Built for Unity5

by **Darren O'Neale**

Thank you for your purchase!

You have taken a leap of faith purchasing this package, and I understand the concerns you may have after spending money on something you have not tried. I deeply appreciate your trust, so let me show my gratitude by giving you an in depth look at what you just purchased. This document will fill you in on all of the details of the window management system and get you managing every window of your game the *right* way once and for all. This package is designed to be as generic as possible - meaning you can use it and reuse it on all of your future projects - and you will be included in all future updates! Your money is beginning to feel pretty well spent by now, isn't it?

Menu Maker

Table of Contents

What You Just Bought

Hopefully you are glancing over this document before you use the product. You should already know what is included in this package based on the description in the store, but I would like to take a moment to re-iterate. This is a window management tool that can be used in all of your future unity games. The generic nature of this product will make it easy for you to create, customize, and publish your future games at a faster pace. In a matter of a couple hours, you can have a complete, stylized menu that your consumers will love.

The tool includes a series of scripts and compliment editors that will make it easy for you to make meaningful changes to your menus in faster iterations. The scripts included are expanded upon in detail below. Essentially, you now own a copy of a state manager, which is convenient for managing the plethora of windows in your game. You also received a window manager that holds prefab references to the windows in your overall system. The window manager allows you to add, remove, and customize existing windows. The customization is broken down into sounds or music that play when a window enters the view and in/out animations. There are several other code files, but the two mentioned above are the most important.

On top of the code you obtained, you also received a collection of window templates and pre-built animators. The window templates are some common styles that can be edited to fit the precise style of your game. The animators are based on an in-out animation system - where the 'in' animation describes how your windows enter the screen when they are called upon, and the 'out' animation describes how your windows will leave the screen when the user is done with them.

Continue reading below for more explanations of the package - and helpful tips for creating and customizing your game menus.

Helpful Notes

Before you jump into the package, I strongly recommend reviewing over this entire PDF. The reason is because you will undoubtedly run into some confusion along the way and there is a good chance the content of this document can answer your questions. While I am happy to help you on a personal level - this document was created as a first-level reference to assist you on your Menu Maker journey.

Just some notes if you are looking for a quick start. The main component of this package is the *WindowManager* prefab located inside the prefabs folder. Study this prefab in the example scene to understand how this system works.

❖

Important Items & Locations

Window Manager Components

Assets -> WindowStateManagement -> WindowManager
It is recommended you do not manipulate any of the code in these scripts.

Menu Pages

Assets -> Resources -> Demo1Pages
Assets -> Resources -> Demo2Pages
These are the pages that exist in the current menu systems starting from the two main menu scenes.

Page Templates

Assets -> WindowStateManagement -> PageTemplates
These template pages can be dragged onto the WindowManager's canvas to be previewed. These templates can be modified and become your custom page prefabs.

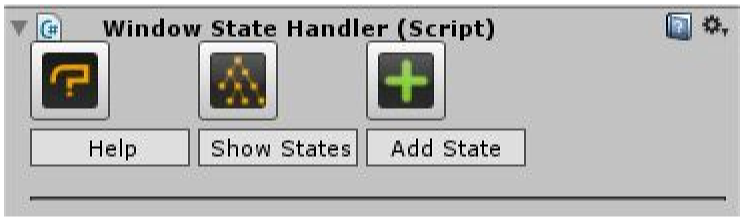
Page Animations & Animators

Assets -> WindowStateManagement -> PageAnimations
Assets -> WindowStateManagement -> PageAnimations -> Controllers
Information on how to use custom animators/animations below.

Menu Maker
Window State Handler

❖

How it Works



The state handler is simply comprised by a list of possible window states. These states are added or removed by the user by pressing the appropriate buttons in the inspector tool. Each window state has a list of actions that may be taken when the window is created. One of the actions is always going to be *Display Page* which will pull the window from the resources folder. Other actions could be things like *Load Player Data* or *Load Scene* just to list a couple.

In many ways, the window manager acts as a game moderator. When a window is created, we will want some other code to execute via events. Using the *Load Player Data* example, let's say we want to start playing the game by pressing a button on our menu. This button will be responsible for setting the state of the window manager to *Game*. The *Game* state will have two actions - *Load Player Data* - which pulls information from a text file about the player, and loads the correct player into the game, and *Display Player HUD* - which is responsible for pulling the window from the resources folder and adding it to the game view. This is just an example of how you can use the window manager tool to act as more than just a UI controller.

❖

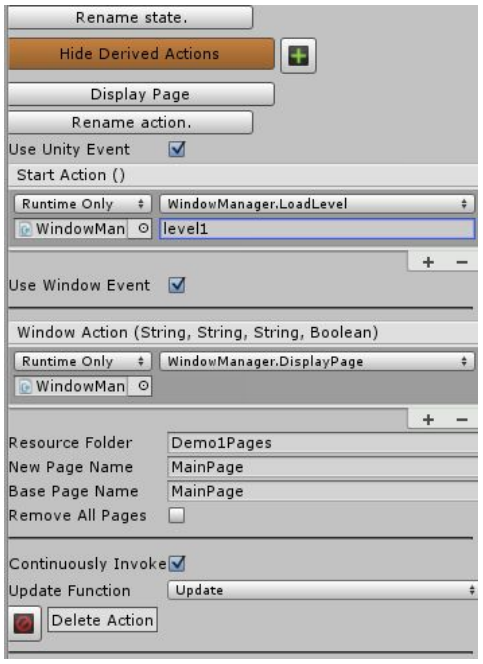
Displaying Menu States

Window states are outlined in the *WindowState* script inspector. You can press the *Show Window States* button to see the existing states in your manager. Continue clicking through the states and their actions to examine all of the logic that is the state manager.

❖

Interpreting States

There is a lot going on with the *WindowState* inspector. In the section, I will work to break everything down for you into understandable chunks. When you expand one of the states in your window state manager, you will see a button for *Actions.* Click this to drop down all of the actions that belong to this state. Now click on one of the actions and you will see a variety of settings. We will talk about these settings in the subsections below.



Use Unity Event ✓

Start Action ()

Runtime Only | WindowManager.LoadLevel

WindowMan | level1

Use Window Event ✓

Window Action (String, String, String, Boolean)

Runtime Only | WindowManager.DisplayPage

WindowMan

Resource Folder	Demo1Pages
New Page Name	MainPage
Base Page Name	MainPage
Remove All Pages	☐
Continuously Invoke	✓
Update Function	Update

Unity Events

You can choose to use a unity event by clicking the *UseUnityEvent* checkbox. When clicking this, you will see a unity event field appear. To add a unity event, you can click the '+' sign. A field will appear asking for a *GameObject* reference. This is going to be the game object that owns a method that will be invoked when this particular window enters the game view. For instance, for an action *LoadPlayerData*, we would use a unity event. The *GameObject* field would be a game object that has a script on it that handles player data, *PlayerDataHandler*. So we would drag that object in, then through the event field, we would search for the method that loads player data. At this point, when this particular window enters the view, player data will be loaded through the *PlayerDataHandler*.

Window Events

The next option you have for each action is to use a window event. A window event is simply a custom unity event. It was created to accept specific parameters that matter when we display a new page to the game view. If you click the *UseWindowEvent* checkbox, you will see a set of parameter fields appear - which will be parameters for the appropriate method. The question is where that method is located. Each window will have an action called *DisplayPage* that is responsible for loading in the next page. The page cannot be loaded in any other way. So when looking at the window event, you will drag the *WindowManager* object in the *GameObject* field for the window event. From the event settings, you will search for the *WindowManager* script and choose the *DisplayPage* method. There are repetitive examples of this in the example project if you are getting confused at all.

Once we have the correct method getting invoked by our window event, we need to fill in the parameters below the window event field. The first parameter is a string for the name of the folder the page prefab is located in. This folder must be a subfolder of 'Resources'. The second parameter is the name of the page prefab you are loading in. The third parameter is the name of the base page. The base page is the page that this window will be loaded on top of. In other words, the base page is a window that would already exist before this page gets loaded in. If there is no base page, you can leave the third parameter blank. You can fill in the same page for the new page and base page fields. This will ensure that when this page is loaded, it will not be destroyed and respawned.

Other Parameters

Even more control is granted to you through the next parameter, which determines whether or not a particular action will be updated over time. If you choose to invoke an event over every frame until this window is destroyed, select *ContinuouslyInvoke*. You will see an option appear that allows you to decide how you want your action to be updated. You can choose to update your action in *Updated, LateUpdate, or FixedUpdate*.

❖

Adding Custom States

Adding or removing states is as easy as clicking a button. You will see a button that says *Add* or *Remove* depending on what you want to do. When you add a new page or action, you will have to name the state or action. The name you choose for the state is important and needs to be unique. Whichever name you choose for a particular state is going to be the string value for what is passed to the handler's *SetState(string state)* method when trying to transition to the state.

❖

Transitioning to Custom States

To understand how to set up a page transition, you need to know about a built-in component which I call a dynamic listener. Attach this script to a button that is responsible for spawning a new page. You might ask why not use Unity's button event to trigger a new page. The reason is because since your pages will be prefabs, the buttons on those pages will not be able to hold references to the *WindowManager* in the hierarchy which is what you will be manipulating. See below for more information on the dynamic listener.

Dynamic Listener

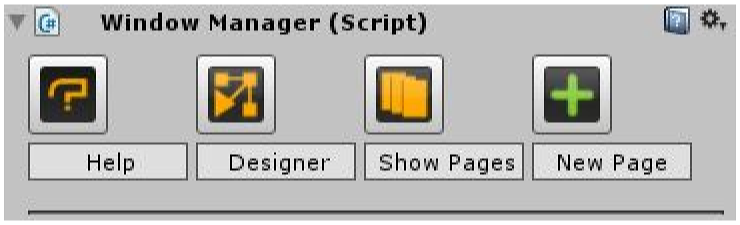
What the dynamic listener does is look up the object that contains the action we need by object tag. In other words, when we use the dynamic listener, we are going to look for the object with the tag *WindowManager*. Then, we are going to set the checkbox *Parameter* to true. The method name field is *SetState* because that is the name of the method we are accessing on the *WindowManager* object. The last field is the parameter that *SetState* accepts - which is the string value of the state this button should trigger. If I have a state called *Pause*, then I would type *Pause* into this field. When I press this button, the pause page will appear.

Menu Maker
Window Manager

❖
How it Works

The *WindowManager* script is attached to the WindowManager prefab. It contains runtime logic for switching to any page (from any page) in your menu system. The method within it, *DisplayPage()*, is what will be triggered anytime a page prefab loads in for the first time. The process of how to trigger new pages into the game view is discussed above in the *Displaying Menu States -> Window Event* section of this document.

The *WindowManager* also contains means for creating a list of menu page references for quick settings editing. You will see options such as displaying existing references and adding new references



❖
What's the Point?

The purpose of this tool is so you have a single space for changing two common settings on each of your pages (music and animation). Otherwise, you would have to find the page prefab in your folders, add or remove components, and tweak settings from there.

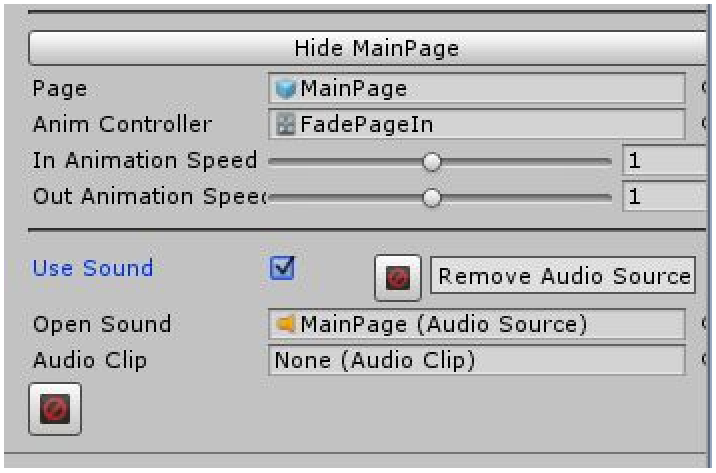
❖
Displaying Page References

To view all page references, click the *Show Pages* button at the top of the component.

❖

Interpreting Page References

There are a few things to note with the page reference settings. In short, we are given options to change animators and sound effects.


Hide MainPage

Page	MainPage
Anim Controller	FadePageIn
In Animation Speed	1
Out Animation Speec	1
Use Sound	✓
Open Sound	MainPage (Audio Source)
Audio Clip	None (Audio Clip)

Animator Settings

Changing a page's animator will modify the in-out transitions of that page when it is added or removed from the game view. Once a controller is found for the page, a *WindowAnimationController* component will be added to the page prefab. This component simply tells your page to animate when it enters the screen. You do not need to do anything else. From there, you can modify the *In* and *Out* animation speeds of the animator controller for the page. Note that if multiple pages use the same controller, these settings will translate to those page prefabs as well.

If you do not like any of the options for animator controllers in this package, you can create your own controllers based on existing page animations. So you can mix and match In/Out animations to your liking! For more information on how to create custom animators, read below in the *Animation Designer* section.

Sound Settings

By adding sound to a page, you can determine if it plays a sound or plays music while the page is alive. You could use this to drive the audio of your game, or you could use it to make certain pages have sound effects when loaded in.

When you click the *UseSound* checkbox, your page prefab is given an *AudioSource* component if it does not already have one. From there you can choose the audio clip you want to play when the page loads in. After adding the sound in the *AudioClip* field, you will be given an option for whether or not you want the sound to play once or on a loop.

If you click *Remove Audio Source*, the *AudioSource* component will be removed from your prefab, so you do not need to worry about manually removing it.

❖

Adding Page References

To add a new page reference, click the *Add New Page* button at the top of the *WindowManager* component.

❖

Adding Animations

Add pre-built animations from the collection of page animations in this package!

Using Existing Animations

The easiest way to do this is to look in the PageAnimations -> Controllers folder and select an animator controller.

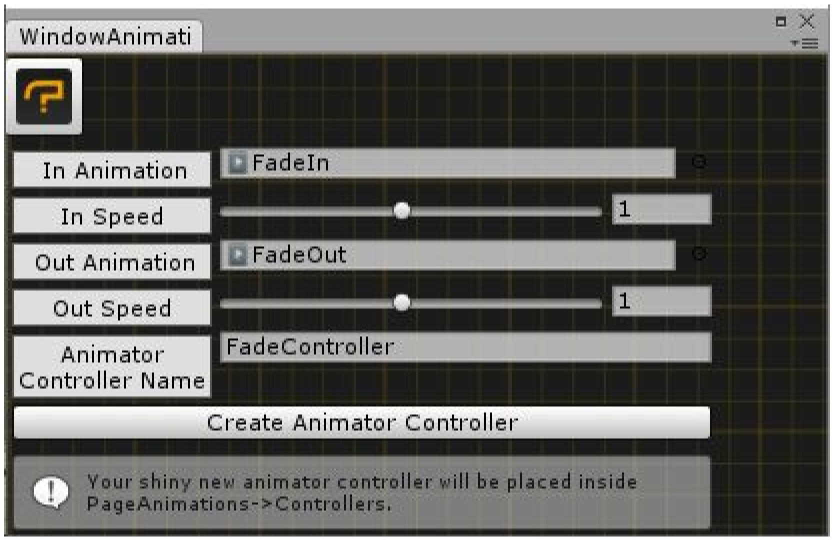
Using Custom Animations

To use custom animations, you will need to create a custom animator to deal with those animations. Instead of going through the trouble of creating an animator that is specific to this system, read below in the *Animation Designer* section for how to use the designer tool to create an animator that *is* specific to this system.

❖
How it Works

The Animation Designer is a tool used to create Animator Controllers based on one *In* animation and one *Out* animation. The controller is then ready for use inside your page references.

❖
Creating Custom Animators



Your shiny new animator controller will be placed inside PageAnimations->Controllers.

From the *PageAnimations* folder, you can drag an animation into both the *In* and *Out* fields. For each animation, you can determine the animation speed. After both of these fields are filled, you can name you controller something descriptive and click *Create Animator Controller*. Then, you can look for your controller in *PageAnimations->Controllers* and use it in your page references.

See More of My Assets

http://darrenoneale.me/
doneale3@gmail.com