

1. Executive Summary

In our modern day, Short Message Service (SMS) is an important means of communication. According to Wikipedia, more than 7 trillion SMS messages were sent in 2011. To prevent SMS spamming, service providers have placed a maximum limit of 10 recipients on any given message. As the technology evolved, the ability to send SMS messages through a web service provider has made sending messages programmatically more efficient. However, using a web service provider makes receiving responses awkward, at best. To compensate, this project attempts to create a client side Android application that would automate sending an SMS message, to more than 10 recipients, through a smart phone's SMS application. The major advantages of this application are automation for the sender, to more than 10 recipients. In addition, this application provides the recipient, the ability to easily respond, directly to the sender's phone.

The Android application is built on the Android API version 2.3.3 or API 10, because this API services nearly 80% of current devices on the market. The application uses an Open Source API called JExcel, to connect to an Excel document and obtain data from the first two columns, of the first sheet of a workbook stored on the device's "/data/" folder. It then runs through a loop to send an SMS message to each recipient listed in the Excel document. Overview of the system begins with a simple splash screen asks the user for a file name and a message. The message is limited to a 160 characters, and anything over is truncated. After supplying the file name and message, the user has two options, either send the message or exit the program. Once the send message button is pressed, the program begins to parse the Excel data, store it in an array, then call the SMS application to send each message individually.

2. Implementation Documentation

The modularity of this application design allowed the independent construction of each component. The different components include:

1. Retrieve the list of recipients
2. Send an SMS message for each recipient

The initial proposal was to retrieve the recipient list through a Google spreadsheet. However, this proved to be more problematic than anticipated. It involves:

1. Negotiating the authentication dialogue with the Google servers
2. Receiving an authorization token
3. Retrieving a list of available workbooks
4. Retrieving the specified workbook
5. Retrieving the first worksheet of the specified workbook
6. Parsing the received data

The initial implementation involved a Google API called Gdata that provided functionality to implement all six steps. The solution found in "fetchSpreadsheet.java" was built using this API. However, when porting the solution into the Android environment, it was discovered that the API was made inaccessible by Google, in its most recently released version, rendering the solution useless.

Additional attempts were made to find a solution to authenticate with Google servers including: using an a third party API from com.prasanta, implementing a new class to manually negotiate the authentication process, and using Webview to build a web application using GWT for authentication. For future versions, Webview using GWT should be used to negotiate the authentication token because it is capable of handling the redirects and the CATCHUPTA In the end, the Google authentication process was deemed too complex for the scope of this project, and an alternative means of retrieving a list of recipients was devised.


The solution chosen was to retrieve the data an Excel spreadsheet located locally on the device. The application uses JExcel, a third party API, to read, parse, and process a local Excel document, to retrieve the recipient list. JExcel can read data from the following Excel formats: '95, '97, 2000, XP, and 2003. Therefore, the user must provide an Excel document in the correct format. This can be done through many different means, including:

1. Emailing the Excel file to an email account registered on the Android device. Using this method, the user would download and save the attachment to the "/data/" folder on the device.
2. Using an SD card to transfer the file to the Android device.
3. Using Bluetooth to transfer the file to the Android device.
4. Using the web browser to download the file from an ftp/http server

Furthermore, to reduce application overhead, data validation should be done within the Excel document. The data validation rules should include the following:

1. One column for the names of each recipient, located in Column A of Sheet 1.
2. One column for phone numbers of each recipient, located in Column B of Sheet 1.
3. No other column or sheet should be populated.
4. Each row represents a unique record, where the phone number corresponds to the recipient name on any given row.
5. The phone number format may include dashes, parenthesis, and spaces.
6. It should be in the format "(XXX) XXX-XXXX", "XXXXXXXXXX", "XXX-XXX-XXXX", or "XXX XXX-XXXX".

Here's a screenshot of the sample file used in testing this application:



	A	B
1	Name	PhoneNumber
2	John Kim	5558
3	Jessmine Kim	5556
4	Josiah Kim	5556
5	Samatha Kim	5558
6	Jane Kim	5558

(The numbers in this example refer to reference numbers identifying Android emulators. Substitute real phone numbers for those values.)

The exhibit illustrates the methods defined for this application:

```

public void onCreate(Bundle savedInstanceState) {}
protected void onStop() {}
protected void onPause() {}
private OnClickListener startApp = new OnClickListener() {}
private void sendSMSList(){}
private void sendSMS(String phoneNumber, String message){}
private void initializeExcelContent(){}
private String getFileContent(String filename){}
private OnClickListener stopApp = new OnClickListener() {}

```

1. onCreate(Bundle), onStop(), and onPause() are Android methods that manage the different states of an Android app. The onCreate() inflates the UI to display the following:



It also links the buttons with appropriate methods. onStop() and onPause() releases the application from memory to free precious resources. Ideally, onPause() should write data pertinent to save the state of the application, and remember where it left off. onStop() and onPause() are essentially the same and they are called when the app is stopped prematurely or when it is paused temporarily. In either case, it would be beneficial to save state data, so that when the application is called again, it can pick up where it left off.

2. OnClickListener startApp() and OnClickListener stopApp() are Android methods that capture button click events, corresponding to the “Go” and the “Stop” button. startApp() calls the remaining four methods, to direct the app flow. The application processes the document and

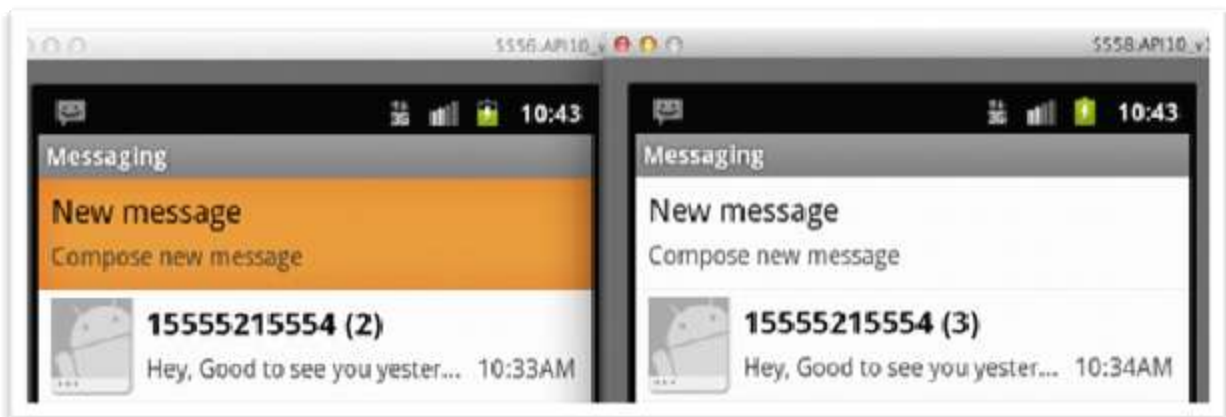
sends SMS in a linear fashion, but parallel activities could be used to improve performance. This would be most beneficial when reading the document, getting the authentication token, or downloading the spreadsheet (in future versions of the app). Because of the simple structure of the app, performance the cost of implementing parallel execution seemed to outweigh the benefits. The stopApp() calls the onStop() method, and releases the application from memory.

3. The first method OnClickListener startApp() calls is the getExcelContent(). getExcelContent() creates an instance of the JExcel-workbook object, from the file in the “/data/[filename.xls]” folder. It then creates a spreadsheet instance of the JExcel-sheets object, from the first sheet of the previously obtained workbook. It then iterates through the cells in column’s A and B, and stores them in a multidimensional array. Here’s the LogCat file:

D	04-25	10:28:46.998	2689	com.johnkim	dalvikvm	GL_EXPLICIT freed 521
D	04-25	10:28:47.017	2689	com.johnkim	ReadFileActi...	Name PhoneNumber
D	04-25	10:28:52.187	2689	com.johnkim	ReadFileActi...	John Kim 5558
D	04-25	10:28:57.766	2689	com.johnkim	ReadFileActi...	Jessmine Kim 5556
D	04-25	10:29:03.036	2689	com.johnkim	ReadFileActi...	Josiah Kim 5556
D	04-25	10:29:04.126	2689	com.johnkim	ReadFileActi...	Samatha Kim 5558
D	04-25	10:29:05.186	2689	com.johnkim	ReadFileActi...	Jane Kim 5558

An improvement that will be implemented in a future version is a status update TextView, in the UI, that updates with the status of the file being read. This field would display the record being read as the method parses through the file.

4. The second method OnClickListener startApp() calls, is the sendSMSList(). sendSMSList() then iterates through the arrays and calls the last method, sendSMS(number, message) for each phone number in the array. Here are the other two emulators showing the messages received from the emulator running the application:



An improvement that will be implemented in a future version is a broadcast intent/receiver model to check if the SMS has been sent properly. This method would also modify the status update TextView, to show which record has

3. Discussion of Learning and Conceptual Difficulties Encountered

This project had an extremely steep learning curve. I not only had to refresh myself on Java concepts, but I had to also learn how to program for the Android environment, how to execute apache server calls, how the OAuth 2.0 authentication process works, and the beginnings of developing a cross platform webapp. I spent more than 70 hours on the project, but it has been a rewarding and fun experience.

The greatest difficulty I had was in accessing Google spreadsheets. In the end I was unable to incorporate this feature into my application. However, I feel comfortable enough to work on a future implementation using WebView, GWT, HTML5, etc. I'm not sure why Google discontinued support for the OAuth 2.0 protocol and the spreadsheets API in client side applications, but doing so made it a lot more difficult for beginning developers to use their services. There are two things I learned through this issue. First, under a deadline, account authentication is too complex to implement a custom solution. Furthermore, the authentication protocols and Google procedures change so quickly that applications based on a custom API would need flexibility or adaptability to be robust. Webapps have the advantage of this flexibility. For example, changes in the authentication process on Google's side of the negotiations, are easily accommodated through webapp's ability to display interfaces (webpages) designed by Google. This leads me to my second learning point, Web Toolkit and webapps are an important part of mobile development, perhaps more significant than proprietary Android SDK.

Outside of Google spreadsheet integration, I would have liked to implement greater functionality to the application more: error control, input validation, ability to scan the inbox for recipient data, etc.

4. Assistance

I tried to meet with Professor Little, but due to the hectic nature of the closing semester, I wasn't able to sit down with him. He did point me in the general direction, and he suggested a book I could use.

I did a lot of reading from the books in the Safari Tech database, as well several online resources: developer.android.com, stackoverflow.com, thenewboston.org, etc.