Simple Bind

Overview

Introduction

Simple Bind from SoftSource Consulting provides a MVVM-style binding experience for Unity applications. Simple Bind lets Unity developers easily implement powerful binding functionality similar to WPF, AngularJS, and other MV* frameworks.

Key functionality includes:

- Property binding
- Event binding to Unity Events
- Inheritable BindingContext (data context)
- Editor for easy configuration of binding
- Value Converters / Multi-Value Converters
- Built-in Value Converters (int to string, double to string, etc.)
- Built-in Multi-Value Converters (doubles to string, RGB to color)
- Data bound controls (Items Control, List Box, Tooltip)
- Simple creation of data bound Item Templates

What Simple Bind is not

Simple Bind is not an architectural framework or comprehensive design pattern.

Simple Bind does not enforce or require any particular MV* pattern or even a MV* pattern at all.

Simple Bind does not require extending from base classes or implementing specific interfaces.

What Simple Bind is

Simple Bind is a collection of Core Components that can easily be added to any Unity GameObject in order to implement different aspects of data and event binding.

Simple Bind is also a collection of pre-defined data and event bound controls that can provide a foundation for highly data driven applications.

Simple Bind provides a simple pattern for implementing data templates within Unity using the Core Components and the concept of Unity Prefabs.

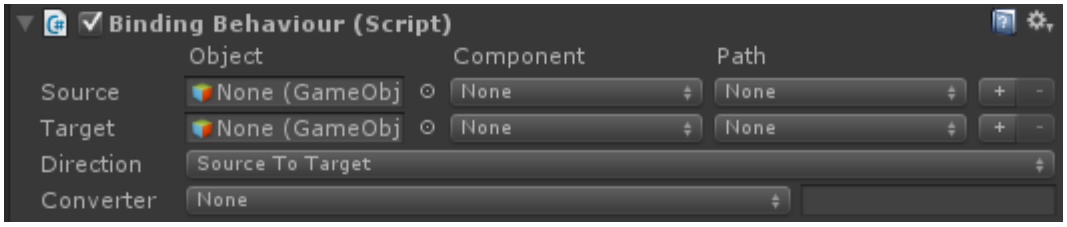
Simple Bind is just simple. Simple to learn, Simple to integrate, Simple to extend.

This Document and Beyond

In this document is a brief overview of what makes up the current Simple Bind Asset, however, this document does not go into details about how to use these Components. There are many additional documents, demos and tutorials available at http://sftsrc.com/unity that will go in-depth on many ways to use Simple Bind, from the basic Component-to-Component binding all the way to complex multi-level MVVM implementations.

Core Components

Binding Behaviour

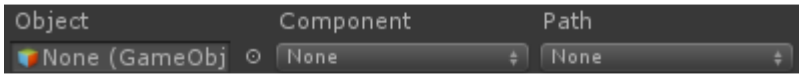


The most basic component in Simple Bind is the Binding Behaviour, which provides core data binding mechanics.

The Binding Behaviour has four key properties:

- Source – Where is the data coming from?
- Target – Where is the data going?
- Direction – Do you need two way binding?
- Converter – Are the Source and Target different data types?
    - Converter Parameter – Additional static data for the Converter

Both the Source and Target are defined as Binding Endpoints. You will notice that many of the Simple Bind components utilize Binding Endpoints.



Binding Endpoints define where a single piece of data exists by allowing the user to set the Game Object, the Component on the Game Object and the Path of the property on the Component.

In the case of the Source property the Binding Endpoint defines where the data for the Binding will be coming from. This could be any property on another Component or the property defined by a multi-level property path when bound to a Binding Context. In a typical MVVM approach the Source would be defined on the View Model.

The Target property defines the opposite end of the binding as the Source and typically would be a property defined in the View. However, a Target could just as easily be any property that a Binding Endpoint can define, including a property on another View Model.

The Source and Target fields have "+" and "-" buttons next to them that allow for adding additional Sources and Targets to the Binding Behaviour. This ability allows the user to create complex multi-binding scenarios. Not all multi-binding configurations make sense and not all are supported. The most common case is aggregating multiple source values through a Converter into a single target value, such as in the RGBToColorConverter.

Direction allows for Source To Target, Target To Source and Both. These values allow the user to configure the direction the data will flow.

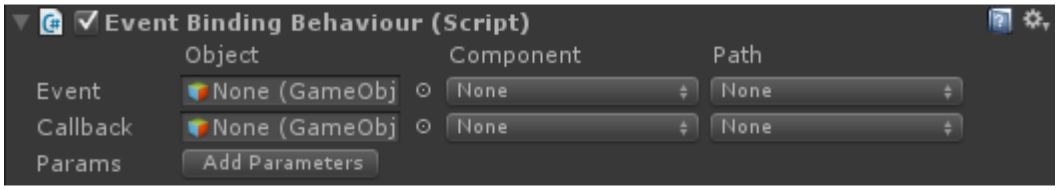
The list of Converters will display all converters that are defined within the Unity project. In the case of a single binding (one Source and one Target) any class that implements IValueConverter. If multiple Sources or Targets are defined then any class that implements IMultiValueConverter will be displayed.

Converters are the one place where, for now, Simple Bind does rely on a specific interface.

Converters typically follow a standard naming convention {SourceType}To{TargetType}Converter. For instance IntToStringConverter will convert a Source data type of int to a Target data type of String. The multi-converter RGBToColorConverter will convert three sources of data type double that represent the Red, Green and Blue components into a single Color data type.

The text field after the Converter drop down selector is for a Converter Parameter. At the moment the Converter Parameter is a single static string value. A StringToPrefixStringConverter could use the Converter Parameter to Prefix the Source string with a static value before updating the Target.

Event Binding Behaviour



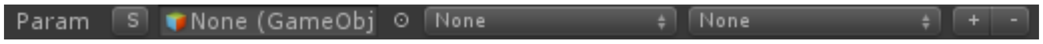
The Event Binding Behaviour is similar to the Binding Behaviour but instead of binding data between Components, it binds a specific Unity Event on one Component to a Callback (Method or Delegate) on another Component.

The Event property looks similar to a Binding Endpoint but is actually an Event Binding Endpoint where instead of the Path displaying properties it will display any UnityEvent or UnityEvent<T> that is exposed by the Component.

The Callback property is again similar but implements a Callback Binding Endpoint and will display all Methods or Delegates exposed by the Component.

Once the Event and Callback properties are set then anytime the associated Event is fired the associated Callback will also get called.

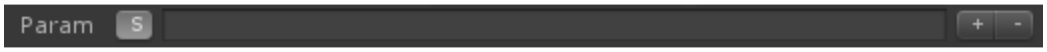
By default no parameters will be passed to the associated Callback. If parameters are needed then clicking on the “Add Parameters” button will display a Binding Endpoint that allows for parameters to be assigned.



This Binding Endpoint works in the same way as the Source and Target properties on the Binding Behaviour and will define where the parameter data will come from.

Once again there are “+” and “-“ buttons that allow for multiple parameters to be added to the Callback.

One difference with the Parameter binding is that there is an “S” button before the Binding Endpoint. Clicking the “S” will change this parameter from being a bound value to being a static string value.

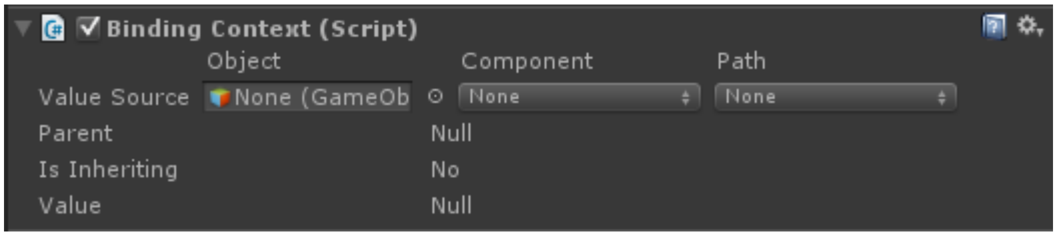


Having both Bound and Static parameters with the ability to add multiple parameters allows for any number of potential scenarios.

The Callback Binding Endpoint inspects the given parameter list and attempts to choose the best Callback on the Component to use. If a single Parameter is specified and the Component has two methods named Foo, one with no parameters and the other with one parameter, then the Callback Binding Endpoint will call the Foo method that has a single parameter. If no Parameter is specified then the Foo method without parameters will be called.

The Event Binding Behaviour is very flexible, however, it is not magic and cannot always make the right decisions. Be cautious when attempting to add too many parameters or too many overloaded methods on the Components.

Binding Context


Binding Context (Script)

	Object	Component	Path
Value Source	None (GameOb	None	None
Parent		Null	
Is Inheriting		No	
Value		Null	

Both the Binding Behaviour and Event Binding Behaviour allow for defining how binding will work but both rely on a Component from a Game Object in the Hierarchy to get their data. The Binding Context allows both Behaviours to reach beyond the Components and utilize multi-level property paths to get at data beyond the Component. In an MMVM application the Binding Context could be thought of as the Data Context.

The only settable property on the Binding Context is the Value Source and like many other properties it is represented as a Binding Endpoint. The Value Source can be set using this Binding Endpoint within the Inspector, but it can also be set from within a script.

One way to think about the Binding Context is that it is the root data source for other Binding Behaviours within a section of the UI, such as setting the context for an Item Template. All other bindings can point to a Binding Context without needing to know where the Binding Context is getting its value from. This idea will be demonstrated later in the Simple Bind Demo series.

Another key aspect of the Binding Behaviour is that it is Inheritable. This means that if a Binding Context is defined on a given Game Object but the Value Source has not been set, then it will look up the Hierarchy for the next Binding Context and Inherit the Value Source from that Binding Context.

This inheritance allows for many interesting scenarios including component based composition UIs that simply inherit their Binding Context from the root of the composition without any additional coding.

Standard Converters

Value Converters

As previously mentioned with Binding Behaviour the Value Converter is used to convert the Source value data type to the Target value data type.

All Value Converters must implement the IValueConverter interface which has two methods:

> public object ConvertToTarget(object value, System.Type targetType, object parameter)
>
> public object ConvertToSource(object value, System.Type sourceType, object parameter)

Both these methods are fairly straight forward. Each takes in a value, the type the value needs to be returned as, and the parameter that was set as the Converter Parameter in the Binding Behaviour.

Implementing a custom converter is as simple as creating a new class, inheriting from IValueConverter and implementing these two methods.

Not all converters need to be two way converters and you may find that many times you only need to implement ConvertToTarget and simply return NULL from ConvertToSource.

Simple Bind currently comes with the following default converters:

- DoubleToStringConverter
- FloatToStringConverter
- IntToStringConverter
- PrependStringConverter
- QuaternionToStringConverter

Multi-Value Converters

Multi-Value converters sound a bit trickier but in reality they are just as simple to implement as single value converters. The difference is that a Multi-Value converter must implement the IMultiValueConverter interface, which has the following two methods:

- public object ConvertToTarget(IEnumerable<object> value, Type targetType, object parameter)
- public object ConvertToSource(IEnumerable<object> value, Type sourceType, object parameter)
    - *Note: As evidenced by the **Type sourceType** argument, IMultiValueConverter. ConvertToSource currently only supports scenarios where there is a **single binding source**.*

As you can see the only difference from IValueConverter is that for both methods the first parameter is an IEnumerable<object> instead of a single object.

Simple Bind currently comes with the following default multi-value converters:

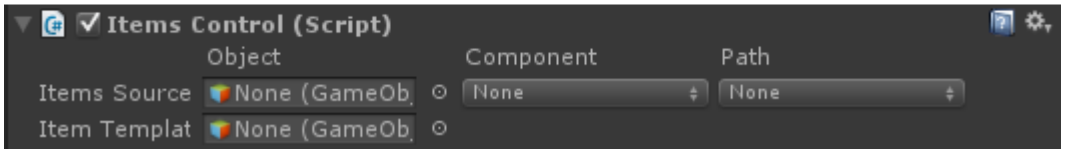
- MultiDoubleToStringConverter (multiplies all source values together and returns the result as a string)
- RGBToColorConverter (takes doubles as the source Red, Green and Blue values and returns a single Color object)

Data Bound Components and Controls

The Core Components already described can be used to create powerfully simple combinations of data and event driven applications. In order to get you started creating your specific application a bit faster Simple Bind also includes pre-defined Data Bound Components and Controls.

Each of the available Components and Controls will be mentioned in this document. Each component and control is also fully covered in the additional documentation and videos.

Items Control



Items Control (Script)

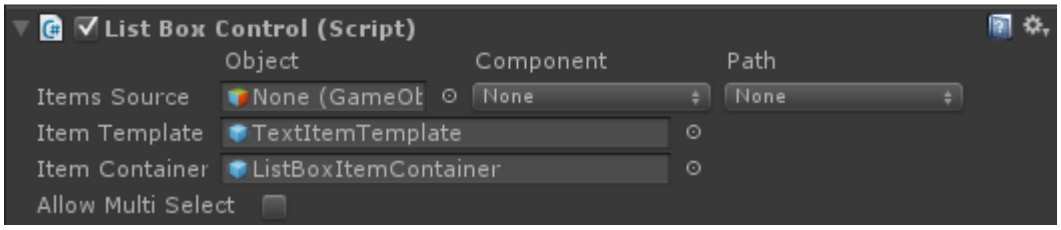
	Object	Component	Path
Items Source	None (GameOb	None	None
Item Templat	None (GameOb		

The Items Control is a single component that can be placed on any UI Game Object that can turn each element in the bound Items Source into a child of the Game Object by creating an instance of the Item Template for each item in the Items Source.

If the Item Template contains a Binding Context then the Value Source will be set to the item from the Items Source.

Using standard Unity Layout Components a simple Panel can turn into a data driven list of templated items.

List Box



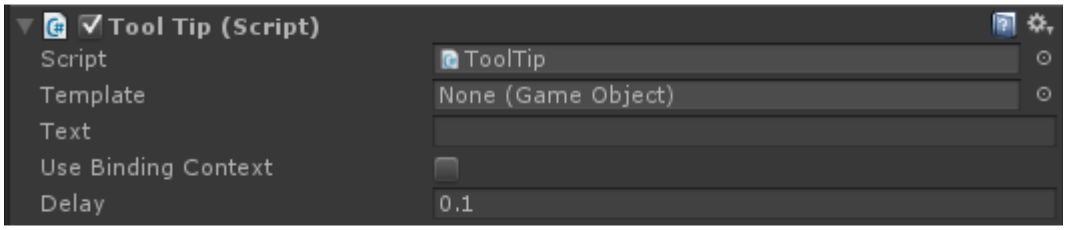
List Box Control (Script)

	Object	Component	Path
Items Source	None (GameOb	None	None
Item Template	TextItemTemplate		
Item Container	ListBoxItemContainer		
Allow Multi Select	☐		

The List Box adds to the simplicity of the Items Control by creating a complete List Box Prefab complete with scrolling, data template, item containers, selection and multi-selection.

There are still many features to be added to the List Box control, but we believe it is a great start.

Tool Tips



Tool Tip (Script)

Script	ToolTip
Template	None (Game Object)
Text	
Use Binding Context	☐
Delay	0.1

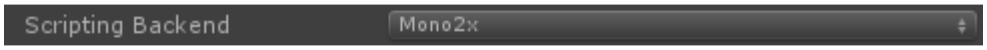
The Tool Tip and Tool Tip Controller components use the same Item Template and Data Binding approach as the other controls and allow you to easily assign custom tool tips to any uGUI Game Object.

iOS Specific Information

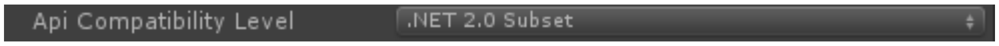
Our goal is to design Simple Bind in such a way that it will be compatible will all platforms that Unity supports.

Currently, Simple Bind has been tested on Windows, OS X, Android and iOS.

However, iOS has some specific requirements. When compiling for iOS you must set the following in the Player Settings:



Scripting Backend Mono2x



Api Compatibility Level .NET 2.0 Subset

Without these two setting on iOS Simple Bind may compile and run but it will not function properly.

We are continuing to investigate this issue and hope to find ways to support but IL2CPP and the full .Net 2.0 API.

More Information

For the latest information about Simple Bind and to find additional documentation, demos and tutorial videos go to http://sftsrc.com/unity.

If you have specific question or comments please e-mail us at unity@sftsrc.com.