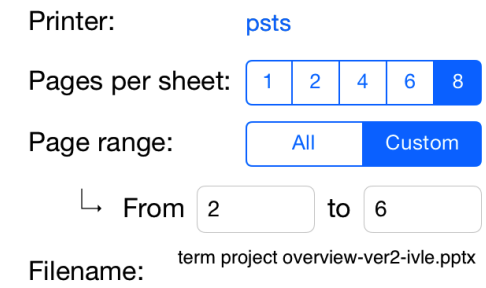
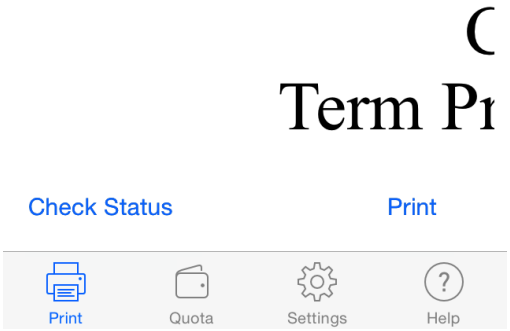
NUS SOC Print User/Dev Guide (iOS)





Yeo Kheng Meng

Sole developer

Credits

NUS SoC Print (iOS) will not be possible without the aid of the following people:

1. Kai Yao and Yong Quan for the initial codes and design.
2. Lenny for the app icon.
3. Zit Seng's help on Sunfire and advice on NUS's intellectual property issues
4. My CS3217 Prof Khe Chai for the initial guidance on publishing iOS apps.
5. Vishnu's advice on iOS App submissions.

Key Dependencies

1. NMSSH SSH library (https://github.com/Lejdborg/NMSSH)
2. Identify device type (https://github.com/erica/uidevice-extension)
3. Docs to PDF converter (https://github.com/yeokm1/docs-to-pdf-converter)
4. nup_pdf PDF formatter (http://blog.rubypdf.com/2007/08/24/how-to-make-n-up-pdf-with-free-software/)
5. Multivalent PDF formatter (http://multivalent.sourceforge.net/Tools/pdf/Impose.html)

NUS SOC Print (iOS) User Manual

Introduction

NUS SOC Print (NSP) is a mobile application that enables students from National University of Singapore (NUS) School of Computing to print Microsoft Office and PDF documents to the school's UNIX printers. You can use other apps like IVLE or your web browser to pass supported files to this app.

Supported File Types

PDF, DOC, DOCX, PPTX and ODT. Although NSP supports many document formats, you are advised to print with PDF whenever possible to give the best result.

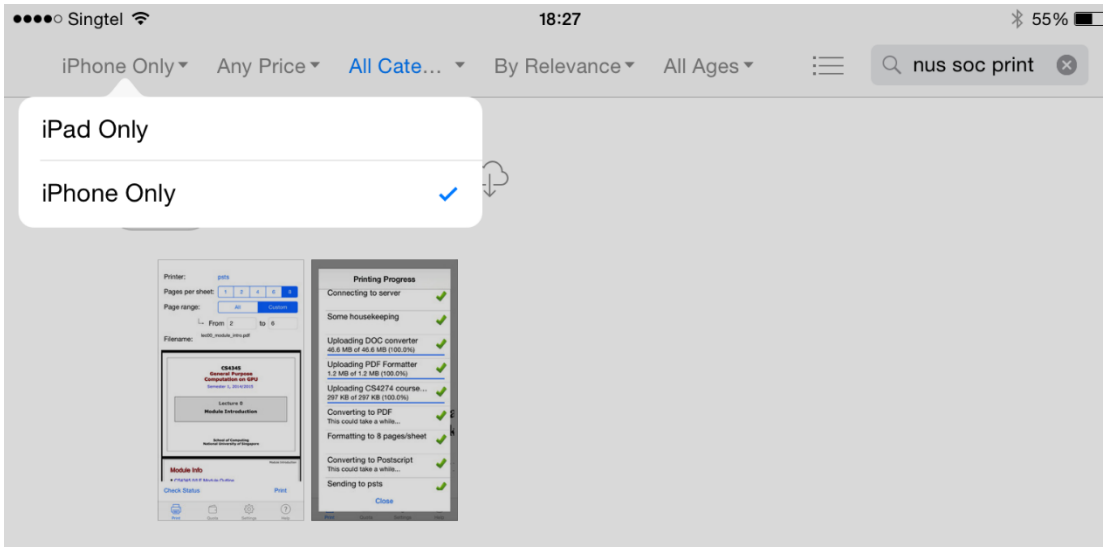
System Requirements

- iOS 7.0 or later
- All iOS devices supported including iPhone, iPad and iPod Touch
- About 60MB of disk space

Installation instructions

Just open the App Store application on your iOS device and do a search for "NUS SOC Print". My app should appear as the first result.

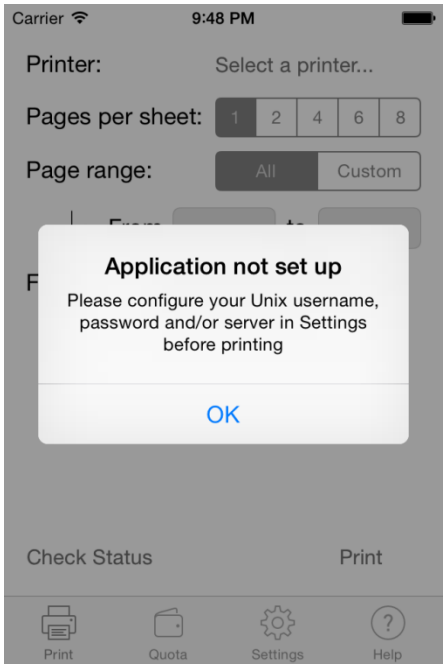
For iPad users, you have to set the search parameters to "iPhone Only" as seen below.



Running NSP

You had to open the App Store app in the first place to download my app don't you? Don't tell me you do not know how to start my app for the first time?

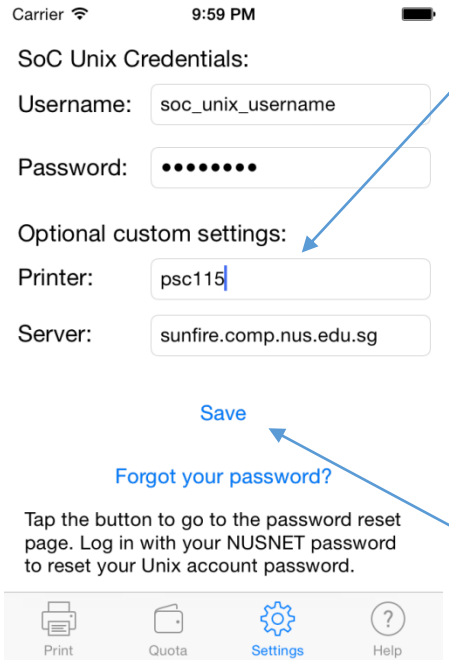
NSP Initial Load Screen



On initial start, you will be prompted to give NSP your SoC Unix's credentials. Note that this is **NOT** your regular NUSNET one. These are the credentials you use to login into Sunfire.

After you dismiss this popup, head over to the Settings area by selecting the Settings tab at the bottom.

NSP Settings Page



(SettingsViewController.swift)

Just type your username and password into the fields.

The printer field is to enable NSP to use a printer that has not been set by me in the app.

Default Printers: psts, pstsb, pstsc, psc008, psc011, psc245 and their single-page counterparts.

For example, there exists a printer psc115 in Embedded Systems Lab in SoC and I wish to use it but NSP does not have it by default. Simply enter psc115 into the field and then you can use it on the main screen.

The server field is to adjust where the SSH connection should be made to. Just leave the field as-is for normal operation.

Remember to hit the **Save** button once you are done.

Using NSP

NSP cannot be used as-is. You are required to use another app like IVLE or Dropbox to pass the document you wish to print to NSP.

As Sunfire has a whitelist of IP addresses, you may not be able to use NSP outside the school although major ISPs are supported. Your mileage may vary.

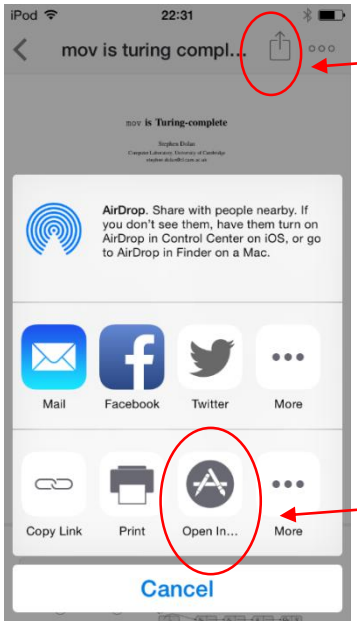
Transferring files to NSP

I recommend viewing this demo video https://www.youtube.com/watch?v=PRGcK7gzbnM as it provides a clearer picture on how to execute this procedure.

Otherwise:

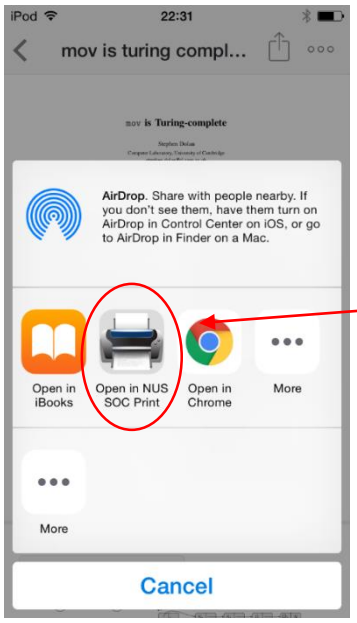
Step 1: Open Dropbox, IVLE or any other app that has the ability to open files in another app. I will use Dropbox in this example.

Step 2: Open the file that you wish to print.



Step 3: Press the export button

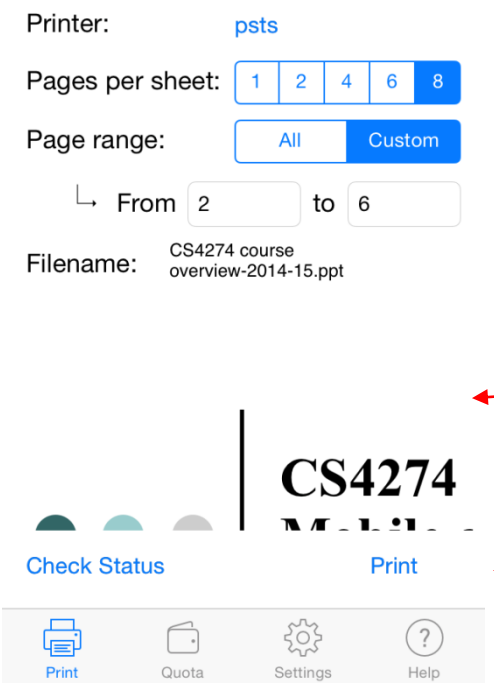
Step 4: Press the “Open In…” button



Step 5: Choose to open the file with NSP

Main Print Screen

Screen here should be pretty self-explanatory if you have used the print dialogs on modern desktop operating systems.



Page range text box selectable only if you set the page range as "custom". There is no function to omit certain pages in between the range

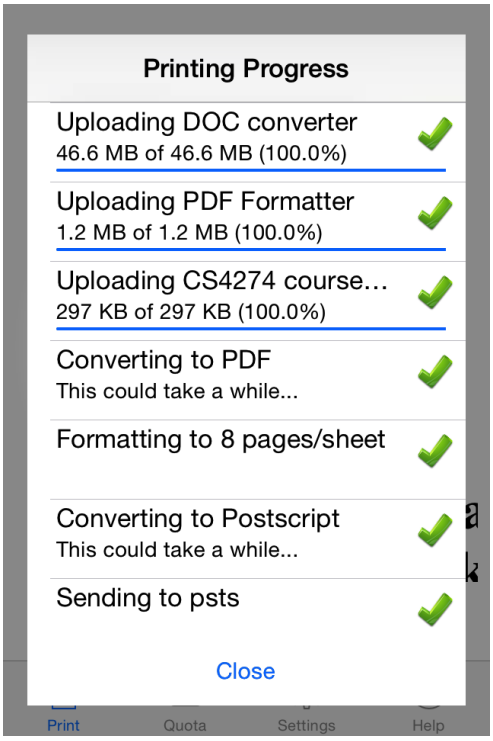
File preview area. Note that this previews only the original file as-is and **NOT** the final printed output.

These two buttons will open a new window which are explained in the next section.

(PrintViewController.swift)

Printing Progress

After you hit "Print" in the main screen, this window will open.



Just wait for the print progress to proceed through the required steps.

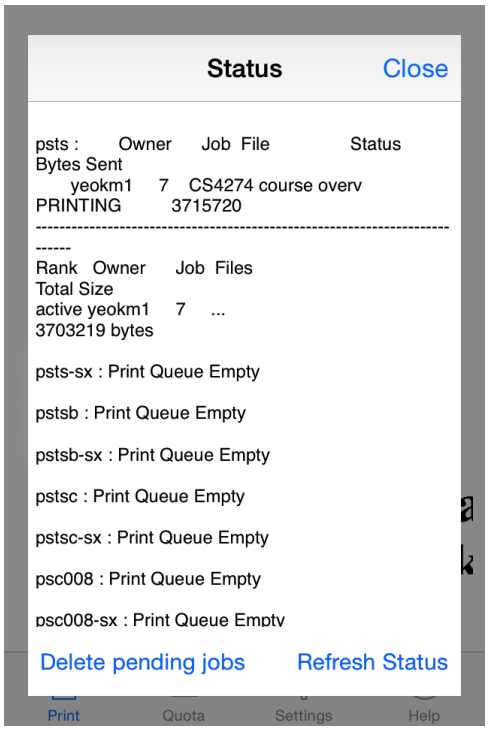
NSP requires certain converters programs to be present on Sunfire before it can operate on your files. These converters will be uploaded and cached on Sunfire the first time your print your files. For subsequent print jobs, these uploads will no longer be necessary.

If you are on a mobile instead of a Wifi connection and the upload of the 50MB document converter is required, you will be prompted if you want to continue. This is to protect your data plan.

Be patient as sometimes the print procedure may take a few minutes.

(PrintingViewController.swift)

Check Print Job Status



This screen enables to view the progress or delete sexisting print jobs sent by you on NSP simply by hitting the buttons at the bottom.

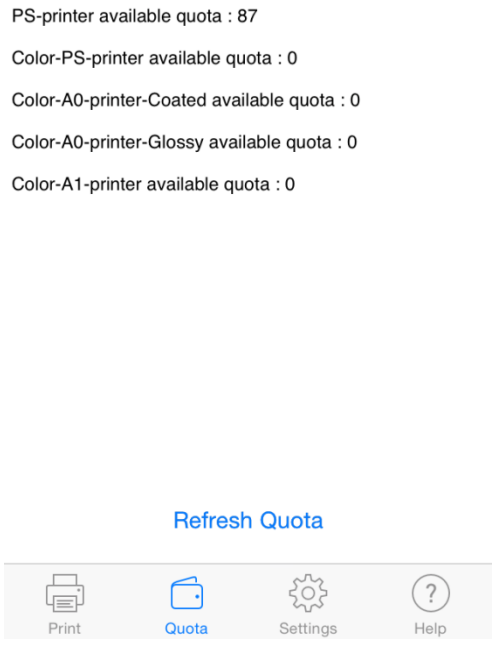
Note that you cannot delete print jobs sent via NUSNET/Samba even though it is associated with your Unix account.

Printer error messages such as “Out of Paper” may also be shown.

(StatusViewController.swift)

Check Quota

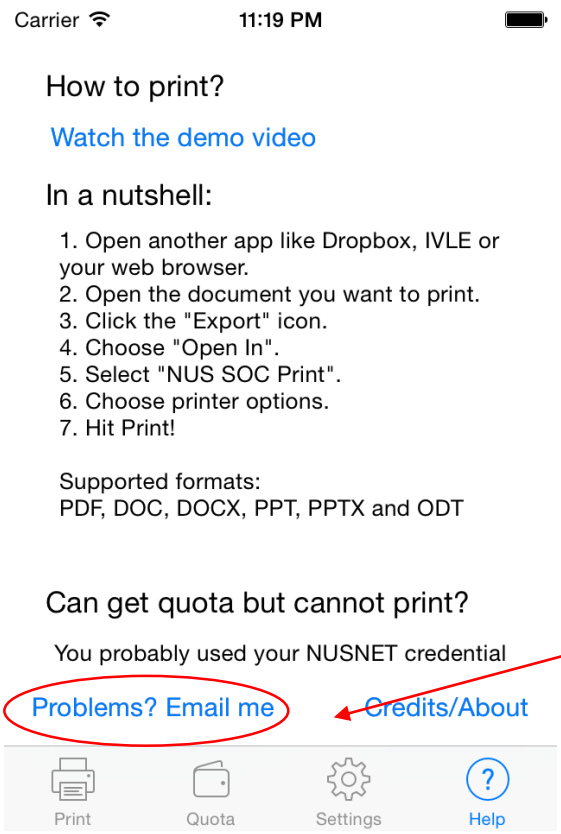
Screen to check for your quota. Hit refresh however many times you like.



(QuotaViewController.swift)

Seeking Help

Found a bug in NSP or have a suggestion? I want to hear from you! Go to the help tab.



Hit this button to open your email client.

(HelpViewController.swift)

Your email client should open and you may see something like this:

I have embedded your device details into the subject for debugging purposes however you can choose to delete it if wish to protect your privacy.

Just type whatever you want into the contents and send.

P/S by Kheng Meng: As I have graduated already, I cannot support this app anymore. This section is just an idealised version for a future maintainer of my app if any.

NUS SOC Print (iOS) Developer Guide

Introduction

This developer guide is for the iOS version of the NUS SOC Print(NSP) mobile app. After reading this guide, you will get an understanding of the architecture of this app and the stuff you may need to look out for that may not be easily derived from the code.

This guide attempts to follow the standard set in CS2103. It should not be treated however, as the actual standards of CS2103, for eg, this does not have the Contents Page and Testing sections because well I'm lazy to do them.

Prerequisites

Equipment

1. Mac OS X Yosemite 10.10.4 with at least Xcode 7.0 (Swift 2)
2. Any modern iOS hardware with iOS >=7.0 preferred although you can start with a simulator

Knowledge

1. Knowledge in Swift and basic iOS programming/APIs is required
2. Some Objective-C knowledge can be helpful as not all the code is in Swift
3. Reading my blog post on this app is highly recommended as well

    http://yeokhengmeng.com/2014/12/nus-soc-print-androidios-background-technical-aspects-and-learning-points/

If you have taken CS3217 as of AY14/15 Sem 2 or later, you should have more than enough knowledge to understand this guide.

I'm too impatient, give me the code now!

Satisfy the prerequisites and itchy to start diving in before reading rest of the guide? For the impatient folks, here you go!

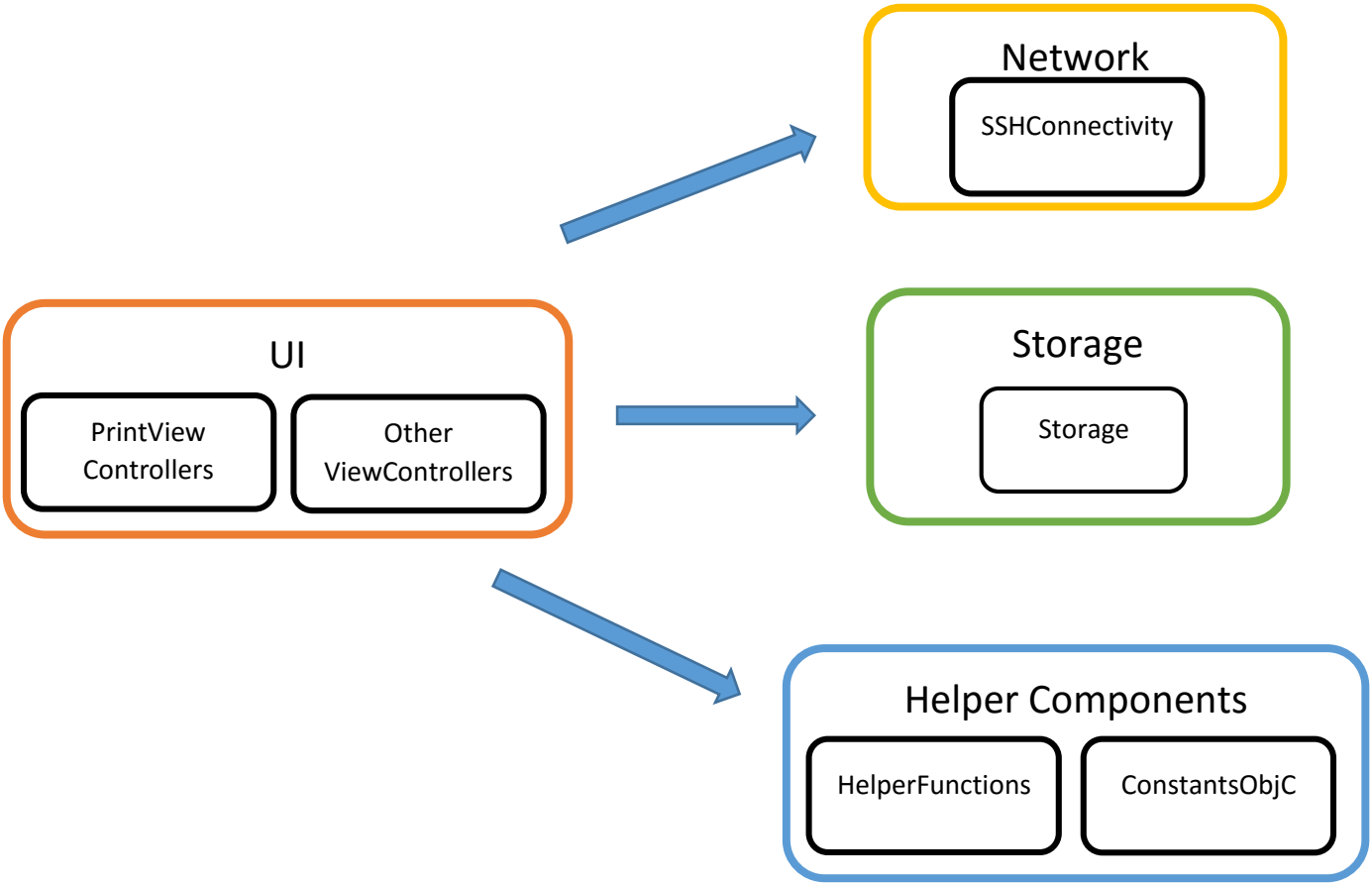
1. Open your terminal and navigate to where you wish to save the project
2. *git clone https://github.com/yeokm1/nus-soc-print-ios.git*
3. Install Cocoapods (iOS Package Manager)if you have not done so:

    sudo gem install cocoapods

4. *cd nus-soc-print-ios*
5. *pod install*
6. Open the project using *NUS SOC Print.xcworkspace* instead of the *.xcodeproject* file as Cocoapods modifies the Xcode project slightly
7. You may need to adjust the team name to deploy on your personal device. Go to Project Settings, General, NUS SOC Print Target. Choose None or your team name from the drop down menu.

Architecture

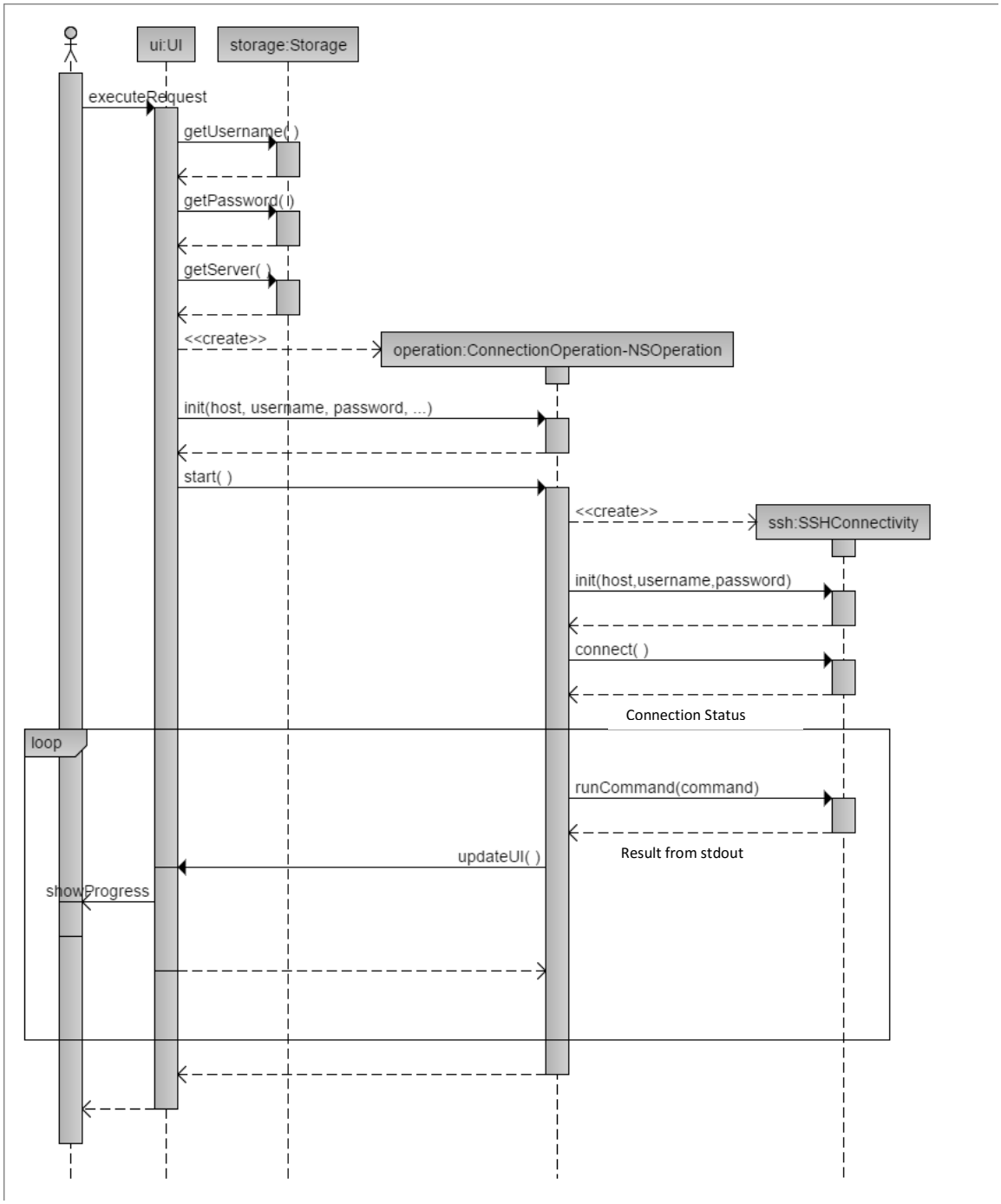
The components in NSP can be roughly classified into the following areas. As this app is very small, the classification here is just a theoretical separation and do not correspond to any groupings within Xcode.



I try to follow a top-down design as far as possible. However, mobile apps are usually heavily UI-driven where program logic usually has a tight integration with the UI. In this case, the UI (usually an extension of UIViewController (iOS API)) is at the top and uses the facilities as provided by the other classes as needed.

Command flow

The sequence diagram below shows the generic process in which the components of my application of NSP. The other areas of my app involve a direct user interaction to execution hence will be easily understood without a need of a diagram.



The ConnectionOperation (extension of NSOperation) is a generic name for an inner class that is placed inside the UI that wishes to conduct SSH operations. This class is placed inside the UI as there is usually a tight coupling between the sequence of network operations and the progress that is shown to the user.

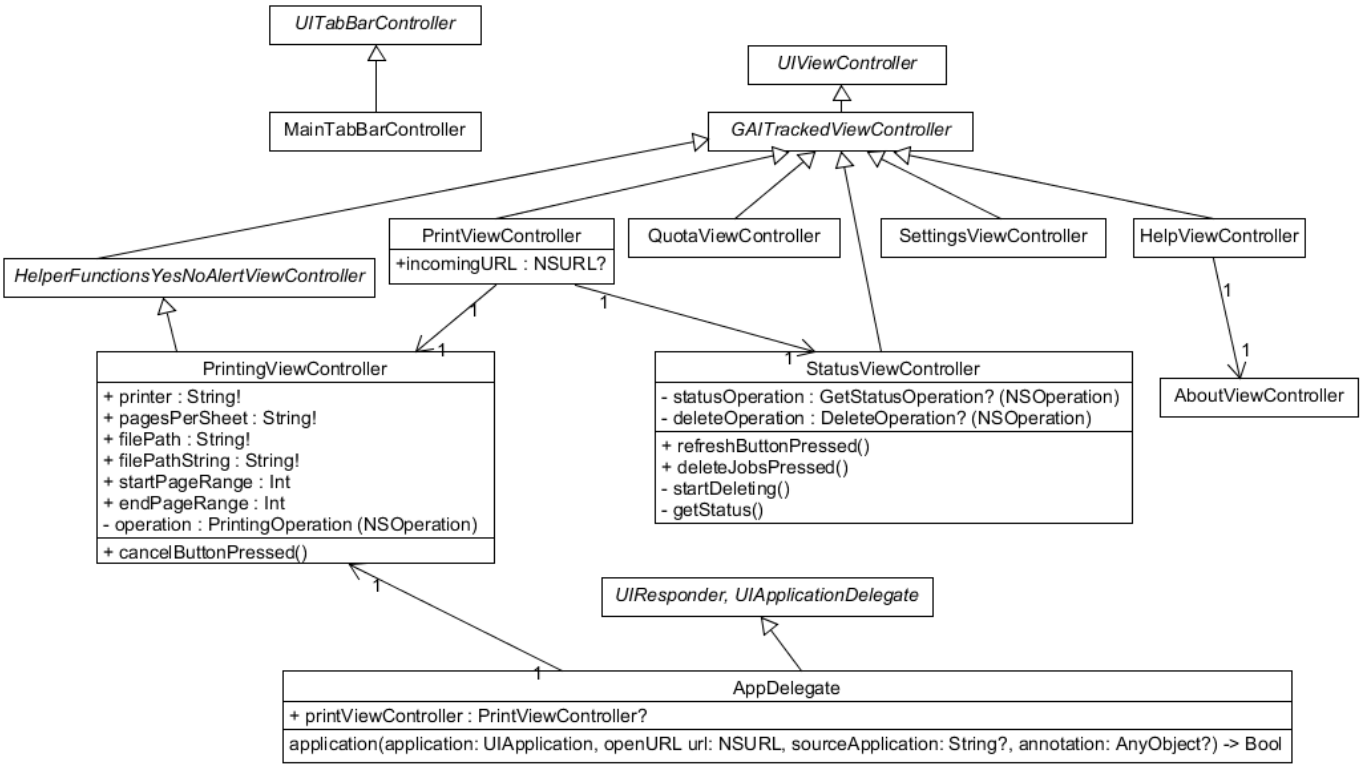
Networking SSH operations in ConnectionOperation are conducted on another thread to prevent blocking the main thread.

Components

You have seen the general architecture of the application and how the components generally work together. Now let's dive into the components starting with the UI.

UI

If you have used the app or have read the user guide, you would have seen the multiple tabs at the bottom of the application as well as the Printing and Status screens. All the screen designs are stored in the sole storyboard file "Main.storyboard". From here on, I will shorten ViewController to VC.



A class diagram to show the relationships between the classes in the UI component and important methods.

Each screen corresponds to a class that is an extension of GAITrackedVC. GAITrackedViewController (which is an extension of UIVC) is part of Google Analytics that helps to analyse how my users are using each screen.

The public methods of PrintingVC and StatusVC are for the benefit of the iOS UI framework to call when the user presses the buttons in the interface, they are not to be called for any other purpose.

MainTabBarController

Under typical circumstances for most iOS apps that use the Tab Bar, a developer need not implement the UITabBarController.

When the user launches the PrintingVC or StatusVC, I wanted to give the window effect where the background is dimmed. This effect is broken in iOS8. Go to http://stackoverflow.com/a/16230701 for more information. The workaround I have implemented require to be loaded before the rest the VCs are loaded which is therefore done in MainTabBarController.

AppDelegate

This class is a standard for all iOS applications and defines the entry point of the application. The critical method to note here is

```
func application(application: UIApplication, openURL url: NSURL, sourceApplication:
String?, annotation: AnyObject?) -> Bool
```

This method is called when the user imports the file into NSP from another app. After the file is received, the PrintVC is given the file path of the incoming file to preview when it comes into view.

If PrintVC has yet to be initialised say after the phone is freshly rebooted, AppDelegate will pass the file path to the global variable “incomingURL” in PrintVC.

PrintViewController

This class contains the backing code for the printing options selection screen and a preview of the imported file. It will launch the PrintingVC and StatusVC depending on the user’s selection.

PrintingViewController

This class is the beef of the application. Based on information handed over via the PrintVC, it will initialise the SSHConnectivity object via an instance of NSOperation. The SSH commands used in this class and other considerations can be obtained from my blog post at early part of this guide.

If the 50MB document to PDF converter is required to be uploaded, this class will check if the user is on a mobile connection or Wifi. If on mobile, request if the user is ok to proceed.

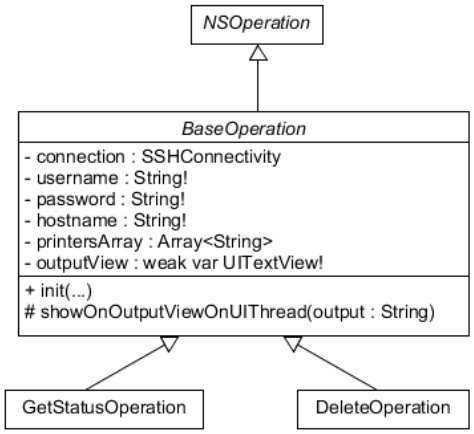
Determining if the device is on mobile or Wifi necessitates the use of the Objective-C Reachability library (https://developer.apple.com/library/ios/samplecode/Reachability/Introduction/Intro.html). The result obtained is an Objective-C enum. Due to limitations of handling Objective-C enums from Swift, I have wrapped the call inside the function “`+(bool) isOn3G;`” `in` ConstantsObjC.

To ask the user, I need to use an AlertView. However, Apple has changed how an AlertView is supposed to be created. In iOS7, a UIAlertView is used with the caller requiring to implement the UIAlertViewDelegate to receive the results of the button press. In iOS8 and later, a UIAlertController is used requiring the caller to pass a Block that will be run when the user has pressed the button.

To reconcile the API differences, the HelperFunctions class has abstracted the creating of the AlertView though the function showYesNoAlert().

StatusViewController

To show the current print jobs of all the printers or to allow the user to delete the current print jobs. Both operations require certain SSH network commands to be called which necessitates a separate thread like NSOperation.



As both status and deletion commands require the same set of connection procedures, I have abstracted the similar code as an abstract BaseOperation class. The decision to launch which operation is dependent on the button press of the user.

QuotaViewController

QuotaVC shows the remaining print quota. This is accomplished by issuing a POST request to login into https://mysoc.nus.edu.sg/images/LOGIN and retrieving the output at "/~eprint/forms/quota.php".

A regular expression is used to parse the raw HTML output to obtain the desired values to be shown to the user.

SettingsViewController

Nothing really special about this. Just take the input and pass to the Storage object to be saved to the system.

HelpViewController

To get users started on how to use the app. If there are issues they can email me. To aid debugging, I inject the device's details into the email title field. This information is obtained via an Objective-C library https://github.com/erica/uidevice-extension .

AboutViewController

Just to show details of the application and credits. I can't seem to retrieve the Build date via Swift APIs. So I had to wrap the Objective-C API via the class ConstantsObjC with the method

"+(NSString *) getBuildDate;".

Network- SSHConnectivity

This class is responsible for interacting with the remote server through the NMSSH library. The methods in this class are blocking so you have to call its methods from a separate thread to prevent blocking your UI thread.

Relevant methods

Return Type(s)	Method name and parameters
void	**init(hostname : String, username : String, password : String)** The constructor of the class with server connection parameters
serverFound: Bool authorised: Bool	**connect()** Connect to the server via SSH with parameters provided in the constructor. Returns a tuple to indicate the results of the connection attempt. If called with existing active connection, will disconnect the previous connection before connecting.
void	**disconnect()** Disconnects active SSH connection. Safe to call even if there is no active connection.
String	**runCommand(command : String)** Runs command on remote server and returns the (stdout) result of the command
Bool	**uploadFile(sourcePath : String, destinationPath : String, progress: ((UInt) -> Bool))** Uploads file via SFTP to the server. **sourcePath** refers to the filepath to be uploaded. **destinationPath** is with respect to the remote user's home directory. **progress** is a Block that will be handed directly over to NMSSH. At quick regular intervals, NMSSH will give the total number of bytes that has been uploaded so far. The Block is then required to return True or False to indicate if the upload should continue to proceed.

Storage - Storage

This is a singleton class that deals with getting/storing the user's connection credentials and other information to the local storage by using NSUserDefaults.

The entire printer list is also hardcoded into this class for the PrintVC to load when ready.

Supporting files

NSP carried a payload of 3 files nup_pdf.jar, Multivalent.jar and docs-to-pdf-convert-1.7.jar. These files will be uploaded to the server to aid in the file conversion process.

Stuff currently tied to my (Kheng Meng's) accounts

If you wish to take over the app, I may have to transfer ownership of certain accounts that are currently used by NSP.

1. Itunes Connect (App Store account)
2. Google Analytics

Library Management

My project currently uses Cocoapods as its library management system. Consult the Cocoapods website https://cocoapods.org/ on how to use this system. Use it instead of copying code snippets as far as possible.

The relevant files are Podfile and Podfile.lock. You may update the Podfiles once you are certain the new library versions does not cause any regressions.

Known Issues

1. The event portion of Google Analytics in PrintingVC is currently commented out although the App Store version has the working code.
   This event code is to collate data on the printing options of NSP's users. Google updated the Analytics package and changed the APIs and I cannot translate the Objective-C code examples/documentation to Swift at this time.
2. The support email address in Help is currently tied to my personal email that needs to be changed too.
3. I just did a massive code upgrade to Swift 2 so there may be unknown regressions.
4. Google has not updated their analytics library to use bitcode. I have temporarily disable bitcode in the project settings. Remember to flip the setting back once Google has made the update. Details here: http://stackoverflow.com/questions/31395260/google-analytics-libadidaccess-a-does-not-contain-bitcode

Future Work

1. Use iOS8 extensions
2. Live Preview of print options by downloading the formatted PDF from Sunfire before actually printing
3. Allow user to quickly set username/password during the initial launch of NSP without going to the Settings Page
4. Short tutorial during initial launch
5. Add Crashlytics or any other debugging libraries
6. Multiple copies
7. Better UI in general
8. Improve print speed/reliability (Postscript conversion)

Appendix

1. The Sequence Diagram is generated with the help from this site http://www.ckwnc.com/
   The code to generate the diagram is from the file ios-ui-sequence-diagram.txt
2. The Class Diagram is generated by this Java Program UMLet which can be downloaded from http://www.umlet.com/
   The file is ios-ui-class-diagram.uxf