CANVAS · FLOW
UI PRESENTATION & STORYBOARDING

For more extensive documentation, please visit https://canvasflow.xyz/.

2. Canvas Controllers

Contents

© 2018 Pelican 7 LTD. All rights reserved.

Introduction

A canvas controller represents a single screen in your user interface. Canvas controllers can present and dismiss other canvas controllers, or screens, allowing you to easily build a user interface flow, presenting and dismissing screens as required.

A canvas controller automatically manages the presentation hierarchy, as well as loading and unloading itself as it is presented or dismissed. This allows you to build large, complex, or dynamic user interface flows, loading only the necessary screens on demand.

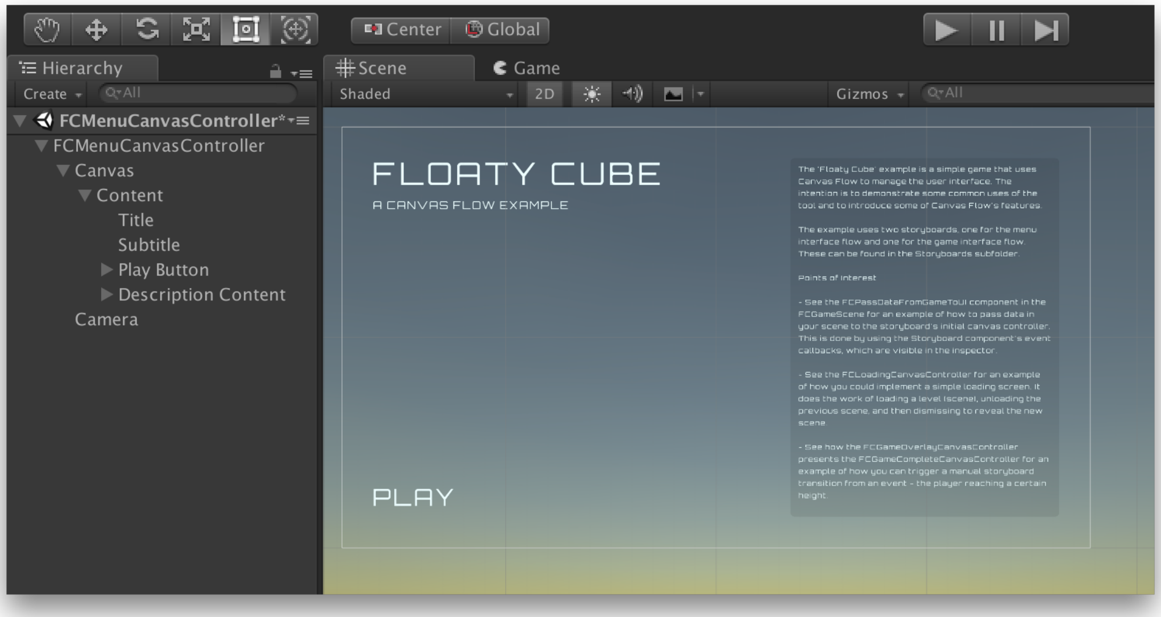
Canvas controllers are built with Unity's native UI Canvas, so any UI components that work with Unity's UI.Canvas are supported and creating the canvas controller's content will feel very familiar if you have worked with Unity UI before.

Canvas controllers can be used in Storyboards to visually create the user interface's flow.

Canvas controllers can be presented in world space by either setting a storyboard's StoryboardPresentationSpace to WorldSpace, or by passing a CanvasControllerWorldSpaceContainer to PresentInitialCanvasController(). Canvas controllers presented from a world space canvas controller will also be presented in world space and contained within the presenter's container.

A canvas controller consists of two files - a scene file (.unity) and a script file (.cs). The scene file is where you can create your screen's contents. See the ***Canvas Controller Scenes*** section for more information. The script file is where you can do any scripting that your screen may require. See the ***Presentation & Dismissal*** and ***Life Cycle*** sections for more information.

© 2018 Pelican 7 LTD. All rights reserved.



The 'Floaty Cube' example is a simple game that uses Canvas Flow to manage the user interface. The intention is to demonstrate some common uses of the tool and to introduce some of Canvas Flow's features.

The example uses two storyboards, one for the menu interface flow and one for the game interface flow. These can be found in the Storyboards subfolder.

Points of Interest

- See the FCPassDataFromGameToUI component in the FCGameScene for an example of how to pass data in your scene to the storyboard's initial canvas controller. This is done by using the Storyboard component's event callbacks, which are visible in the inspector.

- See the FCLoadingCanvasController for an example of how you could implement a simple loading screen. It does the work of loading a level (scene), unloading the previous scene, and then dismissing to reveal the new scene.

- See how the FCGameOverlayCanvasController presents the FCGameCompleteCanvasController for an example of how you can trigger a manual storyboard transition from an event - the player reaching a certain height.

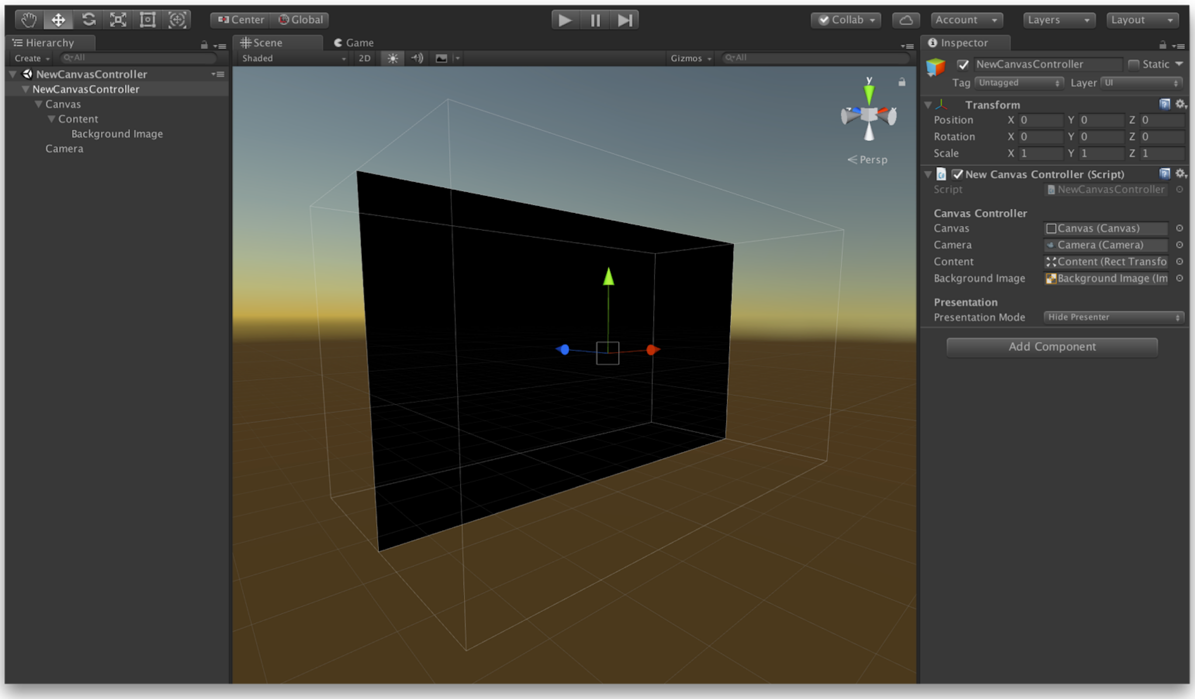
The menu canvas controller from the included 'Floaty Cube' example.

Creating A Canvas Controller

To create a new canvas controller, open Unity's *Create* menu - by either right-clicking in the Project window or selecting Assets in the menu bar - and selecting *Create/Canvas Flow/Canvas Controller*. Enter a name and a directory for your new canvas controller.

This will create two new files at the specified location - a *.unity* scene file and a *.cs* script file - and open the new canvas controller's scene.

© 2018 Pelican 7 LTD. All rights reserved.



NewCanvasController
- NewCanvasController
  - Canvas
    - Content
      - Background Image
  - Camera

NewCanvasController — Static
Tag: Untagged — Layer: UI

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

New Canvas Controller (Script)
Script: NewCanvasController

Canvas Controller

Canvas	Canvas (Canvas)
Camera	Camera (Camera)
Content	Content (Rect Transfo
Background Image	Background Image (Im

Presentation

Presentation Mode	Hide Presenter

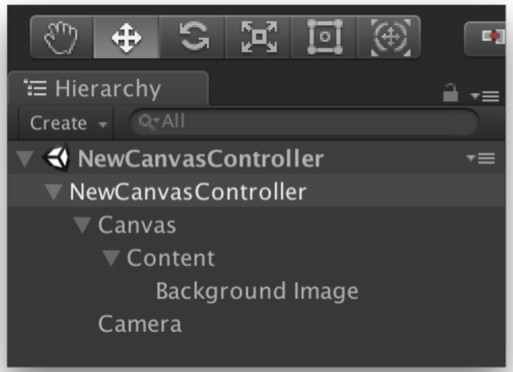
A newly created canvas controller.

© 2018 Pelican 7 LTD. All rights reserved.

CANVAS·FLOW
UI PRESENTATION & STORYBOARDING

Canvas Controller Scenes

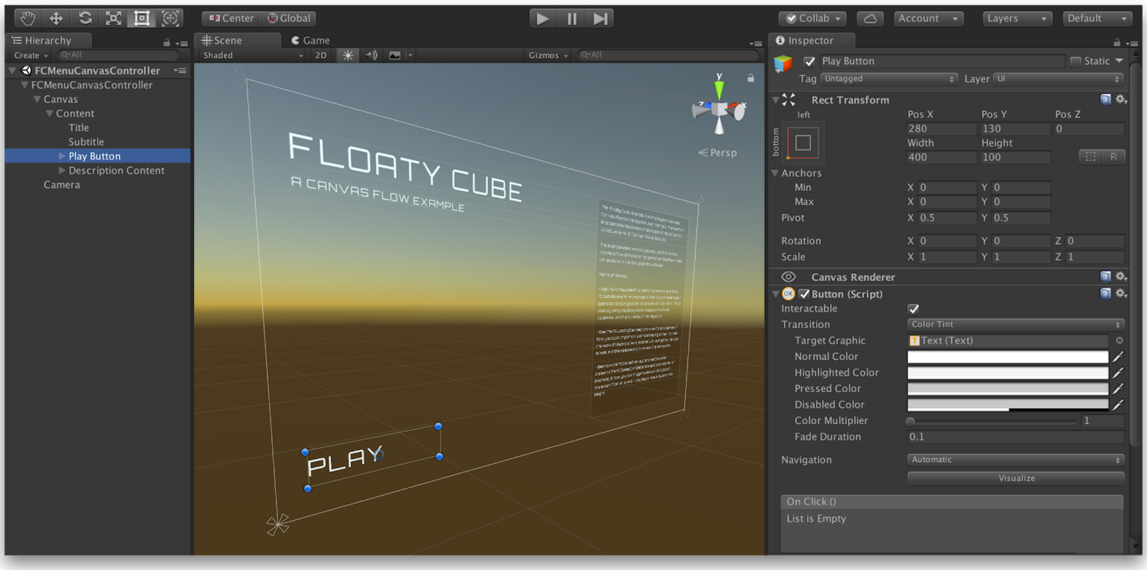
A canvas controller's scene file contains the main canvas and is where you can create your screen's contents. By default, a canvas controller's scene is structured like so:



```
NewCanvasController
  NewCanvasController
    Canvas
      Content
        Background Image
    Camera
```

A canvas controller's default hierarchy.

You should place your UI elements underneath the 'Content' game object, just as the canvas controller's 'Background Image' has been. This ensures that your canvas controller is compatible with the included transition animators[1].



```
FCMenuCanvasController
  FCMenuCanvasController
    Canvas
      Content
        Title
        Subtitle
        Play Button
        Description Content
    Camera
```

```
Play Button                         Static
Tag Untagged        Layer UI
Rect Transform
  left / bottom    Pos X 280    Pos Y 130    Pos Z 0
                   Width 400    Height 100
  Anchors
    Min            X 0      Y 0
    Max            X 0      Y 0
  Pivot            X 0.5    Y 0.5
  Rotation         X 0      Y 0      Z 0
  Scale            X 1      Y 1      Z 1
Canvas Renderer
Button (Script)
  Interactable       ✓
  Transition         Color Tint
    Target Graphic   Text (Text)
    Normal Color
    Highlighted Color
    Pressed Color
    Disabled Color
    Color Multiplier 1
    Fade Duration    0.1
  Navigation         Automatic
                     Visualize
  On Click ()
  List is Empty
```

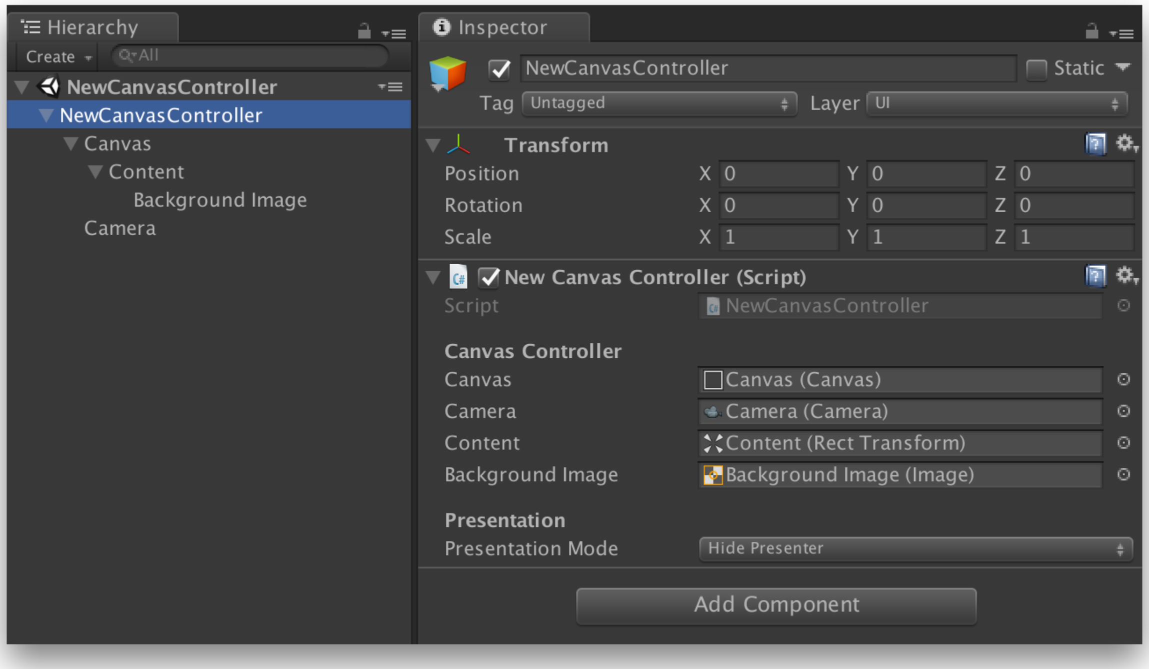
Creating a canvas controller's content.

[1] The included transition animators animate the 'Content' game object. If you place elements outside of the canvas controller's content, it won't be animated by the included transition animators. You could, however, animate it with a custom transition animator.

© 2018 Pelican 7 LTD. All rights reserved.

Root Canvas Controller Object

The top-level game object contains a single component - your canvas controller's script. Here you can see any properties that you declare in your canvas controller and configure any UI elements, such as button callbacks. **You must always keep this object in the root of your canvas controller's scene.**



NewCanvasController — Static
Tag: Untagged — Layer: UI

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

New Canvas Controller (Script)
Script: NewCanvasController

Canvas Controller
Canvas: Canvas (Canvas)
Camera: Camera (Camera)
Content: Content (Rect Transform)
Background Image: Background Image (Image)

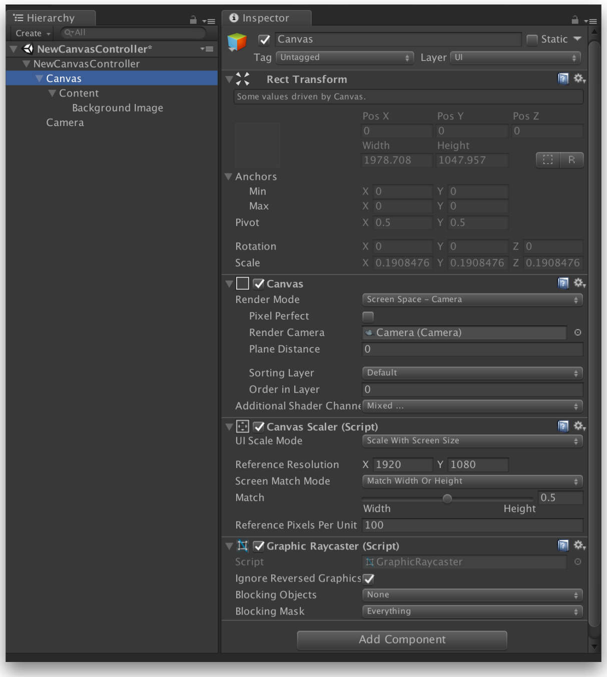
Presentation
Presentation Mode: Hide Presenter

The canvas controller game object and component.

Canvas Object

The canvas game object contains three components - the canvas itself, a Canvas Scaler, and the Canvas' Graphic Raycaster.

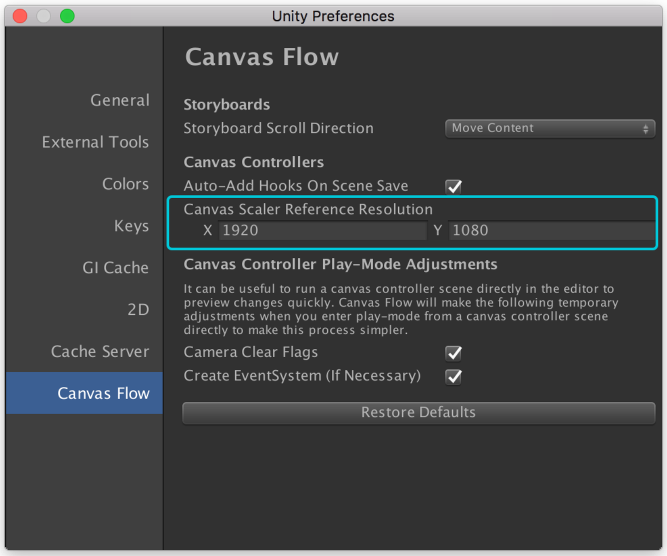
© 2018 Pelican 7 LTD. All rights reserved.



```
Hierarchy
NewCanvasController*
  NewCanvasController
    Canvas
      Content
        Background Image
    Camera

Inspector
Canvas                          Static
Tag  Untagged                   Layer  UI

Rect Transform
Some values driven by Canvas.
                      Pos X        Pos Y        Pos Z
                      0            0            0
                      Width        Height
                      1978.708     1047.957
Anchors
  Min                 X 0          Y 0
  Max                 X 0          Y 0
Pivot                 X 0.5        Y 0.5

Rotation              X 0          Y 0          Z 0
Scale                 X 0.1908476  Y 0.1908476  Z 0.1908476

Canvas
Render Mode           Screen Space - Camera
  Pixel Perfect       ☐
  Render Camera       Camera (Camera)
  Plane Distance      0

  Sorting Layer       Default
  Order in Layer      0
Additional Shader Channe  Mixed ...

Canvas Scaler (Script)
UI Scale Mode         Scale With Screen Size

Reference Resolution  X 1920       Y 1080
Screen Match Mode     Match Width Or Height
Match                 Width             Height   0.5
Reference Pixels Per Unit  100

Graphic Raycaster (Script)
Script                GraphicRaycaster
Ignore Reversed Graphics  ✓
Blocking Objects      None
Blocking Mask         Everything

Add Component
```

The canvas game object and component.

The Canvas Scaler's reference resolution should be set to the resolution that you are designing your UI at. If the screen resolution is larger, the UI will be scaled up, and if it's smaller, the UI will be scaled down. Note that you can change the default value for new canvas controllers in Canvas Flow's preferences, which can be found in the Unity menu at *Unity/Preferences/Canvas Flow*.



```
Unity Preferences

General            Canvas Flow
External Tools
Colors             Storyboards
Keys               Storyboard Scroll Direction    Move Content
GI Cache
2D                 Canvas Controllers
Cache Server       Auto-Add Hooks On Scene Save   ✓
Canvas Flow        Canvas Scaler Reference Resolution
                     X 1920                       Y 1080

                   Canvas Controller Play-Mode Adjustments
                   It can be useful to run a canvas controller scene directly in the editor to
                   preview changes quickly. Canvas Flow will make the following temporary
                   adjustments when you enter play-mode from a canvas controller scene
                   directly to make this process simpler.
                   Camera Clear Flags             ✓
                   Create EventSystem (If Necessary)  ✓

                   Restore Defaults
```

Set the reference resolution for new canvas controllers in the editor preferences window.

© 2018 Pelican 7 LTD. All rights reserved.

Presentation & Dismissal

Canvas controllers can present and dismiss other canvas controllers, allowing you to easily build a user interface flow, presenting and dismissing screens as required.

A canvas controller automatically manages the presentation hierarchy, as well as loading and unloading itself as it is presented or dismissed. This allows you to build large, complex, or dynamic user interface flows, loading only the necessary screens on demand.

Note: Use storyboards to present and dismiss canvas controllers without scripting by visually creating your user interface's flow. Please see the ***Storyboards*** section of the manual for information about using Storyboards.

Presenting From A Canvas Controller

Canvas controllers are presented and dismissed using the `PresentCanvasController<T>()` and `DismissCanvasController()` methods. For example, suppose we have two canvas controllers - `MenuCanvasController` and `SettingsCanvasController`. In our menu canvas controller is a UI Button that should show the settings screen when pressed.

In the *MenuCanvasController.cs* script, we could add the following *OnClick* handler:

```
public void OnSettingsButtonPressed()
{
    PresentCanvasController<SettingsCanvasController>();
}
```

And that's it. When this button is pressed, the menu will present the settings screen, dynamically loading the settings canvas controller and animating it on screen.

We could do a similar thing in our settings canvas controller, except this time the button being pressed is a back button and we want to return to the previous screen when it is pressed.

```
public void OnBackButtonPressed()
{
    DismissCanvasController();
}
```

© 2018 Pelican 7 LTD. All rights reserved.

This will cause the settings screen to be animated off-screen and subsequently unloaded, returning to the menu.

Presenting From A Scene

In a game environment, there is usually an existing scene - say the game's level or other 3D content scene - over which a user interface is presented. Once the initial canvas controller has been presented, the `PresentCanvasController<T>()` and `DismissCanvasController()` methods described above can be used. In order to present the initial canvas controller however, the static method `CanvasController.PresentInitialCanvasController<T>()` is used.

So, for example, say we have an existing game scene called `Level1`, which contains our game's content, and a canvas controller called `GameOverlayCanvasController`. When the game scene `Level1` is run, we wish to present the game overlay screen over the game's content.

To do this, we could create a new script in the `Level1` game scene and present the initial `GameOverlayCanvasController` in the `MonoBehaviour's Awake()` method, like so:

```
private void Awake()
{

CanvasController.PresentInitialCanvasController<GameOverlayCanvasCont
roller>(animated: false)
}
```

Note how we can specify this presentation to not be animated by using the animated parameter.

The game overlay screen could then use the `PresentCanvasController<T>()` and `DismissCanvasController()` methods described above, perhaps presenting pause or game-over screens, as required.

Note: A single Unity EventSystem is required to be present for canvases to receive input events. Therefore, depending on your game's structure, you may wish to place an Event System (*Game Object/UI/Event System*) in the scene presenting your initial canvas controller.

Presenting In World Space

By default, canvas controllers are presented in screen space. To present a canvas controller in world space, you specify a CanvasControllerWorldSpaceContainer component when calling the

© 2018 Pelican 7 LTD. All rights reserved.

aforementioned `CanvasController.PresentInitialCanvasController<T>()` method.

First, create a new GameObject in the scene and add a `CanvasControllerWorldSpaceContainer` component to it. This will give you a world space canvas that you can position freely in your scene.

Then, when calling `CanvasController.PresentInitialCanvasController<T>()`, pass in the world space container like so:

```
CanvasController.PresentInitialCanvasController<GameOverlayCanvasCont
roller>(worldSpaceContainer: yourWorldSpaceContainer)
```

This will cause the canvas controller to be presented in world space and embedded within the specified container.

Note that any subsequent calls to `PresentCanvasController<T>()` will automatically detect whether they should be presented in world space or screen space. If the presenting canvas controller is in world space, the presented canvas controller will also be in world space. Likewise, if the presenting canvas controller is in screen space, the presented canvas controller will also be in screen space.

Configuring A (To-Be) Presented Canvas Controller

All `Present…()` methods include a configuration action parameter to allow configuring the loaded canvas controller before it is presented. The configuration action is invoked by Canvas Flow after the canvas controller has been loaded, but before it has begun presenting. The configuration action is specified like so:

```
PresentCanvasController<GameOverlayCanvasController>(configuration:
(gameOverlayCanvasController) =>
{
    // Configure the newly-loaded gameOverlayCanvasController.
});
```

A common use-case for the configuration action is to pass data to the to-be-presented canvas controller. To illustrate, take the earlier example of presenting a game overlay screen in the game's `Level1` scene. Say we now have `Level2` and `Level3` scenes as well, which also display the same game overlay screen as before. However, we want to display the current level number in the top corner of the screen for the player. After adding a label to display the current

© 2018 Pelican 7 LTD. All rights reserved.

level, we could also add a `ConfigureWithLevel()` method to our `gameOverlayCanvasController`, which sets the level label's text.

```
// In GameOverlayCanvasController.cs
public void ConfigureWithLevel(int levelNumber)
{
    currentLevelLabel.text = string.Format("Level {0}", levelNumber);
}
```

Then when presenting the game overlay screen from the `Awake()` method as before, we can use the configuration action to configure it with the level number, which is set in the scene.

```
public int levelNumber;

private void Awake()
{

CanvasController.PresentInitialCanvasController<GameOverlayCanvasCont
roller>(
        configuration: (gameOverlayCanvasController) =>
    {
        gameOverlayCanvasController.ConfigureWithLevel(levelNumber);
    } animated: false);
}
```

Now, no matter how many levels we create, the game overlay screen will simply display the level number it is configured with.

Note: Canvas controllers also have a similar `completion:` action parameter, which is invoked after the presentation transition has completed.

For information on customizing the transition animation between canvas controllers, please see the Custom Transition Animators section of the online manual at https://canvasflow.xyz/.

© 2018 Pelican 7 LTD. All rights reserved.

Canvas Controller Life Cycle

Canvas controllers allow you to override certain methods in order to be notified when particular events occur in their life cycle. These include appearance related events, as well as all `MonoBehaviour` events.

Appearance

A canvas controller can override the following methods to receive appearance related callbacks. These could be used, for example, to trigger an animation once a canvas controller has completely transitioned on screen.

You can use the `IsBeingPresented` and `IsBeingDismissed` methods within the callback to infer the cause of the appearance or disappearance.

```
protected override void CanvasWillAppear()
{
    /*
     *  Override this method in your canvas controller to be notified
just
     *  before the canvas is about to transition on screen. This
could be
     *  because it is about to be presented or about to dismiss its
     *  PresentedCanvasController.
     */
}

protected override void CanvasDidAppear()
{
    /*
     *  Override this method in your canvas controller to be notified
just
     *  after the canvas has transitioned on screen. This could be
because
     *  it has just been presented or has just dismissed its
     *  PresentedCanvasController.
     */
}

protected override void CanvasWillDisappear()
```

© 2018 Pelican 7 LTD. All rights reserved.

```
{
    /*
     *  Override this method in your canvas controller to be notified
just
     *  before the canvas is about to transition off screen. This
could be
     *  because it is about to be dismissed or about to present a
canvas
     *  controller.
     */
}

protected override void CanvasDidDisappear()
{
    /*
     *  Override this method in your canvas controller to be notified
just
     *  after the canvas has transitioned off screen. This could be
     *  because it has just been dismissed or has just presented a
canvas
     *  controller.
     */
}
```

MonoBehaviour

A canvas controller is a `MonoBehaviour` and as such, all `MonoBehaviour` callbacks will be called on a canvas controller.

If implementing `Awake()` or `Start()`, you should override and call the base implementation like so:

```
protected override void Awake()
{
    base.Awake();

    // Your awake functionality.
}

protected override void Start()
{
```

© 2018 Pelican 7 LTD. All rights reserved.

```
    base.Start();

    // Your start functionality.
}
```

© 2018 Pelican 7 LTD. All rights reserved.