Sample Code

Customizing an image picker controller

Manage user interactions and present custom information when taking pictures by adding an overlay view to your image picker.

Download

iOS 10.0+ | iPadOS 10.0+ | Xcode 11.0+

Overview

This sample applies an overlay view to display a custom view hierarchy on top of the default image picker interface.

The sample app uses an overlay view to:

- Create an interface to respond to user input.
- Display custom information (such as images to enhance the interface).
- Implement a number of unique camera functions such as single picture capture, timed picture capture, and repeated pictures like a camera with a fast shutter speed.

Configure the sample code project

Because the camera isn't available in Simulator, you'll need to build and run this sample on a device with iOS 10 or later installed.

When you first launch the sample app on device, you'll need to grant the app permission to use the camera.

Setup the overlay view

The sample app uses the `cameraOverlayView` property to provide an overlay view that contains the custom view hierarchy. The image picker places the custom overlay view on top of the other image picker views.

```
/*
Apply the overlay view. This view contains a toolbar with custom
controls for capturing still images in various ways.
*/
overlayView?.frame = (imagePickerController.cameraOverlayView?.frame)!
imagePickerController.cameraOverlayView = overlayView
```

An app can access the `cameraOverlayView` property only when the source type of the image picker is set to `UIImagePickerController.SourceType.camera`.

When the user interacts with interface elements in the custom view, the app calls an image picker method, such as `takePicture()` to capture a photo, and implement other features. This sample's custom image picker controller interface provides the following features:

- Take a Picture
- Take a Delayed Picture
- Take Repeated Pictures
- Browse Media in the Photo Library

The `showsCameraControls` property indicates whether the image picker displays the default camera controls. The `showsCameraControls` property is only accessible when the source type of the image picker is `UIImagePickerController.SourceType.camera`. This sample sets `showsCameraControls` to `false` to hide the default controls and provide a custom overlay view.

```
if sourceType == UIImagePickerController.SourceType.camera {
    /*
     The user tapped the camera button in the app's interface which
     specifies the device’s built-in camera as the source for the image
     picker controller.
    */

    /*
     Hide the default controls.
     This sample provides its own custom controls for still image
     capture in an overlay view.
    */
    imagePickerController.showsCameraControls = false

    /*
     Apply the overlay view. This view contains a toolbar with custom
     controls for capturing still images in various ways.
    */
    overlayView?.frame = (imagePickerController.cameraOverlayView?.frame)!
    imagePickerController.cameraOverlayView = overlayView
```

Take a picture

Take a picture with the Snap button. Its action method calls the takePicture() method to actually take a picture.

```
@IBAction func takePhoto(_ sender: UIBarButtonItem) {
imagePickerController.takePicture()
}
```

Take a delayed picture

Take a picture after a short delay with the Delayed button. Its action method calls the takePicture() method to take a picture when the timer expires.

```
@IBAction func delayedTakePhoto(_ sender: UIBarButtonItem) {
    /*
     Disable the photo controls during the delay time period.
     The code in the timer completion block below captures a still image
     when the delay period expires and re-enables the controls.
    */
    doneButton?.isEnabled = false
    takePictureButton?.isEnabled = false
    delayedPhotoButton?.isEnabled = false
    startStopButton?.isEnabled = false

    let fireDate = Date(timeIntervalSinceNow: 5)
    cameraTimer = Timer(fire: fireDate, interval: 1.0, repeats: false, block: { time
        // The time interval expired. Capture a still image.
        self.imagePickerController.takePicture()

        // Enable the delayed photos controls.
        self.doneButton?.isEnabled = true
        self.takePictureButton?.isEnabled = true
        self.delayedPhotoButton?.isEnabled = true
        self.startStopButton?.isEnabled = true
    })
    RunLoop.main.add(cameraTimer, forMode: RunLoop.Mode.default)
}
```

Take repeated pictures

Take repeated pictures at a certain interval with the Start button; for example, one photo every five seconds. Its action method creates a timer to take pictures at certain intervals using the `takePicture()` method.

This sample takes pictures indefinitely, causing it to run out of memory quickly. You must decide upon a proper threshold of the number of captured photos for your own app (for simplicity, this app does not enforce a limit). To avoid memory constraints, save each taken photo to disk rather than keeping all of the pictures in memory. The system may invoke your app's `didReceiveMemoryWarning()` method in low memory situations so the app can recover some memory and continue taking photos.

```
@IBAction func startTakingPicturesAtIntervals(_ sender: UIBarButtonItem) {
    // Start the timer to take a photo every 5 seconds.

    startStopButton?.title = NSLocalizedString("Stop", comment: "Title for overlay v
    startStopButton?.action = #selector(stopTakingPicturesAtIntervals)

    // Enable these buttons while capturing photos.
    doneButton?.isEnabled = false
    delayedPhotoButton?.isEnabled = false
    takePictureButton?.isEnabled = false

    // Start taking pictures.
    cameraTimer = Timer.scheduledTimer(withTimeInterval: 5, repeats: true) { timer i
        self.imagePickerController.takePicture()
    }
}
```

The camera starts taking pictures as soon as the user taps the Start button (which changes to a Stop button). The camera continues to capture photos until the user taps Stop. Captured images appear in the order taken within the app's image view.

```
@IBAction func stopTakingPicturesAtIntervals(_ sender: UIBarButtonItem) {
    // Stop and reset the timer.
    cameraTimer.invalidate()

    finishAndUpdate()

    // Make these buttons available again.
    self.doneButton?.isEnabled = true
    self.takePictureButton?.isEnabled = true
    self.delayedPhotoButton?.isEnabled = true

    // Reset the button to start taking pictures again.
    startStopButton?.title = NSLocalizedString("Start", comment: "Title for overlay
    startStopButton?.action = #selector(startTakingPicturesAtIntervals)
}
```

Browse media in the Photo Library

To browse images saved in the photo albums on the device, add a button the user can press to go to their Photo Library. The button's action method configures the picker for browsing saved media by setting its sourceType property to `UIImagePickerController.SourceType.photoLibrary`, before presenting the picker's media browser user interface.

```
@IBAction func showImagePickerForPhotoPicker(_ sender: UIBarButtonItem) {
showImagePicker(sourceType: UIImagePickerController.SourceType.photoLibrary, button:
}
```

Selecting a photo invokes the app's `imagePickerController(_:didFinishPickingMediaWithInfo:)` delegate method which saves the selected image to an array and displays it in the app's image view.

Tapping the Cancel button invokes the app's `imagePickerControllerDidCancel(_:)` delegate method which calls `dismiss(animated:completion:)` to dismiss the picker.

See Also

Customizing the camera controls

`var showsCameraControls: Bool`

A Boolean value that indicates whether the image picker displays the default camera controls.

```
var cameraOverlayView: UIView?
```

The view to display on top of the default image picker interface.

```
var cameraViewTransform: CGAffineTransform
```

The transform to apply to the camera’s preview image.