Using the proximity sensor in Java ME

This article explains how to get access and retrieve the state of the device proximity sensor in Java ME.

01 Sep 2013

Introduction

The Mobile Sensors API allows Java apps to retrieve information from sensors attached or integrated into a mobile phone.

Among the available sensors, the *Asha software platform* offers access to the device's proximity sensor, which allows users to detect the presence of nearby objects without the need for any physical contact. The proximity sensor is typically used during ongoing phone calls, to prevent accidental touches when the user has the phone located near his/her ear.

Asha app accessing the proximity sensor in Java ME

This article illustrates how the proximity sensor can be used within a Java app to detect the presence of nearby objects.

Implementation

This section shows how to access and retrieve data from the proximity sensor. It covers:

- looking for the sensor availability on the device
- connecting to the sensor
- registering a listener to handle sensor's data
- handle data coming from the proximity sensor

Accessing the proximity sensor

The `Mobile Sensors API` allows to search for available sensors through the SensorManager.findSensors static method, accepting the following two arguments:

- a first `String` indicating the quantity that the app wants to measure
- a second `String` indicating the relevant context for the desired sensor, with available contexts being defined by the following constants:
  - SensorInfo.CONTEXT_TYPE_DEVICE for device-related sensors
  - SensorInfo.CONTEXT_TYPE_USER for user-related sensors
  - SensorInfo.CONTEXT_TYPE_AMBIENT for ambient, environmental sensors
  - SensorInfo.CONTEXT_TYPE_VEHICLE for sensors which gives information of a vehicle

In order to access the proximity sensor, the quantity to be specified must be *proximity*, while the context, being the quantity a measure related to the device, must be SensorInfo.html#CONTEXT_TYPE_DEVICESensorInfo.CONTEXT_TYPE_DEVICE.

```
SensorInfo[] sensors = SensorManager.findSensors("proximity",
SensorInfo.CONTEXT_TYPE_DEVICE);
```

If the proximity sensor is available on the device, the SensorInfo array contains an element that allows us to retrieve information

(C) Copyright Nokia 2014. All rights reserved.

about the sensor.

In order to use the sensor, it is necessary to open a SensorConnection by using the sensor *URL*, that can be retrieved via the `SensorInfo` getUrl method.

```
if(sensors.length > 0)
{
 try
 {
  String sensorURL = sensors[0].getUrl();

  SensorConnection conn = (SensorConnection)Connector.open(sensorURL);
 }
 catch(Exception e)
 {
  e.printStackTrace();
 }
}
```

Listening to sensor data

Once a `SensorConnection` is opened, sensor data can be retrieved using asynchronous or synchronous methods, depending on the app's needs.

In order to be notified when the proximity sensor state changes (when an object is detected), the best approach is to use the asynchronous mode. In this case the DataListener interface is responsible for handling the data sent by the sensor via its dataReceived method.

Retrieved data is passed to the `dataReceived` method as a Data array, containing one element for each channel of the specific sensor: as the proximity listener has only one channel, the `Data` array will contain just one element.

Depending on the type of data sent by a sensor, data values can be retrieved from the `Data` object with one of getIntValues(), getDoubleValues() and getObjectValues methods.

Proximity sensors return a `boolean` state `true` if a near object is detected, `false` otherwise. The value can be retrieved (as an integer) using the `Data` getIntValues() method (which returns an array containing the actual data, whose length depends on the buffering that is used when registering the `DataListener`).

The following `dataReceived` implementation checks if a nearby object is detected, and stores the value in an instance variable, incrementing a counter by one for each detected object.

```
public void dataReceived(SensorConnection sensor, Data[] data, boolean isDataLost)
{
 if(data.length > 0)
 {
  Data dataItem = data[0];

  int[] proximityDataValues = dataItem.getIntValues();

  boolean objectDetected = (proximityDataValues[0] == 1);

  if(objectDetected != objectNearProximitySensor)
  {
   objectNearProximitySensor = objectDetected;

   if(objectNearProximitySensor)
   {
    objectsCounter++;
   }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }
   }
   repaint();
  }
```

Registering the DataListener

Once implemented, the `DataListener` must be registered using the `SensorConnection`'s setDataListener method, that accepts two arguments:

- the `DataListener` instance
- an integer specifying the buffer size: data retrieved from the sensor is buffered until this number of values is reached, and is then sent to the `DataListener.dataReceived` method. In this scenario, in order to receive instant notifications when the proximity sensor state changes, the buffer size must be set equal to 1.

```
sensorConnection.setDataListener(dataListener, 1);
```

Adapting the UI to the proximity listener state

As a phone does during a call when a nearby object is detected, a Java app could need to adapt the UI and functionality to changes in the proximity listener state: the following example shows a `Canvas` that uses a different color scheme for UI elements when a nearby object is detected.

```
protected void paint(Graphics g)
{
 Font font = DirectUtils.getFont(Font.FACE_MONOSPACE, Font.STYLE_PLAIN, 140);

 g.setColor(objectNearProximitySensor ? 0x000000 : 0xffffff);
 g.fillRect(0, 0, getWidth(), getHeight());

 g.setColor(objectNearProximitySensor ? 0xffffff : 0x000000);
 g.setFont(font);
 g.drawString(Integer.toString(objectsCounter), getWidth() / 2, 40,
Graphics.TOP|Graphics.HCENTER);
}
```

Java app detecting changes of the proximity sensor state

Testing

The Nokia Asha SDK 1.0 Emulator does not support emulation of proximity sensor, and the `SensorManager.findSensors` call returns an empty array if searching for the *proximity* quantity. For this reason, testing must be performed on a real device based on the Asha software platform.

(C) Copyright Nokia 2014. All rights reserved.

Summary

This article illustrates the basic steps to access the state of the proximity sensor on the Asha software platform.

Full source code of the Java app illustrated in this article is available here: Media:WikiProximitySensor.zip.

More details are available on the Mobile Sensors API JavaDocs pages.

(C) Copyright Nokia 2014. All rights reserved.