SCAFFOLDING

Version 1.0

Owners Introductory Manual

```
namespace Scaffolding
{
    [AddComponentMenu("Scaffolding/Backend/View Manager")]
    public class ViewManager : MonoBehaviour
    {
        namespace Scaffolding
        {
            [AddComponentMenu("Scaffolding/View/AbstractView")]
            [RequireComponent(typeof(Animation))]
            /// <summary>
            /// Abstract view. The base class for any view you wish to create, whether it
            /// </summary>
            public class AbstractView : MonoBehaviour
            {
                namespace Scaffolding
                {
                    [AddComponentMenu("Scaffolding/Inputs/Abstract Input")]
                    /// <summary>
                    /// Abstract input is responsible for all inputs responding to a touc
                    /// </summary>
                    public class AbstractInput : MonoBehaviour
                    {
                        private InputManager _inputManager;
                        internal AbstractView _view;
                        internal Collider _collider;
                        /**********************************************************
                         * for override
                         **********************************************************/
                        /// <summary>
                        /// Run by the view during it's setup phase.
                        /// </summary>
                        public virtual void Setup(AbstractView view)
                        {
                            _view = view;
                            _inputManager = GameObject.FindObjectOfType(typeof(InputManag
```

An insight into the Unity View Framework.

Scaffolding User Manual.

Introduction

Scaffolding is a View framework tool for Unity. Scaffolding lets you manage and structure your games flow and menu system within the editor, allowing for very quick and easy construction.

Full source code is included so you can build on top of Scaffolding and make what ever modifications you require to suit your needs.

Scaffolding is made by Jon Reid, with the support of Preloaded.

For more guides, and full source code documentation, check out the website:

www.scaffoldingunity.com

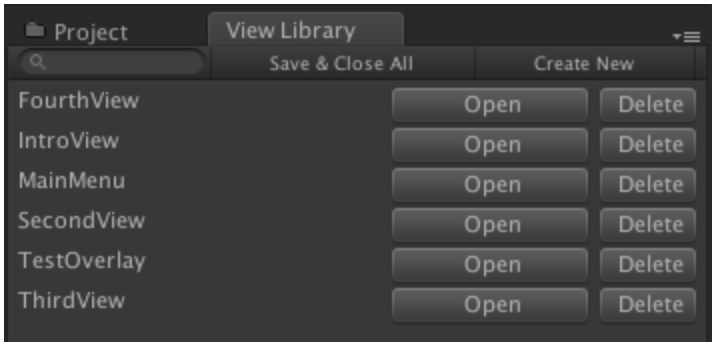
Contents

Initial Setup

First of all, import the Scaffolding package into Unity. The only needed folder for your project is the Scaffolding folder, everything else is part of the included demo. So if you want to use Scaffolding in an existing project, just copy in the Scaffolding folder.

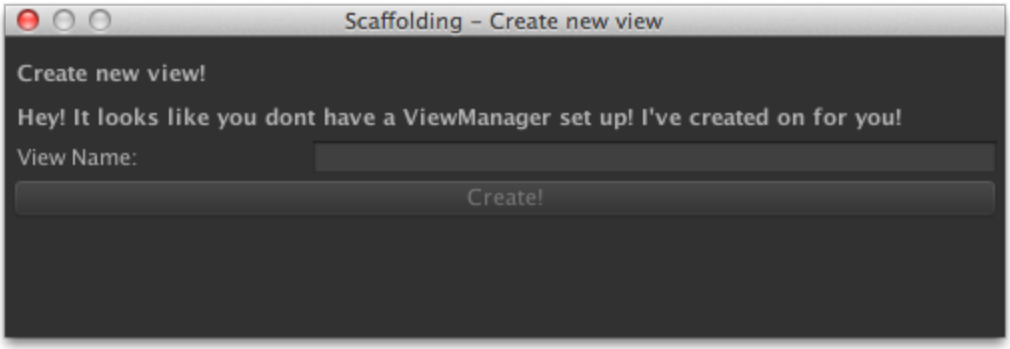
Now that you have Scaffolding in your project, lets open up the most central part of Scaffolding, the View Library. The View Library is where Scaffolding organises all the screens you have created, and where you create new ones from.

Go to Tools > Scaffolding > Open View Library. This will open up the View Library window, which you can dock anywhere within Unity. Keeping it somewhere within easy access is usually pretty handy.



Creating Views

Scaffolding lets you create Views very quickly. With your View Library open, click on Create View. This opens up the Create New View dialogue window, where you simply type in the name



of the view you want to create, for example “MainMenu” and then click on “Create!”.
This will put together a new c# script for your View, attach it to a new GameObject and store it all as a prefab. These are put, by default into Scripts/Views and Resources/Views respectively.

However, these directories can be changed by going to Tools > Scaffolding > Preferences.

Once a View has been created successfully, you’ll see it appear in your View Library, where you can Open or Delete the View, and while its open, you’ll have the option to close, which just removes it from the scene, Save, which applies all the changes you have made to the prefab, and Save and Close, which applies the changes, and then removes it from the scene.

You cannot have two views with the same name! Give each View a unique name!

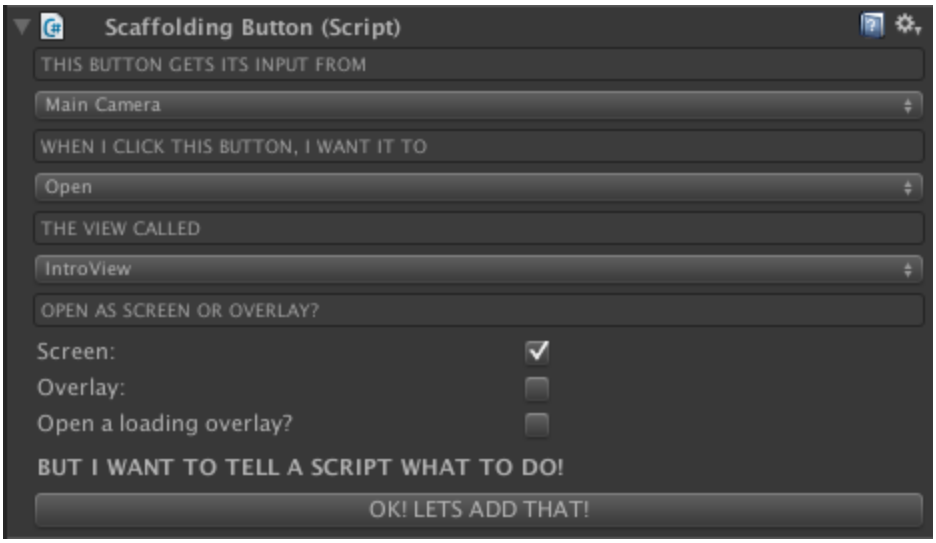
Linking between screens

Now that you know how to create Views, you can link them together using Buttons. For a simple example, I'll guide you through how to have two Screens linked together by a button press, so that when you click on the button during run time, it'll close the current screen and open the Screen of your choice.

First of all, open a View you want to add a button to. I'll open up MainMenu. Now to add a button. Select the View in the hierarchy, and then click on GameObject > Create new > Scaffolding > Button. This will create a simple button with a few basic components allowing you to quickly get things going.

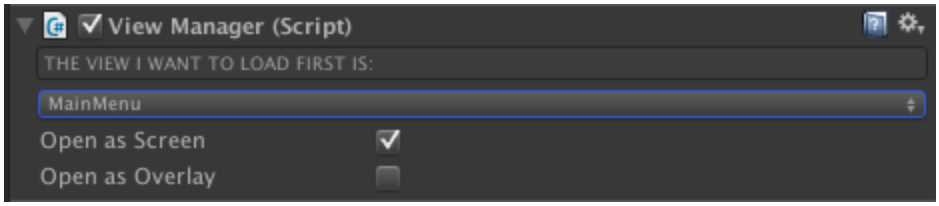
Buttons inside views need to be named differently! This allows you to get it via script later on!

Click on the newly create Button and take a look at the ScaffoldingButton script already attached. You'll be able to see that theres a few options available. You can set how you want the button to respond when you click it, to either Open a View, Close a View, Play an animation or do nothing. For this example, I'll choose Open.



Now that I've selected a way for the button to behave, a whole new set of options becomes available. I can now select what View I want to open, and how I want it to opened. Views can be opened either as a screen, or as an overlay. A screen is a standalone View and you can not have more than one screen opened at a time. These are perfect for menus. Overlays, however, can be opened and stacked on top of other Views. There is no limit on how many overlays you have open at once.

For now, I'll open the next View as a Screen.



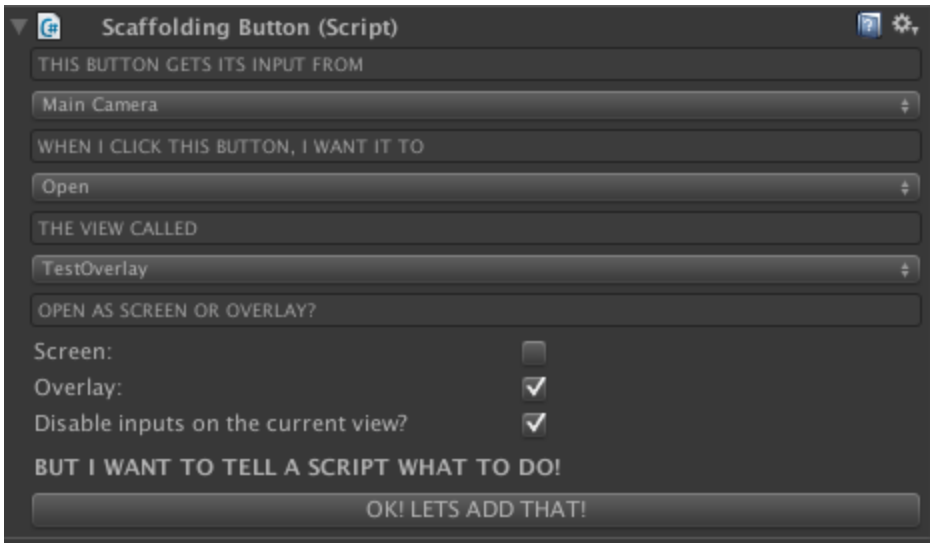
Now that I have set it to open another View as a screen, I'll press save and close and move on to the next step, making sure you have the right View set to open first.

Click on ViewManager in the scene. The ViewManager is what Scaffolding uses to control all the views during run time. From here, you can set which View opens first and if its an overlay or screen. I'll make sure that MainMenu is set to open first.

Now when I run the game, MainMenu opens up and I can click on the button which closes MainMenu and opens up the second View.

Overlays

Overlays are much like screens. Infact, underneath they are exactly the same. Your c# file for Screens and Overlays are identical. The only difference is the way they are opened. Going back to the button in the previous section, you can set the button to open a View as an overlay. Just tick that box and the screen will bo opened on top of the current screen. You get an extra option with overlays too. You have the option to disable all the inputs on the current screen, this allows you to open an overlay and all the buttons on the screen become unusable while the overlay is open.



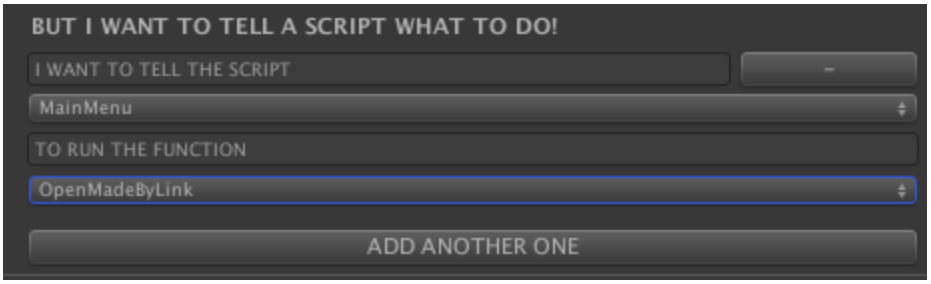
In the demo project bundled with Scaffolding, there is an overlay prefab that you can drop into a View so you can see how it is setup.

As mentioned previously, buttons have the option to close views as well. This option is primarily for Overlays. So you could have a button in an overlay that would close itself.

Making buttons trigger functions

Buttons can do more than just opening or closing other Views, they can trigger functions on any script currently in the scene too!

When you have clicked on a button and are viewing its inspector, you can see that there is a button that lets you tell a script what to do. Clicking this gives you the option to pick from a script in your project, and a public function on that script. With these options selected, when the button is pressed it will attempt to run that function.



There are however a few limitations:

- The target script MUST be in the scene on an active gameobject at the tiime the button is expected to be pressed.
- The function MUST be public.
- The function MUST NOT have any parameters.

Multitouch

Scaffolding has multi touch built right into its core. While accessing any inputs derived from Scaffoldings AbstractInput.cs, you'll be accessing Scaffoldings InputManager. AbstractInput makes it simple to create your own multi touch inputs or gestures, and is the base of all the buttons and gestures that come bundled with Scaffolding.

This makes Scaffolding great for tablets and mobile, having all your menu system respond to multi-touch, and not have to worry about what finger is pressing the button is a huge timesaver. This is also especially great when making applications for kids, as they are more likely to have their hand resting on the screen while pressing buttons, as explained in this sesame street research!

[It's right here if you want to read the entire usability report]

Gestures

Scaffolding has a few select gestures bundled with it in v1.0.

They are built as components, which make them simple to get up and running as you just attach the script to the GameObject you want to be receptive to the particular gesture.

The prebuilt gestures are:

- PinchThisInput.cs
- RotateThisInput.cs
- DragThisInput.cs

These will act on the GameObject they are attached to. For example, PinchThisInput will scale the GameObject it is attached to by the delta of the gesture.

These inputs require a collider to be on the object as well, so they are great for testing out ideas, but you may want to build more actions with these gestures as your project progresses.

If thats the case, then excellent news! All of the above premade gesture responses have a base class that makes it easy to deal with that particular gesture. Just inherit from them and override the necessary functions and you can build whatever response to those particular gestures you need.

Sending data between Views

Scaffoldings functionality doesn't stop at the editor, there are a whole host of tidbits that make coding with it quite powerful.

When moving between Views, its quite common to want to send data to a View, such as a score from the game View, or a name from a form.

You can do that with just a few lines of code.

When you want to send data to a View, you just have to package up the data in a handy SObject.cs which lets you place data in it easily and pull it out just by a name.

Example:

```
SObject data = new SObject();
data.AddInt(“Score”, 100);
```

And from there, you just need to tell which View you want to associate that data with.

Example:

```
SendDataToView(typeof(SecondView), data);
```

Scaffolding uses strong typing for referencing all its views. (typeof(VIEWNAME)) This uses less memory than linking prefabs! Less memory for Scaffolding, more memory for you!

This tells scaffolding that when you are next moving to the View called SecondView, it will pass through this object with it.

Reclaiming this data is done through OnShowStart on the target View. So on SecondView.cs, in `OnShowStart(SObject data){}` you can access the data object and do:
`data.GetInt("Score");`
which will return the value passed through if there was one sent through from the previous View.

And that concludes a brief introduction to Scaffolding and what it can do. Theres plenty more beneath the surface, so check out the API documentation to see what is available to you in code.