

---

## PINRemoteImage

### Fast, non-deadlocking parallel image downloader and cache for iOS



PINRemoteImageManager is an image downloading, processing and caching manager. It uses the concept of download and processing tasks to ensure that even if multiple calls to download or process an image are made, it only occurs one time (unless an item is no longer in the cache). PINRemoteImageManager is backed by **GCD** and safe to **access** from **multiple threads** simultaneously. It ensures that images are decoded off the main thread so that animation performance isn't affected. None of its exposed methods allow for synchronous access. However, it is optimized to call completions on the calling thread if an item is in its memory cache.

PINRemoteImage supports downloading many types of files. It, of course, **supports** both **PNGs** and **JPGs**. It also supports decoding **WebP** images if Google's library is available. It even supports **GIFs** and **Animated WebP** via PINAnimatedImageView.

PINRemoteImage also has two methods to improve the experience of downloading images on slow network connections. The first is support for **progressive JPGs**. This isn't any old support for progressive JPGs though: PINRemoteImage adds an attractive blur to progressive scans before returning them.



---

PINRemoteImageCategoryManager defines a protocol which UIImageView subclasses can implement and provide easy access to PINRemoteImageManager's methods. There are **built-in categories** on **UIImageView**, **PINAnimatedImageView** and **UIButton**, and it's very easy to implement a new category. See UIImageView+PINRemoteImage of the existing categories for reference.

### Download an image and set it on an image view:

#### Objective-C

```
1 UIImageView *imageView = [[UIImageView alloc] init];
2 [imageView pin_setImageFromURL:[NSURL URLWithString:@"http://pinterest.com/kitten.jpg"]];
```

#### Swift

```
1 let imageView = UIImageView()
2 imageView.pin_setImage(from: URL(string: "https://pinterest.com/kitten.jpg")!)
```

### Download a progressive jpeg and get attractive blurred updates:

#### Objective-C

```
1 UIImageView *imageView = [[UIImageView alloc] init];
2 [imageView setPin_updateWithProgress:YES];
3 [imageView pin_setImageFromURL:[NSURL URLWithString:@"http://pinterest.com/progressiveKitten.jpg"]];
```

#### Swift

```
1 let imageView = UIImageView()
2 imageView.pin_updateWithProgress = true
3 imageView.pin_setImage(from: URL(string: "https://pinterest.com/progressiveKitten.jpg")!)
```

### Download a WebP file

#### Objective-C

```
1 UIImageView *imageView = [[UIImageView alloc] init];
2 [imageView pin_setImageFromURL:[NSURL URLWithString:@"http://pinterest.com/googleKitten.webp"]];
```

#### Swift

---

```
1 let imageView = UIImageView()
2 imageView.pin_setImage(from: URL(string: "https://pinterest.com/
  googleKitten.webp")!)
```

## Download a GIF and display with PINAnimatedImageView

### Objective-C

```
1 PINAnimatedImageView *animatedImageView = [[PINAnimatedImageView alloc]
  init];
2 [animatedImageView pin_setImageFromURL:[NSURL URLWithString:@"http://
  pinterest.com/flyingKitten.gif"]];
```

### Swift

```
1 let animatedImageView = PINAnimatedImageView()
2 animatedImageView.pin_setImage(from: URL(string: "http://pinterest.com/
  flyingKitten.gif")!)
```

## Download and process an image

### Objective-C

```
1 UIImageView *imageView = [[UIImageView alloc] init];
2 [self.imageView pin_setImageFromURL:[NSURL URLWithString:@"https://i.
  pinimg.com/736x/5b/c6/c5/5bc6c5387ff6f104fd642f2b375efba3.jpg"]
  processorKey:@"rounded" processor:^(UIImage *(
    PINRemoteImageManagerResult *result, NSUInteger *cost)
3 {
4     CGSize targetSize = CGSizeMake(200, 300);
5     CGRect imageRect = CGRectMake(0, 0, targetSize.width, targetSize.
      height);
6     UIGraphicsBeginImageContext(imageRect.size);
7     UIBezierPath *bezierPath = [UIBezierPath bezierPathWithRoundedRect
      :imageRect cornerRadius:7.0];
8     [bezierPath addClip];
9
10    CGFloat sizeMultiplier = MAX(targetSize.width / result.image.size.
      width, targetSize.height / result.image.size.height);
11
12    CGRect drawRect = CGRectMake(0, 0, result.image.size.width *
      sizeMultiplier, result.image.size.height * sizeMultiplier);
13    if (CGRectGetMaxX(drawRect) > CGRectGetMaxX(imageRect)) {
14        drawRect.origin.x -= (CGRectGetMaxX(drawRect) - CGRectGetMaxX(
      imageRect)) / 2.0;
15    }
```

---

```
16     if (CGRectGetMaxY(drawRect) > CGRectGetMaxY(imageRect)) {
17         drawRect.origin.y -= (CGRectGetMaxY(drawRect) - CGRectGetMaxY(
18             imageRect)) / 2.0;
19     }
20     [result.image drawInRect:drawRect];
21
22     UIImage *processedImage =
23         UIGraphicsGetImageFromCurrentImageContext();
24     UIGraphicsEndImageContext();
25     return processedImage;
26 }];
```

## Swift

```
1 let imageView = FLAnimatedImageView()
2 imageView.pin_setImage(from: URL(string: "https://i.pinimg.com/736x/5b/
   c6/c5/5bc6c5387ff6f104fd642f2b375efba3.jpg")!, processorKey: "
   rounded") { (result, unsafePointer) -> UIImage? in
3
4     guard let image = result.image else { return nil }
5
6     let radius : CGFloat = 7.0
7     let targetSize = CGSize(width: 200, height: 300)
8     let imageRect = CGRect(x: 0, y: 0, width: targetSize.width, height:
   targetSize.height)
9
10    UIGraphicsBeginImageContext(imageRect.size)
11
12    let bezierPath = UIBezierPath(roundedRect: imageRect, cornerRadius:
   radius)
13    bezierPath.addClip()
14
15    let widthMultiplier : CGFloat = targetSize.width / image.size.width
16    let heightMultiplier : CGFloat = targetSize.height / image.size.
   height
17    let sizeMultiplier = max(widthMultiplier, heightMultiplier)
18
19    var drawRect = CGRect(x: 0, y: 0, width: image.size.width *
   sizeMultiplier, height: image.size.height * sizeMultiplier)
20    if (drawRect.maxX > imageRect.maxX) {
21        drawRect.origin.x -= (drawRect.maxX - imageRect.maxX) / 2
22    }
23    if (drawRect.maxY > imageRect.maxY) {
24        drawRect.origin.y -= (drawRect.maxY - imageRect.maxY) / 2
25    }
26
27    image.draw(in: drawRect)
28
29    UIColor.red.setStroke()
30    bezierPath.lineWidth = 5.0
```

---

```

31     bezierPath.stroke()
32
33     let ctx = UIGraphicsGetCurrentContext()
34     ctx?.setBlendMode(CGBlendMode.overlay)
35     ctx?.setAlpha(0.5)
36
37     let logo = UIImage(named: "white-pinterest-logo")
38     ctx?.scaleBy(x: 1.0, y: -1.0)
39     ctx?.translateBy(x: 0.0, y: -drawRect.size.height)
40
41     if let coreGraphicsImage = logo?.cgImage {
42         ctx?.draw(coreGraphicsImage, in: CGRect(x: 90, y: 10, width:
43             logo!.size.width, height: logo!.size.height))
44     }
45
46     let processedImage = UIGraphicsGetImageFromCurrentImageContext()
47     UIGraphicsEndImageContext()
48
49     return processedImage
50 }

```

## Handle Authentication

### Objective-C

```

1  [[PINRemoteImageManager sharedImageManager] setAuthenticationChallenge
   :^(NSURLSessionTask *task, NSURLAuthenticationChallenge *challenge,
   PINRemoteImageManagerAuthenticationChallengeCompletionHandler
   aCompletion) {
2  aCompletion(NSURLSessionAuthChallengePerformDefaultHandling, nil)];

```

### Swift

```

1  PINRemoteImageManager.shared().setAuthenticationChallenge { (task,
   challenge, completion) in
2  completion?(.performDefaultHandling, nil)
3  }

```

## Support for high resolution images

Currently there are two ways `PINRemoteImage` is supporting high resolution images: 1. If the URL contains an `_2x.` or an `_3x.` postfix it will be automatically handled by `PINRemoteImage` and the resulting image will be returned at the right scale. 2. If it's not possible to provide an URL with an `_2x.` or `_3x.` postfix, you can also handle it with a completion handler:

---

```
1 NSURL *url = ...;
2 __weak UIImageView *weakImageView = self.imageView;
3 [self.imageView pin_setImageFromURL:url completion:^(
4     PINRemoteImageManagerResult * _Nonnull result) {
5     CGFloat scale = UIScreen.mainScreen.scale;
6     if (scale > 1.0) {
7         UIImage *image = result.image;
8         weakImageView.image = [UIImage imageWithCGImage:image.CGImage scale
9             :scale orientation:image.imageOrientation];
10    }
11 }];
```

## Set some limits

```
1 // cache is an instance of PINCache as long as you haven't overridden
   defaultImageCache
2 PINCache *cache = (PINCache *)[PINRemoteImageManager
   sharedImageManager] cache];
3 // Max memory cost is based on number of pixels, we estimate the size
   of one hundred 600x600 images as our max memory image cache.
4 [[cache memoryCache] setCostLimit:600 * [[UIScreen mainScreen] scale] *
   600 * [[UIScreen mainScreen] scale] * 100];
5
6 // ~50 MB
7 [[cache diskCache] setByteLimit:50 * 1024 * 1024];
8 // 30 days
9 [[cache diskCache] setAgeLimit:60 * 60 * 24 * 30];
```

## Installation

### CocoaPods

Add `PINRemoteImage` to your `Podfile` and run `pod install`.

If you'd like to use WebP images, add `PINRemoteImage/WebP` to your `Podfile` and run `pod install`.

### Carthage

Add `github "pinterest/PINRemoteImage"` to your `Cartfile`. See Carthage's readme for more information on integrating Carthage-built frameworks into your project.



---

## Manually

Download the latest tag and drag the `Pod/Classes` folder into your Xcode project. You must also manually link against `PINCache`.

Install the docs by double clicking the `.docset` file under `docs/`, or view them online at [cocoapods.org](http://cocoapods.org)

## Git Submodule

You can set up `PINRemoteImage` as a submodule of your repo instead of cloning and copying all the files into your repo. Add the submodule using the commands below and then follow the manual instructions above.

```
1 git submodule add https://github.com/pinterest/PINRemoteImage.git
2 git submodule update --init
```

## Requirements

**PINRemoteImage** requires iOS 7.0 or greater.

## Contact

Garrett Moon @garrettmooon Pinterest

## License

Copyright 2015 Pinterest, Inc.

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.