Creating Custom Presentations

UIKit separates the content of your view controllers from the way that content is presented and displayed onscreen. Presented view controllers are managed by an underlying presentation controller object, which manages the visual style used to display the view controller's view. A presentation controller may do the following:

- Set the size of the presented view controller.
- Add custom views to change the visual appearance of the presented content.
- Supply transition animations for any of its custom views.
- Adapt the visual appearance of the presentation when changes occur in the app's environment.

UIKit provides presentation controllers for the standard presentation styles. When you set the presentation style of a view controller to `UIModalPresentationCustom` and provide an appropriate transitioning delegate, UIKit uses your custom presentation controller instead.

The Custom Presentation Process

When you present a view controller whose presentation style is `UIModalPresentationCustom`, UIKit looks for a custom presentation controller to manage the presentation process. As the presentation progresses, UIKit calls methods of the presentation controller, giving it opportunities to set up any custom views and animate them into position.

A presentation controller works alongside any animator objects to implement the overall transition. The animator objects animate the view controller's contents onto the screen and the presentation controller handles everything else. Normally, your presentation controller animates its own views, but you can also override the presentation controller's `presentedView` method and let the animator objects animate all or some of those views.

During a presentation, UIKit:

1. Calls the `presentationControllerForPresentedViewController:presentingViewController:sourceViewController:` method of the transitioning delegate to retrieve your custom presentation controller
2. Asks the transitioning delegate for the animator and interactive animator objects, if any
3. Calls your presentation controller's `presentationTransitionWillBegin` method

   Your implementation of this method should add any custom views to the view hierarchy and configure the animations for those views.
4. Gets the `presentedView` from your presentation controller

   The view returned by this method is animated into position by the animator objects. Normally, this method returns the root view of the presented view controller. Your presentation controller can replace that view with a custom background view, as needed. If you do specify a different view, you must embed the root view of the presented view controller into your view hierarchy.
5. Performs the transition animations

   The transition animations include the main ones created by the animator objects and any animations you configured to run alongside the main animations. For information on the transition animations, see The Transition Animation Sequence.

   During the animation process, UIKit calls the `containerViewWillLayoutSubviews` and `containerViewDidLayoutSubviews` methods of your presentation controller so that you can adjust the layout of your custom views as needed.
6. Calls the `presentationTransitionDidEnd:` method when the transition animations finish

During a dismissal, UIKit:

1. Gets your custom presentation controller from the currently visible view controller
2. Asks the transitioning delegate for the animator and interactive animator objects, if any
3. Calls your presentation controller's `dismissalTransitionWillBegin` method

   Your implementation of this method should add any custom views to the view hierarchy and configure the animations for those views .
4. Gets the already `presentedView` from your presentation controller
5. Performs the transition animations

The transition animations include the main ones created by the animator objects and any animations you configured to run alongside the main animations. For information on the transition animations, see The Transition Animation Sequence.

During the animation process, UIKit calls the `containerViewWillLayoutSubviews` and `containerViewDidLayoutSubviews` methods of your presentation controller so that you can remove any custom constraints.

6. Calls the `dismissalTransitionDidEnd:` method when the transition animations finish

During the presentation process, The `frameOfPresentedViewInContainerView` and `presentedView` methods of your presentation controller may be called several times, so your implementations should return quickly. Also, your implementation of your `presentedView` method should not try to set up the view hierarchy. The view hierarchy should already be configured by the time the method is called.

Creating a Custom Presentation Controller

To implement a custom presentation style, subclass `UIPresentationController` and add code to create the views and animations for your presentation. When creating a custom presentation controller, consider the following questions:

- What views do you want to add?
- How do you want to animate any additional views onscreen?
- What size should the presented view controller be?
- How should the presentation adapt between a horizontally regular and horizontally compact size class?
- Should the presenting view controller's view be removed when the presentation finishes?

All of these decisions require overriding different methods of the `UIPresentationController` class.

Setting the Frame of the Presented View Controller

You can modify the frame rectangle of the presented view controller so that it fills only part of the available space. By default, a presented view controller is sized to completely fill the container view's frame. To change the frame rectangle, override your presentation controller's `frameOfPresentedViewInContainerView` method. Listing 11-1 shows an example where the frame is changed to cover only the right half of the container view. In this case, the presentation controller uses a background dimming view to cover the other half of the container.

Listing 11-1 Changing the frame of a presented view controller

```
- (CGRect)frameOfPresentedViewInContainerView {
    CGRect presentedViewFrame = CGRectZero;
    CGRect containerBounds = [[self containerView] bounds];

    presentedViewFrame.size = CGSizeMake(floorf(containerBounds.size.width / 2.0),
                                         containerBounds.size.height);
    presentedViewFrame.origin.x = containerBounds.size.width -
                                  presentedViewFrame.size.width;
    return presentedViewFrame;
}
```

Managing and Animating Custom Views

Custom presentations often involve adding custom views to the presented content. Use custom views to implement purely visual adornments or use them to add practical behaviors to the presentation. For example, a background view might incorporate gesture recognizers to track specific actions outside the bounds of the presented content.

The presentation controller is responsible for creating and managing all custom views associated with its presentation. Usually, you create custom views during the initialization of your presentation controller. Listing 11-2 shows the initialization method of a custom view controller that creates its own dimming view. This method creates the view and performs some minimal configuration.

Listing 11-2 Initializing the presentation controller

```
- (instancetype)initWithPresentedViewController:(UIViewController
  *)presentedViewController
```

```
                    presentingViewController:(UIViewController
*)presentingViewController {
    self = [super initWithPresentedViewController:presentedViewController
                        presentingViewController:presentingViewController];
    if(self) {
        // Create the dimming view and set its initial appearance.
        self.dimmingView = [[UIView alloc] init];
        [self.dimmingView setBackgroundColor:[UIColor colorWithWhite:0.0
alpha:0.4]];
        [self.dimmingView setAlpha:0.0];
    }
    return self;
}
```

You animate your custom views onto the screen using the `presentationTransitionWillBegin` method. In this method, configure your custom views and add them to the container view, as shown in Listing 11-3. Use the transition coordinator of either the presented or presenting view controller to create any animations. Do not modify the presented view controller's view in this method. The animator objects are responsible for animating the presented view controller into the frame rectangle you return from your `frameOfPresentedViewInContainerView` method.

Listing 11-3 Animating the dimming view onto the screen

```
- (void)presentationTransitionWillBegin {
    // Get critical information about the presentation.
    UIView* containerView = [self containerView];
    UIViewController* presentedViewController = [self presentedViewController];

    // Set the dimming view to the size of the container's
    // bounds, and make it transparent initially.
    [[self dimmingView] setFrame:[containerView bounds]];
    [[self dimmingView] setAlpha:0.0];

    // Insert the dimming view below everything else.
    [containerView insertSubview:[self dimmingView] atIndex:0];

    // Set up the animations for fading in the dimming view.
    if([presentedViewController transitionCoordinator]) {
        [[presentedViewController transitionCoordinator]

animateAlongsideTransition:^(id<UIViewControllerTransitionCoordinatorContext>
                                            context) {
            // Fade in the dimming view.
            [[self dimmingView] setAlpha:1.0];
        } completion:nil];
    }
    else {
        [[self dimmingView] setAlpha:1.0];
    }
}
```

At the end of a presentation, use the `presentationTransitionDidEnd:` method to handle any cleanup caused by the cancelation of the presentation. An interactive animator object might cancel a transition if its threshold conditions are not met. When that happens, UIKit calls the `presentationTransitionDidEnd:` method with a value of `NO`. When a cancelation occurs, remove any custom views you added at the beginning of the presentation and return any other views to their previous configurations, as shown in Listing 11-4.

Listing 11-4 Handling a cancelled presentation

```
- (void)presentationTransitionDidEnd:(BOOL)completed {
    // If the presentation was canceled, remove the dimming view.
    if (!completed)
        [self.dimmingView removeFromSuperview];
}
```

When the view controller is dismissed, use the `dismissalTransitionDidEnd:` methods to remove your custom views from the view hierarchy. If you want to animate the disappearance of your views, set up those animations in the `dismissalTransitionDidEnd:` method. Listing 11-5 shows implementations of both methods for removing the dimming view in the previous examples. Always check the parameter of the `dismissalTransitionDidEnd:` method to see if the dismissal was successful or was canceled.

Listing 11-5 Dismissing the presentation's views

```
- (void)dismissalTransitionWillBegin {
    // Fade the dimming view back out.
    if([[self presentedViewController] transitionCoordinator]) {
        [[[self presentedViewController] transitionCoordinator]

  animateAlongsideTransition:^(id<UIViewControllerTransitionCoordinatorContext>
                                        context) {
            [[self dimmingView] setAlpha:0.0];
        } completion:nil];
    }
    else {
        [[self dimmingView] setAlpha:0.0];
    }
}

- (void)dismissalTransitionDidEnd:(BOOL)completed {
    // If the dismissal was successful, remove the dimming view.
    if (completed)
        [self.dimmingView removeFromSuperview];
}
```

On This Page

Vending Your Presentation Controller to UIKit

When presenting a view controller, do the following to display it using your custom presentation controller:

- Set the `modalPresentationStyle` property of the presented view controller to `UIModalPresentationCustom`.
- Assign a transitioning delegate to the `transitioningDelegate` property of the presented view controller.
- Implement the `presentationControllerForPresentedViewController:presentingViewController:sourceViewController:` method of the transitioning delegate.

UIKit calls the `presentationControllerForPresentedViewController:presentingViewController:sourceViewController:` method of your transitioning delegate when it needs your presentation controller. Your implementation of this method should be as simple as the one in Listing 11-6. Simply create your presentation controller, configure it, and return it. If you return `nil` from this method, UIKit presents the view controller using a full-screen presentation style.

Listing 11-6 Creating a custom presentation controller

```
- (UIPresentationController *)presentationControllerForPresentedViewController:
                                (UIViewController *)presented
        presentingViewController:(UIViewController *)presenting
            sourceViewController:(UIViewController *)source {

    MyPresentationController* myPresentation = [[MyPresentationController]
       initWithPresentedViewController:presented
  presentingViewController:presenting];

    return myPresentation;
}
```

Adapting to Different Size Classes

While a presentation is onscreen, UIKit notifies your presentation controller when changes occur to the underlying traits or to the size of the container view. Such changes typically occur during a device rotation but may occur at other times. You can use trait and size notifications to adjust your presentation's custom views and update your presentation style as appropriate.

For information about how to adapt to new traits and sizes, see Building an Adaptive Interface.

On This Page