

Easy NSOperations

@raul_mpad

I WISH I HAD...

...better copying than
pioneering..

I NEED...

Download content...

Store it...

Simultaneously...

The screenshot shows the Instacast app interface. At the top, there's a blue header bar with the text "Instacast" and "RAC1 Podcast:..." followed by a black arrow pointing upwards. Below the header, there's a large icon of a radio tower. To its right, the text "RAC1 Podcast: LA COMPETENCIA" and "Podcasts de RAC1" is displayed, along with a grey right-pointing arrow. The main content area lists several episodes of the show "LA COMPETENCIA" with their respective download progress bars:

- LA COMPETENCIA (16-04-12)**: Downloaded 0.0 MB of 17.3 MB. A red circular button with a white "X" is to the right.
- LA COMPETENCIA (13-04-12)**: Emiss.: 13-04-12 a les 12h. A blue circular icon with a white dot and the word "viernes" is to the left.
- LA COMPETENCIA (12-04-12)**: Emiss.: 12-04-12 a les 12h. A blue circular icon with a white dot and the word "jueves" followed by "52 min" is to the left.
- LA COMPETENCIA (11-04-12)**: Emiss.: 11-04-12 a les 12h. A blue circular icon with a white dot and the word "miércoles" followed by "38 min" is to the left.

At the bottom of the screen, there's a toolbar with several icons: a white X, two stars, three horizontal lines, a downward arrow, an upward arrow, and a download icon.

So, Where do I start?

1. **Google**, of course!!
2. **StackOverflow**, am I a real developer or what??
3. **Apple** reference, the one and only...
4. **NSCodersBCN**, new kid on the block syndrome.

NSOperations & NSOperationQueue

“The NSOperation class is an abstract class you use to encapsulate the code and data associated with a single task”

“The NSOperationQueue class regulates the execution of a set of NSOperation objects”

So, what do I need?

- **NSOperation** to encapsulate and manage the download process.
- **NSOperationQueue** to attach multiple and simultaneous NSOperation objects
- **NSURLConnection** to establish http connections to the content
- Some **xib** representation of the process



Subclassing NSOperation

```
@interface RMPDownloadContentOperation : NSOperation {  
  
    // Cell to upload status  
    NSIndexPath *_indexPath;  
  
    // Current status  
    BOOL _finished;  
    BOOL _executing;  
  
    // Connection  
    NSURLConnection *_conn;  
    NSURL *_url;  
  
    // To save to disk  
    NSString *_filePath;  
    NSOutputStream *_stream;  
  
    // Progress values  
    float _progressValue;  
    long _downloadedContentLength;  
    long _expectedContentLength;  
  
    NSError *_error;  
}  
}  
  
NSError *_error;
```

Completion block:

```
    @interface RMPDownloadContentOperation  
        @property (nonatomic) float progressValue;  
        @property (nonatomic) long expectedContentValue;  
        @property (nonatomic) long downloadedContentLength;  
        @property (nonatomic) NSIndexPath *indexPath;  
        @property (nonatomic) NSURLConnection *conn;  
        @property (nonatomic) NSURL *url;  
        @property (nonatomic) NSString *filePath;  
        @property (nonatomic) NSOutputStream *stream;  
        @property (nonatomic) NSError *error;  
        - (void) initWithUrl:(NSURL *)url filePath:(NSString *)filePath stream:(NSOutputStream *)stream conn:(NSURLConnection *)conn error:(NSError **)error;    
```

Implementation:

```
✓ C @implementation RMPDownloadContentOperation  
P progressValue  
P expectedContentValue  
P downloadedContentLength  
P indexPath  
P conn  
P url  
P filePath  
P stream  
P error  
M -initWithUrl:saveToFilePath:updatingCellAtRow:  
M -start  
M -done  
M -cancelled  
Delegate Methods for NSURLConnection  
M -connection:didReceiveResponse:  
M -connection:didFailWithError:  
M -connection:didReceiveData:  
M -connectionDidFinishLoading:  
M -connection:willCacheResponse:
```

Overriding NSOperation

- **start** method begins the execution of the operation.

```
- (void)start {
    // Ensure this operation is not being restarted and that it has not been cancelled
    if (![NSThread isMainThread]) {
        [self performSelectorOnMainThread:@selector(start)
                               withObject:nil waitUntilDone:NO];
        return;
    }

    if( _finished || [self isCancelled] ) {
        // [self done];
        return;
    }

    // KV0 isExecuting
    [self willChangeValueForKey:@"isExecuting"];
    _executing = YES;
    [self didChangeValueForKey:@"isExecuting"];

    // Create the NSURLConnection
    self.conn = [[NSURLConnection alloc] initWithRequest:[NSURLRequest requestWithURL:self.url
                                                               cachePolicy:NSURLRequestReloadIgnoringCacheData
                                                               timeoutInterval:30.0] delegate:self];
}
```

- **done & cancelled** methods are handmade... and called by...

NSURLConnectionDelegate Protocol

```
C @implementation RMPDownloadContentOperation
P  progressValue
P  expectedContentValue
P  downloadedContentLength
P  indexPath
P  conn
P  url
P  filePath
P  stream
P  error
M -initWithUrl:saveToFilePath:updatingCellAtRow:
M -start
M -done
M -cancelled
Delegate Methods for NSURLConnection
✓ M -connection:didReceiveResponse:
M -connection:didFailWithError:
M -connection:didReceiveData:
M -connectionDidFinishLoading:
M -connection:willCacheResponse:
```

1. Open the **output stream** to save the content of the Response. We save the **content-length** of the Response for later calculations
2. If any error, report and mark as **done** the NSOperation object.
3. **Each time** we receive content from the connection we save it through the **stream** and update the **progress indicator** to the user
4. Whenever the download is done we update the NSOperation status with KVO.

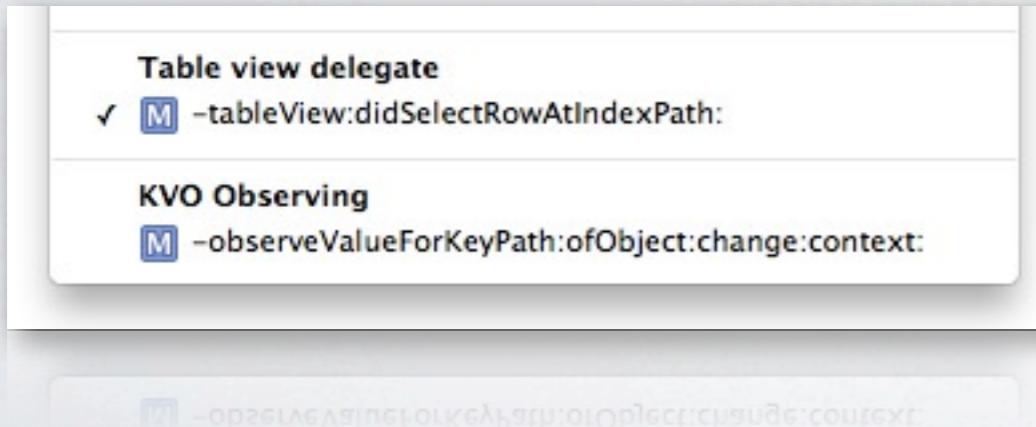
Warning!! Protocol has **change!!**

```
M -connection:willCacheResponse:
M -connectionDidFinishLoading:
M -connection:didReceiveData:
```

NSOperation KVO Pattern implementation

XxxViewController

Add Observers & Observe



NSOperation subclass

Notify changes

```
// KVO isFinished!
[self willChangeValueForKey:@"isExecuting"];
[self willChangeValueForKey:@"isFinished"];
_executing = NO;
_finished = YES;
[self didChangeValueForKey:@"isFinished"];
[self didChangeValueForKey:@"isExecuting"];

// KVO isExecuting
[set didChangeValueForKey:@"isExecuting"];
[set didChangeValueForKey:@"isExecuting"];
```

Using NSOperationQueue

- Alloc and init, as usual...

```
// Create operation queue
self.operationQueue = [NSOperationQueue new];
// set maximum operations possible
[self.operationQueue setMaxConcurrentOperationCount:5];
```

- Queue NSOperations

```
RMPDownloadContentOperation *op = [[RMPDownloadContentOperation
alloc] initWithUrl:[self.urls objectAtIndex:0]
saveToFilePath: filePath
updatingCellAtRow:indexPath];
[self.operationQueue addOperation:op];
```

Progress indication

- User should know...



- And we must implement..

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *DefaultCellIdentifier = @"DefaultCell";
    static NSString *DownCellIdentifier = @"DownCell";

    if ([self.states objectAtIndex:[indexPath row]] == @"0") {
        RMPCustomCell *customCell = [tableView dequeueReusableCellWithIdentifier:DefaultCellIdentifier];
        customCell.lblTitle.text = [self.courses objectAtIndex:[indexPath row]];
        return customCell;
    } else {
        UITableViewCell *downCell = [tableView dequeueReusableCellWithIdentifier:DownCellIdentifier];
        return downCell;
    }
}
```

Demo

GOOGLIGRAPHY

- MUST HAVE: <http://eng.pulse.me/concurrent-downloads-using-nsoperationqueues/>
- NSOperations & GCD: <http://stackoverflow.com/questions/4344884/what-tasks-are-more-suitable-to-nsoperation-than-gcd>

Thank You!

Any easy question?