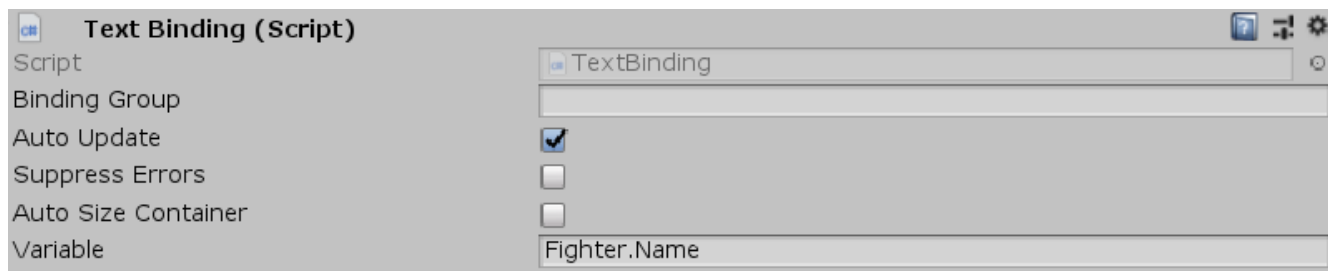Text Binding is a String Binding that will bind text based on a VariableReference. If *Variable* is invalid then the text component will be disabled.

See Variable Bindings for more information.
See the *"Name"* objects in the *"Battle"* scene of the Battle project for an example usage.

Fields

Name	Description
Variable	The VariableReference to retrieve the text from

Text Color Binding

Text Color Binding is a Variable Binding that will set the color of a sibling TextMeshPro component based on the given VariableReference. If *Variable* is invalid, then the text will be disabled.

See Variable Bindings for more information.
See the *"Card"* prefab in the CardGame project for an example usage.
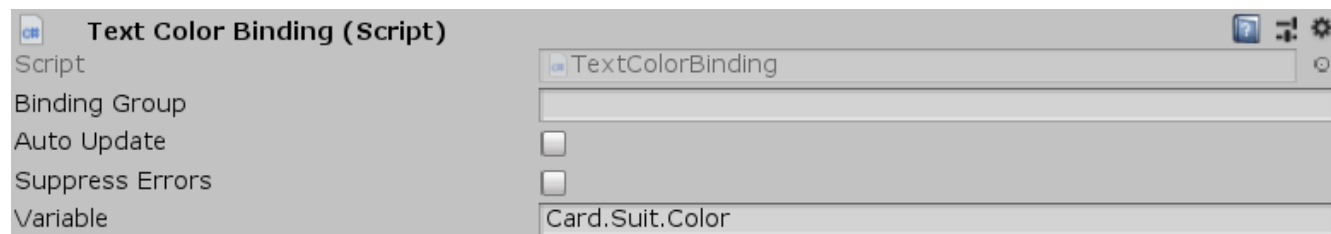
Fields

Name	Description
Variable	A VariableReference to the Color to use

Text Input Binding

Text Input Binding is a Variable Binding that will receive input on a sibling TextMeshProInputField component assign the text to the VariableReference, *Variable*. This is a two-way binding so if `UpdateBindings()` is called on this Variable Binding, then the displayed text will also be updated.

See Variable Bindings for more information.

Fields

Name	Description
Variable	The VariableReference to retrieve/set the input field's text on

Time Scale Node

A Time Scale Node is an Instruction Graph Node that is used to set Unity's TimeScale. This can be useful for pausing. Create a Time Scale Node in the **Create › Sequencing › Time Scale** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.

Fields

Name	Description
Time Scale	A FloatVariableSource to the time scale to set

Transform Node

Transform Node is an Intruction Graph Node, that will animate a Transform component toward a *TargetPosition*, *TargetRotation*, and *TargetScale*. Targets can be specified to be relative to the current *Transform* or in world space. Animation can happen instantly or at either a speed or over a duration. If the target Transform has a Rigidbody attached to it then the animation will use its corresponding movement methods to maintain its state in the collision system. **Create › Sequencing › Transform Object** menu of the Instruction Graph Window.

> If the *Transform* cannot reach its destination (due to collisions or other circumstances) then the node will never cease its execution

See Graphs for more information on instruction graphs.
See the "Move" node on the "MazeGate" Instruction Graph in the Maze project for an example usage.

Fields

Name	Description
Transform	The VariableReference to the Transform to enable
Use Relative Position	If set *TargetPosition* will specified relative to the initial *Transform*

Name	Description
Use Relative Rotation	If set *TargetRotation* will specified relative to the initial *Transform*
Use Relative Scale	If set *TargetScale* will specified relative to the initial *Transform*
Target Position	The Vector3VariableSource to the target position of *Transform*
Target Rotation	The Vector3VariableSource to the target rotation (in euler angles) of *Transform*
Target Scale	The Vector3VariableSource to the target position of *Transform*
Animation Method	The Type of animation to use
Wait For Completion	Whether to wait until *Transfrom* reaches the destination before moving on to the next node
Duration	If *AnimationMethod* is Duration, a FloatVariableSource to the amount of time (in seconds) it takes to reach the destination
Move Speed	If *AnimationMethod* is Speed, FloatVariableSource to the speed the *Transform* will move
Rotation Speed	If *AnimationMethod* is Speed, FloatVariableSource to the speed the *Transform* will rotate
Scale Speed	If *AnimationMethod* is Speed, FloatVariableSource to the speed the *Transform* will change size

Transition

A Transition is an Asset used to create post processing effects on a Camera, usually used during scene changes to hide, obscure, then reshow the screen. Transition itself is an abstract class that should be derived from to achieve the desired effects. Transitions operate using Shaders, which are set in the editor with the *Shader* field. Transitions have three Phases, Out, Obscure, In, with each one being initiated separately, usually by a Play Transition Node. For example implementations of a Transition see the Fade, Dissolve, and Pixelate Transitions.

Fields

Name	Description
Shader	The Shader that this Transition will use to display its effect
Duration	The amount of time this Transition will take to complete.

Transition Manager

The Transition Manager is a MonoBehaviour that manages the transition post-processing system. Because Transition Manager is a GlobalBehaviour, it is created automatically the first time it is accessed so it does not need to be added to any objects in a scene. The Transition Manager works in conjunction with Transition Renderers, attached to Cameras to render post processing effects onto the screen. Generally, Transitions are started by a Play Transition Node, however, they can be run from script using the RunTransition() or StartTransition() method. The methods are both IEnumerators and should be called using the MonoBehaviour.StartCoroutine() method. RunTransition() will fully run the given Phase of the Transition, ending it once it has finished, while StartTransition() will run the Transition until the end without clearing, maintaining its final state until EndTransition() is called or a new Transition is started. << Only a single Transition can be running at a time.

Transition Renderer

Transition Renderer is a MonoBehaviour to be attached to a Camera in order to have the camera's contents rendered as part of a Transition's post-processing.

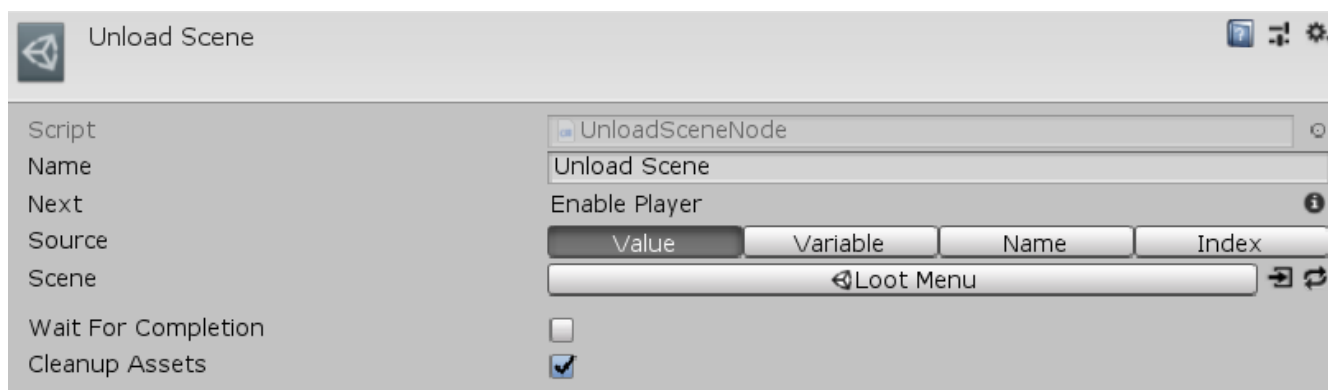# Unload Scene Node

An Unload Scene Node is an Instruction Graph Node that will unload Scene. The scene to be unloaded can be a SceneReference, string name, build index, or a VariableReference to a string name or build index. If *WaitForCompletion* is specified, the graph will not move to the next node until the scene has fully completed unloading. Create an Unload Scene Node in the **Create › Sequencing › Unload Scene** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"Unload Scene"* node on the *"LootHideInventory"* Instruction Graph in the Loot project for an example usage.

Fields

Name	Description
Source	The SourceType to unload the scene based off of
Scene	If `Value`, the SceneReference to unload
Scene Variable	If `Variable`, the VariableReference to a string name or build index of the sceen to unload
Scene Name	If `Name`, the string name of the scene to unload
Scene Index	If `Index`, the build index of the scene to unload
Wait For Completion	Whether to wait until the scene has completed unloading before moving to the next node
Cleanup Assets	Whether to call UnloadUnusedAssets() after scene is unloaded

Update Binding Node

Udpate Binding Node is an Instruction Graph Node that tells the binding root Binding Root, *Object* to update its VariableBindings. If *Object* is an Interface Control then each of the control's *DependentObjects* will also have its bindings updated. If any of the bindings utilize animation (such as Bar Binding) and *WaitForCompletion* is set, then the graph will not move to the next node until the binding animation is finished. Use the *Group* string to identify only the VariableBindings with the cooresponding *BindingGroup* to update. Create an Update Binding Node in the **Create › Interface › Update Binding** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See Variable Bindings for more information on variable bindings.
See the *"Update Binding"* node on the *"BattleScratch"* Instruction Graph in the Battle project for an example usage.

Fields

Name	Description
Object	The VariableReference to the Binding Root to set the binding on
Group	The string name of the *BindingGroup* to update - if empty, all VariableBindings will be updated
Wait For Completion	Whether to wait for the completion of any binding animations before moving on to the next node

Variable Binding

A Variable Binding is an abstract MonoBehaviour used to bind data from the composition system, usually via VariableReferences to visual elements in the scene. By default a Variable Binding has access to the Composition Manager's "global" and "scene" IVariableStores. If a Variable Binding has a parent or parents with a Binding Root then they can access the *Value* property on those Binding Roots via their *ValueName* property. Variable Bindings can be categorized into groups with the *BindingGroup* property so that only certain bindings will update when they are prompted to (usually by an Update Binding Node. If *AutoUpdate* is set then Variable Bindings will update automatically every frame. Most Variable Bindings will disable their corresponding visual element if they fail to retreive their data and report the error, however, sometimes this may be intended behaviour so if set, *SuppressErrors* will hide those errors. Some example implementations of a Variable Binding are Enable Binding, Image Binding, and String Binding.

See Variable Bindings for more information.

Fields

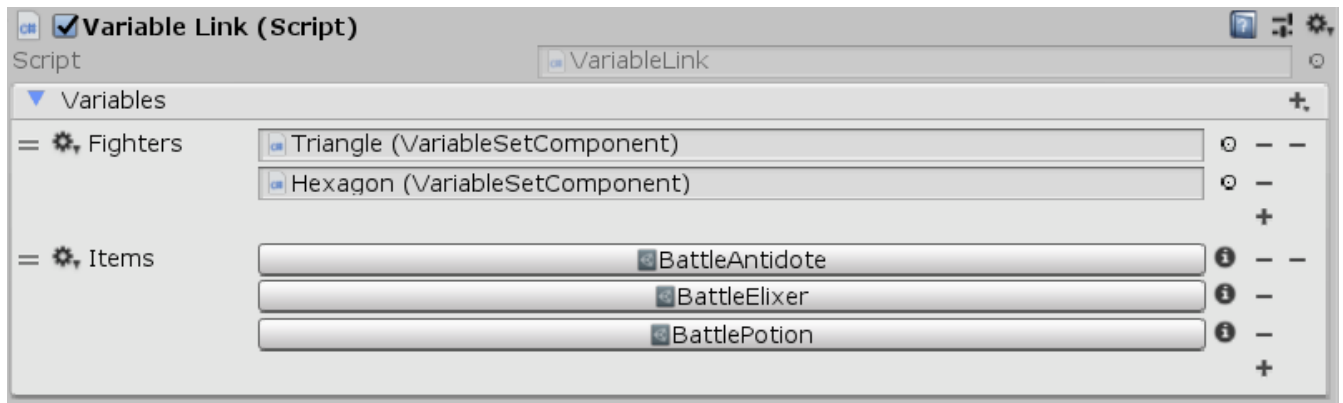| Name	Description
Binding Group	The string name of the group this binding belongs to
Auto Update	Whether to automatically update this binding every frame
Suppress Errors	Whether to hide errors reported from invalid variable accsess

Variable Link

A Variable Link is a MonoBehaviour used to add Variables to the Composition Manager's `global` Variable Store.

See Variables for more information.
See the *"Main Camera"* object in the *"Battle"* scene of the Battle project for an example usage.

Fields

Name	Description
Variables	A Variable Pool, each of whose Variable Values will be added to the `global` store

Variable Pool Asset

A Variable Pool Asset is a ScriptableObject that adds a Variable Pool to the asset. This allows it to have a list of Variables without being constrained by a Variable Schema. Create a Variable Pool Asset through the **Create › PiRho Soft › Variable Pool** menu in the project view.
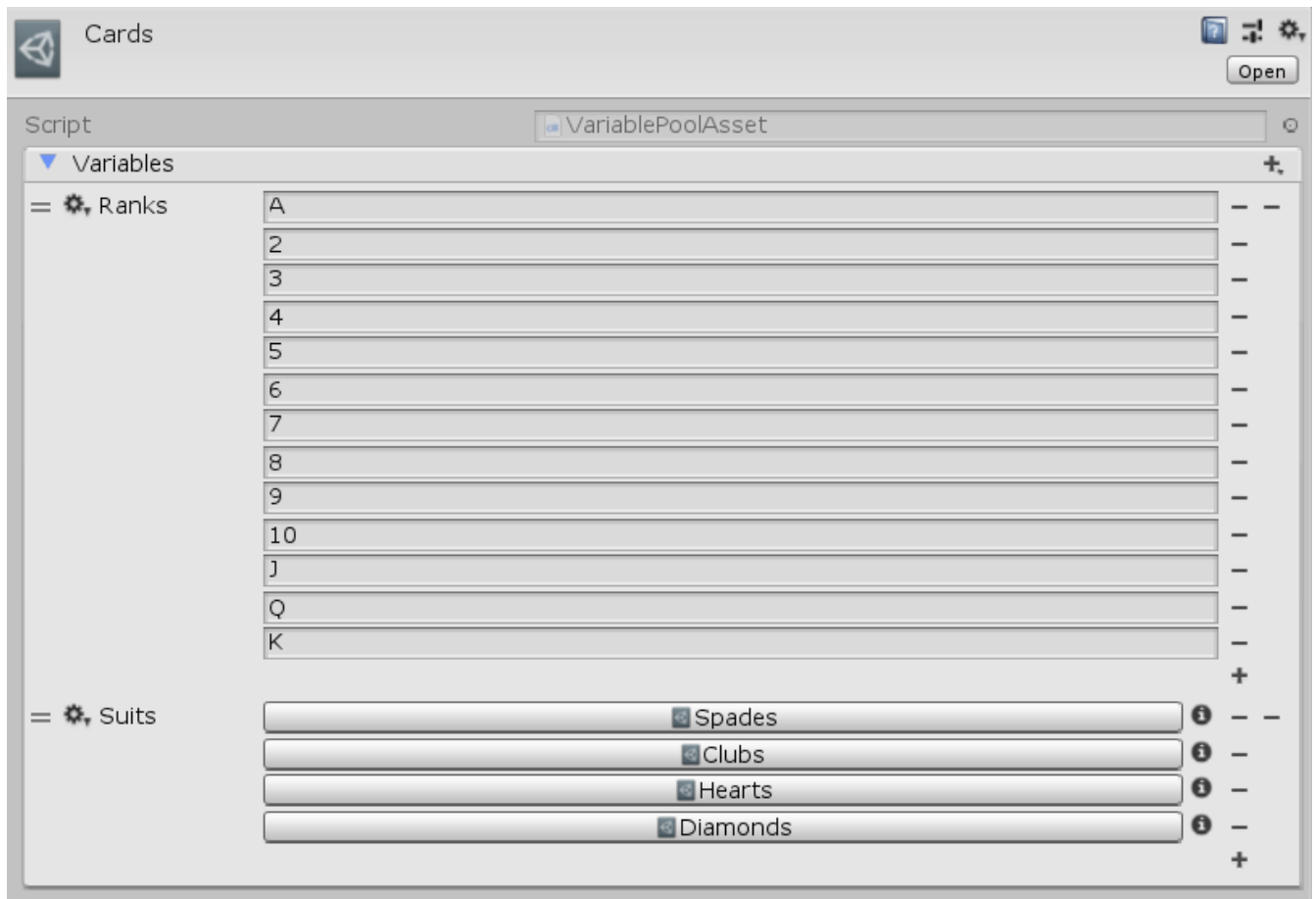
See Variables for more information.

Fields

Name	Description
Variables	The Variable Pool of Values that can be set, stored, and accessed on this object

Variable Pool Component

A Variable Pool Component is a MonoBehaviour that adds a Variable Pool to the object. This allows it to have a list of Variables without being constrained by a Variable Schema.

See Variables for more information.
See the *"LootEquipmentPickup"* prefab in the Loot project for an example usage.
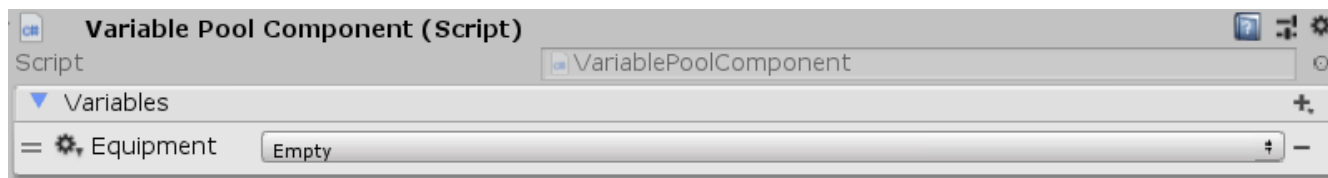
Fields

Name	Description
Variables	The Variable Pool of Values that can be set, stored, and accessed on this object

Variable Schema

A VariableSchema is an Asset used to define the variables that are available to an IVariableStore object, usually a Variable Set Component or Variable Set Asset. This improves the editor experience for working with those object types along with enforcing constraints so typos or other mistaken accesses can be caught and reported at runtime. Create a Variable Schema through the **Create ›** **PiRho Soft › Variable Schema** menu in the project view.

See Variables for more information.

Fields

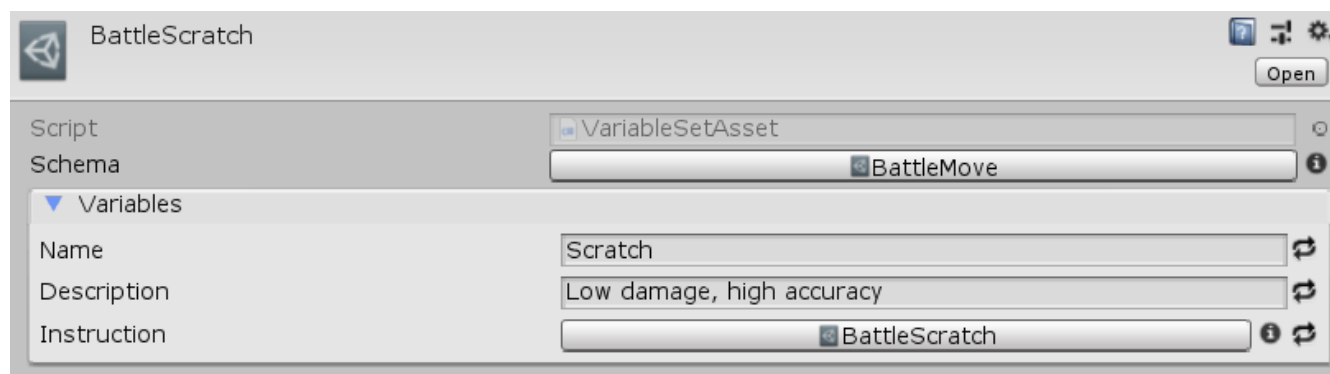| Name	Description
Initializer Type	Whether to initialize the variables in that use this schema with an Expression or a default value
Tags	A list of strings than can be used to categorize each variable in the schema - usually so that they can be reinitialized or saved later

Variable Set Asset

A Variable Set Asset is a ScriptableObject that adds a Variable Set to the asset. This allows it to have a list of Variables that are defined by a reference to a Variable Schema. A derived class of Variable Set Asset can use the MappedVariableAttribute to add its fields and properties to the *Variables* list in addition to those defined by *Schema*. Create a Variable Set Asset through the **Create › PiRho Soft › Variable Set** menu in the project view.

See Variables for more information.
See the assets in the *"Moves"* and *"Items"* folders in the Battle project for an example usage.

Fields

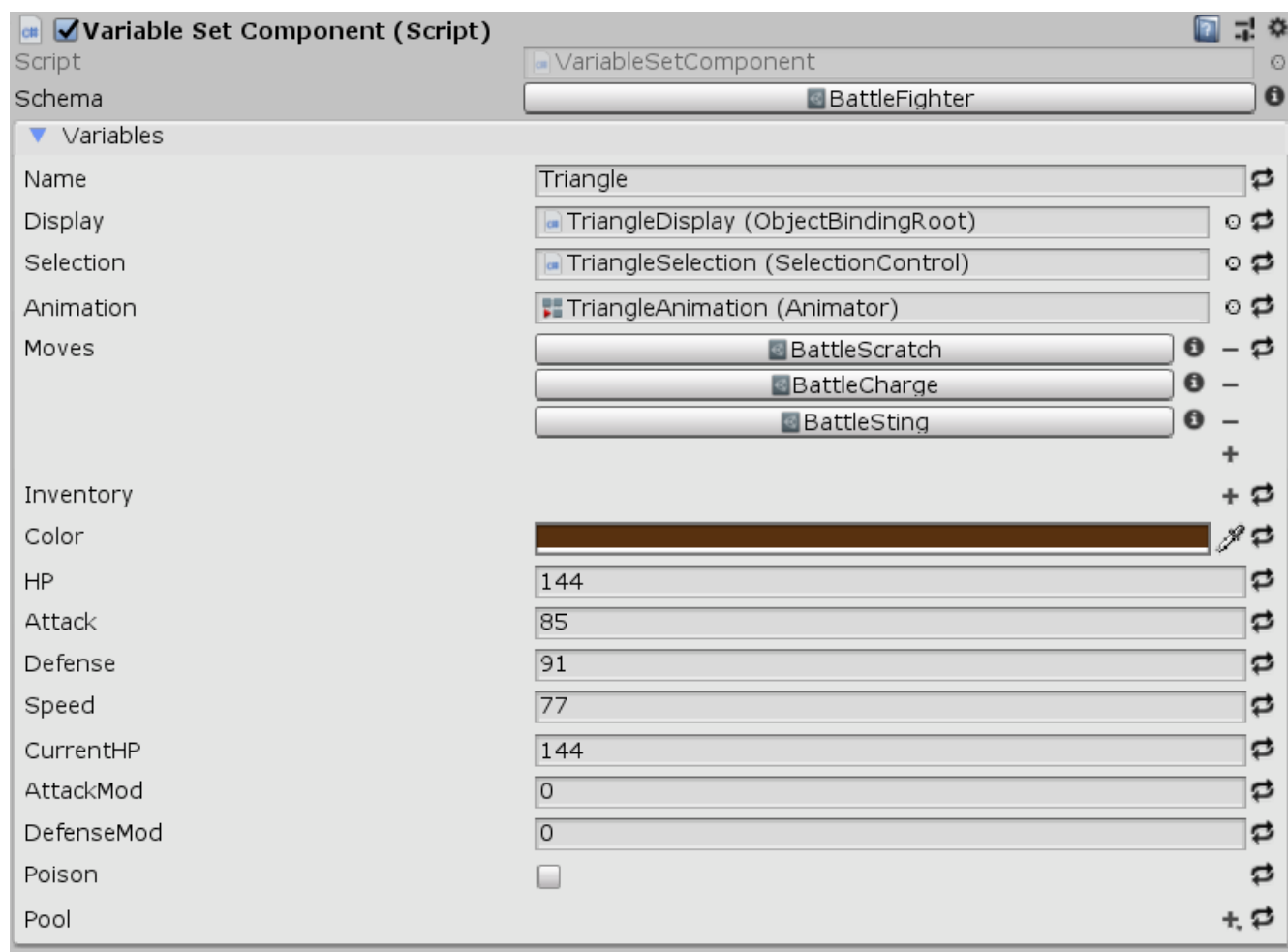| Name	Description
Schema	The Variable Schema to use for this Variable Set
Variables	The list of Variables mapped from *Schema* and derived script classes

Variable Set Component

A Variable Set Component is a MonoBehaviour that adds a Variable Set to the object. This allows it to have a list of Variables that are defined by a reference to a Variable Schema. A derived class of Variable Set Component can use the MappedVariableAttribute to add its fields and properties to the *Variables* list in addition to those defined by *Schema*.

See Variables for more information.
See the *"Triangle"* and *"Hexagon"* objects in the *"Battle"* scene of the Battle project for an example usage. See the *"Player"* script in the Maze project for an example implementation that uses mapped variables.

Fields

Name	Description
Schema	The Variable Schema to use for this Variable Set
Variables	The list of Variables mapped from *Schema* and derived script classes

Wait Node

A Wait Node is an Instruction Graph Node that waits for an amount of time (scaled or realtime) before continuing on to the next node. Create a Wait Node in the **Create › Sequencing › Wait** menu of the Instruction Graph Window.

See Graphs for more information on instruction graphs.
See the *"BoardTakeTurn"* Instruction Graph in the BoardGame project for an example usage.

Fields

Name	Description
Time	A FloatVariableSource to the value of the amount of time to wait
Use Scaled Time	If `true`, use WaitForSeconds() - if `false`, WaitForSecondsRealtime()

Yield Node

A Yield Node is an Instruction Graph Node that yield for one frame before continuing on to the next node. This is useful for long running processes whose execution needs to happen across frames or for idling a graph until a certain condition is met in a Loop Node. Create a Yield Node in the **Create › Control Flow › Yield** menu of the Instruction Graph Window.

See Control Flow for more information.
See the *"BoardTakeTurn"* Instruction Graph in the BoardGame project for an example usage.