Design Tips

View controllers are an essential tool for apps running on iOS, and the view controller infrastructure of UIKit makes it easy to create sophisticated interfaces without writing a lot of code. When implementing your own view controllers, use the following tips and guidelines to ensure that you are not doing things that might interfere with the natural behavior expected by the system.

뷰 컨트롤러는 iOS에서 앱이 구동되는데 필수적인 도구이며 UIKit의 뷰 컨트롤러 구조는 많은 양의 코드 없이 정교한 인터페이스를 쉽게 생성하도록 도와줍니다. 뷰 컨트롤러를 구현할 때 시스템의 기본 동작에 방해가 되지 않도록 아래와 같은 팁과 가이드라인을 참고하세요.

Use System-Supplied View Controllers Whenever Possible

Many iOS frameworks define view controllers that you can use as-is in your apps. Using these system-supplied view controllers saves time for you and ensures a consistent experience for the user.

많은 iOS 프레임 워크는 앱에서 그대로 사용할 수 있는 뷰 컨트롤러를 제공합니다. 이러한 시스템에서 제공하는 뷰 컨트롤러를 사용하는 것은 많은 시간을 절약시켜 줄 수 있으며 사용자에게 일관된 경험을 줄 수 있을 것입니다.

Most system view controllers are designed for specific tasks. Some view controllers provide access to user data such as contacts. Others might provide access to hardware or provide specially tuned interfaces for managing media. For example, the `UIImagePickerController` class in UIKit displays a standard interface for capturing pictures and video and for accessing the user's camera roll.

대부분의 시스템 뷰 컨트롤러는 특별한 일을 수행하기 위해 디자인 되었습니다. 몇몇 뷰 컨트롤러는 연락처와 같은 유저 데이터에 접근 할 수 있도록 하고 다른 뷰 컨트롤러는 하드웨어에 대한 접근을 제공하거나 미디어 관리를 위한 인터페이스를 제공할 수 있습니다. 예를 들어, UIKit의 `UIImagePickerController`클래스는 사진 찍기와 비디오 인터페이스를 제공하고 유저의 카메라 앨범에 접근할 수 있도록 해줍니다.

Before you create your own custom view controller, look at the existing frameworks to see if a view controller already exists for the task you want to perform.

커스텀 뷰 컨트롤러를 구현하기 전에 원하는 기능을 구현한 뷰 컨트롤러가 이미 제공되는지 확인하세요.

The UIKit framework provides view controllers for displaying alerts, taking pictures and video, and managing files on iCloud. UIKit also defines many standard container view controllers that you can use to organize your content.

UIKit 프레임워크는 알림 경고창, 사진 및 비디오 촬영, iClod에서 파일을 관리하기 위한 뷰 컨트롤러들을 제공합니다. UIKit은 또한 컨텐츠를 구성하는 데 사용가능한 여러 컨테이너 뷰 컨트롤러들을 제공합니다.

The GameKit framework provides view controllers for matching players and for managing leaderboards, achievements, and other game features.

GameKit 프레임워크는 플레이어와 매칭시키고 리더 보드, 업적 및 기타 게임 기능을 관리하기 위한 뷰 컨트롤러들을 제공합니다.

The Address Book UI framework provides view controllers for displaying and picking contact information.

Address Book UI 프레임워크는 연락처 정보를 보여주고 선택할 수 있는 뷰 컨트롤러들을 제공합니다.

The MediaPlayer framework provides view controllers for playing and managing video, and for choosing media assets from the user's library.

MediaPlayer 프레임워크는 비디오를 재생하고 관리하고 유저의 라이브러리에서 미디어 리소스를 선택할 수 있는 뷰 컨트롤러들을 제공합니다.

The EventKit UI framework provides view controllers for displaying and editing the user's calendar data.

EventKit UI 프레임워크는 사용자의 달력 데이터를 보여주고 편집할 수 있는 뷰 컨트롤러를 제공합니다.

The GLKit framework provides a view controller for managing an OpenGL rendering surface.

GLKit 프레임워크는 OpenGL 렌더링을 제공하는 뷰 컨트롤러를 제공합니다.

The Multipeer Connectivity framework provides view controllers for detecting other users and inviting them to connect.

Multipeer Connectivity 프레임워크는 다른 사용자를 감지하고 연결을 위해 초대할 수 있는 뷰 컨트롤러들을 제공합니다.

The Message UI framework provides view controllers for composing emails and SMS messages.

Message UI 프레임워크는 이메일 및 SMS 메시지를 위한 뷰 컨트롤러를 제공합니다.

The PassKit framework provides view controllers for displaying passes and adding them to Passbook.

PassKit 프레임워크는 pass를 표시하고 Passbook에 추가할 수 있는 뷰 컨트롤러들을 제공합니다.

The Social framework provides view controllers for composing messages for Twitter, Facebook, and other social media sites.

Social 프레임워크는 Twitter, Facebook, 기타 다른 소셜 미디어 사이트에 메시지 작성을 위한 뷰 컨트롤러들을 제공합니다.

The AVFoundation framework provides a view controller for displaying media assets.

AVFoundation 프레임워크는 미디어 asset을 표시하기 위한 뷰 컨트롤러를 제공합니다.

> IMPORTANT
>
> Never modify the view hierarchy of system-provided view controllers. Each view controller owns its view hierarchy and is responsible for maintaining the integrity of that hierarchy. Making changes might introduce bugs into your code or prevent the owning view controller from operating correctly. In the case of system view controllers, always rely on the publicly available methods and properties of the view controller to make all modifications.
>
> 절대로 시스템에서 제공하는 뷰 컨트롤러의 뷰 계층을 변경하지 마세요. 각 뷰 컨트롤러는 자기의 고유한 뷰 계층을 가지고 있고 계층의 무결성을 유지하기 위한 책임을 가지고 있습니다. 만약 변경한다면 여러분의 코드에 수 많은 버그들이 나타날 수 있으며 뷰 컨트롤러가 정확한 동작을 수행하기 어려울 수 있습니다. 시스템 뷰 컨트롤러의 경우 모든 수정 작업을 할 때 항상 공개해 놓은 메소드들과 프로퍼티들을 사용하세요.

For information about using a specific view controller, see the reference documentation for the corresponding framework.

특정 뷰 컨트롤러를 사용하기 위한 자세한 정보는 해당 프레임워크는 reference 문서를 참고하세요.

Make Each View Controller an Island

View controllers should always be self-contained objects. No view controller should have knowledge about the internal workings or view hierarchy of another view controller. In cases where two view controllers need to communicate or pass data back and forth, they should always do so using explicitly defined public interfaces.

뷰 컨트롤러는 항상 자체적으로 포함된 객체여야 합니다. 어떠한 뷰 컨트롤러도 다른 뷰 컨트롤러의 뷰 계층 또는 내부적 동작방식을 알아선 안됩니다. 두 개의 뷰 컨트롤러가 데이터를 주고 받거나 통신해야 하는 경우 명시적으로 정의된 public 인터페이스를 사용해야 합니다.

The delegation design pattern is frequently used to manage communication between view controllers. With delegation, one object defines a protocol for communicating with an associated delegate object, which is any object that conforms to the protocol. The exact type of the delegate object is unimportant. All that matters is that it implements the methods of the protocol.

delegation 디자인 패턴은 뷰 컨트롤러 간의 커뮤니케이션에 빈번하게 사용됩니다. 델리게이션을 사용하면 하나의 객체를 프로토콜을 따르는 델리게이트 관련 객체와 통신하기 위해 프로토콜을 정의합니다. 델리게이트 객체의 정확한 타입은 중요치 않습니다. 중요한 것은 프로토콜의 메서드를 구현한다는 것입니다.

Use the Root View Only as a Container for Other Views

Use the root view of your view controller solely as a container for the rest of your content. Using the root view as a container gives all of your views a common parent view, which makes many layout operations simpler. Many Auto Layout constraints require a common parent view to lay out the views properly.

뷰 컨트롤러의 루트 뷰를 컨텐츠를 위한 컨테이너로만 사용하세요. 루트 뷰를 컨테이너로 사용하는 것은 모든 뷰 들에게 부모 뷰를 제공하므로 많은 레이아웃 작업들을 간단하게 해줄 수 있게 됩니다. 많은 오토 레이아웃 제약조건은 적절한 뷰의 레이아웃을 주기 위해 부모 뷰를 필요로 합니다.

Know Where Your Data Lives

In the model-view-controller design pattern, a view controller's role is to facilitate the movement of data between your model objects and your view objects. A view controller might store some data in temporary variables and perform some validation, but its main responsibility is to ensure that its views contain accurate information. Your data objects are responsible for managing the actual data and for ensuring the overall integrity of that data.

model-view-controller 디자인 패턴에서 뷰 컨트롤러의 역할은 모델 객체와 뷰 객체간의 데이터 이동을 원할하게 하는 것입니다. 뷰 컨트롤러는 일부 데이터를 임시 변수에 저장하고 유효성 검사를 수행하지만, 중요한 기능은 뷰에 정확한 정보가 포함되도록 보장하는 것입니다. 데이터 객체들은 실제 데이터를 관리하고 데이터의 전반적인 무결성을 보장합니다.

An example of the separation of data and interface exists in the relationship between the `UIDocument` and `UIViewController` classes. Specifically, no default relationship exists between the two. A `UIDocument` object coordinates the loading and saving of data, while a `UIViewController` object coordinates the display of views onscreen. If you create a relationship between the two objects, remember that the view controller should only cache information from the document for efficiency. The actual data still belongs to the document object.

데이터와 인터페이스 구분하는 예는 `UIDocument`클래스와 `UIViewController`클래스 사이의 관계에서 찾아 볼 수 있습니다. 특히 이 둘 사이에는 어떤 관계조차 존재하지 않습니다. `UIDocument` 객체는 데이터를 로드하고 저장을 조절하고 반면에 `UIViewController` 객체는 화면의 뷰의 디스플레이를 조절합니다. 만약 두 객체간의 관계를 생성한다면, 뷰 컨트롤러는 효율성을 위해 document의 정보를 오직 캐시로만 사용하도록 해야합니다. 실제 데이터는 여전히 document 객체에 속해야 합니다.

Adapt to Changes

Apps can run on a variety of iOS devices, and view controllers are designed to adapt to different-sized screens on those devices. Rather than use separate view controllers to manage content on different screens, use the built-in adaptivity support to respond to size and size class changes in your view controllers. The notifications sent by UIKit give you the opportunity to make both large-scale and small-scale changes to your user interface without having to change the rest of your view controller code.

앱은 다양한 iOS 디바이스에서 동작할 수 있으며 뷰 컨트롤러는 다양한 화면 사이즈를 가진 디바이스에도 적용되도록 디자인 되었습니다. 다양한 스크린을 위한 여러 뷰 컨트롤러를 사용하는 것보단 내장된 사이즈 클래스를 사용하여 대응하는 것이 좋습니다. UIKit이 보낸 notification은 코드의 변경하지 않아도 크고 작은 변화를 줄 수 있도록 합니다.

For more information about handling adaptivity changes, see The Adaptive Model.