Project 21: Local Notifications with UILocalNotification

Overview

`Brief`: Send reminders, prompts and alerts even when your app isn't running.

`Learn`: UILocalNotification.

- Setting up
- Scheduling notifications: UILocalNotification
- Acting on responses
- Wrap up

Setting up

This is going to be the easiest technique project in the entire series, and I expect you're extremely relieved to hear that because it can be hard going always having to learn new things!

What you're going to learn about are local notifications, which let you send reminders to your user's lock screen to show them information when your app isn't running. If you set a reminder in your calendar, making it pop up on your lock screen at the right time is a local notification.

These aren't the same as push notifications, and in fact they are quite a different beast from a development perspective. I would love to cover push notifications here, but they require a dedicated server (or service, if you outsource) to send from and that's outside the remit of this course.

To get started, create a new `Single View Application` project in `Xcode`, name it `Project21`, and set it to target any device.

`Scheduling notifications: UILocalNotification`

Open `Main.storyboard` in `Interface Builder` and place two buttons, one above the other. The first should have the title `“Register Local”` and the second the title `“Schedule Local”`. Add whatever constraints you think sensible, but ideally make them centered horizontally so they fit any device. Using the assistant editor, create an action for each: `registerLocal()` and `scheduleLocal()`. Now go back to the standard editor and switch to `ViewController.swift`.

Let me explain how this project needs to work. First, you can't post messages to the user's lock screen unless you have their permission. This was changed in `iOS 8`, but it's quite sensible – it would, after all, be awfully annoying if any app could bother you when it pleased.

So, in order to send local notifications in our app, we first need to request permission, and that's what we'll put in the `registerLocal()` method. You register your settings based on what you actually need, and that's done with a class called `UIUserNotificationSettings`. For this example we're going to request an alert (a message to show), along with a badge (for our icon) and a sound (because users just love those).

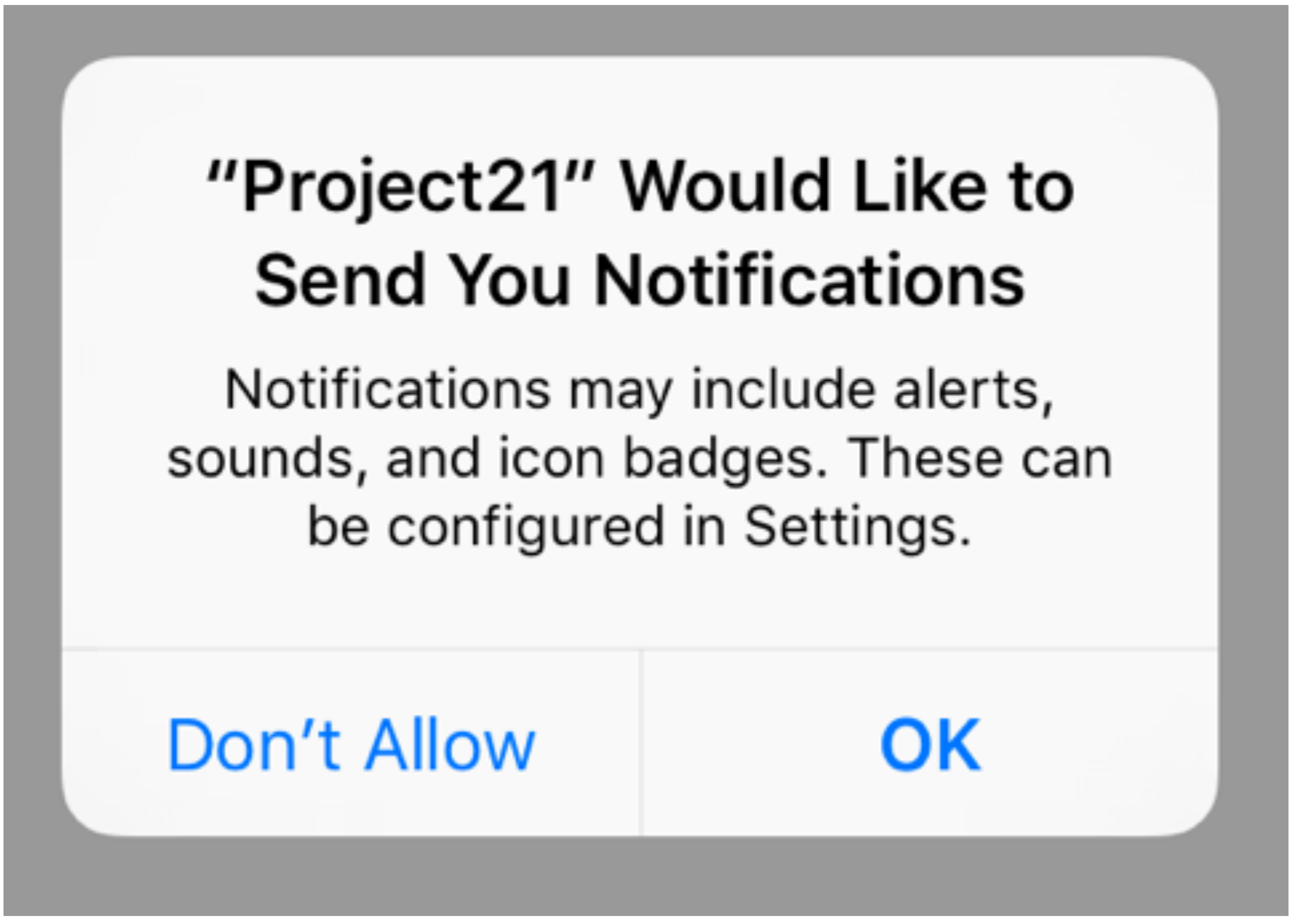
Once you've created your notification settings object, it's just a matter of calling the `registerUserNotificationSettings()` method to tell `iOS` what you want, and it will then prompt the user if needed. If you requested access before and were denied, nothing will be shown.

Change your `registerLocal()` method to be this:

```
@IBAction func registerLocal(sender: AnyObject) {
     let notificationSettings =
UIUserNotificationSettings(forTypes:
[.Alert, .Badge, .Sound], categories: nil)

UIApplication.sharedApplication().registerUserNotificationSet
tings(notificationSettings)
}
```

`Helpful tip`: if you want to test allowing or denying permission, just reset the simulator and run the app again to get a clean slate. Choose the `iOS Simulator` menu then `“Reset Content and Settings”` to make this happen.



“Project21” Would Like to Send You Notifications

Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.

Once we have user permission, it's time to fill in the `scheduleLocal()` method. This will use the `UILocalNotification` class to configure all the data needed to schedule a notification, then call `scheduleLocalNotification()` to schedule it for delivery to the user.

We're going to use the following properties:

- `fireDate` decides when the notification should be shown. iOS tracks this for us, so our app doesn't need to be running when it's time for the notification to be delivered.
- `alertBody` is a string containing the text to show to users. The title of the message will automatically be your app's name.
- `alertAction` is a string shown under your message that completes the sentence, `“Slide to…”. For example, if you set it be "pericombobulate"`, it would read `“Slide to pericombobulate”`.
- `soundName` we'll be using the default alert sound, but it's not hard to specify your own – just make sure you include it in the project!
- `userInfo` is a dictionary of keys and values that you can provide. The system does nothing with these other than hand them back to you when the app launches so you can respond.

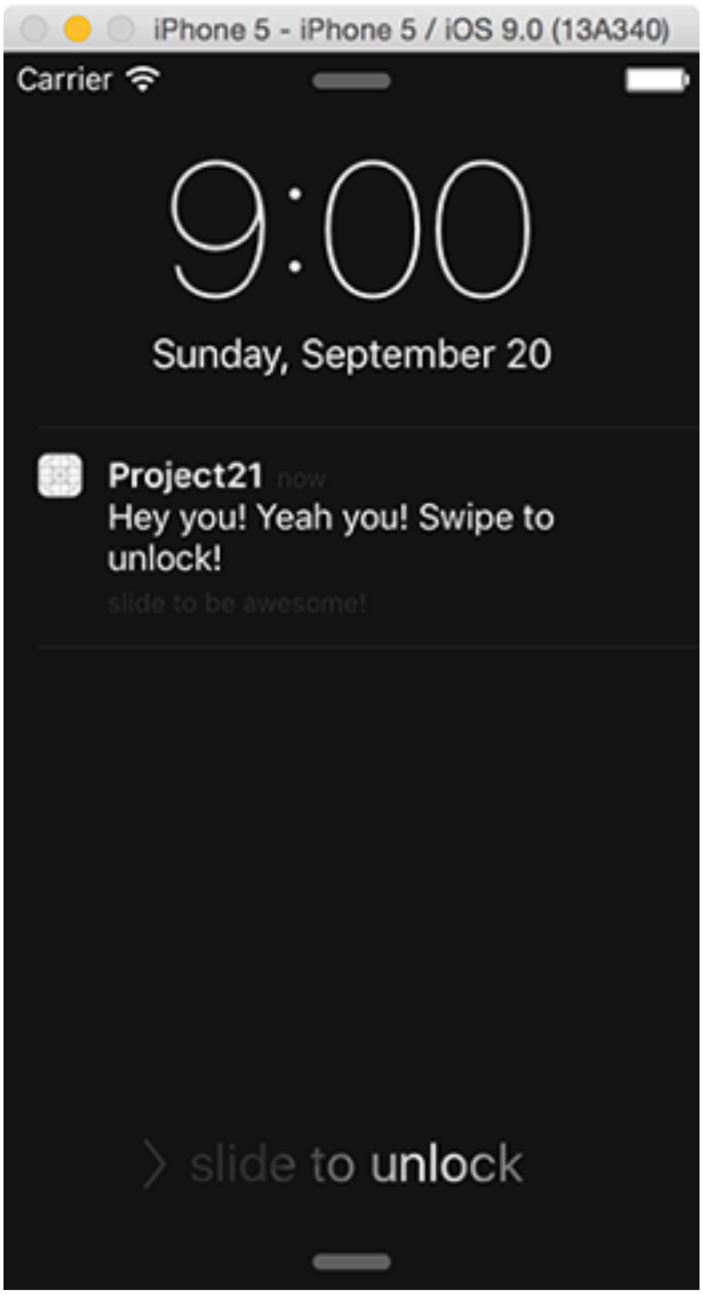
Here's the first draft of the `scheduleLocal()` method:

```
@IBAction func scheduleLocal(sender: AnyObject) {
     let notification = UILocalNotification()
     notification.fireDate = NSDate(timeIntervalSinceNow: 5)
     notification.alertBody = "Hey you! Yeah you! Swipe to
unlock!"
     notification.alertAction = "be awesome!"
```

```
    notification.soundName = 
UILocalNotificationDefaultSoundName
    notification.userInfo = ["CustomField1": "w00t"]

UIApplication.sharedApplication().scheduleLocalNotification(n
otification)
}
```

I'm providing a single key/value pair to the userInfo property; we'll come onto that soon.



Hey you! Yeah you! Swipe to unlock!

slide to be awesome!

That code will trigger a notification five seconds after you click “`Schedule Local`”, so:

- Click “`Register Local`” and agree to let the app show notifications.
- Click “`Schedule Local`”.
- Before the five seconds are up, press `Cmd + L` to lock the `iOS Simulator` screen.
- Wait.

You should see the message appear after a few seconds. Note that the alertAction text can be quite faint.

Before I move on, our `scheduleLocal()` method has a bug: what if the user doesn't grant us permission, and we try showing a local notification? Well, nothing will happen. That's good because your app didn't crash, but it's bad because users will think your app is broken.

To fix the bug, we need to modify `scheduleLocal()` so that it checks if we have permission to show local notifications before proceeding. This is as easy as querying the return value of `currentUserNotificationSettings()` for our application, and if it's `.None` then we need to alert the user and exit the method.

Put this code at the top of the `scheduleLocal()` notification:

```
guard let settings =
UIApplication.sharedApplication().currentUserNotificationSett
ings() else { return }

if settings.types == .None {
    let ac = UIAlertController(title: "Can't schedule",
message: "Either we don't have permission to schedule
notifications, or we haven't asked yet.",
preferredStyle: .Alert)
    ac.addAction(UIAlertAction(title: "OK", style: .Default,
handler: nil))
    presentViewController(ac, animated: true, completion:
nil)
    return
}
```

Acting on responses

There's one more thing to learn before we're done with notifications, and that's what happens to your application when it's given a notification to process. So far we've just been scheduling them, but if the user swipes on a notification to unlock their device, your app is launched and given the notification to process. Also, what happens if the user is actually inside your app while one of your notifications fires?

These are two separate cases under the hood, and both need to be addressed independently. In your `AppDelegate.swift` file we've already briefly looked at the `didFinishLaunchingWithOptions` method, but we didn't look at what the “`options`” might be. Well, if your app was lauched from a notification, this is how `iOS` will tell you, so let's take a closer look now.

First, this is how `launchOptions` is defined:

```
[NSObject: AnyObject]?
```

Translated, it's an optional dictionary where an `NSObject` (or subclass) will be the key and any object can be the value.

In order to read information about the notification, we need first to unwrap this optional so that we have a real dictionary on our ends. Then we need to look up the (*deep breath*) `UIApplicationLaunchOptionsLocalNotificationKey` key. Yes, that's absurdly long. If that exists, it will be a `UILocalNotification` object, so we need to conditionally typecast it using `as?`.

Once we have the `UILocalNotification` object, we need to see if it has a `userInfo` value, which means more optional unwrapping. And finally, if we've made it this far, we will have the same data we set for `userInfo` back when scheduling the notification, so you can do with it as you please.

Modify your `didFinishLaunchingWithOptions` method to this:

```
func application(application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [NSObject:
AnyObject]?) -> Bool {
    if let options = launchOptions {
        if let notification =
options[UIApplicationLaunchOptionsLocalNotificationKey] as?
UILocalNotification {
            if let userInfo = notification.userInfo {
                let customField1 = userInfo["CustomField1"]
as! String
                // do something neat here
            }
        }
    }

    return true
}
```

I called my data `CustomField1` but you can put whatever you want in there.

The other situation that might occur is if your app's notification fires while the app is still running. In this situation, it's obviously too late to call `didFinishLaunchingWithOptions`, so instead your app delegate will have its `didReceiveLocalNotification` method called. This doesn't exist by default, but it's easy enough to create.

As this method can only be called when a notification has definitely been received, you don't need to unwrap any launch options or look for any absurdly long dictionary keys. Instead, just check for the presence of userInfo and go from there. Add this method to your application delegate:

```
func application(application: UIApplication,
didReceiveLocalNotification notification:
UILocalNotification) {
    if let userInfo = notification.userInfo {
        let customField1 = userInfo["CustomField1"] as!
String
        print("didReceiveLocalNotification: \(customField1)")
    }
}
```

And that's it – you're able to go ahead and use local notifications all you want in your apps. But remember, if you abuse the trust your user has placed in you they will undoubtedly delete your app!

Wrap up

That was easy, right? And yet it's such a great feature to have, because now your app can talk to users even when it isn't running. You want to show a step count for how far they've walked? Local notification. You want to trigger an alert because it's their turn to play in a game? Local notification. You want to send them marketing messages to make them buy more stuff? Actually, just don't do that, you bad person.

If you're curious about push notifications then check out something like pushwizard.com, because you need a server somewhere that can store which devices to send to and handle delivery of the messages.