UBindr Script Reference

A script reference for the UBindr asset package. For support contact mattias.fagerlund@carretera.se.

UBindr allows you to publish data from your C# code to the Unity UI with minimal to no code changes. It supports two way binding, so that changes made by the user are passed back to the C# code. It fills some of the same need that Angular, Knockout or React fills for Web applications. It makes building UIs easier and leaves the code cleaner.

Classes

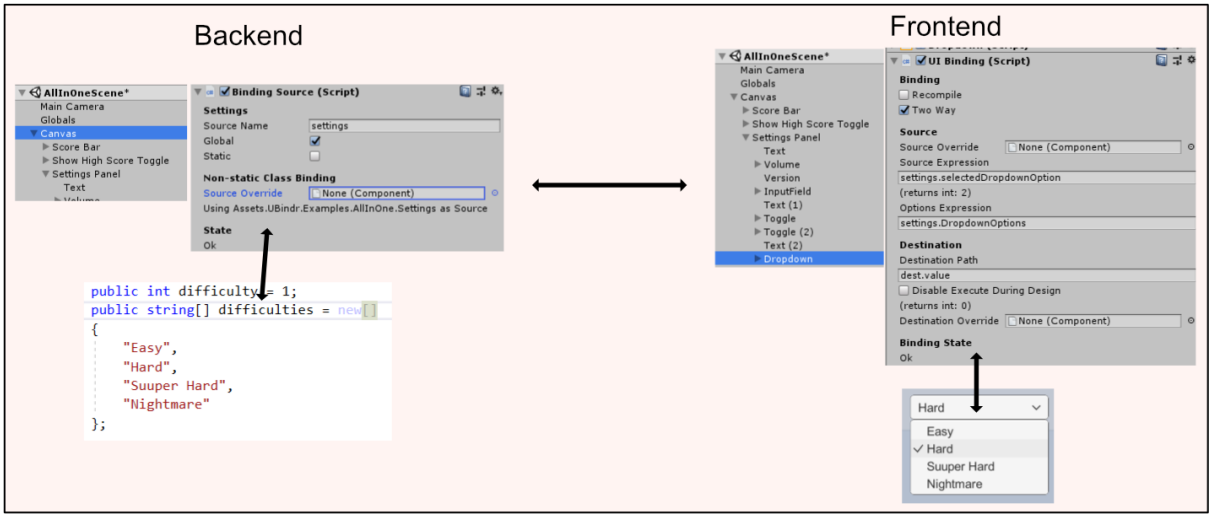
These are the classes available in UBindr. For a tutorial and a more usage based overview, see

- Overview
- Tutorial 1
- Tutorial 2
- Tutorial 3
- Tutorial Movie

BindingSource

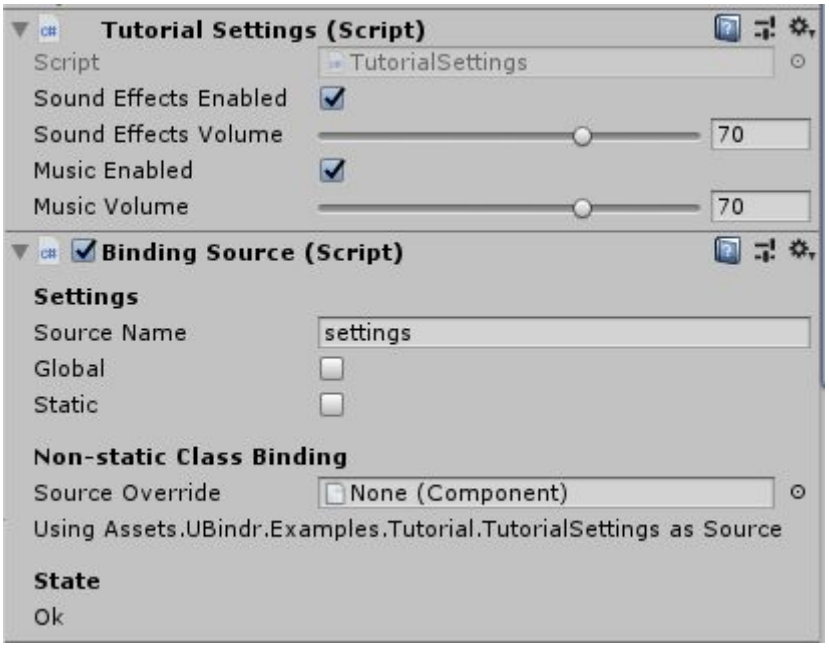
Inherits from MonoBehaviour, implements IBindingSource.

UBindr binds Binding Sources, through Bindings, to Destinations. The BindingSource class represents the data sources where you collect the data you wish to bind to. For instance, if you wanted to bind a Text label in the UI to the name of the player, you'd add a BindingSource that made the player available. Then you'd add a UIBinding to the Text label that bound the BindingSource to the ".text" field of the Text component.



A BindingSource can be global (available anywhere in the hierarchy) or non global (available in the branch of the hierarchy where it's found). It can be static (publishes a static class with static properties, like "Math") or non-static (publishes to an instance of a class).

By default, the BindingSource publishes the component above it in the GameObject on which it's located. You can override this behaviour to publish any other component by populating the sourceOverride field.



In this case, the BindingSource publishes "Tutorial Settings".

public string sourceName;

BindingSources is made available to Bindings using their sourceName. In the binding expression "settings.serverName", the "settings" part represents a Binding Source with the sourceName "settings". The "serverName" part represents a property or field in the source with the name "serverName". SourceNames don't need to be unique, but you can only access one BindingSource per name - if there are multiple only one will be in scope.

public Component sourceOverride;

If you don't want to publish the default source (the Script above the Binding Source) then you can provide an override here. This can be useful if you wish to publish a component that's located in a different hierarchy from the BindingSource.

public bool isGlobal;

A BindingSource can be global (available anywhere in the hierarchy) or non global (available in the branch of the hierarchy where it's found).

public bool isStatic;

A BindingSource can be static (publishes a static class with static properties, like “Math”) or non-static (publishes to an instance of a class).

public string staticClassName;

If the BindingSource is static, the staticClassName represents the static class that’s being published. The staticClassName should be fully qualified, for instance, Mathf has the fully qualified class name “UnityEngine.Mathf”.

Binding

Inherits from DestinedBinding.

A “raw” binding meant to use to bind any two properties together. You’ll rarely use this class, since there are other specialised bindings that can be used instead.

BindingWithBindingSources

Abstract base class, inherits from MonoBehaviour and implements IBinding.

Base class for all subsequent binding classes. Implements functionality to retrieve Source Bindings and make them available for binding expressions.

public bool recompileOnEachUpdate = false;

Compiling expressions is costlier than evaluating precompiled expressions. Therefore the default behaviour is to compile each expression once and re-use the compiled version for subsequent evaluations. You can override this behaviour by setting this field to true.

public Component sourceOverride;

Allows you to provide a specific source to a binding - other than the sources available through Binding Sources. If a binding has a sourceOverride it will be available using the name “source” in the list of Binding Sources.

DestinationContextedBinding

Abstract base class, inherits from SourcedBinding.

Implements “destination” handling, providing a Binding Source called “dest”. The default destination is the script just before the Binding. If the Destination field is populated, the value of that field will be used instead.

public Component Destination;

If this value is set, it will be presented as the Binding Source “dest”, it’s not set, the script/MonoBehaviour above the binding will be used as the Binding Source “dest”.

DestinedBinding

Inherits from DestinationContextedBinding.

public string DestinationPath;

The destination path that the binding will write to. For example: “dest.text”.

public bool TwoWay;

Determines if the binding is two way. A one way binding will copy and changes from the source to the destination. A two way binding will monitor both the source expression and the destination path and copy any changes from one to the other.
This is useful for input controls where the user can change the value of a control. For a label, where users can’t change the value, it’s pointless.

public bool DisableExecuteDuringDesign;

Disables the binding when in design mode. Can be useful when the DestinationPath is a method that alters the state of the controls that are being designed.

public bool ConvertDestinationToString;

Determines whether the result from the binding should be converted to a string before it’s copied over. It’s useful when binding for instance an int to an input field. You want to convert the binding result from an int to a string for the input control, then back to an int when writing the result back to the destination of the binding.

public bool ConvertSourceToTargetType;

Used in TwoWay bindings to determine if the binding value should be converted to the datatype of the source expression before it’s written back to the SourceExpression. See ConvertDestinationToString.

IBinding

An interface that represents a binding in the system.

MethodBinding

Inherits from BindingWithBindingSources.

A binding for binding to methods on the destination. The method binding will pick up all event triggers on the component just above it and when they're triggered, the method in the method binding will be executed. For instance, if you add a MethodBinding after a Button, the MethodBinding will execute when the button is pressed.

public Component Destination;

Usually a Binding will include the component just above it as a Binding Source named "dest". Populate this field if you wish to publish another component as "dest".

public string MethodPath;

The path to the method to invoke, for instance "settings.OpenServerUi()" or "player.SelectedItem.DropItem()".

RectTransformBinding

Inherits from DestinedBinding.

A binding for binding to fields on RectTransforms, fields that aren't usually visible in unity. The binding injects a Binding Source called "dest" that can be used in your expressions to either read from or write to. The exposed fields are;

public float height

The height of the RectTransform.

public float width

The width of the RectTransform.

public float x

The x position of the RectTransform in relation to its parent.

public float y

The y position of the RectTransform in relation to its parent.

RepeatBinding

Inherits from SourcedBinding, implements IBindingSource.

A binding for generating lists of objects. It will iterate over the values retrieved when executing the source expression and it will instantiate a copy of the "ItemPrefab" GameObject for each item it finds. The source expression must return an IEnumerable. If new items are added to the IEnumerable returned by the source expression, new

GameObjects will be instantiated. If existing items are removed, the matching GameObject will be destroyed.
All instantiated game objects will be parented to the transform on which the RepeatBinding exists. If you need to handle the positioning of the instantiated objects, you can either do that by the regular UI positioning scripts (Vertical Layout Group etc) or by using a RectTransformBinding and binding the x/y using expressions.
The binding supports sorting and you can limit how many rows are generated.

public string rowSourceName = "row";

Each child GameObject instantiated for a row in the bound IEnumerable will have access to a binding source representing the row. The name of that Binding Source will be whatever rowSourceName contains. If you have RepeatBindings within RepeatBindings, you can use different names for this property to access destinations higher up in the hierarchy. If all use “row”, the lowest accessible RepeatBinding will hide those higher up in the hierarchy.

public string orderBy;

An expression that’s used to sort the IEnumerable before it’s used. For instance “row.score” or “row.Values.Average()”

public bool descending = true;

Determines the sort order IEnumerable should ascending or descending. Used with orderBy.

public int maxItems = -1;

Allows the user to limit how many items from the IEnumerable should be included. -1 means all items are included.

public GameObject ItemPrefab;

The GameObject that should be instantiated for each item in the IEnumerable. The instantiated GameObject will be parented to the Transform that the RepeatBinding exists on.

SourcedBinding

An abstract class that inherits from BindingWithBindingSources.

A base class for other classes (DestinationContextedBinding and RepeatBinding).

public string SourceExpression;

The expression that’s used to get the source value of a binding. For instance, “settings.currentUser.name” or “row.score”.

TypedDestinedBinding

> *An abstract class that inherits from DestinedBinding. Extended by UIBinding, UIEnabledBinding and UIInteractableBinding*

Used internally to keep track of what GameObject type the destination component has (Button, Text, Input etc).

UIBinding

> *Inherits from TypedDestinationBinding.*

A class for binding a UI component (Toggle, Dropdown, InputField, Text, Slider, Scrollbar). The UIBinding Editor knows what field to bind to on the UI Component and if the binding should be TwoWay or not. The same functionality can be achieved using a regular Binding, but the UIBinding makes it much easier.

public bool copyDestinationToSourceExpression;

When designing the UI, it's convenient to add the binding expressions to the visible part of the ui - so that a Text or InputField component contains the text of the expression that its bound to.

public bool updateOnEndEdit = true;

Determines if the binding should be continuously bound for an InputField component. For some values, like float values, we don't want continuous binding, because float values are invalid throughout the process of editing them "123," is invalid but a necessary step to "123,4". If updateOnEndEdit is true, values are only copied at the end of the edit process. If updateOnEndEdit is false, the value is copied as it's updated.

public string OptionsExpression;

When the bound component is a Dropdown component, the OptionsExpression is used to populate the Options property of the Dropdown. The expression should evaluate to an IEnumerable of Dropdown.OptionData, which will be used as is, or a list of object, where each object will be added as an option with it's "ToString" value as the name.

public bool editorInitialized;

Internal field to keep track of whether the binding has been initialized in the editor or not.

UIEnabledBinding

Inherits from TypedDestinationBinding.

A binding that controls if the destination GameObject is enabled or not. The isEnabled field cannot be bound to directly, changes must be sent through gameObject.SetActive(XX), which this binding does behind the scenes. This binding takes a boolean expression and sets the isActive field. When a control is disabled, it becomes hidden. If you wish to bind to whether the control is readonly or not, use UIInteractableBinding instead.

Note that this binding shouldn't be placed on the GameObject that it's meant to control. When the enabled value is set to false, any scripts on the GameObject will no longer be run. So even if the binding would evaluate to true sometime in the future, it will never again be run. Instead you can place the binding on another GameObject - one that remains enabled - and set the destination to the game object you wish to control.

UIInteractableBinding

Inherits from TypedDestinationBinding.

A binding that controls if the destination control is readonly or not. All UI controls have a property called interactable (Toggle, Dropdown, InputField, Text, Slider, Scrollbar). This binding takes a boolean expression and sets the interactable property. If you want to show/hide a control, use the UIEnabledBinding instead.