Extended SWAN Tutorial

In this section we will explain how to add new sensors to SWAN and how to create an app using TriState Expression of SWAN-Song.

Adding New Sensor

Link: https://github.com/swandroid/swan-sense/commit/0d5dbd3c4b43ff089deecbb067a8b34912621c9a
We will explain how to add a network sensor to SWAN. Network sensor essentially means that the sensor data is obtained through the network.

In this tutorial we show how to add Rain Sensor that gives the expected mm rain per hour. We use the API provided by Dutch Weather Website - http://buienradar.nl/

The API list can be found in : http://gratisweerdata.buienradar.nl/

We use the rain API (http://gps.buienradar.nl/getrr.php?lat=52&lon=4). Output of which is shown below.

```
Roshans-MacBook-Pro:~ goose$ curl http://gps.buienradar.nl/getrr.php?lat=52&lon=4
[1] 22856
Roshans-MacBook-Pro:~ goose$ 000|16:30
000|16:35
000|16:40
000|16:45
000|16:50
000|16:55
000|17:00
000|17:05
000|17:10
000|17:15
000|17:20
000|17:25
000|17:30
000|17:35
000|17:40
000|17:45
000|17:50
000|17:55
000|18:00
000|18:05
000|18:10
000|18:15
000|18:20
000|18:25
000|18:30

[1]+  Done                    curl http://gps.buienradar.nl/getrr.php?lat=52
Roshans-MacBook-Pro:~ goose$
```

Based on the latitude and longitude coordinates, the precipitation two hours ahead can be retrieved in text format. The value 000 indicates that it is dry, 255 indicates that it is heavy rain. To convert the value into a standard format (mm/hour), We use the following formula provided by Buienradar.

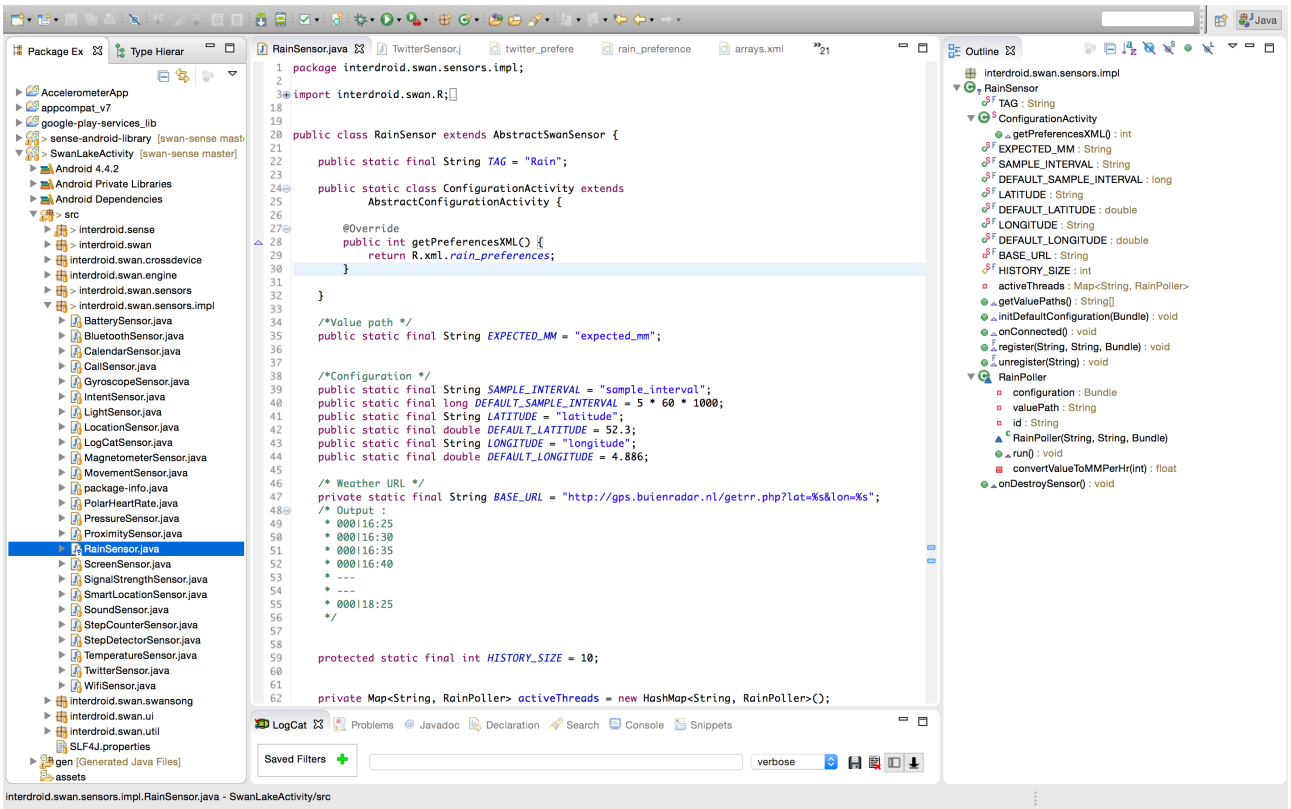
```
mm / hour = 10 ^ ((value -109) / 32)
So, 77 = 0.1 mm / hour
```

Next, we add RainSensor class in interdroid.swan.sensors.impl. It should extend AbstractSwanSensor class. We particularly set the following.

- Value path (expected_mm)
- Configuration(latitude, longitude)
- getValuePaths method
- initDefaultConfiguration method
- onConnected method
- register method
- unregister method

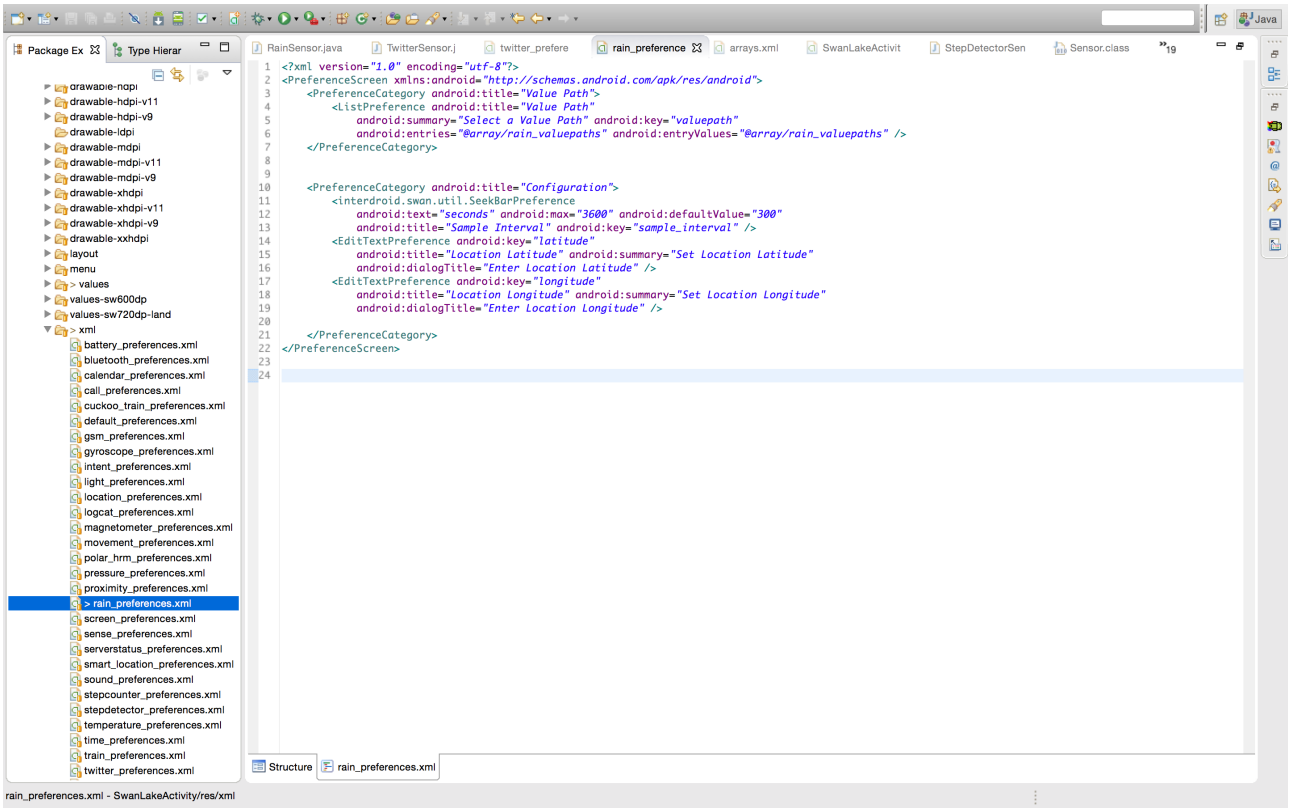
We will add two inner classes

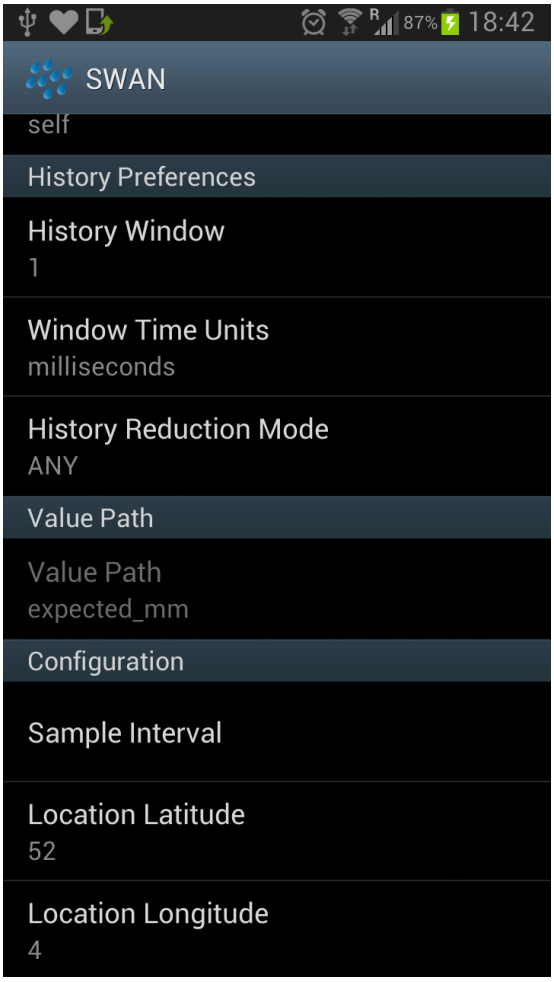
- ConfigurationActivity for setting the preference for creating the expression as discussed in the SWAN tutorial.
- RainPoller class which will run on a separate thread and it will poll rain data from buienradar every x interval.
  - It is important to call putValueTrimSize method to add the value for the given value path to the history



The ConfigurationActivity class uses getPreferencesXML method to get the sensor preference. The xml file for rain sensor preference is shown below.

The output of the rain sensor configuration activity is also shown below.


Package Ex
Type Hierar
RainSensor.java
TwitterSensor.j
twitter_prefere
rain_preference
arrays.xml
SwanLakeActivit
StepDetectorSen
Sensor.class
Java
drawable-hdpi-v11
drawable-hdpi-v9
drawable-ldpi
drawable-mdpi
drawable-mdpi-v11
drawable-mdpi-v9
drawable-xhdpi
drawable-xhdpi-v11
drawable-xhdpi-v9
drawable-xxhdpi
layout
menu
values
values-sw600dp
values-sw720dp-land
xml
battery_preferences.xml
bluetooth_preferences.xml
calendar_preferences.xml
call_preferences.xml
cuckoo_train_preferences.xml
default_preferences.xml
gsm_preferences.xml
gyroscope_preferences.xml
intent_preferences.xml
light_preferences.xml
location_preferences.xml
logcat_preferences.xml
magnetometer_preferences.xml
movement_preferences.xml
polar_hrm_preferences.xml
pressure_preferences.xml
proximity_preferences.xml
rain_preferences.xml
screen_preferences.xml
sense_preferences.xml
serverstatus_preferences.xml
smart_location_preferences.xml
sound_preferences.xml
stepcounter_preferences.xml
stepdetector_preferences.xml
temperature_preferences.xml
time_preferences.xml
train_preferences.xml
twitter_preferences.xml
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
<PreferenceCategory android:title="Value Path">
<ListPreference android:title="Value Path"
android:summary="Select a Value Path" android:key="valuepath"
android:entries="@array/rain_valuepaths" android:entryValues="@array/rain_valuepaths" />
</PreferenceCategory>
<PreferenceCategory android:title="Configuration">
<interdroid.swan.util.SeekBarPreference
android:text="seconds" android:max="3600" android:defaultValue="300"
android:title="Sample Interval" android:key="sample_interval" />
<EditTextPreference android:key="latitude"
android:title="Location Latitude" android:summary="Set Location Latitude"
android:dialogTitle="Enter Location Latitude" />
<EditTextPreference android:key="longitude"
android:title="Location Longitude" android:summary="Set Location Longitude"
android:dialogTitle="Enter Location Longitude" />
</PreferenceCategory>
</PreferenceScreen>
Structure
rain_preferences.xml
rain_preferences.xml - SwanLakeActivity/res/xml


87%
18:42
SWAN
self
History Preferences
History Window
1
Window Time Units
milliseconds
History Reduction Mode
ANY
Value Path
Value Path
expected_mm
Configuration
Sample Interval
Location Latitude
52
Location Longitude
4

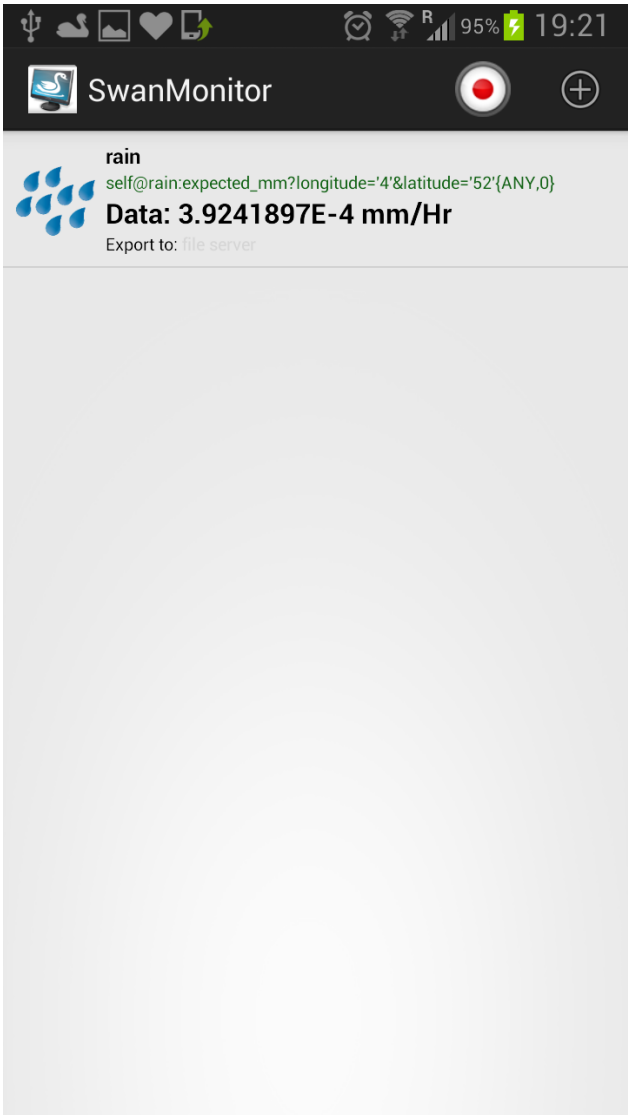
Since a swan sensor is actually a service containing an activity, both the service and the activity need to be declared in the manifest file. The service should contain the “exported” attribute set to true (allows other apps to use the service) and the “process” attribute should be a unique name with a colon “:” at the beginning, setting the service to run in a private process, separate from the default application. The ConfigurationActivity must declare an intent-filter with the “interdroid.swan.sensor.DISCOVER” action. In this way it will be triggered every time the swan-based app sends a broadcast with this message. In addition, it should include any configuration properties for the user’s choice, like value paths and units. Don’t forget to add the required permissions. The AndroidManifest.xml file modification for rain sensor is shown below.

```
<activity
android:name="interdroid.swan.sensors.impl.RainSen$ConfigurationActivity"
android:exported="true"
android:icon="@drawable/rain_icon">
      <intent-filter>
            <action android:name="interdroid.swan.sensor.DISCOVER" />
      </intent-filter>

      <meta-data
      android:name="entityId"
      android:value="rain" />
      <meta-data
      android:name="valuePaths"
      android:value="expected_mm" />
      <meta-data
      android:name="latitude"
      android:value="0L" />
      <meta-data
      android:name="longitude"
      android:value="0L" />
      <meta-data
      android:name="units"
      android:value="mm/Hr" />
</activity>

<service
android:name="interdroid.swan.sensors.impl.RainSensor"
android:exported="true"
android:process=":rainsensor" >
</service>
```

Once the above changes are made, the rain sensor is ready to use. We can use SWAN Monitor app explained in the previous tutorial to check if it works. The screenshot below show the current expected mm precipitation of location with latitude ’52’ and longitude ‘4’ .



Tri State Expression based App

Link: https://github.com/swandroid/LetsBikeApp

We will now discuss how to build a tri state expression based app using SWAN. We take a motivating scenario where the “user need to be motivated to use bicycle in the morning”. To this extend we use three expression to build a scenario as below.

- To check if it is morning.
  - We will use the time sensor to check the hour of the day.
- To if the weather is fine (not raining).
  - We will use the rain sensor which we implemented earlier.
- To check if the user is already awake.
  - We will use the accelerometer sensor to check movement.

If the above condition satisfies, the user needs to be notified about using the bicycle. This scenario essentially uses three different sensors. Implementing a combination of multiple sensors to get a context for an app becomes difficult. Using SWAN it becomes easier to combine multiple sensors.

To program this, we just need to combine multiple expression (based on sensors) to form a conditional expression. For the above scenario, we use the following expression.

```
/* to check if it is morning. 12 value indicates the hour 12 pm*/
String is_morning = "self@time:hour_of_day < 12";

/* to check if it is raining. We assume the value less than 0.1 mm/hr
means it is not raining */
String not_raining = "self@rain:expected_mm?
longitude='4'&latitude='52'<0.1";

/* to check if the smartphone has been moved in the past hour */
String active = "self@movement:total{MAX,3600000} > 15.0";

/* Final expression */
String expression = not_raining+" && "+is_morning+ " && "+ active;
```

The pseudo code to register tri state expression is shown below. Please note that it is very similar to registering a value expression explained in the previous tutorial.

```
ExpressionManager.registerTriStateExpression(this,
String.valueOf(REQUEST_CODE),
(TriStateExpression) ExpressionFactory.parse(myExpression),
new TriStateExpressionListener() {

@Override
public void onNewState(String arg0, long arg1, TriState arg2) {

      /* arg2 is the TriState returned. It can be true, false or
undefined */
      tv.setText("arg0 = "+arg0+" arg1="+arg1+" arg2="+arg2);

      if(arg2==TriState.TRUE){
            AlertDialog.Builder builder1 = new
                        AlertDialog.Builder(MainActivity.this);
            builder1.setMessage("Good Morning!. Looks like a nice weather.
                        How about biking today?");
            builder1.setCancelable(true);
            builder1.setPositiveButton("Yes",
                        new DialogInterface.OnClickListener()
                        {
                              public void onClick(DialogInterface
                                          dialog, int id) {
                                    dialog.cancel();
                              }
                         });
            builder1.setNegativeButton("No",
                         new DialogInterface.OnClickListener()
                        {
                              public void onClick(DialogInterface
                                          dialog, int id) {
                                    dialog.cancel();
                              }
```

```
                                });
        AlertDialog alert11 = builder1.create();
        alert11.show();
        }}});
```

The screen shot of the tri state expression app is shown below. The image on the left shows when there is no movement(meaning the user is sleeping). The image on the right shows a notification when the user starts moving.