Class

UIImageView

A `UIImageView` object displays a single image or a sequence of animated images in your interface. Image views let you efficiently draw any image that can be specified using a **`UIImage`** object. For example, you can use this class to display the contents of many standard image files, such as JPEG and PNG files. You can configure image views programmatically or in your storyboard file and change the images they display at runtime. For animated images, you can also use the methods of this class to start and stop the animation and specify other animation parameters.

Language

Swift | Objective-C

SDKs

iOS 2.0+

tvOS 2.0+

On This Page

Overview

For information about basic view behaviors, see View Programming Guide for iOS.

Understanding How Images Are Scaled

An image view uses its `contentMode` property and the configuration of the image itself to determine how to display the image. It is best to specify images whose dimensions match the dimensions of the image view exactly, but image views can scale your images to fit all or some of the available space. If the size of the image view itself changes, it automatically scales the image as needed.

For an image without cap insets, the presentation of the image is determined solely by the image view's `contentMode` property. The `scaleAspectFit` and `scaleAspectFill` modes scale the image to fit or fill the space while maintaining the image's original aspect ratio. The `scaleToFill` value scales the image without regard to the original aspect ratio, which can cause the image to appear distorted. Other content modes place the image at the appropriate location in the image view's bounds without scaling it.

For a resizable image with cap insets, those insets affect the final appearance of the image. Specifically, cap insets define which parts of the image may be scaled and in which directions. You can create a resizable image that stretches using the `resizableImage(withCapInsets:resizingMode:)` method of `UIImage`. When using an image of this type, you typically set the image view's content mode to `scaleToFill` so that the image stretches in the appropriate places and fills the image view's bounds.

For tips on how to prepare images, see Debugging Issues with Your Image View. For more information on creating resizable images with cap insets, see `UIImage`.

Determining the Final Transparency of the Image

Images are composited onto the image view's background and are then composited into the rest of the window. Any transparency in the image allows the image view's background to show through. Similarly, any further transparency in the

of the image is dependent on the transparency of the image view and the transparency of the `UIImage` object it displays. When the image view and its image both have transparency, the image view uses alpha blending to combine the two.

- The image is composited onto the image view's background.
- If the image view's `isOpaque` property is `true`, the image's pixels are composited on top of the image view's background color and the `alpha` property of the image view is ignored.

- If the image view's `isOpaque` property is `false`, the alpha value of each pixel is multiplied by the image view's `alpha` value, with the resulting value becoming the actual transparency value for that pixel. If the image does not have an alpha channel, the alpha value of each pixel is assumed to be `1.0`.

> **Important**
>
> It is computationally expensive to composite the alpha channel of an image with the alpha channel of a non-opaque image view. The performance impact is further magnified if you use Core Animation shadows, because the shape of the shadow is then based on the contents of the view and must be dynamically computed. If you are not intentionally using the alpha channel of the image or the alpha channel of the image view, set the `isOpaque` property to `false` to improve performance. For additional optimization tips, see Tips for Improving Performance.

Animating a Sequence of Images

An image view can store an animated image sequence and play all or part of that sequence. You specify an image sequence as an array of `UIImage` objects and assign them to the `animationImages` property. Once assigned, you can use the methods and properties of this class to configure the animation timing and to start and stop the animation.

> **Note**
>
> You can also construct a single `UIImage` object from a sequence of individual images using the `animatedImage(with:duration:)` method. Doing so yields the same results as if you had assigned the individual images to the `animationImages` property.

Consider the following tips when displaying a sequence of animated images:

- **All images in the sequence should have the same size.** When scaling is required, the image view scales each image in the sequence separately. If the images are different sizes, scaling may not yield the results you want.

- **All images in the sequence should use the same content scale factor.** Make sure the `scale` property of each image contains the same value.

Responding to Touch Events

Image views ignore user events by default. Normally, you use image views only to present visual content in your interface. If you want an image view to handle user interactions as well, change the value of its `isUserInteractionEnabled` property to `true`. After doing that, you can attach gesture recognizers or use any other event handling techniques to respond to touch events or other user-initiated events.

For more information about handling events, see Event Handling Guide for iOS.

Tips for Improving Performance

Image scaling and alpha blending are two relatively expensive operations that can impact your app's performance. To maximize performance of your image view code, consider the following tips:

- **Cache scaled versions of frequently used images.** If you expect certain large images to be displayed frequently in a scaled-down thumbnail view, consider creating the scaled-down images in advance and storing them in a thumbnail cache. Doing so alleviates the need for each image view to scale them separately.

- **Use images whose size is close to the size of the image view.** Rather than assigning a large image to an image view, created a scaled version that matches the current size of the image view. You can also create a resizable image object using the `tile` option, which tiles the image instead of scaling it.

- **Make your image view opaque whenever possible.** Unless you are intentionally working with images that contain transparency (drawing UI elements, for example), make sure the `isOpaque` property of your image view is set to `true`. For more information about how transparency is determined, see Determining the Final Transparency of the Image.

Debugging Issues with Your Image View

If your image view is not displaying what you expected, use the following tips to help diagnose the problem:

- **Load images using the correct method.** Use the `init(named:in:compatibleWith:)` method of `UIImage` to load images from asset catalogs or your app's bundle. For images outside of your app's bundle, use the `imageWithContentsOfFile:` method.

- **Do not use image views for custom drawing.** The`UIImageView` class does not draw its content using the `draw(_:)` method. Use image views only to present images. To do custom drawing involving images, subclass `UIView` directly and draw your image there.

Interface Builder Attributes

`Table 1` lists the attributes that you configure for image views in Interface Builder.

Table 1 Image view attributes

Attribute	Discussion
Image	The image to display. You can specify any image in your Xcode project, including standalone images and those in image assets. To set this attribute programmatically, use the `image` or `animationImages` property.
Highlighted	The image to display when the image view is highlighted. To set this attribute programmatically, use the `highlightedImage` or `highlightedAnimationImag`

	`es` property.
State	The initial state of the image. Use this attribute to mark the image as highlighted. To set this attribute programmatically, use the `isHighlighted` property.

Internationalization

Internationalization of image views is automatic if your view displays only static images loaded from your app bundle. If you are loading images programmatically, you are at least partially responsible for loading the correct image.

- For resources in your app bundle, you do this by specifying the name in the attributes inspector or by calling the `init(named:)` class method on `UIImage` to obtain the localized version of each image.
- For images that are not in your app bundle, your code must do the following:

    1. Determine which image to load in a manner specific to your app, such as providing a localized string that contains the URL.
    2. Load that image by passing the URL or data for the correct image to an appropriate `UIImage` class method, such as `imageWithData:` or `imageWithContentsOfFile:`.

> **Note**
>
> Screen metrics and layout may also change depending on the language and locale, particularly if the internationalized versions of your images have different dimensions. Where possible, you should try to make minimize dimension differences in internationalized versions of image resources.

For more information, see Internationalization and Localization Guide.

Accessibility

Image views are accessible by default. The default accessibility traits for a image view are Image and User Interaction Enabled.

For more information about making iOS controls accessible, see the accessibility information in `UIControl`. For general information about making your interface accessible, see Accessibility Programming Guide for iOS.

State Preservation

When you assign a value to an image view's `restorationIdentifier` property, it attempts to preserve the frame of the displayed image. Specifically, the class preserves the values of the `bounds`, `center`, and `transform` properties of the view and the `anchorPoint` property of the underlying layer. During restoration, the image view restores these values so that the image appears exactly as before. For more information about how state preservation and restoration works, see App Programming Guide for iOS.

Symbols

Initializing an Image View

`init(image: UIImage?)`

Returns an image view initialized with the specified image.

```
init(image: UIImage?, highlightedImage: UIImage?)
```
Returns an image view initialized with the specified regular and highlighted images.

Accessing the Displayed Images

```
var image: UIImage?
```
The image displayed in the image view.

```
var highlightedImage: UIImage?
```
The highlighted image displayed in the image view.

Animating a Sequence of Images

```
var animationImages: [UIImage]?
```
An array of `UIImage` objects to use for an animation.

```
var highlightedAnimationImages: [UIImage]?
```
An array of `UIImage` objects to use for an animation when the view is highlighted.

```
var animationDuration: TimeInterval
```
The amount of time it takes to go through one cycle of the images.

```
var animationRepeatCount: Int
```
Specifies the number of times to repeat the animation.

```
func startAnimating()
```
Starts animating the images in the receiver.

```
func stopAnimating()
```
Stops animating the images in the receiver.

Configuring the Image View

`var isUserInteractionEnabled: Bool`

A Boolean value that determines whether user events are ignored and removed from the event queue.

`var isHighlighted: Bool`

A Boolean value that determines whether the image is highlighted.

`var tintColor: UIColor!`

A color used to tint template images in the view hierarchy.

Managing Focus-Related Behaviors

`var adjustsImageWhenAncestorFocused: Bool`

Allows `UIImageView` to respond when an ancestor becomes focused.

`var focusedFrameGuide: UILayoutGuide`

The layout guide to use when the image view is focused.

Instance Properties

`var isAnimating: Bool`

Relationships

Inherits From

`UIView`

Conforms To

- `CVarArg`
- `Equatable`
- `Hashable`
- `UIAccessibilityIdentification`