Assignment 05

COS30017 - Software Development for Mobile Devices

Daniel Parker 971328X

October 1, 2014

1. Task 1

1.1. App Wiring

The app uses two activities, one to display a list of locations (loaded from file), and another to show the specific sun rise and set times for that location. The second activity is passed a *Location* object as context for showing the correct times for the selected location. The following code snippets show the specific 'wiring' components, such as intents and ListAdapters that are required, as boilerplate code to make this app function as intended.

LocationsActivity: Responsible for the listview landing activity and how that is initialised with Location data. This activity extends the abstract *ListActivity* class to allow for list item select event overriding. This Activity also creates intents to start the *SuntimeActivity* and show a particular locations sun times.

LocationsAdapter: Responsible for converting the underlying Location data model from it's object form into the UI views. This is set as the list adapter in the *LocationsActivity* class.

SuntimeActivity: Responsible for displaying a given location's sun times. This class makes use of some calculation classes; *AstronomicalCalendar*, *GeoLocation*, and *SunTimesCalculator*. Those classes were supplied, therefor I will not be including code snippets from them.

Location: This is the data model for a location that is shared between the list adapter and the *SuntimeActivity* class. Objects of this class are *Parcelable* so that they can be passed as extra data in intents that are created by the list activity for *SuntimeActivity*.

1.1.1. LocationsActivity

```
@EActivity(R.layout.activity_locations)
public class LocationsActivity extends ListActivity {
    private ArrayList<Location> listData = new ArrayList<Location>();
    private ArrayAdapter<Location> adapter;

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

        initialiseUI();
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        Location selectedItem = (Location) getListView().getItemAtPosition(position);
        showSuntimesForItem(selectedItem);
    }

    private void showSuntimesForItem(Location selectedItem) {
        Intent intent = new Intent(this, SuntimeActivity.class);
        intent.putExtra("location", selectedItem);
        startActivity(intent);
    }

    private void initialiseUI() {
        AssetManager assetManager = getAssets();
        try {
```

```
            InputStream inputStream = assetManager.open("au_locations.txt");
            this.listData = Location.loadLocations(inputStream);
            adapter = new LocationsAdapter(this, this.listData);

            setListAdapter(adapter);
        } catch (IOException e) {
            Log.e("SUNTIME", e.getMessage());
        }
    }
}
```

1.1.2. LocationsAdapter

```
public class LocationsAdapter extends ArrayAdapter<Location> {

    private final Context context;
    private final ArrayList<Location> values;

    public LocationsAdapter(Context context, ArrayList<Location> values) {
        super(context, R.layout.list_location_item, values);
        this.context = context;
        this.values = values;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View rowView = inflater.inflate(R.layout.list_location_item, parent, false);
```

```
        TextView locationName = (TextView)rowView.findViewById(R.id.Location_Name);
        locationName.setText(values.get(position).getName());

        return rowView;
    }
}
```

1.1.3. SuntimeActivity

```
public class SuntimeActivity extends Activity
{
    /** Called when the activity is first created. */
    private Location currentLocation;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_suntime);

        Intent intent = getIntent();
        Bundle bundle = intent.getExtras();

        this.currentLocation = (Location)bundle.getParcelable("location");

        initializeUI();
    }

        private void initializeUI()
        {
```

```
        DatePicker dp = (DatePicker) findViewById(R.id.datePicker);
        Calendar cal = Calendar.getInstance();
        int year = cal.get(Calendar.YEAR);
        int month = cal.get(Calendar.MONTH);
        int day = cal.get(Calendar.DAY_OF_MONTH);
TextView locationTV = (TextView)findViewById(R.id.locationTV);
        dp.init(year,month,day,dateChangeHandler); // setup initial values and reg. handler
        updateTime(year, month, day);
}

private void updateTime(int year, int monthOfYear, int dayOfMonth)
{
GeoLocation geoLocation;
if (currentLocation == null) {
    TimeZone tz = TimeZone.getDefault();
    geoLocation = new GeoLocation("Melbourne", -37.50, 145.01, tz);
} else {
    geoLocation = new GeoLocation(
                        currentLocation.getName(), currentLocation.getLatitude(),
                        currentLocation.getLongitude(), currentLocation.getTimeZone());
}

TextView locationTV = (TextView)findViewById(R.id.locationTV);
locationTV.setText(geoLocation.getLocationName());

        AstronomicalCalendar ac = new AstronomicalCalendar(geoLocation);
        ac.getCalendar().set(year, monthOfYear, dayOfMonth);
        Date srise = ac.getSunrise();
        Date sset = ac.getSunset();
```

```
                SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

                TextView sunriseTV = (TextView) findViewById(R.id.sunriseTimeTV);
                TextView sunsetTV = (TextView) findViewById(R.id.sunsetTimeTV);
                Log.d("SUNRISE Unformatted", srise+"");

                sunriseTV.setText(sdf.format(srise));
                sunsetTV.setText(sdf.format(sset));
        }
```

1.1.4. Location

```
public class Location implements Parcelable {

    private String name;
    private double latitude;
    private double longitude;
    private TimeZone timeZone;

    public static final Parcelable.Creator<Location> CREATOR
            = new Parcelable.Creator<Location>() {
        public Location createFromParcel(Parcel in) {
            return new Location(in);
        }

        public Location[] newArray(int size) {
            return new Location[size];
        }
    };
```

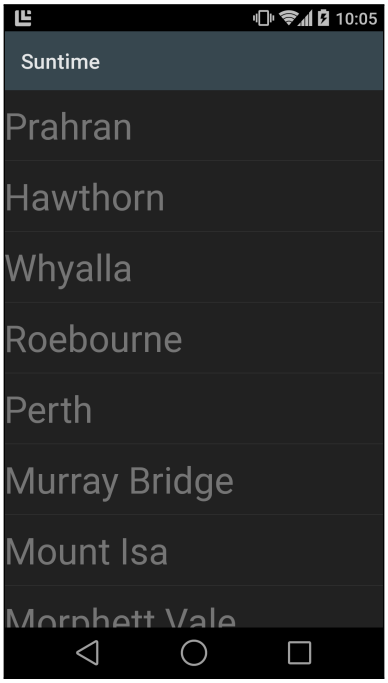
```
public Location(Parcel in) {
    this.name = in.readString();
    this.latitude = in.readDouble();
    this.longitude = in.readDouble();
    this.timeZone = (TimeZone)in.readValue(TimeZone.class.getClassLoader());
}

public Location(String name, double latitude, double longitude, TimeZone timeZone) {
    this.name = name;
    this.latitude = latitude;
    this.longitude = longitude;
    this.timeZone = timeZone;
}

public void writeToParcel(Parcel out, int flags) {
    out.writeString(this.name);
    out.writeDouble(this.latitude);
    out.writeDouble(this.longitude);
    out.writeValue(this.timeZone);
}

public static ArrayList<Location> loadLocations(InputStream locationsFile){
    CSVReader csvReader = new CSVReader(new InputStreamReader(locationsFile));
    String[] nextLine;
    ArrayList<Location> locations = new ArrayList<Location>();
    try {
        while ((nextLine = csvReader.readNext()) != null) {
            String name = nextLine[0];
            double latitude = Double.parseDouble(nextLine[1]);
            double longitude = Double.parseDouble(nextLine[2]);
```

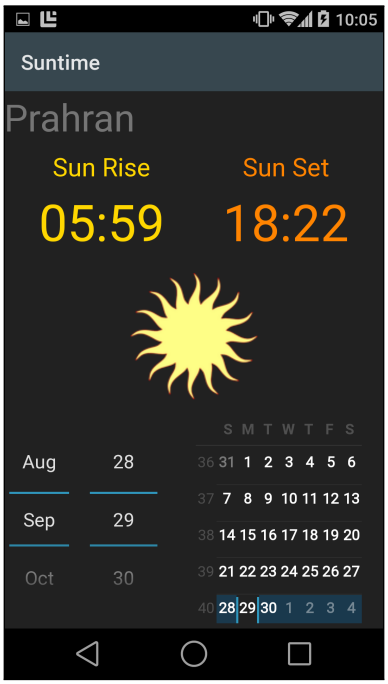
```
                TimeZone timeZone = TimeZone.getTimeZone(nextLine[3]);
                locations.add(new Location(name, latitude, longitude, timeZone));
            }
        } catch (IOException e){
            Log.e("SUNTIME", e.getMessage());
        }

        return locations;
    }
```

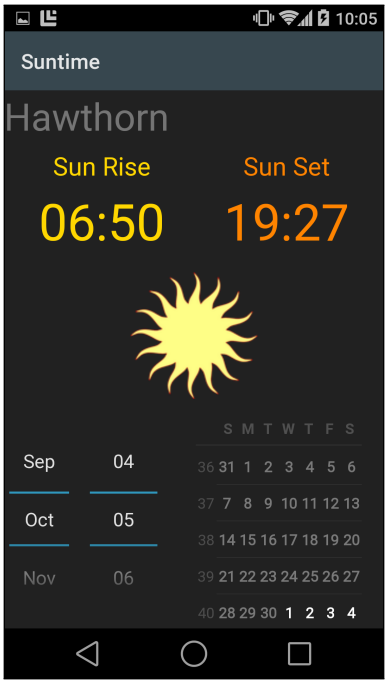
1.2. Custom Locations

```
Prahran,-37.8515240,144.9934840,Australia/Melbourne
Hawthorn,-37.8201480,145.0358000,Australia/Melbourne
```

1.3. Screenshots







2. Task 2

2.1. User Stories

1. As a recreational road-cyclist, I want to get the sun rise time and weather conditions so that I can maximise the time riding in daylight before I go to work.
2. As an adventure tour guide, I want to know the sun set times for a period of a week so that my group of campers can have their tents set up before sundown.
3. As a group runner, I want to be able to share the sun rise time with those that I'm running with so that we can all be ready to run as soon as there's daylight.
4. As a zookeeper, I need to know the sun rise times so that I can plan to clean the animal enclosures and feed the animals before the zoo opens.

2.2. User Stories vs. Scenarios

When comparing user stories and scenarios as a way of showing how an application may be used, I think it user stories are better to use, due to the fact that they are shorter and tend to only focus on one specific feature. The Scenarios have a lot of information, and potentially every single feature can be listed as part of the scenario story. This

leads tohaving to go thoroughly through the scenarios to actually get the important information out.

2.3. Prototype

2.3.1. Screenshots

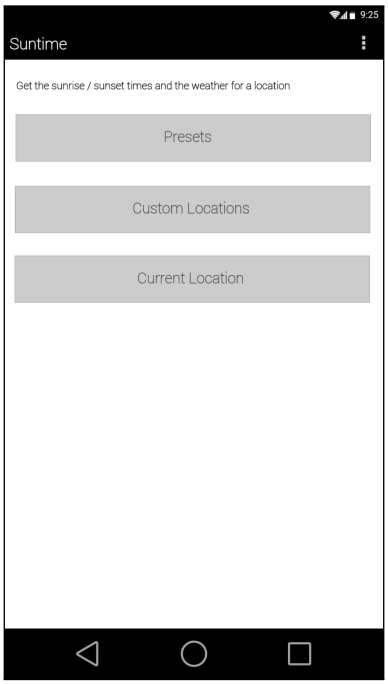


Figure 1: HOME

Description: This is the first screen the user will see. The user has three options for selecting a location; *Presets*, *Custom Locations*, and *Current Location*.

Features: Options for locations as described above. Can detect current location as an option.

Design choices: The choice to go for three buttons on this screen was primarily simplicity. In an actual app this could be made to look a lot nicer using button styles, or even compacted in to one of the other screens.

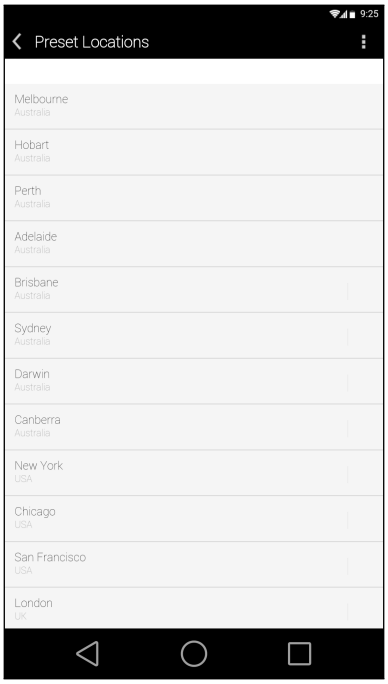


Figure 2: PRESETS

Description: The user can select one of the preset locations from this list and they will then be taken to select a date / date range.

Features: Select from pre-built set of locations.

Design choices: This page is just a list of presets, the design heuristic for a data set like this is to use a list. Styling could be nicer in full implementation.

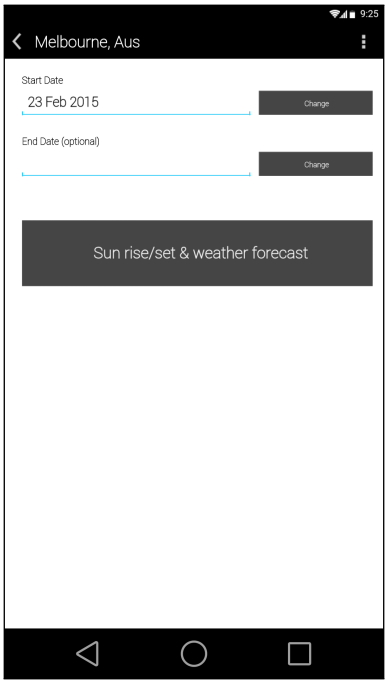


Figure 3: DATE-SELECT

Description: User can select a single date or date range to view the sun rise/set and weather. A different view is shown to them depending on whether a single date or a range is selected.

Features: Select a date or a date range to view sun times and weather forecast.

Design choices: The way that this screen is layed out is not ideal. It's not obvious that you need to omit the second date to just get a single date mode. Need to work out a better way of displaying these options.

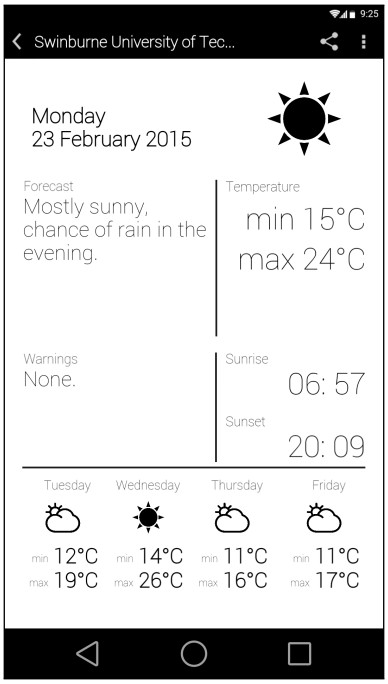


Figure 4: SUN-WEATHER

Description: This screen shows the weather and sun rise/set times for the selected date. The weather for the next 4 days is also shown. User can share this information by pressing the share icon in the action bar.

Features: Show sun rise/set times for date/location. Share information via SMS and email (or any other app). View weather forecast (current, and near future).

Design choices: This compact screen shows a lot of information. It's very logically layed out so that the user knows what each section means. I tried a new style of formatting text, where the label is smaller than the value. This seems to work well, as it puts the emphasis on the value and not the label for it. The next 4 day's weather is shown compactly at the bottom of the screen.

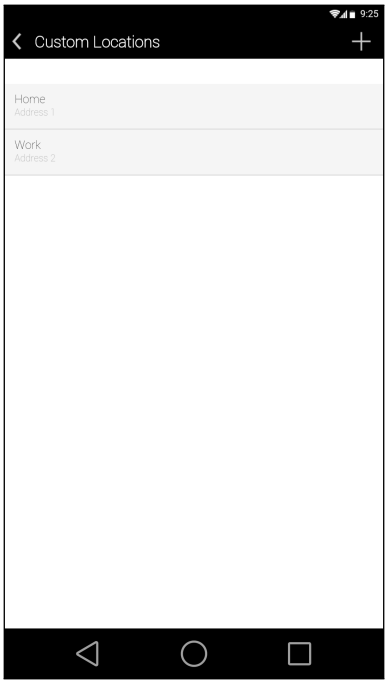


Figure 5: CUSTOM

Description: The user can select one of their previous custom locations or press the + symbol to take them to add another custom location.

Features: Add new custom location (partial).

Design choices: This is a list much like the *PRESETS* screen. The action bar in this case has a plus symbol so that the user knows they can add new locations to the list.

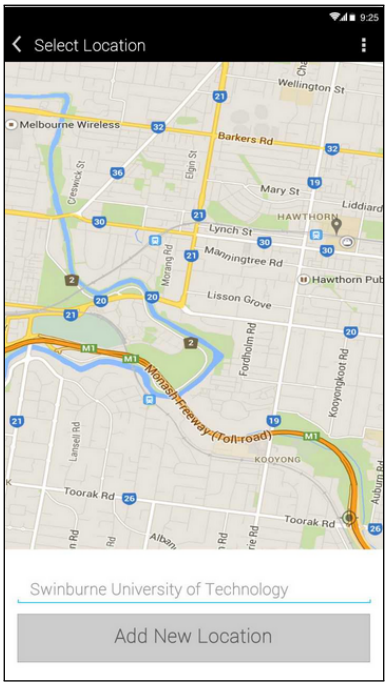


Figure 6: MAP

Description: User can either pin a point on the map or search for a location in the search bar. This will add it to the custom locations list and allow them to see the sun times and weather.

Features: Add custom locations. Integrated into Google maps. View sun rise / set times for various locations on a map.

Design choices: This screen features a map and a search box. The user can either click on a location on the map or search for one using the search box. I'm not sure if that will be an intuitive thing and would need to run some usability tests to find out if users have trouble using this screen.

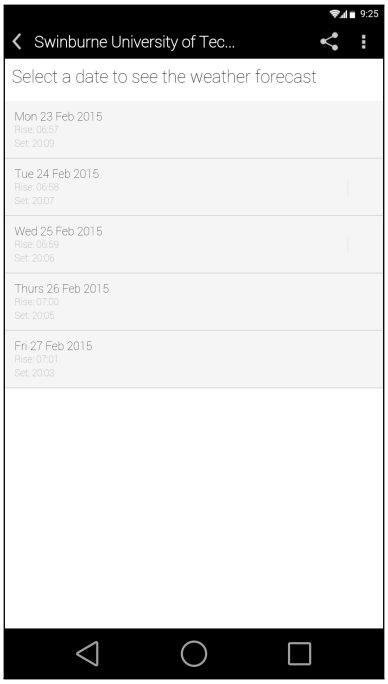


Figure 7: SUN-RANGE

Description: This screen shows the sun times for a range of dates which was requested on the previous screen *DATE-SELECT*. User can click on any date and see the full *WEATHER* screen as well.

Features: Generate table of sun rise/set times for a date range.

Design choices: This screen shows a range of dates and their sun times. It is a list because it's unknown how long the date range selected will be and also because the user can select any date in this list to see the weather.

2.3.2. User scenario validations

1. Brad can enter the Wellington harbour into the app as a custom location. Once added Brad can select the dates he would like sun rise times for and view them. HOME → CUSTOM → MAP → CUSTOM → DATE-SELECT → SUN-WEATHER / SUN-RANGE
2. Sachin can easily achieve this by adding each of the locations he will be and then generating sun rise and set times for the date range required. He can then easily share them by email with himself to print off. HOME → CUSTOM → MAP → CUSTOM → DATE-SELECT → SUN-RANGE → (Share button)

3. Li can find Sydney as a preset location and select the date for tomorrow to see the sun rise time. HOME → PRESETS → DATE-SELECT → SUN-WEATHER
4. Justin and Mary can select the current location option on the homescreen to get the data for their current GPS location and also share that with their friends. HOME → DATE-SELECT → SUN-WEATHER → (Share button)