Unity Composition Bindings

PiRho Soft

Overview

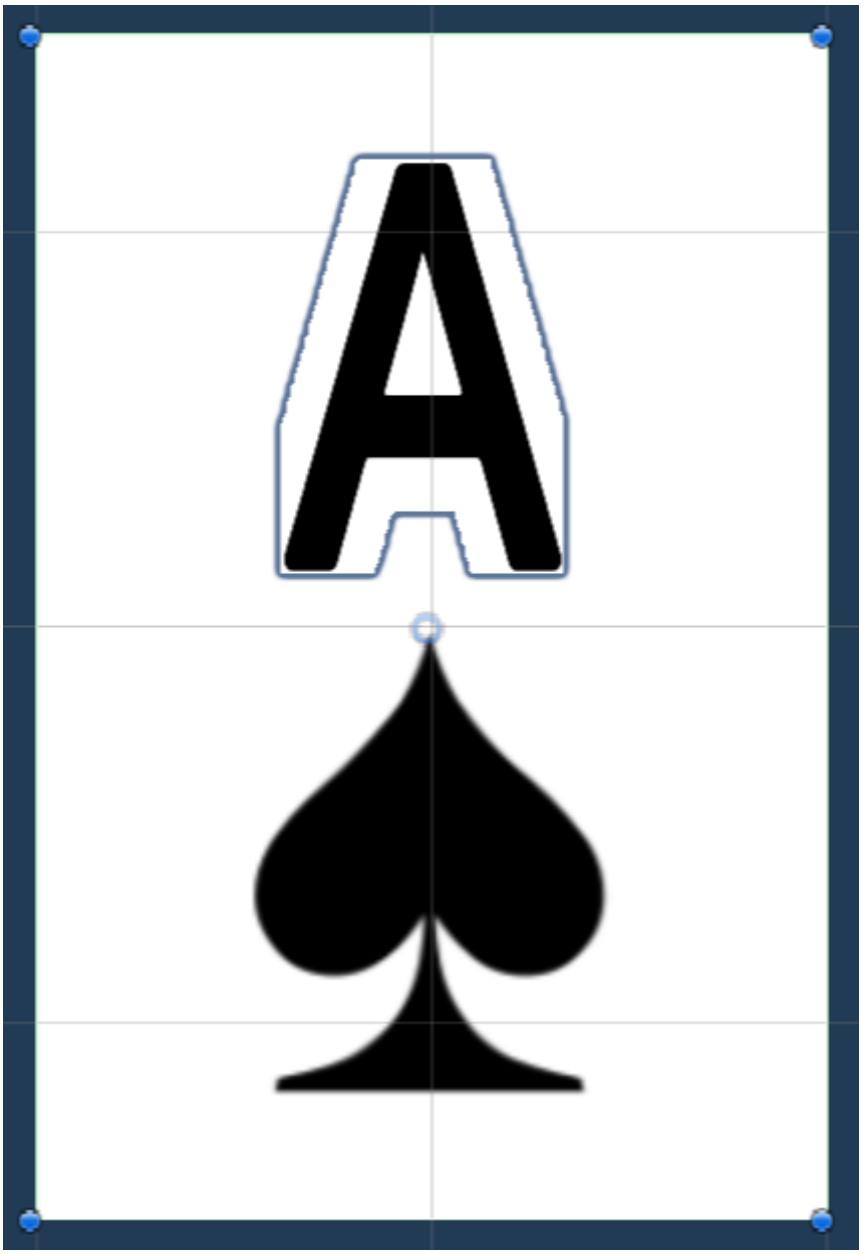
The Bindings system is used to bind data from the composition system, usually via VariableReferences, to elements in a scene. Commonly this is used to automatically update visual elements, like text, to display values stored in code, for example, showing the health of a character, or the amount of money a player has. The two main components of the binding system are Binding Roots and Variable Bindings. Variable Bindings are the specific MonoBehaviours that alter properties of other behaviours, while Binding Roots provide the variable bindings with access to the variables that hold the data to be bound to. Within a binding group, variable bindings can be further categorized into groups so each group can be individually updated. Bindings can be animated and the instruction graph can interface with the animation's progress for sequencing. In addition to numerous built-in variable binding types, bindings can be easily extended to perform custom behaviour.

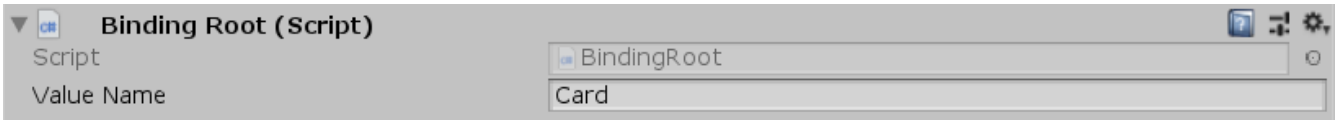
Topics

1. Binding Roots
2. Variable Bindings
3. Creating Custom Binding Roots
4. Creating Custom Variable Bindings

Binding Roots

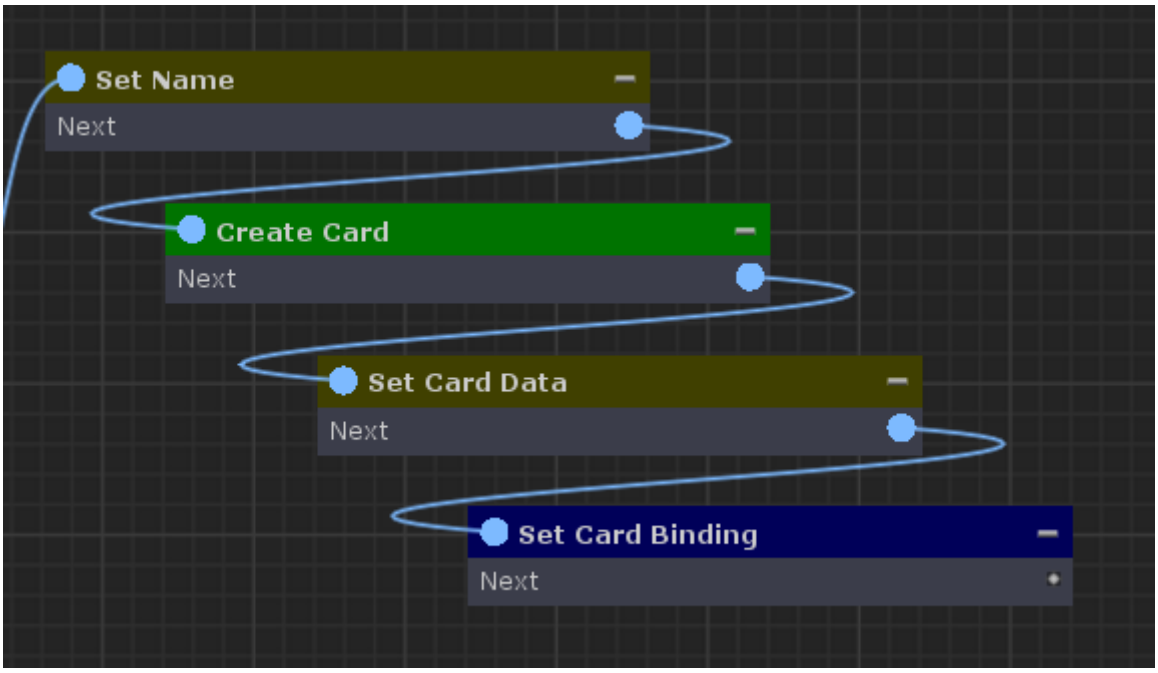
Binding Roots is a MonoBehaviour that acts as a root object for all bindings that are children of the root to bind data from. Each binding root contains a variable, *Value* that child bindings access via the *ValueName* property. Binding roots are hierachical and variable bindings have access to every binding root in its hierarchy as long as each root has a differet *ValueName*. In addition variables can be looked up on the Composition Manager's `"global"` and `"scene"` IVariableStores (detailed in Accessing Variables. A binding root's *Value* can be set in two ways: from a derived class such as Object Binding Root (detailed in Creating Custom Binding Roots, or through a Set Binding Node. Take the following example of the Card prefab from the CardGame example:

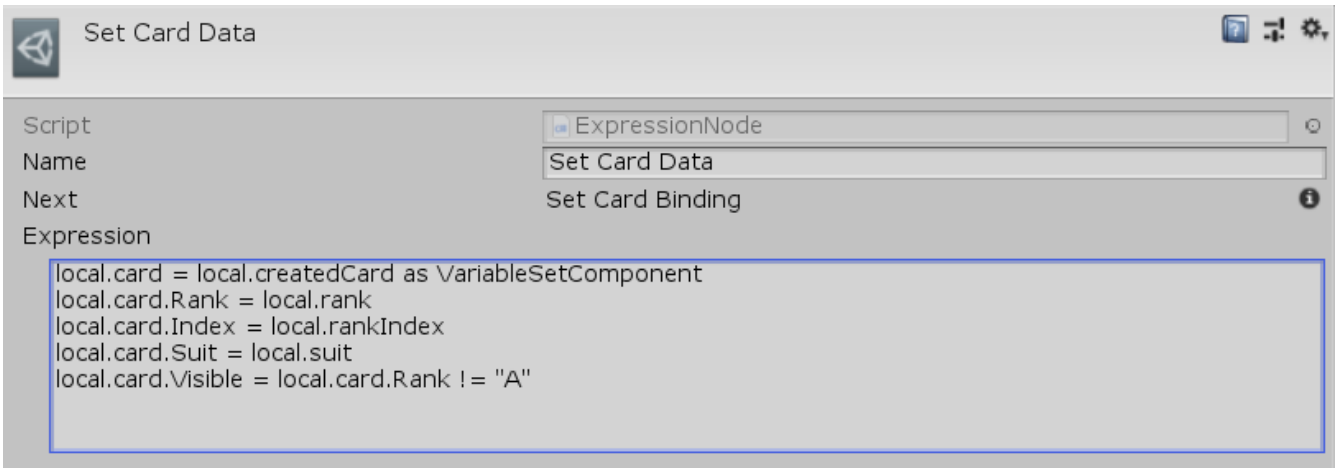


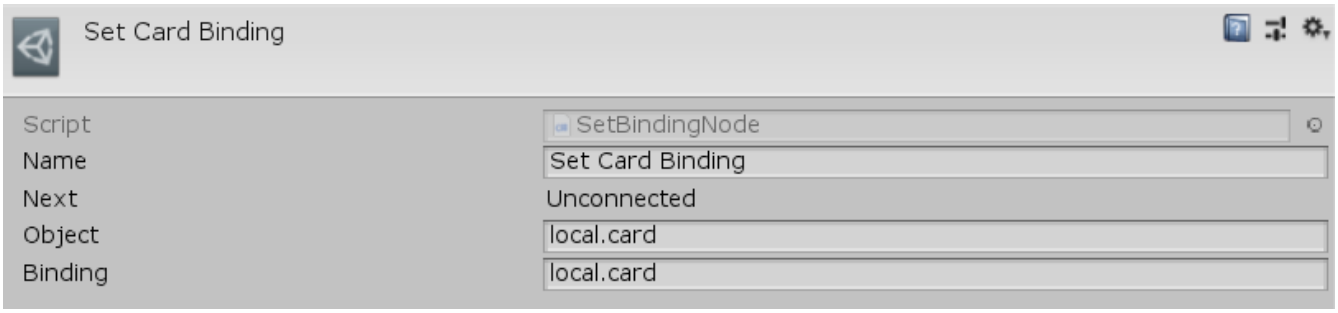


The base object of the prefab has the above binding root behaviour attached. Notice that *Value Name* is set to `"Card"`. This means that all child variable binding behaviours can access the variables of this binding root, through the variable `"Card"`. The variables available on this binding root are set

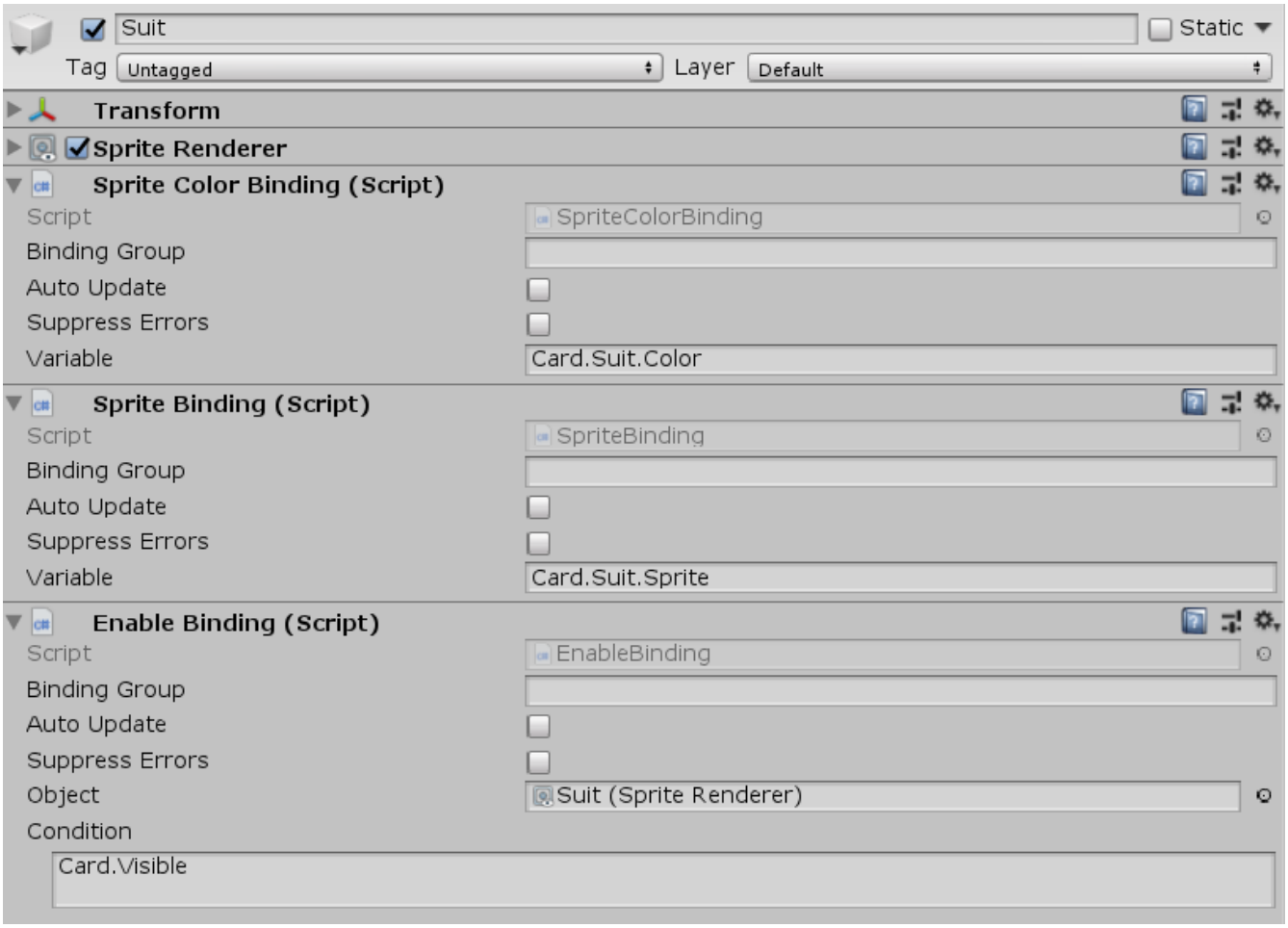
up through the following set of nodes, ending with a Set Binding Node.

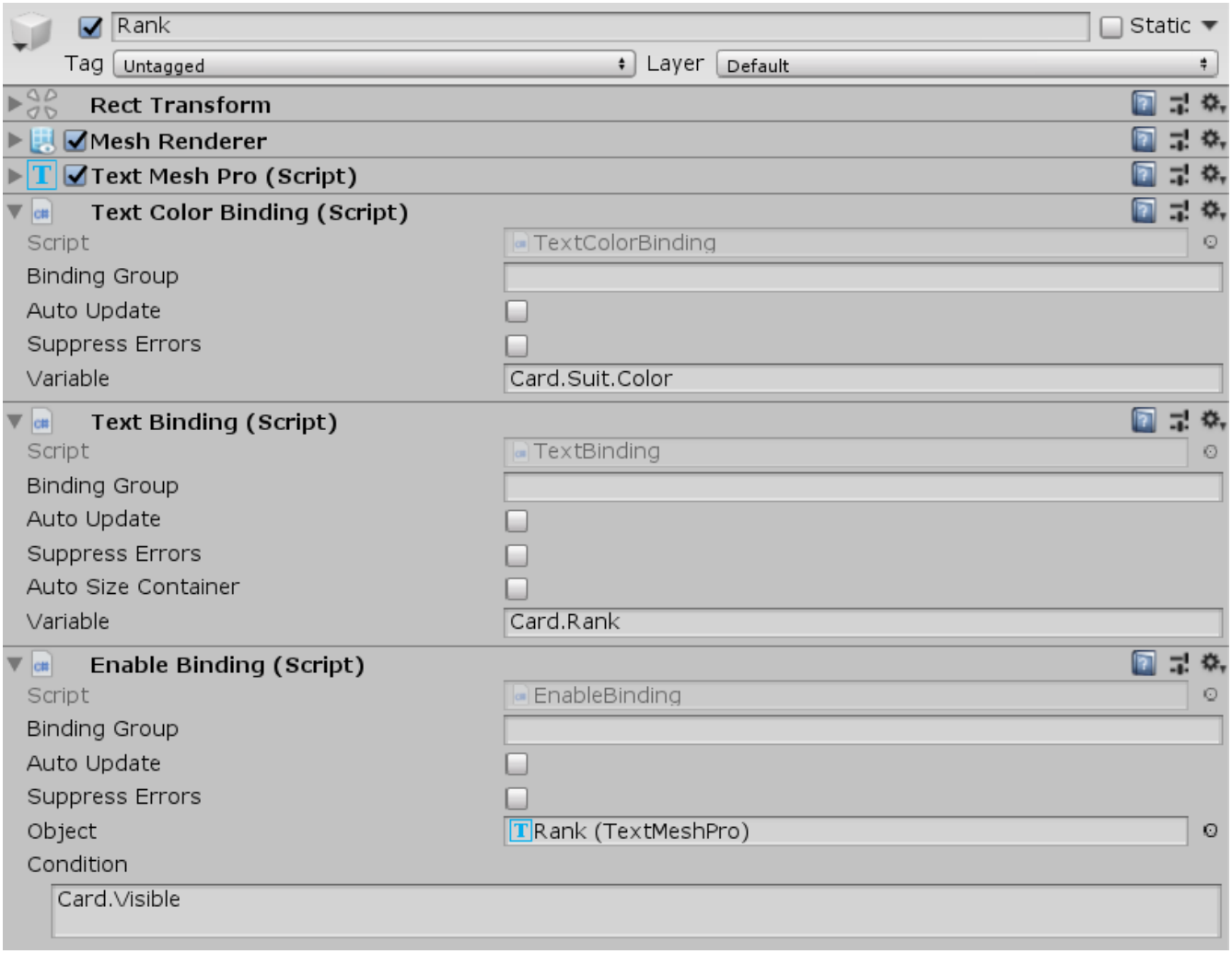






There are two child objects in the prefab, "Suit" and "Rank", each with multiple variable bindings behaviours attached. These use the variables set up in the graph above to set visual elements in the scene.


Suit
Static
Tag
Untagged
Layer
Default
Transform
Sprite Renderer
Sprite Color Binding (Script)
Script
SpriteColorBinding
Binding Group
Auto Update
Suppress Errors
Variable
Card.Suit.Color
Sprite Binding (Script)
Script
SpriteBinding
Binding Group
Auto Update
Suppress Errors
Variable
Card.Suit.Sprite
Enable Binding (Script)
Script
EnableBinding
Binding Group
Auto Update
Suppress Errors
Object
Suit (Sprite Renderer)
Condition
Card.Visible


Rank
Static
Tag
Untagged
Layer
Default
Rect Transform
Mesh Renderer
Text Mesh Pro (Script)
Text Color Binding (Script)
Script
TextColorBinding
Binding Group
Auto Update
Suppress Errors
Variable
Card.Suit.Color
Text Binding (Script)
Script
TextBinding
Binding Group
Auto Update
Suppress Errors
Auto Size Container
Variable
Card.Rank
Enable Binding (Script)
Script
EnableBinding
Binding Group
Auto Update
Suppress Errors
Object
Rank (TextMeshPro)
Condition
Card.Visible

Notice that each binding uses "`Card`" to access the binding root's variables.

Variable Bindings

A Variable Binding itself is a MonoBehaviour designed to be inherited from to either visually change or perform actions based on variables. As outlined in Binding Roots, by default variable bindings have access to the Composition Manager's `"global"` and `"scene"` IVariableStores, in addition to any binding roots in its hierarchy. Generally, bindings are used to display data in a user interface, such as timers, scores, health bars, etc. However, bindings may also perform any desired action, such as the Graph Trigger Binding, which runs an instruction graph when the variable it is bound to changes.

Updating Bindings

Bindings have an *AutoUpdate* flag which if set, means that the binding will receive an update call automatically every frame. If *AutoUpdate* is false then bindings will need to be prompted to update manually, usually via an Update Binding Node. Use this cautiously as too many update calls per frame may have performance implications.

Binding Groups

Variable Bindings can also be categorized into groups with the *BindingGroup* property so that only certain bindings will update when they are prompted. When updating bindings, if `null` is passed to the `VariableBinding.UpdateBindings()` method as the group parameter, then all bindings, regardless of their group will be updated, otherwise only those with a *BindingGroup* that matches the passed string will be updated.

Animation

Bindings optionally support animation through a BindingAnimationStatus object passed to the `UpdateBindings()` method. If a binding utilizes animation (such as the Bar Binding gradually changing its fill value over time), then the `Increment()` method should be called on the status object when the animation begins and the `Decrement()` method when the animation is finished. Now the `IsFinished()` on the status object can be checked and waited on if desired. For example, an Update Binding Node uses this to wait to move on to the next node until all bindings have finished animating.

The following is a list of built-in bindings.

Name	Description
Bar Binding	Bind the fill value of an image to the ratio of two values
Enable Binding	Enable/disable an object based on a condition
Expression Binding	Bind text to the value of an expression
Graph Trigger Binding	Run a graph when a value changes
Image Binding	Bind an image to a value
Image Color Binding	Bind an image's blend color to a value

Name	Description
List Binding	Create objects based on the values in a list
Message Binding	Bind text based on a message
Number Binding	Bind text to a number
Sprite Binding	Bind a sprite to a value
Sprite Color Binding	Bind a sprites blend color to a value
Text Binding	Bind text to a value
Text Color Binding	Bind the color of text to a value
Text Input Binding	Bind a variable a text's user input

Errors

Most Variable Bindings will disable their corresponding visual element if they fail to retreive their data and report the error. Sometimes this may be intended behaviour so if set, *SuppressErrors* will hide those errors. This behaviour is up to each individual binding class to implement so check the specific manual page to clarify behaviour.

Creating Custom Binding Roots

To create custom binding roots, derive from BindingRoot and override the *Value* property. The following is an example of a custom binding root uses an Object that is set in the editor as the value for child variable bindings to look up data on.

```
public class ObjectBindingRoot : BindingRoot
{
    public Object Object;

    public override VariableValue Value
    {
        get
        {
            return Object ? VariableValue.Create(Object) : VariableValue.Empty;
        }
    }
}
```

Other examples of custom binding roots are the Focus Binding Root and the Menu Item.

Creating Custom Variable Bindings

To create custom variable bindings, derive from VariableBinding and implement the abstract method `UpdateBinding(IVariableStore variables, BindingAnimationStatus status)`. The following is an example of a custom binding that activates or deactivates a GameObject based on a bool variable reference.

```
    public class ActivateBinding : VariableBinding
    {
        public GameObject GameObject;

        public VariableReference Variable = new VariableReference();

        protected override void UpdateBinding(IVariableStore variables,
BindingAnimationStatus status)
        {
            if (GameObject)
            {
                Resolve(variables, Variable, out bool active);
                GameObject.SetActive(active);
            }
        }
    }
```

The next example utilizes animation to increment displayed text based on an int variable reference.

```
    public class AnimatedIntBinding : VariableBinding
    {
        public VariableReference Variable = new VariableReference();

        private TMP_Text _text;
        private int _previous = 0;

        private void Start()
        {
            _text = GetComponent<TMP_Text>();
        }

        protected override void UpdateBinding(IVariableStore variables,
BindingAnimationStatus status)
        {
            if (Resolve(variables, Variable, out int target))
            {
                StopAllCoroutines();
                StartCoroutine(AnimateText(target, status));
            }
        }

        private IEnumerator AnimateText(int target, BindingAnimationStatus status)
        {
            status.Increment();

            while (_previous != target)
            {
                _previous = target > _previous ? _previous - 1 : _previous + 1;
                _text.text = _previous.ToString();

                yield return null;
            }

            status.Decrement();
        }
    }
```