**Algorithms for Android App**

**Program Launch**

*User* -> Presses on app launcher.

New activity started.

checkNewDatabase()

if (newDatabase == true){

getDatabaseXMLFromServer();}

databaseXML = getDatabaseXMLFromLocal();}

loadGUI();

checkPreviousUser()

if (!user.exists){

newUserPrompt();}

loadUser(), loadDatabase(), newVisit();

The above method describes how the Android application launches. First it checks if there is a new database version available from the web server using the method described below. Then, once it has retrieved the database (assuming there is a new version available) and saved it locally, it loads the GUI and checks if there has been a previous user login.

If there has been no previous user, it prompts for a new user using the method described below. Once done, or if there is a user saved already, it loads the database and prompts the user for a new visit.

**Database Handling**

boolean checkNewDatabase(){

INT webDatabaseVersion = server.databaseVersion

INT currentDatabaseVersion = local.getDatabaseVersion

if (currentDatabaseVersion < webDatabaseVersion){

return true;}

else{

return false;}

This method simply compares the current local database version with the one found on the server. if the server's version is newer (that is, larger than the current version), then it returns True that there is a new version of the database. Else it returns false.

void getDatabaseXMLFromServer(){

FileOutputStream.getChannel().transferFrom(WEBSERVER);}

After the above, this is called, simply downloading the XML file straight to the android handset in the desired location.

**Creating a User**

PRINT "New User Name";

STRING userName = scanner.nextln();

//Repeat the above for phone and email

FileOutputStream.saveXML(userData);

This algorithm saves the user details file to XML for loading later.

**Taking Recording From User**

For the most part the taking of a recording is handled by the GUI, with strings saved from the text input once the user saves the record. What is automatic is the collection of the GPS location, which can be taken as a snapshot using the Android's location service, and the image file switch, which is a simple boolean indicating that at least one picture has been taken, followed by an int showing the number of total pictures taken.

**Saving Input From User**

STRING userEntry; //The options added to the record for this recording.

if (recording.hasPicture == TRUE){

INT pictureCount = recording.pictureCount();}

userEntry = userEntry + pictureCount

FileOutputStream(visitXML) = visitXML + userEntry;

For the most part, the user will be filling in details based on text boxes. What is automatic is the gathering of the current location. The above algorithm takes everything the user has entered, adds the GPS data, sets whether a photograph has been taken or not, then saves it to a new line in the visit's XML file.

**Creating and Sending the MIME Message**

MIME textPart = new MIMEBodyPart();

textPart.setContent (recordData, XML);

MultiPart.addBodyPart(textPart);

MIME imgPart = new MIMEBodyPart();

if (recordData.imageCount > 0){

for(recordData.imageCount){

imgPart.setContent(image);

MultiPart.addBodyPart(imgPart);}}

MIME.setContent(MultiPart);}

The above algorithm shows how the XML text portion of the data will be added to a MIME message, followed by any images recursively. The images must go after the XML due to an operating standard within the MIME protocol.

MIME.setRecipient(server);

MIME.setSubject(recordName);

MIME.setSentDate(new Date());

addToSendStack(MIME);

VOID sendStack(){

while(itemsOnStack){

if(internetConnection = true){

while(internetConnection)

for(itemsOnStack){

Transport.send(MIME);}}

wait(60seconds);

The rest of this algorithm adds the destination information to the MIME message, then adds it to a stack. This is a loop on its own thread (current design) that, if there are files to send, checks if there is an internet connection every 60 seconds. Once there is, it will upload any visit files on the stack until either the internet cuts out, or it runs out of files.