## 1 Introduction

In this document we are going to go over all of the required information deemed important enough to be a noteworthy to someone maintaining or making changes to the program.

### 1.1 Scope

The scope if this document is to provide all the information we have deemed informant to those that are going to be maintaining or making changes to the program.

### 1.2 Objectives

The objectives of this document are as follows:

1.2.1 Program Description   
1.2.2 Program Structure  
1.2.3 Algorithms  
1.2.4 Main Data Areas  
1.2.5 Files  
1.2.6 Interfaces  
1.2.7 Suggested Improvements   
1.2.8 Things to watch for when making changes  
1.2.9 Physical limitations  
1.2.10 Rebuilding and Testing

## 2.1 Program Description - Java

An android application designed to allow the user to record a catalogue of plants, including their associated location (as GPS co-ordinates) and rarity, within nature reserves. The app allows the user to take photos of the plant and surrounding location, before allowing further recordings to be made. Once a set of recordings (corresponding to a specific visit to the nature reserve) is complete, the app uploads the information to a web database once an internet connection is available.

## 2.2 Program Description - Website

The Web App has been created using HTML, CSS and PHP with all the data (recordings, reserves & species) being stored in a MySQL database. The Main home page of the Web App explains the purpose and usage of the website in order to give the user a general overview of its functionality.

The Web App also has a authentication system where an authorized user can log in to the website in order to gain additional functionality if the username and password they have entered is correct. The websites login username and password is stored within variables in the login PHP file.

## 3 Program Structure - Java

This Android application uses a number of modules to gather, manage and send data for sighting records. The list of the modules used is shown below.

**UI modules:**

* **Main Activity,** main screen that will open up when the application is run. Shows data recorded so far if available and allows adding, editing, deleting and sending of data.
* **Log On Dialog Fragment,** modal window that allows the entry of user contact details.
* **Reserve Entry Dialog Fragment,** modal window that allows the entry/identification of the reserve.
* **Sighting Entry Dialog Fragment,** modal window that allows the entry of sighting details including GPS coordinates and location and specimen photos.
* **Sighting List View,** list view that allows scrolling and clicking of a list of sightings in the current recording.
* **Sighting Edit Dialog Fragment,** modal window that allows re-entering sighting details to replace a sighting record.
* **Menu in Main Activity,** menu that holds a number of options to edit user details and reserve and to send the whole recording to the server or deleting it.

**Data modules:**

* **User,** contains user’s contact details.
* **Sighting,** contains details for a sighting.
* **Visit,** contains one user and a number of sightings.

**HTTP modules:**

* **Send To Server,** uses HTTP post to send the recordings to the server.

These modules will be explained in more detail on the following pages.

Main Activity (creation)

This section explains the steps for the creation of the main activity.

main_activity_control_flow

*Diagram 1 (UML Activity diagram showing the control flow of the creation of the main activity)*

An activity can get destroyed after the user presses the back button or the Android OS has unallocated its memory. Therefore the main activity will always try to restore a previous session. The previous session will be stored by overriding the onSaveInstanceState(Bundle savedInstanceState) method which parcels the current data in the application.

If on creation it did not restore a previous session it will not have a visit and therefore needs to create a new one.

If on creation it did have a visit from a previous session, but no user (contact details), it will create a new one. For this a modal window is opened which will be explained next.

Finally it will create an on item click listener to be used for the list view that contains all the sightings of the current visit.

### Log On Dialog Fragment

This section explains how the log on dialog fragment works and how data is passed between it and the main activity.

log_on_dialog_fragment_control_flow

*Diagram 2 (UML Activity diagram showing the control flow of the log on dialog fragment)*

The dialog fragment is created from within the main activity. This means that the main activity does not get destroyed, it stays behind the dialog (modal window).

The main activity implements a listener that will be used by this dialog to call certain methods.

While building the dialog fragment a view gets created which can be passed through the listener methods as a parameter. This view will make it possible for the main activity to access the elements on the dialog fragment.

If the entry is cancelled due to the user pressing “cancel” or tapping outside of the modal window (and therefore closing it), no new user will be created.

After accepting or cancelling this dialog the main activity checks if a reserve has already been identified. If not it will open a new modal window which will be explained next.

### Reserve Entry Dialog Fragment

This section explains how the reserve entry dialog fragment works and how data is passed between it and the main activity.reserve_entry_dialog_fragment_control_flow

*Diagram 3 (UML Activity diagram showing the control flow of the reserve entry dialog fragment)*

This dialog fragment works the same as the log on dialog fragment explained before.

It does contain an autocomplete field to enter the reserve name. This field uses an array of strings which gets loaded in the main activity. A method of the listener in the main activity will be used at the creation of this dialog using the view for the connection between the two.

The main activity will get the reserve names from the strings file build into the application and create an array with them. This array will be given to the autocomplete field in the dialog fragment.

After the reserve has been identified, the main activity will update the text field showing the name of the reserve for the current visit.

### Sighting Entry Dialog Fragment

This section explains the steps after pressing the “add sighting” button on the main activity and how the sighting entry dialog fragment works and how data is passed between it and the main activity.sighting_entry_dialog_fragment_control_flow

*Diagram 4 (UML Activity diagram showing the control flow of the sighting entry dialog fragment)*

When the “add sighting” button is pressed the location listener in the main activity will try to retrieve the GPS coordinates from the phone. If this succeeds it will store the longitude and latitude values until sighting entry is finished by the user and a new sighting will be created using these values.

The sighting entry dialog fragment works the same way as the reserve entry dialog fragment explained before. It also uses an array of species gathered from the string file on the application for an autocomplete field.

On the dialog fragment there are two buttons that allow taking photos for the specimen and its general location. When these buttons are pressed the listener in the main activity will use an intent in order to use the phone’s camera.

Just like the location, the pictures taken will be saved in the main activity until the sighting is finished.

After saving or cancelling the data entry, the location and photos stored in the main activity will be reset.

### Sighting List View (create/update)

This section explains how the list view is managed using an adapter.sighting_list_view_control_flow

*Diagram 5 (UML Activity diagram showing the control flow for the creation of the sighting list view)*

The main activity contains a list view which will use an adapter to show the list of sightings in the current recording. The diagram above shows the steps made to create the adapter and how it fills the list with the necessary data.

The list view has to be updated every time the sighting list is altered. At the end of successfully adding, replacing or deleting a sighting the method updateSightingList() will go through these steps to refresh the list.

### Sighting Edit Dialog Fragment

This section explains how the available sightings in the list view are accessed to be edited or deleted using the on item click listener.

sighting_edit_dialog_fragment_control_flow

*Diagram 6 (UML Activity diagram showing the control flow for editing and deleting current sightings)*

An instance of the onClickListener set to the sighting list view when the main activity is created. This listener will get the position of the item clicked and store it in the main activity.

It will then open a new dialog fragment that allows re-entering the data for the current sighting. This dialog will reuse the XML layout of the sighting entry dialog fragment. It will also have a delete button that will delete the sighting marked by the sighting’s current position in the main activity.

The save button will use the same method used for the sighting entry, it will also use the Boolean value returned and delete the current sighting (which will be replaced) only if the new sighting is created successfully.

After saving, deleting or cancelling the sighting edit, the current position will be reset.

### Menu in Main Activity

This section explains the options available in the menu on the main activity.

menu_control_flow

*Diagram 7 (UML Activity diagram showing the control flow for the four different options in the main activity menu. Starting from the left, Edit User Contact Details, Edit Current Reserve, Delete whole Recording and Send Recording)*

The main activity has a menu, typically accessed by a button on the top right of the menu bar. This menu holds four different options.

The first option will create a log on dialog fragment as explained before.

The second option will create a reserve entry dialog fragment as explained before.

The third option will create a new empty visit and therefore delete the old visit with all its sightings.

The last option will send the visit (as explained before). It will than create a new empty visit and therefore delete the old visit with all its sightings.

### Methods used in Main Activity

Methods used privately in Main Activity

**protected void onCreate(Bundle savedInstanceState);**

Method that runs when the main activity gets created. It will do the steps explained in Main Activity (creation) using savedInstanceState to recover any previous records.

**public void onItemClick(AdapterView<?> parent, View view, int position, long id);**

Method used in the onItemClickListener used for the sighting list view. It will use int position to determine what item was selected in the list.

**public boolean onCreateOptionsMenu(Menu menu);**

Method that runs when the main activity gets created. It will populate the menu in the main activity.

**public boolean onOptionsItemSelected(MenuItem item);**

Method that runs when an item (MenuItem) inside the main activity’s menu is selected. It will do the steps explained in Menu in Main Activity.

**public void onSaveInstanceState(Bundle savedInstanceState);**

Method used to parcel the current recording into the savedInstanceState bundle when the main activity gets destroyed. The recording can be recovered when rebuilding the main activity.

**public void addUser();**

Method that creates the log on dialog fragment and shows it on screen.

**public void selectReserve();**

Method that creates the reserve entry dialog fragment and shows it on screen. It will also call addDate() to add a date to the current visit.

**public void addDate();**

Method that uses an instance of a calendar to get the current date and time. It will then reformat it to "yy-MM-dd HH:mm:ss" and store it in the current visit’s date string.

**public void updateLocation();**

Method used to reset the text view in the main activity showing the current reserve name.

**public void recordSighting(View view);**

Method that creates the sighting entry dialog fragment and shows it on screen.

**public void updateSightingList();**

Method that gets the current list of sightings in the recording and creates an adapter that populates the sighting list with these sightings.

**public void editSighting();**

Method that creates the sighting edit dialog fragment and shows it on screen.

Methods accessed by the Log On Dialog Fragment

**public void onLogOnDialogPositiveClick(DialogFragment dialog);**

Method that will access the log on dialog fragment elements using the parameter dialog to gather the information entered by the user and create a new user instance.

**public void onLogOnDialogNegativeClick(DialogFragment dialog);**

Method that cancels the user instance creation.

Methods accessed by the Reserve Entry Dialog Fragment

**public void onCreateReserveSearch(View view);**

Method that gathers the list of reserve names and uses an adapter to populate the autocomplete text field on the dialog fragment, accessing it using the parameter view.

**public void onReserveEntryDialogPositiveClick(DialogFragment dialog);**

Method that will access the reserve entry dialog fragment elements using the parameter dialog to gather the information entered by the user and store the reserve identified.

**public void onReserveEntryDialogNegativeClick(DialogFragment dialog);**

Method that cancels the identification of the reserve.

Methods accessed by the Sighting Entry/Edit Dialog Fragments

**public void onCreateSightingSearch(View view);**

Method that gathers the list of species names and uses an adapter to populate the autocomplete text field on the dialog fragment, accessing it using the parameter view.

**public void onCreateGetLocation(DialogFragment dialog);**

Method that uses a location manager to get the latitude and longitude for the last known location of the phone.

**public boolean onSightingEntryPositiveClick(DialogFragment dialog);**

Method that will access the sighting entry/edit dialog fragment elements using the parameter dialog to gather the information entered by the user and create a new sighting instance. Returns value true if the creation of the sighting was successful.

**public void onSightingEntryNeutralClick(DialogFragment dialog);**

Method that cancels the entry/editing of a sighting.

**public void onSightingEditNegativeClick(DialogFragment dialog);**

Method that deletes the current sighting selected on the sighting edit dialog fragment.

**public void onCameraSpeciesClick(DialogFragment dialog);**

**public void onCameraLocationClick(DialogFragment dialog);**

**private void dispatchTakePictureIntent();**

**protected void onActivityResult(int requestCode, int resultCode, Intent data);**

**public void onLocationChanged(Location location);**

Method that gets the longitude and latitude values from the parameter location and stores them in two separate variables.

**public void onStatusChanged(String provider, int status, Bundle extras);**

Method implemented for the location listener.

**public void onProviderEnabled(String provider);**

Method implemented for the location listener.

**public void onProviderDisabled(String provider);**

Method implemented for the location listener.

### Methods used in Log On Dialog Fragment

**public Dialog onCreateDialog(Bundle savedInstanceState);**

Method that runs when the log on dialog fragment is created. It will set its title and create the positive and negative buttons.

Method used by positive button

**public void onClick(DialogInterface dialog, int id);**

Method that passes the current dialog on to the main activity in the parameter of the onLogOnDialogPositiveClick(DialogFragment dialog) method in the LogOnDialogListener.

Method used by negative button

**public void onClick(DialogInterface dialog, int id);**

Method that will call the onLogOnDialogNegativeClick(DialogFragment dialog) method in the LogOnDialogListener.

**public void onAttach(Activity activity);**

Method that runs when the log on dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the LogOnDialogListener and throw an error if the parent activity has not implemented it.

### Methods used in Reserve Entry Dialog Fragment

**public Dialog onCreateDialog(Bundle savedInstanceState);**

Method that runs when the reserve entry dialog fragment is created. It will set its title and create the positive and negative buttons.

Method used by positive button

**public void onClick(DialogInterface dialog, int id);**

Method that passes the current dialog on to the main activity in the parameter of the onReserveEntryDialogPositiveClick (DialogFragment dialog) method in the ReserveEntryDialogListener.

Method used by negative button

**public void onClick(DialogInterface dialog, int id);**

Method that will call the onReserveEntryDialogNegativeClick (DialogFragment dialog) method in the ReserveEntryDialogListener.

**public void onAttach(Activity activity);**

Method that runs when the reserve entry dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the ReserveEntryDialogListener and throw an error if the parent activity has not implemented it.

### Methods used in Sighting Entry Dialog Fragment

**public Dialog onCreateDialog(Bundle savedInstanceState);**

Method that runs when the sighting entry dialog fragment is created. It will set its title and create the positive and negative buttons. It will also call the onCreateGetLocation(DialogFragment dialog) and the onCreateSightingSearch(View view) methods passing the current dialog and the view as parameters respectively.

Method used by positive button

**public void onClick(DialogInterface dialog, int id);**

Method that passes the current dialog on to the main activity in the parameter of the onSightingEntryPositiveClick (DialogFragment dialog) method in the SightingEntryListener.

Method used by neutral button

**public void onClick(DialogInterface dialog, int id);**

Method that will call the onSightingEntryNeutralClick (DialogFragment dialog) method in the SightingEntryListener.

**public void onAttach(Activity activity);**

Method that runs when the sighting entry dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the SightingEntryListener and throw an error if the parent activity has not implemented it.

### Methods used in Sighting Edit Dialog Fragment

**public Dialog onCreateDialog(Bundle savedInstanceState);**

Method that runs when the sighting edit dialog fragment is created. It will set its title and create the positive and negative buttons. It will also call the onCreateSightingSearch(View view) method passing the current view as parameter.

Method used by positive button

**public void onClick(DialogInterface dialog, int id);**

Method that passes the current dialog on to the main activity in the parameter of the onSightingEntryPositiveClick (DialogFragment dialog) method in the SightingEntryListener (reusing this listener).

Method used by neutral button

**public void onClick(DialogInterface dialog, int id);**

Method that will call the onSightingEntryNeutralClick (DialogFragment dialog) method in the SightingEntryListener (reusing this listener).

Method used by negative button

**public void onClick(DialogInterface dialog, int id);**

Method that will call the onSightingEditNegativeClick (DialogFragment dialog) method in the SightingEditListener.

**public void onAttach(Activity activity);**

Method that runs when the sighting entry dialog fragment is created, attaching it to the parent activity. It will try to run an instance of the SightingEntryListener and the SightingEditListener and throw an error if the parent activity has not implemented them.

### Methods used in Sighting List Adapter

**public SightingListAdapter(Context context, ArrayList<Sighting> values)**

Constructor for the adapter that takes an array list of sightings that is used to populate the list view.

**public View getView(int position, View convertView, ViewGroup parent)**

Method that uses an XML layout for the list view items and uses the provided sighting list to fill in the data species name and description.

## 3 Program Structure - Website

It is possible to view a list of all the recordings made (sent from users of the android app) by navigating to the "recordings" page of the website. On this page it is also possible to view only recordings made at a specific reserve by using the drop down menu provided, this search functionality is achieved by using an sql statement which retrieves only recordings which match the specified criteria from the database. The list of recordings can also be sorted in order based on the species name or the date the recording was made which is achieved by using the "ORDER BY" keyword in sql. In addition to all of the above the user can also click on the small thumbnail image of either the scene or specimen photo in order to view the full-sized image, this functionality was achieved by using a resource called Lightbox by Lokesh Dhakar (http://lokeshdhakar.com/projects/lightbox2/).

A complete list of all the reserves stored in the database can be viewed by navigating to the "reserves" page of the website, on this page it is also possible to edit, delete or add reserves however these options are only available to authenticated users. The reserves are sorted in alphabetical order for convenience.

To edit or delete a specific reserve there are buttons at the end of each reserve row. Clicking the delete button executes an sql statement which removes the corresponding reserve from the reserves database table. Clicking on the edit button takes the user to an edit reserve page ("edit.php") which displays the current information about the reserve and gives the user the option to change it. The edit page contains field validation code which validates any user input to make sure it is not malicious and is correctly formatted, regular expressions and built-in php functions like "strlen" are used in order to achieve this.

If the user does enter invalid input, a message is shown to the user pointing out the errors and the data is not stored in the database. It is possible to add a reserve by clicking on the "Add Reserve" button again found on the "reserves" page of the website, this works in a similar fashion to the edit functionality however in this case the form fields will not be pre-populated with existing data. The user of the web app can enter information about a reserve they wish to add and this input will be validated before the new reserve is added to the 'reserves' database table.

The recordings data sent from the android app via HTTP Post is received by a page called "fetch\_data". This page validates then stores the received recordings data into the 'recordings' database table but only if the validation passes. Every time a request is received information about the HTTP Post request is stored within a log file.

**Data Areas**

See Significant Data Structures for the Web App in the Design Document.

**Algorithms**

See Web Algorithms section in the Design Document.

**Files**

When the Web App receives a HTTP Post Request a log file containing details of the request is created. If the log file already exists however, future requests will be appended to the bottom of the file. The information in these logs is useful as it displays any errors that may have occurred during the receipt of the data.