IoTivity Programmer's Guide – Soft Sensor Manager for Android

1 CONTENTS

2 INTRODUCTION

This document provides interaction details of SoftSensorManager (SSM) and how it helps in receiving data from various sensors. The intent of this document is to provide details for developers to understand how to use SDK APIs and how the SSM works to support those APIs.

The first part, SDK API, describes how an application uses the SoftSensorManager.

The second part, SSM Architecture and Components, describes how the main operations of the SDK API are operated in the SSM. In this part, the architecture of SSM will be presented and the components in the architecture will be described in details.

In the third part, SSM Query Statement, a query language called CQL is explained with multiple examples. The query language is the language used in SSM for applications to get sensing data.

The fourth part, Soft Sensor, explains how developers can implement a soft sensor and deploy it in the SSM. An example of SoftSensor, DiscomfortIndexSensor (DISoftSensor), is presented to help understanding Soft Sensors.

An Android cum linux based sample application is also illustrated in this document. The sample application includes the functionality of registering and unregistering query statements to get sensing data.

3 TERMINOLOGY

3.1 PHYSICAL SENSOR APPLICATION

A software application deployed in an open hardware device, such as Arduino board where the device composes hardware sensors such as temperature, humidity, or gyro sensors.

This application gets physical sensor data from the hardware board and sends the sensor data to other devices using the Iotivity Base framework.

3.2 SOFT SENSOR (= LOGICAL SENSOR, VIRTUAL SENSOR)

A Soft Sensor is a software module which presents user-defined sensing data.
The soft sensor:

- Collects sensing data from physical and/or other soft sensors,
- Manipulates the collected sensing data by aggregating and fusing them based on its own composition algorithms, and
- Provides the manipulated data to applications registered.

3.3 Soft Sensor Manager (SSM)

A software service which receives query statements about physical and logical sensors from applications, executes the queries, and returns results to the application through the Iotivity Base.

A more detailed description of Soft Sensor Manager and its relevant components has been provided later in this document.

4 ANDROID SDK API

SDK API is the facet of SSM to applications. It includes SSMInterface and ISSMClientListener as shown in the chart below.

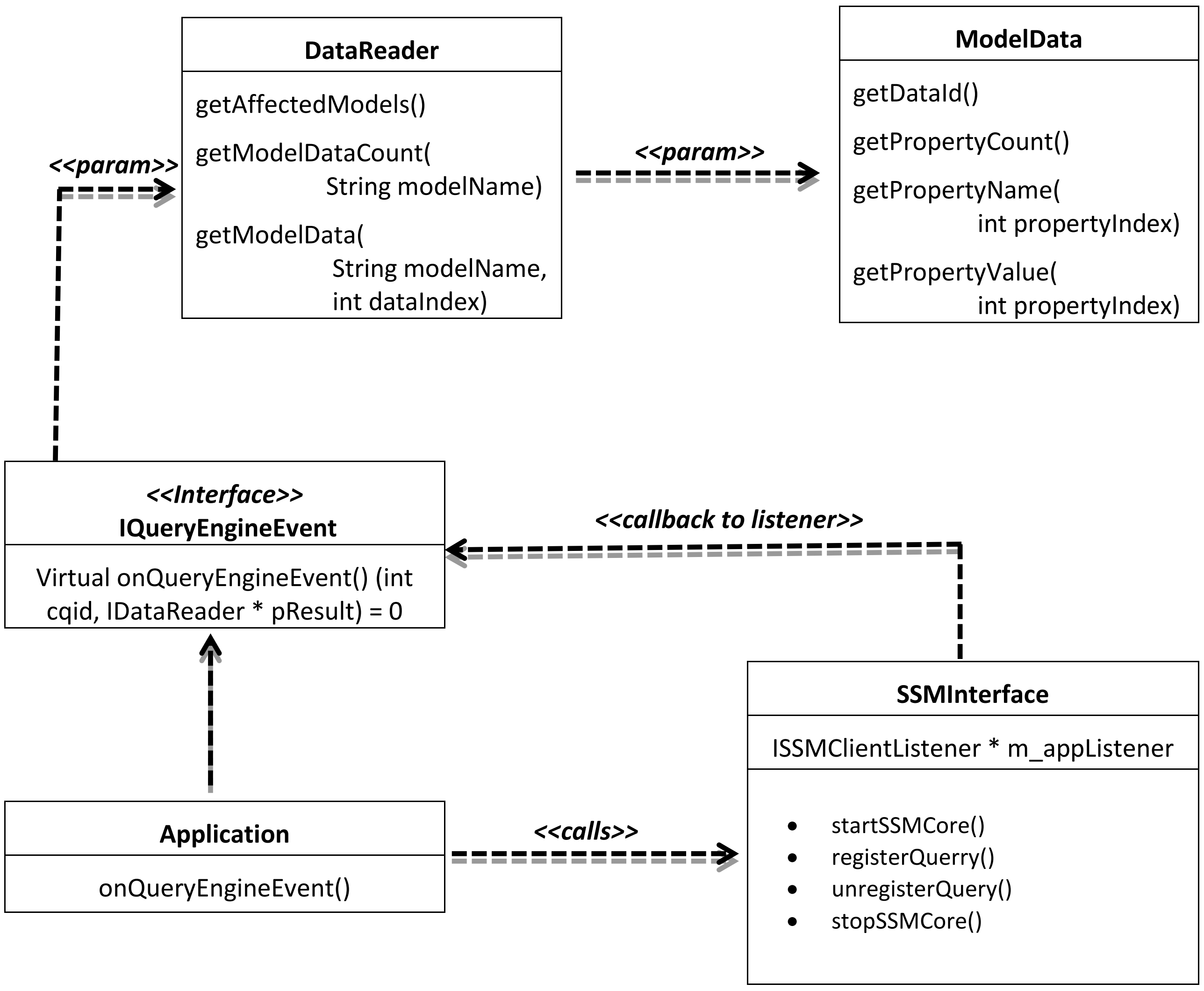

DataReader
getAffectedModels()
getModelDataCount(String modelName)
getModelData(String modelName, int dataIndex)
ModelData
getDataId()
getPropertyCount()
getPropertyName(int propertyIndex)
getPropertyValue(int propertyIndex)
<<param>>
<<param>>
<<Interface>>
IQueryEngineEvent
Virtual onQueryEngineEvent() (int cqid, IDataReader * pResult) = 0
<<callback to listener>>
SSMInterface
ISSMClientListener * m_appListener
startSSMCore()
registerQuerry()
unregisterQuery()
stopSSMCore()
Application
onQueryEngineEvent()
<<calls>>

Figure 1. SoftSensorManager SDK APIs and Application

4.1 SSMINTERFACE

SSMinterface is a class that provides APIs to the application for sending sensing data requests to SSM, and receiving the requested sensing data from it. These requests are sent in a query statement format. Context query language (CQL) is taken as the standard query language here.

For example, if we have a soft sensor providing an indoor discomfort index, called the DiscomfortIndexSensor (DISoftSensor). The DISoftSensor provides different index levels as follows;

- **ALL_FEEL_COMFORT**
- **BEGIN_TO_FEEL_UNCOMFORTABLE**
- **HALF_OF_YOU_FEEL_UNCOMFORTABLE**
- **ALL_FEEL_UNCOMFORT**

We can make a query statement to the Soft Sensor Manager (SSM) as below:

Subscribe DISoftSensor if DISoftSensor.level = **BEGIN_TO_FEEL_UNCOMFORTABLE**

In this scenario, application receives notification of DISoftSensor data from SSM when the condition, DISoftSensor.level = BEGIN_TO_FEEL_UNCOMFORTABLE, is satisfied. Therefore, there can be a time difference between the time the application sends the request and the time the application receives the response as SSM sends notification only if the condition is satisfied.

The operations provided in the SDK are listed below:

- startSSMCore()
- registerQuery()
- unregisterQuery()
- stopSSMCore()

The first operation, startSSMCore(), is to initialize the Soft Sensor Manager with the device and configuration information.

The second operation, registerQuery(), is to register the query statement to the Soft Sensor Manager. After successful response message for the registration, SSM will send an event to the client if the specified condition is satisfied.

The third operation, unregisterQuery (), is to un-register the previously registered query to Soft Sensor Manager. Application will not receive any further callbacks after successful un-registration.

The fourth operation, stopSSMCore (), is to terminate the Soft Sensor Manager.

The query statement composition and decomposition, has been described in details in the following sections.

4.2 IQUERYENGINEEVENT AND APPLICATION

This is an interface class for the application which sends a query statement to get the return from Soft Sensor Manager asynchronously.

The precondition for listening for the callback is as following:

- The application should implement the provided abstract class such as IQueryEngineEvent:

```
public class MainActivity extends Activity {

	private IQueryEngineEvent mQueryEngineEventListener = new IQueryEngineEvent() {
		@Override
		public void onQueryEngineEvent(int cqid, DataReader result) {
		}
	};
}
```

The application must provides listener to receive the notifications whenever specified condition is satisfied.

```
SSMInterface SoftSensorManager = new SSMInterface();
int cqid = SoftSensorManager.registerQuery("Query String", mQueryEngineEventListener);
```

SSMInterface registers this query with the query engine and sends event notifications to the IQueryEngineEvent Listener registered.

The onQueryEngineEvent() takes two parameters, *int cqid* and DataReader *result.* The cqid in this context means the entered context query ID. The result parameter gives the result received from SSMCore. Using this result as a reference, the onQueryEngineEvent() obtains a list of models using the method GetAffectedModels() , which in turn provides getModelData() methods with which soft sensor output data (i.e. ModelData) can be accessed as property name and property value. (i.e. : GetPropertyName()s and GetPropertyValue()).

- DataReader

Operation Name	Parameter/Return		Function
getAffectedModels	P	NULL	Get affected ContextModels. The CQL can specify multiple ContextModels for retrieving data.
	R	List<String> - Affected context model list.	
getModelDataCount	P	String modelName - Affected context model name.	Get affected data count. There are multiple data can exist from given condition.
	R	Int - Affected dataID count.	
getModelData	P1	String modelName - Affected context model name.	Get actual Context Model data
	P2	int dataIndex - Affected dataID count.	
	R	ModelData – Affected context model data reader.	

- ModelData

Operation Name	Parameter/Return		Function
getDataId	P	NULL	ContextModel has multiple data so returned data is matched from given condition.
	R	Int- Get affected DataId.	
getPropertyCount	P	NULL	ContextModel has at least one property that contains data property is described from its specification.
	R	Int – Get property count	
getPropertyName	P	int propertyIndex -Index of property to read	Retrieve propertyName
	R	String - PropertyName	
getPropertyValue	P	int propertyIndex - index of property to read	Retrieve propertyValue
	R	String - PropertyValue	

5 SSM ARCHITECTURE AND COMPONENTS

5.1 CONTEXT DIAGRAM:

The Soft Sensor Manager service is basically operated in the Iotivity Base messaging environment.

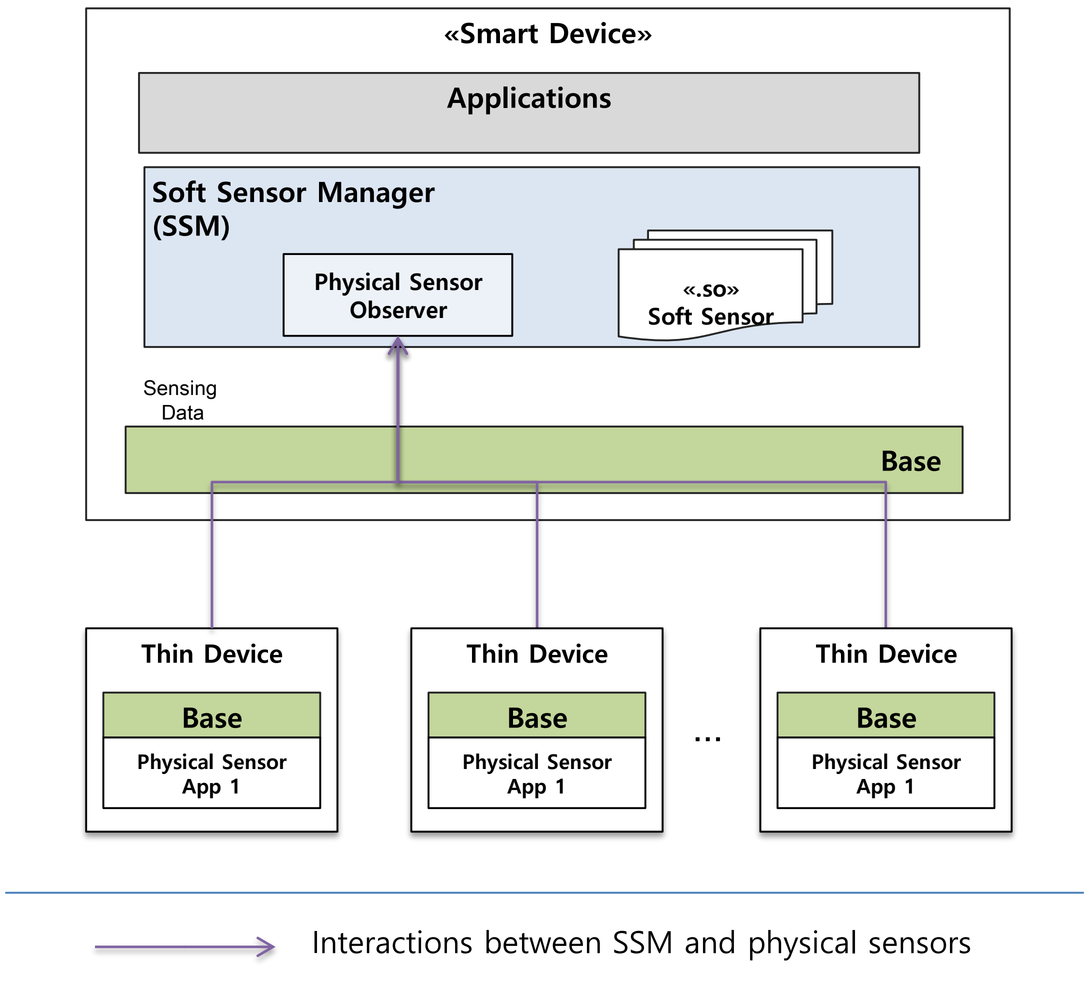


Figure 1. Soft Sensor Manager Context Diagram

There are two types of interactions with Soft Sensor Manager:

- Interactions between Application and Soft Sensor Manager.
- Interactions between Soft Sensor Manager and physical sensors.

For the first interaction, Soft Sensor Manager provides SDK APIs described in the previous section.

The second interaction is implemented within a resource model where a physical sensor is registered as a Resource in the Base and the Soft Sensor Manager observes the resource by using the APIs provide by the Base.

5.2 SSM Architecture

There is the Soft Sensor Manager service between applications and physical sensors, and it consists of the three main components such as SSMInterface, QueryProcessor, and SensorManager, as shown below.

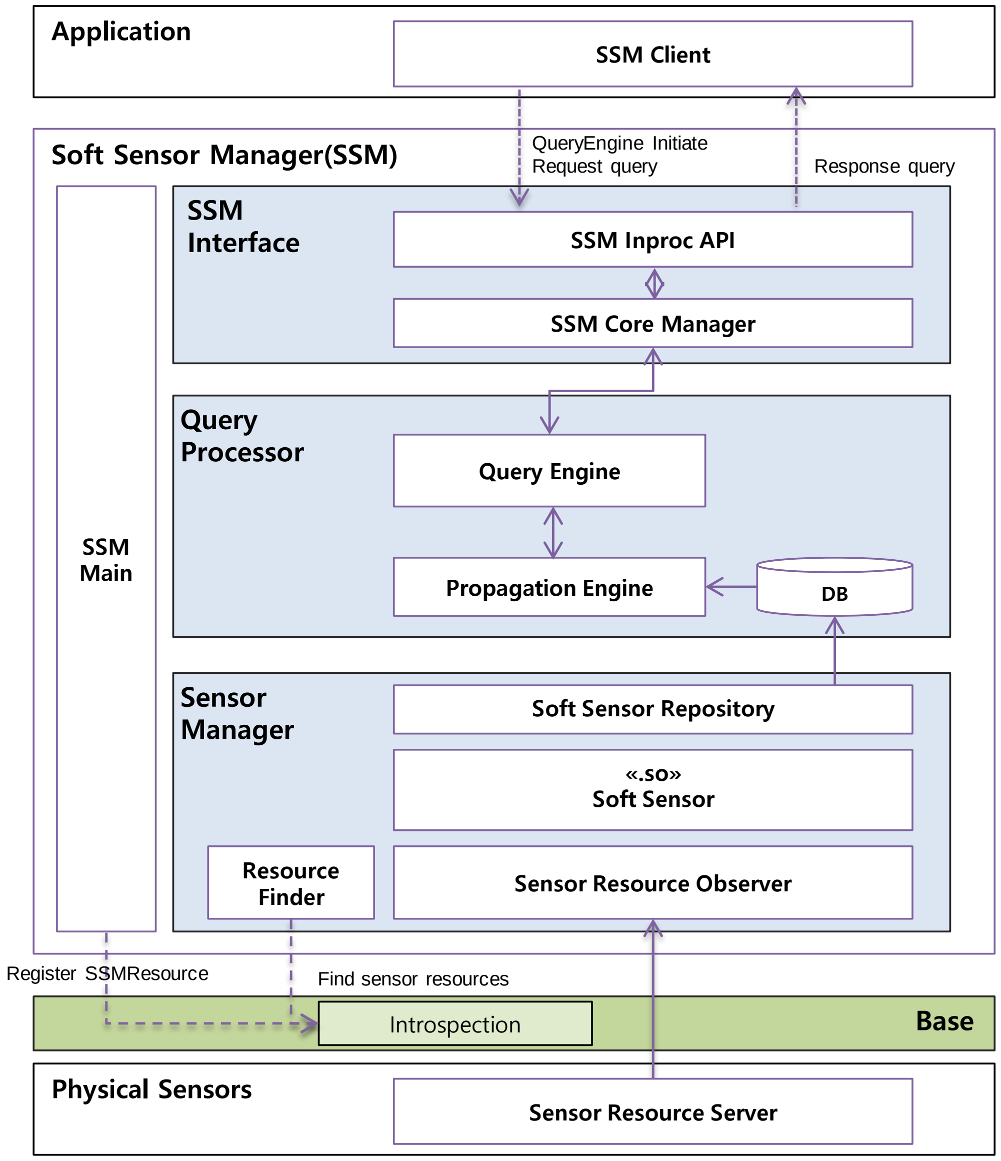


Figure 3. Soft Sensor Manager Architecture.

SSMInterface is an interface to the application (SSMClient) which sends requests to SSM and get responses from SSM via callback. The SSMInterface includes two main components: SSM Inproc API and SSM Core Manager. The SSM Inproc API is a wrapping class to communicate with SSMClient, and the SSM Core Manager is an interface class communicating with the Query Processor component.

QueryProcessor is a processing engine to get query statements, parse the statements, extract conditions and register the conditions. It also monitors the registered conditions whether they are satisfied or not. Once satisfied, it sends the notification to the Soft Sensor Manager Core Manager. It includes two main components, Query Engine and Propagation Engine. The Query Engine component is responsible for parsing the query statements and extracting conditions. The Propagation Engine component gets the conditions and registers them into the database. It also registeres the triggers to the database, so that the DB initiates callback when conditions are satisfied.

Sensor Manager is a component to maintain Soft Sensors registration and collect physical sensors data required by the Soft Sensors. To register Soft Sensors, a Soft Sensor needs to be deployed in the shared library form (*.so) with a manifest file (*.xml) describing the structure of the sensor. Soft sensor and its deployment are described in the fourth section. To collect physical sensors, there is SensorResourceFinder class which for finding specific resources, and registering an Observer to the found resources. It allows the physical sensor to send its sensing data when there is a change in the state or data.

6 SSM Query Statement

In query statements, there is a target model called ContextModel, which provides data for applications. In a device, there are three different types of context models; Device, SoftSensor, and PhysicalSensor.

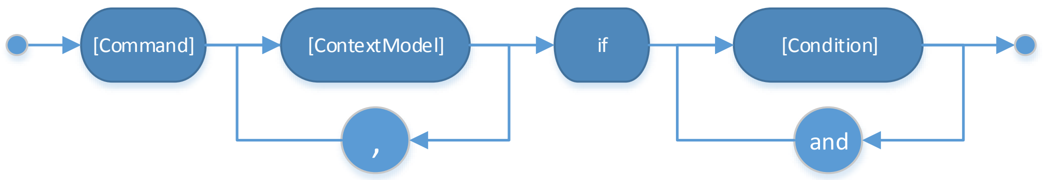
The Device Context Model corresponds to the device provides three properties, UDN, Name, and Type. The UDN contains unique key value with 128bit length, the Name represents device's own name like 'My Phone', and the Type is device's attribute like Mobile, TV, and PC.

Every Context Model includes dataId with which applications can access the Context Model data directly. For example, if Device Context Model contains five data, you can get 4th device information through query engine using 'get Device if Device.dataId = 4' query statement.

For soft sensors and physical sensors, they generally have their own structures and the Context Models of the sensors are generated with manifest files (*.xml) which are packaged together with the sensors.

6.1 Context Query Language (CQL)

The grammar of the CQL is as shown below



[Command]

The query engine provides two commands; Subscribe and Get.

The Subscribe keyword is used for asynchronous query request that is affective till the registered query is removed. That is, the result can be delivered to client multiple times whenever conditions are met. The Get keyword is almost same as subscribe, but the result data is delivered only once.

One of the main differences between the Subscribe and the Get is number of callback calls and inclusion of cached data. The result of Subscribe contains cached data at the execution time and new data after execution untill the query statement is unregistered. The Get command returns the most recent data.

[ContextModel]

Application developer should describe what Context Model data to retrieve from running result of CQL. The application developer can use comma (‘,’) to retrieve multiple Context Models.

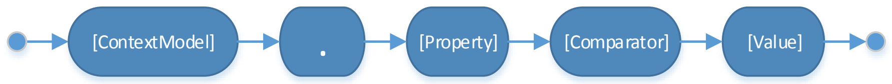
The Context Model can be defined by CQL’s [Condition] part description. For example, if [Condition] part is described like ‘Device.EPG.CurrentInfo.value != “null”’ then [ContextModel] contains Device, Device.EPG, Device.CurrentInfo Context Models.

If [Condition] part is described to combine conditions like ‘Device.EPG.value != “null” and Device.EPG.CurrentInfo.value != “null” using ‘and’ keyword, the [ContextModel] part contains Device, Device.EPG Context Models because these two things are the only intersect of two conditions.

[Condition]

It is used for application developers to search and trigger data using conditions.

The [Condition] grammar is as shown below:



The [property] part represents output properties that [ContextModel] has. Every Context Model has ‘dataId’ (i.e. Property). The [ContextModel] part must be declared with its parent [ContextModel] names. The [comparator] field can hold six operators like = (==), !=, >, >=, <, <=. The [value] field is set value for comparison. Following type of [value] is possible integer, float, double, text, Boolean and text, Boolean must be capsulated using double quotes.

Ex: Device.LiftUpSmartPhone.status = “true” or Device.type = “Mobile”

6.2 Examples of CQL Statements

```
subscribe Device if Device.type == "Mobile"
```

When Mobile Device appears on the network, notify Device information which satisfy this condition.

```
subscribe Device if Device.type == "TV" and
Device.NumberOfPeopleWatchingTV.number > 2
```

When type of Device is "TV" and the number of people watching TV is greater than 2, notify Device information.

```
subscribe if Device.LiftUpSmartPhone.value == "true" and
Device.NumberOfPeopleWatchingTV.number > 0
```

When Conditions are that, The Device's value of LiftUpSmartPhone is "true" and Device's number of NumberOfPeopleWatchingTV is greater than 0, if Device appears to satisfy above conditions inform us of Device information. (‘Device’ keyword must present every [ContextModel] of [Condition]).

```
Get Device if Device.BatteryStatus.percentage > 50
```

If Battery's percentage is greater than 50, notify Device information.

```
Get Device if Device.PhoneTodaySchedule.title = "study” and
Device.UserAtHome.value = "true"
```

Of peripheral devices, if title of PhoneTodaySchedule is "study" and value of UserAtHome is "true", retrieve Device information.

```
Get Device. PhoneGPS[2]
```

Retrieve PhoneGPS information of which dataId is 2. (Always the lowest [ContextModel] only has Index. Only "Get" query can contain index, not “if” statement(a conditional sentence)).

```
Subscribe Device if Device.CallStatus.callername = "lee" and Device.type =
"TV"
```

When callername is "lee" and type is "TV", notify Device information

```
Subscribe Device.TVZipcode if Device.TVZipcode.value != "null" and
Device.dataId = 3
```

If value of TVZipcode is not "null" and dataId of Device is 3, notify TVZipcode's information

```
Get Device.BatteryStatus[1]
```

Notify BatteryStatus information of which dataId is 1. (If the appropriate data doesn't exist, do nothing.)

```
Get Device.UserAtHome if Device.UserAtHome.value = "true"
```

If UserAtHome's value is "true", nofity UserAtHome's information (If the appropriate data doesn't exist, do nothing.)

```
Subscribe Device if Device.LiftUpSmartPhone.value = "true" and
Device.PhoneGPS.latitude != 20
```

If Phone's status is "LiftUp" and PhoneGPS's latitude is not 20, notify Device information.

```
Subscribe Device if Device.BatteryStatus.percentage >= 50 and
Device.LiftUpSmartPhone.value = "true"
```

If BatteryStatus's percentage is greater than or equal to 50 and Phone's status is "LiftUp", notify Device's information.

7 Soft Sensor

Soft sensor, also called Virtual sensor or Logical sensor, is a software component which gets physical sensor data and generates new data by data aggregation and fusion. This part shows how to develop a soft sensor, deploy in the SSM, and use the deployed soft sensor.

7.1 Development

SSM loads a shared library, (*.so) as a soft sensor unit. A soft sensor should be developed and deployed as a shared library which includes the entry operation, defined in the Interface, ICtxEvent.

Soft Sensor Definition: A soft sensor consists of three main elements; input data, output data, and execution logic. To be deployed in SSM the three elements can be implemented as follows;

Input data: the required data by the target soft sensor and it is generally the sensing data from physical sensors. For example, a DiscomfortIndexSensor, requires temperature sensors and humidity sensors as inputs. Output data: the result of data fusion by the target soft sensor. The unit of the output data should be different from the types of soft sensors.

In SSM, the main properties of Soft Sensor, name, input, output, should be described in a manifest **(**i.e. **SoftSensorDescription.xml)**.

Here is an example of DiscomfortIndexSensor:

The first tag, <name>, is referred in the query statement from applications. It is also used for SSM to load shared library (*.so). It should be the same as the name of the shared library file.

```
<softsensors>
  <softsensor>
    <name>DiscomfortIndexSensor</name>
    <attributes>
      <attribute>
        <name>version</name>
        <type>string</type>
        <value>1.0</value>
      </attribute>
      <attribute>
        <name>lifetime</name>
        <type>int</type>
        <value>60</value>
      </attribute>
    </attributes>
```

```
        <outputs>
          <output>
            <name>timestamp</name>
            <type>string</type>
          </output>
          <output>
            <name>temperature</name>
            <type>string</type>
          </output>
          <output>
            <name>humidity</name>
            <type>string</type>
          </output>
          <output>
            <name>discomfortIndex</name>
            <type>int</type>
          </output>
        </outputs>
        <inputs>
          <input>Thing_TempHumSensor</input>
          <input>Thing_TempHumSensor1</input>
        </inputs>
      </softsensor>
    </softsensors>
```

For inputs, the physical sensors required by the target soft sensor can be specified in this tag. Moreover soft sensors can be used for input sensors.

Execution Logic: With the input data, Soft sensor generates the output, based on its own algorithm.

Soft sensor should implement the ICtxEvent interface which provides the OnCtxEvent() operation, as a pure virtual operation. In SSM, the operation is called by CContextExecutor class right after the class loads the soft sensor's .so file, and when the SSM receives sensing data from physical sensors.

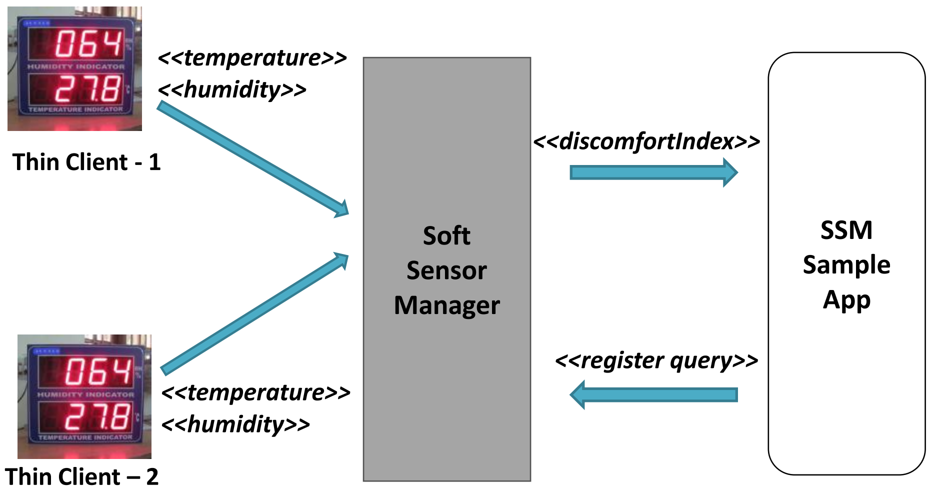
The OnCtxEvent operation requires two input parameters, eventType, and contextDataList as follows:

- eventType: It is the time point the onCtxEvent() called by SSM and includes three types, SPF_START, SPF_UPDATE, and SPF_END, where SPF_START is the time when the soft sensor library is loaded, and currently SSM only uses this option.
- contextDataList: It is the input value the soft sensor required and it is provided as an attribute map(key,string) of the sensing data from the physical sensors specified in the input tag in the manifest file. That is, SSM, CContextExecutor generates the attribute map of the input data and delegates to the soft sensor when it calls the OnCtxEvent().

8 EXAMPLE SCENARIO

8.1 SAMPLE DESCRIPTION

In this use case, we have two different thin devices (i.e. Thermo cum Humidity sensors), Discomfort Soft Sensor and the client application. The client application registers its query for obtaining discomfort level index with the soft sensor manager. The soft sensor manager begins observing the thin clients. While observing any change in temperature or humidity in any thin device it reads the values, the discomfort soft sensor uses its own algorithm to generate discomfort level index. Then notify this index value back to the client application. After obtaining the unregister query request from the client application, it stops to observe the thin devices.



8.2 PREREQUISITES

- For running the **SSM Sample** android application, all the required libraries(.so) need to be copied to proper location.

- Additional to this, as mentioned earlier a soft sensor should be developed and deployed as a shared library. Copy these required files into the ***application/assets/lib*** folder.
    - libDiscomfortIndexSensor.so
    - SoftSensorDescription.xml

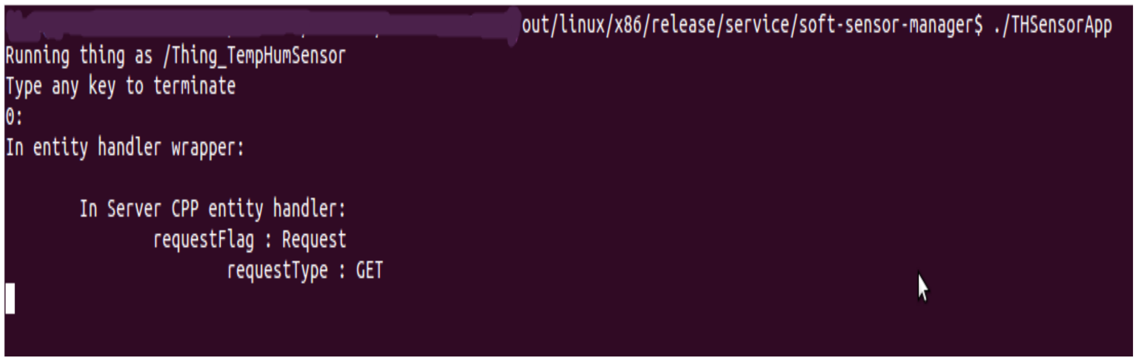
- All the three applications (i.e. **THSensorApp** (Linux), **THSensorApp1** (Linux) and the **SSM Sample Application**(Android)) should run on the same network.

8.3 SAMPLE SET UP

This section introduces an example setup for the above explained scenario.

- Run the thin devices application **THSensorApp** (Linux) and **THSensorApp1** (Linux) and the **SoftSensorManager Sample Application**(Android).

  To run the thin clients on Linux, run the commands as shown below (to run **THSensorApp**):



```
[redacted]out/linux/x86/release/service/soft-sensor-manager$ ./THSensorApp
Running thing as /Thing_TempHumSensor
Type any key to terminate
0:
In entity handler wrapper:

        In Server CPP entity handler:
                requestFlag : Request
                        requestType : GET
```

- Run **THSensorApp1** as well on Linux system.
- Run SSM Sample app on android device.

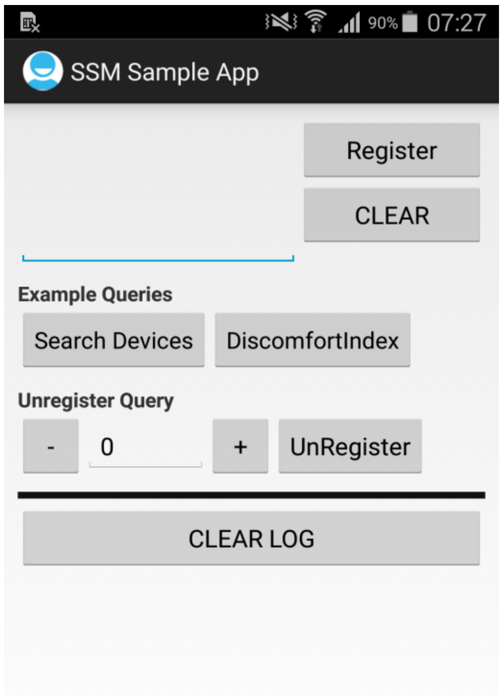


Figure 4: HomeScreen of the SSM sample application

8.4 Application Screen Description

- Search Devices : This button is to compose a query for searching available thin devices. On clicking this button, search query is generated and displayed on the left top corner of the screen (i.e. The Query Textbox)
  Note : This button only composes the search query and displays it to the user. It does not register it. The register button needs to be pressed after composing the search query.

- DiscomfortIndex : This button is to compose a query for receiving discomfort index if it's greater than 0, with available DiscomfortIndexSensor. On clicking this button, query is generated and displayed on the left top corner of the screen (i.e. The Query Textbox)
  Note : This button only composes the search query and displays it to the user. It does not register it. The register button needs to be pressed after composing the search query.

- Register : This button registers the displayed query to the soft sensor manager.

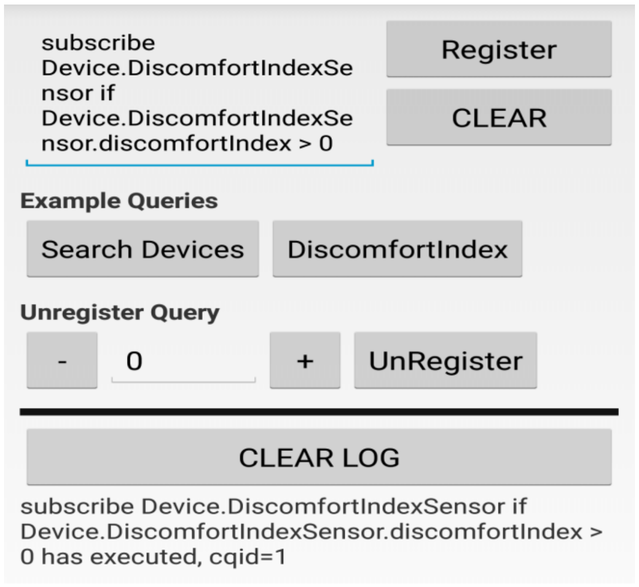
- UnRegister : This button unregisters the query that have already been registered using the register button. The counter is used to input the context query ID for the query to be unregistered.

- CLEAR : This button is to clear the Query Textbox field.

- CLEAR LOG : This button is for clearing the logs coming in the log display box.

8.5 Working flow

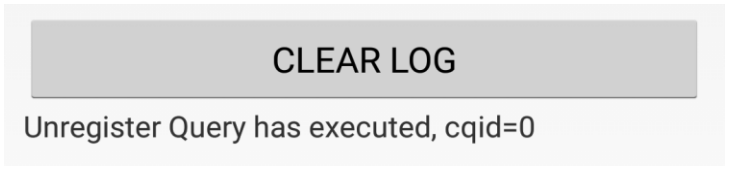
- Compose the DiscomfortIndex Query using the DiscomfortIndex button. The query will be displayed in the Query text box. Once query is registered, log will be displayed as shown below.



```
subscribe
Device.DiscomfortIndexSe
nsor if
Device.DiscomfortIndexSe
nsor.discomfortIndex > 0
```

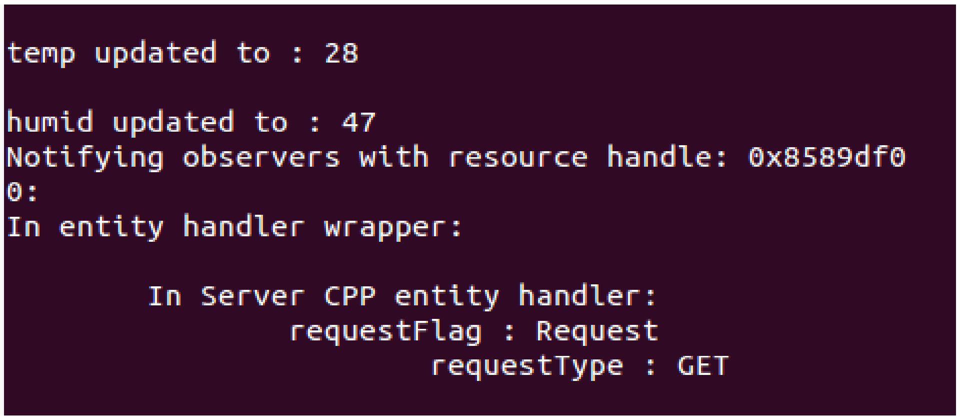
```
subscribe Device.DiscomfortIndexSensor if
Device.DiscomfortIndexSensor.discomfortIndex >
0 has executed, cqid=1
```

- To unregister registered query, input the query id value to be unregistered and press the unregister button.



```
Unregister Query has executed, cqid=0
```

- The **THSensorApp** and **THSensorApp1** have been thus configured to vary the temperature and humidity value periodically. It displays the varied values as shown :

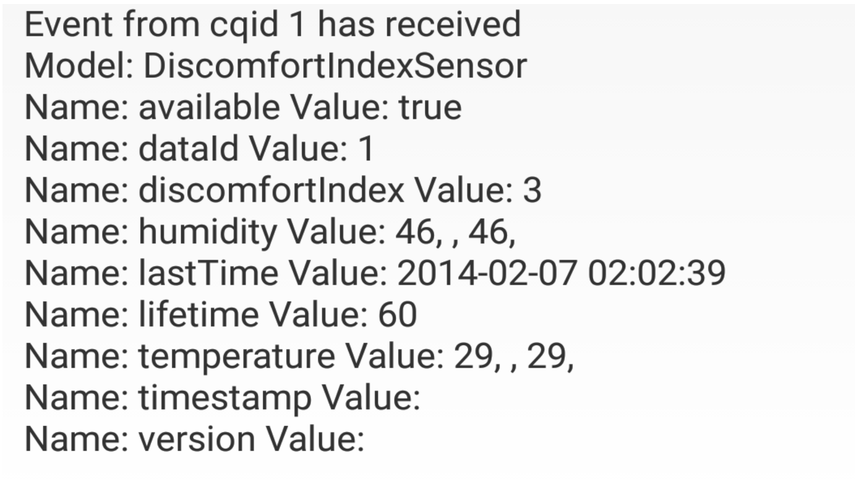


```
temp updated to : 28

humid updated to : 47
Notifying observers with resource handle: 0x8589df0
0:
In entity handler wrapper:

        In Server CPP entity handler:
                requestFlag : Request
                        requestType : GET
```

- On receiving changed values from the thin clients the soft sensor manager manipulates the data and sends back the callback to the application with the discomfortIndex.



```
Event from cqid 1 has received
Model: DiscomfortIndexSensor
Name: available Value: true
Name: dataId Value: 1
Name: discomfortIndex Value: 3
Name: humidity Value: 46, , 46,
Name: lastTime Value: 2014-02-07 02:02:39
Name: lifetime Value: 60
Name: temperature Value: 29, , 29,
Name: timestamp Value:
Name: version Value:
```