

Extracted from:

iOS Recipes

Tips and Tricks for Awesome iPhone and iPad Apps

This PDF file contains pages extracted from *iOS Recipes*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2010 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

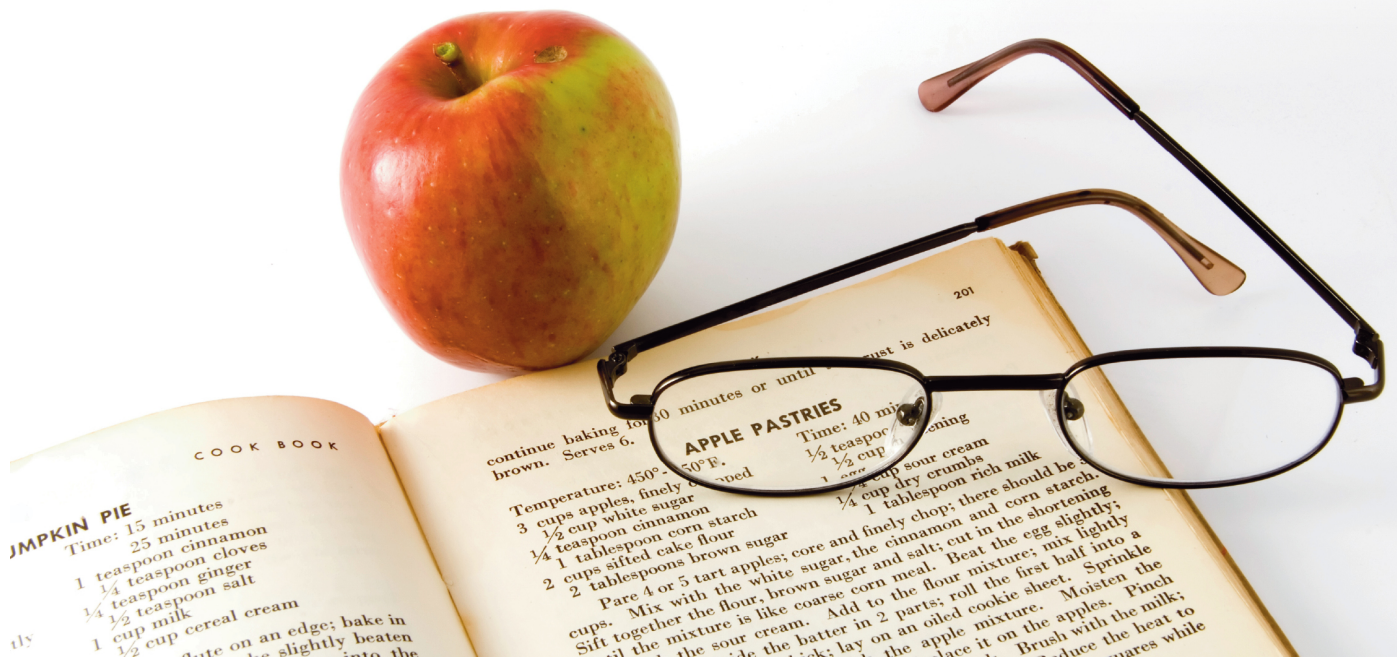
Dallas, Texas • Raleigh, North Carolina

iOS Recipes

Tips and Tricks for Awesome iPhone and iPad Apps

Matt Drance
Paul Warren

Edited by Jill Steinberg



COOK BOOK

JMPKIN PIE
Time: 15 minutes
25 minutes
1 teaspoon cinnamon
1/4 teaspoon cloves
1/4 teaspoon ginger
1/2 teaspoon salt
1 cup milk
1/2 cup cereal cream

continue baking for 30 minutes or until brown. Serves 6.
Temperature: 450°-500° F.
3 cups apples, finely chopped
1/2 cup white sugar
1/4 teaspoon cinnamon
2 cups sifted cake flour
2 tablespoons brown sugar

APPLE PASTRIES

Time: 40 minutes
1/2 teaspoon cinnamon
1/2 cup dry crumbs
1 cup sour cream
1/4 cup rich milk
1 tablespoon corn starch
1 tablespoon rich milk
1/4 cup dry crumbs
1 cup sour cream
1/2 teaspoon cinnamon
1/2 cup white sugar
3 cups apples, finely chopped

iOS Recipes

Tips and Tricks for Awesome iPhone and iPad Apps

Matt Drance

Paul Warren

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <http://pragprog.com>.

The team that produced this book includes:

Jill Steinberg (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Copyright © 2011 Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-934356-74-6
Printed on acid-free paper.
Book version: P1.0—July 2011

Tame the Network Activity Indicator

Problem

Your application performs downloads and uploads in multiple places, queuing or parallelizing them under heavy user activity. You need to reliably display network status without actively tracking every network operation.

Solution

We can use the `networkActivityIndicatorVisible` property on `UIApplication` to conveniently show and hide the network “spinner” in the status bar. This binary switch has no context, however. If we write an application that performs concurrent uploads and downloads, it quickly becomes hard to accurately report ongoing activity. Showing the indicator when every transaction starts is easy, but how do we know when to hide it? Whether we’re using `NSURLConnection` or `NSStream`, our networking code should not necessarily be responsible for maintaining the context required to manage the network activity indicator. We’ll solve this problem with a category on `UIApplication` that tracks network connections, automatically showing the indicator when activity begins and hiding it when it is finished. By using a category, we can call the existing `UIApplication` instance rather than managing another object. This especially makes sense since the activity indicator itself is managed by `UIApplication`.

This `PRPNetworkActivity` category maintains a read-only count of active connections. Two methods, `-prp_pushNetworkActivity` and `-prp_popNetworkActivity`, allow any code to notify the application of network activity. A `-prp_resetNetworkActivity` method clears the current state and starts from scratch.

Download `NetworkActivityCenter/Classes/UIApplication+PRPNetworkActivity.h`

```
@interface UIApplication (PRPNetworkActivity)
```

```
@property (nonatomic, assign, readonly) NSInteger prp_networkActivityCount;
```

```
- (void)prp_pushNetworkActivity;  
- (void)prp_popNetworkActivity;  
- (void)prp_resetNetworkActivity;
```

```
@end
```

Remember that because this is a category, it's important to prefix all of the method names to ensure they don't conflict with any methods Apple adds to UIApplication in future SDK releases.

The implementation is very simple: we declare a static `prp_networkActivityCount` variable, which the `-prp_pushNetworkActivity` and `-prp_popNetworkActivity` methods respectively increment and decrement. A simple getter method exposes the count in a read-only fashion.

```
Download NetworkActivityCenter/Classes/UIApplication+PRPNetworkActivity.m
- (NSInteger)prp_networkActivityCount {
    @synchronized(self) {
        return prp_networkActivityCount;
    }
}

- (void)prp_pushNetworkActivity {
    @synchronized(self) {
        prp_networkActivityCount++;
    }
    [self prp_refreshNetworkActivityIndicator];
}

- (void)prp_popNetworkActivity {
    @synchronized(self) {
        if (prp_networkActivityCount > 0) {
            prp_networkActivityCount--;
        } else {
            prp_networkActivityCount = 0;
            NSLog(@"%s Unbalanced network activity: count already 0.",
                __PRETTY_FUNCTION__);
        }
    }
    [self prp_refreshNetworkActivityIndicator];
}
```

A few notes about this approach:

- We use a global to store the activity count, but our category methods operate on an instance of UIApplication. Always be careful when sharing statics between object instances. An ideal solution might use the associated object approach explained in [Recipe 40, Store Data in a Category, on page ?](#), but since there is only a single UIApplication instance in a given app, we stuck with the global in the interest of simplicity.
- The methods listed earlier access the activity count while synchronizing on self, which is the shared application instance since we've written a category on UIApplication. We have added this synchronization because

networking code that uses these category methods is likely to run on multiple threads. There is more than one way to synchronize Objective-C code, so we've chosen what we saw as the clearest solution.

The `-prp_refreshNetworkActivityIndicator` method sets the standard `networkActivityIndicatorVisible` property on `UIApplication` according to the current activity count: if the count is positive, the network activity indicator is shown; when it goes back down to 0, the indicator is hidden. Because most of the `UIKit` is not understood to be thread-safe and the `networkActivityIndicatorVisible` property is not explicitly documented as such, we write a check to ensure the network activity indicator is touched only from the main thread.

Download `NetworkActivityCenter/Classes/UIApplication+PRPNetworkActivity.m`

```
- (void)prp_refreshNetworkActivityIndicator {
    if (![NSThread isMainThread]) {
        SEL sel_refresh = @selector(prp_refreshNetworkActivityIndicator);
        [self performSelectorOnMainThread:sel_refresh
            withObject:nil
            waitUntilDone:NO];
        return;
    }

    BOOL active = (self.prp_networkActivityCount > 0);
    self.networkActivityIndicatorVisible = active;
}
```

We now have reliable network state management accessible from anywhere in our application and completely decoupled from the rest of our code. Just call `-prp_pushNetworkActivity` whenever starting a connection, and call `-prp_popNetworkActivity` whenever the connection terminates.

The `NetworkActivityCenter` sample project demonstrates this code in action. We've modified the `PRPDownload` class from an earlier recipe to push and pop activity based on the status of each download. Neither these download objects nor the test app's view controller has any idea of one another, let alone what each is doing with the network. Each object reports its state to the `UIApplication` category methods, which decide when the network activity indicator should be activated or deactivated.

This project illustrates an application of the asynchronous `PRPConnection` mechanism from [Recipe 32, Simplify Web Service Connections, on page ?](#). We've tied a download to each row in the table and modified the `PRPConnection` class to use the category methods from this recipe. The network activity indicator shows as soon as downloads begin and automatically hides when the last download is finished or interrupted. The code you see in this class stays the same whether 1 or 100 downloads are in progress.