Local and Remote Notifications in Depth

The essential purpose of both local and remote notifications is to enable an app to inform its users that it has something for them—for example, a message or an upcoming appointment—when the app isn't running in the foreground. The essential difference between local notifications and remote notifications is simple:

- Local notifications are scheduled by an app and delivered on the same device.
- Remote notifications, also known as *push* notifications, are sent by your server to the Apple Push Notification service, which pushes the notification to devices.

Local and Remote Notifications Appear the Same to Users

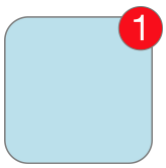
Users can get notified in the following ways:

- An onscreen alert or banner
- A badge on the app's icon
- A sound that accompanies an alert, banner, or badge

From a user's perspective, both local and remote notifications indicate that there is something of interest in the app.

For example, consider an app that manages a to-do list, and each item in the list has a date and time when the item must be completed. The user can request the app to notify it at a specific interval before this due date expires. To implement this behavior, the app schedules a local notification for that date and time. Instead of specifying an alert message, the app chooses to specify a badge number (1) and a sound. At the appointed time, iOS plays the sound and displays the badge number in the upper-right corner of the icon of the app, such as illustrated in Figure 1-1.

Figure 1-1 An app icon with a badge number (iOS)



The user hears the sound and sees the badge and responds by launching the app to see the to-do item. Users control how the device and specific apps installed on the device should handle notifications. They can also selectively enable or disable remote notification types (that is, icon badging, alert messages, and sounds) for specific apps.

Local and Remote Notifications Appear Different to Apps

When your app is frontmost, UIKit delivers local and remote notifications directly to your app delegate object without displaying any system UI. UIKit calls the `application:didReceiveLocalNotification:` method for incoming local notifications and the `application:didReceiveRemoteNotification:fetchCompletionHandler:` method for

incoming remote notifications. Use the provided notification dictionary to update your app accordingly. Because your app is running, you can incorporate the notification data quietly or update your user interface and let the user know that new information is available.

When your app must be launched to receive a notification, UIKit includes the `UIApplicationLaunchOptionsLocalNotificationKey` or `UIApplicationLaunchOptionsRemoteNotificationKey` key in the launch options dictionary passed to your app delegate's `application:willFinishLaunchingWithOptions:` and `application:didFinishLaunchingWithOptions:` methods. The presence of those keys lets you know that there is notification data waiting to be handled and gives you a chance to configure your app's interface appropriately. You do not need to handle the notification in these methods, though. After your app is running, UIKit calls other methods of your app delegate, such as the `application:didReceiveLocalNotification:` method, to give you an opportunity to process the notification data. Which methods are called depends on which methods you implemented and whether the user interacted with the system UI for the message.

When your app is running but not frontmost, UIKit displays the system UI and then delivers the results to your app in the background when possible. Similar to when your app is running, UIKit calls methods of your app delegate to receive the notification data and perform any actions. If it is unable to deliver the notification data in the background, UIKit waits until the next time your app runs to deliver it.

For more information about handling notifications, see Registering, Scheduling, and Handling User Notifications.

More About Local Notifications

Local notifications are ideally suited for apps with time-based behaviors, such as calendar and to-do list apps. Apps that run in the background for the limited period allowed by iOS might also find local notifications useful. For example, apps that depend on servers for messages or data can poll their servers for incoming items while running in the background; if a message is ready to view or an update is ready to download, they can handle the data as needed, and notify users if appropriate.

A local notification is an instance of `UILocalNotification` or `NSUserNotification` with three general kinds of properties:

- **Scheduled time.** You must specify the date and time the operating system delivers the notification; this is known as the *fire date*. You can qualify the fire date with a specific time zone so that the system can make adjustments to the fire date when the user travels. You can also request the operating system to reschedule the notification at a regular interval (weekly, monthly, and so on).
- **Notification type.** These properties include the alert message, the title of the default action button, the app icon badge number, a sound to play, and optionally in iOS 8 and later, a category of custom actions.
- **Custom data.** Local notifications can include a user info dictionary of custom data.

Listing 2-5 describes these properties in programmatic detail. After an app has created a local-notification object, it can either schedule it with the operating system or present it immediately.

Each app on a device is limited to 64 scheduled local notifications. The system discards scheduled notifications in excess of this limit, keeping only the 64 notifications that will fire the soonest. Recurring notifications are treated as a single notification.

More About Remote Notifications

An iOS or Mac app is often a part of a larger application based on the client/server model. The client side of the app is installed on the device or computer; the server side of the app has the main function of providing data to its client apps, and hence is termed a *provider*. A client app periodically connects with its provider and downloads any data that is waiting for it. Email and social-networking apps are examples of this client/server model.

But what if the app is not connected to its provider or even running on the device or computer when the provider has new data for it to download? How does it learn about this waiting data? Remote (or push) notifications are the solution to this dilemma. A remote notification is a short message that a provider has delivered to the operating system of a device or computer; the operating system, in turn, can inform the user of a client app that there is data to be downloaded, a message to be viewed, and so on. If the user enables this feature (on iOS) and the app is properly registered, the notification is delivered to the operating system and possibly to the app. Apple Push Notification service (APNs) is the primary technology for the remote-notification feature.

Remote notifications serve much the same purpose as a background app on a desktop system, but without the additional overhead. For an app that is not currently running—or, in the case of iOS, not running in the foreground—the notification occurs indirectly. The operating system receives a remote notification on behalf of the app and alerts the user. If the user then launches the app, it downloads the data from its provider. If an app is running when a notification comes in, the app can choose to handle the notification directly.

As its name suggests, Apple Push Notification service uses a remote design to deliver remote notifications to devices and computers. A push design differs from its opposite, a pull design, in that the recipient of the notification passively listens for updates rather than actively polling for them. A push design makes possible a wide and timely dissemination of information with few of the scalability problems inherent with pull designs. APNs uses a persistent IP connection for implementing remote notifications.

Most of a remote notification consists of a payload: a JSON dictionary containing APNs-defined properties specifying how the user is to be notified. The smaller you make the payload, the better the performance of your notifications. Although you can define custom properties, do not use the remote-notification mechanism for data transport. Delivery of remote notifications is best-effort with a high success rate but is not guaranteed. For more on the payload, see The Remote Notification Payload.

When a device is not online, APNs retains the last notification it receives from a provider for an app on that device. If the device then comes online, APNs pushes the stored notification to it. A device running iOS receives remote notifications over both Wi-Fi and cellular connections; a computer running OS X receives remote notifications over both Wi-Fi and Ethernet connections.

> **Note:** In iOS, remote notifications use Wi-Fi if it is turned on and connected. If Wi-Fi connection fails, remote notifications attempt to use the device's cellular connection.
>
> For some devices to receive notifications via Wi-Fi, the device's display must be on (that is, it cannot be sleeping) or it must be plugged in. iPad remains associated with its Wi-Fi access point while asleep, permitting the delivery of remote notifications. The iPad Wi-Fi radio wakes the host processor for incoming traffic as needed.
>
> Sending notifications too frequently negatively impacts device battery life because a device must access the network to receive a notification.

Adding the remote-notification feature to your app requires that you obtain the proper certificate from Member Center and then write the requisite code for the client and provider sides of the app. Provisioning and Development explains the provisioning and setup steps, and APNs Provider API and Registering, Scheduling, and Handling User Notifications describe the details of implementation.

For details on using Member Center for obtaining an APNs certificate, read Configuring Push Notifications in *App Distribution Guide*.