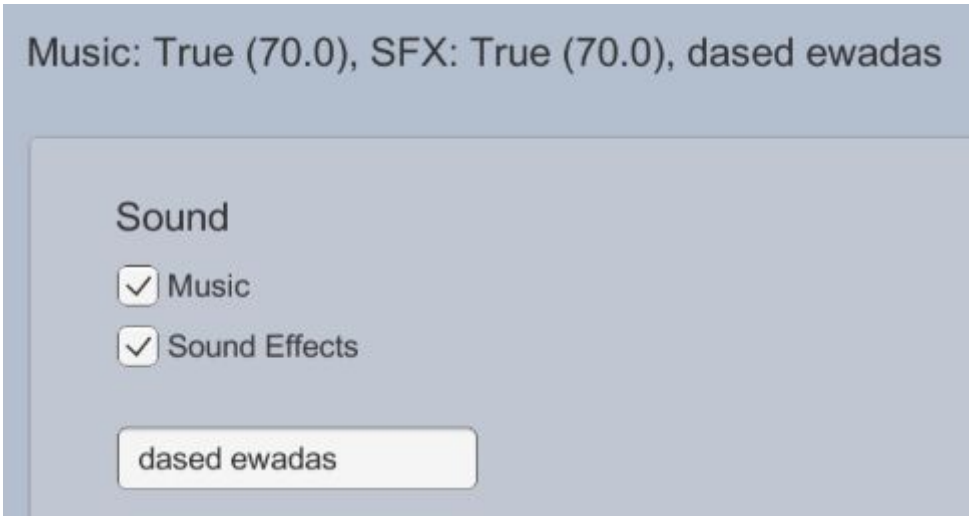
UBindr Tutorial 2

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.

Slightly Trickier Bindings

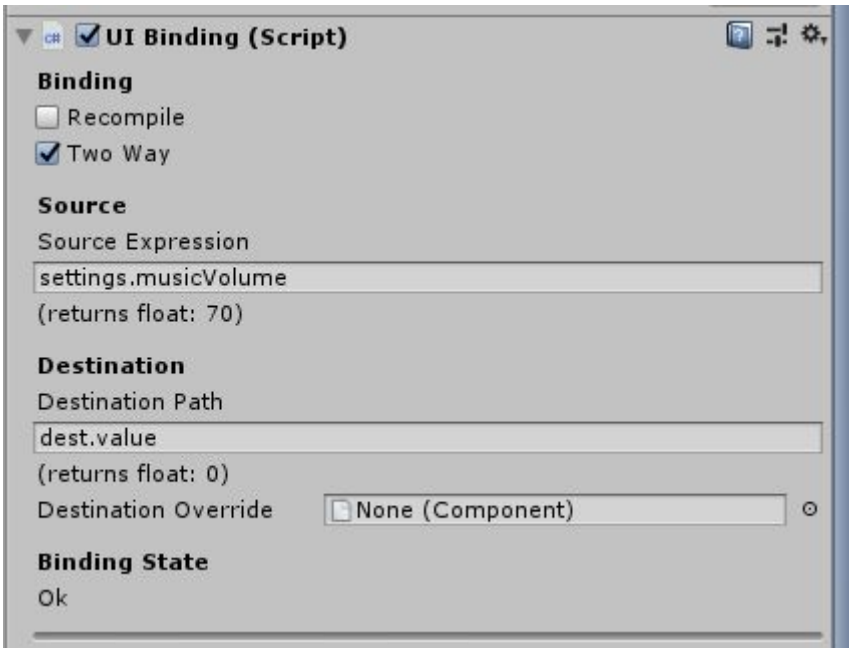
In this tutorial, we'll demonstrate how UBindr can be used to create a GUI that looks like the picture below and uses bindings to connect the data from the model to the gui and back again (Two Way binding).

I'm assuming you already have a UI that looks like this (from Tutorial 1);



- Create a Slider
- Place the Slider to the right of the Music Toggle
- Rename the slider "Music Volume Slider"
- Add a UI Binding to the Slider

- Bind the Slider to “settings.musicVolume” (using the Binding Helper)



Running the game, you’ll notice that the sliding the slider updates the Status Text - but the value varies from 0 to one. This is because that’s the default range of the slider. You can change the min/max values of the slider, but another way of doing that is to edit the model and add a Range attribute to the volume fields!

- Edit the script to look like this;

```
public class TutorialSettings : MonoBehaviour
{
        public bool soundEffectsEnabled = true;
        [Range(0, 100)]
        public float soundEffectsVolume = 70;

        public bool musicEnabled = true;
        [Range(0, 100)]
        public float musicVolume = 70;

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

- Save the script
- Select the Slider in the Hierarchy
- Min Value and Max Value should now have been updated to 0 and 100.
- In fact, you cannot change them in the Editor, as long as the range attribute is there, the UI Binding will keep updating them.

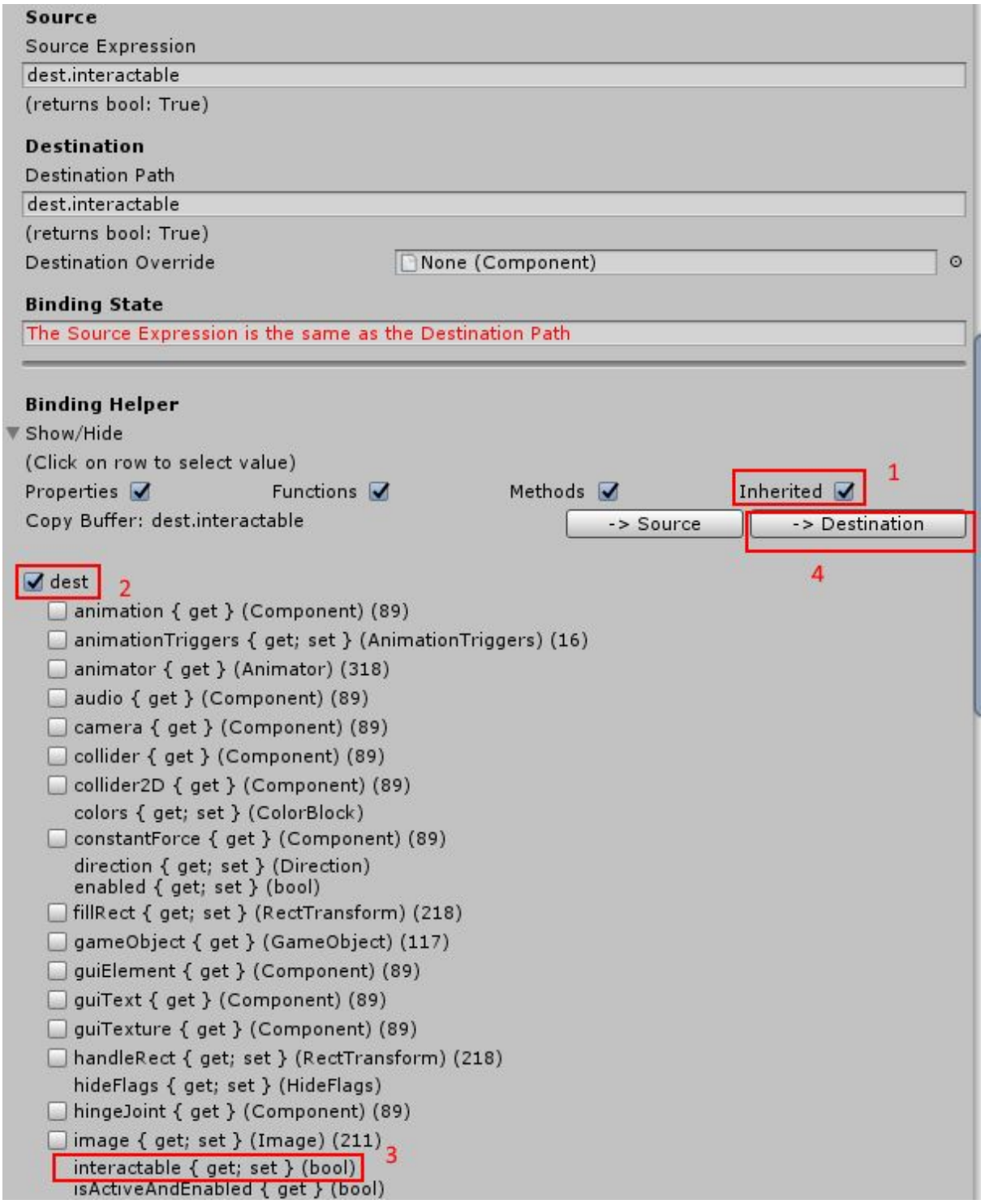
If you run the game again, you should find that the binding can be changed from 0-100

Disabling Fields

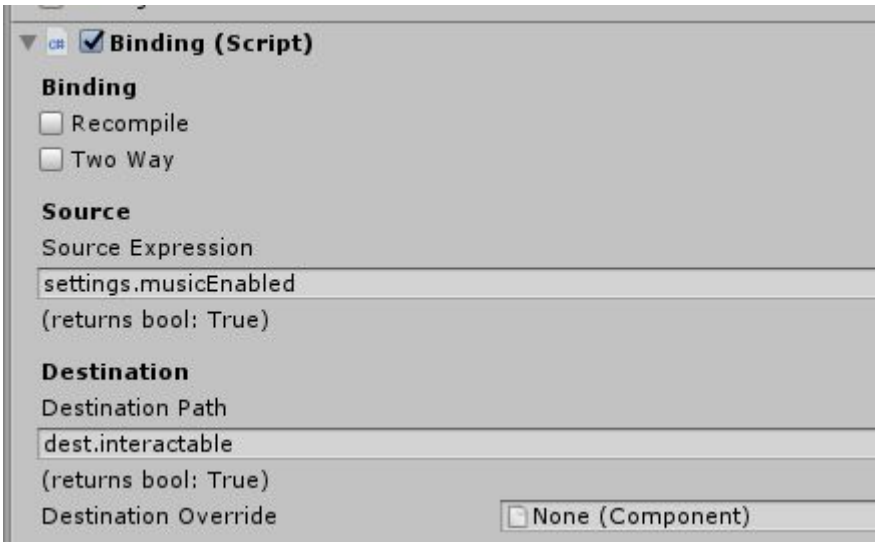
You may decide that if users don't want music, they shouldn't be able to change the music volume. That's easy enough;

- Add a **Binding** to the Music Volume Slider (not a UI Binding, just a Binding)
- In the Binding Helper
    - Enable the "inherited" checkbox
    - Expand the "dest" hierarchy
    - Click the interactable field
    - Click the -> Destination button

- The binding should look like this;



- Bind the Source to settings.musicEnabled



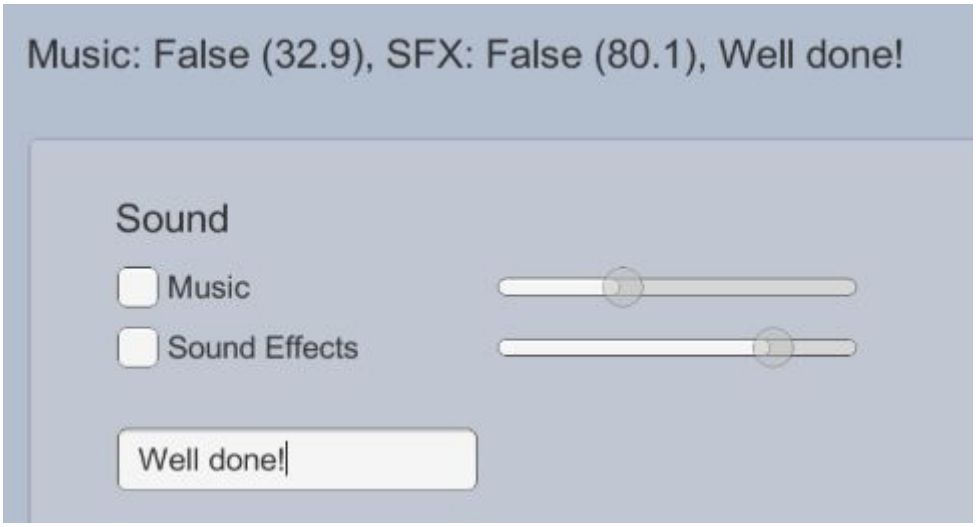
If you run the game, you should find that the slider is disabled when music is disabled;

and enabled when music is enabled;

Create Another Slider

- Create another slider to the right of the Sound Effects Toggle
- Rename this to Sound Effects Slider
- Create a UI Binding on the Sound Effects Slider
- Bind it to settings.soundEffectsVolume
- Add a new **UI Interactable Binding**
    - Bind it to settings.soundEffectsEnabled
    - It should automatically bind the Destination Path to dest.interactable
- Run the game

Both sliders should be connected.

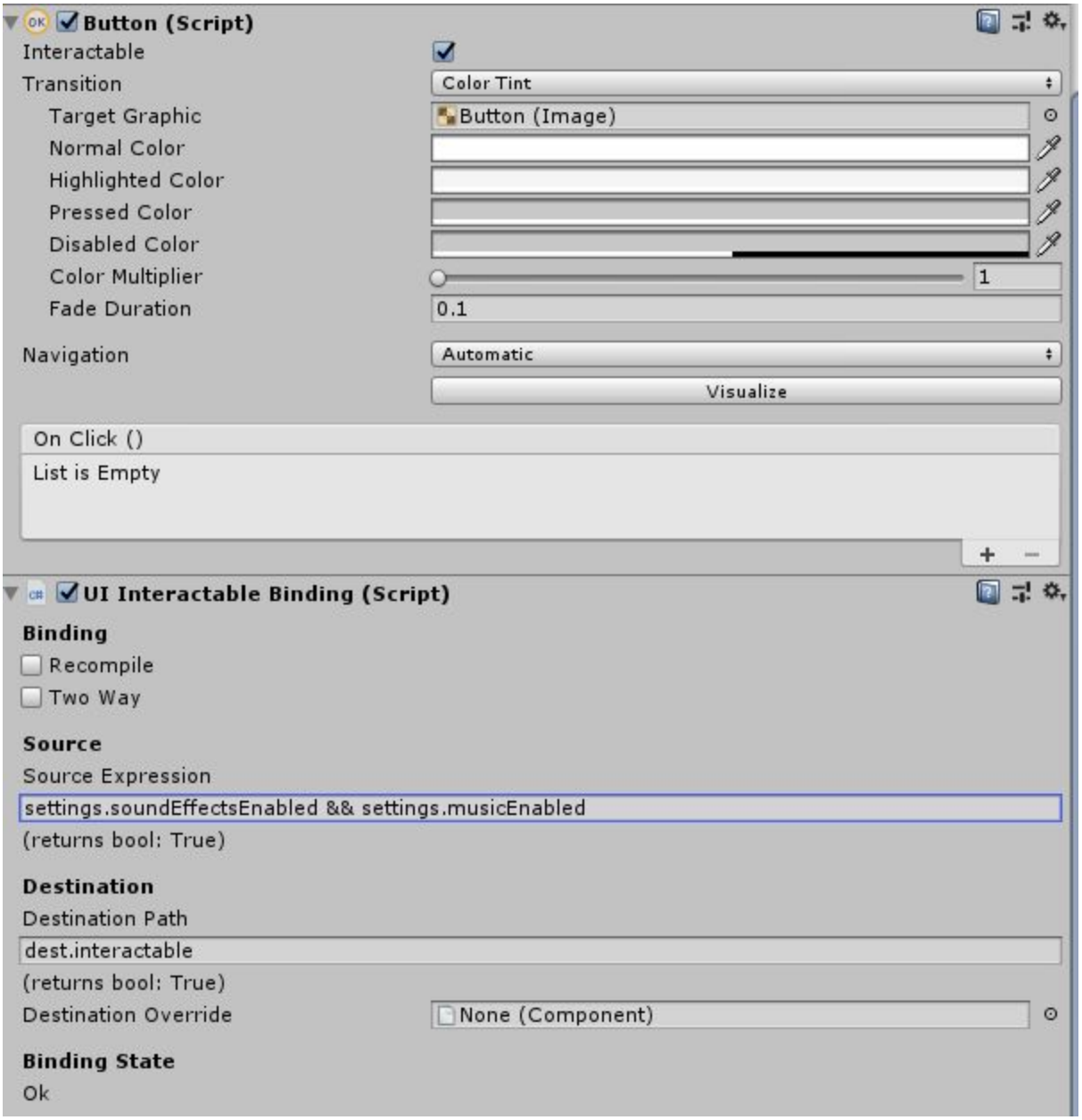


The “UI Interactable Binding” automatically binds to the dest.interactable field of the component it belongs to. It’s easier than using the binding field.

The Source Expression is Really an Expression

The source binding is an Expression, so it can be made as complicated as you like. For instance, let’s make a Sound Fest button that’s only enabled whenever BOTH Music and Sound Effects are enabled.

- Add a Button
- Place it to the right of the InputField
- Change the text to “Sound Fest”
- On the Button Game Object, add a UI Interactable Binding
    - Not that when you changed the text of the button to “Sound Fest”, you selected the Text of the button. Your binding should **not** go there, it should be placed on the button
- Change the Source Expression to settings.soundEffectsEnabled && settings.musicEnabled

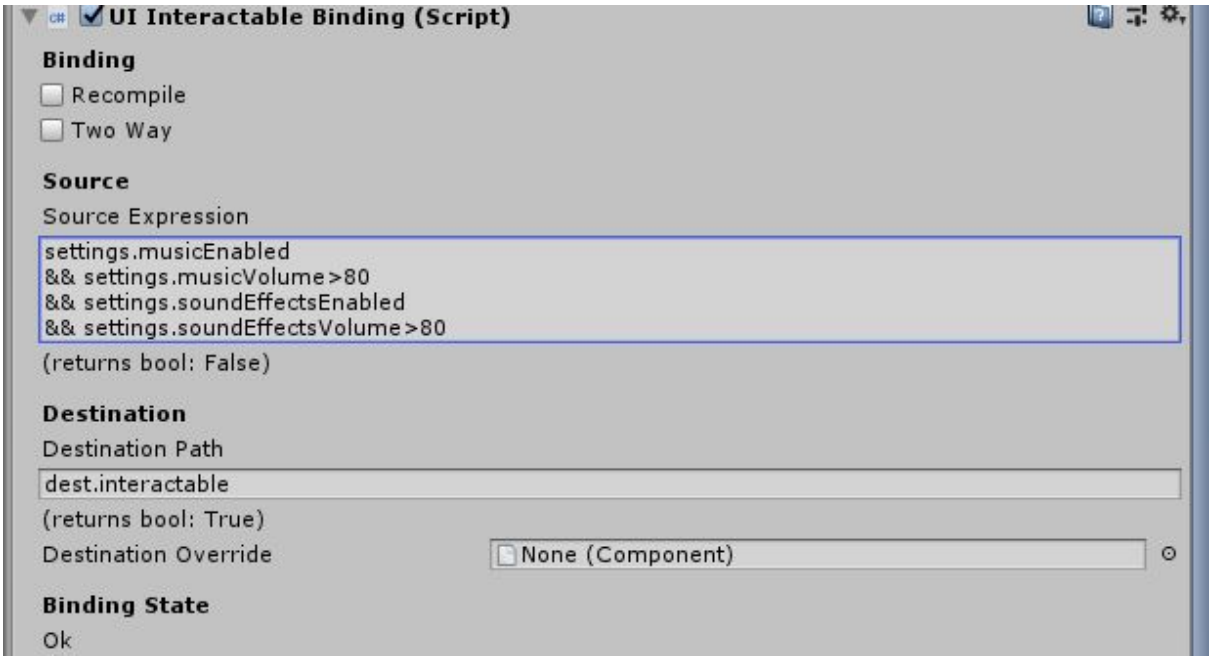


If you run the game, the button should only be enabled when both musicEnabled && soundEffectsEnabled.

But any Sound Fest would be pitiful if the sound volume was very low! So let's make the sound fest button be enabled only when the volumes are higher than 80.

- Edit the UI Interactable Binding, change the Source Expression to
  settings.musicEnabled
  && settings.musicVolume>80
  && settings.soundEffectsEnabled

&& settings.soundEffectsVolume>80



Run the game now and make sure the binding works as intended.

How About That Sound Fest Button?

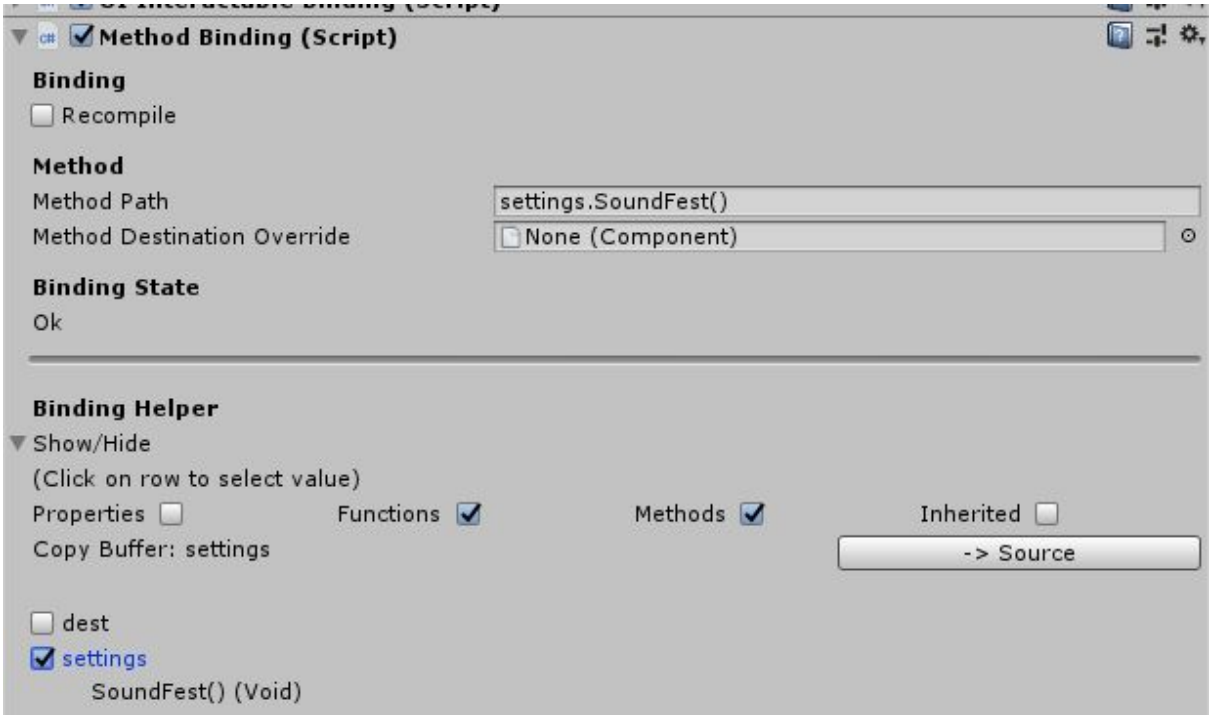
Clicking the Sound Fest Button does *nothing*, which is very sad indeed. Not very festive. So let's remedy that.

- Edit your TutorialSettings scripts
- Add a method called SoundFest;

```
public void SoundFest()
{
        message = "FIESTA!";
        musicVolume += 1;
        soundEffectsVolume -= 1;
}
```

- Save your script
- Select your Button
- Rename it Sound Fest Button
- Add a Method Binding

- Use the Binding Helper to bind the settings.SoundFest() method to the Method Path



If you run the game, clicking the button should increase the music and decrease the sound effects. If you keep clicking the button, it should eventually be disabled because the sound effects becomes too low.

Oh, and the word “FIESTA!” should appear in your Input Field.

One More Expression for the Road

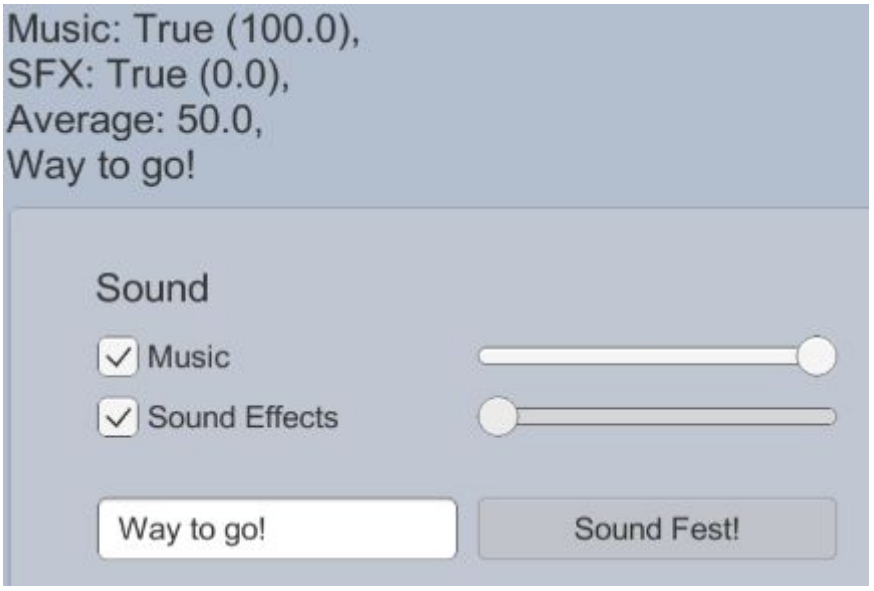
The Status Text component already contains an expression that should look like this;

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

The format is a simplified c# string interpolation format. Lets add an average volume for good measure.

- Change the Status Text Component to contain this text;
  "Music: {settings.musicEnabled} ({settings.musicVolume:0.0}),
  SFX: {settings.soundEffectsEnabled} ({settings.soundEffectsVolume:0.0}),
  Average: {(settings.musicVolume+settings.soundEffectsVolume)/2:0.0},
  {settings.message}"

Note that I added line breaks, you may have to move stuff around to make room for the new text.



The average of 100 and 0 is 50, so it seems to be correct.

Continue

That's it for UBindr Tutorial 2. For Tutorial 3, we'll hook up a Dropdown component and we'll create a list of objects where a Prefab is used as a template for each row. And we'll sort the list. Really. It'll be glorious.