MSSE 652: iOS Enterprise Software Development

Topic 3: AFNetworking

Agenda

- Resources
- Introduction
- Preliminaries
- AFNetworking

Resources

- Online resources
  - http://afnetworking.com
  - https://github.com/AFNetworking/AFNetworking/wiki/Introduction-to-AFNetworking
  - http://cocoadocs.org/docsets/AFNetworking/2.0.0/
  - https://github.com/AFNetworking/AFNetworking/wiki/Getting-Started-with-AFNetworking
  - http://www.raywenderlich.com/30445/
  - http://www.raywenderlich.com/30445/afnetworking-crash-course
  - http://mobile.tutsplus.com/tutorials/iphone/ios-sdk_afnetworking/

Introduction

- AFNetworking is a “delightful” networking library for iOS and Mac OS X …
  - “…extends the high-level networking abstractions built into Cocoa. It has a modular architecture with well-designed, feature-rich APIs that are a joy to use.”
    - http://afnetworking.com
    - http://cocoadocs.org/docsets/AFNetworking/2.0.0/
  - and provides parsing for the following data formats:
    - XML, JSON, Property Lists

Preliminaries

- Since AFNetworking is a 3rd party product
  - it needs to be downloaded and installed in your Xcode project
- Techniques for downloading AFNetworking
  - as a zip file
  - using "git"
  - using CocoaPods

    Our focus

    - note: Cocoa Pods is an Objective-C dependency manager which simplifies the process of using 3rd-party libraries

First, shutdown Xcode

- If the Xcode is open, close it (i.e., quit the app)
- Note: this step is probably not necessary, but ...
  - the installation of AFNetworking (using CocoaPods) will convert your Xcode <u>project</u> into an Xcode “<u>workspace</u>”
    - and what’s a workspace?
      - a workspace is a collection of projects
  - once the installation of AFNetworking is complete, you should then <u>always</u> open the project as …
    - a workspace and <u>not</u> as a project
      - more on this in a moment

CocoaPods

- CocoaPods is an Objective-C dependency manager
  - i.e., it manages the dependencies between various third party products that your project uses
- Here’s the idea; you …
  - first identify the products in a “Podfile”
  - then run cocoapods to make sure all the products are installed properly
- But first you need to install CocoaPods …

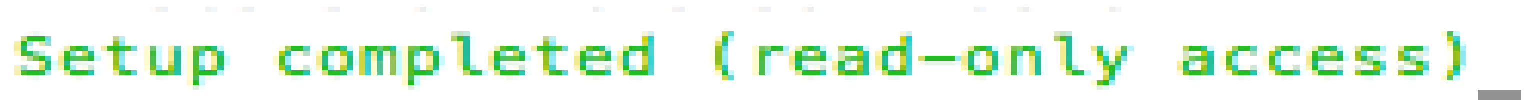
Install CocoaPods

- In a Terminal window,
  - cd (change directory) to the Xcode project folder and then issue the following two commands …

```
sudo gem install cocoapods
pod setup
```

Note: you will be asked for your login password

- Once the setup complete, you should see something like …



```
Setup completed (read-only access)_
```

Create the PodFile

- In the Terminal window, issue the cmds:

```
touch Podfile
open -e Podfile
```

Note: TextEdit launches

- Enter the following lines in the Podfile (via TextEdit)

note: the single quotes can be a problem ☹

```
platform :ios, '7.0'
pod 'AFNetworking', '~> 2.0’
```

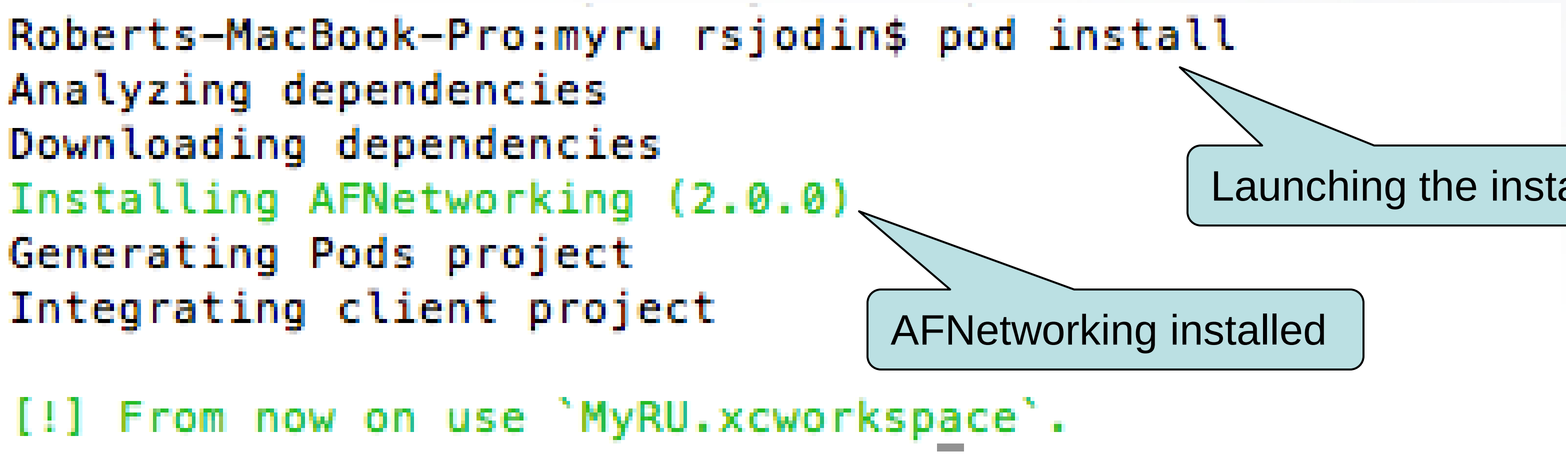
Note: we are using version 2.0;
Most online tutorials are 1.0

- Close the file

Installing AFNetworking

- Then, back in the Terminal Window, do the install

  pod install
- You should see something like the following in the Terminal window (it may take a minute or two)



```
Roberts-MacBook-Pro:myru rsjodin$ pod install
Analyzing dependencies
Downloading dependencies
Installing AFNetworking (2.0.0)
Generating Pods project
Integrating client project

[!] From now on use `MyRU.xcworkspace`.
```

Launching the install

AFNetworking installed

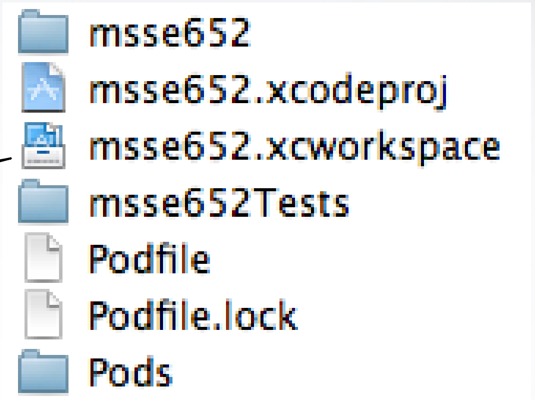
Please see next chart …

Close & reopen (workspace)

- If the Xcode project is open, close it
- Now open the project, but please note:
  - your Xcode project has been converted to a workspace
    - where a workspace contains multiple projects
  - from now on, you need to open the workspace
    - and not the project
- For instance, see next chart …

Open the Workspace

- Inside your Xcode project folder, you now have the following items:



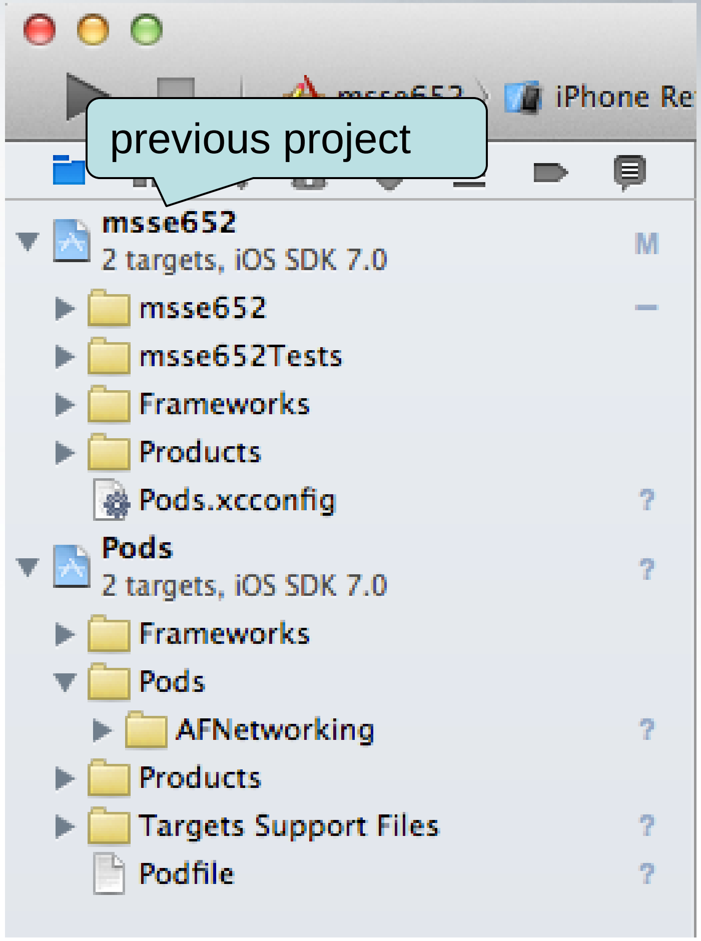
msse652
msse652.xcodeproj
msse652.xcworkspace
msse652Tests
Podfile
Podfile.lock
Pods

*.xcworkspace

- You need to open the project as a <u>workspace</u> …
  - either double click on *.xcworkspace
  - or use the Xcode menu File -> Open
    - and navigate to the *.xcworkspace file, and select it

The Workspace opened in Xcode

- If you opened the workspace, you should see two projects in the Project Navigator:
  - your previous project
  - and a “new” Pods project



```
msse652
2 targets, iOS SDK 7.0                 M
  msse652                              –
  msse652Tests
  Frameworks
  Products
  Pods.xcconfig                        ?
Pods                                   ?
2 targets, iOS SDK 7.0
  Frameworks
  Pods
    AFNetworking                       ?
  Products
  Targets Support Files                ?
  Podfile                              ?
```

Now, check it out

- You should now have visibility to AFNetworking’s header files
  - Verification:
    - open any class
      - e.g., a view controller’s *.m file
    - and insert the following #import statement

      ```
      #import "AFNetworking.h"
      ```
    - there should not be any errors

AFNetworking in a nutshell

- Core Classes
  - AFURLConnectionOperation
  - AFHTTPRequestOperation

- Specialized classes
  - AFJSONRequestOperation
  - AFXMLRequestOperation
  - AFPropertyListRequestOperation
  - AFImageRequestOperation

subclasses of
AFHTTPRequestOperation

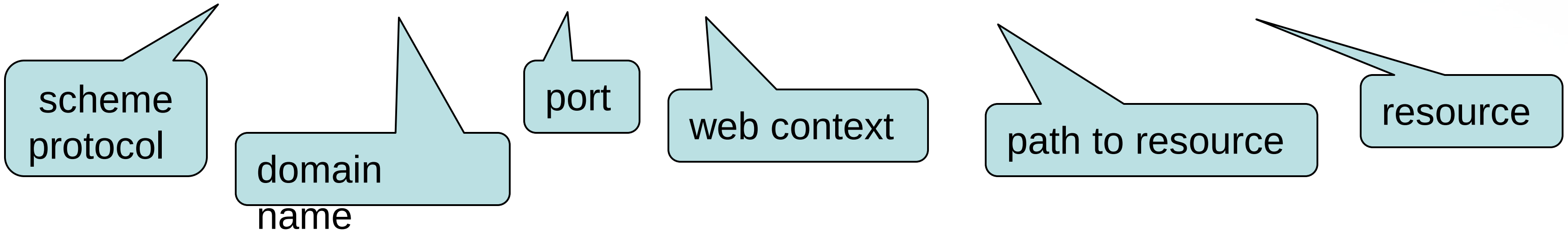
Here’s the basic idea …

- Create an NSRURL containing a URL
- Create an NSRURLRequest object (using the NSURL)
- Create an AFN operation, either …
  - AFJSONRequestOperation
    - if the response should contain JSON formatted data
  - AFXMLRequestOperation
    - if the response should contain XML formatted data
- Start the request operation
- Process the response

Class NSURL

- Class NSURL is used to hold a properly constructed URL
- Provides various methods to initialize the string
  - e.g., initWithString
    - returns nil if the URL is not properly constructed
- Example:

```
NSURL *url = [[NSURL alloc] initWithString:
        @”http://localhost:8080/SCIS/webresources/domain.programs"];
```



Class NSURLRequest

- Class NSURLRequest is used to hold a properly constructed URL request
  - e.g., an HTTP request object
- Often constructed using an NSURL
  - that contains a properly formatted URL
- Example:

```
NSURLRequest *request =
        [[NSURLRequest alloc] initWithURL:url];
```

e.g., the url from the previous page

Creating an AF* Operation

- Once you have an NSURLRequest, you then create either one of the following objects…
  - AFJSONRequestOperation
    - if the response should contain JSON formatted data
  - AFXMLRequestOperation
    - if the response should contain XML formatted data

- We'll illustrate the use of AFJSONRequestOperation …
  - which fetches the JSON data and parses the response

AFJSONRequestOperation

- We'll use the following static method to construct an AFJSONRequestOperation

```
[AFJSONRequestOperation
  JSONRequestOperationWithRequest:request
   success:^(NSURLRequest *request,
                 NSHTTPURLResponse *response, id JSON) {
                     NSLog(@"%@", JSON);}
   failure:^(NSURLRequest *request,
               NSHTTPURLResponse *response,
               NSError *error, id JSON) {
               NSLog(@"Request FAILED: %@, %@", error,
                          error.userInfo);}];
```

The NSURLRequest

runs if the request succeeded

success block

failure block

runs if the request failed

All together now …

```
NSURL *url = [[NSURL alloc] initWithString:
            @”http://localhost:8080/SCIS/webresources/domain.programs"];
NSURLRequest *request = [[NSURLRequest alloc] initWithURL:url];
AFJSONRequestOperation *operation = [AFJSONRequestOperation
    JSONRequestOperationWithRequest:request
    success:^(NSURLRequest *request,
                    NSHTTPURLResponse *response, id JSON) {
                          NSLog(@"%@", JSON);}
    failure:^(NSURLRequest *request,
                NSHTTPURLResponse *response,
                NSError *error, id JSON) {
                NSLog(@”Request FAILED: %@, %@", error, error.userInfo);}];
[operation start];
```

For testing

For testing

The success block

- Is executed in the “main event loop” ☺
  - so you can update the UI with the returned data

```
success:^(NSURLRequest *request,
                NSHTTPURLResponse *response, id JSON) {
                    // update a table data source and then reload the table
                    [self.mytableview reloadData];
}
```

The failure block

- Is executed in the “main event loop” ☺
  - so you can notify the user with an alert popup:

```
failure:^(NSURLRequest *request,
          NSHTTPURLResponse *response, NSError *error, id JSON) {
          UIAlertView *av = [[UIAlertView alloc]
              initWithTitle:@"Error with request”
              message:[NSString stringWithFormat:@"%@",error]
              delegate:nil
              cancelButtonTitle:@"OK" otherButtonTitles:nil];
          [av show];
}];
```

What’s the big deal?

- Notice, unlike NSURLConnection (topic 2), …
  - with AFNetworking you did not have to use GCD and place the tasks in an event queue
- Why is that?
  - the AFNetworking RequestOperation does that for you under the hood
    - so you don’t have to worry about it