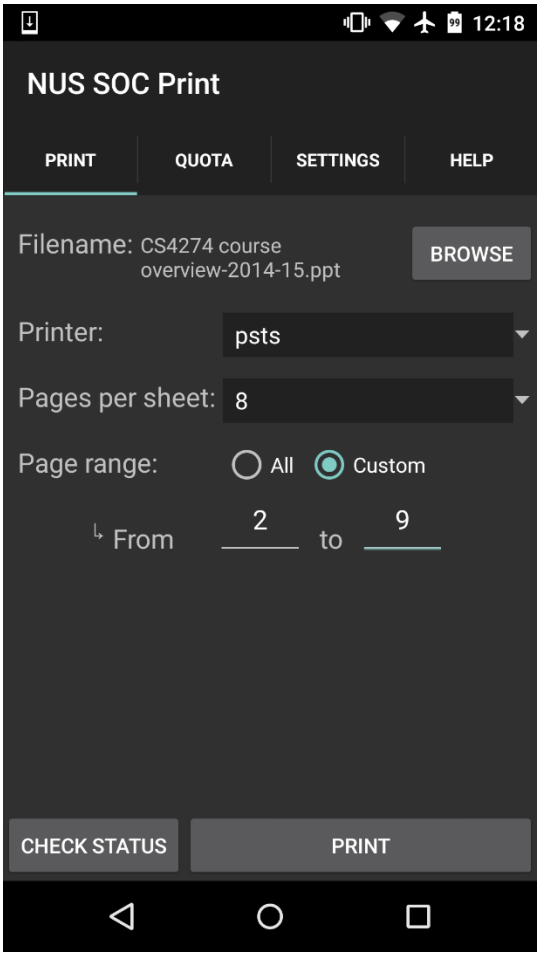
NUS SOC Print User/Dev Guide (Android)



Yeo Kheng Meng

Sole developer

Credits

NUS SoC Print (Android) 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

Key Dependencies

1. Jsch SSH library (http://www.jcraft.com/jsch/)
2. aFileChooser file browser library (https://github.com/iPaulPro/aFileChooser)
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)
6. PreferenceListFragment(https://github.com/artiomchi/AndroidExtensions/blob/master/AndroidExtensions/src/main/java/org/flexlabs/androidextensions/preference/PreferenceListFragment.java)
7. Craft Support Email Intent(https://github.com/yeokm1/craft-support-email-intent)

NUS SOC Print (Android) 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 or open directly from the file system.

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

- Android 2.1 or later
- Should be majority of Android devices
- About 13MB of disk space

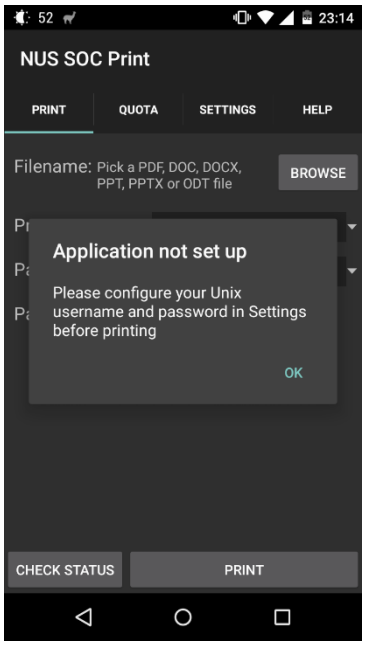
Installation instructions

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

Running NSP

You had to open the Play 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?

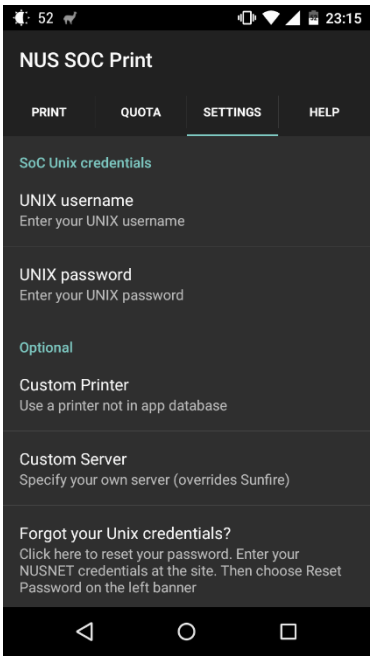
NSP Initial Load Screen



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



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.

(SettingsFragment.java, fragment_settings.xml)

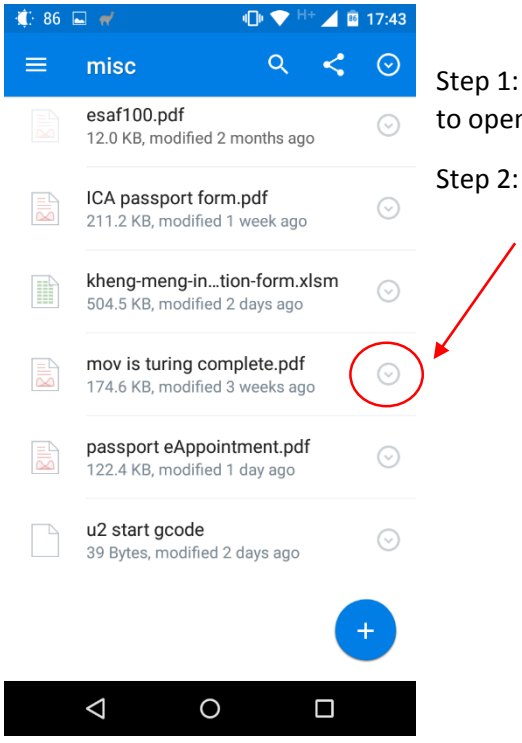
Using NSP

There are import a file into NSP

1. Share the file to NSP from another app.
2. Use NSP to open a file already saved to the file system

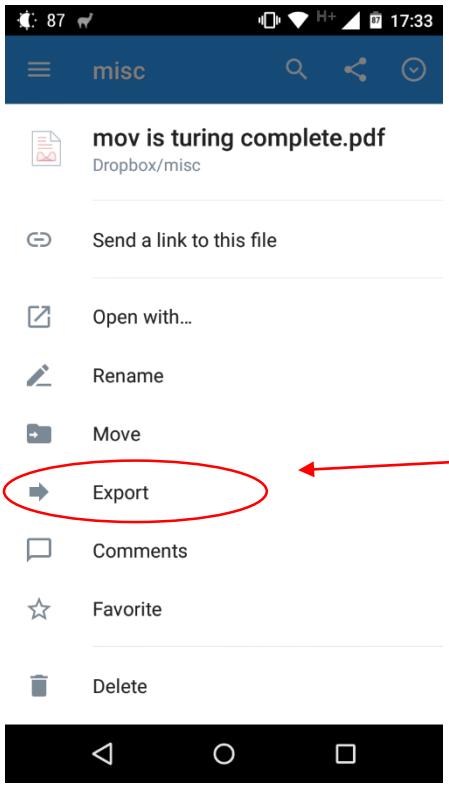
As Sunfire has a whitelist of IP addresses, you may not be able to use NSP outside the school although major ISPs are supported. Your mileage may vary.

Share the file from another app

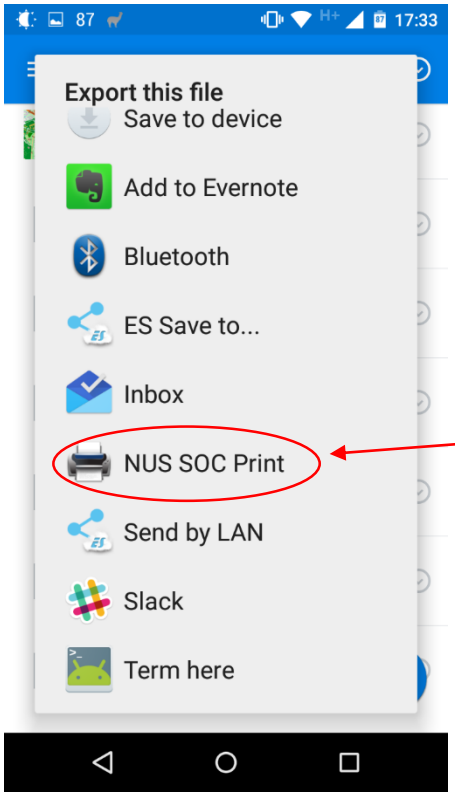


Step 1: Open Dropbox, IVLE or any other app that has the ability to open files in another app. I will use Dropbox in this example.

Step 2: Click the Properties icon of the file you wish to print



Step 3: Press the export button



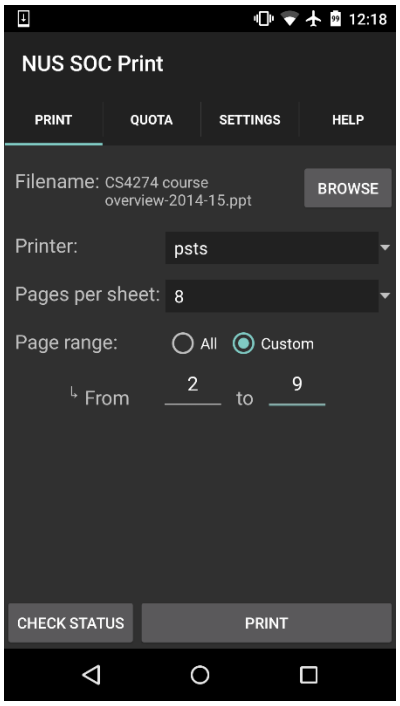
Step 4: Choose to export the file to NSP

Open existing file from file system

Just click the “Browse” button the main screen and select the file via the integrated file browser.

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

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

(PrintFragment.java, fragment_print.xml)

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.

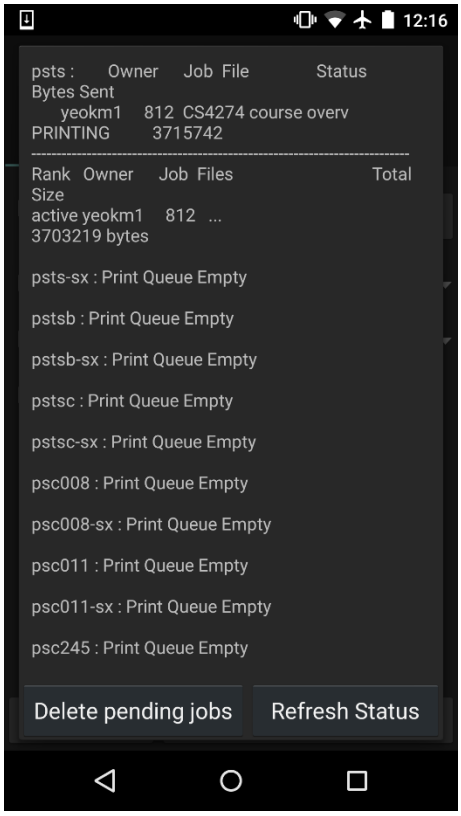
NSP requires certain converter programs to be present on Sunfire before it can operate on your files. These converters will be uploaded and cached on Sunfire the first time your print your files. For subsequent print jobs, these uploads will no longer be necessary.

If you are on a mobile instead of a Wifi connection and the upload of the document converter is required, you will be prompted if you want to continue.

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

(PrintingActivity.java, printing_progress_view.xml)

Check Print Job Status



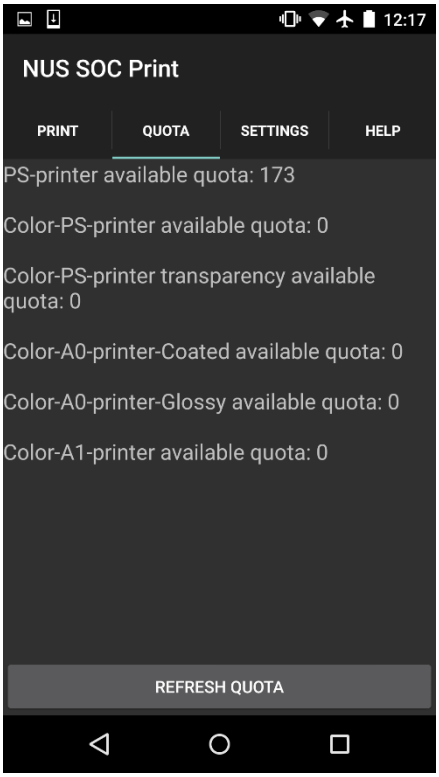
This screen enables to view the progress or delete or existing print jobs sent by you on NSP simply by hitting the buttons at the bottom.

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.

(StatusActivity.java, activity_status.xml)

Check Quota

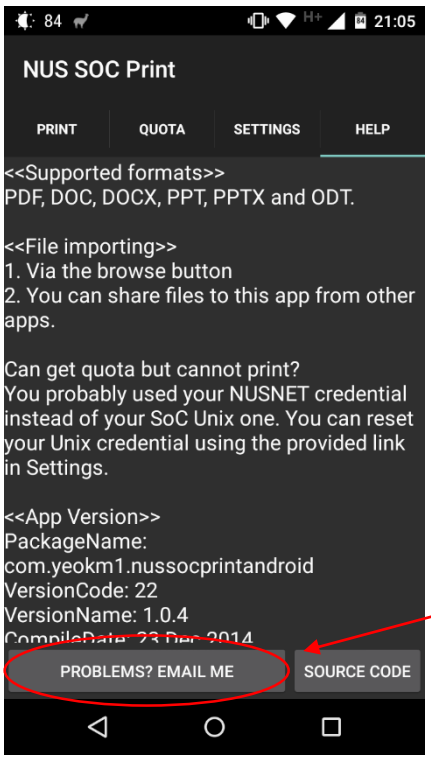


Screen to check for your quota. Hit refresh however many times you like.

(QuotaFragment.java, fragment_quota.xml)

Seeking Help

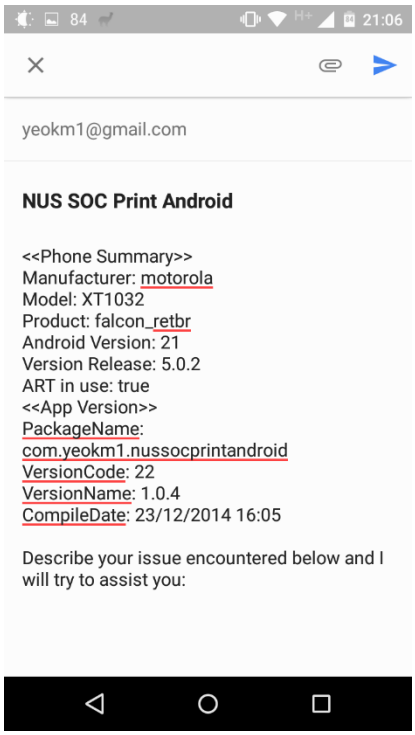
Found a bug in NSP or have a suggestion? I want to hear from you! Go to the help tab.



Hit this button to open your email client.

(HelpFragment.java, fragment_help.xml)

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



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

Just type whatever you want into the bottom of the device details.

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 (Android) Developer Guide

Introduction

This developer guide is for the Android 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. Any modern OS with at least Android Studio 1.3.2 installed
2. JDK 7.0 or higher
3. Any Android hardware with OS >= 2.1 although you can start with an emulator

Knowledge

1. Knowledge in Java and basic Android programming/APIs is required
2. 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/

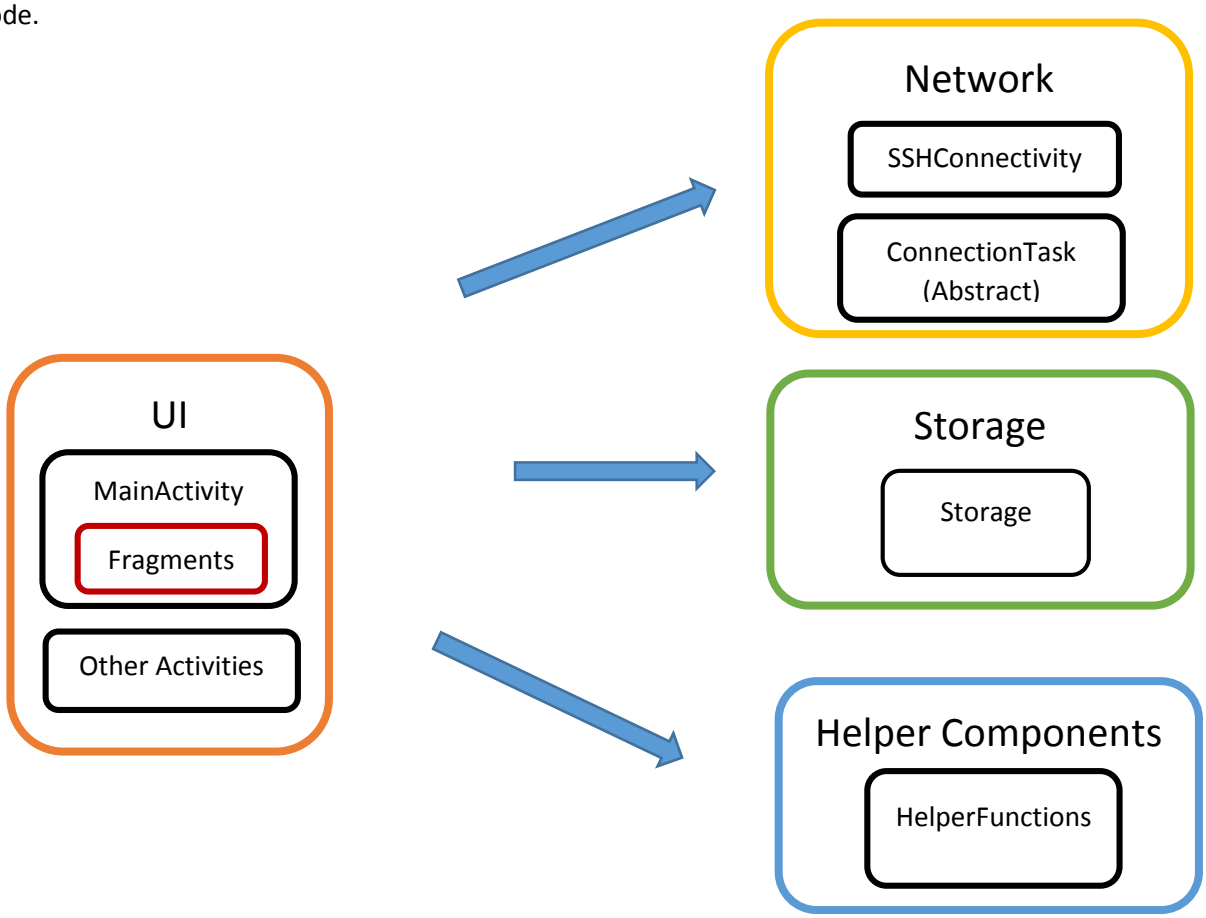
I'm too impatient, give me the code now!

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.git*
3. Open Android Studio
4. Point the directory to open to where you pulled NSP
5. Give Android Studio some time to download the necessary dependencies then you are good to go!

Architecture

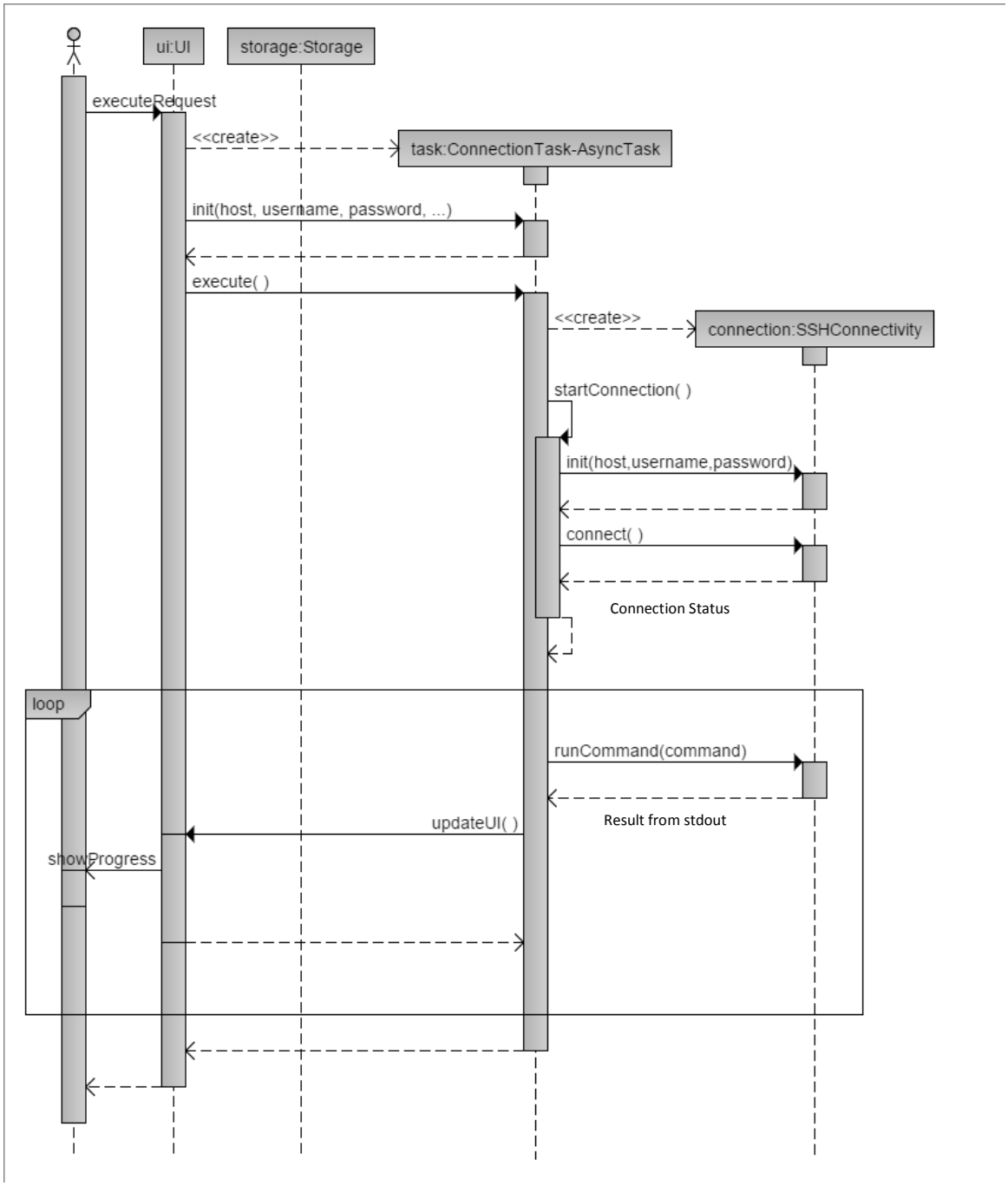
The components in NSP can be roughly classified into the following areas. As this app is very small, the classification here is just a theoretical separation and do not correspond to any groupings within Xcode.



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 Activity/Fragment (Android 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 ConnectionTask (extension of AsyncTask) is an abstract class. Another class should be extended from ConnectionTask as an inner class 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.

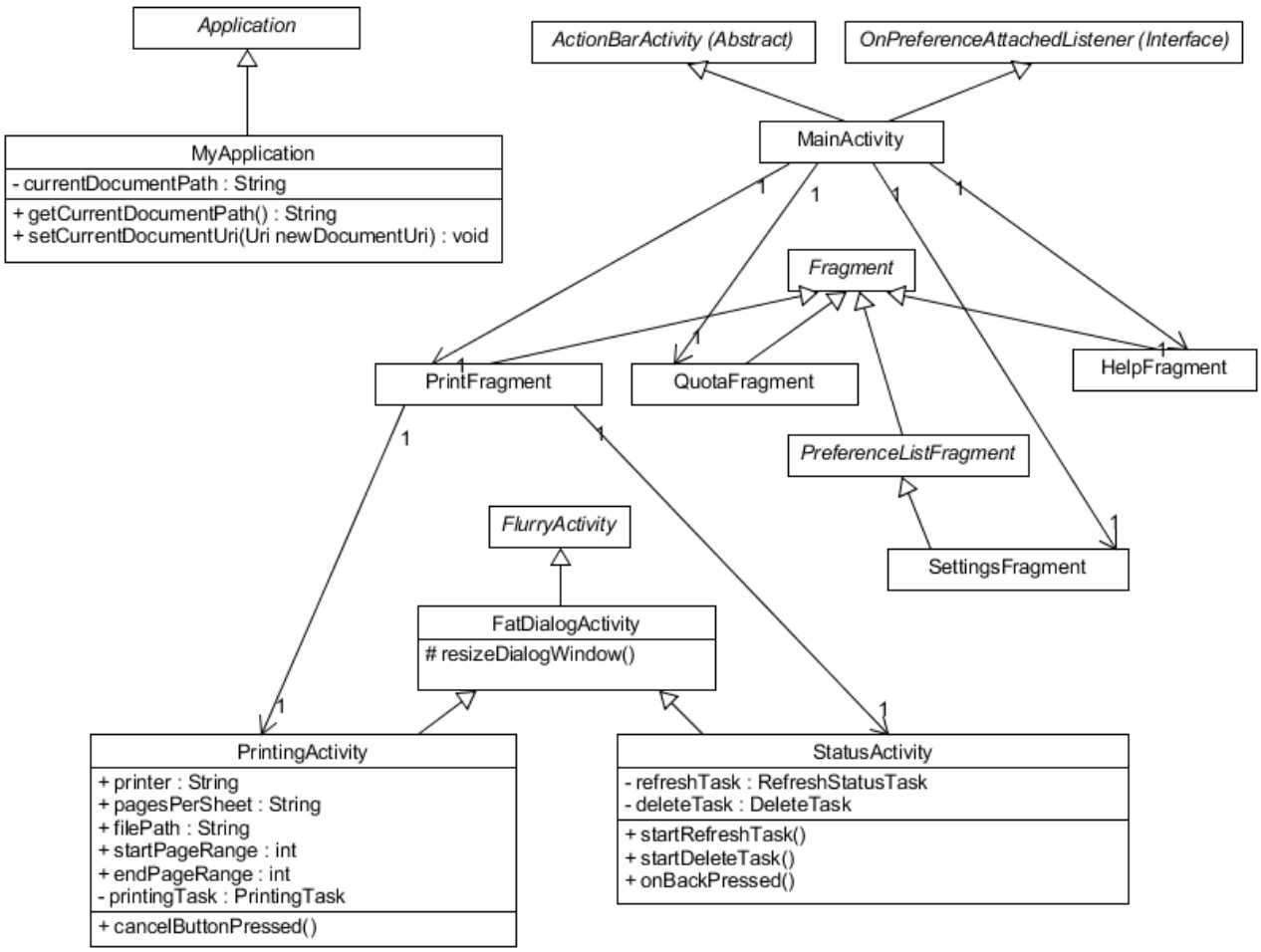
Networking SSH operations in ConnectionTask are conducted on another thread to prevent blocking the main thread.

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 located in the res/layout directory which is the standard for Android apps. Reflected in the class diagram below are the UI classes.



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

Each Activity extends FlurryActivity which is itself an extension of the Android Activity class. FlurryActivity is part of Flurry Analytics that helps to analyse how my users are using each screen.

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

MainActivity

The MainActivity is the starter activity that is required for all typical Android applications. This Activity hosts the ActionBar at the top and its associated Tab fragments. Whenever a user changes Tabs, the MainActivity will launch a new instance of the Fragment associated with that Tab.

MainActivity is also responsible for receiving the file path when the user imports the file into NSP from another app. The file path is then handed over to MyApplication to be kept until another file path comes in. (See MyApplication section)

MainActivity also implements the "OnPreferenceAttachedListener" interface of only one method:

```
public void onPreferenceAttached(PreferenceScreen root, int xmlId)
```

This method is a dummy method that is deliberately left empty for the benefit of the SettingsFragment. The use of this interface will be explained in the SettingsFragment section.

MyApplication

Under typical circumstances for most Android apps, a developer need not override the default Application object with their own implementation.

However, I need a way to preserve the path of the last file path. I cannot leave the file path in MainActivity as that Activity can be terminated when NSP enters the background and thus lose the reference to all local variables.

MyApplication will however remain indefinitely and help keep the state of the file path variable.

PrintFragment

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

FatDialogActivity

An abstract Activity class for Activities(PrintingActivity and StatusActivity) that require a window mode. Any Activity that wishes to enter a window mode just has to subclass this and call "resizeDialogWindow()" immediately after "setContentView(R.layout.activity_layout);" in "onCreate()".

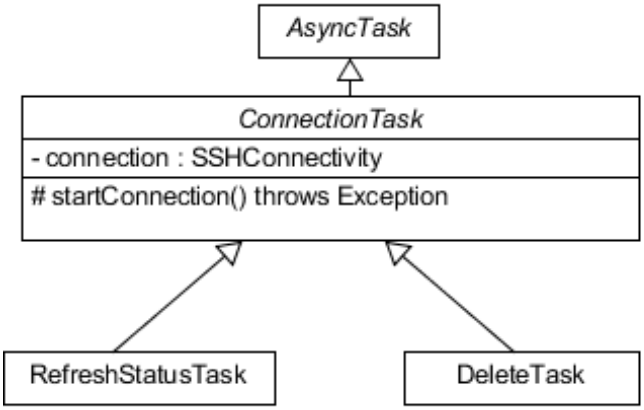
The reason I called this fat is because by default Windowed mode makes the Activity rather skinny thus wasting a lot of space on the vertical edges. "resiveDialogWindow()" adjusts the window width to 98% of the screen width.

PrintingActivity

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

StatusActivity

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 AsyncTask.



As interaction with the SSHConnectivity class has a lot of boilerplate code, I have abstracted interaction with that object via a ConnectionTask class that includes many helper functions. The decision to launch which task is dependent on the button press of the user.

QuotaFragment

QuotaFragment 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.

SettingsFragment

For convenience, one would typically use PreferenceFragment(Android API) for screens that primarily deal with saving data to the Shared Preferences. This screen is no exception. However, this API only exists from API Level 11 (HoneyComb, 3.0) and above.

As I intend to support as low as API Level 7 (Éclair, 2.1), I used a third party (reverse-engineered from Android AOSP) library known as PreferenceListFragment. Look up the details here http://blog.fordemobile.com/2012/11/display-preference-fragment-compatible.html

HelpFragment.

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 contents area of the email.

The code to retrieve the device's details actually comes from a library I have written.

https://github.com/yeokm1/craft-support-email-intent

Network- SSHConnectivity

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

Relevant methods

Return Type(s)	Method name and parameters
void	**SSHConnectivity(String hostname, String username, String password, Context context)** The constructor of the class with server connection parameters
void	**connect() throws Exception** Connect to the server via SSH with parameters provided in the constructor. Throws Exception with error message if any issue is encountered.
void	**disconnect()** Disconnects active SSH connection. Safe to call even if there is no active connection.
String	**runCommand(String command) throws Exception** Runs command on remote server and returns the (stdout) result of the command
void	**uploadFile(InputStream toBePrinted, String directory, String filename, SftpProgressMonitor progressMonitor) throws SftpException, JSchException, IOException** Uploads file via SFTP to the server. **directory** refers to the directory of the file to be uploaded with respect to the remote user's home directory. **filename** is with final filename of the file on the remote serer **progressMonitor** is a callback (interface) object that one is required to implement to receive file upload progress. This is handed to the Jsch library. They key method to implement is "*public boolean count(long count)",* where count is the number of bytes transferred since the last invocation of this method. Return true to continue uploading and false otherwise.

Storage - Storage

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

The entire printer list can be obtained from this class via the helper method *"List<String> getPrinterList()"*. This method returns an array of the hardcoded printers stored in strings.xml and the user's custom printer at the top.

Supporting files

NSP carried a payload of 2 files nup_pdf.jar and Multivalent.jar. These files will be uploaded to the server to aid in the file conversion process. docs-to-pdf-convert-1.7.jar however will be download from my Sunfire account at

http://www.comp.nus.edu.sg/~yeokm1/nus-soc-print-tools/docs-to-pdf-converter-1.7.jar

with backup at

https://github.com/yeokm1/docs-to-pdf-converter/releases/download/v1.7/docs-to-pdf-converter-1.7.jar

once my Sunfire Unix account is terminated. This is due to APK size limitations of an Android app.

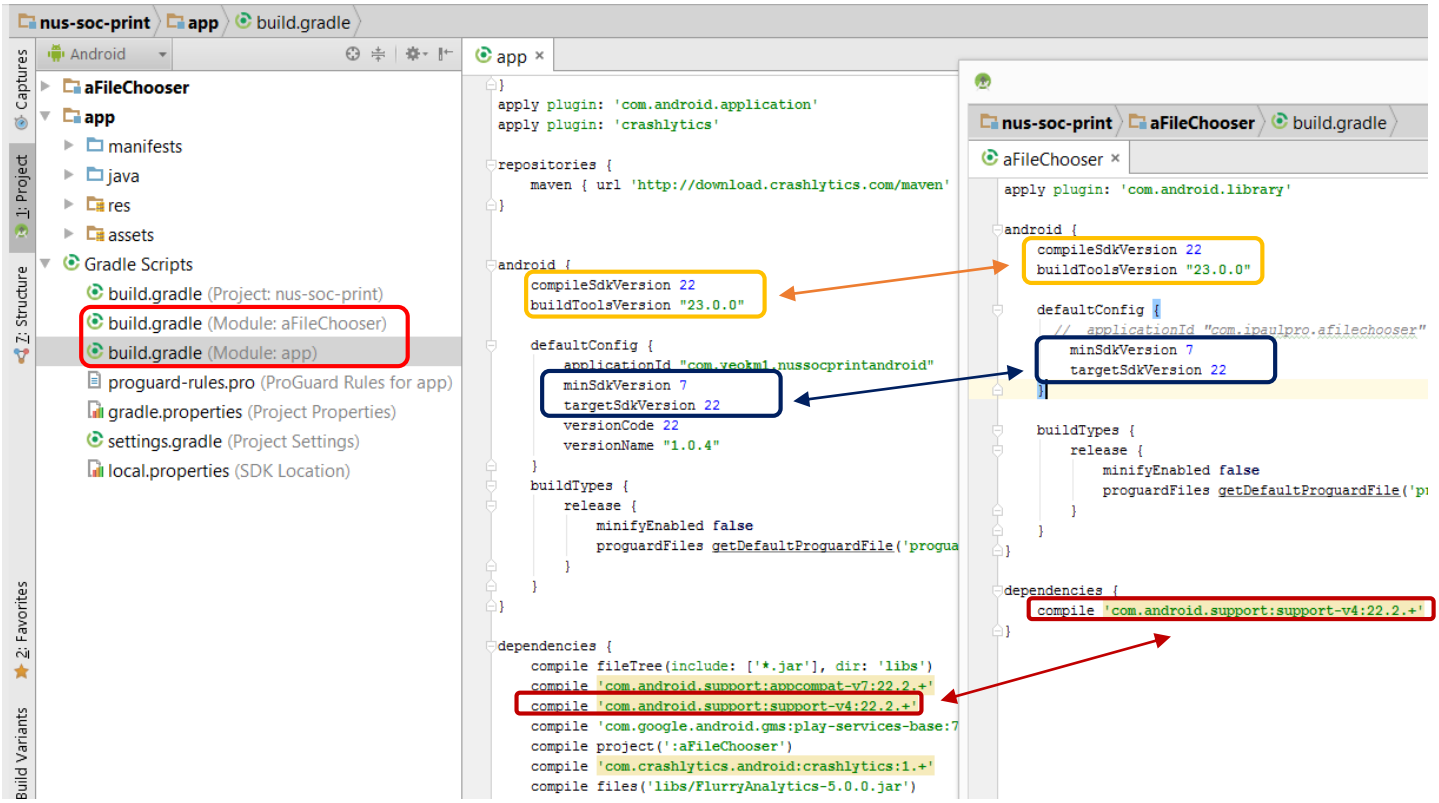
Stuff currently tied to my (Kheng Meng's) accounts

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

1. Google Developer's Console (Play Store account)
2. Flurry Analytics
3. Crashlytics

Target SDK/Build Tools changes

It is inevitable that you have to change the target SDK and/or Build Tools as time goes by to ensure trouble-free running on newer Android OSes. When you do so, remember to update and keep consistent the details in build.gradle for both Module:aFilechoose and Module:app as seen below.



Ensure the related enclosed sections are kept identical.

Known Issues

1. For the next developer to take over this project, it is IMPERATIVE that you relocate the path of “docs-to-pdf-convert-1.7.jar” to your own public folder in your Unix account at the earliest opportunity. My Unix account is set to expire on December 2015. The backup Github link will work but is very slowww….
2. The support email address in Help is currently tied to my personal email that needs to be changed too.
3. In SettingsFragment, the virtual keyboard does dismiss itself automatically once the user has set a particular field. I noticed this issue only ever since I raised the Target SDK to Android 5.0. I cannot find the cause of this issue at this time.
4. The entire source code for the Jsch library is dumped into my project. This is because the official jar library has missing classes which may cause my app to crash on phones with Android Runtime (ART) enabled. You may need to check whether the new versions of Jsch still have this issue.

Future Work

1. Use Android 4.4 Printing APIs
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. Multiple copies
6. Better UI in general
7. Improve print speed/reliability (Postscript conversion)
8. Choice of flip with long/short edge

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 android-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 android-ui-class-diagram.uxf