PsySlug

Abstract:

In fall quarter 2011, our group took the Computer Science 115: Software Methodology course. In this class, teams of students were expected to independently complete a software project using the software project management and development methods learned. For our project, Professor Margarita Azmitia of the Psychology department approached us with a research problem that she hoped we could help solve. She and her graduate student, Gina Thomas, were interested in conducting an experiment on how adolescents develop their identity. Their research methods required that participants in this study would fill out a survey of questions concerning their mood 5 to 6 times per day at random times throughout the day. This process would continue for approximately two weeks. After this period, the data could then be analyzed to see how the participant's responses changed throughout the day, based on their environment and the people they were interacting with.

Professor Azmitia and Gina Thomas thought that a mobile application would be an effective way of administering this survey, but they did not have the technical knowledge required to develop one to meet their needs. After discussing the project with them, our group designed and developed a solution that we thought would be both easy to administer for the researchers, as well as intuitive for the participants in the study. This plan consisted of two major components.

The first component was a basic PC application that the researchers could use to create and edit the contents of their surveys, as well as view the responses collected from all of the participants. The second component was a mobile application to run on the Android platform that would distribute this survey at the required random times throughout the day, and then collect each participant's responses into an easy to read spreadsheet (using Microsoft Excel) which the researchers could access later for analysis.

After completing the course, we are happy to announce that we have developed a successful solution for Professor Margarita and Gina Thomas' that is specific to the needs of their research project. They plan on utilizing our application to perform their research starting at the end of spring quarter 2012. Additionally, due to our generalized approach to the design of our application, it can be easily adapted to other research projects. Due to this generality, many other research groups at UC Santa Cruz have expressed interest in utilizing our application and we hope to accommodating them.

Table of Contents:

1. Introduction

This project was independently developed by five students in the Computer Science 115: Software Methodology course during fall quarter 2011. Software Methodology is a computer science course that focuses on well-engineered software systems. The course covers analysis and specification, design, programming and project management. At the heart of this course was learning the real world industry standard of agile software development practices, and in particular, Scrum to manage the development of a medium scale software project. Scrum promotes team organization by bringing the customer and all team members together. A key principle of Scrum is the assumption that during a project development the customer's needs will evolve over the course of development. Using Scrum allowed us to quickly respond to changing customer requirements and deliver the product according to the customer's specifications.

Scrum development is composed of several Sprints. A Sprint usually lasts from two to four weeks in duration. In the CS 115 course, our sprints were 2-3 weeks long. At the beginning of each sprint there is a planning stage when the team decides on the list of requirements that will be finished by the end of the sprint. During the planning stage, portions of the project that the team decides to complete are broken down into user stories. A user story describes in everyday language what a user or customer needs and defines what is to be built in the software project.

2. Requirements and Approach

For this project our customers were Professor Margarita Azmitia of the Psychology department here at UC Santa Cruz and her graduate student Gina Thomas. Before the project development, our team met with Professor Azmitia to identify user stories. From these user stories our team developed the technical design and requirements necessary to implement these user stories. Based on our design and technical requirements, we began implementing the application to be as close as possible to the user stories provided by Professor Azmitia. About half-way through the development we met with both Professor Azmitia and Gina Thomas to demonstrate an early working version of the application, and determine the areas that needed to be changed and improved. After we identified new requirements, we adjusted our development plan according to the new customer specifications. At the end of the quarter we delivered our application to our customer, having addressed all of their needs.

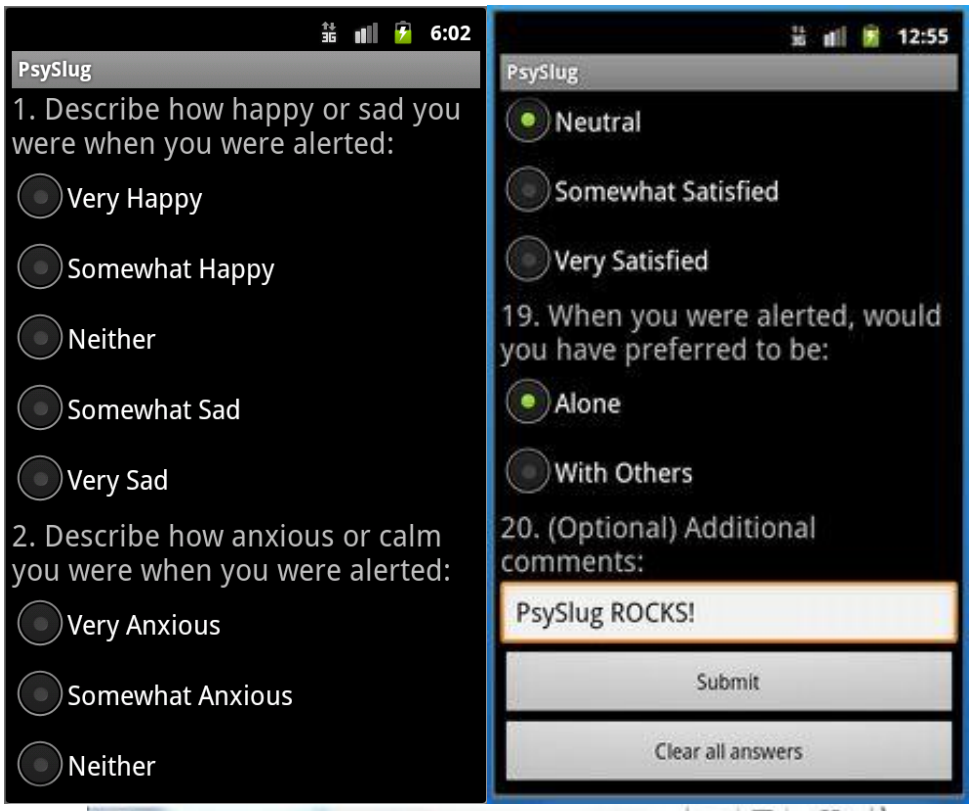
2.1 High Level User Stories from the Customer

- Application needs to run on the Android devices.
- Application must distribute predetermined survey at randomly specified times throughout the day.
- Application needs to collect data in the Excel file format.
- There must be an easy way to store and view all the data and analyze at a later time.

2.2 Our Approach: Derived Technical Requirements from User Stories

- Use the Android SDK, which allows us to install the application on virtually any device that runs the Android Operating System.
- Rather than having a single hard-coded survey, we decided that it would be best if our application could load surveys dynamically, allowing the researchers to change their survey as needed, as well as allowing the application to be adopted for other research projects in the future.
- Use the JXL API to programmatically generate Excel documents from the participant's survey responses.
- Use the cloud storage utility Dropbox's API to create a repository for completed participant survey responses. This makes the data files available from any computer immediately after the research participant completes the survey.

3. Screen-Shots



Visit the project web-site for a video demonstration.

4. Developers

- Vadim Maximov
- Maria Arbuzova
- Brian Nguyen
- Maria Mishkova
- Nick Lutz

5. Project web-site

Visit the project web-site for videos and user documentation
https://sites.google.com/site/psyslugapplication/

Appendix A: Application Development Process Using Scrum

A.1 Release Plan

Team Name: PsySlug
Release Name: Natty Narwhal 2
Release Date: November 29

High Level Goals

- Distributable App
- Working UI
- Alert System
- Database for survey answers

User Stories for Release

Sprint 1

- 3 pts - As a user I need a client that runs on my android phone so I can respond to surveys
- 3 pts - As a researcher I need the ability to display a question on a client's phone for them to respond to
- 2 pts - As a user I need working buttons so I can respond to questions
- 3 pts - As a programmer I need a User Class to store information about each survey participant including: Name, Age, Sex, List of answers, etc.
- 3 pts - As a user I need to receive alerts to know when to do the survey.
- 3 pts - As a programmer I need a graphic to use when I create the UI.

Sprint 2

- 2 pts - As a user I need the app to run at startup so I don't have to keep re-entering information and start the application.
- 3 pts - As a researcher I need a protected setting page which contains a login and pw for the dropbox, Also I need to edit or delete research participants' user IDs.
- 3 pts - As a researcher I need to be able to upload a survey file to a dropbox and then have the survey app pull the questions from this location.
- 3 pts - As a researcher I need all of the answers research participants upload into 1 master file so it is easy to read and manipulate.
- 3 pts - As a researcher I need files uploaded to a location where I can access them.
- 3 pts - As a user I need to be able to upload my survey answers if there is no connection

Sprint 3

- 3 pts - As a researcher I need to be able to merge all of the separate user answer files into a single excel document, separated by page.

- 3 pts - As a researcher I need the answers uploaded by a user to be formatted in an .xls file also the name should be the useridnumber.xls.
- 3 pts - As a researcher I need a configuration app to generate a new survey file.
- 1 pts - As a researcher I need to be able to change the dropbox login information.
- 1 pts - As a researcher I need to change the researcher's ID number in the settings page.
- 2 pts - As a researcher I need a setting page that is password protected to hide information from the research participant.
- 1 pt - As a researcher I need to know when a user was beeped to complete a survey, when they actually submitted their answers, and their user ID number.
- 3 pts - As a researcher I need to know how the application works so that I can use it properly
- 3 pts - As a user I need to makes sure the app works on different types of phones
- 3 pts - As researcher I need to be able to install the app on the phones.

Product Backlog

- Empty.

A.2 Sprint 1

Sprint Completion Date: Oct 14, 2011
Goal: Working client UI, ability to perform survey, and, alarm notifications.

Tasks/User Stories:

- As a programmer I need a User Class to store information about each survey participant including: Name, Age, Sex, List of answers, etc.
- Program user class (15 hours)
- As a user I need a client that runs on my android phone so I can respond to surveys
- Ability for user text input to be saved (10 hours)
- As a researcher I need the ability to display a question on a client's phone for them to respond to
- Display questions from survey file (10 hours)
- As a programmer I need a graphic to use when I create the UI
- Make icons (2 hours)
- Make background (1 hour)
- Make splash screen (4 hours)
- As a user I need working buttons so I can respond to questions
- Ability for the user to select options from a checklist (10 hours)
- As a user I need to receive alerts to know when to do the survey.
- Send intent to android phone for notification alerts (15 hours)
- Save alerts (in case phone turns off) (15 hours)

Team Roles:

- Nick - Product Owner

- Maria A. - Scrum Master
- Maria M. - Scrum Master
- Vadim - Developer
- Brian – Developer

Task Assignments:

- Nick + Maria M. - Survey display, Input, Checklist
- Maria A. + Brian - Notifications
- Vadim - User Class
- Everyone - Art Assets

User Stories Completed:

- As a programmer I need a User Class to store information about each survey participant including: Name, Age, Sex, List of answers, etc.
- As a user I need a client that runs on my android phone so I can respond to surveys.
- As a researcher I need the ability to display a question on a client's phone for them to respond to.
- As a programmer I need a graphic to use when I create the UI.
- As a user I need working buttons so I can respond to questions.
- As a user I need to receive alerts to know when to do the survey.
- Work completion rate:
- .3pts per person per day (1 pt = ~4 Hours)

A.3 Sprint 2

Sprint Completion Date: Oct 14, 2011

Tasks/User Stories:

- As a user I need the app to run at startup so I don't have to keep reentering information and start the application
- Set boot up broadcastReciever to read from time database(8 hours)
- As a researcher I need a pw protected setting page which contains a login and pw for the dropbox, Also I need to edit or delete research participants' user IDs.
- Make settings class(12 hours)
- As a researcher I need to be able to upload a survey file to a dropbox and then have the survey app pull the questions from this location.
- Set up survey load from database(12 hours)
- As a researcher I need all of the answers research participants upload into 1 master file so it is easy to read and manipulate.
- Combine uploaded answers into one file(12 hours)
- As a researcher I need files uploaded to a location where I can access them.
- Upload answer files to database to be combined (12 hours)

- As a user I need to be able to upload my survey answers if there is no connection
- Make service to upload answers when connection available(12 hours)

Team Roles:
- Nick - Product Owner
- Maria A. - Developer
- Maria M. - Developer
- Vadim - Developer
- Brian – Scrum Master

Task Assignments:
- Nick – Combine User uploaded answers, Custom Graphics (Sprint 1)
- Maria M. – Settings Page
- Maria A. – Survey Config File Location
- Brian - Alarms (Sprint 1), Run at Start Up
- Vadim – Survey Answer Uploading (no connection), Change Files upload Location

User Stories Completed:
- As a user I need the app to run at startup so I don't have to keep re-entering information and start the application
- As a researcher I need files uploaded to a location where I can access them.
- As a user I need to be able to upload my survey answers if there is no connection
- As a researcher I need to be able to upload a survey file to a dropbox and then have the survey app pull the questions from this
- Location.
- As a researcher I need all of the answers research participants upload into 1 master file so it is easy to read and manipulate.

Work completion rate:
- User Stories Completed - 4
- Hours Completed - 68 hours
- Days of Sprint - 18 Days
- Sprint 1 Figures - .3pts per person per day (1 pt = ~4 hours)
- Sprint 2 Figures - .2pts per person per day (1 pt = ~4 hours)

A.4 Sprint 3

Sprint Completion Date: Dec 1, 2011 (Release Date)
Goal: Finish Product, Polish, Test

Tasks/User Stories:

- 3 pts - As a researcher I need to be able to merge all of the separate user answer files into a single excel document, separated by page. (12 hours)
- 3 pts - As a researcher I need the answers uploaded by a user to be formatted in an .xls file also the name should be the useridnumber.xls.(12 hours)
- 3 pts - As a researcher I need a configuration app to generate a new survey file.(12 hours)
- 1 pts - As a researcher I need to be able to change the dropbox login information.(4 hours)
- 1 pts - As a researcher I need to change the researcher's ID number in the settings page.(4 hours)
- 2 pts - As a researcher I need a setting page that is password protected to hide information from the research participant. (8 hours)
- 1 pt - As a researcher I need to know when a user was beeped to complete a survey, when they actually submitted their answers, and their user ID number.
- Include time stamp in survey answers (4 hours)
- 3 pts - As a researcher I need to know how the application works so that I can use it properly (12 hours)
- Write user documentation (12 hours)
- 3 pts - As a user I need to makes sure the app works on different types of phones
- Test on multiple devices (8 hours)
- 3 pts - As researcher I need to be able to install the app on the phones(12 hours)
- Install application on given survey phones

Team Roles:

- Nick - Product Owner
- Maria A. - Developer
- Maria M. - Developer
- Vadim - Scrum Master
- Brian – Developer

Task Assignments:

- Nick – Custom Graphics, Compatibility Testing, Distribution
- Maria M. – Setting page (3 stories), Compatibility Testing
- Maria A. – Change to .xls file, timestamp, Compatibility Testing
- Brian - Merging excel files, Compatibility Testing
- Vadim – Changing the survey filename, generating the survey, Compatibility Testing

Work completed:

- 3 pts - As a researcher I need to be able to merge all of the separate user answer files into single excel document, separated by page.
- 3 pts - As a researcher I need the answers uploaded by a user to be formatted in an .xls file also the name should be the useridnumber.xls.
- 3 pts - As a researcher I need a configuration app to generate a new survey file.
- 1 pts - As a researcher I need to be able to change the dropbox login information.

- 1 pts - As a researcher I need to change the researcher's ID number in the settings page.
- 2 pts - As a researcher I need a setting page that is password protected to hide information from the research participant.
- 1 pt - As a researcher I need to know when a user was beeped to complete a survey, when they actually submitted their answers, and their user ID number.
- 3 pts - As a researcher I need to know how the application works so that I can use it properly.
- 3 pts - As a user I need to makes sure the app works on different types of phones.
- 3 pts - As researcher I need to be able to install the app on the phones.

Work completion rate:

- User Stories Completed - 10
- Hours Completed - 92 hours
- Days of Sprint - 27 Days
- Sprint 1 Figures - .3pts per person per day (1 pt = ~4 hours)

Appendix B: Testing

Sprint Completion Date: Nov 29, 2011 (Release Date)

Unit Tests:

- AnswerWriter:
    1. Used a driver to create a workbook, and tested creating a header from a survey class and inputting answers by calling various AnswerWriter functions from the driver. We tested different surveys and input answers. For answers, we tested multiple choice answer, text-only answers, “Other:” text answers, and a mix of all the listed.

- BootBroadcastR:
    1. Turn on android device, check to see if alarms are set from database.

- CancelAlarmBCR:
    1. Use driver to set off notification (psyBroadcastR), check to see if notification is removed in 15 minutes.

- DropboxLogin:
    1. Log into Dropbox from device, check if authenticated.

- LoginActivity:
    1. Enter user id and enter to see if user is saved and is redirected to the questionnaire.

- NoSurveyActivity:
    1. Manually run application from device. The application should block the user from taking the survey without a notification and only allow them to access settings page.

- psyBroadcastR:
    1. Use driver to call this broadcast receiver, check notification, notification click, and notification removal.

- PsySlugDropboxService:
    1. Upon first app use, go to Settings and set the Dropbox login information. Next, click on “Download Survey” and check that you are redirected to the newly downloaded survey.

- PsySlugIdleService:
    1. Use driver to call and check if ScheduleAlarmBR is set on repeat for everyday at 15 minutes before survey starting time.

● ScheduleAlarmBR:
1. Use driver to call and check if 7 alarms are being set throughout the day at random times (using requirement rules), and that those 7 generated times are being saved to database

● SettingsActivity: Open application, go to Settings. Tests:
1. Click “Access the app at any time”, unclick to see that this bullet works.
2. Click “Stop Notifications”; unclick to see that this bullet works.
3. Click “Set Dropbox Account Info”; check that the pop-up box for login info comes up. (1) Click OK, check pop-up disappears and “Thank you!” appears. (2) Click
4. Cancel, check pop-up disappears.
5. Click “Change User ID”; check that a pop up to change the ID comes up. (1)
6. Click OK, check pop-up disappears. (2) Click Cancel, check pop-up disappears.
7. Click “Download Survey”; check that you are redirected to the new survey.

● Survey:
1. Run the application. Check that the survey questions are listed correctly with appropriate answers. Use different survey versions: survey with multiple choice answers without “Other:” answers; survey with multiple choice answers with “Other:” answers; survey with only text input answers; survey which is a mix of the above surveys.

● MergeWindow:
1. Pull the .xls files from Dropbox and run through Merge.
2. Open the merged file and see if its contents are as expected.

System Tests:

Launch the PsySlug application. Enter any user ID. Go to Settings, enter and confirm password to get to the Settings page. Once the Settings page is accessed, user can change drop box password, user ID, and user password.
If want to download survey enter Dropbox username and password and click on download survey button and the survey shows up.

● Client UI:
1. Log-in: User can log into application (first use)
2. Survey: Survey is being displayed with answers. Answers will be answered in random manner and submitted.
3. Settings: Settings page can be accessed (password only) and it contains notification enable / disable, editing log in information for user and Dropbox, and downloading new survey.
4. Application: Application is locked from being run manually.

● Survey:

1. Generating: Survey can be generated from utility application, test that file in application is displayed correctly in UI.
2. Prompting: Notifications are displayed randomly 7 times throughout the day in 2 hour blocks (randomly within each block) every day. The survey can be taken from that notification; the notification will disappear after 15 minutes.
3. Downloading: Survey can be downloaded from settings page.

● Answers:
1. Saving: Answers are saved into an excel file. If user misses or refuses to take
2. the survey, a time stamp with empty answers in-line will be inputted into the excel file.
3. Uploading: The excel worksheets with answers are uploaded to Dropbox. If no connection is available, the answers will keep trying to upload until uploaded.
4. Merging: With the use of PsySlugUtility, excel answers from multiple users can be merged into one master file with each user being a separate sheet.

Appendix C: Code

To download source code please visit: https://sites.google.com/site/psyslugapplication/

C.1 Answer.java

```
public class Answer {
	private String type;
	private String answer;

	public Answer(String type, String answer){
		this.type = type;
		this.answer = answer;
	}

	public void setType(String type) {
		this.type = type;
	}

	public String getType() {
		return type;
	}

	public void setAnswer(String answer) {
		this.answer = answer;
	}

	public String getAnswer() {
		return answer;
	}
}
```

C.2 AnswerWriter.java

```
/**
 * AnswerWriter.java Creates and writes survey answers to a new Excel Workbook
 * and saves the contents as a .xls file.
 */
public class AnswerWriter {
   // Constants
   private static final int COL_WIDTH = 4;
   private static final int ROW_HEIGHT = 400;
   // private static final int ID_COL = 0;
   private static final int DATE_COL = 0;
   // private static final int TIME_COL = 2;
   private static final int HEADER_TOP = 0;
   private static final int HEADER_BOT = 1;
   // Fields
   private WritableWorkbook workbook;
   private Workbook input_wb;
   private int total_column;
   private int total_row;
   private File xls_file;
   private File tempfile;

   Context fileContext;

   /**
```

```
 * Constructor
 *
 * @param wb
 * @throws BiffException
 * @throws IOException
 */
AnswerWriter (Context fileContext, String wb) {
   xls_file = new File(fileContext.getFilesDir() + "/" + wb + ".xls");
   if(!xls_file.exists()) {
      try {
         WritableWorkbook wwb = Workbook.createWorkbook(xls_file);
         wwb.createSheet(wb, 0);
         wwb.write();
         wwb.close();
      } catch (IOException e) {
         Log.d("AnswerWriter Error: ", "Failed to open " + wb + ".xls");
      } catch (WriteException e) {
         Log.d("AnswerWriter Error: ", "Failed to write "+ wb + ".xls");
      }
   }
   tempfile = new File(fileContext.getFilesDir() + "/"
      + "tempfile.xls");

   try {
      input_wb = Workbook.getWorkbook(xls_file);
      workbook = Workbook.createWorkbook(tempfile, input_wb);
   } catch (Exception e) {
      // put some message here
   }

}

/**
 * Creates the column headers for the excel workbook based on a provided list
 * of questions. The sheet to be edited is sheet_num The format of the header
 * for a list of questions of length n is: USERID | DATE | TIME | QUESTION 1
 * | ... | QUESTION N
 *
 * @param sheet
 * @param lq
 * @throws RowsExceededException
 * @throws WriteException
 */
public void createHeader (WritableSheet sheet, List<Question> lq)
   throws RowsExceededException, WriteException {
   int col_width = COL_WIDTH;
   int row_height = ROW_HEIGHT;
   // Basic cell formatting
   WritableCellFormat wcf = new WritableCellFormat();
   wcf.setAlignment(Alignment.CENTRE);
   wcf.setVerticalAlignment(jxl.format.VerticalAlignment.CENTRE);
   wcf.setBorder(jxl.format.Border.ALL,
      jxl.format.BorderLineStyle.THIN);
   sheet.setRowView(HEADER_TOP, row_height);
```

```
// Creates USERID, DATE and TIME header cells
/*
 * NO LONGER USED Label userid = new Label(ID_COL, HEADER_TOP, "USER ID",
 * wcf); sheet.addCell(userid); sheet.mergeCells(ID_COL, HEADER_TOP,
 * ID_COL, HEADER_BOT); sheet.setColumnView(ID_COL, 10);
 */

String time_stamp = "SURVEY COMPLETION TIME";
Label date = new Label(DATE_COL, HEADER_TOP, time_stamp, wcf);
sheet.addCell(date);
sheet.mergeCells(DATE_COL, HEADER_TOP, DATE_COL, HEADER_BOT);
sheet.setColumnView(DATE_COL, 8);

/*
 * NO LONGER USED Label time = new Label(TIME_COL, HEADER_TOP, "TIME",
 * wcf); sheet.addCell(time); sheet.mergeCells(TIME_COL, HEADER_TOP,
 * TIME_COL, HEADER_BOT); sheet.setColumnView(TIME_COL, 8);
 */

// Iterates through all questions in the list and creates
// a super column titled QUESTION # for all questions, with
// sub column for each answer corresponding to that question.
int index = 1;
// Label quest_label;
Label ans_label;
String ans_header = "";
String quest_header = "";
List<Answer> la;
int col_start = index;
int col_end;
int answer_count = 0;
WritableCellFeatures cellFeatures;// = new WritableCellFeatures();
int current_col;

for (Question question : lq) {
   quest_header = "QUESTION " + index;
   cellFeatures = new WritableCellFeatures();
   cellFeatures.setComment(question.getText());
   Label quest_label = new Label(col_start, 0, quest_header, wcf);
   quest_label.setCellFeatures(cellFeatures);
   sheet.addCell(quest_label);
   Log.d("AnswerW ", "number: " + question.getQuestionNumber());
   col_end = question.getAnswerList().size() + col_start - 1;
   la = question.getAnswerList();
   if (question.getAnswerList().size() > 1) {
      sheet.mergeCells(col_start, 0, col_end, 0);
   }
   total_column = col_end;
   // Sets column width
   for (int argi = col_start; argi <= col_end; ++argi) {
      for (Answer ans : la) {
         if (ans != null) {
            ans_header = ans.getAnswer();
            col_width = ans_header.length() + 4;
            current_col = col_start + answer_count;
```

```
                sheet.setColumnView(current_col, col_width);
                ans_label = new Label(current_col, 1, ans_header, wcf);
                sheet.addCell(ans_label);
                ++answer_count;
             }
          }
          answer_count = 0;
          /*
           * if (question.getType().equals("text")) {
           * sheet.setColumnView(col_start, col_width); ans_label = new
           * Label(col_start, 1, "q12", wcf); sheet.addCell(ans_label); }
           */
          if (question.getType().equals("text")) {
             for (Answer ans : la) {
                if (ans != null) {
                   ans_header = ans.getAnswer();
                   col_width = ans_header.length() + 6;
                   sheet.setColumnView(col_start, col_width);
                   ans_label = new Label(col_start, 1, ans_header,
                      wcf);
                   sheet.addCell(ans_label);
                }
             }
          }
       }
       col_start += question.getAnswerList().size();
       ++index;
    }
 }

 public void newEntry (WritableSheet sheet) throws WriteException {
    int new_row = 2;
    while (sheet.getCell(DATE_COL, new_row).getContents() != "") {
       ++new_row;
    }
    WritableCellFormat wcf = new WritableCellFormat();
    wcf.setAlignment(Alignment.CENTRE);
    wcf.setVerticalAlignment(jxl.format.VerticalAlignment.CENTRE);
    wcf.setBorder(jxl.format.Border.ALL,
       jxl.format.BorderLineStyle.THIN);

    /*
     * NO LONGER USED Label userid = new Label(ID_COL, new_row,
     * sheet.getName(), wcf); sheet.addCell(userid);
     */

    Instant time = new Instant();
    String time_str = time.toDate().toString();
    Label date = new Label(DATE_COL, new_row, time_str, wcf);
    sheet.addCell(date);
    sheet.setColumnView(DATE_COL, time_str.length());
 }

 public void borderEmptyCells (WritableSheet sheet)
    throws WriteException {
```

```
      int col_width = COL_WIDTH;
      int row_height = ROW_HEIGHT;
      // Basic cell formatting
      WritableCellFormat wcf = new WritableCellFormat();
      wcf.setAlignment(Alignment.CENTRE);
      wcf.setVerticalAlignment(jxl.format.VerticalAlignment.CENTRE);
      wcf.setBorder(jxl.format.Border.ALL,
         jxl.format.BorderLineStyle.THIN);
      sheet.setRowView(HEADER_TOP, row_height);
      Cell temp;

      int curr_row = 2;
      while(sheet.getCell(DATE_COL, curr_row).getContents() != ""){
         ++curr_row;
      }
      --curr_row;

     for (int curr_column = 1; curr_column <= total_column; curr_column++) {
        temp = sheet.getWritableCell(curr_column, curr_row);
        if (temp.getContents().equals("")) {
           Label input = new Label(curr_column, curr_row, "", wcf);
           sheet.addCell(input);
           Log.d("AnswerWriter", "Bordering empty cell " + curr_row + " , " +
curr_column);
        }
     }

   }

   /**
    * Inputs the passed in string answer into the respective cell on
    * spreadsheet.
    *
    * @param sheet
    * @param questnum
    * @param answer
    * @throws WriteException
    */
   public void inputAnswer (WritableSheet sheet, String question,
      String answer, boolean question_type) throws WriteException {

      int xcolumn = 1;
      int column_end = 0;
      int column_begin = xcolumn;
      int input_column = 0;
      int input_row = 2;
      Label qlabel = (Label) sheet.getWritableCell(xcolumn, 0);
      Label anslabel;
      int questnum = Integer.parseInt(question);
      String cell_input = "X";

      int col_width = COL_WIDTH;
      int row_height = ROW_HEIGHT;
      // Basic cell formatting
      WritableCellFormat wcf = new WritableCellFormat();
```

```
      wcf.setAlignment(Alignment.CENTRE);
      wcf.setVerticalAlignment(jxl.format.VerticalAlignment.CENTRE);
      wcf.setBorder(jxl.format.Border.ALL,
         jxl.format.BorderLineStyle.THIN);
      sheet.setRowView(HEADER_TOP, row_height);
      Cell cell_below = sheet.getWritableCell(xcolumn, 1);

      // Finds input row by checking time stamps row.

      while (sheet.getCell(DATE_COL, input_row).getContents() != "") {
         ++input_row;
      }
      --input_row;
      /*
      Cell cell_below2 = sheet.getWritableCell(0, 0);
      for (int y = 0; cell_below2 instanceof Label; y++) {
         input_row = y;
         cell_below2 = sheet.getWritableCell(0, y);
      }
      input_row++;
      */
      total_row = input_row;

      Log.d("InputAnswer", "Question: " + questnum + " Answer: "
         + answer);

      // Finds the question title and finds the cell's beginning (column_begin)
      // and end (column_end)
      while (column_end == 0) {
         Cell temp = sheet.getWritableCell(xcolumn, 0);
         String qlabel_text = "";
         if (temp instanceof Label) {
            qlabel = (Label) temp;
            qlabel_text = qlabel.getString();
            if (qlabel_text.contains(question)) {
               column_begin = qlabel.getColumn();
            }
         }

         if (column_begin > 0
            && (qlabel_text.contains(String.valueOf(questnum + 1)) || cell_below
instanceof EmptyCell)) {
            column_end = xcolumn - 1;
         }
         if (!(cell_below instanceof Label)) {
            break;
         }
         xcolumn++;
         cell_below = sheet.getWritableCell(xcolumn, 1);
      }
      Log.d("InputAnswer", "column begin: " + column_begin
         + " column end: " + column_end);

      // question_type: (true) - Mult Choice, (false) - non-mult Choice
      // If Mult choice, we go through the template answers, and find the input
```

```
      // column.
      if (question_type) {
         for (int column_curr = column_begin; column_curr <= column_end;
column_curr++) {
            anslabel = (Label) sheet.getWritableCell(column_curr, 1);
            if (anslabel.getString().equals(answer)) {
               Log.d("InputAnswer", "found column_curr : "
                  + column_curr);
               if (answer.contains("Other:")) {
                  cell_input = answer;
               }
               input_column = column_curr;
            }
         }
      } else {
         input_column = column_begin;
         cell_input = answer;
      }
      Log.d("InputAnswer", "input_column: " + input_column);
      Label input_answer = new Label(input_column, input_row,
         cell_input, wcf);
      sheet.addCell(input_answer);

      Log.d("InputAnswer", "put X to: " + input_column + " "
         + input_row);
      return;
   }

   /**
    * Creates a workbook sheet with a sheet name of userid and sheet number of
    * sheet_num.
    *
    * @param userid
    * @param sheet_num
    * @return Returns the new sheet.
    */
   public WritableSheet createSheet (String userid, int sheet_num) {
      String title = userid;
      WritableSheet sheet = workbook.createSheet(title, sheet_num);
      return sheet;
   }

   /**
    * Writes out the data held in AnswerWriter's workbook field in Excel format.
    *
    * @throws IOException
    */
   public void writeWorkBook () throws IOException {
      workbook.write();
   }

   /**
    * Closes AnswerWriter's workbook, and makes any memory allocated available
    * for garbage collection. Also closes the underlying output stream if
    * necessary.
```

```
     *
     * @throws WriteException
     * @throws IOException
     */
    public void closeWorkBook () throws WriteException, IOException {
       workbook.close();
    }

    public boolean sheetExists (int sheetnum) {
       Sheet tempsheet = workbook.getSheet(sheetnum);
       if (tempsheet != null) {
          return true;
       }
       return false;
    }

    public WritableSheet getSheet (int sheetnum) {
       WritableSheet tempsheet = workbook.getSheet(0);
       return tempsheet;
    }

    public void replaceXLS(Context fileContext, String wb) {
       xls_file.delete();
       tempfile.renameTo(new File(fileContext.getFilesDir() + "/" + wb + ".xls"));
    }
}
```

C.3 BootBroadcastR.java

```
//Schedules the 7 alarms per day when phone boots from a database
//Please note this code is similar to ScheduleAlarmBR as both do similar things at
different phone states
public class BootBroadcastR extends BroadcastReceiver {

   private AlarmManager [] am;
   private int [] numbers;
   private final int MAX_ALARMS = ScheduleAlarmBR.MAX_ALARMS;
   private final int START_TIME_HOUR = ScheduleAlarmBR.START_TIME_HOUR;   //Starting
hour for experiment (USE 24-hour clock)
   private final int START_TIME_MINUTE = ScheduleAlarmBR.START_TIME_MINUTE;
//Starting minute
   private final int START_TIME_SECOND = ScheduleAlarmBR.START_TIME_SECOND;
   private final int TIME_INTERVAL_SEC = ScheduleAlarmBR.TIME_INTERVAL_SEC;
   private final int MILLI = 1000;
   private static final String BBRTAG = "BootBroadcastR";

    @Override
    public void onReceive(Context context, Intent intent) {
      User user = new User("unknown", context.getApplicationContext()); // id is
unknown
      user.loadUserInfo();

      if(user.getNotifications().equals("true")){
          Calendar cur_cal = new GregorianCalendar();
          cur_cal.setTimeInMillis(System.currentTimeMillis());//set the current time
and date for this calendar


          Calendar cal = new GregorianCalendar();
          cal.set(Calendar.HOUR_OF_DAY, START_TIME_HOUR);
```

```
          cal.set(Calendar.MINUTE, START_TIME_MINUTE);
          cal.set(Calendar.SECOND, START_TIME_SECOND);

         numbers = new int[MAX_ALARMS];


         if(LoadFromFile(context))
         {
            Log.d(BBRTAG, "BootReceiver loaded database");
            am =new AlarmManager[MAX_ALARMS];
            for(int i = 0; i < MAX_ALARMS; ++i)
            {
               Intent notif_intent = new Intent(context, psyBroadcastR.class);
               PendingIntent pendingIntent = PendingIntent.getBroadcast(context, i,
notif_intent, PendingIntent.FLAG_ONE_SHOT);
               am[i] = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);


               //checks current time compared to random times
               if(System.currentTimeMillis() <= (cal.getTimeInMillis() + (i *
TIME_INTERVAL_SEC * MILLI) + (numbers[i] * 1000)))
               {
                  am[i].set(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis() + (i *
TIME_INTERVAL_SEC * MILLI) + (numbers[i] * 1000), pendingIntent);
                  Date d = new Date(cal.getTimeInMillis() + (i * TIME_INTERVAL_SEC *
MILLI) + (numbers[i] * 1000));
                  Log.d(BBRTAG, "Setting alarm at " + d);
               }
               else
               {
                  Date d = new Date(cal.getTimeInMillis() + (i * TIME_INTERVAL_SEC *
MILLI) + (numbers[i] * 1000));
                  Log.d(BBRTAG, "Alarm set at " + d + " was not set");
               }
            }
         }
         else{
            Log.d(BBRTAG, "Alarm database not found, no alarms set");
         }
      }
      else
         Log.d(BBRTAG, "Boot Alarms are disabled");

      //Start idleService to reinstate daily alarm scheduler
        context.startService(new Intent( context, PsySlugIdleService.class));
    }

    private boolean LoadFromFile(Context context){
       File path = context.getFilesDir();
       BufferedReader br;
        try {
            br = new BufferedReader(new FileReader(path + "/alarms.txt"));
            String strLine = "";
            StringTokenizer st = null;
            int tokenNumber = 0;

         while ((strLine = br.readLine()) != null) {
            st = new StringTokenizer(strLine, ",");

            while (st.hasMoreTokens()) {
               numbers[tokenNumber] = Integer.parseInt(st.nextToken());
               tokenNumber++;
            }
```

```
         }
            br.close();
            return true;
        } catch (FileNotFoundException ex) {
            Log.d(BBRTAG, "Error loading alarms.txt file: " + ex);
        } catch (IOException ex) {
            Log.d(BBRTAG, "Error loading alarms.txt file: " + ex);
        }
        return false;
    }
}
```

C.4 CancelAlarmBCR.java

```
//Will cancel an alarm if it doesn't get used in 15 minutes
public class CancelAlarmBCR extends BroadcastReceiver  {

       private static final String TAG = "CancelAlarmBCR";
       NotificationManager nm;

       @Override
        public void onReceive(Context context, Intent intent) {
              Log.d(TAG, "onStart");
              nm = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
              nm.cancel(psyBroadcastR.NOTIF_ID);

              Date d = new Date(System.currentTimeMillis());
              Log.d(TAG, "Notification Removed at " + d);

              User user = new User("unknown", context.getApplicationContext()); //id is
unknown
              Survey survey = new Survey(context);

              if (user.loadUserInfo()) {
                     AnswerWriter aw = new AnswerWriter(context, user.getUserID());
                     survey.loadQuestions(new Activity());
                     try {
                     Log.d(TAG, "Writing empty answers to xls");
                     WritableSheet sheet;
                     if(aw.sheetExists(0)) {
                        sheet = aw.getSheet(0);
                     } else {
                        sheet = aw.createSheet(user.getUserID(), 0);
                     }
                     aw.createHeader(sheet, survey.getQuestionList());
                     aw.newEntry(sheet);
                     aw.borderEmptyCells(sheet);
                     aw.writeWorkBook();
                     aw.closeWorkBook();
                     aw.replaceXLS(context, user.getUserID());
                } catch (Exception e) {
                      Log.d(TAG, "Error: " + e);
                    e.printStackTrace();
```

```
                }
            }

        }
}
```

C.5 DropboxLogin.java

```
public class DropboxLogin{
    private static final String TAG = "LoginAsyncTask";

    String mUser;
    String mPassword;
    String mErrorMessage="";
    PsySlugDropboxService mDropbox;
    DropboxAPI.Config mConfig;
    DropboxAPI.Account mAccount;

    public DropboxLogin(PsySlugDropboxService act, String user, String password,
DropboxAPI.Config config) {
        mDropbox = act;
        mUser = user;
        mPassword = password;
        mConfig = config;
    }

    public Integer login() {
        try {
             DropboxAPI api = mDropbox.getAPI();

             int success = DropboxAPI.STATUS_NONE;
             if (!api.isAuthenticated()) {
                  mConfig = api.authenticate(mConfig, mUser, mPassword);
                  mDropbox.setConfig(mConfig);

                  success = mConfig.authStatus;

                  if (success != DropboxAPI.STATUS_SUCCESS) {
                    return success;
                  }
             }
             mAccount = api.accountInfo();

             if (!mAccount.isError()) {
                    return DropboxAPI.STATUS_SUCCESS;
             } else {
                    Log.e(TAG, "Account info error: " + mAccount.httpCode + " " +
mAccount.httpReason);
                    return DropboxAPI.STATUS_FAILURE;
             }
        } catch (Exception e) {
            Log.e(TAG, "Error in logging in.", e);
            return DropboxAPI.STATUS_NETWORK_ERROR;
        }
    }
    public void getAccountInfo(){
       if (mAccount != null) {
             mDropbox.displayAccountInfo(mAccount);
       }
    }
}
```

C.6 LoginActivity.java

```
public class LoginActivity extends Activity {
    private static final String TAG = "LoginActivity";
       private static final int menuSettings = Menu.FIRST;
       private static final int menuQuit = Menu.FIRST + 1;
       public static User user;
    private EditText userID;

   @Override
       public void onBackPressed() {
              //Do nothing if back button is pressed
          return;
       }

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       //deleteFile("answers.csv"); We are not deleting old file anymore,
       //because we want to append to the old file

       //Stop services while taking the survey
       stopService(new Intent(this, PsySlugIdleService.class));
       stopService(new Intent(this, PsySlugDropboxService.class));

       user = new User("unknown", getApplicationContext()); //id is unknown

       /*if the user file was loaded, this means that the user already
       * entered his/her user id, so we go straight to the Survey
       * if user info was not loaded we show this login activity.
       */
              if (user.loadUserInfo()) {
                     if (canTakeSurvey()) {
                            startActivity(new Intent(this, PsyslugActivity.class));
                            finish();
                     } else
                            this.finish();
              }
              setContentView(R.layout.login);

              //Start idleservice incase it isnt already running (multiple runs are
taken care of)
              startService(new Intent(this, PsySlugIdleService.class));

              //Cancel the notificationCanceller alarm
              cancelNotificationRemover();
    }
    /**
     * When submit button is pressed we check:
     * if there was anything entered into the text field
     * and if there is something in the text field,
     * we store the id and proceed to the survey.
     * if there is nothing in the text field we display error message
     * @param v
     */
       public void submitPress(View v) {
              switch (v.getId()) {
              case R.id.submit: {
                     {
```

```
                        userID = (EditText) findViewById(R.id.editText1);

                        if (userID.getText().length() == 0){
                              Toast.makeText(this, "Please enter a valid ID",
Toast.LENGTH_LONG).show();
                              return;
                        }
                        user.setUserID(userID.getText().toString());
                        user.storeUserInfo();
                        //new File(user.getUserID() + ".xls");
                        startActivity(new Intent(this, PsyslugActivity.class));
                        finish();
                  }
            }
            }
      }
      /**
       * Creates menu buttons
       */
  public boolean onCreateOptionsMenu(Menu menu){
        menu.add(0, menuSettings, 0, "Settings");
        menu.add(0, menuQuit, 0, "Quit");
        return true;
  }
  /**
   * Processes menu button presses
   */
  public boolean onOptionsItemSelected(MenuItem item){
        switch(item.getItemId()){
        case menuSettings:
              startActivity(new Intent(this, SettingsActivity.class));
              return true;
        case menuQuit:
              finish();
              return true;
        }
        return false;
  }
  /**
   * Deletes the file from app's directory
   * @param fileName - name of the file to be deleted
   * @return returns true if the file was deleted otherwise returns false.
   */
      public boolean deleteFile(String fileName) {
            File path = getFilesDir();

            for (File f : path.listFiles()) {
                  if (f.isFile()) {
                        if (f.getName().equals(fileName)) {
                              f.delete();
                              return true;
                        }
                  }
            }
            return false;
      }

      public boolean canTakeSurvey() {
            /*
             * check if we want to bypass this lockout scree
             * set in settings.
             */
            if(user.getAccess().equals("true")){
```

```
                return true;
            }
            /*
             * Check if this is a legal app launch: the notification went off or
             * this is a first time
             */
            Bundle extra = getIntent().getExtras();
            if (extra == null) {
                Log.d(TAG, "Can't take the Survey");
                startActivity(new Intent(this, NoSurveyActivity.class));
                return false;
            } else {
                String surveyUpdate = extra.getString("notification");
                if (surveyUpdate.equals("true")) {
                    Log.d(TAG, "Can take Survey");
                    return true;
                } else {
                    Log.d(TAG, "Can't take Survey");
                    startActivity(new Intent(this, NoSurveyActivity.class));
                    return false;
                }
            }
        }
        private void cancelNotificationRemover(){
             AlarmManager am;
             Intent intentCancel = new Intent(this, CancelAlarmBCR.class);
          PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0,
intentCancel, PendingIntent.FLAG_UPDATE_CURRENT);
          am = (AlarmManager) this.getSystemService(Context.ALARM_SERVICE);
          am.cancel(pendingIntent);

          Log.d(TAG, "Notification alarm remover canceled");

        }
    }
}
```

C.7 NoSurveyActivity.java

```
public class NoSurveyActivity extends Activity {
    private static final String TAG = "NoSurveyActivity";
       private static final int menuSettings = Menu.FIRST;
       private static final int menuQuit = Menu.FIRST + 1;
       private TextView message;
       private Button ok;

       private class ButtonOKHandler implements View.OnClickListener{
              public void onClick(View v){
                    closeActivity();
              }
       }
       @Override
       public void onBackPressed() {
              //Do nothing if back button is pressed
          return;
       }

    private void closeActivity() {this.finish();}

       @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        message = new TextView(this);
        message.setText("You can't take the survey at this time.\nPlease wait for the
next survey notification.");
        message.setTextSize(25);
        message.setTextColor(Color.RED);
        LinearLayout layout = new LinearLayout(this);
       layout.setOrientation(LinearLayout.VERTICAL);

       ok = new Button(this);
       ok.setText("OK");
       ok.setOnClickListener(new ButtonOKHandler());
       layout.addView(message, new LayoutParams(LayoutParams.FILL_PARENT,
LayoutParams.WRAP_CONTENT));
       layout.addView(ok, new LayoutParams(LayoutParams.FILL_PARENT,
LayoutParams.WRAP_CONTENT));

       setContentView(layout);
    }

       /**
        * Creates menu buttons
        */
  public boolean onCreateOptionsMenu(Menu menu){
        menu.add(0, menuSettings, 0, "Settings");
        menu.add(0, menuQuit, 0, "Quit");
        return true;
  }
  /**
   * Processes menu button presses
   */
  public boolean onOptionsItemSelected(MenuItem item){
        switch(item.getItemId()){
        case menuSettings:
              this.finish();
              startActivity(new Intent(this, SettingsActivity.class));
              return true;
        case menuQuit:
              finish();
              return true;
        }
        return false;
  }
 }
}
```

C.8 PsySlugDropboxService.java

```
  @Override
   public IBinder onBind(Intenpublic class PsySlugDropboxService extends Service {
   private static final String TAG = "DropboxService";
   private Update update;
   private static int delay = 60000; // 1min
   private User user;

   final static private String CONSUMER_KEY = "REMOVED FOR SECURITY";
   final static private String CONSUMER_SECRET = "REMOVED FOR SECURITY:

   private DropboxAPI api = new DropboxAPI();

   final static public String ACCOUNT_PREFS_NAME = "prefs";
   final static public String ACCESS_KEY_NAME = "ACCESS_KEY";
   final static public String ACCESS_SECRET_NAME = "ACCESS_SECRET";
```

```
   private String userName;
   private String password;

   private Config mConfig;

@Override
   public IBinder onBind(Intent intent) {
      return null;
   }


   public void startPsySlugActivity(){
      Intent i = new Intent(this, PsyslugActivity.class);
      i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
       startActivity(i);
    }
   @Override
   public void onCreate() {
      Log.d(TAG, "onCreate");
   }

   @Override
   public void onDestroy() {
      Log.d(TAG, "onDestroy");
      if (update.isRunning()) {
         update.interrupt();
         update = null; // helps with garbage collectionvadim
      }
   }

   @Override
   public synchronized void onStart(Intent intent, int startid) {
      Log.d(TAG, "onStart");
      user = new User("unknown", this);
      user.loadUserInfo();

      userName = user.getDropBoxuser();
      password = user.getDropboxPassword();

      Bundle extra = intent.getExtras();
      if(extra == null)      {
         update = new Update();
      }
      else{
      String surveyUpdate = extra.getString("surveyUpdate");
      if(surveyUpdate.equals("true"))
            update = new Update(true);
      }

      if (!update.isRunning())
         update.start();
   }

   public DropboxAPI getAPI() {
      return api;
```

```
}

/**
 * Displays error messages
 * @param msg - message to be displayed
 */
public void showToast(String msg) {
   Toast error = Toast.makeText(this, msg, Toast.LENGTH_LONG);
   error.show();
}

/**
 * Logging in to the Dropbox account and getting some info
 * about the account
 * @return true if we logged in, false if failed
 */
private boolean getAccountInfo() {
   if (api.isAuthenticated()) {
      // If we're already authenticated, we don't need to get the login
      // info
      DropboxLogin login = new DropboxLogin(this, null, null, getConfig());
      if (login.login() == 1) {
         login.getAccountInfo();
         return true;
      } else
         return false;
   } else {
      DropboxLogin login = new DropboxLogin(this, userName, password,
            getConfig());
      if (login.login() == 1) {
         login.getAccountInfo();
         return true;
      } else
         return false;
   }
}
/**
 * Displays some useful info about the account
 *
 * @param account
 */
public void displayAccountInfo(DropboxAPI.Account account) {
   if (account != null) {
      Log.d(TAG, "Account info:");
      String info = "Name: " + account.displayName + "\n" + "E-mail: "
            + account.email + "\n" + "User ID: " + account.uid + "\n"
            + "Quota: " + account.quotaQuota;
      Log.d(TAG, info);
   }
}
/**
 * If there are no files in the directory will display:
 * Files in: unknown
 * @param account
 */
```

```
   public void showFiles(DropboxAPI.Account account) {
      String files = "";
      String path = "uknown";
      for (DropboxAPI.Entry e : listFiles(account)) {
         files += "\n" + e.fileName();
         path = e.path.replace(e.fileName(), ""); //cut off the filename, we want just
the path
      }
      Log.d(TAG, "Files in: " + path);
      Log.d(TAG, files);
   }

   /**
    * Gets the list of files in the specified account
    * in the PsySlug folder
    * @param account
    * @return list of files in the PsySlug folder
    */
   public List<DropboxAPI.Entry> listFiles(DropboxAPI.Account account) {
      if (account != null) {
         DropboxAPI.Entry entry = api.metadata("dropbox", "/PsySlug", 0, null, true);
         if (!entry.is_dir)
            return null;
         return entry.contents;
      }
      return null;
   }

   public Config getConfig() {
      if (mConfig == null) {
         mConfig = api.getConfig(null, false);
         mConfig.consumerKey = CONSUMER_KEY;
         mConfig.consumerSecret = CONSUMER_SECRET;
         mConfig.server = "api.dropbox.com";
         mConfig.contentServer = "api-content.dropbox.com";
         mConfig.port = 80;
      }
      return mConfig;
   }

   public void setConfig(Config conf) {
      mConfig = conf;
   }
   /**
    * Uploads file from the app's file directory
    * to the PsySlug  folder
    * @param fileName
    * @return
    */
   public boolean uploadFile(String fileName) {
      File file = new File(getFilesDir() + "/" + fileName);
      Log.d(TAG, "Uploading file: " + file.getName());
      if (file.exists()) {
         api.putFile("dropbox", "/PsySlug", file);
         Log.d(TAG, "Finished uploading file: " + file.getName());
```

```
      showFiles(api.accountInfo());
      return true;
   } else {
      Log.d(TAG, "File: " + fileName + " not found.");
      Log.d(TAG, "Upload cancelled");
      return false;
   }
}
/**
 * Lists all files in the directory in Dropbox
 * specified by path
 * @param path
 * @return
 */
public List<Entry> listDirectory(String path){
   Entry entry = api.metadata("dropbox", path, 0, null, true);
   if (!entry.is_dir) return null;
   return entry.contents;
}
/**
 * Downloads file specified by the full path
 * for this app path is /PsySlug
 * and puts it in the apps directory which is
 * /data/data/android.psyslug/files
 * @param path
 * @param fileName
 * @throws IOException
 */
public void downloadFile(String path, String fileName, boolean survey){
   Log.d(TAG, "Downloading file: " + fileName);
   BufferedInputStream br = null;
   BufferedOutputStream bw = null;
   File localFile = null;
   if(survey){
      localFile = new File(getFilesDir().toString() + "/Survey/" + fileName);
   }
   else{
      localFile = new File(getFilesDir().toString() + "/" + fileName);
   }
   try {
      if (!localFile.exists()) {
         localFile.createNewFile(); // otherwise Dropbox client will fail
                              // silently
      }

      FileDownload fd = api.getFileStream("dropbox", path + fileName, null);
      br = new BufferedInputStream(fd.is);
      bw = new BufferedOutputStream(new FileOutputStream(localFile));

      byte[] buffer = new byte[4096];
      int read;
      while (true) {
         read = br.read(buffer);
         if (read <= 0) {
            break;
```

```
            }
            bw.write(buffer, 0, read);
        }
    } catch (IOException e) {
        Log.d(TAG, "Failed downloading file: e" + fileName);
    } finally {
        // in finally block:
        if (bw != null) {
            try {
                bw.close();
            } catch (IOException e) {
                Log.d(TAG, "Failed closing bw" + e);
            }
        }
        if (br != null) {
            try {
                br.close();
            } catch (IOException e) {
                Log.d(TAG, "Failed closing be" + e);
            }
        }
    }
    Log.d(TAG, "Finished downloading file: " + fileName);
}

class Update extends Thread {
    private boolean isRunning = false;
    private int loginCounter = 0;
    private boolean surveyUpdate = false;

    public Update() {
        super("PsySlugDropboxService"); // name of the thread
    }
    public Update(boolean surveyUpdate) {
        super("PsySlugDropboxService"); // name of the thread
        this.surveyUpdate = surveyUpdate;
    }

    @Override
    public void run() {
        isRunning = true;
        while (isRunning) {
            Log.d(TAG, "Dropbox update is running");
            try {
                if (getAccountInfo()) {
                    if (surveyUpdate) {
                        Log.d(TAG, "Updating survey file...");
                        List<Entry> entry = listDirectory("/PsySlug/Survey/");
                        for (Entry e : entry) {
                            downloadFile("/PsySlug/Survey/", e.fileName(), true);
                            Log.d(TAG, "Updated survey file with: "   + e.fileName());
                        }
                        Log.d(TAG, "Update finished, stopping the service");
                        stopSelf();
                        isRunning = false;
```

```
                    startPsySlugActivity();
                    continue;
                 } else {
                    uploadFile(user.getUserID() + ".xls"); // this will be changed to
userID.xls
                    Log.d(TAG, "Update finished, stopping the service");
                    stopSelf();
                    isRunning = false;
                    continue;
                 }
              }
              /* At this point failed to login
               * Try to login 3 times, if that fails go to sleep for 1
               * hour and try 3 times again
               */
              if (loginCounter >= 3) {
                 loginCounter = 0;
                 Thread.sleep(delay * 60);
              } else {
                 loginCounter++;
                 Log.d(TAG, "Error logging in, going to sleep");
                 Thread.sleep(delay);
              }
           } catch (InterruptedException e) {
              isRunning = false;
           }
           finally{
              //probably should add something here
           }
        }
     }
     public boolean isRunning() {
        return isRunning;
     }
  }
  public static void setDelay(int time) {
     delay = time;
  }
}
```

C.9 PsySlugIdleService.java

```
//When started, it will start a scheduler which runs every day
//The scheduler (ScheduleAlarmBR) will set 7 notifications per day
//This will run 15 mintues before the start time
public class PsySlugIdleService extends Service  {

   AlarmManager am;
   final int START_TIME_HOUR = ScheduleAlarmBR.START_TIME_HOUR;      //Hour / minute
for repeating alarm (alarm to schedule notification alarms)
   final int START_TIME_MINUTE = ScheduleAlarmBR.START_TIME_MINUTE;
   final int START_TIME_SECOND = ScheduleAlarmBR.START_TIME_SECOND;
   private static final String TAG = "PsySlugIdleService";

   @Override
   public IBinder onBind(Intent intent) {
```

```
      return null;
   }

   @Override
   public void onCreate() {
      Log.d(TAG, "onCreate");
      Calendar cur_cal = new GregorianCalendar();
      cur_cal.setTimeInMillis(System.currentTimeMillis());//set the current time and
date for this calendar


      Calendar cal = new GregorianCalendar();
      cal.set(Calendar.HOUR_OF_DAY, START_TIME_HOUR);
      cal.set(Calendar.MINUTE, START_TIME_MINUTE);
      cal.set(Calendar.SECOND, START_TIME_SECOND);
      cal.setTimeInMillis(cal.getTimeInMillis() -
AlarmManager.INTERVAL_FIFTEEN_MINUTES);

      Intent intent = new Intent(this, ScheduleAlarmBR.class);
      PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent,
PendingIntent.FLAG_CANCEL_CURRENT);

      am = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
      Date d = new Date(cal.getTimeInMillis() + AlarmManager.INTERVAL_DAY);

      User user = new User("unknown", getApplicationContext()); // id is unknown
      user.loadUserInfo();

      if(user.getNotifications().equals("true")){
         am.setRepeating(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis() +
AlarmManager.INTERVAL_DAY, AlarmManager.INTERVAL_DAY,pendingIntent);
         Log.d(TAG, "Scheduler Alarm set at " + d + " and for every day after");
      }
      else{
         am.cancel(pendingIntent);   //cancels scheduler

         Intent notif_intent = new Intent(this, psyBroadcastR.class);
         PendingIntent notif_PI = PendingIntent.getBroadcast(this, 0, notif_intent,
PendingIntent.FLAG_ONE_SHOT);
         am.cancel(notif_PI);

         Log.d(TAG, "Scheduler Alarm and Notifications Canceled");
      }
   }


   @Override
   public void onDestroy() {
      Log.d(TAG, "onDestroy");
      }


   @Override
   public synchronized void onStart(Intent intent, int startid) {
      Log.d(TAG, "onStart");
      stopSelf();
      }
}
```

C.10 PsySlugActivity.java

```
public class PsyslugActivity extends Activity {
   private static final String TAG = "PsySlugActivity";
   private static final int menuSettings = Menu.FIRST;
   private static final int menuQuit = Menu.FIRST + 1;
   List<EditText> answerFields = new ArrayList<EditText>();
   private Button submit = null;
   private Survey survey = new Survey(this);
   private User user = new User("unknown", this);
   private Button reset = null;
   private AnswerWriter aw;
   // Object = radio group, radio button, text; Integer = question number
   private Map<Integer, List<Object>> responses = new HashMap<Integer,
List<Object>>();

   @Override
   public void onBackPressed () {
      showExitDialog();
   }

   private class ButtonSubmitHandler implements View.OnClickListener {
      public void onClick (View v) {
         submitPress();
      }
   }

   private class ButtonResetHandler implements View.OnClickListener {
      public void onClick (View v) {
         resetPress();
      }
   }

   /**
    * This is used to disable the questions, but its not working very well,
    * because once you disable the question it seems to be impossible to enable
    * it. Also there is no way to change the answer to the question that
    * disables other questions Reset also doesn't help. Something is messed up
    * with radio buttons and radio group
    */
   private class RadioButtonHandler implements View.OnClickListener {
      private Context context;

      public RadioButtonHandler (Context context) {
         this.context = context;
      }

      public void onClick (View v) {
         for (Question q : survey.getQuestionList()) {
            if (v.getId() == q.getQuestionNumber()) {
               List<Object> val = responses.get(q.getQuestionNumber());
               RadioButton rb = new RadioButton(context);
               for (Object ob : val) {
                  if (ob.getClass().isInstance(rb)) {
                     rb = (RadioButton) ob;
                     if (rb.isChecked()) {
```

```
                        if (rb.getText().toString()
                           .equals(q.getDisableAnswer())) {
                           for (Integer questionNumber : q
                              .getDisabledQuestions()) {
                              disableQuestion(questionNumber);
                              Log.d(TAG, "Disabled Question: "
                                 + questionNumber);
                           }
                        }
                     }
                     if (rb.getText().toString()
                        .equals(q.getDisableAnswer())) {
                        if (!rb.isChecked()) {
                           for (Integer questionNumber : q
                              .getDisabledQuestions()) {
                              enableQuestion(questionNumber);
                              Log.d(TAG, "Enabled Question: "
                                 + questionNumber);
                           }
                        }
                     }
                  }
               }
            }
         }
      }
   }

   @Override
   public void onCreate (Bundle savedInstanceState) {
      survey.loadQuestions(this);

      LinearLayout layout = new LinearLayout(this);
      layout.setOrientation(LinearLayout.VERTICAL);

      for (int i = 0; i < survey.getQuestionList().size(); ++i) {
         TextView tv = new TextView(this);
         tv.setTextSize(20);
         tv.setText(survey.getQuestionList().get(i).getText());

         // Input field is created and added to list for every question in
         // order to track which field goes to which question.
         // If it is not needed for that question it is not displayed.
         EditText input = new EditText(this);
         answerFields.add(input);
         addToLayout(layout, tv);

         if (survey.getQuestionList().get(i).getType()
            .contentEquals("multiple choice")) {

            List<Object> rbList = new ArrayList<Object>();

            for (Answer A : survey.getQuestionList().get(i)
               .getAnswerList()) {
```

```
            RadioButton rb = new RadioButton(this);
            rb.setText(A.getAnswer());
            rb.setTextSize(18);
            // Uncomment this for question disabling
            // set radio button id = its question number
            // rb.setId(survey.getQuestionList().get(i).getQuestionNumber());
            // rb.setOnClickListener(new RadioButtonHandler(this));
            addToLayout(layout, rb);
            rbList.add(rb);

            if (A.getType().contentEquals("text")) {
               addToLayout(layout, answerFields.get(i));
               rbList.add(answerFields.get(i));
            }
         }
         responses.put(survey.getQuestionList().get(i)
            .getQuestionNumber(), rbList);

      } else if (survey.getQuestionList().get(i).getType()
         .contentEquals("text")) {

         addToLayout(layout, answerFields.get(i));

         List<Object> textList = new ArrayList<Object>();
         textList.add(answerFields.get(i));

         responses.put(survey.getQuestionList().get(i)
            .getQuestionNumber(), textList);

      } else if (survey.getQuestionList().get(i).getType()
         .equals("single choice")) {
         RadioGroup rg = new RadioGroup(this);
         List<Object> rbList = new ArrayList<Object>();

         for (Answer A : survey.getQuestionList().get(i)
            .getAnswerList()) {
            RadioButton rb = new RadioButton(this);
            rb.setText(A.getAnswer());
            rb.setTextSize(18);
            // Uncomment this for question disabling
            // int question =
            // survey.getQuestionList().get(i).getQuestionNumber();
            // rb.setId(question);
            // rb.setOnClickListener(new RadioButtonHandler(this));
            rg.addView(rb);
            rbList.add(rb);

            if (A.getType().contentEquals("text")) {
               addToLayout(layout, answerFields.get(i));
            }
         }
         responses.put(survey.getQuestionList().get(i)
            .getQuestionNumber(), rbList);
         addToLayout(layout, rg);
      }
```

```
    }

    super.onCreate(savedInstanceState);

    reset = new Button(this);
    submit = new Button(this);
    submit.setText("Submit");
    reset.setText("Clear all answers");
    ScrollView sv = new ScrollView(this);

    addToLayout(layout, submit);
    addToLayout(layout, reset);

    submit.setOnClickListener(new ButtonSubmitHandler());
    reset.setOnClickListener(new ButtonResetHandler());

    sv.addView(layout);
    setContentView(sv);
}

public void submitPress () {
    showConfirmedDialog();
}

public void createWorkBook () {
    user.loadUserInfo();
    aw = new AnswerWriter(this, user.getUserID());
    try {
        Log.d(TAG, "writing xls file");
        // aw = new AnswerWriter(this, user.getUserID());
        WritableSheet sheet;
        // sheet = sheet_result;
        if (aw.sheetExists(0)) {
            sheet = aw.getSheet(0);
        } else {
            sheet = aw.createSheet(user.getUserID(), 0);
        }
        aw.createHeader(sheet, survey.getQuestionList());
        aw.newEntry(sheet);
        saveAnswers(aw, sheet);
        aw.borderEmptyCells(sheet);
        aw.writeWorkBook();
        aw.closeWorkBook();
        aw.replaceXLS(this, user.getUserID());
    } catch (Exception e) {
        Log.d(TAG, "Error: " + e);
        e.printStackTrace();
    }
}

public void resetPress () {
    for (Entry<Integer, List<Object>> entry : responses.entrySet()) {

        List<Object> val = entry.getValue();
        RadioButton rb = new RadioButton(this);
```

```
      EditText et = new EditText(this);

      for (Object ob : val) {
         if (ob.getClass().isInstance(rb)) {
            rb = (RadioButton) ob;
            if (rb.isChecked()) {
               rb.setChecked(false);
               rb.setEnabled(true);
               rb.setClickable(true);
            }
         } else if (ob.getClass().isInstance(et)) {
            et = (EditText) ob;
            if (et.getText().toString().length() > 0) {
               et.setText("");
               et.setEnabled(true);
            }
         }
      }
   }
}

// WritableSheet sheet = null;
public void showConfirmedDialog () {
   AlertDialog.Builder builder = new AlertDialog.Builder(this);
   builder.setTitle("Submit Survey?");
   builder
      .setMessage("Are you sure you want to submit your responses?");
   builder.setPositiveButton("Yes",
      new DialogInterface.OnClickListener() {

         public void onClick (DialogInterface dialog, int which) {
            Toast.makeText(getApplicationContext(),
               "Thank you for completing our survey!",
               Toast.LENGTH_LONG).show();

            createWorkBook();

            startServices();
            PsyslugActivity.this.finish();
         }
      }).setNegativeButton("No",
      new DialogInterface.OnClickListener() {
         public void onClick (DialogInterface dialog, int which) {
            dialog.cancel();
         }
      });
   AlertDialog alert = builder.create();
   alert.show();
}

public void showExitDialog () {
   AlertDialog.Builder builder = new AlertDialog.Builder(this);
   builder.setTitle("Quit?");
   builder
      .setMessage("WARNING: The survey will be marked as missed.\nAre you sure you
```

```
want to quit?");
      builder.setPositiveButton("Yes",
         new DialogInterface.OnClickListener() {

            public void onClick (DialogInterface dialog, int which) {
               if (user.loadUserInfo()) {
                  aw = new AnswerWriter(getBaseContext(),
                     user.getUserID());
                  try {
                     Log.d(TAG, "Writing empty answers to xls");
                     WritableSheet sheet;
                     if (aw.sheetExists(0)) {
                        sheet = aw.getSheet(0);
                     } else {
                        sheet = aw.createSheet(user.getUserID(), 0);
                     }
                     aw.createHeader(sheet, survey.getQuestionList());
                     aw.newEntry(sheet);
                     aw.borderEmptyCells(sheet);
                     aw.writeWorkBook();
                     aw.closeWorkBook();
                     aw.replaceXLS(getBaseContext(), user.getUserID());
                  } catch (Exception e) {
                     Log.d(TAG, "Error: " + e);
                     e.printStackTrace();
                  }
               }
               closeActivity();
            }
         }).setNegativeButton("No",
         new DialogInterface.OnClickListener() {
            public void onClick (DialogInterface dialog, int which) {
               dialog.cancel();
            }
         });
      AlertDialog alert = builder.create();
      alert.show();

   }

   public void closeActivity () {
      this.finish();
   }

   public void saveAnswers (AnswerWriter aw, WritableSheet sheet) {

      Map<Integer, List<Object>> sortedResponses = new TreeMap<Integer, List<Object>>(
         responses);
      for (Map.Entry<Integer, List<Object>> entry : sortedResponses
         .entrySet()) {
         List<Object> val = entry.getValue();
         int key = entry.getKey();
         RadioButton rb = new RadioButton(this);
         EditText et = new EditText(this);
```

```
      for (Object ob : val) { // going through all the answer options in
         // this question
         if (ob.getClass().isInstance(rb)) {
            rb = (RadioButton) ob;
            if (rb.isChecked()) {
               if (rb.getText().toString().equals("Other:")) {
                  continue;
               } else {
                  // answers += "Question Number," + key + " \n"
                  // + "Answer," + rb.getText().toString()
                  // + "\n";
                  try {
                     // true for multiple choice
                     aw.inputAnswer(sheet, Integer.toString(key), rb
                        .getText().toString(), true);
                  } catch (WriteException e) {
                     e.printStackTrace();
                  }
               }
            }
         }

         if (ob.getClass().isInstance(et)) {
            et = (EditText) ob;
            if (et.getText().toString().length() > 0) {
               // answers += "Question Number," + key + " \n" + "Answer,"
               // + et.getText().toString() + "\n";
               try {
                  // false for non-multiple choice
                  aw.inputAnswer(sheet, Integer.toString(key), et
                     .getText().toString(), false);
               } catch (WriteException e) {
                  e.printStackTrace();
               }
            }
         }

      }
   }

}

/*
 * public void saveAnswersToFile(String answer) { FileOutputStream fos =
 * null; try { fos = openFileOutput("answers.csv", MODE_APPEND);
 * fos.write(answer.getBytes()); } catch (IOException ex) { Log.d(TAG,
 * "Error writing to file" + ex); } finally { try { fos.close(); } catch
 * (IOException e) { Log.d(TAG, "Error closing file" + e); } } }
 */
public boolean onCreateOptionsMenu (Menu menu) {
   menu.add(0, menuSettings, 0, "Settings");
   menu.add(0, menuQuit, 0, "Quit");
   return true;
}
```

```
public boolean onOptionsItemSelected (MenuItem item) {
   switch (item.getItemId()) {
      case menuSettings:
         startActivity(new Intent(this, SettingsActivity.class));
         // this.finish(); //should we close psyslugactivity?
         return true;
      case menuQuit:
         showExitDialog();
         return true;
   }
   return false;
}

public void startServices () {
   startService(new Intent(this, PsySlugDropboxService.class));
}

/**
 * Adds items to the layout
 *
 * @param layout
 * @param child
 */
public void addToLayout (LinearLayout layout, View child) {
   layout.addView(child, new LayoutParams(LayoutParams.FILL_PARENT,
      LayoutParams.WRAP_CONTENT));
}

/**
 * Disables the question. Takes in a question number to disable
 *
 * @param questionNumber
 */
public void disableQuestion (int questionNumber) {
   for (Entry<Integer, List<Object>> entry : responses.entrySet()) {

      int questionNum = entry.getKey();
      List<Object> val = entry.getValue();
      RadioButton rb = new RadioButton(this);
      EditText et = new EditText(this);
      if (questionNum == questionNumber) {
         for (Object ob : val) {
            if (ob.getClass().isInstance(rb)) {
               rb = (RadioButton) ob;
               rb.setClickable(false);
               rb.setEnabled(false);
            } else if (ob.getClass().isInstance(et)) {
               et = (EditText) ob;
               et.setEnabled(false);
            }
         }
      }
   }
}
```

```
   /**
    * Should enable the question but doesn't work for some reason. Code works,
    * but no changes are seen on the screen
    *
    * @param questionNumber
    */
   public void enableQuestion (int questionNumber) {
      for (Entry<Integer, List<Object>> entry : responses.entrySet()) {

         int questionNum = entry.getKey();
         List<Object> val = entry.getValue();
         RadioButton rb = new RadioButton(this);
         EditText et = new EditText(this);
         if (questionNum == questionNumber) {
            for (Object ob : val) {
               if (ob.getClass().isInstance(rb)) {
                  rb = (RadioButton) ob;
                  rb.setClickable(true);
                  rb.setEnabled(true);
               } else if (ob.getClass().isInstance(et)) {
                  et = (EditText) ob;
                  et.setEnabled(true);
               }
            }
         }
      }
   }
}
```

C.11 Question.java

```
public class Question {
	private int number;
	private String text; //Question text->The question itself.
	private String type; // "multiple choice" or "text"
	private List<Answer> answerList; //A list with answer choices for this question
	private List<Integer> disabledQuestions;
	private String disableAnswer = "";

	public Question(){
		answerList = new ArrayList<Answer>();
		number = 0;
		text = "";
		type = "unknown";
		disabledQuestions = new ArrayList<Integer>();
	}
	/**
	 * public Constructor
	 * @param number
	 * @param text
	 * @param type
	 */
	public Question(int number, String text, String type){
		this.number = number;
		this.text = text;
```

```
        this.type = type;
        answerList = new ArrayList<Answer>();
        disabledQuestions = new ArrayList<Integer>();
}

public int getQuestionNumber() {
        return number;
}

public void setQuestionNumber(int number) {
        this.number = number;
}

public void setText(String text) {
        this.text = text;
}

public String getText() {
        return text;
}

public void setType(String type) {
        this.type = type;
}

public String getType() {
        return type;
}

public List<Answer> getAnswerList() {
        return answerList;
}
/**
 * Deletes everything from the answer list.
 */
public void clearAnswerList(){
        answerList.clear();
}
/**
 * Adds an answer element to the answerList.
 * @param a
 */
public void addAnswer(Answer answer){
        answerList.add(answer);
}
public List<Integer> getDisabledQuestions() {
        return disabledQuestions;
}
public void setDisableAnswer(String disableAnswer) {
        this.disableAnswer = disableAnswer;
}
public String getDisableAnswer() {
        return disableAnswer;
}
public void addDisabledQuestion(int questionNumber){
```

```
            disabledQuestions.add(questionNumber);
        }
        public void reset() {
            number = 0;
            text = "";
            type = "";
            answerList.clear();
            disabledQuestions.clear();
            disableAnswer = "";
        }

}
```

C.12 ScheduleAlarmBR.java

```
//Schedules the MAX_ALARMS alarms per day.
//One alarm in every TIME_INTERVAL_SEC block, starting at
START_TIME_HOUR:START_TIME_MINUTE
//Each alarm is at least MIN_SECONDS_ALARMS seconds away from each other

//Currently, 7 alarms set one per 2 hour block at 45 minutes minimum apart

//Please note this code is similar to BootBroadcastR as both do similar things at
different phone states
public class ScheduleAlarmBR extends BroadcastReceiver {

   private AlarmManager [] am;
   private int [] numbers;
   public static final int MAX_ALARMS = 7;
   public static final int START_TIME_HOUR = 7;   //Starting hour for experiment (USE
24-hour clock)
   public static final int START_TIME_MINUTE = 0;   //Starting minute
   public static final int START_TIME_SECOND = 0;
   public static final int MAX_TIME = 54000;      //54000s, or 15 hours (7am to 11pm)
   public static final int MIN_SECONDS_ALARMS = 1800; //minimum seconds between each
alarm (30 minutes)
   public static final int TIME_INTERVAL_SEC = 7200;   //time blocks for alarms (1
alarm in each 2 hour block)
   public static final int MILLI = 1000;
   public static final String saBRTAG = "ScheduleAlarmBR";

    @Override
    public void onReceive(Context context, Intent intent) {
       Log.d(saBRTAG, "ScheduleAlarmBR Started");

      Calendar cur_cal = new GregorianCalendar();
      cur_cal.setTimeInMillis(System.currentTimeMillis());//set the current time and
date for this calendar


      Calendar cal = new GregorianCalendar();
      cal.set(Calendar.HOUR_OF_DAY, START_TIME_HOUR);
      cal.set(Calendar.MINUTE, START_TIME_MINUTE);
      cal.set(Calendar.SECOND, START_TIME_SECOND);

       //cal is set to 7am of current day

      numbers = new int[MAX_ALARMS];
      generateRandomTimes();
      boolean timeCheck = true;
      boolean randomCheck = false;
```

```
      while(timeCheck)
      {
         timeCheck = false;
         for(int i = 0; i < MAX_ALARMS - 1; ++i)   //Look through each number
         {
            if((numbers[i] - numbers[i+1]) > MIN_SECONDS_ALARMS)   //if numbers are
not spread apart enough, repeat random gen and check
            {
               timeCheck = true;
               randomCheck = true;
            }
         }
         if(randomCheck)   //if true, randomize times again
         {
            generateRandomTimes();
            randomCheck = false;
         }
      }

      saveToFile(context);


      am =new AlarmManager[MAX_ALARMS];
      for(int i = 0; i < MAX_ALARMS; ++i)
      {
         Intent notif_intent = new Intent(context, psyBroadcastR.class);
         PendingIntent pendingIntent = PendingIntent.getBroadcast(context, i,
notif_intent, PendingIntent.FLAG_ONE_SHOT);
         am[i] = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);

         //checks current time compared to randomized times
         //(i + TIME_INTERVAL_SEC) is to set the alarm relative to calendar time +
interval
         if(System.currentTimeMillis() <= (cal.getTimeInMillis() + (i *
TIME_INTERVAL_SEC * MILLI) + (numbers[i] * MILLI) ))
         {
            am[i].set(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis() + (i *
TIME_INTERVAL_SEC * MILLI) + (numbers[i] * MILLI), pendingIntent);
            Date d = new Date(cal.getTimeInMillis() + (i * TIME_INTERVAL_SEC * MILLI)
+ (numbers[i] * MILLI));
            Log.d(saBRTAG, "Setting alarm at " + d);
         }
         else
         {
            Date d = new Date(cal.getTimeInMillis() + (i * TIME_INTERVAL_SEC * MILLI)
+ (numbers[i] * MILLI));
            Log.d(saBRTAG, "Alarm for " + d + " was not set");
         }

      }

    }

    private void generateRandomTimes(){
       Random r = new Random();

       for(int i = 0; i < MAX_ALARMS; ++i)
       {
          numbers[i] = r.nextInt(TIME_INTERVAL_SEC) + 1;
       }
       //Arrays.sort(numbers);
    }
```

```
    private void saveToFile(Context context){
      FileOutputStream fos = null;
      try
      {
         //TODO change to MODE_PRIVATE later so user can't see or mess with the file
         fos = context.openFileOutput("alarms.txt", Context.MODE_WORLD_READABLE);
         Log.d(saBRTAG, "Writing alarm schedule to alarms.txt in: " +
context.getFilesDir());
      }
      catch(IOException ex)
      {
         Log.d(saBRTAG, "Error opening file" + ex);
      }

      PrintStream ps = new PrintStream(fos);

      for(int i = 0; i < MAX_ALARMS; ++i)
      {
         ps.print(numbers[i] + ",");
      }
      try
      {
         fos.close();
      }
      catch (IOException e)
      {
         Log.d(saBRTAG, "Error closing file" + e);
      }
    }
}
```

C.13 SettingsActivity.java

```
public class SettingsActivity extends Activity {
   /** Called when the activity is first created. */
   private static final int menuBack = Menu.FIRST;
   private Button downloadSurvey = null;
   private Button changePass = null;
   private Button changeId = null;
   private CheckBox notification = null;
   private CheckBox disableLockScreen = null;

   private static final String TAG = "SettingsActivity";
   private User user;

   private class ButtonChangePassword implements View.OnClickListener {
      public void onClick(View v) {
         showDropBoxDialogPass();
      }
   }
   private class NotificationCheckBox implements View.OnClickListener {
      public void onClick(View v) {
         if(notification.isChecked()){
            user.setNotifications("false");
            startServices();
         }
         else{
            user.setNotifications("true");
```

```
            startServices();
         }
      }
   }
   private class AccessCheckBox implements View.OnClickListener {
      public void onClick(View v) {
         if(disableLockScreen.isChecked()){
            user.setAccess("true");
         }
         else{
            user.setAccess("false");
         }
      }
   }

   private class ButtonChangeId implements View.OnClickListener {
      public void onClick(View v) {
         showUserID();
      }
   }

   private class ButtondownloadSurveyHandler implements View.OnClickListener {
      public void onClick(View v) {
         downloadSurveyPress(user);
      }

      /**
       * Goes to Dropbox and downloads new file but checks if directory
       * already exists on the phone
       */
      private void downloadSurveyPress(User user) {
         if(user.getDropBoxuser().equals("unknown") ||
user.getDropboxPassword().equals("unknown")){
            Toast.makeText(getApplicationContext(), "Enter your Dropbox account info
before downloading survey!",
                  Toast.LENGTH_LONG).show();
            return;
         }
         File surveyFolder = new File(getFilesDir() + "/Survey");
         Log.d(TAG, "SurveyPath: " + surveyFolder.getAbsolutePath());
         if (surveyFolder.exists()) {
            // Delete everything first
            for (File f : surveyFolder.listFiles()) {
               if (f.isFile()) {
                  Log.d(TAG, "Deleting: " + f.getName());
                  f.delete();
               }
            }
            Log.d(TAG, "Survey folder already exists");
            Intent update = new Intent(
                  "android.psyslug.PsySlugDropboxService");
            update.putExtra("surveyUpdate", "true");
            startService(update);
         } else {
            if (surveyFolder.mkdir()) {
```

```
            Log.d(TAG, "Created Survey Folder");
            Intent update = new Intent(
                  "android.psyslug.PsySlugDropboxService");
            update.putExtra("surveyUpdate", "true");
            startService(update);
         }
      }
   }
}

@Override
public void onCreate(Bundle savedInstanceState) {
   user = new User("unknown", this); // id is unknown
   user.loadUserInfo();
   if(user.getSettingspass().equals("unknown")){showConfirmedDialog();   }
   else{showPasswordWindow();}

   super.onCreate(savedInstanceState);

   disableLockScreen = new CheckBox(this);
   disableLockScreen.setText("Access the app at any time");

   if(user.getAccess().equals("false")){
      disableLockScreen.setChecked(false);
   }
   else{
      disableLockScreen.setChecked(true);
   }
   notification = new CheckBox(this);
   notification.setText("Stop Notifications");
   if(user.getNotifications().equals("false")){
      notification.setChecked(true);
   }
   else{
      notification.setChecked(false);
   }
   disableLockScreen.setOnClickListener(new AccessCheckBox ());
   notification.setOnClickListener(new NotificationCheckBox ());

   downloadSurvey = new Button(this);
   downloadSurvey.setText("Download Survey");
   downloadSurvey.setWidth(200);
   downloadSurvey.setHeight(10);
   downloadSurvey.setGravity(Gravity.CENTER);

   changePass = new Button(this);
   changePass.setText("Set Dropbox Account Info");
   changePass.setWidth(300);
   changePass.setHeight(10);
   changePass.setGravity(Gravity.CENTER);

   changeId = new Button(this);
   changeId.setText("Change User ID ");
   changeId.setWidth(200);
   changeId.setHeight(10);
```

```
    changeId.setGravity(Gravity.CENTER);

    LinearLayout layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);

    layout.addView(disableLockScreen, new LayoutParams(LayoutParams.WRAP_CONTENT,
          LayoutParams.WRAP_CONTENT));
    layout.addView(notification, new LayoutParams(LayoutParams.WRAP_CONTENT,
          LayoutParams.WRAP_CONTENT));

    layout.addView(changePass, new LayoutParams(LayoutParams.WRAP_CONTENT,
          LayoutParams.WRAP_CONTENT));
    changePass.setOnClickListener(new ButtonChangePassword());

    layout.addView(changeId, new LayoutParams(LayoutParams.WRAP_CONTENT,
          LayoutParams.WRAP_CONTENT));
    changeId.setOnClickListener(new ButtonChangeId());

    layout.addView(downloadSurvey, new LayoutParams(
          LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
    downloadSurvey.setOnClickListener(new ButtondownloadSurveyHandler());

    setContentView(layout);

}

public void showConfirmedDialog() {
    final AlertDialog.Builder builder = new AlertDialog.Builder(this);
    LinearLayout layout = new LinearLayout(this);

    builder.setTitle("Create Settings Page Password");

    final EditText pass = new EditText(this);
    final EditText confirmpass = new EditText(this);
    pass.setTransformationMethod(new PasswordTransformationMethod());
    confirmpass.setTransformationMethod(new PasswordTransformationMethod());

    TextView confirmPass = new TextView(this);
    TextView passw = new TextView(this);
    passw.setText("Enter Password:");
    confirmPass.setText("Confirm Password:");

    layout.setOrientation(LinearLayout.VERTICAL);
    layout.addView(passw, new LayoutParams(LayoutParams.FILL_PARENT,
          LayoutParams.WRAP_CONTENT));
    layout.addView(pass, new LayoutParams(LayoutParams.FILL_PARENT,
          LayoutParams.WRAP_CONTENT));
    layout.addView(confirmPass, new LayoutParams(LayoutParams.FILL_PARENT,
          LayoutParams.WRAP_CONTENT));
    layout.addView(confirmpass, new LayoutParams(LayoutParams.FILL_PARENT,
          LayoutParams.WRAP_CONTENT));
    builder.setView(layout);

    builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int which) {
```

```
            if (pass.getText().toString()
                  .equals(confirmpass.getText().toString())) {
               Toast.makeText(getApplicationContext(), "Thank you!",
                     Toast.LENGTH_LONG).show();
               user.setSettingspass(pass.getText().toString());
               user.storeUserInfo();
            } else {
               Toast.makeText(getApplicationContext(),
                     "The password you entered do not match.",
                     Toast.LENGTH_LONG).show();
               showConfirmedDialog();
            }
         }
      }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
            closeSettings();
         }
      });
      AlertDialog alert = builder.create();
      alert.show();
   }

   public void showDropBoxDialogPass() {
      final AlertDialog.Builder builder = new AlertDialog.Builder(this);
      LinearLayout layout = new LinearLayout(this);

      builder.setTitle("Dropbox Account Info");

      final EditText userName = new EditText(this);
      final EditText password = new EditText(this);
      if(!user.getDropBoxuser().equals("unknown") ||
!user.getDropBoxuser().equals("unknown")){
         userName.setText(user.getDropBoxuser());
         password.setText(user.getDropboxPassword());
      }

      TextView text2 = new TextView(this);
      TextView text = new TextView(this);
      text.setText("Enter Dropbox User ID:");
      text2.setText("Enter Dropbox Password:");

      layout.setOrientation(LinearLayout.VERTICAL);
      layout.addView(text, new LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT));
      layout.addView(userName, new LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT));
      layout.addView(text2, new LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT));
      layout.addView(password, new LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT));
      builder.setView(layout);

      builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
```

```
      public void onClick(DialogInterface dialog, int which) {

         Toast.makeText(getApplicationContext(), "Thank you!",
               Toast.LENGTH_LONG).show();
         user.setDropBoxuser(userName.getText().toString());
         user.setDropboxPassword(password.getText().toString());
         user.storeUserInfo();
      }

   }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
         dialog.cancel();
      }
   });
   AlertDialog alert = builder.create();
   alert.show();
}

public void showUserID() {
   final AlertDialog.Builder builder = new AlertDialog.Builder(this);
   LinearLayout layout = new LinearLayout(this);

   builder.setTitle("Change User ID");

   final EditText userId = new EditText(this);
   userId.setText(user.getUserID());

   TextView writeId = new TextView(this);
   writeId.setText("Enter new User ID:");

   layout.setOrientation(LinearLayout.VERTICAL);
   layout.addView(writeId, new LayoutParams(LayoutParams.FILL_PARENT,
         LayoutParams.WRAP_CONTENT));
   layout.addView(userId, new LayoutParams(LayoutParams.FILL_PARENT,
         LayoutParams.WRAP_CONTENT));

   builder.setView(layout);

   builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {

         user.setUserID(userId.getText().toString());
         user.storeUserInfo();

      }
   }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
         dialog.cancel();
      }
   });
   AlertDialog alert = builder.create();
   alert.show();
}

public void showPasswordWindow() {
```

```
      final AlertDialog.Builder builder = new AlertDialog.Builder(this);
      LinearLayout layout = new LinearLayout(this);

      builder.setTitle("Settings");

      final EditText password = new EditText(this);
      password.setTransformationMethod(new PasswordTransformationMethod());

      TextView writeId = new TextView(this);
      writeId.setText("Enter Settings Page Password:");

      layout.setOrientation(LinearLayout.VERTICAL);
      layout.addView(writeId, new LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT));
      layout.addView(password, new LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT));
      builder.setView(layout);

      builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int which) {

            if(!user.getSettingspass().equals(password.getText().toString())){
               Toast.makeText(getApplicationContext(), "Incorrect Password, Please try
again!",
                     Toast.LENGTH_LONG).show();
               showPasswordWindow();
            }

         }
      }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
            closeSettings();
         }
      });
      AlertDialog alert = builder.create();
      alert.show();
   }

   public void closeSettings() {
      this.finish();
   }
   public void startServices(){
      startService(new Intent(this, PsySlugIdleService.class));
   }

   /**
    * Creates menu buttons
    */
   public boolean onCreateOptionsMenu(Menu menu) {
      menu.add(0, menuBack, 0, "Back");
      return true;
   }

   /**
```

```
    * Processes menu button presses
    */
   public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case menuBack:
         finish();
         return true;
      }
      return false;
   }
}
```

C.14 Survey.java

```
public class Survey
{
   private List<Question> listQuestions; //added
   Context fileContext;
   private static final String TAG = "Survey";

   public Survey(Context fileContext){
      listQuestions = new ArrayList<Question>();
      this.fileContext = fileContext;
   }
   /**
    * New loading function to support disabling of questions
    * New question format:
    *  Question number:
    *   2
    *  Disables questions:
    *  3
    *  4
    *  5
    *  Disable answer:
    *  Yes
    *   Question text:
    *   2. Were you alone?
    *   Answer type:
    *   single choice
    *   Answers:
    *   Yes
    *   No
    *   End of Answer
    * @param app
    */
   public void loadQuestions(Activity app){
       /* the input text file is stored into /data/data/android.psyslug/files/Survey/
       */
        File surveyDir = new File(fileContext.getFilesDir() + "/Survey");

        if(!surveyDir.exists()){
           return;
        }
        File file = null;
```

```
        //temp for test
        //file = new File("/data/data/android.psyslug/files/Survey.txt");

        /* there should be only one file*/
      for(File f : surveyDir.listFiles()) {
         file = new File(f.getAbsolutePath());
      }

       BufferedReader br = null;
       if(file == null){
          return;
       }
       try{
          br = new BufferedReader(new FileReader(file));
             String line;
             Question temp  = new Question(0,"","");
             while((line = br.readLine()) != null){
                if(line.equals("Question number:")){
                   try{
                      int questionNum = Integer.parseInt(br.readLine());
                      temp.setQuestionNumber(questionNum);
                   }
                   catch(NumberFormatException e){
                      Log.d(TAG, "Error getting question number:" + e);
                   }
                  }
                else if(line.equals("Disables questions:")){
                     int questionNumber = 0;
                     line = br.readLine();
                     while (!line.equals("Disable answer:")) {
                        try {
                           questionNumber = Integer.parseInt(line);
                           temp.addDisabledQuestion(questionNumber);
                        } catch (NumberFormatException e) {
                           Log.d(TAG, "Error converting disabled question number: " +
e);

                        }
                        line = br.readLine();
                     }
                     //get disabled answer
                     if(line.equals("Disable answer:")){
                        temp.setDisableAnswer(br.readLine());
                     }
                   }
                else if(line.equals("Question text:") || line.equals("Question
text")){
                   temp.setText(br.readLine());
                }
                else if(line.equals("Answer type:") || line.equals("Answer type")){
                   line = br.readLine();
                   temp.setType(line);
                   if(line.equals("text")) {
                      Answer answer = new Answer("text field", "no text");
                      temp.getAnswerList().add(answer);
```

```
                }
            }
            else if(line.equals("Answers:") || line.equals("Answers")){
                    line = br.readLine();
                    while (!line.equals("End of Answer")){
                       Answer answer = new Answer("","");
                       answer.setAnswer(line);
                       if (line.equals("Other:") || line.equals("Other")){
                          answer.setType("text");
                       }else {
                          answer.setType("button");
                       }
                       temp.getAnswerList().add(answer);
                       line = br.readLine();
                    }
                    if(line.equals("End of Answer")){
                       listQuestions.add(temp);
                       temp  = new Question(0,"","");
                    }
            }
            else if(line.equals("End of Answer")){
                 listQuestions.add(temp);
                 temp  = new Question(0,"","");
            }
          }
         br.close();
    } catch(IOException e){
       Log.d(TAG, "Error while reading survey file: " + e);
    }
    finally{
       try {
         br.close();
      } catch (IOException e) {
         Log.d(TAG, "Error closing survey file: " + e);
      }
    }
}
public void loadQuestionsOld(Activity app){
    /* the input text file is stored into /data/data/android.psyslug/files/Survey/
    */
     File surveyDir = new File(fileContext.getFilesDir() + "/Survey");

     if(!surveyDir.exists()){
        return;
     }
     File file = null;
     /* there should be only one file*/
   for(File f : surveyDir.listFiles()) {
      file = new File(f.getAbsolutePath());
   }
    InputStream in = null;
    if(file == null){
       return;
    }
```

```
         //still have to add exceptions to give out errors for when the user loads in
wrongly formatted questions
       try{
          in = new BufferedInputStream(new FileInputStream(file));
          Scanner scanner = new Scanner(in);
            while (scanner.hasNextLine()) {
               String line = scanner.nextLine();
               Question temp = new Question(0,"","");

               if(line.equals("Question number:")){
                  temp.setQuestionNumber(scanner.nextInt());
                  scanner.nextLine();
               }

               line = scanner.nextLine();
               if(line.equals("Question text:")){
                  String text = scanner.nextLine();
                  temp.setText(text);
               }

               line = scanner.nextLine();
               if(line.equals("Answer type:")){
                  temp.setType(scanner.nextLine());
               }

               line = scanner.nextLine();
               if(line.equals("Answers:")){
                  line = scanner.nextLine();
                  while (!line.equals("End of Answer")){
                     Answer answer = new Answer("","");
                     answer.setAnswer(line);
                     if (line.equals("Other:")){
                        answer.setType("text");
                     }else {
                        answer.setType("button");
                     }
                     temp.getAnswerList().add(answer);
                     line = scanner.nextLine();
                  }
               }
            listQuestions.add(temp);
          }
           in.close();
       } catch(IOException e){
          Log.d(TAG, "Error loading questions: " + e);
          e.printStackTrace();
       }
   }

   public void clearQuestions(){
      listQuestions.clear();
   }
   public void addQuestion(Question q){
      listQuestions.add(q);
   }
```

```
   public List<String> getQuestions(){
      List<String> strings = new ArrayList<String>();
      Iterator<Question> itr = listQuestions.iterator();
      while(itr.hasNext()){
         strings.add(itr.next().getText());
      }
      return strings;
   }

   public List<Question> getQuestionList(){
      return listQuestions;
   }
}
```

C.15 User.java

```
public class User
{
   Context fileContext;
   private static final String userTAG = "User";
   private String userID;
   private String Settingspass = "unknown";
   private String DropBoxuser = "unknown";
   private String DropboxPassword = "unknown";
   private String notifications = "true";
   private String access = "false"; //determines if survey can be taken at any time

   public User(String userID, Context c){
      this.setUserID(userID);
      fileContext = c;
   }
   /**
    * checks if the user file exists and loads user info/settings
    * on app startup
    * if the file exists returns true, if not false
    * @return
    */
   public boolean loadUserInfo() {
      File path = fileContext.getFilesDir();
      Log.d(userTAG, "Loading user file from: " + path);

      for (File f : path.listFiles()) {
         if (f.isFile()) {
            if (f.getName().equals("user.txt")) {
               readUserFile(path + "/user.txt");
               Log.d(userTAG, "Loaded user file");
               return true;
            }
         }
      }
      Log.d(userTAG, "No user file found");
      return false;
   }
   /**
    * Returns a string with user id
    * @param filename
    * @return
    */
   public void readUserFile(String filename){
      BufferedReader br;
```

```
      String strLine = "";

      try {
         br = new BufferedReader(new FileReader(filename));
         while ((strLine = br.readLine()) != null) {
            if (strLine.equals("User ID:")) {
               strLine = br.readLine();
               userID = strLine;
            } else if (strLine.equals("Settings Page Password:")) {
               strLine = br.readLine();
               Settingspass = strLine;
            } else if (strLine.equals("Dropbox password:")) {
               strLine = br.readLine();
               DropboxPassword = strLine;
            } else if (strLine.equals("Dropbox user name:")) {
               strLine = br.readLine();
               DropBoxuser = strLine;
            } else if (strLine.equals("Notifications:")) {
               strLine = br.readLine();
               notifications = strLine;
            } else if(strLine.equals("App access:")){
               strLine = br.readLine();
               access = strLine;
            } else {
               Log.d(userTAG, "Error in the user file: info is missing");
            }
         }
         br.close();
      } catch (FileNotFoundException ex) {
         Log.d(userTAG, "Error loading user.txt file: " + ex);
      } catch (IOException ex) {
         Log.d(userTAG, "Error loading user.txt file: " + ex);
      }
   }
   public void storeUserInfo()
   {
      FileOutputStream fos = null;
      String newline = "\n";
      try   {
         Log.d(userTAG, "Writing user info to user.txt in: " +
fileContext.getFilesDir());
         fos = fileContext.openFileOutput("user.txt", Context.MODE_PRIVATE);
         fos.write("User ID:".getBytes());
         fos.write(newline.getBytes());
         fos.write(userID.getBytes());
         fos.write(newline.getBytes());

         fos.write("Settings Page Password:".getBytes());
         fos.write(newline.getBytes());
         fos.write(Settingspass.getBytes());
         fos.write(newline.getBytes());

         fos.write("Dropbox user name:".getBytes());
         fos.write(newline.getBytes());
         fos.write(DropBoxuser.getBytes());
         fos.write(newline.getBytes());

         fos.write("Dropbox password:".getBytes());
         fos.write(newline.getBytes());
         fos.write(DropboxPassword.getBytes());
         fos.write(newline.getBytes());

         fos.write("Notifications:".getBytes());
```

```
            fos.write(newline.getBytes());
            fos.write(notifications.getBytes());
            fos.write(newline.getBytes());

            fos.write("App access:".getBytes());
            fos.write(newline.getBytes());
            fos.write(access.getBytes());
            fos.write(newline.getBytes());
         }
         catch (IOException ex){
            Log.d(userTAG, "Error writing to file" + ex);
         }
         finally{
            try{
               fos.close();
            }
            catch (IOException e){
               Log.d(userTAG, "Error closing file" + e);
            }
         }
      }
      public void setNotifications(String n){
         notifications = n;
         storeUserInfo();
      }
      public String getNotifications(){
         return notifications;
      }
      public void setUserID(String userID) {
         this.userID = userID;
      }
      public String getUserID() {
         return userID;
      }
      public void setSettingspass(String settingspass) {
         Settingspass = settingspass;
      }
      public String getSettingspass() {
         return Settingspass;
      }
      public void setDropBoxuser(String dropBoxuser) {
         DropBoxuser = dropBoxuser;
      }
      public String getDropBoxuser() {
         return DropBoxuser;
      }
      public void setDropboxPassword(String dropboxPassword) {
         DropboxPassword = dropboxPassword;
      }
      public String getDropboxPassword() {
         return DropboxPassword;
      }
      public void setAccess(String access) {
         this.access = access;
         storeUserInfo();
      }
      public String getAccess() {
         return access;
      }
   }
```