

Unity Composition Reference

PiRho Soft

AnimationClipVariableSource	1
Description	1
AnimationPlayer	2
Description	2
Public Properties	2
Public Methods	2
AnimationType	3
Description	3
Values	3
AssetLocation	4
Description	4
Values	4
AssignmentOperator	5
Description	5
Protected Methods	5
AudioClipVariableSource	6
Description	6
AudioPlayer	7
Description	7
Public Properties	7
Public Methods	7
AxisInput	8
Description	8
Public Fields	8
BarBinding	9
Description	9
Public Fields	9
BindingAnimationStatus	10
Description	10
Public Methods	11
BindingFormatter	12
Description	12
Public Fields	12
Public Methods	12
BindingRoot	13
Description	13
Public Fields	13
Public Properties	13
Public Methods	13
BoolVariableSource	14
Description	14

Constructors	14
BoundsVariableSource	15
Description	15
Constructors	15
BranchNode	16
Description	16
Public Fields	16
BreakNode	17
Description	17
ButtonGraphTrigger	18
Description	18
Public Fields	18
ButtonInput	19
Description	19
Public Fields	19
ButtonState	20
Description	20
Constructors	20
Public Fields	20
ButtonType	21
Description	21
Values	21
ClassMap	22
Description	22
Static Methods	22
ClassMap	23
Description	23
Public Methods	23
ClickGraphTrigger	24
Description	24
CollisionGraphTrigger	25
Description	25
Public Fields	25
CollisionNotifier	26
Description	26
Colors	27
Description	27
Static Fields	27
ColorVariableSource	29
Description	29
Constructors	29

Command	30
Description	30
Public Fields	30
Public Methods	30
CommandEvaluationException	31
Description	31
Static Methods	31
Constructors	32
Public Fields	32
CommentNode	33
Description	33
Public Fields	33
CompositionManager	34
Description	34
Static Fields	34
Public Properties	34
Public Methods	34
ConditionalNode	36
Description	36
Public Fields	36
ConnectionData	37
Description	37
ConstrainedStore	38
Description	38
Constructors	38
Public Properties	38
CreateGameObjectNode	39
Description	39
Public Fields	39
CreateInstructionGraphNodeMenuAttribute	40
Description	40
Constructors	40
CreateScriptableObjectNode	41
Description	41
Public Fields	41
CutoffTransition	42
Description	42
Protected Methods	42
DependentObjectList	43
Description	43
DestroyObjectNode	44

Description	44
Public Fields	44
DisableObjectNode.....	45
Description	45
Public Fields	45
DissolveTransition	46
Description	46
Public Fields	46
EnableBinding.....	47
Description	47
Public Fields	47
EnableGraphTrigger	48
Description	48
EnableObjectNode	49
Description	49
Public Fields	49
EnumVariableConstraint	50
Description	50
Public Fields	50
ExitNode	51
Description	51
Expression	52
Description	52
Public Properties	52
Public Methods.....	52
ExpressionBinding.....	54
Description	54
Public Fields	54
ExpressionCompilationResult.....	55
Description	55
Public Fields	55
ExpressionDisplayAttribute.....	56
Description	56
Public Fields	56
ExpressionEvaluationException.....	57
Description	57
Constructors	57
ExpressionLexer.....	58
Description	58
Static Methods.....	58
ExpressionNode	59

Description	59
Public Fields	59
ExpressionParseException	60
Description	60
Constructors	60
Public Fields	60
ExpressionParser	61
Description	61
Static Methods	61
Public Methods	61
ExpressionToken	63
Description	63
Public Fields	63
ExpressionTokenType	64
Description	64
Values	64
Fade Transition	66
Description	66
Public Fields	66
FloatVariableConstraint	67
Description	67
Public Fields	67
FloatVariableSource	68
Description	68
Constructors	68
FocusBindingRoot	69
Description	69
Public Fields	69
FormatType	70
Description	70
Values	70
GameObjectVariableSource	71
Description	71
GlobalBehaviour	72
Description	72
Static Properties	72
Static Methods	72
GraphTriggerBinding	73
Description	73
Public Fields	73
HideControlNode	74

Description	74
Public Fields	74
IAssignableOperation	75
Description	75
Public Methods	75
IClassMap	76
Description	76
Public Methods	76
ICollisionTrigger	77
Description	77
Public Methods	77
ICommand	78
Description	78
Public Methods	78
ICompletionNotifier	79
Description	79
Public Properties	79
ILoopNode	80
Description	80
ImageBinding	81
Description	81
Public Fields	81
Public Properties	81
ImageColorBinding	82
Description	82
Public Fields	82
Public Properties	82
InfixOperation	83
Description	83
Public Properties	83
Protected Fields	83
Protected Methods	83
InputHelper	84
Description	84
Static Methods	84
InputNode	87
Description	87
Public Fields	87
InputNodeButton	88
Description	88
Public Fields	88

InputNodeButtonList	89
Description	89
Instruction	90
Description	90
Public Fields	90
Public Properties	90
Public Methods	90
Protected Methods	91
InstructionCaller	92
Description	92
Public Properties	92
Public Methods	92
InstructionGraph	93
Description	93
Static Fields	93
Public Methods	93
Protected Methods	94
InstructionGraphNode	95
Description	95
Public Fields	95
Public Properties	95
Public Methods	95
InstructionGraphNodeDictionary	98
Description	98
InstructionGraphNodeList	99
Description	99
InstructionInput	100
Description	100
Public Fields	100
InstructionInputType	101
Description	101
Values	101
InstructionNode	102
Description	102
Public Fields	102
InstructionOutput	103
Description	103
Public Fields	103
InstructionOutputType	104
Description	104
Values	104

InstructionSource	105
Description	105
Values	105
InstructionStore	106
Description	106
Static Fields	106
Static Methods	106
Public Properties	106
Public Methods	107
InstructionTrigger	109
Description	109
Public Fields	109
Public Methods	109
Int2VariableSource	110
Description	110
Constructors	110
Int3VariableSource	111
Description	111
Constructors	111
IntBoundsVariableSource	112
Description	112
Constructors	112
InterfaceControl	113
Description	113
Public Fields	113
Public Properties	113
Public Methods	113
Protected Methods	113
IntRectVariableSource	114
Description	114
Constructors	114
IntVariableConstraint	115
Description	115
Public Fields	115
IntVariableSource	116
Description	116
Constructors	116
ISchemaOwner	117
Description	117
Public Properties	117
Public Methods	117

ISequenceNode	118
Description	118
IterateNode	119
Description	119
Public Fields	119
IVariableList	120
Description	120
Public Properties	120
Public Methods	120
IVariableListener	121
Description	121
Public Methods	121
IVariableReset	122
Description	122
Public Methods	122
IVariableStore	123
Description	123
Public Methods	123
ListAdapter	124
Description	124
Static Methods	124
ListBinding	125
Description	125
Public Fields	125
ListVariableConstraint	126
Description	126
Public Fields	126
ListVariableSource	127
Description	127
LoadSceneNode	128
Description	128
Public Fields	128
LogNode	129
Description	129
Public Fields	129
LoopNode	130
Description	130
Public Fields	130
MappedVariableAttribute	131
Description	131
Constructors	131

Public Properties	131
MappedVariableStore	132
Description	132
Public Properties	132
Public Methods	132
MaterialAnimation	133
Description	133
Public Fields	133
Public Properties	133
Protected Methods	133
Menu	134
Description	134
Public Fields	134
Public Properties	134
Public Methods	135
Protected Methods	135
MenuInput	136
Description	136
Public Fields	136
Public Methods	137
MenuInputPointerAction	139
Description	139
Values	139
MenuItem	140
Description	140
Public Fields	140
Public Properties	140
Public Methods	140
MenuItemTemplate	141
Description	141
Public Fields	141
Public Properties	141
Message	142
Description	142
Public Fields	142
Public Properties	142
Public Methods	142
MessageBinding	143
Description	143
Public Fields	143
MessageControl	144

Description	144
Public Fields	144
Public Properties	144
Public Methods	144
Protected Methods	144
MessageInput	145
Description	145
Public Fields	145
Protected Properties	145
MessageNode	146
Description	146
Public Fields	146
MockupConnection	147
Description	147
Public Fields	147
MockupConnectionList	148
Description	148
MockupGraph	149
Description	149
Public Fields	149
MockupNode	150
Description	150
Public Fields	150
NodeData	151
Description	151
NumberBinding	152
Description	152
Public Fields	152
NumberFormatType	153
Description	153
Values	153
ObjectBindingRoot	154
Description	154
Public Fields	154
ObjectPositioning	155
Description	155
Values	155
ObjectPositioning	156
Description	156
Values	156
ObjectSource	157

Description	157
Values	157
ObjectVariableConstraint	158
Description	158
Public Fields	158
ObjectVariableSource	159
Description	159
Operation	160
Description	160
Public Methods	160
OperatorPrecedence	161
Description	161
Static Fields	161
Static Methods	162
Public Properties	162
Parameter	163
Description	163
Public Fields	163
ParameterList	164
Description	164
PixelateTransition	165
Description	165
Public Fields	165
PlayAnimationNode	166
Description	166
Public Fields	166
PlayAnimationStateNode	167
Description	167
Public Fields	167
PlaybackState	168
Description	168
Values	168
PlayEffect	169
Description	169
Public Fields	169
PlaySoundNode	170
Description	170
Public Fields	170
PlayTimelineNode	171
Description	171
Public Fields	171

PlayTransitionNode	172
Description	172
Public Fields	172
PrefixOperation	173
Description	173
Public Fields	173
Protected Methods	173
PrimaryAxis	174
Description	174
Values	174
QuaternionVariableSource	175
Description	175
Constructors	175
ReadOnlyStore	176
Description	176
RectVariableSource	177
Description	177
Constructors	177
ResetTagNode	178
Description	178
Public Fields	178
ResetVariableList	179
Description	179
ResetVariablesNode	180
Description	180
Public Fields	180
SceneReference	181
Description	181
Static Fields	181
Public Fields	181
Public Properties	181
Public Methods	181
SceneReferenceAttribute	182
Description	182
Public Fields	182
SceneSource	183
Description	183
Values	183
SceneSource	184
Description	184
Values	184

SceneVariableStore	185
Description	185
Public Methods.....	185
ScopedGraph	186
Description	186
Public Fields	186
SelectionControl	187
Description	187
Public Properties	187
Public Methods.....	188
Protected Methods.....	188
SelectionNode	189
Description	189
Public Fields	189
SelectionNodeItem.....	190
Description	190
Public Fields	190
SelectionNodeItemList	191
Description	191
SequenceNode.....	192
Description	192
Public Fields	192
SerializedDictionary	193
Description	193
Public Methods.....	193
SerializedList.....	194
Description	194
SetAnimationParameterNode.....	195
Description	195
Public Fields	195
SetBindingNode	196
Description	196
Public Fields	196
SetVariableResult	197
Description	197
Values	197
ShowControlNode	198
Description	198
Public Fields	198
ShuffleNode	199
Description	199

Public Fields	199
SimpleGraph	200
Description	200
Public Fields	200
SortConditionList	201
Description	201
SortNode	202
Description	202
Public Fields	202
SpriteBinding	203
Description	203
Public Fields	203
Public Properties	203
SpriteColorBinding	204
Description	204
Public Fields	204
Public Properties	204
StartGraphTrigger	205
Description	205
StopTransitionNode	206
Description	206
Public Fields	206
StoreVariableConstraint	207
Description	207
Public Fields	207
StoreVariableSource	208
Description	208
StringBinding	209
Description	209
Public Fields	209
Public Properties	209
Protected Methods	209
StringVariableConstraint	210
Description	210
Public Fields	210
StringVariableSource	211
Description	211
Constructors	211
TagList	212
Description	212
TextBinding	213

Description	213
Public Fields	213
TextColorBinding	214
Description	214
Public Fields	214
Public Properties	214
TextInputBinding	215
Description	215
Public Fields	215
Public Properties	215
TimeFormatType	216
Description	216
Values	216
TimelineVariableSource	217
Description	217
TimeScaleNode	218
Description	218
Public Fields	218
TransformNode	219
Description	219
Public Fields	219
Transition	221
Description	221
Public Fields	221
Public Methods	221
Protected Properties	221
Protected Methods	221
TransitionList	222
Description	222
TransitionManager	223
Description	223
Public Properties	223
Public Methods	223
TransitionPhase	224
Description	224
Values	224
TransitionRenderer	225
Description	225
TransitionVariableSource	226
Description	226
UnloadSceneNode	227

Description	227
Public Fields	227
UpdateBindingNode	228
Description	228
Public Fields	228
ValueDefinition	229
Description	229
Static Methods	229
Public Properties	230
Public Methods	230
ValueDefinitionList	231
Description	231
Variable	232
Description	232
Static Properties	232
Static Methods	232
Public Properties	232
VariableBinding	233
Description	233
Static Methods	233
Public Fields	233
Public Properties	234
Public Methods	234
Protected Methods	234
VariableConstraint	236
Description	236
Public Methods	236
VariableConstraintAttribute	237
Description	237
Constructors	237
VariableDefinition	238
Description	238
Public Fields	238
VariableDefinitionList	239
Description	239
VariableHandler	240
Description	240
Static Methods	240
VariableInitializerType	243
Description	243
Values	243

VariableLink	244
Description	244
Public Fields	244
VariableList	245
Description	245
Constructors	245
Public Properties	245
Public Methods	245
VariablePool	246
Description	246
Public Fields	246
Public Methods	246
VariablePoolAsset	247
Description	247
Public Fields	247
Public Methods	247
VariablePoolComponent	248
Description	248
Public Fields	248
Public Methods	248
VariableReference	249
Description	249
Static Fields	249
Public Properties	249
Public Methods	249
VariableSchema	251
Description	251
Public Fields	251
Public Properties	251
Public Methods	251
VariableSet	253
Description	253
Public Properties	253
Public Methods	253
VariableSetAsset	255
Description	255
Public Fields	255
Public Properties	255
Public Methods	255
VariableSetComponent	256
Description	256

Public Fields	256
Public Properties	256
Public Methods	256
VariableSource	257
Description	257
Public Fields	257
Public Methods	257
Protected Methods	257
VariableSource	258
Description	258
Public Fields	258
VariableSourceType	259
Description	259
Values	259
VariableStore	260
Description	260
Public Properties	260
Public Methods	260
Protected Methods	261
VariableType	262
Description	262
Values	262
VariableValue	264
Description	264
Static Properties	264
Static Methods	264
Public Properties	266
Public Methods	269
VariableValueSource	271
Description	271
Constructors	271
Public Fields	271
Vector2VariableSource	272
Description	272
Constructors	272
Vector3VariableSource	273
Description	273
Constructors	273
Vector4VariableSource	274
Description	274
Constructors	274

WaitNode	275
Description	275
Public Fields	275
WritableStore	276
Description	276
YieldNode	277
Description	277
Public Fields	277

AnimationClipVariableSource

PiRhoSoft.CompositionEngine.AnimationClipVariableSource : [VariableSource](#)<[AnimationClip](#)>

Description

A [VariableSource](#) for [AnimationClips](#).

AnimationPlayer

PiRhoSoft.CompositionEngine.AnimationPlayer : [MonoBehaviour](#), [ICompletionNotifier](#)

Description

Add this to any [GameObject](#) to provide an interface for playing [AnimationClips](#).

Public Properties

bool *IsComplete* (read only)

This will return true as soon as the animation has completed. If the animation has not yet started, it is not considered complete, so this will return false. If the animation is set to loop, this will always return false.

Public Methods

void **PlayAnimation**([AnimationClip](#) animation)

Plays *animation* and returns immediately.

IEnumerator **PlayAnimationAndWait**([AnimationClip](#) animation)

Plays *animation* and returns an enumerator so it can be run as or from a [coroutine](#). The enumerator will yield until *animation* has completed. If *animation* is set to loop, the enumerator will break immediately and an error will be printed. Call *PlayAnimation* instead to run looping animations.

void **Pause()**

Pauses playback of the currently running animation.

void **Unpause()**

Resumes playback of the currently running animation.

AnimationType

PiRhoSoft.CompositionEngine.AnimationType

Description

Defines the available options for the *AnimationMethod* of a [TransformNode](#).

Values

[AnimationType](#) *None*

The [Transform](#) will be updated immediately without any animation.

[AnimationType](#) *Speed*

Position, rotation, and scale will each animate according to an individually set number of units per second.

[AnimationType](#) *Duration*

The animation will take a set amount of time with position, rotation, and scale advancing linearly to their target.

AssetLocation

PiRhoSoft.UtilityEngine.AssetLocation

Description

Defines the settings available to *SaveLocation* on [SceneReferenceAttribute](#).

Values

AssetLocation *None*

A scene cannot be created via the [SceneReference](#).

AssetLocation *AssetRoot*

A scene created via the [SceneReference](#) will be placed at the root of the "Assets" directory.

AssetLocation *Selectable*

The location for a scene created via the [SceneReference](#) will be user specified with a "Save As" dialog.

AssignmentOperator

PiRhoSoft.CompositionEngine.AssignmentOperator : [InfixOperation](#)

Description

A base class for [InfixOperations](#) that perform assignment of [VariableValues](#).

Protected Methods

VariableValue Assign(**IVariableStore** variables, **VariableValue** value)

Call this from a subclass to assign *value* to a variable on *variables* based on the result of evaluating the *Left* operation.

AudioClipVariableSource

PiRhoSoft.CompositionEngine.AudioClipVariableSource : [VariableSource](#)<[AudioClip](#)>

Description

A [VariableSource](#) for [AudioClips](#).

AudioPlayer

PiRhoSoft.CompositionEngine.AudioPlayer : [MonoBehaviour](#), [ICompletionNotifier](#)

Description

Add this to any [GameObject](#) to provide an interface for playing [AudioClips](#).

Public Properties

bool *IsComplete (read only)*

This will return true as soon as the sound has completed. If the sound has not yet started, it is not considered complete, so this will return false. If the sound is set to loop, this will always return false.

Public Methods

void **PlaySound**([AudioClip](#) *sound*, **float** *volume*)

Plays *sound* at *volume* and returns immediately.

IEnumerator **PlaySoundAndWait**([AudioClip](#) *sound*, **float** *volume*)

Plays *sound* at *volume* and returns an enumerator so it can be run as or from a [coroutine](#). The enumerator will yield until *sound* has completed. If *sound* is set to loop, the enumerator will break immediately and an error will be printed. Call *PlaySound* instead to run looping sounds.

AxisInput

PiRhoSoft.CompositionEngine.AxisInput : [MonoBehaviour](#), [IEventSystemHandler](#), [IPointerDownHandler](#), [IPointerUpHandler](#)

Description

Add this to a [Graphic](#) or [Collider](#) to set the value of an axis on the [InputHelper](#) when the object is clicked or touched.

Public Fields

string *AxisName*

The name of the axis that is set to *AxisValue* when the object is clicked or touched.

float *AxisValue*

The value to set *AxisName* to when the object is clicked or touched.

BarBinding

PiRhoSoft.CompositionEngine.BarBinding : [VariableBinding](#)

Description

Add this to an [Image](#) to set the *fillAmount* and *color* based on two bound values.

Public Fields

VariableReference *AmountVariable*

The int or float variable indicating the amount the bar should be filled.

VariableReference *TotalVariable*

The int or float variable indicating the 'full' amount. [Image.fillAmount](#) is set to the result of $AmountVariable / TotalVariable$.

Gradient *FillColors*

The color to set [Image.color](#) to depending on the current fill amount.

float *Speed*

If this is greater than 0, the fill amount will animate when it changes. The value specifies the speed of the animation in percent per second. So, for example, a value of 0.1 would cause the bar to change its fill at a rate of 10% every second.

bool *UseScaledTime*

If this is set, *Speed* will be based on [scaled time](#), otherwise it will be based on [real time](#).

BindingAnimationStatus

PiRhoSoft.CompositionEngine.BindingAnimationStatus

Description

Used with [VariableBinding](#) to provide feedback for binding updates that are animated or otherwise completed asynchronously. Callers, such as [UpdateBindingNode](#), can pass an instance of this type to the [UpdateBinding](#) method and query it to determine when the binding has completed.

```
using PiRhoSoft.CompositionEngine
using UnityEngine;

namespace PiRhoSoft.CompositionExample
{
    public class UpdateBindingExample : MonoBehaviour
    {
        private BindingAnimationStatus _status = new BindingAnimationStatus();

        public override IEnumerator Run()
        {
            _status.Reset();

            VariableBinding.UpdateBinding(gameObject, string.Empty, _status);

            while (!_status.IsFinished())
                yield return null;
        }
    }
}
```

[VariableBindings](#), such as [BarBinding](#), use the *Increment* and *Decrement* methods to indicate when an animation has started and finished respectively.

```

using PiRhoSoft.CompositionEngine
using UnityEngine;

namespace PiRhoSoft.CompositionExample
{
    public class ExampleBinding : VariableBinding
    {
        private WaitForSeconds _wait = new WaitForSeconds(1);

        protected override void UpdateBinding(IVariableStore variables,
        BindingAnimationStatus status)
        {
            // update the binding
            StartCoroutine(Animate(status));
        }

        private IEnumerator Animate(BindingAnimationStatus status)
        {
            status.Increment();
            yield return _wait; // do animation stuff
            status.Decrement();
        }
    }
}

```

Public Methods

void Reset()

Call this method before passing a `BindingAnimationStatus` instance to a binding method to re-initialize it.

bool IsFinished()

Call this method to determine if all animations resulting from a bindings update have completed.

void Increment()

Call this method from a [VariableBinding](#) implementation to indicate the binding is starting an animation. This can be called multiple times if the binding is performing multiple animations. Each call to *Increment* should have a corresponding call to *Decrement* when the animation completes.

void Decrement()

Call this method from a [VariableBinding](#) implementation to indicate the binding has finished an animation. This should be called one time for each time *Increment* is called.

BindingFormatter

PiRhoSoft.CompositionEngine.BindingFormatter

Description

A type to use for fields on text [VariableBindings](#) to provide number formatting support.

Public Fields

string *Format*

The format of the resulting string. Use "{0}" to indicate the location in the string to insert the formatted number.

FormatType *Formatting*

Whether to format the number as a time or number, or skip formatting altogether.

TimeFormatType *TimeFormatting*

If *Formatting* is set to *Time*, specifies the format to use for the number.

NumberFormatType *NumberFormatting*

If *Formatting* is set to *Number*, specifies the format to use for the number.

string *ValueFormat*

If *Formatting* is set to *Time* and *TimeFormatting* is set to *Custom* or *Formatting* is set to *Number* and *NumberFormatting* is set to *Custom*, specifies the custom format to use. The syntax is the same as the .net [DateTime format strings](#) for *Formatting Time* and [numeric format strings](#) for *Formatting Number*.

Public Methods

string *GetFormattedString(float number)*

Returns *number* as a string based on the configured properties. For *Formatting Time* *number* is interpreted as a number of seconds.

string *GetFormattedString(int number)*

Returns *number* as a string based on the configured properties. For *Formatting Time* *number* is interpreted as a number of seconds.

BindingRoot

PiRhoSoft.CompositionEngine.BindingRoot : [MonoBehaviour](#), [IVariableStore](#)

Description

Add this to any [GameObject](#) to insert a [VariableValue](#) into the scene hierarchy that can be accessed by sibling or child [VariableBindings](#).

See the [Binding Roots](#) section in the [Bindings](#) topic for more information.

Public Fields

string *ValueName*

The name for [VariableBindings](#) to use to look up *Value*.

Public Properties

VariableValue *Value* (virtual)

The value to return when *ValueName* is looked up.

Public Methods

IList<string> **GetVariableNames()** (virtual)

Returns a list with *ValueName* as its only item.

VariableValue **GetVariable(string name)** (virtual)

If *name* is *ValueName*, returns *Value*, otherwise calls *GetVariable* on the next BindingRoot up in the object hierarchy. If this is the highest BindingRoot, *DefaultStore* on [CompositionManager](#) is used instead.

SetVariableResult **SetVariable(string name, VariableValue value)** (virtual)

If *name* is *ValueName*, returns *ReadOnly*, otherwise calls *SetVariable* on the next BindingRoot up in the object hierarchy. If this is the highest BindingRoot, *DefaultStore* on [CompositionManager](#) is used instead.

BoolVariableSource

PiRhoSoft.CompositionEngine.BoolVariableSource : [VariableSource](#)<bool>

Description

A [VariableSource](#) for bools.

Constructors

BoolVariableSource(bool defaultValue)

Initializes *Value* to *defaultValue*

BoundsVariableSource

PiRhoSoft.CompositionEngine.BoundsVariableSource : [VariableSource](#)<[Bounds](#)>

Description

A [VariableSource](#) for [Bounds](#).

Constructors

BoundsVariableSource([Bounds](#) *defaultValue*)

Initializes *Value* to *defaultValue*

BranchNode

PiRhoSoft.CompositionEngine.BranchNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to run an [InstructionGraphNode](#) based on the result of an [Expression](#).

Public Fields

[Expression](#) *Switch*

The [Expression](#) to execute to determine which of the [InstructionGraphNodes](#) in *Outputs* to run.

[InstructionGraphNodeDictionary](#) *Outputs*

The set of [InstructionGraphNodes](#) to run depending on the result of *Switch*.

[InstructionGraphNode](#) *Default*

If the result of *Switch* is not found in *Outputs*, this [InstructionGraphNode](#) will be run.

BreakNode

PiRhoSoft.CompositionEngine.BreakNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to return execution to the closest parent [InstructionGraphNode](#) that is an [ILoopNode](#).

ButtonGraphTrigger

PiRhoSoft.CompositionEngine.ButtonGraphTrigger : [InstructionTrigger](#)

Description

Add this to any [GameObject](#) to run *Graph* (declared on [InstructionTrigger](#)) when a button is pressed.

Public Fields

string *Button*

The name of the button that is checked for presses. The name corresponds to those recognized by [InputHelper](#).

ButtonInput

PiRhoSoft.CompositionEngine.ButtonInput : [MonoBehaviour](#), [IEventSystemHandler](#), [IPointerDownHandler](#), [IPointerUpHandler](#)

Description

Add this to a [Graphic](#) or [Collider](#) to set the pressed state of a button on the [InputHelper](#) when the object is clicked or touched.

Public Fields

string *ButtonName*

The name of the button whose state will be changed when this object is clicked or released.

ButtonState

PiRhoSoft.UtilityEngine.ButtonState : ValueType

Description

Describes the current state of an input on the [InputHelper](#). This could be a virtual button press, mouse button, touch, key, controller button, or controller axis. The properties on this class are set when the state is queried from the [InputHelper](#) rather than updating automatically, so the state needs to be looked up every frame to get up to date results.

Constructors

ButtonState(bool pressed, bool held, bool released)

Initializes the state to specific values.

ButtonState(string button)

Initializes the state to the current state of *button* where *button* is a virtual button defined in the [InputManager](#).

ButtonState(KeyCode key)

Initializes the state to the current state of *key*.

ButtonState(int touch)

Initializes the state to the current state of the touch with index *touch*.

Public Fields

bool Pressed

If this is true, the queried input was not pressed on the previous frame and is on the current frame.

bool Held

If this is true, the queried input is currently pressed. This will be true on the first frame (i.e when *Pressed* is true).

bool Released

If this is true, the queried input is no longer pressed but it was pressed on the previous frame.

ButtonType

PiRhoSoft.CompositionEngine.ButtonType

Description

Defines the available input types for an [InputNodeButton](#) in an [InputNode](#).

Values

ButtonType *Axis*

The [InputNodeButton.Name](#) refers to the name of an axis on the [InputHelper](#).

ButtonType *Button*

The [InputNodeButton.Name](#) refers to the name of a button on the [InputHelper](#).

ButtonType *Key*

The [InputNodeButton.Key](#) is used instead of [InputNodeButton.Name](#).

ClassMap

PiRhoSoft.CompositionEngine.ClassMap

Description

Manages registration of [IClassMaps](#).

Static Methods

void Add<T>(ClassMap<T> map)

Adds *map* as the [IClassMap](#) for the type *T*.

bool Get(Type type, IClassMap map (out))

If the Type *type* has an [IClassMap](#) registered, sets *map* to that [IClassMap](#) and returns true. Otherwise, returns false.

ClassMap

PiRhoSoft.CompositionEngine.ClassMap<T> : [IClassMap](#)

Description

Derive from this class to implement an [IClassMap](#) for type *T*. The derived class should be registered with [ClassMap.Add](#) in a static constructor or a [RuntimeInitializeOnLoadMethod](#).

Public Methods

[IList<string>](#) **GetVariableNames()** (*abstract*)

This method should return the list of variable names that are accessible in *GetVariable*.

[VariableValue](#) **GetVariable(T obj, string name)** (*abstract*)

This method should return a [VariableValue](#) containing the value of the property *name* on *obj*.

[SetVariableResult](#) **SetVariable(T obj, string name, [VariableValue](#) value)** (*abstract*)

This method should set the property *name* on *obj* to *value*.

ClickGraphTrigger

PiRhoSoft.CompositionEngine.ClickGraphTrigger : [InstructionTrigger](#), [IEventSystemHandler](#), [IPointerDownHandler](#), [IPointerUpHandler](#)

Description

Add this to a [Graphic](#) or [Collider](#) to to run *Graph* (declared on [InstructionTrigger](#)) when the object is clicked or touched.

CollisionGraphTrigger

PiRhoSoft.CompositionEngine.CollisionGraphTrigger : [MonoBehaviour](#), [ICollisionTrigger](#)

Description

Add this to a [Collider](#) to to run an [InstructionGrap](#) when a [CollisionNotifier](#) informs this object that it has collided.

Public Fields

InstructionCaller *EnterGraph*

The [InstructionGraph](#) to run when a [CollisionNotifier](#) begins colliding with this object.

InstructionCaller *ExitGraph*

The [InstructionGraph](#) to run when a [CollisionNotifier](#) stops colliding with this object.

CollisionNotifier

PiRhoSoft.CompositionEngine.CollisionNotifier : [MonoBehaviour](#)

Description

Add this to a [Collider](#) to notify an [ICollisionTrigger](#) when this object has started or stopped colliding with it.

Colors

PiRhoSoft.CompositionEngine.Colors

Description

Defines several colors that can be used by [InstructionGraphNode.NodeColor](#) derivations to indicate the color of the node in the graph editor. Using an appropriate color from here can improve the consistency in the editor and make it easier to quickly identify the function of a node.

Static Fields

Color *Start*

The color of the entry point node.

Color *Default*

The color used for nodes that don't implement [InstructionGraphNode.NodeColor](#).

Color *ExecutionLight*

The color used for nodes that defer execution to other systems.

Color *ExecutionDark*

The color used for nodes that perform a specific execution process.

Color *Animation*

The color used for nodes that interact with Unity's animation systems.

Color *Sequence*

The color used for nodes that perform many actions in a sequence.

Color *Loop*

The color used for nodes that repeat an action many times.

Color *Branch*

The color used for nodes that select an action to perform based on some input.

Color *Break*

The color used for nodes that alter the control flow of the graph.

Color *Sequencing*

A color used for nodes that are used in making scripted sequences.

Color *SequencingLight*

A color used for nodes that are used in making scripted sequences.

Color *SequencingDark*

A color used for nodes that are used in making scripted sequences.

Color *Interface*

A color used for nodes that interact with the user interface.

Color *InterfaceLight*

A color used for nodes that interact with the user interface.

Color *InterfaceDark*

A color used for nodes that interact with the user interface.

Color *InterfaceCyan*

A color used for nodes that interact with the user interface.

Color *InterfaceTeal*

A color used for nodes that interact with the user interface.

ColorVariableSource

PiRhoSoft.CompositionEngine.ColorVariableSource : [VariableSource](#)<[Color](#)>

Description

A [VariableSource](#) for [colors](#).

Constructors

ColorVariableSource([Color](#) defaultValue)

Initializes *Value* to *defaultValue*

Command

PiRhoSoft.CompositionEngine.Command : [ScriptableObject](#), [ICommand](#)

Description

Defines an [Expression](#) that can be called from other [Expressions](#).

See the [Writing Custom Commands](#) topic for more information.

Public Fields

string *Name*

The name to use in an [Expression](#) to run this command.

ParameterList *Parameters*

The list of [CommandParameters](#) that should be passed to the command.

Expression *Expression*

The [Expression](#) that is evaluated when this command is called.

Public Methods

VariableValue **Evaluate**([IVariableStore](#) *variables*, **string** *name*, [List](#)<[Operation](#)> *parameters*)

Validates *parameters* against the types defined in *Parameters* and, if valid, evaluates *Expression*. The result of executing *Expression* is returned. If the *parameters* are not valid, a [CommandEvaluationException](#) will be thrown. If execution of *Expression* fails, an [ExpressionEvaluationException](#) will be thrown.

CommandEvaluationException

PiRhoSoft.CompositionEngine.CommandEvaluationException : [Exception](#)

Description

The [Exception](#) type that is thrown during execution of [Commands](#).

Static Methods

[CommandEvaluationException](#) WrongParameterCount(string *commandName*, int *got*, int *expected*)

Returns an exception that can be thrown to indicate the command *commandName* was passed an incorrect number of parameters (*got*) when an exact amount (*expected*) is expected.

[CommandEvaluationException](#) WrongParameterCount(string *commandName*, int *got*, int *expected1*, int *expected2*)

Returns an exception that can be thrown to indicate the command *commandName* was passed an incorrect number of parameters (*got*) when one of two amounts (*expected1* or *expected2*) were expected.

[CommandEvaluationException](#) WrongParameterRange(string *commandName*, int *got*, int *expectedMinimum*, int *expectedMaximum*)

Returns an exception that can be thrown to indicate the command *commandName* was passed a number of parameters (*got*) outside of an expected range (*expectedMinimum* and *expectedMaximum*)

[CommandEvaluationException](#) TooFewParameters(string *commandName*, int *got*, int *expected*)

Returns an exception that can be thrown to indicate the command *commandName* was passed fewer parameters (*got*) than expected (*expected*).

[CommandEvaluationException](#) TooManyParameters(string *commandName*, int *got*, int *expected*)

Returns an exception that can be thrown to indicate the command *commandName* was passed more parameters (*got*) than expected (*expected*).

[CommandEvaluationException](#) WrongParameterType(string *commandName*, int *index*, [VariableType](#) *got*, [VariableType](#) *expected*)

Returns an exception that can be thrown to indicate the command *commandName* was passed a parameter at index *index* with the type *got* instead of the type *expected*.

[CommandEvaluationException](#) WrongParameterType(string *commandName*, int *index*, [VariableType](#) *got*, [VariableType](#) *expected1*, [VariableType](#) *expected2*)

Returns an exception that can be thrown to indicate the command *commandName* was passed a parameter at index *index* with type *got* instead of either of the types *expected1* or *expected2*.

[CommandEvaluationException](#) WrongParameterType(string *commandName*, int *index*, [VariableType](#) *got*, [VariableType](#)[] *expected*)

Returns an exception that can be thrown to indicate the command *commandName* was passed a parameter at index *index* with type *got* instead of any of the types in *expected*.

Constructors

CommandEvaluationException(string command, string error)

Creates an exception indicating the command *command* failed with error *error*.

CommandEvaluationException(string command, string errorFormat, Object[] arguments)

Creates an exception indicating the command *command* failed with error built from *errorFormat* formatted with *arguments*.

Public Fields

string Command

The name of the [Command](#) that threw this exception.

CommentNode

PiRhoSoft.CompositionEngine.CommentNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to add notes in the editor. This [InstructionGraphNode](#) has no outputs or runtime functionality.

Public Fields

string *Comment*

The text of the comment that will be displayed directly in the [graph window](#).

CompositionManager

PiRhoSoft.CompositionEngine.CompositionManager : [GlobalBehaviour](#)<[CompositionManager](#)>

Description

Globally manages execution of [Instructions](#). A single instance of this [MonoBehaviour](#) will be created automatically and can be accessed from the static *Instance* property.

Static Fields

string *GlobalStoreName*

The name to use to access *GlobalStore* from *DefaultStore* or any [InstructionStore](#).

string *SceneStoreName*

The name to use to access *SceneStore* from *DefaultStore* or any [InstructionStore](#).

string *CommandFolder*

The name of the folder that any custom [Commands](#) are placed in. By default this is "Commands". All folders with this name that are inside a folder called "Resources" will be loaded when the *CompositionManager* is created.

bool *LogTracking*

When this is true, information gathered in editor builds about the execution of [Instructions](#) will be logged to the [console](#). This will include the number of enumerator iterations, the number of frames, and the amount of time it took to complete execution of each [Instruction](#). The <<topics/graphs-5.html,Watch Window> in the editor exposes this variable as a toggle.

Public Properties

[IVariableStore](#) *DefaultStore* (read only)

An [IVariableStore](#) that exposes *GlobalStore* under the name *GlobalStoreName* and *SceneStore* under the name *SceneStoreName*.

[VariableStore](#) *GlobalStore* (read only)

An [IVariableStore](#) that stores user defined values which can be arbitrarily added, changed, and removed.

[SceneVariableStore](#) *SceneStore* (read only)

An [IVariableStore](#) implementation that allows scene objects to be looked up by name.

Public Methods

void *RunInstruction*([Instruction](#) instruction, [VariableValue](#) context)

Runs an [Instruction](#), usually an [InstructionGraph](#) without setting any inputs other than *context* or reading any outputs.

void RunInstruction([InstructionCaller](#) *caller*, [IVariableStore](#) *store*, [VariableValue](#) *context*)

Runs an [Instruction](#), usually an [InstructionGraph](#), reading the inputs specified in *caller* from *store* to an [InstructionStore](#) that is passed to the instruction, and reading the outputs from that [InstructionStore](#) to *store* when *caller* has completed.

Read more about [Instruction](#) inputs and outputs in the [Instruction Store](#) topic.

ConditionalNode

PiRhoSoft.CompositionEngine.ConditionalNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to run an [InstructionGraphNode](#) based on the result of a conditional [Expression](#).

Public Fields

[InstructionGraphNode](#) *OnTrue*

If *Condition* evaluates to true, this node will run.

[InstructionGraphNode](#) *OnFalse*

If *Condition* evaluates to false, this node will run.

[Expression](#) *Condition*

The expression to execute to determine which [InstructionGraphNode](#) should run. The expression should return a `Bool`, otherwise an error will be logged.

ConnectionData

PiRhoSoft.CompositionEngine.ConnectionData

Description

Stores data about a connection between two [InstructionGraphNode](#)s. This is managed automatically by the editor and can be ignored.

ConstrainedStore

PiRhoSoft.CompositionEngine.ConstrainedStore : [WritableStore](#), [ISchemaOwner](#)

Description

Holds a set of [Variables](#) that are defined in a [VariableSchema](#).

Constructors

ConstrainedStore([VariableSchema](#) *schema*)

Adds the variables defined in *schema* to this store.

Public Properties

[VariableSchema](#) *Schema (read only)*

The [VariableSchema](#) that was used to initialize this store.

CreateGameObjectNode

PiRhoSoft.CompositionEngine.CreateGameObjectNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to create a [GameObject](#) from a [prefab](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[GameObjectVariableSource](#) *Prefab*

The [prefab](#) to use as a template for the [GameObject](#) that will be created.

[StringVariableSource](#) *ObjectName*

The name to assign to the newly created [GameObject](#).

[VariableReference](#) *ObjectVariable*

The variable to assign the newly created [GameObject](#) to.

[ObjectPositioning](#) *Positioning*

The way the value of *Position* and *Rotation* should be interpreted.

[VariableReference](#) *Object*

When *Positioning* is Relative, specifies the [GameObject](#) the created [GameObject](#) should be positioned relative to.

[VariableReference](#) *Parent*

When *Positioning* is Child, specifies the [GameObject](#) the created [GameObject](#) should be added to as a child.

[Vector3VariableSource](#) *Position*

The position at which to place the newly created [GameObject](#).

[Vector3VariableSource](#) *Rotation*

The rotation to set the newly created [GameObject](#) to.

CreateInstructionGraphNodeMenuAttribute

PiRhoSoft.CompositionEngine.CreateInstructionGraphNodeMenuAttribute : Attribute

Description

This attribute should be added to custom [InstructionGraphNode](#)s to add them to the create list in the [graph editor window](#).

Constructors

CreateInstructionGraphNodeMenuAttribute(string *menuName*, int *order*)

The name to use for this [InstructionGraphNode](#) in the menu. Submenus will be created for each section of *menuName* that precedes a backslash. *order* specifies the relative order of entries in the lowest submenu.

CreateScriptableObjectNode

PiRhoSoft.CompositionEngine.CreateScriptableObjectNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to create a [ScriptableObject](#) of the specified type.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

string *ScriptableObjectType*

The [AssemblyQualifiedName](#) of the type of object to create. This type should be a concrete type with a default constructor that is derived from [ScriptableObject](#).

[VariableReference](#) *ObjectVariable*

The variable to assign the newly created [ScriptableObject](#) to.

CutoffTransition

PiRhoSoft.CompositionEngine.CutoffTransition : [Transition](#)

Description

Provides a custom [Shader](#) with an interface to fade, distort, and dissolve the screen image over time using an input texture. [FadeTransition](#), [DissolveTransition](#), and [DistortTransition](#) derive from this class to provide specific [Transition](#) functionality.

Protected Methods

void SetTexture([Texture2D](#) texture)

Sets the *TransitionTexture* property of the material which is used to lookup the animation properties according to the description in the [manual](#).

void SetColor([Color](#) color)

Sets the *_Color* property of the material.

void SetCutoff(float cutoff)

Sets the *_Cutoff* property of the material which specifies the portion of the texture that is used as the mask based on the current elapsed time.

void SetFade(float fade)

Sets the *_Fade* property of the material which specifies the interpolated position between the color from the input texture and the value set to *_Color*.

void SetDistort(bool distort)

Sets the *_Distort* property which indicates whether or not the material should distort the input texture based on the R and G channels in *_TransitionTexture*.

void Setup() (*virtual*)

Override this in subclasses to set additional material properties. The base implementation sets *TransitionTexture* to [Texture2D.blackTexture](#), *color* to black, and *Distort* to false. *Cutoff* and *Fade* are set in [Process](#) to the percentage of [Duration](#) that has elapsed.

DependentObjectList

PiRhoSoft.CompositionEngine.DependentObjectList : [SerializedList](#)<[GameObject](#)>

Description

Used by [InterfaceControl](#) to store a list of [GameObjects](#).

DestroyObjectNode

PiRhoSoft.CompositionEngine.DestroyObjectNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to destroy a [GameObject](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Target*

The object to destroy. The object can be any [Object](#). If it is a [MonoBehaviour](#), the owning [GameObject](#) will be destroyed.

DisableObjectNode

PiRhoSoft.CompositionEngine.DisableObjectNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to deactivate a [GameObject](#) or disable a [Behaviour](#) or [Renderer](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Target*

The object to deactivate. If the object is a [GameObject](#) it will be deactivated, if it is a [Behaviour](#) it will be disabled, and if it is a [Renderer](#) it will be disabled (effectively made invisible).



To deactivate a [GameObject](#) from a [Component](#) reference use as `GameObject` (see the [Accessing Variables](#) topic).



If the object is already inactive or disabled there will be no effect.

DissolveTransition

PiRhoSoft.CompositionEngine.DissolveTransition : [CutoffTransition](#)

Description

Performs a dissolve effect to or from a solid color from or to the rendered scene using a custom texture or perlin noise.

Public Fields

Color *Color*

The [Color](#) to dissolve the screen to.

Texture2D *Texture*

The input [Texture](#) that gives the dissolve pattern. If this is null, a texture filled with perlin noise will be generated and used.

Vector2Int *TextureSize*

If *Texture* is null, the size of the [Texture](#) to generate.

float *NoiseScale*

If *Texture* is null, the scale value of the perlin noise generated as the [Texture](#).

EnableBinding

PiRhoSoft.CompositionEngine.EnableBinding : [VariableBinding](#)

Description

Add this to any [GameObject](#) to enable or disable a [GameObject](#), [Behaviour](#), or [Renderer](#) based on the result of an [Expression](#).

Public Fields

Object *Object*

The [GameObject](#), [Behaviour](#), or [Renderer](#) to enable or disable based on *Condition*.

Expression *Condition*

The [Expression](#) to evaluate when updating the binding. If this evaluates to true, *Object* will be enabled otherwise it will be disabled (if it is not already).

EnableGraphTrigger

PiRhoSoft.CompositionEngine.EnableGraphTrigger : [InstructionTrigger](#)

Description

Add this to any [GameObject](#) to to run an [InstructionGrap](#) when the object is enabled.

EnableObjectNode

PiRhoSoft.CompositionEngine.EnableObjectNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to activate a [GameObject](#) or enable a [Behaviour](#) or [Renderer](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Target*

The object to activate. If the object is a [GameObject](#) it will be activated, if it is a [Behaviour](#) it will be enabled, and if it is a [Renderer](#) it will be enabled (effectively made visible).



To activate a [GameObject](#) from a [Component](#) reference use as `GameObject` (see the [Accessing Variables](#) topic).



If the object is already active or enabled there will be no effect.

EnumVariableConstraint

PiRhoSoft.CompositionEngine.EnumVariableConstraint : [VariableConstraint](#)

Description

Specifies the [enum](#) type for [VariableValues](#) using this constraint.

Public Fields

Type *Type*

The enum type.

ExitNode

PiRhoSoft.CompositionEngine.ExitNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to force the currently running branch of the graph to exit and stop running.

Expression

PiRhoSoft.CompositionEngine.Expression

Description

Add this as a field on a class to provide an interface for specifying simple, repeatable operations in the editor. The full expression syntax and a guide on writing and using expressions can be found in the [Expressions topic](#).

Public Properties

ExpressionCompilationResult *CompilationResult (read only)*

This will hold the result of the most recent expression compilation. If *HasError* is true, this can be queried to retrieve more information about the error. That same information will be visible in the editor when viewing the expression, and any expressions that are loaded with an invalid statement will have this information logged.

Operation *LastOperation (read only)*

The last **Operation** that was evaluated when evaluating the expression. If an **ExpressionEvaluationException** is thrown, this will be the **Operation** that was being evaluated when the error occurred. If the evaluation is successful this will hold the last **Operation** in the expression.

bool *IsValid (read only)*

This will be true if *Statement* has been set and was parsed successfully.

bool *HasError (read only)*

This will be true if *Statement* has been set but failed to parse correctly.

string *Statement (read only)*

The statement containing the text of the expression.

Public Methods

ExpressionCompilationResult **SetStatement(string statement)**

Sets *Statement* to *statement* and attempts to parse it. The parse result is returned.

void **GetInputs(IList<VariableDefinition> inputs, string source)**

Analyzes the expression to determine the variables that are being accessed on the variable store identified with name *source* and adds them to *inputs*.

void **GetOutputs(IList<VariableDefinition> outputs, string source)**

Analyzes the expression to determine the variables that are being set on the variable store identified with name *source* and adds them to *outputs*.

VariableValue **Execute(Object context, IVariableStore variables)**

Evaluates the expression using *Evaluate* and catches any [ExpressionEvaluation](#) or [CommandEvaluation](#) exceptions that are thrown and logs them. *context* should be the object that owns the expression and is passed along to the log.

VariableValue Execute([Object](#) context, [IVariableStore](#) variables, [VariableType](#) expectedType)

Evaluates the expression using *Evaluate* and catches any [ExpressionEvaluation](#) or [CommandEvaluation](#) exceptions that are thrown and logs them. Additionally, the result is checked to ensure it has the [VariableType](#) *expectedType* and an error is logged if it does not. *context* should be the object that owns the expression and is passed along to the log.

VariableValue Evaluate([IVariableStore](#) variables)

Evaluates the expression using *variables* as the root store for resolving [VariableReferences](#). The return value is the result of the last statement in the expression. If an error is encountered an [ExpressionEvaluation](#) or [CommandEvaluation](#) exception will be thrown.

ExpressionBinding

PiRhoSoft.CompositionEngine.ExpressionBinding : [StringBinding](#)

Description

Add this to a [TextMeshPro](#) to set the text to the result of an [Expression](#).

Public Fields

[BindingFormatter](#) *Formatting*

Specifies how the result of *Expression* should be formatted. This is only relevant if *Expression* results in an Int or Float [Variable](#).

[Expression](#) *Expression*

The *Expression* to evaluate when the binding is updated. The result will be applied to the sibling [TextMeshPro](#) component. If the result is an Int or Float it will be formatted according to *Formatting*. If it is a string, it will be used directly. If it is any other type, ToString() will be used.

ExpressionCompilationResult

PiRhoSoft.CompositionEngine.ExpressionCompilationResult

Description

Stores the results of compiling an [Expression](#).

Public Fields

bool *Success*

true if the [Expression](#) was compiled successfully.

int *Location*

If compilation failed, the index in the source text where the error was encountered.

string *Token*

If compilation failed, the text of the token in the source text where the error was encountered.

string *Message*

If compilation failed, a message giving details about why it failed.

ExpressionDisplayAttribute

PiRhoSoft.CompositionEngine.ExpressionDisplayAttribute : [PropertyAttribute](#)

Description

Apply this to an [Expression](#) field to customize the way the editor displays the [Expression](#). If an [Expression](#) is not given this attribute, it is interpreted as if all the following properties have been set to their default value.

Public Fields

bool *Foldout*

If this is true the expression will be expandable and collapsable with a foldout. The default is false.

bool *FullWidth*

If this is true the text area for the [Expression](#) will appear beneath its label and expanded to the full width of the inspector. Otherwise it will appear to the right of its label. The default is true.

int *MinimumLines*

This specifies the minimum number of lines that will be shown in the text area regardless of the length of the [Expression](#). The default is 2.

int *MaximumLines*

This specifies the number of lines the text area will grow to as the [Expression](#) gets longer before using a scroll bar. The default is 8.

ExpressionEvaluationException

PiRhoSoft.CompositionEngine.ExpressionEvaluationException : Exception

Description

The exception type that is thrown when the evaluation of an [Expression](#) fails.

Constructors

ExpressionEvaluationException(string error)

Specifies the message that gives more information about why evaluation failed.

ExpressionEvaluationException(string errorFormat, Object[] arguments)

Specifies the message that gives more information about why evaluation failed by formatting *errorFormat* with *arguments*.

ExpressionLexer

PiRhoSoft.CompositionEngine.ExpressionLexer

Description

Converts [Expression](#) statements into a series of tokens for processing by the [ExpressionParser](#). [Expression](#) handles this process automatically.

Static Methods

List<[ExpressionToken](#)> Tokenize(string input)

Converts *input* into a list of [tokens](#) that can then be processed by the [ExpressionParser](#). This method will always successfully convert *input*, with any unknown character sequences being given [ExpressionTokenType](#) Unknown. It is the responsibility of the [ExpressionParser](#) to report these errors as well as errors for invalid token sequences.

void AddConstant(string text, [VariableValue](#) value)

Adds the string *text* as a sequence of characters the lexer should identify as a Constant [token](#) that is always [parser](#) as [VariableValue](#) *value*.

void AddKeyword(string text)

Adds the string *text* as a sequence of characters the lexer should identify as an Operator [token](#). The [parser](#) should be given an operator with *symbol text* using [AddPrefixOperator](#) or [AddInfixOperator](#) to define the functionality for the keyword.

[VariableValue](#) GetConstant(string text)

Returns the [VariableValue](#) that was assigned to *text* using *AddConstant*.

ExpressionNode

PiRhoSoft.CompositionEngine.ExpressionNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to run an [Expression](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[Expression](#) *Expression*

The [Expression](#) to run. The result of the [Expression](#) is ignored.

ExpressionParseException

PiRhoSoft.CompositionEngine.ExpressionParseException : Exception

Description

The exception type thrown by the [ExpressionParser](#) or [Operations](#) when encountering an error during parsing.

Constructors

ExpressionParseException([ExpressionToken](#) token, string error)

Specifies *token* as the token that caused the error and sets the exception message to *error*.

ExpressionParseException([ExpressionToken](#) token, string errorFormat, Object[] arguments)

Specifies *token* as the token that caused the error and sets the exception message to *errorFormat* formatted with *arguments*.

Public Fields

[ExpressionToken](#) Token

The token at which the error was encountered.

ExpressionParser

PiRhoSoft.CompositionEngine.ExpressionParser

Description

Converts a sequence of [ExpressionTokens](#) as interpreted by the [ExpressionLexer](#) into an executable [Operation](#).

Static Methods

List<Operation> Parse(string input, List<ExpressionToken> tokens)

Performs the conversion. *input* is the entire source text that was sent to the [ExpressionLexer](#) for use in printing friendly error messages. *tokens* is the set of tokens interpreted by the [ExpressionLexer](#). If the tokens cannot be parsed, an [ExpressionParseException](#) will be thrown.

void AddCommand(string name, ICommand command)

Associates the name *name* with *command*. When this name is encountered in an [Expression command](#) will be executed. If a [Command](#) has already been added with name *name* an error will be logged.

void RemoveCommand(string name)

Removes the [Command](#) registered with name *name*. If no [Command](#) has been added with name *name* an error will be logged.

ICommand GetCommand(string name)

Returns the [Command](#) that was registered with name *name*, or null if no command has been registered with that name.

void AddPrefixOperator<OperatorType>(string symbol)

Associates the string *symbol* with the [PrefixOperation OperatorType](#). If a [PrefixOperation](#) has already been registered with *symbol* an error will be logged.

void AddInfixOperator<OperatorType>(string symbol, OperatorPrecedence precedence)

Associates the string *symbol* with the [InfixOperation OperatorType](#). If an [PrefixOperation](#) has already been registered with *symbol* an error will be logged.



A [PrefixOperation](#) and [InfixOperation](#) can be added with the same symbol.

Public Methods

Operation ParseLeft(OperatorPrecedence precedence)

This should only be called from [Operation.Parse](#) implementations to parse the next sequence of tokens with the given *precedence* using left associativity.

Operation ParseRight(OperatorPrecedence precedence)

This should only be called from [Operation.Parse](#) implementations to parse the next sequence of

tokens with the given *precedence* using right associativity.

string GetText([ExpressionToken](#) token)

Gets the text that *token* was parsed from.

bool HasText([ExpressionToken](#) token, string text)

Returns true if *token* has the text *text*

bool HasToken([ExpressionTokenType](#) type)

Returns true if the next token in the current parse has type *type*.

void SkipToken([ExpressionTokenType](#) type, string expected)

Skips the next token in the current parse. If the next token does not have type *type*, an [ExpressionParseException](#) will be thrown. *expected* is the text that was expected at the current location and is used to provide a friendlier error message.

ExpressionToken

PiRhoSoft.CompositionEngine.ExpressionToken

Description

Stores the data for a sequence of characters as identified by the [ExpressionLexer](#).

Public Fields

ExpressionTokenType *Type*

Specifies how the [ExpressionParser](#) should interpret this token.

int *Location*

The index in the source text that identifies the beginning of this token.

int *Start*

The index in the source text that identifies the beginning of the relevant text of this token. As opposed to *Location* this will not include any introductory characters and instead identifies the index relevant to the [ExpressionParser](#).

int *End*

The index in the source text that identifies the beginning of the relevant text of this token. Similarly to *Start*, this will not include any trailing characters in the token that are not relevant to the [ExpressionParser](#).

ExpressionTokenType

PiRhoSoft.CompositionEngine.ExpressionTokenType

Description

Specifies the set of [ExpressionTokens](#) the [ExpressionLexer](#) and [ExpressionParser](#) understand.

Values

[ExpressionTokenType](#) *Sentinel*

Separates for two distinct statements. This is either ; or a line break with multiple of these concatenated into a single token.

[ExpressionTokenType](#) *Constant*

A [VariableValue](#) that has been added to the [ExpressionLexer](#) with *AddConstant*.

[ExpressionTokenType](#) *Type*

A [VariableType](#) for use with testing the type of values.

[ExpressionTokenType](#) *Int*

A literal value that should be interpreted as an int. An int is any continuous sequence of digits.

[ExpressionTokenType](#) *Float*

A literal value that should be interpreted as a float. A float is any continuous sequence of digits that includes a decimal point.

[ExpressionTokenType](#) *String*

A literal value that should be interpreted as a string. A string is a sequence of characters bounded by double quotes (")

[ExpressionTokenType](#) *Color*

A literal value that should be interpreted as a color. A color is a sequence of 6 digits following a hash (#)

[ExpressionTokenType](#) *Identifier*

A name that is used to look up variable values. Identifiers can be any sequence of letters, numbers, spaces, or underscores beginning with a letter or underscore.

[ExpressionTokenType](#) *Command*

A name that is used to look up a [Command](#) that has been registered with the [ExpressionParser](#) using *AddCommand*. A command is an *Identifier* that is followed by an opening paren (().

[ExpressionTokenType](#) *Operator*

An operator that is used to look up a [PrefixOperation](#) or [InfixOperation](#) that has been registered with the [ExpressionParser](#) using *AddPrefixOperator* or *AddInfixOperator*. Valid operator characters are any of +-!^*/%<>=&|?. in any sequence and any character sequence that has been

registered with the [ExpressionLexer](#) using *AddKeyword*.

ExpressionTokenType *StartLookup*

Indicates the following tokens should be interpreted as part of a [variable lookup](#). This is the [character.

ExpressionTokenType *EndLookup*

Indicates the following tokens are no longer part of a [variable lookup](#). This is the] character.

ExpressionTokenType *StartGroup*

Indicates the following tokens should be isolated and evaluated as a group, just as would be done in a math expression. This is the (character.

ExpressionTokenType *EndGroup*

Ends a group that was started with a *StartGroup* token or a command that was started with a *Command* token. This is the) character.

ExpressionTokenType *Separator*

Seperates parameters in a *Command* token. This is the , character.

ExpressionTokenType *Alternation*

This is the character used as the separator for the true and false statements of a ternary expression. This is the : character.

ExpressionTokenType *Unknown*

Any token that does not meet the requirements for one of the preceding types will be given this type.

Fade Transition

PiRhoSoft.CompositionEngine.FadeTransition : [CutoffTransition](#)

Description

Performs a linear fade to or from a solid color.

Public Fields

Color *Color*

The color to fade in to or out from.

FloatVariableConstraint

PiRhoSoft.CompositionEngine.FloatVariableConstraint : [VariableConstraint](#)

Description

A [VariableConstraint](#) for Float [VariableValues](#) that restricts the value to a range.

Public Fields

float *Minimum*

The smallest value allowed.

float *Maximum*

The largest value allowed.

FloatVariableSource

PiRhoSoft.CompositionEngine.FloatVariableSource : [VariableSource](#)<float>

Description

A [VariableSource](#) for Float [VariableValues](#)

Constructors

FloatVariableSource(float defaultValue)

Initializes the source to *Type* Value with *Value* _defaultValue.

FocusBindingRoot

PiRhoSoft.CompositionEngine.FocusBindingRoot : [BindingRoot](#)

Description

Add this to any [GameObject](#) to add the focused item of a [Menu](#) to the [BindingRoot](#) hierarchy.

Public Fields

[Menu](#) *Menu*

The [Menu](#) to query for the currently focused item, which will then be used as *Value* for this [BindingRoot](#).

FormatType

PiRhoSoft.CompositionEngine.FormatType

Description

Defines the types available to set for the *Format* of a [BindingFormatter](#).

Values

FormatType *None*

The [BindingFormatter](#) will not apply any formatting and instead return the result of ToString directly.

FormatType *Time*

The [BindingFormatter](#) will apply formatting while interpreting the input value as a [TimeSpan](#)

FormatType *Number*

The [BindingFormatter](#) will apply formatting while interpreting the input value as a [number](#)

GameObjectVariableSource

PiRhoSoft.CompositionEngine.GameObjectVariableSource : [VariableSource](#)<[GameObject](#)>

Description

A [VariableSource](#) for Object [VariableValues](#) that hold [GameObjects](#).

GlobalBehaviour

PiRhoSoft.UtilityEngine.GlobalBehaviour<T> : [MonoBehaviour](#)

Description

Derive from this class to ensure only a single instance of the derived class will be created. The instance will be created on demand (when *Instance* is first accessed) and should not be added to a scene.

Pass the derived class itself as *T*: `public class CompositionManager : GlobalBehaviour<CompositionManager> {}`

Static Properties

bool *Exists* (*read only*)

Checks if the instance has been created.

T *Instance* (*read only*)

The instance of the class. This will create the instance if it hasn't been already.

Static Methods

void *Destroy()*

Destroys the created instance.

GraphTriggerBinding

PiRhoSoft.CompositionEngine.GraphTriggerBinding : [VariableBinding](#)

Description

Add this to any [GameObject](#) to run an [InstructionGraph](#) when a variable value changes.

Public Fields

InstructionCaller *Graph*

The [graph](#) to run when the value referenced by *Variable* changes.

VariableReference *Variable*

The variable to watch for changes.

HideControlNode

PiRhoSoft.CompositionEngine.HideControlNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to hide an [InterfaceControl](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Control*

The [InterfaceControl](#) to hide.

IAssignableOperation

PiRhoSoft.CompositionEngine.IAssignableOperation

Description

Implement this interface in an [Operation](#) subclass to support assigning values when the [Operation](#) appears on the left hand side of an assignment.

Public Methods

SetVariableResult **SetValue**(**IVariableStore** *variables*, **VariableValue** *value*) (*abstract*)

Implement this method to assign *value* to a variable in *variables* and return the result.

IClassMap

PiRhoSoft.CompositionEngine.IClassMap

Description

Implement this interface to expose properties as [Variables](#) in a class that is not an [IVariableStore](#). Generally deriving from [ClassMap](#) is a better option than implementing this interface directly.

Public Methods

[IList<string>](#) GetVariableNames() (*abstract*)

This method should return the list of variable names that are accessible in *GetVariable*.

[VariableValue](#) GetVariable(object obj, string name) (*abstract*)

This method should return a [VariableValue](#) containing the value of the property *name* on *obj*.

[SetVariableResult](#) SetVariable(Object obj, string name, [VariableValue](#) value) (*abstract*)

This method should set the property *name* on *obj* to *value*.

ICollisionTrigger

PiRhoSoft.CompositionEngine.ICollisionTrigger

Description

Implement this interface on a [MonoBehaviour](#) to allow an object to respond to collisions between itself and a [CollisionNotifier](#).

Public Methods

void Enter() (*abstract*)

Called by a [CollisionNotifier](#) to indicate a collision has started.

void Exit() (*abstract*)

Called by a [CollisionNotifier](#) to indicate a collision has ended.

ICommand

PiRhoSoft.CompositionEngine.ICommand

Description

Implement this interface to create a custom command that can be added to the [ExpressionParser](#) and ultimately called from [Expressions](#).

Public Methods

VariableValue Evaluate(**IVariableStore** variables, **string** name, **List<Operation>** parameters)
(abstract)

Implement this method to perform the command's function. *variables* contains the [IVariableStore](#) that should be passed to each [Operation](#) in *parameters* when evaluating them as well as to look up any custom variables. *name* is the name that was used to call this command and *parameters* is the parsed expressions that were passed to the command.

Throw a [CommandEvaluationException](#) to indicate any errors in execution.

ICompletionNotifier

PiRhoSoft.CompositionEngine.ICompletionNotifier

Description

Implement this interface in a [MonoBehaviour](#) subclass to add support for using the behaviour as an *Effect* for a [PlayEffectNode](#) in an [InstructionGraph](#).

Public Properties

bool *IsComplete* (read only) (abstract)

This property should return true when the effect has completed.

ILoopNode

PiRhoSoft.CompositionEngine.ILoopNode

Description

Implement this interface in an [InstructionGraphNode](#) subclass to inform an [InstructionGraph](#) that the node should be run repeatedly. The graph will continue to run the node until the node does not call [InstructionGraph.GoTo](#) (or calls `GoTo(null)`) or a [BreakNode](#) is encountered.

ImageBinding

PiRhoSoft.CompositionEngine.ImageBinding : [VariableBinding](#)

Description

Add this to an [Image](#) to bind *sprite* to a variable.

Public Fields

[VariableReference](#) *Variable*

The [Sprite](#) that should be set on *Image*.

Public Properties

[Image](#) *Image (read only)*

The [Image](#) that will be updated.

ImageColorBinding

PiRhoSoft.CompositionEngine.ImageColorBinding : [VariableBinding](#)

Description

Add this to an [Image](#) to bind *color* to a variable.

Public Fields

[VariableReference](#) *Variable*

The [Color](#) that should be set on *Image*.

Public Properties

[Image](#) *Image (read only)*

The [Image](#) that will be updated.

InfixOperation

PiRhoSoft.CompositionEngine.InfixOperation : [Operation](#)

Description

Derive from this class to implement an [Operations](#) that has a left and right side.

Public Properties

OperatorPrecedence *Precedence (read only) (abstract)*

The [OperatorPrecedence](#) of the operation relative to other operations.

Protected Fields

Operation *Left*

The [Operation](#) that makes up the left hand side.

string *Symbol*

The symbol identifying this [Operation](#).

Operation *Right*

The [Operation](#) that makes up the right hand side.

Protected Methods

ExpressionEvaluationException **TypeMismatch**(**VariableType** *left*, **VariableType** *right*)

Creates an exception to be thrown by the caller indicating the [Operation](#) cannot operate on values with types *left* and *right*.

InputHelper

PiRhoSoft.UtilityEngine.InputHelper

Description

Provides a static interface for accessing user input in a consistent manner independent of how the input was set up.

Static Methods

void LateUpdate()

This is called by the [CompositionManager](#) to update the input state and can be ignored.

void SetButton(string *button*, bool *down*)

Performs a virtual press (*down* = true) or release (*down* = false) of a button named *button*. *button* can be any string, including those defined in [InputManager](#). This is used by [ButtonInput](#) to provide on screen control support.

void RemoveButton(string *button*)

Removes the tracking data for the virtual button named *button*.

void SetAxis(string *axis*, float *value*)

Sets the value of a virtual access named *axis* to *value*. *axis* can be any string, including those defined in [InputManager](#). This is used by [AxisInput](#) to provide on screen control support.

void RemoveAxis(string *axis*)

Removes the tracking data for the virtual axis named *axis*.

[ButtonState](#) GetKeyState([KeyCode](#) *key*)

Returns the [ButtonState](#) for the keyboard key *key*.

bool GetKeyDown([KeyCode](#) *key*)

Returns true if the keyboard key *key* is currently pressed.

bool GetWasKeyPressed([KeyCode](#) *key*)

Returns true if the keyboard key *key* is currently pressed and was not pressed on the previous frame.

bool GetWasKeyReleased([KeyCode](#) *key*)

Returns true if the keyboard key *key* is not currently pressed and was pressed on the previous frame.

bool GetIsPointerDown()

Returns true if the left mouse button is currently pressed or touch with index 0 is touching the screen

bool GetWasPointerPressed()

Returns true if the left mouse button is currently pressed or touch with index 0 is currently touching the screen but neither was happening on the previous frame.

bool GetWasPointerReleased()

Returns true if the left mouse button or touch with index 0 is not currently pressed and was pressed on the previous frame.

ButtonState GetTouchState(int touch)

Returns the [ButtonState](#) for the touch with index *touch*.

Nullable<TouchPhase> GetTouchPhase(int touch)

Returns the [TouchPhase](#) for the touch with index *touch*. If the touch is not currently tracked, null will be returned.

bool GetIsTouching(int touch)

Returns true if the touch with index *touch* is currently touching the screen.

bool GetTouchStarted(int touch)

Returns true if the touch with index *touch* is currently touching the screen but was not on the previous frame.

bool GetTouchEnded(int touch)

Returns true if the touch with index *touch* is not currently touching the screen but was on the previous frame.

bool IsButtonAvailable(string button)

Returns true if *button* exists either as a virtual button created with *SetButton* or in the [InputManager](#).

ButtonState GetButtonState(string button)

Returns the [ButtonState](#) of the virtual button or [InputManager](#) axis named *button*. If *button* is defined as both, the virtual button state will be used.

bool GetButtonDown(string button)

Returns true if the virtual button or [InputManager](#) axis named *button* is pressed.

bool GetWasButtonPressed(string button)

Returns true if the virtual button or [InputManager](#) axis named *button* was pressed this frame.

bool GetWasButtonReleased(string button)

Returns true if the virtual button or [InputManager](#) axis named *button* was released this frame.

float GetAxis(string axis)

Returns the value of the virtual axis or [InputManager](#) axis named *axis*. If *axis* is defined in both places, the value with the greater magnitude will be returned.

ButtonState GetAxisState(string axis, float magnitude)

Returns the [ButtonState](#) of the virtual axis or [InputManager](#) axis named *axis* according to the same rules defined in the following 3 methods.

bool GetAxisDown(string *axis*, float *magnitude*)

Returns true if the value of the virtual axis or [InputManager](#) axis named *axis* matches the sign of and is farther from 0 than *magnitude*. If *axis* is defined in both places, the value with the greater magnitude will be used.

bool GetWasAxisPressed(string *axis*, float *magnitude*)

Returns true if the value of the virtual axis or [InputManager](#) axis named *axis* matches the sign of and is farther from 0 than *magnitude* and wasn't the previous frame. If *axis* is defined in both places, the value with the greater magnitude will be used.

bool GetWasAxisReleased(string *axis*, float *magnitude*)

Returns true if the value of the virtual axis or [InputManager](#) axis named *axis* matched the sign of and was farther from 0 than *magnitude* on the previous frame and isn't this frame. If *axis* is defined in both places, the value with the greater magnitude will be used.

InputNode

PiRhoSoft.CompositionEngine.InputNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to wait for user input then run an [InstructionGraphNode](#) depending on the input.

Public Fields

[InputNodeButtonList](#) *Buttons*

The list of [InputNodeButtons](#) that indicate the [InstructionGraphNode](#) to advance to when a particular input is triggered.

InputNodeButton

PiRhoSoft.CompositionEngine.InputNodeButton

Description

Holds data for a button or key in an [InputNode](#).

Public Fields

ButtonType *Type*

The type of input to use for this button.

string *Name*

The name of the button if *Type* is Button or the name of the axis if *Type* is Axis as defined by the [InputHelper](#).

float *Value*

If *Type* is Axis, the minimum amount of the axis value for it to be considered pressed. If this value is negative, the axis value must be more negative than this value.

KeyCode *Key*

If *Type* is Key, the keyboard key for the button.

InstructionGraphNode *OnSelected*

The [InstructionGraphNode](#) to go to when this input is triggered.

InputNodeButtonList

PiRhoSoft.CompositionEngine.InputNodeButtonList : [SerializedList](#)<[InputNodeButton](#)>

Description

The serializable list of [InputNodeButtons](#) in an [InputNode](#).

Instruction

PiRhoSoft.CompositionEngine.Instruction : [ScriptableObject](#)

Description

Implements the core functionality for an [InstructionGraph](#) or any other asset to allow it to be run by the [CompositionManager](#).

Public Fields

string *ContextName*

The name to assign to the variable that is passed as *context* to either of the [CompositionManager.Run](#) methods.

ValueDefinition *ContextDefinition*

The definition to use to validate the variable passed as *context* to either of the [CompositionManager.Run](#) methods. If the definition specifies a [MonoBehaviour](#) type and the *context* value is not that behaviour type, it will be converted by attempting to look up a sibling behaviour.

VariableDefinitionList *Inputs*

The list of definitions for the input variables this instruction expects to be set when called from an [InstructionCaller](#). This list will be automatically populated by the editor and each definition can optionally be set to constrain the corresponding input. If the definition is set, the input will be validated at runtime to ensure the correct data was passed, with a message being logged if it is not.

VariableDefinitionList *Outputs*

The list of definitions for the output variables indicating the values this instruction will set for an [InstructionCaller](#) when it has completed. The outputs are not validated because it is not required that they are set by the instruction, but setting these will improve the editor experience.

Public Properties

IVariableStore *Variables (read only)*

The [InstructionStore](#) that was passed to the *Execute* method. This will be null if the instruction is not currently running.

bool *IsRunning (read only)*

This will return true while the *Execute* coroutine is running.

Public Methods

IEnumerator **Execute**(**VariableValue** context)

Executes the instruction. When inside an existing [coroutine](#) this can be called directly as part of

a `yield return` statement. When outside a coroutine, the [CompositionManager.RunInstruction](#) methods should be used.

IEnumerator Execute(InstructionStore variables)

Executes the instruction. When inside an existing [coroutine](#) this can be called directly as part of a `yield return` statement. When outside a coroutine, the [CompositionManager.RunInstruction](#) methods should be used.

void RefreshInputs()

Used by the editor to refresh the input list when necessary. This will happen automatically and can be ignored.

void RefreshOutputs()

Used by the editor to refresh the input list when necessary. This will happen automatically and can be ignored.

Protected Methods

void OnEnable() (virtual)

Performs important setup for the instruction. If overridden make sure to call the base implementation.

void OnDisable() (virtual)

Performs important teardown for the instruction. If overridden make sure to call the base implementation.

void GetInputs(IList<VariableDefinition> inputs) (virtual)

Implement this in subclasses to populate the *inputs* list with definitions for values the instruction expects to be available when called.

void GetOutputs(IList<VariableDefinition> outputs) (virtual)

Implement this in subclasses to populate the *outputs* list with definitions for values the instruction will set after it finishes running.

IEnumerator Run(InstructionStore variables) (abstract)

Implement this in subclasses to perform the function of the instruction.

InstructionCaller

PiRhoSoft.CompositionEngine.InstructionCaller

Description

Add this as a field on a [MonoBehaviour](#) or [ScriptableObject](#) to serve as a bridge between code and an [InstructionGraph](#). This class will automatically manage configuring and applying input and output values to the [InstructionGraph](#) and enable full editor support. Read the [Running Graphs From Script](#) topic for more information.

Public Properties

Instruction *Instruction*

The instruction, usually an [InstructionGraph](#), to run when this caller is executed.

IList<InstructionInput> *Inputs (read only)*

The list of [InstructionInputs](#) to add to the [InstructionStore](#) when running *Instruction*.

IList<InstructionOutput> *Outputs (read only)*

The list of [InstructionOutputs](#) to read from the [InstructionStore](#) after running *Instruction*.

bool *IsRunning (read only)*

This will return true when *Instruction* is being executed. [Instructions](#) that are already running cannot be run again until they have completed.

Public Methods

IEnumerator **Execute**([IVariableStore](#) store, [VariableValue](#) context)

Call this as a [Coroutine](#) or from another coroutine to run *Instruction*.

void **UpdateVariables()**

This is an editor support function.

VariableDefinition **GetInputDefinition**([InstructionInput](#) input)

This is an editor support function.

VariableDefinition **GetOutputDefinition**([InstructionOutput](#) output)

This is an editor support function.

InstructionGraph

PiRhoSoft.CompositionEngine.InstructionGraph : [Instruction](#)

Description

The main piece of the composition system, implementing all the functionality necessary to manage and execute a set of [InstructionGraphNode](#)s. Read the [graph topic](#) for a more thorough breakdown of creating and using graphs.

Static Fields

bool *IsDebugBreakEnabled*

Indicates the editor should pause graph execution when it encounters a breakpoint. This is on (true) by default but can be turned off in the graph editor window to disable all [node](#) breakpoints. The breakpoints are not removed, so when this setting is re-enabled, any previously set breakpoints will continue to function.



This setting is saved with [EditorPrefs](#) so it will persist across Unity launches on the local machine for all projects.

bool *IsDebugLoggingEnabled*

Enable this setting to log execution events when running a graph. The events that will be logged are:

- A branch has started
- A branch has been manually stopped
- Execution of a branch has completed
- Execution has paused at a breakpoint or after a step
- A connection has been followed to a new node

The current frame number is printed with the log message to make it easy to determine how many frames a particular node has taken to complete (since they are run as coroutines). Additional profiling and debugging information can be enabled with [CompositionManager.LogTracking](#).



This setting is saved with [EditorPrefs](#) so it will persist across Unity launches on the local machine for all projects.

Public Methods

void *GoTo*([InstructionGraphNode](#) *node*, **string** *name*)

Call this from a [node](#) to tell the graph to traverse to *node*. *name* should be the name of the property the node was assigned to for use in log messages.

The following two overloads perform the same task but can be used to provide more information in

log messages when *node* comes from a list (*index* would be the index of *node*) or dictionary (*key* would be the key of *node*).

- void GoTo([InstructionGraphNode](#) node, string name, int index)::
- void GoTo([InstructionGraphNode](#) node, string name, string key)::

void Break()

Call this from a [node](#) to tell the graph to return to the closest [node](#) in the call stack that is an [ILoopNode](#). The [BreakNode](#) calls this.

void BreakAll()

Call this from a [node](#) to tell the graph to stop running is current branch.

Protected Methods

IEnumerator Run([InstructionStore](#) variables, [InstructionGraphNode](#) root, string source)

Editor Support

The following properties and methods are exposed for use by the editor and only available in editor builds. They can be ignored.

- Action<[InstructionGraph](#), [InstructionGraph](#)> OnBreakpointHit::
- [Vector2](#) StartPosition::
- [InstructionGraphNodeList](#) Nodes (read only)::
- [PlaybackState](#) DebugState (read only)::
- bool CanDebugPlay (read only)::
- bool CanDebugPause (read only)::
- bool CanDebugStep (read only)::
- bool CanDebugStop (read only)::
- void DebugPlay():
- void DebugPause():
- void DebugStep():
- void DebugStop():
- int IsInCallStack([InstructionGraphNode](#) node)::
- bool IsInCallStack([InstructionGraphNode](#) node, string source)::
- bool IsExecuting([InstructionGraphNode](#) node)::
- void GetConnections([NodeData](#) data) (virtual)::
- void SetConnection([ConnectionData](#) connection, [InstructionGraphNode](#) target) (virtual)::

InstructionGraphNode

PiRhoSoft.CompositionEngine.InstructionGraphNode : [ScriptableObject](#)

Description

Derive from this class to implement a custom node for use in an [InstructionGraph](#).

Public Fields

string *Name*

The name of the node. This is used to display the node in the graph editor and in log messages to identify the node the message is related to.

Vector2 *GraphPosition*

Used by the editor to store the location of the node in the graph editor.

bool *IsBreakpoint*

Used by the editor to indicate whether this node has been marked as a breakpoint in the graph editor for debugging.

Public Properties

Color *NodeColor (read only) (virtual)*

The color the node should be displayed with in the graph editor. By default this will use [InstructionGraphNode.Colors.Default](#) but can be customized as a way to visually differentiate nodes in the graph editor.

Public Methods

IEnumerator **Run**([InstructionGraph](#) graph, [InstructionStore](#) variables, **int** iteration) (*abstract*)

Implement this method in derived classes to perform the execution of the node. Read the [graphs topic](#) for a complete overview of writing and using custom nodes.

Resolve

This collection of methods will lookup the value referenced by a [VariableReference](#) or [VariableSource](#). The resolved value is set to the output parameter *result* and the return value will indicate whether the value was resolved successfully. The *variables* parameter should be the *variables* parameter passed to the *Run* method. If the resolution fails, either due to the variable not being found or it being an invalid type, a warning will be printed to the [Console](#).

- **bool** Resolve([IVariableStore](#) variables, [VariableValueSource](#) source, [VariableValue](#) result (out))
- **bool** Resolve([IVariableStore](#) variables, [VariableReference](#) reference, [VariableValue](#) result (out))
- **bool** Resolve([IVariableStore](#) variables, [BoolVariableSource](#) source, **bool** result (out))
- **bool** Resolve([IVariableStore](#) variables, [VariableReference](#) reference, **bool** result (out))::

- `bool Resolve(IVariableStore variables, IntVariableSource source, int result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, int result (out))::`
- `bool Resolve(IVariableStore variables, FloatVariableSource source, float result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, float result (out))::`
- `bool Resolve(IVariableStore variables, Int2VariableSource source, Vector2Int result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Vector2Int result (out))::`
- `bool Resolve(IVariableStore variables, Int3VariableSource source, Vector3Int result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Vector3Int result (out))::`
- `bool Resolve(IVariableStore variables, IntRectVariableSource source, RectInt result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, RectInt result (out))::`
- `bool Resolve(IVariableStore variables, IntBoundsVariableSource source, BoundsInt result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, BoundsInt result (out))::`
- `bool Resolve(IVariableStore variables, Vector2VariableSource source, Vector2 result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Vector2 result (out))::`
- `bool Resolve(IVariableStore variables, Vector3VariableSource source, Vector3 result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Vector3 result (out))::`
- `bool Resolve(IVariableStore variables, Vector4VariableSource source, Vector4 result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Vector4 result (out))::`
- `bool Resolve(IVariableStore variables, QuaternionVariableSource source, Quaternion result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Quaternion result (out))::`
- `bool Resolve(IVariableStore variables, RectVariableSource source, Rect result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Rect result (out))::`
- `bool Resolve(IVariableStore variables, BoundsVariableSource source, Bounds result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Bounds result (out))::`
- `bool Resolve(IVariableStore variables, ColorVariableSource source, Color result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Color result (out))::`
- `bool Resolve(IVariableStore variables, StringVariableSource source, string result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, string result (out))::`
- `bool Resolve<EnumType>(IVariableStore variables, VariableSource<EnumType> source, EnumType result (out))::`
- `bool Resolve<EnumType>(IVariableStore variables, VariableReference reference, EnumType result (out))::`
- `bool Resolve(IVariableStore variables, StoreVariableSource source, IVariableStore result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, IVariableStore result (out))::`

- `bool Resolve(IVariableStore variables, ListVariableSource source, IVariableList result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, IVariableList result (out))::`
- `bool ResolveObject<ObjectType>(IVariableStore variables, VariableSource<ObjectType> source, ObjectType result (out))::`
- `bool ResolveObject<ObjectType>(IVariableStore variables, VariableReference reference, ObjectType result (out))::`
- `bool ResolveStore<StoreType>(IVariableStore variables, VariableReference reference, StoreType result (out))::`
- `bool ResolveList<ListType>(IVariableStore variables, VariableReference reference, ListType result (out))::`
- `bool ResolveInterface<InterfaceType>(IVariableStore variables, VariableReference reference, InterfaceType result (out))::`
- `bool ResolveReference(IVariableStore variables, VariableReference reference, Object result (out))::`

void Assign(IVariableStore variables, VariableReference reference, VariableValue value)

Assigns *value* to the variable referenced by *reference*. The *variables* parameter should be the *variables* parameter passed to the *Run* method. If the assignment fails, a warning will be logged.

void GetInputs(IList<VariableDefinition> inputs) (virtual)

Implement this method to customize the set of variables the node expects to have available as inputs on the `InstructionStore` when it is run. This rarely needs to be implemented as the base implementation should be sufficient most of the time. The base implementation will automatically find all `VariableReferences`, `VariableSources`, and `Expressions`.

void GetOutputs(IList<VariableDefinition> outputs) (virtual)

Implement this method to customize the set of variables this node will set as outputs on the `InstructionStore` when it is run. This rarely needs to be implemented as the base implementation should be sufficient most of the time. The base implementation will automatically find all `VariableReferences` and `Expressions`.

void GetConnections(NodeData data) (virtual)

Implement this method to specify the nodes this node has connections to. This rarely needs to be implemented as the base implementation should be sufficient most of the time.

void SetConnection(ConnectionData connection, InstructionGraphNode target) (virtual)

Used by the editor to update a connection. This only needs to be overridden if *GetConnections* is overridden.

InstructionGraphNodeDictionary

PiRhoSoft.CompositionEngine.InstructionGraphNodeDictionary : [SerializedDictionary](#)<string, string>

Description

Use this class as a field on an [InstructionGraphNode](#) to store an editable list of nodes that are accessed by name.

InstructionGraphNodeList

PiRhoSoft.CompositionEngine.InstructionGraphNodeList : [SerializedList](#)<[InstructionGraphNode](#)>

Description

Use this class as a field on an [InstructionGraphNode](#) to store an editable list of nodes that are accessed by index.

InstructionInput

PiRhoSoft.CompositionEngine.InstructionInput

Description

Used by [InstructionCaller](#) to store the data for an input [Variable](#).

Public Fields

string *Name*

The name used to access the value on the *Input store* from a [VariableReference](#) or [Expression](#).

InstructionInputType *Type*

Specifies how the value of the input is retrieved.

VariableReference *Reference*

If *Type* is Reference, holds the [VariableReference](#) used to look up the value.

VariableValue *Value*

If *Type* is Value, holds the value directly.

InstructionInputType

PiRhoSoft.CompositionEngine.InstructionInputType

Description

Defines the available types for an [InstructionInput](#).

Values

[InstructionInputType](#) *Reference*

The input is looked up using a [VariableReference](#).

[InstructionInputType](#) *Value*

The input [VariableValue](#) is set directly.

InstructionNode

PiRhoSoft.CompositionEngine.InstructionNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to run an external [InstructionGraph](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[InstructionSource](#) *Source*

Indicates how the [InstructionGraph](#) to run is specified.

[InstructionCaller](#) *Instruction*

If *Source* is Value, the [InstructionGraph](#) to run when this node is entered. [InstructionStore.Local](#) variables available to this node are not transferred to this graph - to share variables use the [InstructionStore.Global](#) store, [InstructionStore.Input](#) store, or *Context*.

[VariableReference](#) *Reference*

If *Source* is Reference, the reference to the [InstructionGraph](#) to run when this node is entered. [InstructionStore.Local](#) variables available to this node are not transferred to this graph - to share variables use the [InstructionStore.Global](#) store, [InstructionStore.Input](#) store, or *Context*.

[VariableValueSource](#) *Context*

The variable to use as the [InstructionStore.Context](#) for *Instruction*.

bool *WaitForCompletion*

If true, *Next* will not be run until execution of *Instruction* is complete. If false, *Next* will be run immediately and continue in parallel with *Instruction*.

InstructionOutput

PiRhoSoft.CompositionEngine.InstructionOutput

Description

Used by [InstructionCaller](#) to store the data for an output [Variable](#).

Public Fields

string *Name*

The name used to access the value on the [Output store](#) from a [VariableReference](#) or [Expression](#).

InstructionOutputType *Type*

Specifies how the value of the output is handled.

VariableReference *Reference*

If *Type* is Reference, holds the [VariableReference](#) that specifies where the output value should be stored after execution of the instruction finishes.

InstructionOutputType

PiRhoSoft.CompositionEngine.InstructionOutputType

Description

Defines the available types for an [InstructionOutput](#).

Values

InstructionOutputType *Ignore*

The output will be ignored.

InstructionOutputType *Reference*

The input is set using a [VariableReference](#).

InstructionSource

PiRhoSoft.CompositionEngine.InstructionSource

Description

Defines the options for the *Source* of a [InstructionGraph](#) in an [InstructionNode](#).

Values

[InstructionSource](#) *Value*

The [InstructionGraph](#) is specified directly in *Instruction*.

[InstructionSource](#) *Reference*

The [InstructionGraph](#) is resolved from the [VariableReference](#) *Reference*.

InstructionStore

PiRhoSoft.CompositionEngine.InstructionStore : [IVariableStore](#)

Description

The [IVariableStore](#) used with [InstructionGraphs](#) to provide a robust interface for accessing and isolating variables for use by [InstructionGraphNode](#)s. When using an [InstructionCaller](#) all management of the store will be handled automatically including creation of the store, reading input variables, and writing output variables.

Static Fields

string *InputStoreName*

The name used to access the *Input* store from a [VariableReference](#) or [Expression](#). This is set to "input".

string *OutputStoreName*

The name used to access the *Output* store from a [VariableReference](#) or [Expression](#). This is set to "output".

string *LocalStoreName*

The name used to access the *Local* store from a [VariableReference](#) or [Expression](#). This is set to "local".

Static Methods

bool **IsInput**([VariableReference](#) variable)

Determines if *variable* reads from the *Input* store. This can be used from overridden implementations of *GetInputs* in rare cases where the default implementation isn't sufficient.

bool **IsOutput**([VariableReference](#) variable)

Determines if *variable* writes to the *Output* store. This can be used from overridden implementations of *GetOutputs* in rare cases where the default implementation isn't sufficient.

bool **IsInput**([InstructionInput](#) input)

Determines if *input* reads from the *Input* store. This can be used from overridden implementations of *GetInputs* in rare cases where the default implementation isn't sufficient.

bool **IsOutput**([InstructionOutput](#) output)

Determines if *inputs* writes to the *Output* store. This can be used from overridden implementations of *GetOutputs* in rare cases where the default implementation isn't sufficient.

Public Properties

string *ContextName* (read only)

The name used to access the *Context* from a [VariableReference](#) or [Expression](#).

VariableValue *Context (read only)*

The value sent as *context* in the `InstructionStore` constructor.

VariableStore *Input (read only)*

The variable store, accessed with `InputStoreName`, that holds variables passed from the [caller](#). [Variables](#) in this store can be accessed and changed, but new [variables](#) cannot be added.

VariableStore *Output (read only)*

The variable store, accessed with `OutputStoreName`, that holds variables set by the instruction and returned to the [caller](#). [Variables](#) in this store can be accessed and changed, but new [variables](#) cannot be added. The store will be pre-populated with variables specified as *Outputs* on the [caller](#).

VariableStore *Local (read only)*

The variable store, accessed with `LocalStoreName`, that holds variables that are isolated to the execution of the *instruction* this store was created with. When execution begins, this store will be empty, but variables can be added or changed on this store at any time without affecting any other stores.



It is not required to use `LocalStoreName` when accessing the local store but it can improve readability or resolve ambiguities in some cases.

Public Methods

void WriteInputs([InstructionCaller](#) instruction, [IList<InstructionInput>](#) inputs, [IVariableStore](#) caller)

Takes each of the [InstructionInputs](#) from *inputs*, resolves them using *caller* if they are [VariableReferences](#), and adds them to the *Input* store.

void WriteOutputs([IList<InstructionOutput>](#) outputs)

Takes each of the [InstructionOutputs](#) from *outputs* and adds them to the *Output* store.

void ReadOutputs([IList<InstructionOutput>](#) outputs, [IVariableStore](#) caller)

Takes each of the [InstructionOutputs](#) from *outputs* and resolves them using this store if they are [VariableReferences](#), and adds them to *caller*.

[VariableValue](#) GetVariable(string name)

Returns the value of the variable with name *name* on this store. If *name* is not found, the *Local* store will be searched.

[SetVariableResult](#) SetVariable(string name, [VariableValue](#) value)

Each of the names exposed by this store are read only, but if *name* is unrecognized, this will attempt to set *value* on the *Local* store.

[IList<string>](#) GetVariableNames()

Returns the names of all variables exposed by this store. This is `InputStoreName`, `OutputStoreName`, `LocalStoreName`, `CompositionManager.GlobalStoreName`,

CompositionManager.SceneStoreName, and ContextName.

InstructionTrigger

PiRhoSoft.CompositionEngine.InstructionTrigger : [MonoBehaviour](#)

Description

Add this to an object to provide an interface for specifying an [InstructionGraph](#) in the editor that can be run from code. This is also used as a base class for behaviours that run [graphs](#) on certain events. Built in implementations are:

- [ButtonGraphTrigger](#)
- [ClickGraphTrigger](#)
- [EnableGraphTrigger](#)
- [StartGraphTrigger](#)

Additionally, [InstructionGraphTrigger](#) and [CollisionGraphTrigger](#) are included that, while not deriving from this class, perform a similar function.

Public Fields

[InstructionCaller](#) *Graph*

The [InstructionGraph](#) to execute when *Run* is called.

Public Methods

void Run()

Runs *Graph* using the [CompositionManager](#). [CompositionManager.DefaultStore](#) is used to read input variables from and this is used as the *Context*

Int2VariableSource

PiRhoSoft.CompositionEngine.Int2VariableSource : [VariableSource](#)<[Vector2Int](#)>

Description

A [VariableSource](#) for [Vector2Int](#).

Constructors

Int2VariableSource([Vector2Int](#) defaultValue)

Initializes *Value* to *defaultValue*.

Int3VariableSource

PiRhoSoft.CompositionEngine.Int3VariableSource : [VariableSource](#)<[Vector3Int](#)>

Description

A [VariableSource](#) for [Vector3Int](#).

Constructors

Int3VariableSource([Vector3Int](#) defaultValue)

Initializes *Value* to *defaultValue*.

IntBoundsVariableSource

PiRhoSoft.CompositionEngine.IntBoundsVariableSource : [VariableSource](#)<[BoundsInt](#)>

Description

A [VariableSource](#) for [BoundsInt](#).

Constructors

IntBoundsVariableSource([BoundsInt](#) defaultValue)

Initializes *Value* to *defaultValue*.

InterfaceControl

PiRhoSoft.CompositionEngine.InterfaceControl : [MonoBehaviour](#)

Description

Add this to any [MonoBehaviour](#) to provide support for enabling and disabling the object from an [InstructionGraph](#) using [ShowControlNode](#) and [HideControlNode](#). Read the [Interface Control topic](#) for more information on how and when to use InterfaceControls.



An InterfaceControl will always start inactive.

Public Fields

[DependentObjectList](#) *DependentObjects*

A list of [GameObjects](#) whose enabled state should always match the enabled state of this object.

Public Properties

bool *IsActive* (*read only*)

true if the control is currently enabled, false otherwise.

Public Methods

void *Activate()*

Enables the control (and *DependentObjects*) if it is not already enabled. *Setup* will be called only if the control is not already enabled.

void *Deactivate()*

Disables the control (and *DependentObjects*) regardless of how many times *Activate* was called. *Teardown* will be called only if the control is not already disabled.

Protected Methods

void *Setup()* (*virtual*)

Implement this method in a subclass to perform setup when the object becomes enabled. The base implementation does nothing.

void *Teardown()* (*virtual*)

Implement this method in a subclass to perform clean up when the object becomes disabled. The base implementation does nothing.

IntRectVariableSource

PiRhoSoft.CompositionEngine.IntRectVariableSource : [VariableSource](#)<[RectInt](#)>

Description

A [VariableSource](#) for [RectInt](#).

Constructors

IntRectVariableSource([RectInt](#) defaultValue)

Initializes *Value* to *defaultValue*.

IntVariableConstraint

PiRhoSoft.CompositionEngine.IntVariableConstraint : [VariableConstraint](#)

Description

A [VariableConstraint](#) for Int [VariableValues](#) that restricts the value to a range.

Public Fields

int *Minimum*

The smallest value allowed for the value.

int *Maximum*

The largest value allowed for the value.

IntVariableSource

PiRhoSoft.CompositionEngine.IntVariableSource : [VariableSource](#)<int>

Description

A [VariableSource](#) for ints.

Constructors

IntVariableSource(int *defaultValue*)

Initializes *Value* to *defaultValue*.

ISchemaOwner

PiRhoSoft.CompositionEngine.ISchemaOwner

Description

Implement this interface on a class that also implements [IVariableStore](#) to indicate to other systems that this store is constrained by a [VariableSchema](#). This is used to improve the editing experience and enable runtime serialization of the store data. The built in classes [ConstrainedStore](#), [VariableSetComponent](#), and [VariableSetAsset](#) implement this and should be sufficient for most use cases.

Public Properties

VariableSchema *Schema (read only) (abstract)*

The [VariableSchema](#) that is constraining this store.

Public Methods

void SetupSchema() *(abstract)*

This method should apply the schema to the store.

ISequenceNode

PiRhoSoft.CompositionEngine.ISequenceNode

Description

Implement this interface in an [InstructionGraphNode](#) subclass to inform an [InstructionGraph](#) that the node should be run repeatedly. The graph will continue to run the node until the node does not call [InstructionGraph.GoTo](#) (or calls `GoTo(null)`).

IterateNode

PiRhoSoft.CompositionEngine.IterateNode : [InstructionGraphNode](#), [ILoopNode](#)

Description

Add this to an [InstructionGraph](#) to execute an [InstructionGraphNode](#) repeatedly for each [VariableValue](#) in an [IVariableList](#).

Public Fields

[VariableReference](#) *Container*

The [IVariableList](#) holding each of the [VariableValues](#) to iterate.

[VariableReference](#) *Index*

The variable to set to the current number of times the node has been repeated.

[VariableReference](#) *Value*

The variable to set to the current value being iterated.

[InstructionGraphNode](#) *Loop*

The [InstructionGraphNode](#) to run for each [VariableValue](#) in *Container*.

IVariableList

PiRhoSoft.CompositionEngine.IVariableList

Description

Implement this interface on a class to allow the class to be stored with type `List` in a [VariableValue](#). [VariableList](#) provides an implementation that is sufficient for most use cases.

Public Properties

int *Count* (read only) (abstract)

The number of items in the list.

Public Methods

VariableValue **GetVariable(int index)** (abstract)

Returns the value at the index *index* in the list.

SetVariableResult **SetVariable(int index, VariableValue value)** (abstract)

Sets the value at index *index* to *value*.

SetVariableResult **AddVariable(VariableValue value)** (abstract)

Adds the value *value* to the end of the list.

SetVariableResult **RemoveVariable(int index)** (abstract)

Removes the value at index *index* from the list.

IVariableListener

PiRhoSoft.CompositionEngine.IVariableListener

Description

Implement this in a class that uses a [MappedVariableStore](#) to receive notifications whenever a variable in the store changes.

Public Methods

void VariableChanged(int *index*, [VariableValue](#) *value*) (*abstract*)

Called by [MappedVariableStore](#) to indicate the value at index *index* was changed to *value*.

IVariableReset

PiRhoSoft.CompositionEngine.IVariableReset

Description

Implement this interface to add support for the class to be resolved from *Object* in a [ResetTagNode](#) or [ResetVariablesNode](#). Although there is no restriction on how this interface can be used, it is intended as a way to reset [Variables](#) in a [VariableSchema](#) based on the [ValueDefinition.Tag](#) (with *ResetTag*) or [Variable.Name](#) (with *ResetVariables*).

Public Methods

void ResetTag(string tag) (abstract)

Called from [ResetTagNode](#) with *tag* as the tag that should be reset.

void ResetVariables(IList<string> variables) (abstract)

Called from [ResetVariablesNode](#) with *variables* as the list of names that should be reset.

IVariableStore

PiRhoSoft.CompositionEngine.IVariableStore

Description

Implement this interface on a class to allow the class to be stored with type `Store` in a [VariableValue](#). Many built in implementations are provided for various use cases:

- [VariableStore](#)
- [ConstrainedStore](#)
- [ReadOnlyStore](#)
- [WritableStore](#)
- [SceneVariableStore](#)
- [MappedVariableStore](#)



This interface is one of the most important pieces to the variable system. Read the [variables topic](#) for a complete description of this interface and how it interacts with the rest of the system.

Public Methods

[VariableValue](#) GetVariable(string name) (*abstract*)

Returns the value of the variable with name *name*.

[SetVariableResult](#) SetVariable(string name, [VariableValue](#) value) (*abstract*)

Sets the value of the variable with name *name* to *value*

[IList<string>](#) GetVariableNames() (*abstract*)

Returns the complete list of variable names that exist in this store.

ListAdapter

PiRhoSoft.CompositionEngine.ListAdapter : [IVariableList](#)

Description

This serves as a base class for several internal classes that wrap specific [IList](#) types so they can be accessed as a [VariableValue](#) with *Type* List. To use a ListAdapter call the static *Create* method.

Static Methods

[IVariableList](#) **Create**([IList](#) *list*)

Creates an [IVariableList](#) that wraps and modifies *list* when it is accessed.

ListBinding

PiRhoSoft.CompositionEngine.ListBinding : [VariableBinding](#)

Description

Add this to any [GameObject](#) to add child objects instantiated from a [prefab](#) for each item in an [IVariableList](#).

Public Fields

[VariableReference](#) *Variable*

The [IVariableList](#) to bind to.

[BindingRoot](#) *Template*

The [prefab](#) that will be instantiated as a child of this object for each item in the [list](#) referenced by *Variable*.

ListVariableConstraint

PiRhoSoft.CompositionEngine.ListVariableConstraint : [VariableConstraint](#)

Description

A [VariableConstraint](#) for List [VariableValues](#) that specifies the [VariableType](#) of [VariableValues](#) that can be added to the list.

Public Fields

[VariableType](#) *ItemType*

The type of items in the list. If this is [Empty](#), any value can be added.

[VariableConstraint](#) *ItemConstraint*

The constraint to enforce for each item in the list.

ListVariableSource

PiRhoSoft.CompositionEngine.ListVariableSource : [VariableSource](#)<[IVariableList](#)>

Description

A [VariableSource](#) for [IVariableLists](#).

LoadSceneNode

PiRhoSoft.CompositionEngine.LoadSceneNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to load a scene.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[SceneSource](#) *Source*

Specifies how the scene to load is retrieved.

[SceneReference](#) *Scene*

If *Source* is Value, holds the scene to load.

[VariableReference](#) *SceneVariable*

If *Source* is Variable, references the scene to load. If the resolved value is an Int, the scene will be loaded by build index. If it is a String, it will be loaded by name.

string *SceneName*

If *Source* is Name, the name of the scene to load.

int *SceneIndex*

If *Source* is Index, the build index of the scene to load.

bool *WaitForCompletion*

If this is true (the default), the node will block until the scene has been loaded. Otherwise the scene will be loaded and the graph will continue in parallel.

bool *CleanupAssets*

If this is true (the default), [Resources.UnloadUnusedAssets](#) will be called after the scene has been loaded.

bool *Additive*

If this is true (the default), the scene will be loaded in additive mode, meaning all other loaded scenes will remain loaded. If this is false, all currently loaded scenes will be unloaded first.

LogNode

PiRhoSoft.CompositionEngine.LogNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to log a message to the [console](#) to aid in debugging.

Public Fields

Message *Message*

The message to log.

InstructionGraphNode *Next*

The [InstructionGraphNode](#) to run when this node finishes.

LoopNode

PiRhoSoft.CompositionEngine.LoopNode : [InstructionGraphNode](#), [ILoopNode](#)

Description

Add this to an [InstructionGraph](#) to executes an [InstructionGraphNode](#) repeatedly until a condition [Expression](#) evaluates to false.

Public Fields

[InstructionGraphNode](#) *Loop*

The [InstructionGraphNode](#) to run repeatedly while *Condition* is true.

[VariableReference](#) *Index*

The variable to set to the current number of times the node has been repeated.

[Expression](#) *Condition*

The [Expression](#) to evaluate to determine if the node should continue to repeat.

MappedVariableAttribute

PiRhoSoft.CompositionEngine.MappedVariableAttribute : Attribute

Description

Add this to a property or field on a [VariableSetComponent](#) or [VariableSetAsset](#) to expose it to the [variables](#) system.

Constructors

MappedVariableAttribute(bool readOnly)

Pass true as *readOnly* to indicate the property or field cannot be set by the variables system. If the default constructor is used the variable is allowed to be set.

Public Properties

bool ReadOnly (read only)

If this is true, the property or field cannot be set through a [MappedVariableStore](#).

MappedVariableStore

PiRhoSoft.CompositionEngine.MappedVariableStore : [IVariableStore](#)

Description

An [IVariableStore](#) implementation that provides an [attribute](#) based interface for exposing properties and fields defined in code to the [variables](#) system. This is used by [VariableSetComponent](#) and [VariableSetAsset](#).

Public Properties

int *VariableCount* (*read only*)

The total number of variables in the store.

Public Methods

void **Setup**(**Object** *owner*, [VariableSchema](#) *schema*, [VariableSet](#) *variables*)

Initializes the store with [MappedVariables](#) from *owner* and [Variables](#) defined by *schema*. *variables* is initialized with *schema* as well.

[VariableValue](#) **GetVariable**(**string** *name*)

Returns the value of the variable with name *name*.

[SetVariableResult](#) **SetVariable**(**string** *name*, [VariableValue](#) *value*)

Sets the value of the variable with name *name* to *_value*.

[IList](#)<**string**> **GetVariableNames**()

Returns a list of the names of all the variables in this store.

string **GetVariableName**(**int** *index*)

Returns the name of the variable at index *index*.

[VariableValue](#) **GetVariableValue**(**int** *index*)

Returns the value of the variable at index *index*.

[SetVariableResult](#) **SetVariableValue**(**int** *index*, [VariableValue](#) *value*)

Sets the value of the variable at index *index* to *value*.

MaterialAnimation

PiRhoSoft.CompositionEngine.MaterialAnimation : [MonoBehaviour](#), [ICompletionNotifier](#)

Description

Add this to a [Renderer](#) to animate a `_Progress` [material property](#). This can be used standalone but is most useful when used with the [PlayEffectNode](#).

Public Fields

bool *AutoAdvance*

If this is true (the default), *Progress* will be automatically updated every frame according to the values set for *UseScaledTime* and *Duration*.

float *Progress*

The value that is set on the `_Progress` [material property](#) for a sibling [Renderer](#).

bool *UseScaledTime*

If *AutoAdvance* is true, specifies how *Progress* will be updated when [Time.timeScale](#) is changed. When true, [Time.deltaTime](#) is used, otherwise [Time.unscaledDeltaTime](#) is used.

float *Duration*

If *AutoAdvance* is true, specifies the total duration of the animation. The `_Progress` [material property](#) will be set to `_Progress_ / _Duration`.

Public Properties

bool *IsComplete* (*read only*)

Returns true as soon as the animation has completed. When *AutoAdvance* is true, the animation is complete when *Progress* \geq *Duration*. When *AutoAdvance* is false, the animation is complete when *Progress* \geq 1.0

Protected Methods

void *LateUpdate()* (*virtual*)

Performs the update of the `_Progress` [material property](#).

Menu

PiRhoSoft.CompositionEngine.Menu : [MonoBehaviour](#), [IVariableStore](#)

Description

Add this to any [GameObject](#) to manage a list of child [MenuItems](#). Additionally add a [MenuInput](#) to manage input. [MenuItems](#) can be added directly as children in the editor or at runtime, or by using [ListBinding](#). Additionally, this can be used with a [SelectionControl](#) to automate the process of selecting from a menu in an [InstructionGraph](#).

Public Fields

Action<[MenuItem](#)> *OnItemAdded*

Subscribe to this callback to receive a notification any time a [MenuItem](#) is added to this menu.

Action<[MenuItem](#)> *OnItemRemoved*

Subscribe to this callback to receive a notification any time a [MenuItem](#) is removed from this menu.

Action<[MenuItem](#)> *OnItemMoved*

Subscribe to this callback to receive a notification any time a [MenuItem](#)'s position in the menu changes.

Action<[MenuItem](#)> *OnItemBlurred*

Subscribe to this callback to receive a notification any time a [MenuItem](#) loses focus.

Action<[MenuItem](#)> *OnItemFocused*

Subscribe to this callback to receive a notification any time a [MenuItem](#) gains focus.

Action<[MenuItem](#)> *OnItemSelected*

Subscribe to this callback to receive a notification any time a [MenuItem](#) is selected.

Action *OnCancelled*

Subscribe to this callback to receive a notification any time the menu is closed without a selection being made.

Public Properties

List<[MenuItem](#)> *Items (read only)*

The [MenuItems](#) in this menu. This is automatically updated to reflect the current set of [MenuItems](#) that are children of this object.

MenuItem *FocusedItem*

The [MenuItem](#) that has focus.

int *FocusedIndex*

The index of the [MenuItem](#) that has focus.

Public Methods

void SelectItem([MenuItem](#) item)

Selects *item*. The result of an item being selected is only that *OnItemSelected* will be triggered. Selection is most commonly used indirectly through a [SelectionControl](#).

void Cancel()

Triggers *OnCancelled* with no other effect.

[IList<string>](#) GetVariableNames()

Returns the names of the [Variables](#) exposed by this [IVariableStore](#). These are "FocusedItem" and "FocusedIndex".

[VariableValue](#) GetVariable(string name)

Returns the variable with name *name*.

[SetVariableResult](#) SetVariable(string name, [VariableValue](#) value)

Sets the value of the variable with *name* to *value*.

Protected Methods

void ItemAdded([MenuItem](#) item) (virtual)

Called when a [MenuItem](#) (*item*) is added to the menu. The base implementation triggers *OnItemAdded*.

void ItemRemoved([MenuItem](#) item) (virtual)

Called when a [MenuItem](#) (*item*) is removed from the menu. The base implementation triggers *OnItemRemoved*.

void ItemMoved([MenuItem](#) item) (virtual)

Called when a [MenuItem](#) (*item*)'s position in menu changes. The base implementation triggers *OnItemMoved*.

void ItemFocused([MenuItem](#) item) (virtual)

Called when a [MenuItem](#) (*item*) gains focus. The base implementation triggers *OnItemFocused*.

void ItemBlurred([MenuItem](#) item) (virtual)

Called when a [MenuItem](#) (*item*) loses focus. The base implementation triggers *OnItemBlurred*.

void ItemSelected([MenuItem](#) item) (virtual)

Called when a [MenuItem](#) (*item*) is selected. The base implementation triggers *OnItemSelected*.

void Cancelled() (virtual)

Called when the menu is cancelled. The base implementation triggers *OnCancelled*.

MenuInput

PiRhoSoft.CompositionEngine.MenuInput : [MonoBehaviour](#)

Description

Add this to a [Menu](#) to provide navigation and selection of [MenuItems](#).

Public Fields

string *HorizontalAxis*

The name of the axis, as used by [InputHelper](#), that moves focus left and right through the [Menu](#).

string *VerticalAxis*

The name of the axis, as used by [InputHelper](#), that moves focus up and down through the [Menu](#).

string *SelectButton*

The name of the button, as used by [InputHelper](#), that will select the focused item on the [Menu](#).

string *CancelButton*

The name of the button, as used by [InputHelper](#), that will cancel the menu.

MenuInputPointerAction *HoverAction*

The action to perform when the mouse moves over an item in the [Menu](#).

MenuInputPointerAction *ClickAction*

The action to perform when the mouse is clicked while over an item in the [Menu](#).

PrimaryAxis *PrimaryAxis*

Specifies how [MenuItems](#) are laid out in the [Menu](#) relative to their child index in the object. If items are laid out top to bottom (potentially with multiple columns), use Column. If items are laid out left to right (potentially with multiple rows), use Row.

int *RowCount*

If *PrimaryAxis* is Row, specifies the number of rows of [MenuItems](#) in the [Menu](#).

int *ColumnCount*

If *PrimaryAxis* is Column, specifies the number of columns of [MenuItems](#) in the [Menu](#).

MenuInput *NextLeft*

Specifies the menu to transfer focus to when moving past the left most [MenuItem](#) in the [Menu](#). This can be set to this menu input to cause focus to wrap back to the right. If this is not set, focus will be clamped to the left most column.

MenuInput *NextRight*

Specifies the menu to transfer focus to when moving past the right most [MenuItem](#) in the [Menu](#). This can be set to this menu input to cause focus to wrap back to the left. If this is not set, focus

will be clamped to the right most column.

MenuItem *NextUp*

Specifies the menu to transfer focus to when moving past the top most **MenuItem** in the **Menu**. This can be set to this menu input to cause focus to wrap back to the bottom. If this is not set, focus will be clamped to the top most row.

MenuItem *NextDown*

Specifies the menu to transfer focus to when moving past the bottom most **MenuItem** in the **Menu**. This can be set to this menu input to cause focus to wrap back to the top. If this is not set, focus will be clamped to the bottom most row.

bool *FocusOnLoad*

Set this to true to have the first **MenuItem** gain focus when this behaviour is loaded.

float *ScrollPadding*

When inside a <https://docs.unity3d.com/Manual/script-ScrollRect.html>, indicates the amount of padding to maintain around the focused item when menu navigation causes the menu to scroll.

Public Methods

void *EnterFromBeginning()*

Focuses the first **MenuItem**.

void *EnterFromEnd()*

Focuses the last **MenuItem**.

void *EnterFromLeft(int fromRow)*

Focuses the left most **MenuItem** in row *fromRow*.

void *EnterFromRight(int fromRow)*

Focuses the right most **MenuItem** in row *fromRow*.

void *EnterFromTop(int fromColumn)*

Focuses the top most **MenuItem** in column *fromColumn*.

void *EnterFromBottom(int fromColumn)*

Focuses the bottom most **MenuItem** in column *fromColumn*.

void *Leave()*

Clear focus so no **MenuItem** has focus.

void *MoveFocusUp(int amount)*

Focus the **MenuItem** *amount* rows above the current focused item.

void *MoveFocusDown(int amount)*

Focus the **MenuItem** *amount* rows below the current focused item.

void MoveFocusLeft(int amount)

Focus the [MenuItem](#) *amount* columns to the left of the current focused item.

void MoveFocusRight(int amount)

Focus the [MenuItem](#) *amount* columns to the right of the current focused item.

void RefreshLayout()

Re-layout the [MenuItems](#). Layout is maintained automatically when [MenuItems](#) are added, moved, or removed, but if *PrimaryAxis*, *ColumnCount*, or *RowCount* changes without altering the [MenuItems](#), this should be called.

[MenuItem](#) GetItem(Vector2 screenPoint)

Returns the [MenuItem](#) at position *screenPoint*. *screenPoint* is in the same coordinate system as [Input.mousePosition](#).

void ScrollToItem(MenuItem item)

When inside a <https://docs.unity3d.com/Manual/script-ScrollRect.html>, ensures *item* is visible with *ScrollPadding* space around it on all sides.

MenuInputPointerAction

PiRhoSoft.CompositionEngine.MenuInputPointerAction

Description

Defines the available options for mouse actions on [MenuInput](#).

Values

MenuInputPointerAction *None*

The action will have no effect.

MenuInputPointerAction *Focus*

The action will focus the [MenuItem](#).

MenuInputPointerAction *Select*

The action will select the [MenuItem](#).

MenuItem

PiRhoSoft.CompositionEngine.MenuItem : [BindingRoot](#)

Description

Add this to any [GameObject](#) that is a child of a [Menu](#) to indicate the object should be managed by the [Menu](#).

Public Fields

string *ItemName*

The name to use to access the this item from child [VariableBindings](#). Available variables are Index, Column, Row, Label, and Focused.

Public Properties

int *Index (read only)*

The index of the item in the [Menu](#).

int *Column (read only)*

The index of the column the item is in in the [Menu](#).

int *Row (read only)*

The index of the row the item is in in the [Menu](#).

string *Label (read only)*

The label assigned to the item by a [SelectionControl](#).

bool *Focused (read only)*

true when this item is the focused item in its [Menu](#)

MenuItemTemplate *Template (read only)*

The template this item was generated from or initialized with.

bool *Generated (read only)*

true if this item was generated from a prefab set by a [MenuItemTemplate](#).

Public Methods

void **Setup**([MenuItemTemplate](#) *template*, **bool** *generated*)

Initializes *Template* and *Generated* after the item has been associated with a [Menu](#).

void **Move**(**int** *index*)

Moves the item in its [Menu](#).

MenuItemTemplate

PiRhoSoft.CompositionEngine.MenuItemTemplate

Description

Holds information about how a [MenuItem](#) should be setup in a [Menu](#).

Public Fields

VariableReference *Variables*

The variable that should be used as the [BindingRoot](#) *Value* for the [MenuItem](#).

ObjectSource *Source*

Specifies whether the [MenuItem](#) should be looked up in the scene using *Name* (Scene) or created from a [prefab](#) using *Template* (Asset).

string *Name*

When *Source* is Name, the name of the [GameObject](#) containing the [MenuItem](#) in the loaded scenes.

MenuItem *Template*

When *Source* is Asset, the [prefab](#) to create the [MenuItem](#) from.

string *Label*

When *Source* is Asset, the label to assign to the [MenuItem](#).

bool *Expand*

When *Source* is Asset, this is true, and *Variables* references a [List](#), a [MenuItem](#) will be created from *Template* for each item in the [List](#).

Public Properties

string *Id* (*read only*)

The identifier used for the item when referenced by string. If *Source* is Scene this will be *Name*. If *Source* is Asset this will be *Label*.

Message

PiRhoSoft.CompositionEngine.Message

Description

Add this as a field of a class to provide an editable text field that can be formatted with [VariableReferences](#).

Public Fields

string *Text*

The string that will be formatted at runtime. [VariableReferences](#) to resolve can be inserted in the text by surrounding it with braces ({ and }). Access the resolved text with the *GetText* method. An example message is shown in the [Messages](#) topic.

Public Properties

bool *HasText* (read only)

Indicates that *Text* has been set and *GetText* will return a non-empty string.

Public Methods

void *GetInputs*(**IList**<[VariableDefinition](#)> *inputs*)

Adds a definition for each [VariableReference](#) in *Text* to *inputs* if the [VariableReference](#) accesses [InstructionStore.Input](#).

string *GetText*(**IVariableStore** *variables*, **bool** *suppressErrors*)

Formats and returns *Text*, looking up any [VariableReferences](#) on *variables*. If *suppressErrors* is false, an error will be logged when a [VariableReference](#) cannot be resolved.

MessageBinding

PiRhoSoft.CompositionEngine.MessageBinding : [StringBinding](#)

Description

Add this to a [TextMeshPro](#) to set the text to the string retrieved from a [Message](#).

Public Fields

Message *Message*

The [Message](#) to resolve and apply to the [TextMeshPro](#) when the binding is updated.

MessageControl

PiRhoSoft.CompositionEngine.MessageControl : [InterfaceControl](#)

Description

Add this to a [TextMeshPro](#) to display messages from a [MessageNode](#). Add a [MessageInput](#) to support dismissing the control with a button press.

Public Fields

[TMP_Text](#) *DisplayText*

The [TextMeshPro](#) that the text will be displayed on. This component will be enabled and disabled along with this MessageControl.

Public Properties

bool *IsRunning* (read only)

Returns true when this MessageControl is displaying text.

bool *IsAdvancing*

Returns true when this MessageControl should have its text advanced. This can be set from subclasses to reset the flag after it has been consumed.

Public Methods

void **Show**(string *text*)

Activates the MessageControl if necessary, sets *text* on *DisplayText*, and enables *DisplayText*. If this is called a second time before it is hidden, the text will simply be replaced and the control will continue to function as normal.

void **Advance**()

Sets the *IsAdvancing* flag so the text will advance on the next frame.

Protected Methods

[IEnumerator](#) **Run**() (virtual)

This method can be overridden to perform custom handling of advancement. By default, the control will be dismissed when *Advance* is called, but this could be changed to add support for, for example, paging.

MessageInput

PiRhoSoft.CompositionEngine.MessageInput : [MonoBehaviour](#)

Description

Add this to a [MessageControl](#) to add support for advancing the text when a button is pressed.

Public Fields

string *AcceptButton*

The name of the button that advances the [MessageControl](#) as defined in [InputHelper](#).

Protected Properties

MessageControl *Message (read only)*

The [MessageControl](#) attached as a sibling to this component.

MessageNode

PiRhoSoft.CompositionEngine.MessageNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to display a [Message](#) on a [MessageControl](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Control*

The [MessageControl](#) to display *Message* on.

bool *WaitForCompletion*

When *true*, this [InstructionGraphNode](#) will not complete until the [MessageControl](#) has been dismissed.

bool *AutoHide*

When *true*, the [MessageControl](#) will be automatically dismissed after *WaitTime* seconds.

float *WaitTime*

When *AutoHide* is *true*, the number of seconds to wait before dismissing the [MessageControl](#).

[Message](#) *Message*

The [Message](#) to display on the [MessageControl](#).

MockupConnection

PiRhoSoft.CompositionEngine.MockupConnection

Description

Holds data for a connection in a [MockupNode](#).

Public Fields

string *Name*

The name to display in the graph window for this connection.

InstructionGraphNode *Node*

The node this connection is connected to.

MockupConnectionList

PiRhoSoft.CompositionEngine.MockupConnectionList : [SerializedList](#)<[MockupConnection](#)>

Description

The serializable list of [MockupConnections](#) for a [MockupGraph](#) or [MockupNode](#).

MockupGraph

PiRhoSoft.CompositionEngine.MockupGraph : [InstructionGraph](#)

Description

Performs no function but can have an arbitrary number of entry points for quickly making [InstructionGraph](#) blueprints.

Public Fields

[MockupConnectionList](#) *EntryPoints*

The list of [MockupConnections](#) to show as entry points in the start node of the graph editor window.

MockupNode

PiRhoSoft.CompositionEngine.MockupNode : [InstructionGraphNode](#)

Description

Add this to a [MockupGraph](#) to quickly make an [InstructionGraph](#) blueprint.

Public Fields

[MockupConnectionList](#) *Connections*

The connections that have been added to this node.

[Color](#) *DisplayColor*

The color of the node in the graph editor window.

NodeData

PiRhoSoft.CompositionEngine.NodeData

Description

Stores data about an [InstructionGraphNode](#). This is managed automatically by the editor and can be ignored.

NumberBinding

PiRhoSoft.CompositionEngine.NumberBinding : [StringBinding](#)

Description

Add this to a [TextMeshPro](#) to set the text to a formatted number.

Public Fields

[BindingFormatter](#) *Format*

Specifies how the number in *Variable* should be interpreted and formatted when converting it to a string.

[VariableReference](#) *Variable*

The Int or Float that will be formatted and applied when the binding is updated.

NumberFormatType

PiRhoSoft.CompositionEngine.NumberFormatType

Description

Defines the number formats available to set for the *NumberFormatting* of a [BindingFormatter](#).

Values

NumberFormatType *Percentage*

The number will be formatted as a percentage. Equivalent to setting the [custom format string](#) to "0.##%".

NumberFormatType *Commas*

The number will be formatted as a number with commas separating every 3 digits. Equivalent to setting the [custom format string](#) to ",,0".

NumberFormatType *Rounded*

The number will be rounded before converting it to a string. Equivalent to setting the [custom format string](#) to "0".

NumberFormatType *Decimal*

The number will be rounded to 2 decimal places before converting it to a string. Equivalent to setting the [custom format string](#) to "0.00".

NumberFormatType *Custom*

The format string will be read from the *ValueFormat* property of the [BindingFormatter](#).

ObjectBindingRoot

PiRhoSoft.CompositionEngine.ObjectBindingRoot : [BindingRoot](#)

Description

Add this to any [GameObject](#) to add a specified object to the [BindingRoot](#) hierarchy.

Public Fields

Object *Object*

The [Object](#) to return in *Value* for this [BindingRoot](#).

ObjectPositioning

PiRhoSoft.CompositionEngine.ObjectPositioning

Description

Defines the available settings for the *Positioning* property of [PlayEffectNode](#).

Values

ObjectPositioning *Absolute*

The created object will be placed at the scene root and positioned at the value of [PlayEffectNode.Position](#) in world space.

ObjectPositioning *Relative*

The created object will be placed at the scene root and positioned at the value of [PlayEffectNode.Position](#) relative to [PlayEffectNode.Object](#).

ObjectPositioning *Child*

The created object will be placed as a child of [PlayEffectNode.Parent](#) and positioned at the value of [PlayEffectNode.Position](#) in [PlayEffectNode.Parent's](#) coordinates.

ObjectPositioning

PiRhoSoft.CompositionEngine.ObjectPositioning

Description

Defines the available settings for the *Positioning* property of [CreateGameObjectNode](#).

Values

ObjectPositioning *Absolute*

The created [GameObject](#) will be placed at the scene root and positioned at the value of [CreateGameObjectNode.Position](#) in world space.

ObjectPositioning *Relative*

The created [GameObject](#) will be placed at the scene root and positioned at the value of [CreateGameObjectNode.Position](#) relative to [CreateGameObjectNode.Object](#).

ObjectPositioning *Child*

The created [GameObject](#) will be placed as a child of [CreateGameObjectNode.Parent](#) and positioned at the value of [CreateGameObjectNode.Position](#) in [CreateGameObjectNode.Parent's](#) coordinates.

ObjectSource

PiRhoSoft.CompositionEngine.ObjectSource

Description

Defines the available options for the *Source* property of [MenuItemTemplate](#).

Values

ObjectSource *Scene*

The [MenuItem](#) should be looked up by name in the loaded scenes.

ObjectSource *Asset*

The [MenuItem](#) should be instantiated from a [prefab](#).

ObjectVariableConstraint

PiRhoSoft.CompositionEngine.ObjectVariableConstraint : [VariableConstraint](#)

Description

A [VariableConstraint](#) for Object [VariableValues](#) that restricts the value to a specific type.

Public Fields

Type *Type*

The type the object must be or be derived from..

ObjectVariableSource

PiRhoSoft.CompositionEngine.ObjectVariableSource : [VariableSource](#)<[Object](#)>

Description

A [VariableSource](#) for Object [VariableValues](#).

Operation

PiRhoSoft.CompositionEngine.Operation

Description

The base class for all operations in an [Expression](#). Custom operations should derive from either [PrefixOperation](#) or [InfixOperation](#) rather than deriving from this class directly.

Public Methods

void Parse([ExpressionParser](#) parser, [ExpressionToken](#) token) (*abstract*)

Implement this in a subclass to initialize the operation by reading [ExpressionTokens](#) from *parser*. *token* is the [ExpressionToken](#) that led to the creation of this operation.

[VariableValue](#) Evaluate([IVariableStore](#) variables) (*abstract*)

Implement this in a subclass to perform the execution of the operation. Any [VariableReferences](#) should use *variables* for lookups and assignments.

void ToString([StringBuilder](#) builder) (*abstract*)

Writes a reversible representation of this operation to *builder*.

void GetInputs([IList](#)<[VariableDefinition](#)> inputs, [string](#) source) (*virtual*)

Implement this in a subclass to add [VariableDefinitions](#) to *inputs* that access an [IVariableStore](#) named *source*.

void GetOutputs([IList](#)<[VariableDefinition](#)> outputs, [string](#) source) (*virtual*)

Implement this in a subclass to add [VariableDefinitions](#) to *outputs* that write [VariableValues](#) to an [IVariableStore](#) named *source*.

OperatorPrecedence

PiRhoSoft.CompositionEngine.OperatorPrecedence : ValueType

Description

Specifies the necessary information to determine the evaluation order for different [Operations](#). Lower values will have lower precedence, meaning they will be evaluated first. The static values defined on this class follow the same precedence rules as math and other programming languages and are listed here in order of lowest precedence to highest.

Static Fields

[OperatorPrecedence](#) *Default*

This should be the precedence passed to [ExpressionParser.ParseLeft](#) when parsing a new statement or sub-statement.

[OperatorPrecedence](#) *Assignment*

The precedence for all assignment operations. This is right associative so assignments can be chained..

[OperatorPrecedence](#) *Ternary*

The precedence for a ternary (condition ? trueStatement : falseStatement) statement.

[OperatorPrecedence](#) *Or*

The precedence for a logical or.

[OperatorPrecedence](#) *And*

The precedence for a logical and.

[OperatorPrecedence](#) *Equality*

The precedence for an equality or inequality check.

[OperatorPrecedence](#) *Comparison*

The precedence for comparisons.

[OperatorPrecedence](#) *Addition*

The precedence for addition and subtraction.

[OperatorPrecedence](#) *Multiplication*

The precedence for multiplication and division.

[OperatorPrecedence](#) *Exponentiation*

The precedence for exponents.

[OperatorPrecedence](#) *Prefix*

The precedence for all prefix operations.

OperatorPrecedence *Postfix*

The precedence for all postfix operations.

OperatorPrecedence *MemberAccess*

The precedence for all member access operations.

Static Methods

OperatorPrecedence **LeftAssociative**(*int value*)

Creates a precedence with left associativity meaning operations with the same precedence will be evaluated left to right.

OperatorPrecedence **RightAssociative**(*int value*)

Creates a precedence with right associativity meaning operations with the same precedence will be evaluated right to left.

Public Properties

int *Value* (*read only*)

The precedence value when parsed standalone or as the left hand side of an [InfixOperation](#).

int *AssociativeValue* (*read only*)

The precedence value when parsed as the right hand side of an [InfixOperation](#).

Parameter

PiRhoSoft.CompositionEngine.Parameter

Description

Holds the name and [VariableType](#) of a parameter passed to a [Command](#).

Public Fields

string *Name*

The name the [Command](#) uses to reference the parameter in its [Expression](#).

VariableType *Type*

The [VariableType](#) the [Command](#) is expecting for the parameter.

ParameterList

PiRhoSoft.CompositionEngine.ParameterList : [SerializedList](#)<[Parameter](#)>

Description

A [SerializedList](#) for [CommandParameters](#).

PixelateTransition

PiRhoSoft.CompositionEngine.PixelateTransition : [Transition](#)

Description

Animates the resolution of the rendered image by making it more and more pixelated over time. The [material property](#) `_Amount` will be set to a number between 1 and `_MaxAmount_`, with the number incrementing (or decrementing if the [phase](#) is In) every frame.

Public Fields

int *MaxAmount*

The number of pixels for the dimension of the pixelation when the [Transition](#) is at its extreme.

PlayAnimationNode

PiRhoSoft.CompositionEngine.PlayAnimationNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to play an [AnimationClip](#) on an [AnimationPlayer](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *AnimationPlayer*

The [AnimationPlayer](#) to play *Animation* on.

[AnimationClipVariableSource](#) *Animation*

The [AnimationClip](#) to play on *AnimationPlayer*.

bool *WaitForCompletion*

If this is true, this node will not complete until *Animation* has completed. Otherwise, this node will complete immediately.

PlayAnimationStateNode

PiRhoSoft.CompositionEngine.PlayAnimationStateNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to activate a trigger using *SetTrigger* on an [Animator](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Animator*

The [Animator](#) to set *State* on.

[StringVariableSource](#) *State*

The name of the trigger to set on *Animator* using *SetTrigger*

PlaybackState

PiRhoSoft.CompositionEngine.PlaybackState

Description

Used internally by the editor to determine the current execution state of an [InstructionGraph](#).

Values

PlaybackState *Running*

The graph is running.

PlaybackState *Paused*

The graph has stopped at a breakpoint.

PlaybackState *Step*

The graph is running a single node before pausing again.

PlaybackState *Stopped*

The graph has been manually stopped.

PlayEffect

PiRhoSoft.CompositionEngine.PlayEffectNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to instantiate a [prefab](#) containing one or more [ParticleSystem](#)s or [ICompletionNotifiers](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[GameObjectVariableSource](#) *Effect*

The [prefab](#) to instantiate.

[StringVariableSource](#) *EffectName*

The name to assign to the instantiated [prefab](#).

[VariableReference](#) *EffectVariable*

The variable to assign the instantiated [prefab](#) to.

[ObjectPositioning](#) *Positioning*

The way the value of *Position* and *Rotation* should be interpreted.

[VariableReference](#) *Object*

When *Positioning* is Relative, specifies the object the created object should be positioned relative to.

[VariableReference](#) *Parent*

When *Positioning* is Child, specifies the object the created object should be added to as a child.

[Vector3VariableSource](#) *Position*

The position at which to place the newly created object.

[Vector3VariableSource](#) *Rotation*

The rotation to set the newly created object to.

bool *WaitForCompletion*

If this is true, this node will not complete until all [ParticleSystem](#)s and [ICompletionNotifiers](#) in *Effect* have completed. Otherwise, this node will complete immediately.

bool *DestroyOnComplete*

If this is true, the [GameObject](#) created from *Effect* will be destroyed when it finishes playing.

PlaySoundNode

PiRhoSoft.CompositionEngine.PlaySoundNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to play an [AudioClip](#) on an [AudioPlayer](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *AudioPlayer*

The [AudioPlayer](#) to play *Sound* on.

[AudioClipVariableSource](#) *Sound*

The [AudioClip](#) to play on *AudioPlayer*.

[FloatVariableSource](#) *Volume*

The volume to set on *AudioPlayer* when playing *Sound*.

bool *WaitForCompletion*

If this is true, this node will not complete until *Sound* has completed. Otherwise, this node will complete immediately.

PlayTimelineNode

PiRhoSoft.CompositionEngine.PlayTimelineNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to play a [TimelineAsset](#) on a [PlayableDirector](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Director*

The [PlayableDirector](#) to play *Timeline* on.

[TimelineVariableSource](#) *Timeline*

The [TimelineAsset](#) to play.

[DirectorWrapMode](#) *Mode*

The [DirectorWrapMode](#) to play *Timeline* with.

bool *WaitForCompletion*

If this is true, this node will not complete until *Timeline* has completed. Otherwise, this node will complete immediately.

PlayTransitionNode

PiRhoSoft.CompositionEngine.PlayTransitionNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to start a [Transition](#) on the [TransitionManager](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[TransitionVariableSource](#) *Transition*

The [Transition](#) to play.

[TransitionPhase](#) *Phase*

The [TransitionPhase](#) to play the [Transition](#) in.

bool *AutoFinish*

If this is true, the [Transition](#) will be ended as soon as it has completed. If this is false, the [Transition](#) will persist in its final state until another [Transition](#) (perhaps the same one with a different [TransitionPhase](#)) is started.

bool *WaitForCompletion*

If this is true, this node will not complete until *Transition* has completed. Otherwise, this node will complete immediately.

PrefixOperation

PiRhoSoft.CompositionEngine.PrefixOperation : [Operation](#)

Description

The base class for all [Operations](#) that have a right side.

Public Fields

string *Symbol*

The symbol for this operation.

Operation *Right*

The operation that makes up the right hand side.

Protected Methods

ExpressionEvaluationException **TypeMismatch**(**VariableType** *type*)

Creates an exception to be thrown by the caller indicating the operation cannot operate on a value with type *type*.

PrimaryAxis

PiRhoSoft.CompositionEngine.PrimaryAxis

Description

Defines the options available for the *PrimaryAxis* property of [MenuInput](#)

Values

[PrimaryAxis](#) *Column*

[MenuItems](#) are laid out in column order, meaning each [MenuItem](#) is visually below its predecessor before optionally wrapping to new columns.

[PrimaryAxis](#) *Row*

[MenuItems](#) are laid out in row order, meaning each [MenuItem](#) is visually to the right of its predecessor before optionally wrapping to new rows.

QuaternionVariableSource

PiRhoSoft.CompositionEngine.QuaternionVariableSource : [VariableSource](#)<[Quaternion](#)>

Description

A [VariableSource](#) for Quaternion [VariableValues](#).

Constructors

QuaternionVariableSource([Quaternion](#) *defaultValue*)

Initializes the source to *Type* Value with *Value* _defaultValue.

ReadOnlyStore

PiRhoSoft.CompositionEngine.ReadOnlyStore : [VariableStore](#)

Description

An [IVariableStore](#) implementation that disallows contained [VariableValues](#) to be assigned or added.

RectVariableSource

PiRhoSoft.CompositionEngine.RectVariableSource : [VariableSource](#)<[Rect](#)>

Description

A [VariableSource](#) for Rect [VariableValues](#).

Constructors

RectVariableSource([Rect](#) *defaultValue*)

Initializes the source to *Type* Value with *Value* *_defaultValue*.

ResetTagNode

PiRhoSoft.CompositionEngine.ResetTagNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to reset all [Variables](#) on an object implementing [IVariableReset](#) with a given tag. To reset a specific set of [Variables](#) use [ResetVariablesNode](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Object*

The [IVariableReset](#) to call *ResetTag* on.

string *Tag*

The tag to reset on *Object*.

ResetVariableList

PiRhoSoft.CompositionEngine.ResetVariableList : [SerializedList](#)<string>

Description

The list of variables for a [ResetVariablesNode](#).

ResetVariablesNode

PiRhoSoft.CompositionEngine.ResetVariablesNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) reset a specific set of [Variables](#) on an object implementing [IVariableReset](#). To reset [Variables](#) by tag use [ResetTagNode](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Object*

The [IVariableReset](#) to call *ResetVariables* on.

[ResetVariableList](#) *Variables*

The list of variable names that should be reset.

SceneReference

PiRhoSoft.UtilityEngine.SceneReference

Description

Add this as a field on a class to expose an editor selectable scene.

Static Fields

Action<string, string> *SceneMoved*

This is exposed for editor support and should be ignored.

Public Fields

string *Path*

The path to the scene in the "Assets" directory.

Public Properties

bool *IsAssigned (read only)*

true if the scene has been assigned.

bool *IsLoaded (read only)*

true if the scene is currently loaded.

Scene *Scene (read only)*

The [Scene](#) at *Path*.

int *Index (read only)*

The build index of the [Scene](#) at *Path*.

Public Methods

void Setup([Object](#) owner)

This should be called in the setup method of the class this is a field on (for example `OnEnable` if it is a [ScriptableObject](#)). This is necessary for the editor to update the scene when it is moved, renamed, or deleted.

void Teardown()

This should be called in the destroy method of the class this is a field on (for example `OnDisable` if it is a [ScriptableObject](#)). This is necessary for the editor to update the scene when it is moved, renamed, or deleted.

SceneReferenceAttribute

PiRhoSoft.UtilityEngine.SceneReferenceAttribute : Attribute

Description

Apply this to a [SceneReference](#) to indicate to the editor how scene creation should be handled. If this attribute is not applied to a [SceneReference](#) field, it will act as if *SaveLocation* is None.

Public Fields

[AssetLocation](#) *SaveLocation*

The folder to save the newly created scene in. Set this to None to disable scene creation.

string *DefaultName*

The name to assign to new scenes by default. If *SaveLocation* is Selectable, the name can be set when picking the save location. If *SaveLocation* is AssetRoot, this name will automatically have a number appended if a scene with the same name already exists.

string *Creator*

The name of a method on the owning object that can add objects to the newly created scene. The method should be static, return void, and take no parameters. The created scene will be the active scene so any objects created in this method will be added to the new scene. If this is not specified, an empty scene will be created.

SceneSource

PiRhoSoft.CompositionEngine.SceneSource

Description

Defines how the scene to load is retrieved in a [LoadSceneNode](#).

Values

SceneSource *Value*

The scene is specified directly in [LoadSceneNode.Scene](#).

SceneSource *Variable*

The scene is resolved from the [LoadSceneNode.SceneVariable](#) [VariableReference](#).

SceneSource *Name*

The scene is loaded by name as specified by [LoadSceneNode.SceneName](#).

SceneSource *Index*

The scene is loaded by build index as specified by [LoadSceneNode.SceneIndex](#).

SceneSource

PiRhoSoft.CompositionEngine.SceneSource

Description

Defines how the scene to unload is retrieved in an [UnloadSceneNode](#).

Values

SceneSource *Value*

The scene is specified directly in [UnloadSceneNode.Scene](#).

SceneSource *Variable*

The scene is resolved from the [UnloadSceneNode.SceneVariable](#) [VariableReference](#).

SceneSource *Name*

The scene is unloaded by name as specified by [UnloadSceneNode.SceneName](#).

SceneSource *Index*

The scene is unloaded by build index as specified by [UnloadSceneNode.SceneIndex](#).

SceneVariableStore

PiRhoSoft.CompositionEngine.SceneVariableStore : [IVariableStore](#)

Description

An [IVariableStore](#) implementation that allows the retrieval of [GameObjects](#) from the loaded scenes.

Public Methods

[VariableValue](#) GetVariable(string name)

Returns a [VariableValue](#) containing the [GameObject](#) with name *name*. The [GameObject](#) does not need to be enabled in order to access it with this method. If no [GameObject](#) is found with name *name*, [VariableValue.Empty](#) will be returned.

[SetVariableResult](#) SetVariable(string name, [VariableValue](#) value)

This will always return `ReadOnly`.

[IList<string>](#) GetVariableNames()

This will always return an empty list.

ScopedGraph

PiRhoSoft.CompositionEngine.ScopedGraph : [InstructionGraph](#)

Description

An [InstructionGraph](#) with an entry branch, a main branch, and an exit branch. These branches will run in sequence but for organization purposes it is useful to think of *Enter* as a setup branch and *Exit* a cleanup branch that reverses any changes made in *Enter*.

Public Fields

[InstructionGraphNode](#) *Enter*

The branch that will run when the [InstructionGraph](#) is first run.

[InstructionGraphNode](#) *Process*

The branch that will run after *Enter* has completed.

[InstructionGraphNode](#) *Exit*

The branch that will run after *Process* has completed.

SelectionControl

PiRhoSoft.CompositionEngine.SelectionControl : [InterfaceControl](#)

Description

Add this to a [Menu](#) with a [MenuInput](#) to allow a [MenuItem](#) to be selected.

Public Properties

bool *IsRunning (read only)*

This will be true when a selection is in progress.

bool *IsSelectionRequired (read only)*

This will be true if the current selection requires an item to be selected.

bool *IsClosing (read only)*

This will be true when the selection will be closed on the next frame.

bool *HasFocusedItem (read only)*

This will be true when the [Menu](#) has a focused [MenuItem](#).

bool *HasSelectedItem (read only)*

This will be true when a selection has been made.

MenuItem *FocusedItem (read only)*

The [MenuItem](#) that currently has focus, or null if there is no focused item.

int *FocusedIndex (read only)*

The index of the [MenuItem](#) that currently has focus, or -1 if there is no focused item.

VariableValue *FocusedValue (read only)*

The value associated with the [MenuItem](#) that currently has focus, or `VariableValue.Empty` if there is no focused item.

MenuItem *SelectedItem (read only)*

The [MenuItem](#) that has been selected, or null if no selection has been made.

int *SelectedIndex (read only)*

The index of the [MenuItem](#) that has been selected, or -1 if no selection has been made.

VariableValue *SelectedValue (read only)*

The value associated with the [MenuItem](#) that has been selected, or `VariableValue.Empty` if no selection has been made.

Public Methods

void Show(**IVariableStore** *variables*, **IEnumerable**<**MenuItemTemplate**> *items*, **bool** *isSelectionRequired*, **bool** *resetIndex*)

Show *items* on the sibling **Menu**. This will start a **coroutine** that waits for a selection to be made. If *isSelectionRequired* is true, the **Menu** will be required to have a selection made. If *resetIndex* is true, the **Menu**'s focus will be set to the first item, otherwise the focus will not change. *variables* is used with *items* to resolve any **VariableReferences**.

void Select(**MenuItem** *item*)

Makes *item* the selected item and closes the menu.

void Close()

Closes the menu. If *IsSelectionRequired* is true, this will only succeed if a selection has been made.

Protected Methods

Transform GetItemParent() (*virtual*)

Implement this in subclasses to specify the **Transform** that created items should be added to. By default this is the **Transform** of this object.

void OnInitialize() (*virtual*)

Implement this in subclasses to perform setup when *Show* is called after the items have been created and menu has been set up.

void OnCreate() (*virtual*)

Implement this in subclasses to perform setup when *Show* is called after the items have been created but before the menu has been set up.

IEnumerator Run() (*virtual*)

Implement this in subclasses to perform custom handling for waiting for a selection. The default implementation will do nothing but yield until the control closes.

SelectionNode

PiRhoSoft.CompositionEngine.SelectionNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to show a [SelectionControl](#) and retrieve a selection from it.

Public Fields

[InstructionGraphNode](#) *OnCanceled*

The [InstructionGraphNode](#) to run when the selection is cancelled.

[VariableReference](#) *Control*

The [SelectionControl](#) to perform the selection with.

[VariableReference](#) *SelectedItem*

The variable to store the selected item in.

[VariableReference](#) *SelectedIndex*

The variable to store the index of the selected item in.

bool *IsSelectionRequired*

If this is true, a selection must be made before the node will complete.

bool *AutoHide*

If this is true, *Control* will be hidden once a selection has been made.

[SelectionNodeItemList](#) *Items*

The list of [SelectionNodeItems](#) available to be selected.

SelectionNodeItem

PiRhoSoft.CompositionEngine.SelectionNodeItem : [MenuItemTemplate](#)

Description

The information for an item in a [SelectionNode](#).

Public Fields

[InstructionGraphNode](#) *OnSelected*

The [InstructionGraphNode](#) to run when this item is selected.

SelectionNodeItemList

PiRhoSoft.CompositionEngine.SelectionNodeItemList : [SerializedList](#)<[SelectionNodeItem](#)>

Description

A list of [SelectionNodeItems](#) used by [SelectionNode](#).

SequenceNode

PiRhoSoft.CompositionEngine.SequenceNode : [InstructionGraphNode](#), [ISequenceNode](#)

Description

Add this to an [InstructionGraph](#) to run a set of [InstructionGraphNodes](#) one after the other.

Public Fields

[InstructionGraphNodeList](#) *Sequence*

The list of [InstructionGraphNodes](#) to run.

SerializedDictionary

PiRhoSoft.UtilityEngine.SerializedDictionary<KeyType, KeyType> : [Dictionary](#)<KeyType, KeyType>

Description

Adds serialization and editor support to the built in [Dictionary](#) class.

Public Methods

void PrepareForEdit()

This is an editor support function and can be ignored.

void ApplyEdits()

This is an editor support function and can be ignored.

SerializedList

PiRhoSoft.UtilityEngine.SerializedList<T> : [ICollection](#), [IEnumerable<T>](#), [IList<T>](#), [IEnumerable](#), [IReadOnlyCollection<T>](#), [IReadOnlyList<T>](#), [IList](#), [ICollection<T>](#)

Description

Adds improved editor support to the built in [List](#) class.

SetAnimationParameterNode

PiRhoSoft.CompositionEngine.SetAnimationParameterNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to set a parameter on an [Animator](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[StringVariableSource](#) *Parameter*

The name of the parameter to set.

[AnimatorControllerParameterType](#) *Type*

The type of parameter to set.

[VariableReference](#) *Animator*

The [Animator](#) to set the parameter on.

[BoolVariableSource](#) *BoolValue*

If *Type* is [Bool](#), the value to set using [SetBool](#)

[IntVariableSource](#) *IntValue*

If *Type* is [Bool](#), the value to set using [SetInteger](#)

[FloatVariableSource](#) *FloatValue*

If *Type* is [Float](#), the value to set using [SetFloat](#)

SetBindingNode

PiRhoSoft.CompositionEngine.SetBindingNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to change the *Value* of a [BindingRoot](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Object*

The [BindingRoot](#) to change *Value* on.

[VariableReference](#) *Binding*

The [IVariableStore](#) to set as the *Value* on *Object*.

SetVariableResult

PiRhoSoft.CompositionEngine.SetVariableResult

Description

The result returned from calls to [IVariableStore.SetVariable](#) indicating if a [VariableValue](#) was set successfully or why it failed.

Values

SetVariableResult *Success*

The [VariableValue](#) was set.

SetVariableResult *NotFound*

The [VariableValue](#) was not set because it could not be found and values cannot be added.

SetVariableResult *ReadOnly*

The [VariableValue](#) was not set because it is not allowed to be changed.

SetVariableResult *TypeMismatch*

The [VariableValue](#) was not set because the [VariableType](#) is not allowed to be changed.

ShowControlNode

PiRhoSoft.CompositionEngine.ShowControlNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to show an [InterfaceControl](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Control*

The [InterfaceControl](#) to show.

ShuffleNode

PiRhoSoft.CompositionEngine.ShuffleNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to shuffle the [VariableValues](#) in an [IVariableList](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Variable*

The [IVariableList](#) to shuffle.

SimpleGraph

PiRhoSoft.CompositionEngine.SimpleGraph : [InstructionGraph](#)

Description

A basic [InstructionGraph](#) with a single branch.

Public Fields

[InstructionGraphNode](#) *Process*

The [InstructionGraphNode](#) to run when this graph runs.

SortConditionList

PiRhoSoft.CompositionEngine.SortConditionList : [SerializedList](#)<[VariableReference](#)>

Description

The list of [VariableReferences](#) used as conditions for a [SortNode](#).

SortNode

PiRhoSoft.CompositionEngine.SortNode : [InstructionGraphNode](#)

Description

Sorts the [VariableValues](#) in an [VariableList](#).

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *List*

The [VariableList](#) to sort.

bool *SortByProperty*

If this is true, *SortConditions* is used to sort the [VariableValues](#) by properties on each value. Otherwise the [VariableValues](#) are sorted directly.

[SortConditionList](#) *SortConditions*

The [Variables](#) on each item in *List* to sort by. When sorting by more than one property, the result will be fully sorted by the last property with equal values sorted by each previous property.

SpriteBinding

PiRhoSoft.CompositionEngine.SpriteBinding : [VariableBinding](#)

Description

Add this to a [SpriteRenderer](#) to bind *sprite* to a variable.

Public Fields

[VariableReference](#) *Variable*

The [Sprite](#) that will be applied when the binding is updated.

Public Properties

[SpriteRenderer](#) *Sprite (read only)*

The component to set the sprite on.

SpriteColorBinding

PiRhoSoft.CompositionEngine.SpriteColorBinding : [VariableBinding](#)

Description

Add this to a [SpriteRenderer](#) to bind *color* to a variable.

Public Fields

[VariableReference](#) *Variable*

The Color [VariableValue](#) that will be applied when the binding is updated.

Public Properties

[SpriteRenderer](#) *Sprite (read only)*

The component to set the color on.

StartGraphTrigger

PiRhoSoft.CompositionEngine.StartGraphTrigger : [InstructionTrigger](#)

Description

Add this to any [GameObject](#) to run an [InstructionGraph](#) when *Start* is called.

StopTransitionNode

PiRhoSoft.CompositionEngine.StopTransitionNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to end the [Transition](#) currently running on the [TransitionManager](#). If there is no [Transition](#) running this has no effect.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

StoreVariableConstraint

PiRhoSoft.CompositionEngine.StoreVariableConstraint : [VariableConstraint](#)

Description

A [VariableConstraint](#) for Store [VariableValues](#) that specifies the [Variables](#) that are available in the store.

Public Fields

[VariableSchema](#) *Schema*

The schema defining the [Variables](#) that are in the store.

StoreVariableSource

PiRhoSoft.CompositionEngine.StoreVariableSource : [VariableSource](#)<[IVariableStore](#)>

Description

A [VariableSource](#) for [IVariableStores](#).

StringBinding

PiRhoSoft.CompositionEngine.StringBinding : [VariableBinding](#)

Description

Derive from this class to implement a [VariableBinding](#) that sets the text on a [TextMeshPro](#).

Public Fields

bool *AutoSizeContainer*

Set this to true to set *autoSizeTextContainer* on *Text*. This property is otherwise not exposed to the editor, but is necessary in some situations to ensure an [Auto Layout](#) ui is sized correctly.

Public Properties

[TMP_Text](#) *Text (read only)*

The component to set the text on.

Protected Methods

void **SetText**(string *text*, bool *enabled*)

Call this from subclasses to set *Text*'s text to *text*. *Text* will also be enabled or disabled according to the *enabled* parameter.

StringVariableConstraint

PiRhoSoft.CompositionEngine.StringVariableConstraint : [VariableConstraint](#)

Description

A [VariableConstraint](#) for String [VariableValues](#) that restricts the value to one of a set of values.

Public Fields

string[] *Values*

The allowed values.

StringVariableSource

PiRhoSoft.CompositionEngine.StringVariableSource : [VariableSource](#)<string>

Description

A [VariableSource](#) for String [VariableValues](#).

Constructors

StringVariableSource(string *defaultValue*)

Initializes the source to *Type* Value with *Value* *_defaultValue*.

TagList

PiRhoSoft.CompositionEngine.TagList : [SerializedList](#)<string>

Description

The serializable list of tags in a [VariableSchema](#).

TextBinding

PiRhoSoft.CompositionEngine.TextBinding : [StringBinding](#)

Description

Add this to a [TextMeshPro](#) to bind the text to a variable.

Public Fields

VariableReference *Variable*

The [VariableValue](#) that will be converted to a string and applied when the binding is updated. To perform custom formatting for Int or Float [VariableValues](#) use [NumberBinding](#).

TextColorBinding

PiRhoSoft.CompositionEngine.TextColorBinding : [VariableBinding](#)

Description

Add this to a [TextMeshPro](#) to bind the text color to a variable.

Public Fields

VariableReference *Variable*

The Color [VariableValue](#) that will be applied when the binding is updated.

Public Properties

TMP_Text *Text (read only)*

The component to set the color on.

TextInputBinding

PiRhoSoft.CompositionEngine.TextInputBinding : [VariableBinding](#)

Description

Add this to a [TMP_InputField](#) to apply entered text to a variable.

Public Fields

[VariableReference](#) *Variable*

The variable to apply the text to when it changes.

Public Properties

[TMP_InputField](#) *Text (read only)*

The component to get the text from.

TimeFormatType

PiRhoSoft.CompositionEngine.TimeFormatType

Description

Defines the time formats available to set for the *TimeFormatting* of a [BindingFormatter](#).

Values

TimeFormatType *SecondsMilliseconds*

The number will be printed in seconds and milliseconds. Equivalent to setting the [custom format string](#) to "s\\.fff".

TimeFormatType *MinutesSeconds*

The number will be printed in minutes and seconds. Equivalent to setting the [custom format string](#) to "m\\:ss".

TimeFormatType *MinutesSecondsMilliseconds*

The number will be printed in minutes, seconds, and milliseconds. Equivalent to setting the [custom format string](#) to "m\\:ss\\.fff".

TimeFormatType *HoursMinutes*

The number will be printed in hours and minutes. Equivalent to setting the [custom format string](#) to "h\\:mm".

TimeFormatType *Custom*

The format string will be read from the *ValueFormat* property of the [BindingFormatter](#).

TimelineVariableSource

PiRhoSoft.CompositionEngine.TimelineVariableSource : [VariableSource](#)<[TimelineAsset](#)>

Description

A [VariableSource](#) for Object [VariableValues](#) that must be [TimelineAssets](#).

TimeScaleNode

PiRhoSoft.CompositionEngine.TimeScaleNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to set *timeScale*.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[FloatVariableSource](#) *TimeScale*

The value to set *timeScale* to.

TransformNode

PiRhoSoft.CompositionEngine.TransformNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to animate the [Transform](#) of a [GameObject](#).

Public Fields

[InstructionGraphNode](#) *Next*

Add this to an [InstructionGraph](#) to run an [Expression](#).

[VariableReference](#) *Transform*

The [Transform](#) to animate.

bool *UseRelativePosition*

If this is true, *TargetPosition* will be added to the position of the [Transform](#) when the node starts. Otherwise, *TargetPosition* will be used directly.

bool *UseRelativeRotation*

If this is true, *TargetRotation* will be added to the rotation of the [Transform](#) when the node starts. Otherwise, *TargetRotation* will be used directly.

bool *UseRelativeScale*

If this is true, *TargetScale* will be multiplied with the scale of the [Transform](#) when the node starts. Otherwise, *TargetScale* will be used directly.

[Vector3VariableSource](#) *TargetPosition*

The position to move *Transform* toward.

[Vector3VariableSource](#) *TargetRotation*

The rotation to rotate *Transform* toward.

[Vector3VariableSource](#) *TargetScale*

The size to scale *Transform* toward.

[AnimationType](#) *AnimationMethod*

Specifies the advancement method of the animation.

bool *WaitForCompletion*

If this is true, this node will not complete until the animation has completed. Otherwise, this node will complete immediately.

[FloatVariableSource](#) *Duration*

If *AnimationMethod* is *Duration*, the number of seconds the animation will take.

FloatVariableSource *MoveSpeed*

If *AnimationMethod* is Speed, the number of units per second to move the [Transform](#).

FloatVariableSource *RotationSpeed*

If *AnimationMethod* is Speed, the number of radians per second to rotate the [Transform](#).

FloatVariableSource *ScaleSpeed*

If *AnimationMethod* is Speed, the number of units per second to scale the [Transform](#).

Transition

PiRhoSoft.CompositionEngine.Transition : [ScriptableObject](#)

Description

The base class for assets that perform postprocessing of the rendered scene over a time period.

Public Fields

Shader *Shader*

The [Shader](#) that the transition will use to display its effect

float *Duration*

The time in seconds the transition should last.

Public Methods

void Begin([TransitionPhase](#) *phase*) (*virtual*)

Implement this to setup properties when the transition is started.

void Process(**float** *time*, [TransitionPhase](#) *phase*) (*virtual*)

Implement this to animate properties of the transition as time advances.

void End() (*virtual*)

Implement this to perform any clean up of the transition.

void Render([RenderTexture](#) *source*, [RenderTexture](#) *destination*) (*virtual*)

Renders the transition using *source* as the input scene and *destination* as the target. The [Graphics.Blit](#) methods are used to copy the texture using *Material*. To fully customize rendering, this can be overridden, but for most situations updating properties of *Material* in *Update* is sufficient.

Protected Properties

Material *Material* (*read only*)

The [Material](#) the effect will be rendered with. This is created with a call to *SetShader*.

Protected Methods

void SetShader(**string** *name*)

Creates the material using the specified shader. *name* is the name set for the shader at the beginning of the shader script. This should be called from subclasses during initialization.

void Update() (*virtual*)

Implement this to update the [material properties](#) of *Material*.

TransitionList

PiRhoSoft.CompositionEngine.TransitionList : [SerializedList](#)<[Transition](#)>

Description

A serializable list of [Transitions](#).

TransitionManager

PiRhoSoft.CompositionEngine.TransitionManager : [GlobalBehaviour](#)<[TransitionManager](#)>

Description

Manages the loaded [TransitionRenderers](#) for playback of [Transitions](#). This is created on demand and should not be added to a scene.

Public Properties

[Transition](#) *CurrentTransition (read only)*

The [Transition](#) that is currently running, or null if no [Transition](#) is running.

Public Methods

[IEnumerator](#) **RunTransition**([Transition](#) transition, [TransitionPhase](#) phase)

Runs *transition* in [TransitionPhase](#) *phase* and ends it when it has completed - *EndTransition* will not need to be called.



If a [Transition](#) is already running, it will be ended.

[IEnumerator](#) **StartTransition**([Transition](#) transition, [TransitionPhase](#) phase)

Runs *transition* in [TransitionPhase](#) *phase*. *EndTransition* (or a subsequent call to *StartTransition*) should be called manually later.



If a [Transition](#) is already running, it will be ended.

void EndTransition()

Ends the currently running [Transition](#) if one is running.

TransitionPhase

PiRhoSoft.CompositionEngine.TransitionPhase

Description

Defines the phases of a [Transition](#) to allow a [Transition](#) to perform differently depending on how it is being used.

Values

[TransitionPhase](#) *Out*

The [Transition](#) should transition away from the rendered scene into its obscured state (fade out for example).

[TransitionPhase](#) *Obscure*

The [Transition](#) should obscure the rendered scene for an indeterminate amount of time while the loaded content is changing.

[TransitionPhase](#) *In*

The [Transition](#) should transition from its obscured state into the rendered scene (fade in for example).

TransitionRenderer

PiRhoSoft.CompositionEngine.TransitionRenderer : [MonoBehaviour](#)

Description

Add this to a [Camera](#) to have any running [Transition](#) include the cameras rendered output in its post processing.

TransitionVariableSource

PiRhoSoft.CompositionEngine.TransitionVariableSource : [VariableSource](#)<[Transition](#)>

Description

A [VariableSource](#) for Object [VariableValues](#) that must be [Transitions](#).

UnloadSceneNode

PiRhoSoft.CompositionEngine.UnloadSceneNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to unload a scene.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[SceneSource](#) *Source*

Specifies how the scene to unload is retrieved.

[SceneReference](#) *Scene*

If *Source* is Value, holds the scene to unload.

[VariableReference](#) *SceneVariable*

If *Source* is Variable, references the scene to unload. If the resolved value is an Int, the scene will be unloaded by index. If it is a String, it will be unloaded by name.

string *SceneName*

If *Source* is Name, the name of the scene to unload.

int *SceneIndex*

If *Source* is Index, the build index of the scene to unload.

bool *WaitForCompletion*

If this is true, this node will not complete until the scene has completed unloading. Otherwise, this node will complete immediately and the [InstructionGraph](#) will continue.

bool *CleanupAssets*

If this is true (the default), [Resources.UnloadUnusedAssets](#) will be called after the scene has been unloaded.

UpdateBindingNode

PiRhoSoft.CompositionEngine.UpdateBindingNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to update the [VariableBindings](#) on a [GameObject](#) and its descendants.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[VariableReference](#) *Object*

The [GameObject](#) containing the [VariableBindings](#) to update.

string *Group*

The *BindingGroup* of [VariableBindings](#) to update. If this is empty, all [VariableBindings](#) in *Object* will be updated.

bool *WaitForCompletion*

If this is true, this node will not complete until any animated bindings have finished animating. Otherwise, this node will complete immediately.

ValueDefinition

PiRhoSoft.CompositionEngine.ValueDefinition : ValueType

Description

Defines properties for how a [VariableValue](#) is initialized and can be used, usually as part of a [VariableSchema](#). The *Generate* method can be used to create a [VariableValue](#) that satisfies the properties of the definition.

Static Methods

ValueDefinition Create(VariableType type)

Creates a definition for a [VariableValue](#) with [type](#) *type* and no other constraints.

ValueDefinition Create(int minimum, int maximum)

Creates a definition for a [VariableValue](#) with [type](#) *Int* and with a constraint restricting it to values between *minimum* and *maximum*.

ValueDefinition Create(float minimum, float maximum)

Creates a definition for a [VariableValue](#) with [type](#) *Float* and with a constraint restricting it to values between *minimum* and *maximum*.

ValueDefinition Create(string[] values)

Creates a definition for a [VariableValue](#) with [type](#) *String* and with a constraint restricting it to one of the values in *values*.

ValueDefinition Create<T>()

Creates a definition for a [VariableValue](#) with [type](#) *Object* whose value can be set to any [Object](#) that is type *T* or is derived from type *T*.

ValueDefinition Create(Type type)

Creates a definition for a [VariableValue](#) with [type](#) determined by [VariableValue.GetType](#).

ValueDefinition Create(VariableType type, VariableConstraint constraint)

Creates a definition for a [VariableValue](#) with [type](#) *type* and [VariableConstraint](#) *constraint*.



The caller must ensure *constraint* is a [VariableConstraint](#) of the appropriate type for *type*.

ValueDefinition Create(VariableType type, VariableConstraint constraint, string tag, Expression initializer, bool isTypeLocked, bool isConstraintLocked)

Creates a definition with the given properties.



The caller must ensure *constraint* is a [VariableConstraint](#) of the appropriate type for *type*.

Public Properties

VariableType *Type (read only)*

The **VariableType** to assign to **VariableValues** created by *Generate* or to test against with *IsValid*. If this is Empty, the **VariableValue** can be any type.

VariableConstraint *Constraint (read only)*

The **VariableConstraint** laying out *Type* dependent requirements for **VariableValues** using this definition. This can be null, and in that case, the only constraint for **VariableValues** will be *Type*.

string Tag *(read only)*

An arbitrary string that can be used to group different definitions in the same **VariableSchema**. This has no impact on the validity of a **VariableValue** or how it is generated. It is most commonly used to indicate a set of variables that should be saved (for runtime saves) or group variables for resetting to their defaults (i.e by **VariableSetComponent.ResetTag**) or any other class that implements **IVariableReset.ResetTag**).

Expression *Initializer (read only)*

The **Expression** to evaluate when calling *Generate* to determine the initial value of the **VariableValue**. If the **Expression** is not set, a **VariableValue** with the default for *Type* will be generated.

bool IsTypeLocked *(read only)*

Indicates to the editor that this definition cannot have *Type* changed.

bool IsConstraintLocked *(read only)*

Indicates to the editor that this definition cannot have *Constraint* changed. The properties of *Constraint* can be changed, but the *Constraint* itself cannot.

Public Methods

VariableValue Generate(**IVariableStore** variables)

Generates a **VariableValue** that satisfies the constraints laid out by this definition with initial value determined by *Initializer*.

bool IsValid(**VariableValue** value)

Returns true if *value* satisfies the runtime constraints specified by this definition.

ValueDefinitionList

PiRhoSoft.CompositionEngine.ValueDefinitionList : [SerializedList](#)<[ValueDefinition](#)>

Description

A serializable list of [ValueDefinitions](#).

Variable

PiRhoSoft.CompositionEngine.Variable : ValueType

Description

Associates a name with a [VariableValue](#).

Static Properties

[Variable](#) *Empty* (read only)

A [Variable](#) with an empty *Name* and *Value* with [VariableType](#) *Empty*.

Static Methods

[Variable](#) **Create**(string *name*, [VariableValue](#) *value*)

Creates a [Variable](#) with *Name* *name* and *_Value* *value*.

Public Properties

string *Name* (read only)

The name assigned to the variable.

[VariableValue](#) *Value* (read only)

The value assigned to the variable.

VariableBinding

PiRhoSoft.CompositionEngine.VariableBinding : [MonoBehaviour](#)

Description

Derived from this class to provide support for automatically updating properties of loaded [GameObjects](#) (for instance, user interface elements) based on [VariableValues](#) stored in the [Variables System](#).

Read the [Bindings Topic](#) for a complete overview of how to use bindings and how to implement custom bindings.

Static Methods

void UpdateBinding([GameObject](#) obj, string group, [BindingAnimationStatus](#) status)

Triggers an update for bindings on *obj* and its descendants. If *group* is null or empty, all bindings will be updated, otherwise all bindings with *BindingGroup* matching *group* will be updated. Optionally pass a [BindingAnimationStatus](#) instance as *status* to access information about bindings that perform an animation or otherwise take multiple frames to complete.

void UpdateBinding([GameObject](#) obj, string group, [BindingAnimationStatus](#) status, List<[VariableBinding](#)> bindings)

Performs the same function as the other *UpdateBinding* method but uses *bindings* as a location to store the [VariableBindings](#) looked up on *obj*. It is not necessary to use this overload exception when called from the *UpdateBinding* instance method of a [VariableBinding](#) subclass.

Public Fields

string BindingGroup

An arbitrary string used to allow the binding to be targeted by calls to *UpdateBinding*. This has two common uses: for performance, if a [GameObject](#) has many bindings that don't all need to be updated at the same time, different bindings can be updated individually or as a group. And, if the value behind a binding is updated but that update shouldn't be indicated to the player until some point in the future, the update can be deferred until that time.

bool AutoUpdate

If this is true, the binding will be updated every frame, thus always keeping it up to date with the [VariableValues](#) it is bound to.

bool SuppressErrors

If this is true, failure to resolve [VariableReferences](#) when updating the binding will be considered a valid condition and therefore not log error messages.



Errors when executing [Expressions](#) will not be suppressed. This is because [Expressions](#) handle errors with exceptions which result in allocations.

Public Properties

IVariableStore *Variables (read only)*

Returns the **IVariableStore** to use to resolve **VariableReferences** for this binding. The **IVariableStore** will be found using **BindingRoot.FindParent**.

Public Methods

void UpdateBinding(string group, BindingAnimationStatus status)

Use this method to update this specific binding when *group* is either null, empty, or matches *BindingGroup*. To update all bindings on an **GameObject**, use the static *UpdateBinding* method instead. Optionally pass a **BindingAnimationStatus** instance as *status* to access information about bindings that perform an animation or otherwise take multiple frames to complete.

Protected Methods

void UpdateBinding(IVariableStore variables, BindingAnimationStatus status) (abstract)

Implement this in subclasses to perform the binding. *variables* is the **IVariableStore** **VariableReferences** should be looked up with. For bindings that take multiple frames to complete, *status* should be updated to indicate when the binding has started and finished.



status will always be a valid instance so does not need to be checked for null.

Resolve

This collection of methods will lookup the value referenced by a **VariableReference**. The resolved value is set to the output parameter *result* and the return value will indicate whether the value was resolved successfully. The *variables* parameter should be the *variables* parameter passed to the *UpdateBinding* method. If the resolution fails, either due to the variable not being found or it being an invalid type, a warning will be printed to the **Console**.

- **bool Resolve(IVariableStore variables, VariableReference reference, VariableValue result (out))**
- **bool Resolve(IVariableStore variables, VariableReference reference, bool result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, int result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, float result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, Vector2Int result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, Vector3Int result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, RectInt result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, BoundsInt result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, Vector2 result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, Vector3 result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, Vector4 result (out))::**
- **bool Resolve(IVariableStore variables, VariableReference reference, Quaternion result (out))::**

- `bool Resolve(IVariableStore variables, VariableReference reference, Rect result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Bounds result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, Color result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, string result (out))::`
- `bool Resolve<EnumType>(IVariableStore variables, VariableReference reference, EnumType result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, IVariableStore result (out))::`
- `bool Resolve(IVariableStore variables, VariableReference reference, IVariableList result (out))::`
- `bool ResolveObject<ObjectType>(IVariableStore variables, VariableReference reference, ObjectType result (out))::`
- `bool ResolveStore<StoreType>(IVariableStore variables, VariableReference reference, StoreType result (out))::`
- `bool ResolveList<ListType>(IVariableStore variables, VariableReference reference, ListType result (out))::`
- `bool ResolveInterface<InterfaceType>(IVariableStore variables, VariableReference reference, InterfaceType result (out))::`
- `bool ResolveReference(IVariableStore variables, VariableReference reference, Object result (out))::`

void Assign(IVariableStore variables, VariableReference reference, VariableValue value)

Assigns *value* to the variable referenced by *reference*. The *variables* parameter should be the *variables* parameter passed to the *UpdateBinding* method. If the assignment fails, a warning will be logged.

VariableConstraint

PiRhoSoft.CompositionEngine.VariableConstraint

Description

VariableConstraint is the base class for constraints applied to [ValueDefinitions](#). For each relevant [VariableType](#) an implementation of this class is provided. These are:

Type	Constraint
Enum	EnumVariableConstraint
Float	FloatVariableConstraint
Int	IntVariableConstraint
List	ListVariableConstraint
Object	ObjectVariableConstraint
Store	StoreVariableConstraint
String	StringVariableConstraint

Public Methods

bool IsValid([VariableValue](#) value) (*abstract*)

Returns true if *value* satisfies the rules of this constraint.

VariableConstraintAttribute

PiRhoSoft.CompositionEngine.VariableConstraintAttribute : Attribute

Description

Add this to a [VariableReference](#) or [VariableSource](#) field to inform the editor of the type of [VariableValue](#) the code is expecting, thereby providing a more useful editor for the field.

Constructors

VariableConstraintAttribute([VariableType](#) type)

Specifies the corresponding field should have [VariableType](#) type.

VariableConstraintAttribute(int *minimum*, int *maximum*)

Specifies the corresponding field should have [VariableType](#) Int and value between *minimum* and *maximum*.

VariableConstraintAttribute(float *minimum*, float *maximum*)

Specifies the corresponding field should have [VariableType](#) Float and value between *minimum* and *maximum*.

VariableConstraintAttribute(string[] *values*)

Specifies the corresponding field should have [VariableType](#) String and value one of the options in *values*.

VariableConstraintAttribute(Type type)

Specifies the corresponding field should have [VariableType](#) as determined by [VariableValue.GetType](#).

VariableDefinition

PiRhoSoft.CompositionEngine.VariableDefinition : ValueType

Description

Extends a [ValueDefinition](#) by associating it with a name. All of the heavy lifting for for constraining [VariableValues](#) is provided by [ValueDefinition](#).

Public Fields

string *Name*

The name of *Definition*.

ValueDefinition *Definition*

The [ValueDefinition](#) being given *Name*.

VariableDefinitionList

PiRhoSoft.CompositionEngine.VariableDefinitionList : [SerializedList](#)<[VariableDefinition](#)>

Description

A serializable list of [VariableDefinitions](#).

VariableHandler

PiRhoSoft.CompositionEngine.VariableHandler

Description

A utility class for working with [VariableValues](#) independent of their [VariableType](#).

Static Methods

string ToString([VariableValue](#) value)

Returns the string representation of *value* depending on the [VariableType](#) it is holding.

[VariableValue](#) CreateDefault([VariableType](#) type, [VariableConstraint](#) constraint)

Creates and returns a variable with type *type* whose value meets the requirements of *constraint*. If *constraint* is null, the default value for type *type* is returned.

void ToString([VariableValue](#) value, [StringBuilder](#) builder)

Appends the string representation of *value* to *builder*.

[List](#)<string> SaveVariables([IList](#)<[Variable](#)> variables, [List](#) objects (ref))

Converts the [Variables](#) in *variables* to a representation that can be serialized by Unity. The returned list of strings, as well as *objects*, should be assigned to serializable fields an an object.

[List](#)<[Variable](#)> LoadVariables([List](#) data (ref), [List](#) objects (ref))

Creates a set of variables from the serialized representation in *data* and *objects*. *data* and *objects* will be cleared.

string SaveVariable([Variable](#) variable, [List](#) objects (ref))

Converts *variable* to a representation that can be serialized by Unity. The returned string and *objects* should be assigned to serializable fields on an object.

[Variable](#) LoadVariable(string data (ref), [List](#) objects (ref))

Creates a variable from the serialized representation in *data* and *objects*. *data* and *objects* will be cleared.

string SaveValue([VariableValue](#) value, [List](#) objects (ref))

Converts *value* to a representation that can be serialized by Unity. The returned string and *objects* should be assigned to serializable fields on an object.

[VariableValue](#) LoadValue(string data (ref), [List](#) objects (ref))

Creates a value from the serialized representation in *data* and *objects*. *data* and *objects* will be cleared.

string SaveConstraint([VariableType](#) type, [VariableConstraint](#) constraint, [List](#) objects (ref))

Saves *constraint* with type *type* to a representation that can be serialized by Unity. The returned string and *objects* should be assigned to serializable fields on an object.

VariableConstraint LoadConstraint(string *data* (ref), **List** *objects* (ref))

Creates a constraint from the serialized representation in *data* and *objects*. *_data* and *objects* will be cleared.

VariableValue Add(**VariableValue** *left*, **VariableValue** *right*)

Returns the result of adding *left* to *right*. If the result cannot be computed due to invalid or incompatible types, **VariableValue.Empty** will be returned.

VariableValue Subtract(**VariableValue** *left*, **VariableValue** *right*)

Returns the result of subtracting *right* from *left*. If the result cannot be computed due to invalid or incompatible types, **VariableValue.Empty** will be returned.

VariableValue Multiply(**VariableValue** *left*, **VariableValue** *right*)

Returns the result of multiplying *left* and *right*. If the result cannot be computed due to invalid or incompatible types, **VariableValue.Empty** will be returned.

VariableValue Divide(**VariableValue** *left*, **VariableValue** *right*)

Returns the result of dividing *left* by *right*. If the result cannot be computed due to invalid or incompatible types, **VariableValue.Empty** will be returned.

VariableValue Modulo(**VariableValue** *left*, **VariableValue** *right*)

Returns the remainder of dividing *left* by *right*. If the result cannot be computed due to invalid or incompatible types, **VariableValue.Empty** will be returned.

VariableValue Exponent(**VariableValue** *left*, **VariableValue** *right*)

Returns the result of raising *left* to the *right* power. If the result cannot be computed due to invalid or incompatible types, **VariableValue.Empty** will be returned.

VariableValue Negate(**VariableValue** *value*)

Returns the result of inverting *value*. If the result cannot be computed due to invalid or incompatible types, **VariableValue.Empty** will be returned.

Nullable<bool> IsEqual(**VariableValue** *left*, **VariableValue** *right*)

Returns true if *left* and *right* are equal, false if *left* and *right* can be legally compared but are not equal, and null if the types cannot be compared.

Nullable<int> Compare(**VariableValue** *left*, **VariableValue** *right*)

Returns -1 if *left* is less than *right*, 1 if *left* is greater than *right*, 0 if *left* is equal to *right*, and null if the types cannot be compared.

VariableValue Lookup(**VariableValue** *owner*, **VariableValue** *lookup*)

Returns a value contained in *owner* based on the value of *lookup*. If *lookup* is not found, **VariableValue.Empty** will be returned.

SetVariableResult Apply(**VariableValue** *owner* (ref), **VariableValue** *lookup*, **VariableValue** *value*)

Sets a value contained in *owner* based on *lookup* to *value*. *owner* will be updated to the new value. If *owner* is not holding a reference as determined by **VariableValue.IsReference** it must be reassigned to the container it is being held in. The return value indicates success or the reason

for failure.

VariableValue Cast(VariableValue owner, string type)

Returns the value of *owner* converted to type *type*. For object values, this is used to lookup sibling components.

bool Test(VariableValue owner, string type)

Determines if a *Cast* to *type* would be successful.

VariableInitializerType

PiRhoSoft.CompositionEngine.VariableInitializerType

Description

Defines the options for how a [VariableValue](#) will be initialized when created from a [ValueDefinition](#) in a [VariableSchema](#).

Values

VariableInitializerType *Expression*

The [VariableValue](#) will be initialized with the result of an [Expression](#).

VariableInitializerType *DefaultValue*

The [VariableValue](#) will be initialized to a specific value set in the editor.

VariableInitializerType *None*

The [VariableValue](#) will be initialized to the default value for its type.

VariableLink

PiRhoSoft.CompositionEngine.VariableLink : [MonoBehaviour](#)

Description

Add this to any [GameObject](#) to add a custom set of [Variables](#) to [CompositionManager.Global](#).

Public Fields

[VariablePool](#) *Variables*

The [Variables](#) to add to [CompositionManager.Global](#) while this [MonoBehaviour](#) is enabled.

VariableList

PiRhoSoft.CompositionEngine.VariableList : [IVariableList](#)

Description

An implementation of [IVariableList](#) that has no constraints on the [ValueValues](#) it can hold.

Constructors

VariableList(int count)

Adds *count* [Empty ValueValues](#) to the list.

Public Properties

List<VariableValue> Values (read only)

The [VariableValues](#) in the list.

int Count (read only)

The number of [VariableValues](#) in the list.

Public Methods

VariableValue GetVariable(int index)

Returns the [VariableValue](#) at index *index* in the list. If *index* is not between 0 and *Count*, [VariableValue.Empty](#) will be returned.

SetVariableResult AddVariable(VariableValue value)

Adds *value* to the list. This will always succeed and return [Success](#).

SetVariableResult RemoveVariable(int index)

Removes the [VariableValue](#) at index *index* from the list.

SetVariableResult SetVariable(int index, VariableValue value)

Changes the [VariableValue](#) at index *index* to *value*.

VariablePool

PiRhoSoft.CompositionEngine.VariablePool : [VariableStore](#)

Description

An [IVariableStore](#) that allows an arbitrary set of [VariableValues](#) to be added in the editor with a [ValueDefinition](#).

Public Fields

[List<ValueDefinition>](#) *Definitions*

Provides the editor access to the definitions. This should not be accessed at runtime.

Public Methods

void ChangeName(int index, string name)

This is an editor support function that can be ignored.

void ChangeDefinition(int index, [ValueDefinition](#) definition)

This is an editor support function that can be ignored.

[SetVariableResult](#) SetVariable(int index, [VariableValue](#) value)

This is an editor support function that can be ignored.

VariablePoolAsset

PiRhoSoft.CompositionEngine.VariablePoolAsset : [ScriptableObject](#), [IVariableStore](#)

Description

An asset for storing an arbitrary set of [Variables](#) using a [VariablePool](#).

Public Fields

[VariablePool](#) *Variables*

The [Variables](#) stored by this asset.

Public Methods

[VariableValue](#) **GetVariable**(string *name*)

The names of all the [Variables](#) stored by this asset.

[SetVariableResult](#) **SetVariable**(string *name*, [VariableValue](#) *value*)

Returns the [VariableValue](#) with name *name*.

[IList<string>](#) **GetVariableNames**()

Sets the [VariableValue](#) with name *name* to *_value*.

VariablePoolComponent

PiRhoSoft.CompositionEngine.VariablePoolComponent : [MonoBehaviour](#), [IVariableStore](#)

Description

Add this to any [GameObject](#) to define an arbitrary set of [Variables](#) using a [VariablePool](#).

Public Fields

[VariablePool](#) *Variables*

The [Variables](#) stored by this behaviour.

Public Methods

[IList<string>](#) **GetVariableNames()**

The names of all the [Variables](#) stored by this behaviour.

[VariableValue](#) **GetVariable(string name)**

Returns the [VariableValue](#) with name *name*.

[SetVariableResult](#) **SetVariable(string name, [VariableValue](#) value)**

Sets the [VariableValue](#) with name *name* to *_value*.

VariableReference

PiRhoSoft.CompositionEngine.VariableReference

Description

Specifies the name and location of a [VariableValue](#) for lookup or assignment. Read the [Accessing Variables](#) topic for more information.

Static Fields

string *Cast*

The text to use in *Variable* to lookup a sibling [Component](#) when referencing a [Component](#) or [GameObject](#). This is set to `as`.

Char *Separator*

The character to use to separate the variable names in *Variable*. This is set to `'.'`.

Char *LookupOpen*

The character to use in *Variable* to specify an index. This is set to `'['`.

Char *LookupClose*

The character to use in *Variable* after specifying an index. This is set to `']'`.

Public Properties

bool *IsValid (read only)*

Returns true if *Variable* contains a valid statement. This only verifies the syntax, it does not check if the variable exists.

bool *IsAssigned (read only)*

Returns true if *Variable* has been assigned regardless of if it's valid or not.

string *StoreName (read only)*

The first part of *Variable* (i.e the section before the first *Separator*).

string *RootName (read only)*

The second part of *Variable* (i.e the section between the first and second *Separator*).

string *Variable*

The reference to the [VariableValue](#).

Public Methods

VariableValue *GetValue(IVariableStore variables)*

Returns the referenced [VariableValue](#) by looking up *Variable* on *variables*.

SetVariableResult SetValue(**IVariableStore** *variables*, **VariableValue** *value*)

Sets the referenced **VariableValue** by looking up *Variable* on *variables* and assigning it *value*.

VariableSchema

PiRhoSoft.CompositionEngine.VariableSchema : [ScriptableObject](#)

Description

A VariableSchema is used to define the variables that are available to a [variable store](#) object - usually a [VariableSetComponent](#) or [VariableSetAsset](#). 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.

Public Fields

[VariableInitializerType](#) *InitializerType*

Specifies how the initializer for each [ValueDefinition](#) will be displayed in the editor and ultimately how [VariableValues](#) created by this schema will be initialized.

[TagList](#) *Tags*

Specifies the set of tags that can be selected in the editor for each [ValueDefinition](#) added to this schema.

Public Properties

int *Version (read only)*

The current version of the schema. This is incremented every time any change is made to the schema so objects using it know to update themselves. These updates are automatically managed by [VariableSetComponent](#) and [VariableSetAsset](#) and any class derived from them.

int *Count (read only)*

The number of [VariableDefinitions](#) that have been added to this schema.

[VariableDefinition](#) **this**[**int** *index*]

Sets or returns the [VariableDefinition](#) at index *index*.



[VariableDefinition](#) is a struct so any changes made to the returned [definition](#) will not change the actual schema. Reassign the [definition](#) using this indexer to apply the change.

Public Methods

int **GetIndex**(string *name*)

Returns the index of the [VariableDefinition](#) with *Name* *name* or -1 if no [VariableDefinition](#) has been added with that name.



Variable names are case sensitive.

bool HasDefinition(string *name*)

Returns true if this schema has a [VariableDefinition](#) with *Name* *name*.

bool AddDefinition(string *name*, [VariableType](#) *type*)

Adds a new [VariableDefinition](#) to the schema with *Name* *name* and *Type* *type*. If a [definition](#) with that name has already been added, nothing will happen and this method will return false. If the definition is successfully added this method will return true.

void RemoveDefinition(int *index*)

Removes the [VariableDefinition](#) at index *index* from this schema.

VariableSet

PiRhoSoft.CompositionEngine.VariableSet : [IVariableReset](#)

Description

Holds a serializable list of [Variables](#) that are defined by a [VariableSchema](#). This is most often used with a [MappedVariableStore](#).

Public Properties

[VariableSchema](#) *Schema (read only)*

The [VariableSchema](#) that defines the [Variables](#) in the set.

[IVariableStore](#) *Owner (read only)*

The [IVariableStore](#) that this is a member of.

bool *NeedsUpdate (read only)*

This will be true if *Schema* has changed since the last time this set was updated.

int *VariableCount (read only)*

The number of [Variables](#) in the set.

Public Methods

void LoadFrom([VariableSet](#) variables, string tag)

Copy all the [Variables](#) in *variables* with *Tag tag* into this set. This is for runtime persistence of game state.

void SaveTo([VariableSet](#) variables, string tag)

Copy all the [Variables](#) in this set with *Tag tag* into *variables*. This is for runtime persistence of game state.

void Setup([VariableSchema](#) schema, [IVariableStore](#) owner)

Associate this set with *schema* and *owner*. If *schema* has changed since the last time this was called, the [Variables](#) will be updated.

void Update()

This is an editor support function and can be ignored.

void Reset(int index)

Resets the [Variable](#) at *index* to its default value defined in *Schema*.

void Clear()

Disassociates this set with *Schema* and *Owner* and removes all its [Variables](#).

string GetVariableName(int index)

Returns the name of the [Variable](#) at index *index*.

[VariableValue](#) GetValueValue(int *index*)

Returns the [VariableValue](#) of the [Variable](#) at index *index*

[SetVariableResult](#) SetVariableValue(int *index*, [VariableValue](#) *value*)

Sets the [VariableValue](#) of the [Variable](#) at index *index* to *value*.

void ResetTag(string *tag*)

Resets all [Variables](#) with *Tag tag* to their default value defined in *Schema*.

void ResetVariables([IList](#)<string> *variables*)

Resets all [Variables](#) in *variables* to their default value defined in *Schema*.

VariableSetAsset

PiRhoSoft.CompositionEngine.VariableSetAsset : [ScriptableObject](#), [ISchemaOwner](#), [IVariableReset](#), [IVariableStore](#)

Description

An asset for storing [Variables](#) that are defined by a [VariableSchema](#). This can also be used as a base class for assets that need to expose variables defined in code to the variables system.

Public Fields

[VariableSet](#) *Variables*

The [Variables](#) stored by this asset that are defined in *Schema*.

Public Properties

[VariableSchema](#) *Schema (read only)*

The [VariableSchema](#) used to define *Variables*.

[MappedVariableStore](#) *Store (read only)*

The store providing the mapping for all the [Variables](#) in this asset - both *Variables* and those defined in code using [VariableMapping](#).

Public Methods

void SetupSchema()

One time setup to initialize *Store*. This is managed automatically.

VariableValue **GetVariable(string name)**

Returns the variable, defined by either *Schema* or with [VariableMappings](#) with name *name*.

SetVariableResult **SetVariable(string name, VariableValue value)**

Sets the variable, defined by either *Schema* or with [VariableMappings](#), with name *name* to *value*.

IList<string> **GetVariableNames()**

Returns the names of all the variables, defined by either *Schema* or with [VariableMappings](#).

void ResetTag(string tag)

Resets all the variables defined in *Schema* with with tag *tag*.

void ResetVariables(IList<string> variables)

Resets all the variables in *variables*.

VariableSetComponent

PiRhoSoft.CompositionEngine.VariableSetComponent : [MonoBehaviour](#), [ISchemaOwner](#), [IVariableReset](#), [IVariableStore](#)

Description

Add this to any [GameObject](#) to store [Variables](#) that are defined by a [VariableSchema](#). This can also be used as a base class for behaviours that need to expose variables defined in code to the variables system.

Public Fields

VariableSet *Variables*

The [Variables](#) stored by this asset that are defined in *Schema*.

Public Properties

VariableSchema *Schema (read only)*

The [VariableSchema](#) used to define *Variables*.

MappedVariableStore *Store (read only)*

The store providing the mapping for all the [Variables](#) in this asset - both *Variables* and those defined in code using [VariableMapping](#).

Public Methods

void SetupSchema()

One time setup to initialize *Store*. This is managed automatically.

VariableValue **GetVariable(string name)**

Returns the variable, defined by either *Schema* or with [VariableMappings](#) with name *name*.

SetVariableResult **SetVariable(string name, VariableValue value)**

Sets the variable, defined by either *Schema* or with [VariableMappings](#), with name *name* to *value*.

IList<string> **GetVariableNames()**

Returns the names of all the variables, defined by either *Schema* or with [VariableMappings](#).

void ResetTag(string tag)

Resets all the variables defined in *Schema* with with tag *tag*.

void ResetVariables(IList<string> variables)

Resets all the variables in *variables*.

VariableSource

PiRhoSoft.CompositionEngine.VariableSource

Description

A wrapper type for fields that allows a value to be set directly or set to a [VariableReference](#). [VariableSource<T>](#) provides a generic implementation that is sufficient for all use cases.

Public Fields

[VariableSourceType](#) *Type*

Whether this source has a value or [VariableReference](#). If this is set to *Value* the subclass will include the value field of the correct type.

[VariableReference](#) *Reference*

If *Type* is set to *Reference*, this holds the [VariableReference](#) where the [VariableValue](#) should be looked up.

Public Methods

void GetInputs(IList<[VariableDefinition](#)> inputs)

If *Type* is set to *Reference* and *Reference* accesses [InstructionStore.Inputs](#), adds the definition for *Reference* to *inputs*.

Protected Methods

[ValueDefinition](#) GetInputDefinition() (*abstract*)

Implement this in a subclass to return a definition for the represented type.

VariableSource

PiRhoSoft.CompositionEngine.VariableSource<T> : [VariableSource](#)

Description

An implementation of [VariableSource](#) that exposes the value to use when *Type* is set to *Value*. Because Unity cannot serialize fields of generic types this class is defined as abstract. Therefore, concrete types for each value type must be implemented. The following built in variable sources are included:

	Type
BoolVariableSource	bool
IntVariableSource,	int
FloatVariableSource,	float
Int2VariableSource,	Vector2Int
Int3VariableSource,	Vector3Int
IntRectVariableSource,	RectInt
IntBoundsVariableSource,	BoundsInt
Vector2VariableSource,	Vector2
Vector3VariableSource,	Vector3
Vector4VariableSource,	Vector4
QuaternionVariableSource,	Quaternion
RectVariableSource,	Rect
BoundsVariableSource,	Bounds
ColorVariableSource,	Color
StringVariableSource,	string
ObjectVariableSource,	Object
GameObjectVariableSource,	GameObject
StoreVariableSource,	IVariableStore
ListVariableSource,	IVariableList
VariableValueSource,	VariableValue

Variable sources for additional types can be added by deriving from this class.

Public Fields

T *Value*

If *Type* is set to *Value*, this holds the value the owner should use for this variable.

VariableSourceType

PiRhoSoft.CompositionEngine.VariableSourceType

Description

Used by [VariableSource](#) to specify how a [VariableValue](#) is retrieved.

Values

[VariableSourceType](#) *Value*

The value is specified directly.

[VariableSourceType](#) *Reference*

The value is looked up from a [VariableReference](#).

VariableStore

PiRhoSoft.CompositionEngine.VariableStore : [IVariableStore](#)

Description

An [IVariableStore](#) that allows an arbitrary set of [VariableValues](#) to be added.

Public Properties

[List](#)<[string](#)> *Names (read only)*

The names of the [VariableValues](#) in the store.

[List](#)<[VariableValue](#)> *Variables (read only)*

The [VariableValues](#) in the store.

[Dictionary](#)<[string](#), [string](#)> *Map (read only)*

The dictionary that maps names to indexes of the [VariableValues](#).

Public Methods

void AddVariable([string](#) *name*, [VariableValue](#) *value*) (*virtual*)

Adds *value* to the store and assigns it the name *name*.

bool RemoveVariable([string](#) *name*)

Removes the [VariableValue](#) with name *name* from the store. If *name* does not exist, false is returned.

void RemoveVariable([int](#) *index*)

Removes the [VariableValue](#) at index *index* from the store.

void VariableMoved([int](#) *from*, [int](#) *to*) (*virtual*)

This is an editor support function that can be ignored.

void Clear() (*virtual*)

Removes all [VariableValues](#) from the store.

[IList](#)<[string](#)> **GetVariableNames**() (*virtual*)

Returns *Names*.

[VariableValue](#) **GetVariable**([string](#) *name*) (*virtual*)

Returns the [VariableValue](#) with name *name*.

[SetVariableResult](#) **SetVariable**([string](#) *name*, [VariableValue](#) *value*) (*virtual*)

Sets the [VariableValue](#) with *name* name to *value*. If *name* does not exist, it will be added.

Protected Methods

void RemoveVariable(string name, int index) (*virtual*)

Removes the variable with *name* name and index *index*.

SetVariableResult SetVariable(string name, VariableValue value, bool allowAdd)

Sets the **VariableValue** with *name* name to *value*. If *name* does not exist, it will be added only if *allowAdd* is true.

VariableType

PiRhoSoft.CompositionEngine.VariableType

Description

Defines the set of types a [VariableValue](#) can hold.

Values

VariableType *Empty*

The [VariableValue](#) has no value.

VariableType *Bool*

The [VariableValue](#) is a bool.

VariableType *Int*

The [VariableValue](#) is an int.

VariableType *Float*

The [VariableValue](#) is a float.

VariableType *Int2*

The [VariableValue](#) is a [Vector2Int](#).

VariableType *Int3*

The [VariableValue](#) is a [Vector3Int](#).

VariableType *IntRect*

The [VariableValue](#) is a [RectInt](#).

VariableType *IntBounds*

The [VariableValue](#) is a [BoundsInt](#).

VariableType *Vector2*

The [VariableValue](#) is a [Vector2](#).

VariableType *Vector3*

The [VariableValue](#) is a [Vector3](#).

VariableType *Vector4*

The [VariableValue](#) is a [Vector4](#).

VariableType *Quaternion*

The [VariableValue](#) is a [Quaternion](#).

VariableType *Rect*

The [VariableValue](#) is a [Rect](#).

VariableType *Bounds*

The **VariableValue** is a **Bounds**.

VariableType *Color*

The **VariableValue** is a **Color**.

VariableType *String*

The **VariableValue** is a string.

VariableType *Enum*

The **VariableValue** is an enum. The type of enum is stored in *EnumType* on **VariableValue**.

VariableType *Object*

The **VariableValue** is an **Object**. If the type is constrained the base type is stored in *ReferenceType* on **VariableValue**.



If a value is both an **Object** and **IVariableStore**, its *Type* will be **Object**.

VariableType *Store*

The **VariableValue** is an **IVariableStore**.

VariableType *List*

The **VariableValue** is an **IVariableList**.

VariableValue

PiRhoSoft.CompositionEngine.VariableValue : ValueType

Description

Stores a value or object in a generic fashion without boxing value types (except enums). The possible types that can be stored are defined in [VariableType](#).

Static Properties

VariableValue *Empty (read only)*

Creates a value with [VariableType](#) Empty.

Static Methods

VariableType **GetType**(Type type)

Returns the [VariableType](#) that would be used to store a value of Type *type*. If *type* is not supported, Empty will be returned.

VariableValue **Create**(bool value)

Creates a [VariableValue](#) with Type Bool that holds *value*.

VariableValue **Create**(int value)

Creates a [VariableValue](#) with Type Int that holds *value*.

VariableValue **Create**(float value)

Creates a [VariableValue](#) with Type Float that holds *value*.

VariableValue **Create**(Vector2Int value)

Creates a [VariableValue](#) with Type Int2 that holds *value*.

VariableValue **Create**(Vector3Int value)

Creates a [VariableValue](#) with Type Int3 that holds *value*.

VariableValue **Create**(RectInt value)

Creates a [VariableValue](#) with Type RectInt that holds *value*.

VariableValue **Create**(BoundsInt value)

Creates a [VariableValue](#) with Type BoundsInt that holds *value*.

VariableValue **Create**(Vector2 value)

Creates a [VariableValue](#) with Type Vector2 that holds *value*.

VariableValue **Create**(Vector3 value)

Creates a [VariableValue](#) with Type Vector3 that holds *value*.

VariableValue Create(Vector4 value)

Creates a **VariableValue** with *Type* Vector4 that holds *value*.

VariableValue Create(Quaternion value)

Creates a **VariableValue** with *Type* Quaternion that holds *value*.

VariableValue Create(Rect value)

Creates a **VariableValue** with *Type* Rect that holds *value*.

VariableValue Create(Bounds value)

Creates a **VariableValue** with *Type* Bounds that holds *value*.

VariableValue Create(Color value)

Creates a **VariableValue** with *Type* Color that holds *value*.

VariableValue Create(string str)

Creates a **VariableValue** with *Type* String that holds *str*.

VariableValue Create(Enum e)

Creates a **VariableValue** with *Type* Enum and *EnumType* the type of *e* that holds *e*.

VariableValue Create(Object obj)

Creates a **VariableValue** with *Type* Object that holds *obj*.

VariableValue Create(IVariableStore store)

Creates a **VariableValue** with *Type* Store that holds *store*.

VariableValue Create(IVariableList list)

Creates a **VariableValue** with *Type* List that holds *list*.

VariableValue CreateValue<T>(T value)

Creates a **VariableValue** with *Type* determined from *T* that holds *value*. This can be used for all **VariableTypes** except Enum, Object, Store, and List.

VariableValue CreateReference(object reference)

Creates a **VariableValue** with *Type* determined from the type of *reference* that holds *reference*. This can be used for the **VariableTypes** Enum, Object, Store, and List.



If *reference* is both an **IVariableStore** and an **Object**, the value will have type Object.

VariableValue CreateAny(object obj)

Creates a **VariableValue** with *Type* determined from the type of *obj*. This can be used for any **VariableType** when it is unknown whether *obj* is a value or reference type.

Public Properties

VariableType *Type (read only)*

The **VariableType** of the value.

bool *IsEmpty (read only)*

Returns true if *Type* is Empty.

bool *IsNull (read only)*

Returns true if *Type* is Object, Store, or List and no value is stored.

bool *HasValue (read only)*

Returns true if *Type* is a value type (i.e anything other than String, Enum, Object, Store, or List).

bool *HasString (read only)*

Returns true if *Type* is String.

bool *HasEnum (read only)*

Returns true if *Type* is Enum.

bool *HasReference (read only)*

Returns true if *Type* is Object, Store, or List.

bool *HasObject (read only)*

Returns true if the stored object is an **Object** or derived from **Object**.

bool *HasStore (read only)*

Returns true if the stored object is an **IVariableStore**.



Even if *Type* is Object, this will still return true if the stored object is also an **IVariableStore**.

bool *HasList (read only)*

Returns true if the stored object is an **IVariableList**.



Even if *Type* is Object, this will still return true if the stored object is also an **IVariableList**.

bool *HasNumber (read only)*

Returns true if *Type* is Int or Float.

bool *HasNumber2 (read only)*

Returns true if *Type* is Int2 or Vector2.

bool *HasNumber3 (read only)*

Returns true if *Type* is Int3, Vector3, Int2, or Vector2.

bool *HasNumber4 (read only)*

Returns true if *Type* is Vector4, Int3, Vector3, Int2, or Vector2.

bool *HasRect (read only)*

Returns true if *Type* is IntRect or Rect.

bool *HasBounds (read only)*

Returns true if *Type* is IntBounds or Bounds.

bool *Bool (read only)*

Returns the stored value if *Type* is Bool or an undefined value if it is not.

int *Int (read only)*

Returns the stored value if *Type* is Int or an undefined value if it is not.

float *Float (read only)*

Returns the stored value if *Type* is Float or an undefined value if it is not.

Vector2Int *Int2 (read only)*

Returns the stored value if *Type* is Int2 or an undefined value if it is not.

Vector3Int *Int3 (read only)*

Returns the stored value if *Type* is Int3 or an undefined value if it is not.

RectInt *IntRect (read only)*

Returns the stored value if *Type* is IntRect or an undefined value if it is not.

BoundsInt *IntBounds (read only)*

Returns the stored value if *Type* is IntBounds or an undefined value if it is not.

Vector2 *Vector2 (read only)*

Returns the stored value if *Type* is Vector2 or an undefined value if it is not.

Vector3 *Vector3 (read only)*

Returns the stored value if *Type* is Vector3 or an undefined value if it is not.

Vector4 *Vector4 (read only)*

Returns the stored value if *Type* is Vector4 or an undefined value if it is not.

Quaternion *Quaternion (read only)*

Returns the stored value if *Type* is Quaternion or an undefined value if it is not.

Rect *Rect (read only)*

Returns the stored value if *Type* is Rect or an undefined value if it is not.

Bounds *Bounds (read only)*

Returns the stored value if *Type* is Bounds or an undefined value if it is not.

Color *Color (read only)*

Returns the stored value if *Type* is Color or an undefined value if it is not.

string *String (read only)*

Returns the stored value if *Type* is String or null if it is not.

Enum *Enum (read only)*

Returns the stored value if *Type* is Enum or null if it is not.

Object *Object (read only)*

Returns the stored object if *Type* is Object or null if it is not.

IVariableStore *Store (read only)*

Returns the stored object if the object is an **IVariableStore** or null if it is not.

IVariableList *List (read only)*

Returns the stored object if the object is an **IVariableList** or null if it is not.

float *Number (read only)*

Returns the stored value if *Type* is Int or Float or 0.0 if it is not.

Vector2 *Number2 (read only)*

Returns the stored value if *Type* is Int2 or Vector2 or (0.0, 0.0) if it is not.

Vector3 *Number3 (read only)*

Returns the stored value if *Type* is Int3 or Vector3, *Number2* with z = 0.0 if *Type* is Int2 or Vector2, or (0.0, 0.0, 0.0) otherwise.

Vector4 *Number4 (read only)*

Returns the stored value if *Type* is Vector4, *Number3* with w = 1.0 if *Type* is Int3, Vector3, Int2, or Vector2, or (0.0, 0.0, 0.0, 1.0) otherwise.

Rect *NumberRect (read only)*

Returns the stored value if *Type* is IntRect or Rect or a 0 sized rect at (0.0, 0.0) if it is not.

Bounds *NumberBounds (read only)*

Returns the stored value if *Type* is IntBounds or Bounds or a 0 sized bounds at (0.0, 0.0, 0.0) if it is not.

Object *Reference (read only)*

The stored reference value whether *Type* is Object, Store, or List.

Type *EnumType (read only)*

The type of the stored *Enum* if *Type* is Enum or null if it is not.

Type *ReferenceType (read only)*

The type of the stored *Object* if *Type* is Object or null if it is not.

Public Methods

bool HasEnumType<Type>()

true if *Type* is Enum and *EnumType* is *Type*.

bool HasReferenceType<Type>()

true if *Type* is Object and *ReferenceType* is *Type* or is derived from *Type*.

bool HasEnumType(Type type)

true if *Type* is Enum and *EnumType* is *type*.

bool HasReferenceType(Type type)

true if *Type* is Object and *ReferenceType* is *type* or is derived from *type*.

object GetBoxedValue()

Returns the stored value, regardless of *Type*. Value types will be boxed.

bool TryGetBool(bool value (out))

If *Type* is Bool, sets *value* to the stored value and returns true. Otherwise sets *value* to false and returns false.

bool TryGetInt(int value (out))

If *Type* is Int, sets *value* to the stored value and returns true. Otherwise sets *value* to 0 and returns false.

bool TryGetFloat(float value (out))

If *Type* is Float, sets *value* to the stored value and returns true. Otherwise sets *value* to 0.0 and returns false.

bool TryGetInt2(Vector2Int value (out))

If *Type* is Int2, sets *value* to the stored value and returns true. Otherwise sets *value* to (0, 0) and returns false.

bool TryGetInt3(Vector3Int value (out))

If *Type* is Int3, sets *value* to the stored value and returns true. Otherwise sets *value* to (0, 0, 0) and returns false.

bool TryGetIntRect(RectInt value (out))

If *Type* is IntRect, sets *value* to the stored value and returns true. Otherwise sets *value* to a 0 sized rect at (0, 0) and returns false.

bool TryGetIntBounds(BoundsInt value (out))

If *Type* is IntBounds, sets *value* to the stored value and returns true. Otherwise sets *value* to a 0 sized bounds at (0, 0, 0) and returns false.

bool TryGetVector2(Vector2 value (out))

If *Type* is Vector2, sets *value* to the stored value and returns true. Otherwise sets *value* to (0.0, 0.0) and returns false.

bool TryGetVector3(Vector3 value (out))

If *Type* is Vector3, sets *value* to the stored value and returns true. Otherwise sets *value* to (0.0, 0.0, 0.0) and returns false.

bool TryGetVector4(Vector4 value (out))

If *Type* is Vector4, sets *value* to the stored value and returns true. Otherwise sets *value* to (0.0, 0.0, 0.0, 1.0) and returns false.

bool TryGetQuaternion(Quaternion value (out))

If *Type* is Quaternion, sets *value* to the stored value and returns true. Otherwise sets *value* to Quaternion.identity and returns false.

bool TryGetRect(Rect value (out))

If *Type* is Rect, sets *value* to the stored value and returns true. Otherwise sets *value* to a 0 sized rect at (0.0, 0.0) and returns false.

bool TryGetBounds(Bounds value (out))

If *Type* is Bounds, sets *value* to the stored value and returns true. Otherwise sets *value* to a 0 sized bounds at (0.0, 0.0, 0.0) and returns false.

bool TryGetColor(Color value (out))

If *Type* is Color, sets *value* to the stored value and returns true. Otherwise sets *color* to white and returns false.

bool TryGetString(string s (out))

If *Type* is String, sets *s* to the stored value and returns true. Otherwise sets *s* to an empty string and returns false.

bool TryGetEnum<EnumType>(EnumType value (out))

If *Type* is Enum and *EnumType* is *EnumType*, sets *value* to the stored value and returns true. Otherwise sets *value* to 0 and returns false.

bool TryGetObject(Object obj (out))

If *Type* is Object, sets *obj* to the stored object and returns true. Otherwise sets *obj* to null and returns false.

bool TryGetStore(IVariableStore store (out))

If the stored object is an IVariableStore, sets *store* to the stored object and returns true. Otherwise sets *store* to null and returns false.

bool TryGetList(IVariableList list (out))

If the stored object is an IVariableList, sets *list* to the stored object and returns true. Otherwise sets *list* to null and returns false.

bool TryGetReference<T>(T t (out))

If *Type* is Object, Store, or List and the stored object has type *T* or is derived from type *T*, sets *t* to the stored object and returns true. Otherwise sets *t* to null and returns false.

VariableValueSource

PiRhoSoft.CompositionEngine.VariableValueSource : [VariableSource](#)<[VariableValue](#)>

Description

A [VariableSource](#) for any [VariableValue](#).

Constructors

VariableValueSource([VariableType](#) type, [ValueDefinition](#) definition)

Initializes the source to *Type* Value with *Value* a [VariableValue](#) with [VariableType](#) *type* and initialized with *definition*.

Public Fields

ValueDefinition *Definition*

The [ValueDefinition](#) the source was initialized with.

Vector2VariableSource

PiRhoSoft.CompositionEngine.Vector2VariableSource : [VariableSource](#)<[Vector2](#)>

Description

A [VariableSource](#) for Vector2 [VariableValues](#).

Constructors

Vector2VariableSource([Vector2](#) defaultValue)

Initializes the source to *Type* Value with *Value* _defaultValue.

Vector3VariableSource

PiRhoSoft.CompositionEngine.Vector3VariableSource : [VariableSource](#)<[Vector3](#)>

Description

A [VariableSource](#) for [Vector3](#) [VariableValues](#).

Constructors

Vector3VariableSource([Vector3](#) *defaultValue*)

Initializes the source to *Type* *Value* with *Value* *_defaultValue*.

Vector4VariableSource

PiRhoSoft.CompositionEngine.Vector4VariableSource : [VariableSource](#)<[Vector4](#)>

Description

A [VariableSource](#) for Vector4 [VariableValues](#).

Constructors

Vector4VariableSource([Vector4](#) defaultValue)

Initializes the source to *Type* Value with *Value* _defaultValue.

WaitNode

PiRhoSoft.CompositionEngine.WaitNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to delay execution for a specified amount of time.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.

[FloatVariableSource](#) *Time*

The number of seconds to delay the [InstructionGraph](#).

bool *UseScaledTime*

If this is set, the delay will be based on [scaled time](#), otherwise it will be based on [real time](#).

WritableStore

PiRhoSoft.CompositionEngine.WritableStore : [VariableStore](#)

Description

An [IVariableStore](#) implementation that disallows contained [VariableValues](#) to be added. [Variables](#) that already exist in the store can have their value changed.

YieldNode

PiRhoSoft.CompositionEngine.YieldNode : [InstructionGraphNode](#)

Description

Add this to an [InstructionGraph](#) to delay execution for one frame.

Public Fields

[InstructionGraphNode](#) *Next*

The [InstructionGraphNode](#) to run when this node finishes.