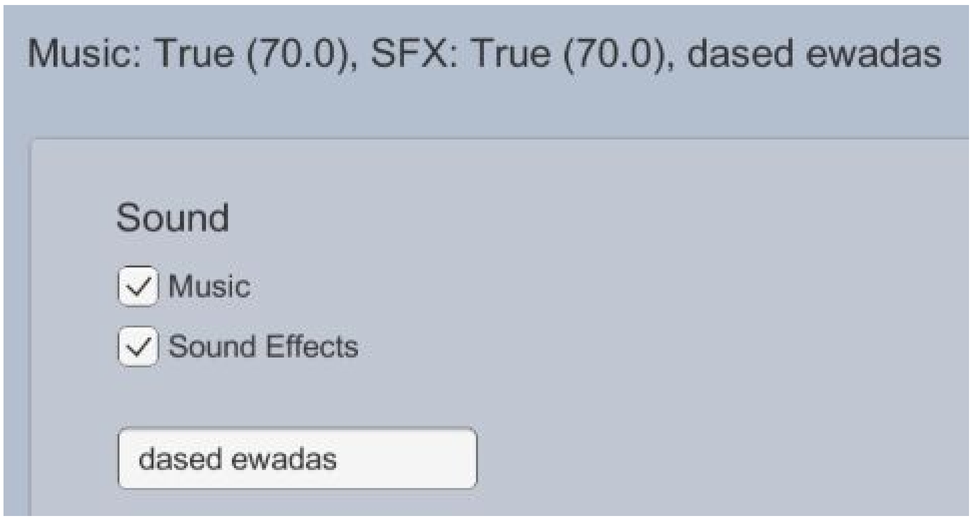
UBindr Tutorial 1

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.

The Simplest Of Bindings

In this tutorial, we'll demonstrate how UBindr can be used to create a GUI that looks like the picture below and uses bindings to connect the data from the model to the gui and back again (Two Way binding).

(This Tutorial assumes that you already know how to create panels and other GUI components)



Create a project

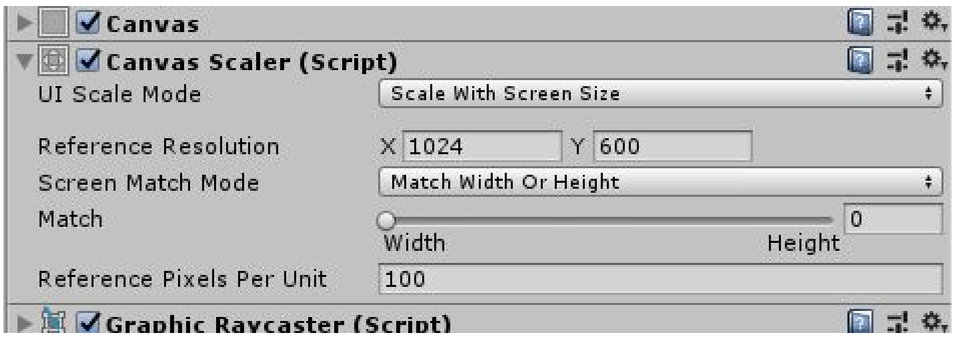
Create a scene

Include the UBindr Free or the UBinder Pro Asset.

Create a Panel and Setup the Camera

- Create a Panel (Right Click in the scene hierarchy panel and select UI | Panel).

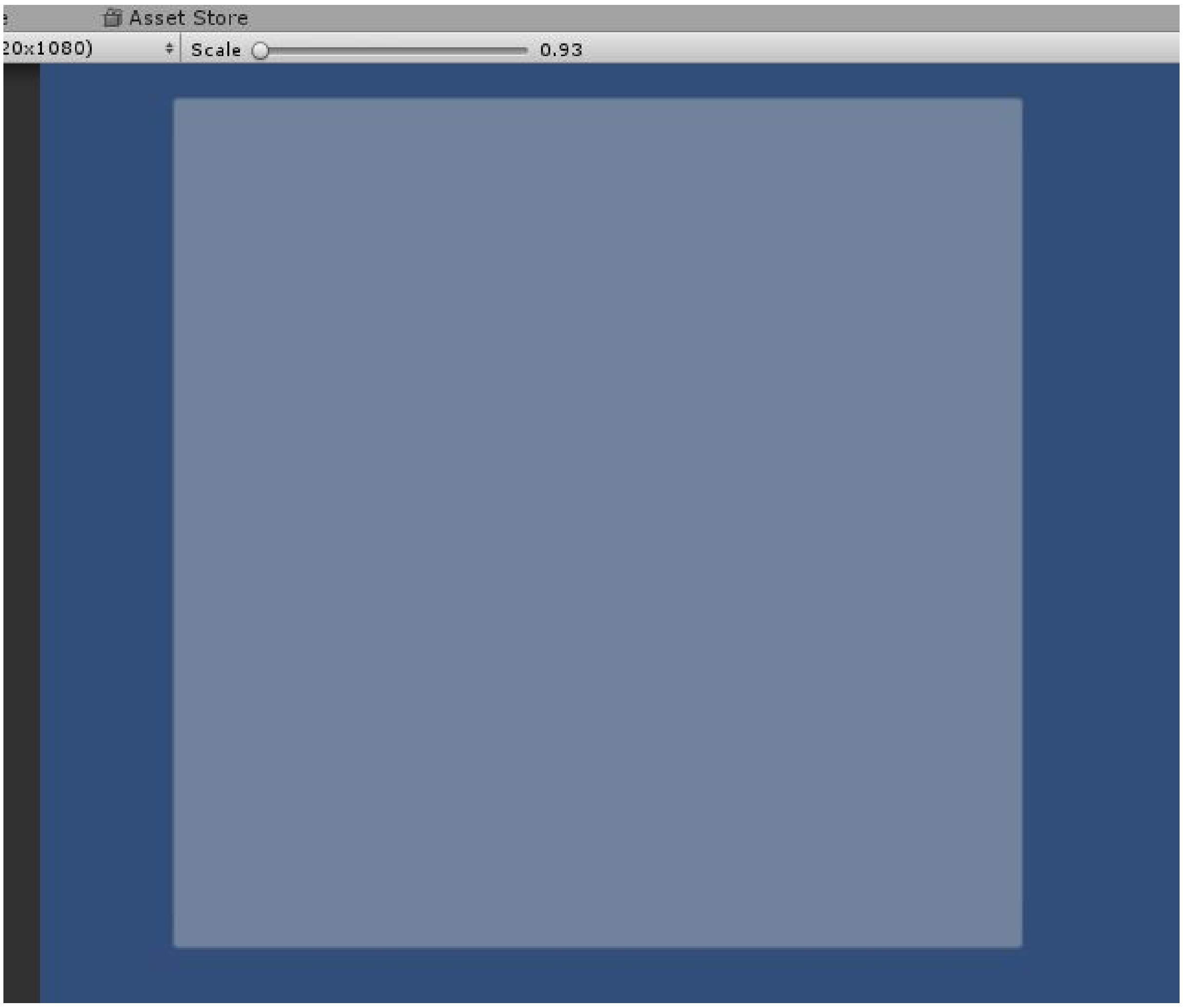
At this point a Canvas will be created, let's make the canvas scale with the screen size
- Select the Canvas component (it's located at the root of your hierarchy)
- Update your Canvas Scaler properties to look like this;



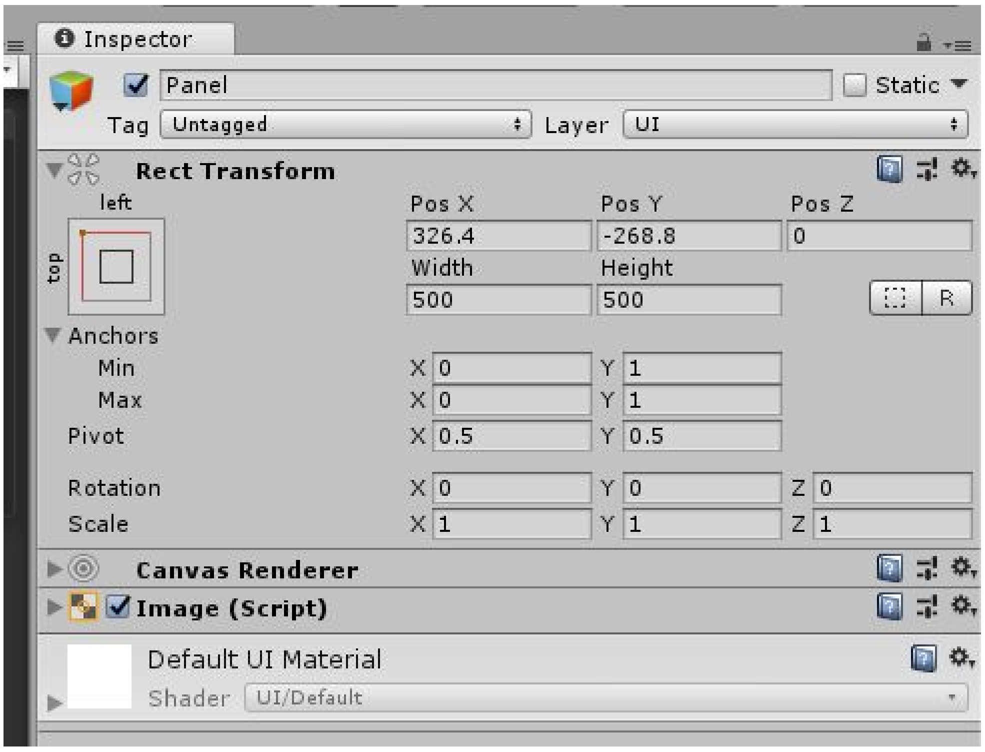
Canvas
Canvas Scaler (Script)
UI Scale Mode: Scale With Screen Size
Reference Resolution: X 1024 Y 600
Screen Match Mode: Match Width Or Height
Match: 0 (Width — Height)
Reference Pixels Per Unit: 100
Graphic Raycaster (Script)

Now let's get back to the panel
- Select the Panel you previously created
- Make it about 500x500
- Name it "Settings Panel"
- Place it close to the top left corner
- Anchor it to the top left corner

Gameview should look like this something like this;



The properties for the panel something like this;



Panel
Tag Untagged Layer UI

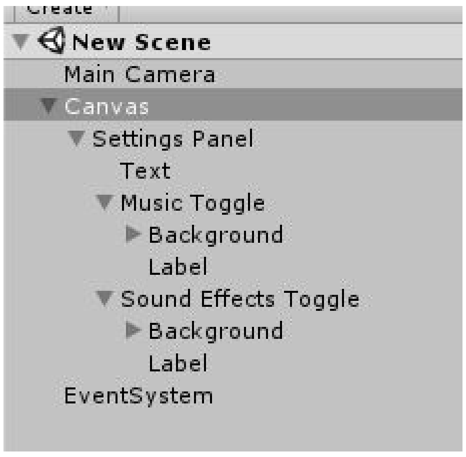
	Pos X	Pos Y	Pos Z
	326.4	-268.8	0
	Width	Height	
	500	500	

	X	Y	Z
Anchors Min	0	1	
Anchors Max	0	1	
Pivot	0.5	0.5	
Rotation	0	0	0
Scale	1	1	1

Shader UI/Default

- Add a Text component
- Change the font size to 18 and the text to Sound
- Add a Toggle Component,
    - name it Music Toggle
    - change the label to Music
- Add another Toggle Component
    - name it Sound Effects Toggle
    - change the label to Sound Effects
- Make sure these components are childed to the panel
- Make sure these components are placed in an aesteticly pleasing manner

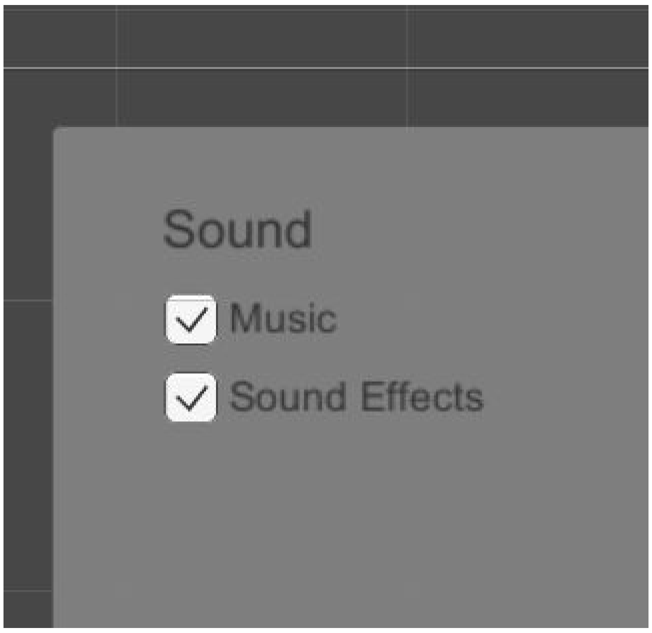
Your Hierarchy should look like this;



```
New Scene
  Main Camera
  Canvas
    Settings Panel
      Text
      Music Toggle
        Background
        Label
      Sound Effects Toggle
        Background
        Label
  EventSystem
```

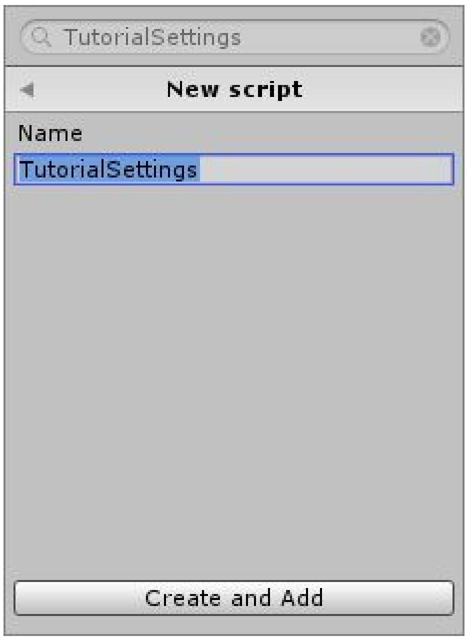
And your Scene View like this;

Rev 1.0. For support contact mattias.fagerlund@carretera.se.



Create the model

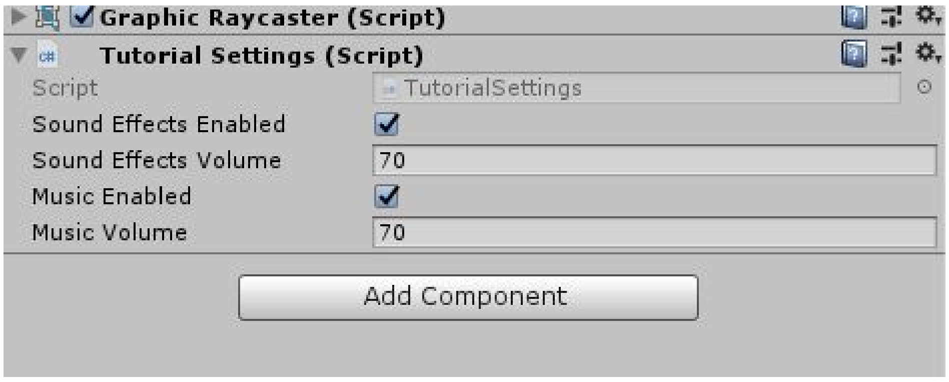
Add a new component called “TutorialSettings” to your Canvas game object;



- Edit the script to look like this;

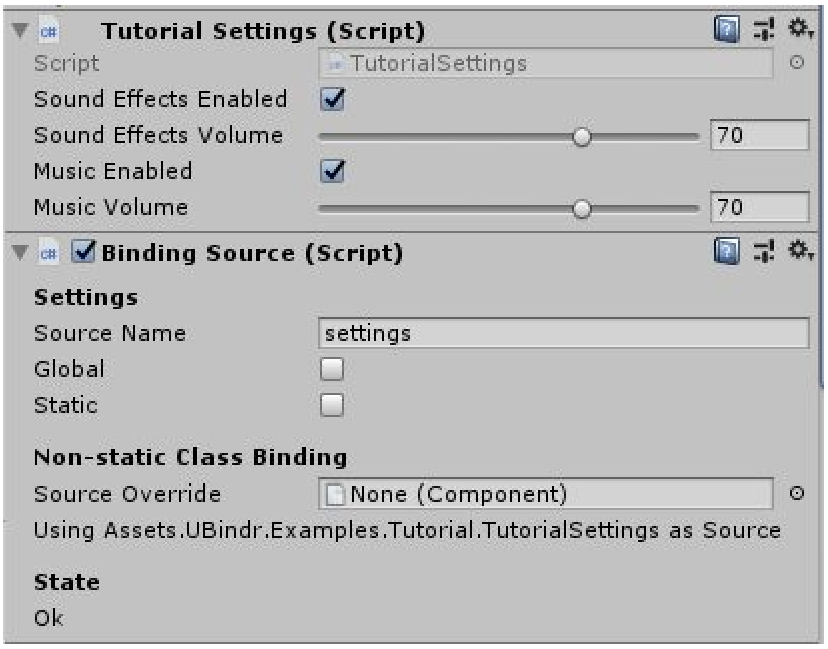
```
public class TutorialSettings : MonoBehaviour
{
        public bool soundEffectsEnabled = true;
        public float soundEffectsVolume = 70;

        public bool musicEnabled = true;
        public float musicVolume = 70;
}
```

Your canvas properties should not look like this;


Graphic Raycaster (Script)
Tutorial Settings (Script)
Script TutorialSettings
Sound Effects Enabled ✓
Sound Effects Volume 70
Music Enabled ✓
Music Volume 70
Add Component

Now we'll make the Settings available to the UBindr Bindings.

- Select the canvas
- Add a Binding Source component
- Give it the name "settings"
- Leave everything else as is

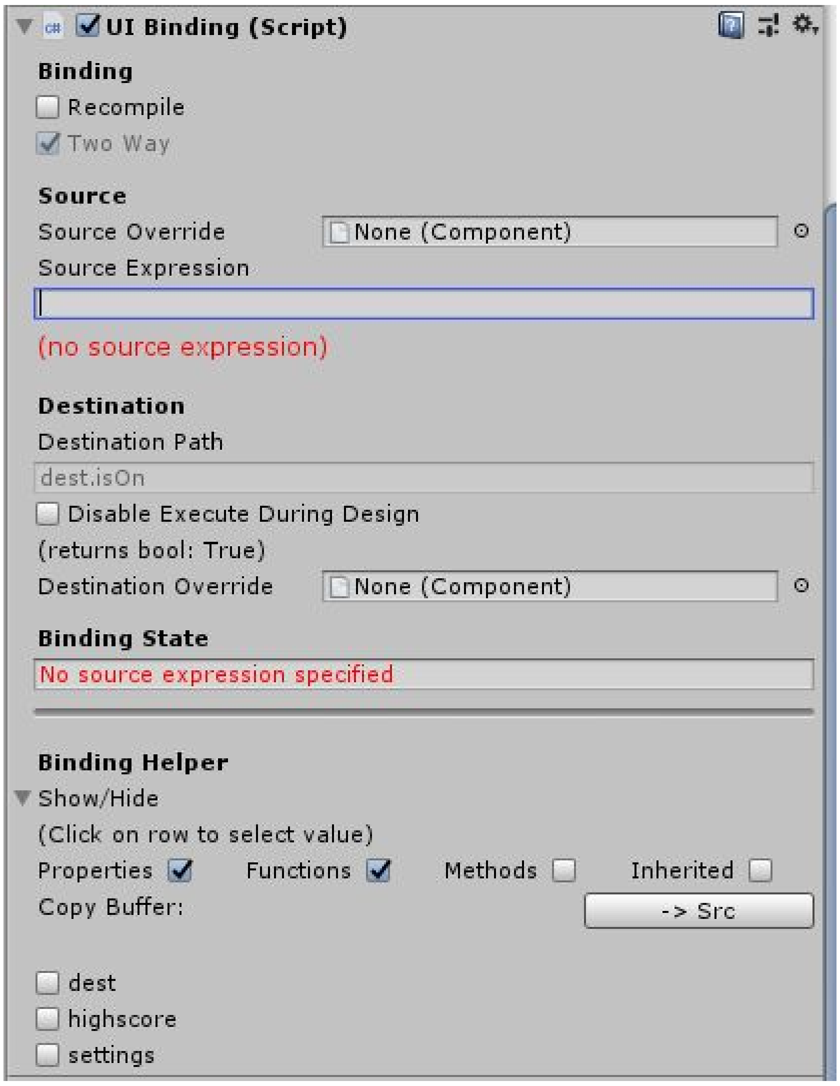

Tutorial Settings (Script)
Script TutorialSettings
Sound Effects Enabled ✓
Sound Effects Volume 70
Music Enabled ✓
Music Volume 70
Binding Source (Script)
Settings
Source Name settings
Global
Static
Non-static Class Binding
Source Override None (Component)
Using Assets.UBindr.Examples.Tutorial.TutorialSettings as Source
State
Ok

Any binding that is a child or grandchild of the Canvas will now see the settings component and be able to bind to it - using the name "settings".

Add the First Binding

- Select the "Music Toggle" toggle in the scene view
- Add a "UI Binding" component (not any other kind of Binding, specifically a UI Binding)

It'll look really angry at first, like this;



UI Binding (Script)

Binding

☐ Recompile

✓ Two Way

Source

Source Override: None (Component)

Source Expression

(no source expression)

Destination

Destination Path

dest.isOn

☐ Disable Execute During Design

(returns bool: True)

Destination Override: None (Component)

Binding State

No source expression specified

Binding Helper

Show/Hide

(Click on row to select value)

Properties ✓ Functions ✓ Methods ☐ Inherited ☐

Copy Buffer: -> Src

☐ dest

☐ highscore

☐ settings

You'll notice how the Destination Path is pre-populated and disabled. This is because the UI Binding knows how to connect to a Text Component. In fact, the UI component knows how to connect to the following components;

Regular Unity UI Components

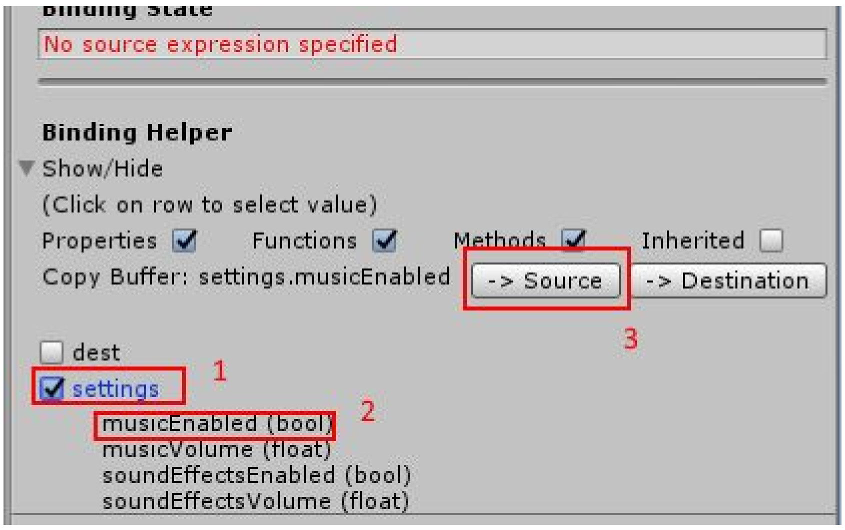
- Toggle
- Dropdown
- InputField
- Text
- Slider

Text Mesh Pro UI Components

- TextMeshProUGUI
- TMP_InputField
- TMP_Dropdown

It automatically enables "Two Way" binding, all we have to do is hook up the source to our model;

- Make sure you've selected the Music Toggle
- Scroll to the UI Binding that you just created
- In the Binding Helper Region, Click the "settings" checkbox
- Click the musicEnabled (boolean) label (it's really a button)
- Click the -> Source button

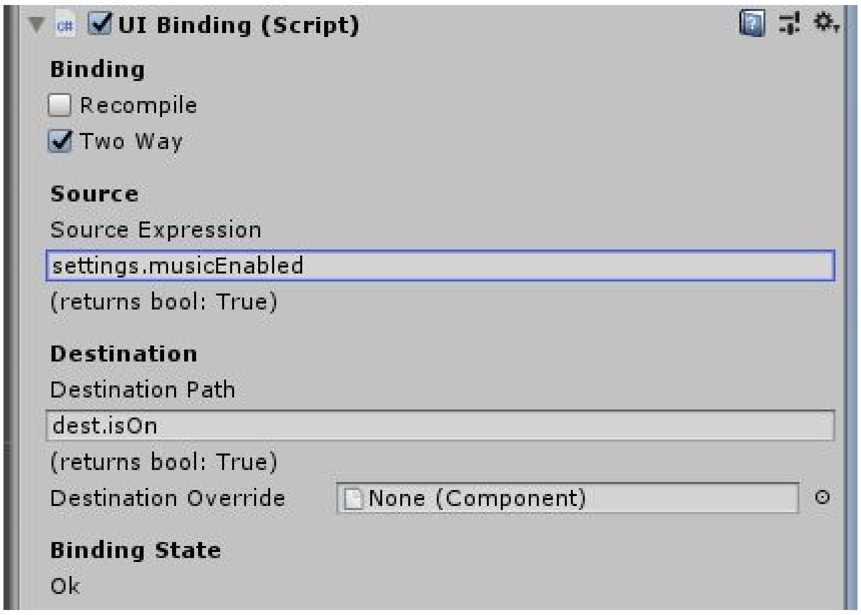


Binding State
No source expression specified

Binding Helper
▼ Show/Hide
(Click on row to select value)
Properties ☑ Functions ☑ Methods ☑ Inherited ☐
Copy Buffer: settings.musicEnabled

☐ dest
☑ settings
musicEnabled (bool)
musicVolume (float)
soundEffectsEnabled (bool)
soundEffectsVolume (float)

- Make sure you now have the text settings.musicEnabled in the Source Expression field and the component isn't so angry anymore



☑ UI Binding (Script)

Binding
☐ Recompile
☑ Two Way

Source
Source Expression
settings.musicEnabled
(returns bool: True)

Destination
Destination Path
dest.isOn
(returns bool: True)
Destination Override None (Component)

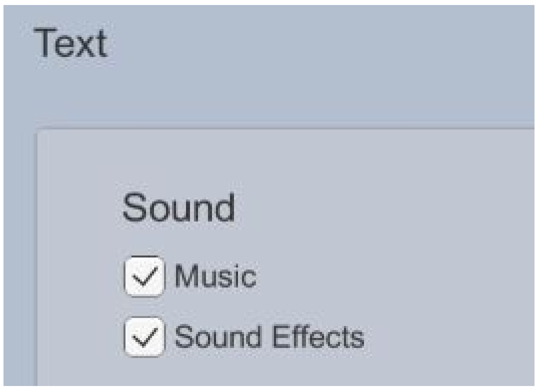
Binding State
Ok

That's it! First binding done! If you run the game and click the "Music" Toggle, you should find that the Settings component is updated as well.

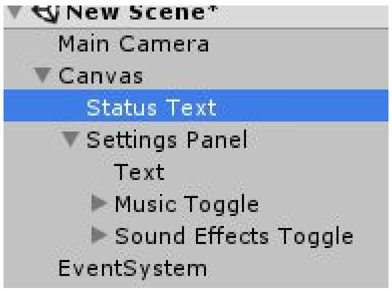
Some Feedback

Looking at the Settings component to verify that everything is working quickly becomes tiresome. So we'll add some feedback instead.

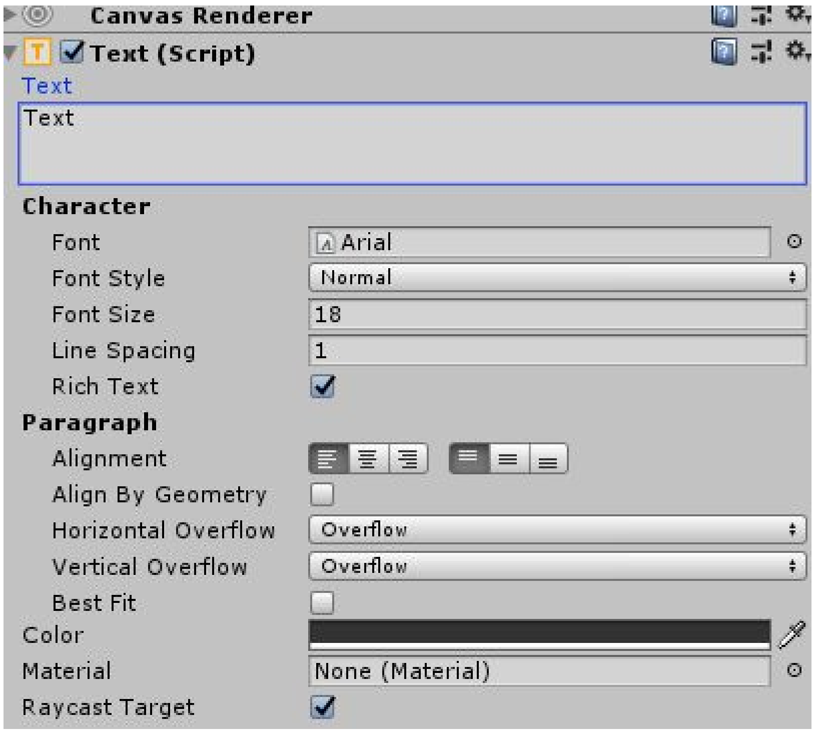
- Add a new Text UI Component above the panel we've just created



- Rename it to Status Text

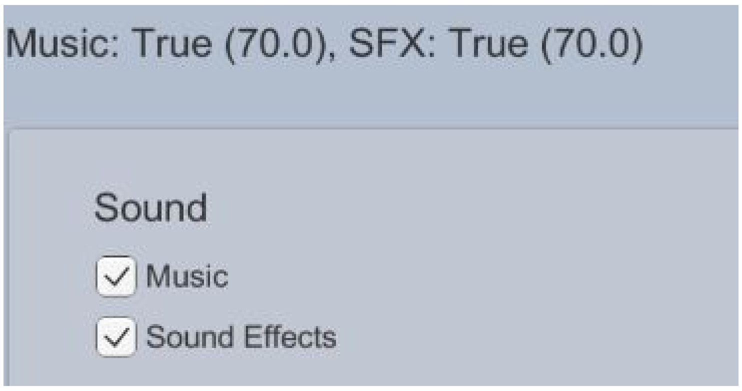


- Make sure it will overflow if the text is too long


Canvas Renderer
Text (Script)
Text
Text
Character
Font: Arial
Font Style: Normal
Font Size: 18
Line Spacing: 1
Rich Text: ✓
Paragraph
Alignment
Align By Geometry: ☐
Horizontal Overflow: Overflow
Vertical Overflow: Overflow
Best Fit: ☐
Color
Material: None (Material)
Raycast Target: ✓

- Change the text to "Music: {settings.musicEnabled} ({settings.musicVolume:0.0}), SFX: {settings.soundEffectsEnabled} ({settings.soundEffectsVolume:0.0})" **(make sure to include the quotation marks)**
- Add a UI Binding below the text component

If you got all that correct, the UI Binding should Bind to the Text component above, and it will already know what Source Expression to use, the one that's present in the text field. So if you run the game, it should look something like this;

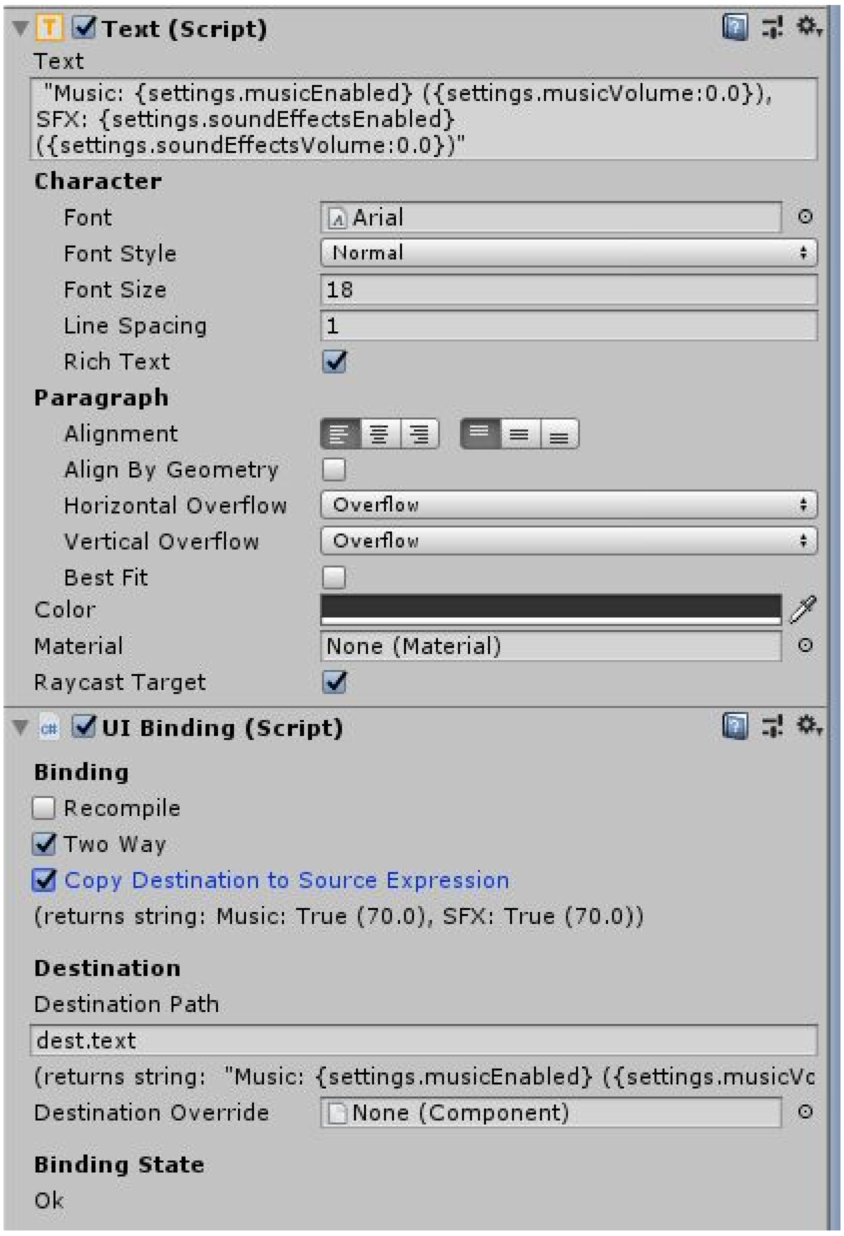

Music: True (70.0), SFX: True (70.0)

Sound

✓ Music

✓ Sound Effects

The UI Binding should look like this;


Text (Script)

Text

```
"Music: {settings.musicEnabled} ({settings.musicVolume:0.0}),
SFX: {settings.soundEffectsEnabled}
({settings.soundEffectsVolume:0.0})"
```

Character

Font	Arial
Font Style	Normal
Font Size	18
Line Spacing	1
Rich Text	✓

Paragraph

Alignment	
Align By Geometry	☐
Horizontal Overflow	Overflow
Vertical Overflow	Overflow
Best Fit	☐
Color	■
Material	None (Material)
Raycast Target	✓

UI Binding (Script)

Binding

☐ Recompile

✓ Two Way

✓ Copy Destination to Source Expression

(returns string: Music: True (70.0), SFX: True (70.0))

Destination

Destination Path

dest.text

(returns string: "Music: {settings.musicEnabled} ({settings.musicVc

Destination Override: None (Component)

Binding State

Ok

Rev 1.0. For support contact mattias.fagerlund@carretera.se.

Hook up the Second One

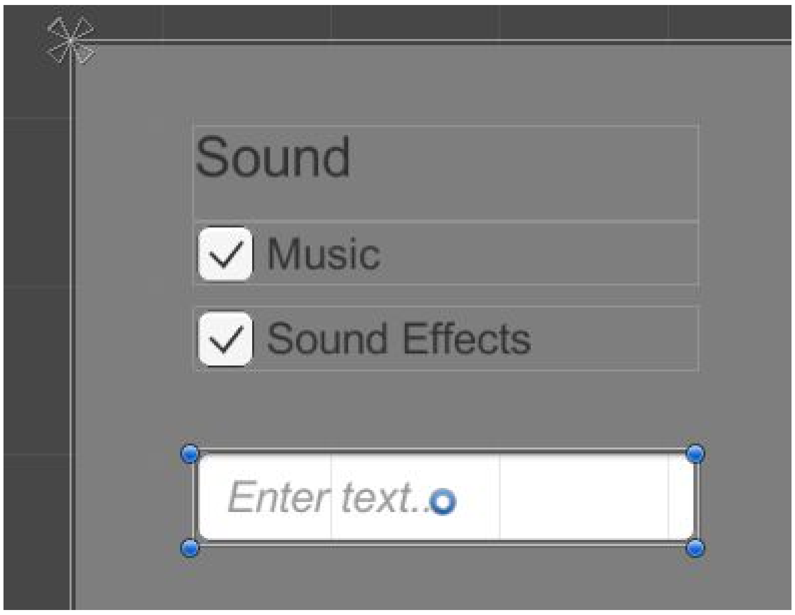
Hook up the Sound Effects Toggle to the soundEffects field on the settings component. This should be easy enough, check “Add the First Binding” above if you get stuck.

Add a String Input Field

- Edit the Tutorial Settings class
- Add a string field called “message”
- It should now look like this;

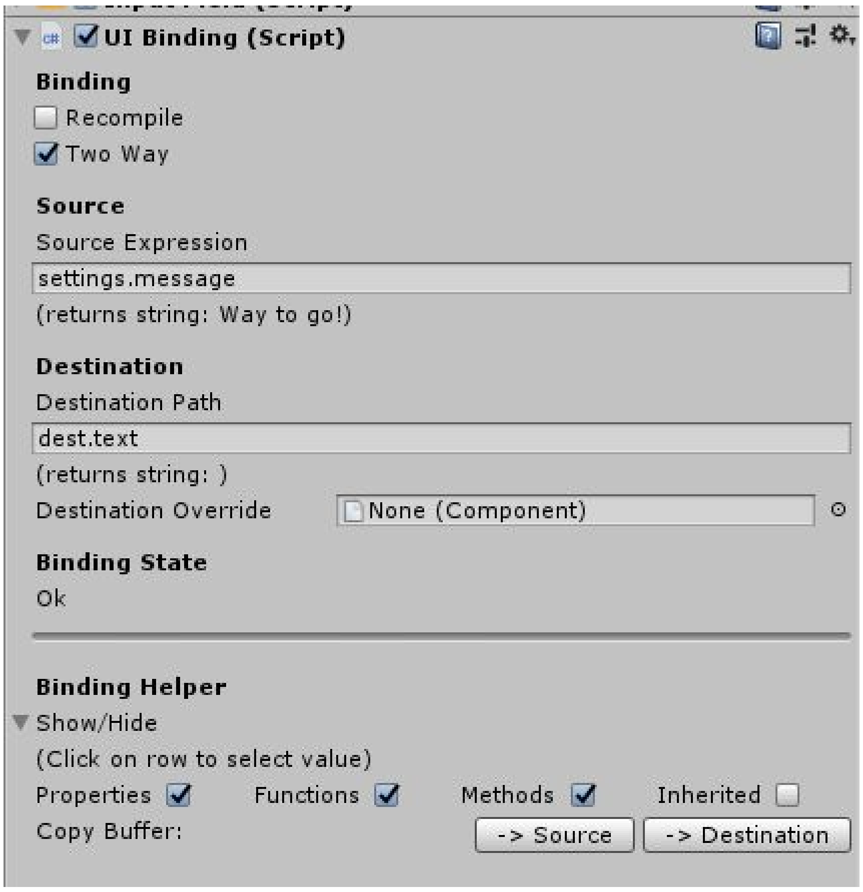
```
public class TutorialSettings : MonoBehaviour
{
        public bool soundEffectsEnabled = true;
        public float soundEffectsVolume = 70;

        public bool musicEnabled = true;
        public float musicVolume = 70;

        public string message = "Way to go!";
}
```

- Create an UI Input Field Component
- Place it under the Toggles



- Change the name to Message InputField
- Add a UI Binding

- Bind it to settings.message by using the Binding Helper



UI Binding (Script)

Binding

☐ Recompile

☑ Two Way

Source

Source Expression

settings.message

(returns string: Way to go!)

Destination

Destination Path

dest.text

(returns string:)

Destination Override: None (Component)

Binding State

Ok

Binding Helper

Show/Hide

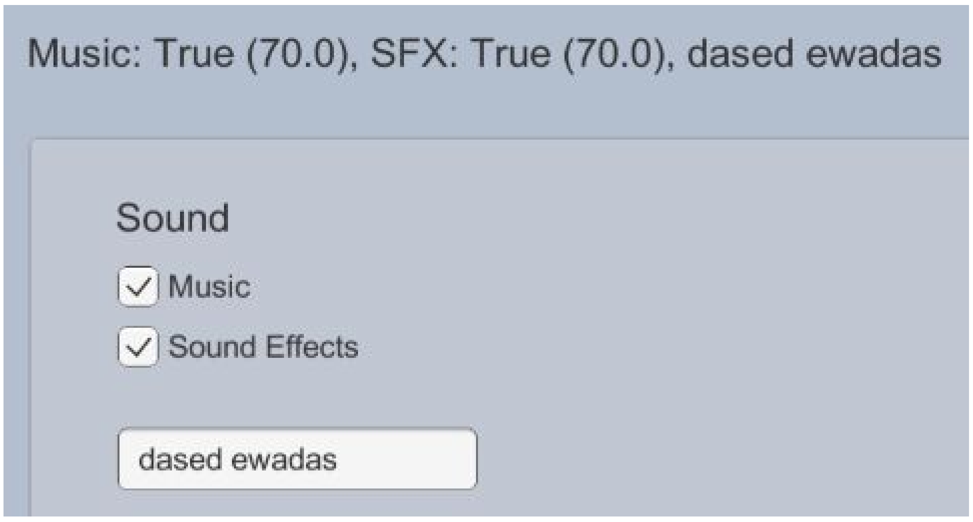
(Click on row to select value)

Properties ☑ Functions ☑ Methods ☑ Inherited ☐

Copy Buffer:

- Edit the Status Text Component to contain the text "Music: {settings.musicEnabled} ({settings.musicVolume:0.0}), SFX: {settings.soundEffectsEnabled} ({settings.soundEffectsVolume:0.0}), {settings.message}"

When you run the program, whatever you enter into the InputField will also be shown in the Status Text;



Music: True (70.0), SFX: True (70.0), dased ewadas

Sound

☑ Music

☑ Sound Effects

dased ewadas

That’s it for UBindr Tutorial 1. For Tutorial 2*, we’ll hook up a slider. You may already be able to guess how that’s done. Keep the code from this tutorial, because Tutorial 2 will build on what we created here.*