About Windows and Views

In iOS, you use windows and views to present your application's content on the screen. Windows do not have any visible content themselves but provide a basic container for your application's views. Views define a portion of a window that you want to fill with some content. For example, you might have views that display images, text, shapes, or some combination thereof. You can also use views to organize and manage other views.

At a Glance

Every application has at least one window and one view for presenting its content. UIKit and other system frameworks provide predefined views that you can use to present your content. These views range from simple buttons and text labels to more complex views such as table views, picker views, and scroll views. In places where the predefined views do not provide what you need, you can also define custom views and manage the drawing and event handling yourself.

Views Manage Your Application's Visual Content

A view is an instance of the `UIView` class (or one of its subclasses) and manages a rectangular area in your application window. Views are responsible for drawing content, handling multitouch events, and managing the layout of any subviews. Drawing involves using graphics technologies such as Core Graphics, OpenGL ES, or UIKit to draw shapes, images, and text inside a view's rectangular area. A view responds to touch events in its rectangular area either by using gesture recognizers or by handling touch events directly. In the view hierarchy, parent views are responsible for positioning and sizing their child views and can do so dynamically. This ability to modify child views dynamically lets your views adjust to changing conditions, such as interface rotations and animations.

You can think of views as building blocks that you use to construct your user interface. Rather than use one view to present all of your content, you often use several views to build a view hierarchy. Each view in the hierarchy presents a particular portion of your user interface and is generally optimized for a specific type of content. For example, UIKit has views specifically for presenting images, text and other types of content.

Relevant Chapters: View and Window Architecture, Views

Windows Coordinate the Display of Your Views

A window is an instance of the `UIWindow` class and handles the overall presentation of your application's user interface. Windows work with views (and their owning view controllers) to manage interactions with, and changes to, the visible view hierarchy. For the most part, your application's window never changes. After the window is created, it stays the same and only the views displayed by it change. Every application has at least one window that displays the application's user interface on a device's main screen. If an external display is connected to the device, applications can create a second window to present content on that screen as well.

Relevant Chapters: Windows

Animations Provide the User with Visible Feedback for Interface Changes

Animations provide users with visible feedback about changes to your view hierarchy. The system defines standard animations for presenting modal views and transitioning between different groups of views. However, many attributes of a view can also be animated directly. For example, through animation you can change the transparency of a view, its position on the screen, its size, its background color, or other attributes. And if you work directly with the view's underlying Core Animation layer object, you can perform many other animations as well.

Relevant Chapters: Animations

The Role of Interface Builder

Interface Builder is an application that you use to graphically construct and configure your application's windows and views. Using Interface Builder, you assemble your views and place them in a nib file, which is a resource file that stores a freeze-dried version of your views and other objects. When you load a nib file at runtime, the objects inside it are reconstituted into actual objects that your code can then manipulate programmatically.

Interface Builder greatly simplifies the work you have to do in creating your application's user interface. Because support for Interface Builder and nib files is incorporated throughout iOS, little effort is required to incorporate nib files into your application's design.

For more information about how to use Interface Builder, see *Interface Builder User Guide*. For information about how view controllers manage the nib files containing their views, see Creating Custom Content View Controllers in *View Controller Programming Guide for iOS*.

See Also

Because views are very sophisticated and flexible objects, it would be impossible to cover all of their behaviors in one document. However, other documents are available to help you learn about other aspects of managing views and your user interface as a whole.

- View controllers are an important part of managing your application's views. A view controller presides over all of the views in a single view hierarchy and facilitates the presentation of those views on the screen. For more information about view controllers and the role they play, see *View Controller Programming Guide for iOS*.
- Views are the key recipients of gesture and touch events in your application. For more information about using gesture recognizers and handling touch events directly, see *Event Handling Guide for iOS*.
- Custom views must use the available drawing technologies to render their content. For information about using these technologies to draw within your views, see *Drawing and Printing Guide for iOS*.
- In places where the standard view animations are not sufficient, you can use Core Animation. For information about implementing animations using Core Animation, see *Core Animation Programming Guide*.