Image Views

On This Page

An image view displays an image or an animated sequence of images. An image view lets you efficiently draw an image (such as a JPEG or PNG file) or an animated series of images onscreen, scaling the images automatically to fit within the current size of the view. Image views can optionally display a different image or series of images whenever the view is highlighted. Image views support the same file formats as the `UIImage` class—TIFF, JPEG, PNG, Windows bitmap (bmp), Windows icon (ico), Windows cursor (cur), and X Window System bitmap (xbm) formats.

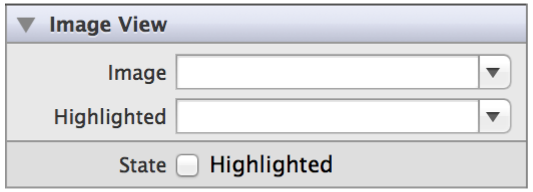
Purpose. Image views allow users to:

- View images within an app

Implementation. Image views are implemented in the `UIImageView` class and discussed in the *UIImageView Class Reference*.

Configuration. Configure image views in Interface Builder, in the Image View section of the Attributes Inspector. A few

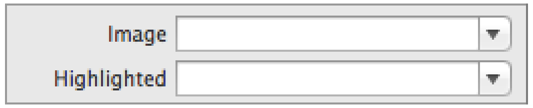
programmatically. You can set other configurations programmatically, too, if you prefer.



Content of Image Views

Image views provide significant programmatic customization by modifying properties on the view objects and properties on whatever image object you have loaded into the view.

If you are displaying a single image, most image views require minimal configuration beyond setting the image. If you are displaying an animated series of images, you must also configure the animation settings.



When you first use an image view object to display a single image, you can select an image to display using the Image (`image`) field in the Attributes Inspector. If you did not choose an image in the Attributes Inspector, you must set the initial image by calling `initWithImage:` or by setting the `image` property to a `UIImage` object that describes the image you want to display.

If you want to show a different image when the view is highlighted, you can specify this information in the Highlighted (`highlightedImage`) field. Alternatively, either call `initWithImage:highlightedImage:` when you initialize the image or set the `highlightedImage` property to the alternative image.

If you want your image view to display an animated sequence of images, you must do this programmatically. Because you cannot specify an array of images in the Attributes Inspector, you must write some code to tell the image view which images to use. To do this, set the `animationImages` property to an array of `UIImage` objects in the order in which they should be shown. Optionally set the `highlightedAnimationImages` property if you want to sh

On This Page

> IMPORTANT
>
> All images associated with a `UIImageView` object should have the same `scale` value. If your application uses images with different scales, they may render incorrectly.

Behavior of Image Views

Use the Highlighted (`highlighted`) checkbox to specify whether the view should show the standard or highlighted image or image sequence.

You can change the image view's state at any time.

If you are using an image sequence, you can instead configure the animation behavior programmatically:

- Set the `animationDuration` to the desired animation period (in seconds). By default, this property is computed based on the number of images at 30 frames per second.
- Set the `animationRepeatCount` to limit the number of iterations through the set of images. By default, this property has a value of zero, which means that the animation repeats forever.

You start the animation by calling `startAnimating`.

Appearance of Image Views

You cannot customize the appearance of an image view directly. However, you can determine how images appear in the view by setting properties at the `UIImage` and `UIView` levels.

Content Mode

The view's `contentMode` property specifies how the image should be scaled and aligned within the view. It is recommended (but not required) that you use images that are all the same size. If the images are different sizes, each will be adjusted to fit separately based on this mode.

Images

The image's `capInsets`, `leftCapWidth`, and `topCapHeight` properties specify the width and height of a central portion of the image that should be scaled differently than the border areas (outside that central portion). The top and bottom border areas are tiled horizontally. The left and right border areas are tiled vertically. The corners are displayed as-is. Additionally, the image's `alignmentRectInsets` property specifies portions of the image to ignore for alignment purposes (such as shadow and glow effects).

You can create images for images views in a number of ways, including:

- Using the `imageWithAlignmentRectInsets:` method, which returns a derived image with nonzero alignment insets.
- Using the `resizableImageWithCapInsets:` or `resizableImageWithCapInsets:resizingMode:` methods, which return a derived *static* image with nonzero cap insets. The image's `resizingMode` property indicates whether the image should be scaled or tiled.
- Using the `animatedResizableImageNamed:capInsets:duration:` or `animatedResizableImageNamed:capInsets:resizingMode:duration:` methods, which return a derived *animated* image with nonzero cap insets.

These methods cannot be set after the image is created or loaded.

Transparency and Alpha Blending

Transparency of an image view is defined by properties of both the underlying image and the view as follows:

- If the view’s Opaque (`opaque`) flag is set, the image is alpha blended with the background color of the view, and the view itself is opaque. The view’s Alpha (`alpha`) setting is ignored.
- If the view’s Opaque (`opaque`) flag is not set, the alpha channel for each pixel (or `1.0` if the image has no alpha cha
  transparency for that pixel.

> IMPORTANT
>
> It is computationally expensive to perform alpha compositing of non-opaque image views containing images with alpha channels. If you are not intentionally using the image alpha channel or the view’s Alpha setting, you should set the Opaque flag to improve performance. See the last bullet point of Debugging Image Views for more information.

Using Auto Layout with Image Views

You can create Auto Layout constraints between a image view and other user interface elements. You can create any type of constraint for a image view besides a baseline constraint.

You generally want the image view to fill the full width of your screen. To ensure that this happens correctly on all devices and orientations, you can create Leading Space to Superview and Trailing Space to Superview constraints, and set both values equal to `0`. This will ensure that the image view remains pinned to the edges of the device screen.

For general information about using Auto Layout with iOS views, see Using Auto Layout with Views.

Making Image Views Accessible

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

For general information about making iOS views accessible, see Making Views Accessible.

Internationalizing Image Views

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.

On This Page

- For resources in your app bundle, you do this by specifying the name in the attributes inspector or by calling the `imageNamed:` 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:
    - Determine which image to load in a manner specific to your app, such as providing a localized string that contains the URL.
    - Load that image by passing the URL or data for the correct image to an appropriate `UIImage` class method, such as `imageWithData:` or `imageWithContentsOfFile:`.

> TIP
>
> 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*.

Debugging Image Views

When debugging issues with image views, watch for these common pitfalls:

- **Not loading your image with the correct method.** If you are loading an image from your app bundle, use `imageNamed:`. If you are loading an image from a file (with a full path or URL), use `imageWithContentsOfFile:`.
- **Not making animated image frames the same size.** This helps you avoid having to deal with scaling, tiling, or positioning differences between frames.
- **Not using a consistent scale value for all animated image frames.** Mixing images with different scale factors may produce undefined behavior.

- **Doing custom drawing in a subclass of an image view.** The `UIImageView` class is optimized to draw its images to the display. `UIImageView` does not call the `drawRect:` method of its subclasses. If your subclass needs to include custom drawing code, you should subclass the `UIView` class instead.
- **Not enabl**

  disregard

  must explicitly change the value of the `userInteractionEnabled` property to `YES` after initializing the object.

On This Page

- **Not providing prescaled images where possible.** For example, if you expect certain large images to be frequently displayed in a scaled-down thumbnail view, you might consider keeping the scaled-down images in a thumbnail cache. Scaling the image is a relatively expensive operation.
- **Not limiting image size.** Consider prescaling or tiling large images. The *MVCNetworking* sample code project (`QImageScrollView.m`) demonstrates how to determine what model of iOS device your software is running on. You can then use that information to help you determine the image dimension thresholds to use when scaling or tiling.
- **Not disabling alpha blending except where needed.** Unless you are intentionally working with images that contain transparency (drawing UI elements, for example), you should generally mark the view as opaque by selecting the Opaque checkbox in the Attributes Inspector, or setting the `opaque` property on the view itself.

  For views that are not opaque, the device must perform a lot of unnecessary computation if alpha blending is enabled and the image contains an alpha channel. This performance impact is further magnified if you are using Core Animation shadows, because the shape of the shadow is then based on the contents of the view, and must be dynamically computed.

  To learn more about how alpha blending works, see Transparency and Alpha Blending.

Elements Similar to an Image View

The following elements provide similar functionality to a web view:

- **Button.** You can set the background image of a button control (of type `UIButtonTypeCustom`). For more information, see Buttons.
- **Scroll View.** An image view typically scales content up or down to fit the dimensions of the view. If you need to display an image with user-controlled zooming and scaling, you should place that image view inside a scroll view. For more information, see Scroll Views.

- **Custom Views.** If you create a custom subclass of `UIView`, you can programmatically draw images inside its `drawRect:` method. (For maximum performance, you should do this only when absolutely necessary.) For more information, see About Views.

On This Page