

Motion

Programming Guide

Version 2.2.1

Revision History

Version	Date	Description
1.0		Created this document.
2.0.0	January 08, 2014	Added Activity feature.
2.1.0	September 29, 2014	Added SmotionActivity .updateInfo() and SmotionActivity.isUpdateInfoBatchModeSupport()
2.2.0	February 12, 2015	Added SmotionActivity.isActivitySupported()
2.2.1	March 25, 2016	Change logging functions

Table of Contents

1. OVERVIEW	4
1.1. ARCHITECTURE.....	4
1.2. CLASS DIAGRAM	5
1.3. SUPPORTED PLATFORMS.....	6
1.4. SUPPORTED FEATURES	6
1.5. COMPONENTS	6
1.6. IMPORTING LIBRARIES	7
2. HELLO MOTION	8
3. USING THE SMOTION CLASS	9
3.1. USING THE INITIALIZE() METHOD.....	9
3.2. HANDLING SSDKUNSUPPORTEDEXCEPTION	9
3.3. CHECKING THE AVAILABILITY OF MOTION FEATURES	10
4. USING MOTION	11
4.1. RECEIVING MOTION DATA	11
4.2. USING THE MOTION TYPES.....	12
4.2.1. <i>Using Call Motion</i>	12
4.2.2. <i>Using Pedometer</i>	14
4.2.3. <i>Tracking User Activities</i>	16
4.2.4. <i>Using Activity Notifications</i>	17
COPYRIGHT	19

1. Overview

Motion allows you to retrieve pedometer and activity information in your application. Motion processes raw data from the device motion sensors to collect pedometer and activity information.

- Access information about calls initiated by device motion.
- Access pedometer information.
- Access activity information.

1.1. Architecture

The following figure shows the Motion architecture.

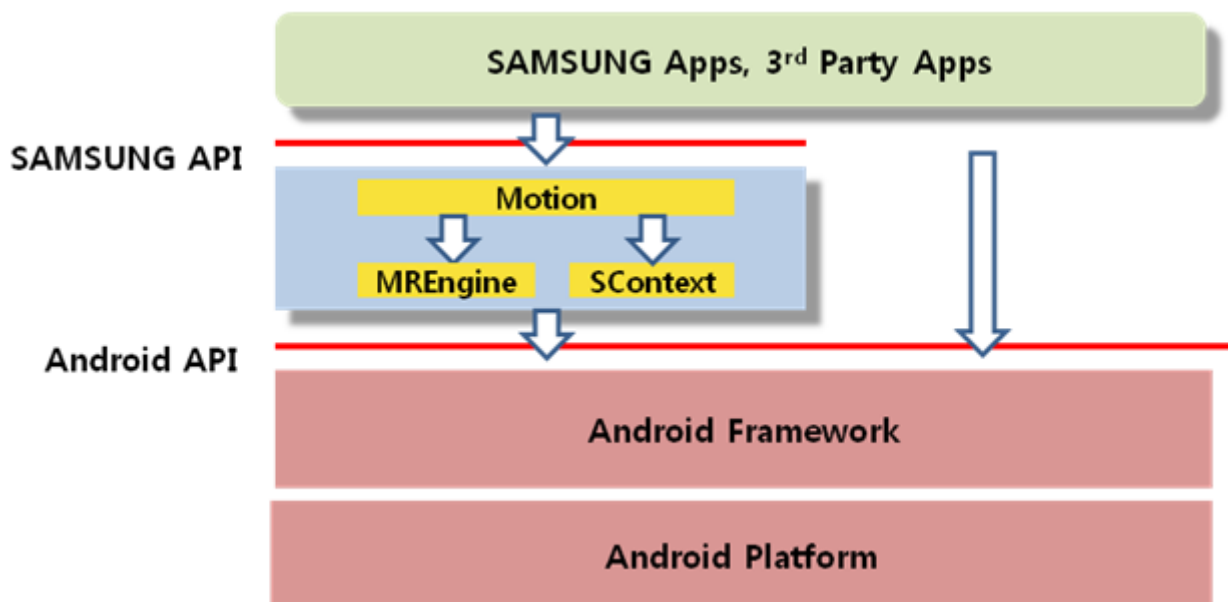


Figure 1: Motion architecture

The architecture consists of:

- **Applications:** One or more applications that use Motion.
- **Motion:** Motion components for managing specific pedometer and activity events.
- **MREngine:** Motion components for providing Motion with call motion events.
- **SContext:** Motion components for providing Motion with pedometer and activity events.

1.2. Class Diagram

The following figure shows the Motion classes and interfaces that you can use in your application.

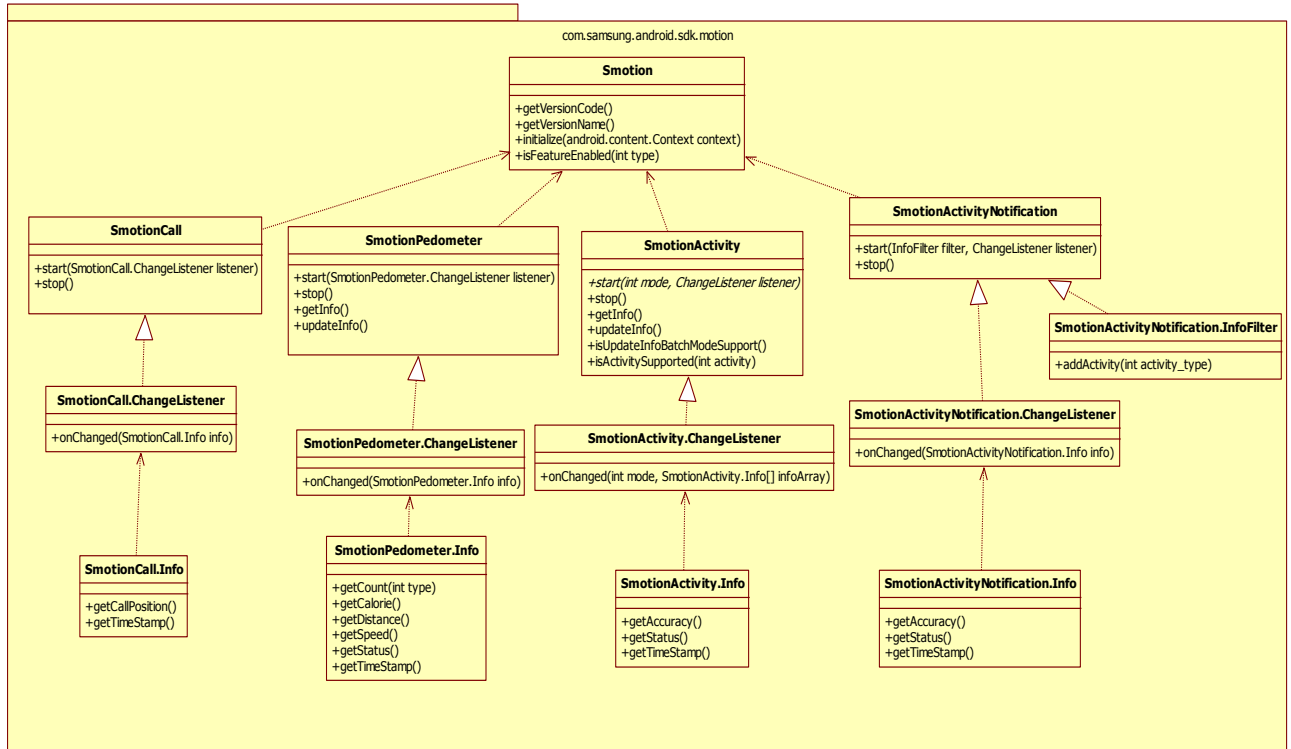


Figure 2: Motion classes and interfaces

The Motion classes and interfaces include:

- **Smotion**: Initializes the Motion.
- **SmotionCall**: Provides access to call information for calls placed based on device motion.
- **SmotionPedometer**: Provides access to pedometer information.
- **SmotionActivity**: Provides access to activity information.
- **SmotionActivityNotification**: Provides access to activity notification information for specified activity events.
- **SmotionCall.Info**: Contains call motion information.
- **SmotionPedometer.Info**: Contains pedometer information.
- **SmotionActivity.Info**: Contains activity information.
- **SmotionActivityNotification.Info**: Contains activity notification information.
- **SmotionCall.ChangeListener**: Listens for call motion events.
- **SmotionPedometer.ChangeListener**: Listens for pedometer events.
- **SmotionActivity.ChangeListener**: Listens for activity events.

- **SmotionActivityNotification.Changelistener**: Listens for activity notification events.
- **SmotionActivityNotification.InfoFilter**: Creates specified activity notification actions.

1.3. Supported Platforms

- Android 4.3 (Android API level 18) or above supports Motion.
- Android 4.4 (Android API level 19) or above supports SmotionActivity and SmotionActivityNotification.

1.4. Supported Features

Motion supports the following features:

- Accessing information on calls placed based on device motion
- Accessing pedometer information
- Accessing activity information

1.5. Components

- Components
 - motion-v2.2.0.jar
- Package to be imported:
 - com.samsung.android.sdk.motion

1.6. Importing Libraries

To import Motion libraries to the application project:

1. Add the motion-v2.2.0.jar file to the libs folder in Eclipse.

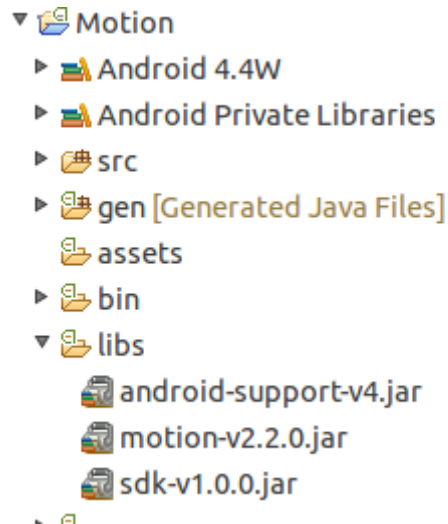


Figure 3: libs folder in Eclipse

The following permission has to be specified in the AndroidManifest.xml file to initialize Motion.

```
<uses-permission android:name="com.samsung.android.providers.context.permission.WRITE_USE_APP_FEATURE_SURVEY"/>
```

If you don't add the permission,

- o Galaxy S5 and above: SecurityException is thrown and your application doesn't work.
- o Prior to Galaxy S5: No exception. And the application works properly.

2. Hello Motion

Hello Motion is a simple program that:

1. Creates Smotion and SmotionPedometer instances.
2. Implements, registers, and starts an SmotionPedometer.ChangeListener instance.
3. Handles motion events in the ChangeListener.onChanged() method.
4. Stops the ChangeListener instance.

```
public class MainActivity extends Activity {
    private Smotion mMotion;
    private SmotionPedometer mPedometer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mMotion = new Smotion();

        try {
            mMotion.initialize(this);
        } catch (IllegalArgumentException e) {
            // Error handling
        } catch (SsdkUnsupportedException e) {
            // Error handling
        }

        // Create SmotionPedometer instance
        mPedometer = new SmotionPedometer(Looper.getMainLooper(), mMotion);
        // Start Pedometer
        mPedometer.start(changeListener);
    }

    @Override
    protected void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();
        // Stop pedometer
        mPedometer.stop();
    }

    final SmotionPedometer.ChangeListener changeListener = new
        SmotionPedometer.ChangeListener() {

        @Override
        public void onChanged(Info info) {
            // TODO Auto-generated method stub
            SmotionPedometer.Info pedometerInfo = info;
            System.out.println("HelloMotion Pedometer");
        }
    };
}
```


3. Using the Smotion Class

The Smotion class provides the following methods:

- `initialize()` initializes Motion. You need to initialize Motion before you can use it. If the device does not support Motion, `SsdkUnsupportedException` is thrown.
- `getVersionCode()` gets the Motion version number as an integer.
- `getVersionName()` gets the Motion version name as a string.
- `isFeatureEnabled(int type)` checks if the Motion feature is available on the device.

```
Smotion mMotion = new Smotion();
try {
    mMotion.initialize(this);
} catch (IllegalArgumentException e) {
    // Error handling
} catch (SsdkUnsupportedException e) {
    // Error handling
}
```

3.1. Using the initialize() Method

The `Smotion.initialize()` method:

- Initializes Motion
- Checks if the device is a Samsung device
- Checks if the device supports Motion
- Checks if the Motion libraries are installed on the device

```
void initialize(Context context) throws SsdkUnsupportedException
```

If Motion fails to initialize, the `initialize()` method throws an `SsdkUnsupportedException` exception. To find out the reason for the exception, check the exception message.

3.2. Handling SsdkUnsupportedException

If an `SsdkUnsupportedException` exception is thrown, check the exception message type using `SsdkUnsupportedException.getType()`.

The following types of exception messages are defined in the Smotion class:

- **VENDOR_NOT_SUPPORTED:** The device is not a Samsung device.
- **DEVICE_NOT_SUPPORTED:** The device does not support the Motion.

3.3. Checking the Availability of Motion Features

You can check if a Motion feature is supported on the device with the `isFeatureEnabled()` method. The feature types are defined in the `Smotion` class. Pass the feature type as a parameter when calling the `isFeatureEnabled()` method. The method returns a Boolean value that indicates the support for the feature on the device.

```
boolean isFeatureEnabled(int type);
```

The following types are defined in the `Smotion` class:

- `TYPE_CALL`
- `TYPE_PEDOMETER`
- `TYPE_PEDOMETER_WITH_UPDOWN_STEP`
- `TYPE_ACTIVITY`
- `TYPE_ACTIVITY_NOTIFICATION`

4. Using Motion

This section describes how to use the Motion in your application.

4.1. Receiving Motion Data

To initialize Motion and receive motion data:

1. Create an Smotion instance.
2. Pass the Smotion instance as a parameter to create an SmotionCall, SmotionPedometer, SmotionActivity, or SmotionActivityNotification instance.
3. Call `start()` to register a `ChangeListener` instance for SmotionCall, SmotionPedometer, SmotionActivity, or SmotionActivityNotification. When Motion starts, the SmotionCall, SmotionPedometer, SmotionActivity, or SmotionActivityNotification instance receives a callback to the `ChangeListener`.
4. In the `onChanged(Info info)` method, handle the Motion events.
5. Call `stop()` to remove the `ChangeListener` instance.

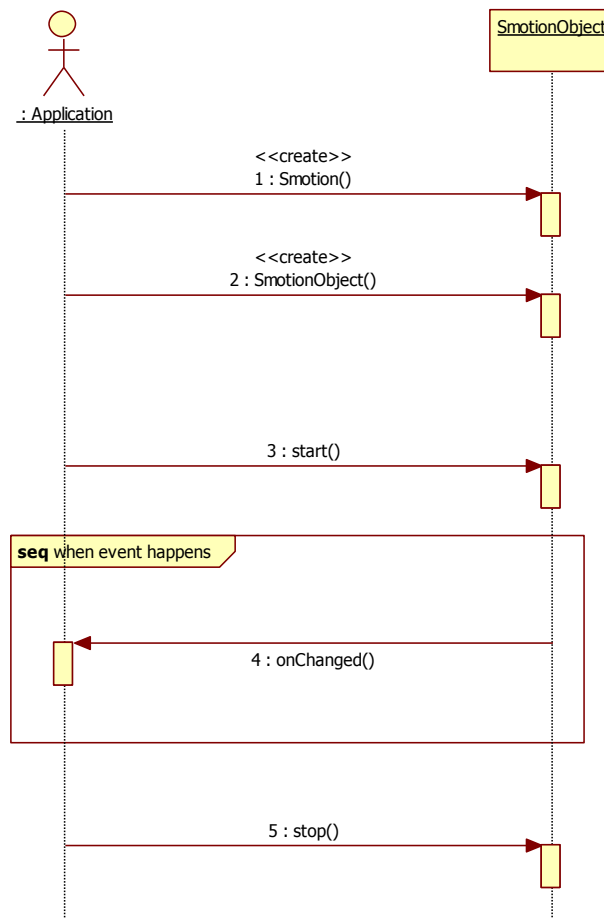


Figure 4: Receiving data from Motion

```
Smotion mMotion;
SmotionPedometer mPedometer;

// Initialize
mMotion = new Smotion();
try {
    mMotion.initialize(this);
} catch (IllegalArgumentException e) {
    // Error handling
} catch (SsdkUnsupportedException e) {
    // Error handling
}

// Create SmotionPedometer instance (SmotionCall, SmotionActivity, and
// SmotionActivityNotification follows the same pattern)
mPedometer = new SmotionPedometer(Looper.getMainLooper(), mMotion);

// Implement ChangeListener
final SmotionPedometer.ChangeListener changeListener = new
    SmotionPedometer.ChangeListener() {
        @Override
        public void onChanged(Info info) {
            // TODO Auto-generated method stub
            SmotionPedometer.Info pedometerInfo = info;
        }
    };

// Add Smotion Listener
mPedometer.start(changeListener);

// Remove Smotion Listener
mPedometer.stop();
```

4.2. Using the Motion Types

This section describes how to use the various motion types in your application.

4.2.1. Using Call Motion

The call motion recognizes the motion of first watching the device and then bringing the device up to your ear. It includes distinguishing between holding the device next to the left ear or the right ear.



Figure 5: Call motion in action

SmotionCall recognizes when you place the device next to your ear. When the call motion is recognized, the device dials the currently displayed on-screen Contact entry as soon as you place the device to your ear.

You can use the `getTimeStamp()` method to get the timestamp to measure a duration by comparing it against another timestamp from the same process on the same device. The timestamp does not have a defined correspondence to wall clock times. The zero value is typically whenever the device was last booted. You can use `System.currentTimeMillis()` to get the current time.

You can use the `getCallPosition()` method to get which ear the device is next to.

```
public class MainActivity extends Activity {
    private Smotion mMotion;
    private SmotionCall mCall;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mCall= new SmotionCall(Looper.getMainLooper(), mMotion);
        mCall.start(changeListener);
    }

    final SmotionCall.ChangeListener changeListener = new
        SmotionCall.ChangeListener(){

        @Override
        public void onChanged(Info info) {
            // TODO Auto-generated method stub
            switch(info.getCallPosition()){
                case SmotionCall.POSITION_LEFT:
                    break;
                case SmotionCall.POSITION_RIGHT:
                    break;
            }
        }
    };
}
```

4.2.2. Using Pedometer

SmotionPedometer offers you the following methods to get pedometer data:

- `getCount(int type)` gets the steps by type.
- `getSpeed()` gets the walking speed.
- `getDistance()` gets the distance moved.
- `getCalorie()` gets the calories burned.
- `getStatus()` gets the walking status.
- `updateInfo()` gets the accumulated data from the selected date.
- `getInfo()` gets the accumulated pedometer data by type from when the device was last booted.



Figure 6: Pedometer usage example

If you start walking or running after `SmotionPedometer.start()` is called, `SmotionPedometer` captures the number of steps, the speed, the distance covered, and the calories consumed.

You can use the `getTimeStamp()` method to get the timestamp to measure a duration by comparing it against another timestamp from the same process on the same device. The timestamp does not have a defined correspondence to wall clock times. The zero value is typically whenever the device was last booted. You can use `System.currentTimeMillis()` to get the current time.

Note

TYPE_PEDOMETER_WITH_UPDOWN_STEP

If a device does not support this feature,

- when current status is STATUS_RUN_UP or STATUS_RUN_DOWN or STATUS_WALK_UP or STATUS_WALK_DOWN, getCount(int type) always returns 0.
- getStatus() does not return STATUS_RUN_UP, STATUS_RUN_DOWN, STATUS_WALK_UP, STATUS_WALK_DOWN.

Setting the user profile

The user height, weight, and sex can only be set in the SHealth application on the device. The profile settings are provided exclusively in SHealth because they can affect the measurement of speed, distance or calories. For more precise measurement of calories, distance or speed, encourage the users of your application to set their profile (height, weight and sex) in SHealth.

The following sample code shows how to use a listener to receive data.

```
public class MainActivity extends Activity {
    private Smotion mMotion;
    private SmotionPedometer mPedometer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        mPedometer = new SmotionPedometer(Looper.getMainLooper(), mMotion);
        mPedometer.start(changeListener);
    }

    @Override
    protected void onDestroy() {
        ...
    }

    // Update the pedometer data by listener callback.
    final SmotionPedometer.ChangeListener changeListener =
        new SmotionPedometer.ChangeListener() {

            @Override
            public void onChanged(Info info) {
                // TODO Auto-generated method stub
                double calorie = info.getCalorie();
                double distance = info.getDistance();
                double speed = info.getSpeed();
                long count = info.getCount(SmotionPedometer.Info.COUNT_TOTAL);
                int status = info.getStatus();
            }
        };
}
```

The following sample code shows how to receive data from the pedometer on an hourly basis by using the `SmotionPedometer.getInfo()` method. You can use this method to retrieve the latest `SmotionPedometer.Info` object for your application without waiting for a change event.

```
public class MainActivity extends Activity {
    private Smotion mMotion;
    private SmotionPedometer mPedometer;
    private SmotionPedometer.Info mInfo;
    private final int WAITING_TIME = 3600000;
    private Timer mTimer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mPedometer = new SmotionPedometer(Looper.getMainLooper(), mMotion);
        mPedometer.start(changeListener);
        mTimer = new Timer();
        mTimer.schedule(new CustomTimer(), WAITING_TIME);
    }

    private class CustomTimer extends TimerTask{
        @Override
        public void run(){
            mInfo = mPedometer.getInfo();
        }
    }
}
```

When you want to receive the current data from the pedometer through the listener, call the `SmotionPedometer.updateInfo()` method.

Note: If you call `SmotionPedometer.getInfo()` or `SmotionPedometer.updateInfo()` when the screen is off, the retrieved status value is invalid. All other values are valid.

4.2.3. Tracking User Activities

`SmotionActivity` offers you the following methods to get activity data:

- `getStatus()` gets the user activity status.
- `getAccuracy()` gets the accuracy of the detected activity.
- `getTimeStamp()` gets the timestamp in milliseconds.

If you start walking, running or getting in a vehicle after calling `SmotionActivity.start()`, `SmotionActivity` captures the activity status and accuracy of the activity.

You can track activities in the following modes:

- Real time: When the status or accuracy changes, your application can receive activity information while the device's screen is on.

- Batch: The batch FIFO stores the timestamp, status and accuracy. When the FIFO is full, your application can receive activity information.

The following sample code shows how to use a listener for receiving data.

```
public class MainActivity extends Activity {
    private Smotion mMotion;
    private SmotionActivity mActivity;
    private int mActivityMode = SmotionActivity.Info.MODE_ALL;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mActivity = new SmotionActivity(Looper.getMainLooper(), mMotion);
        mActivity.start(mActivityMode, changeListener);
    }

    @Override
    protected void onDestroy() {
    }

    // Update the activity data by listener callback.
    final SmotionActivity.ChangeListener changeListener =
        new SmotionActivity.ChangeListener() {

        @Override
        public void onChanged(int mode, Info[] infoArray) {
            // TODO Auto-generated method stub
            if(mode==SmotionActivity.Info.MODE_REALTIME){
                int status = infoArray[0].getStatus();
                int accuracy = infoArray[0].getAccuracy();
                long timestamp = infoArray[0].getTimeStamp();
            }else if(mode==SmotionActivity.Info.MODE_BATCH){
                for(int i=0;i<infoArray.length;i++){
                    int status = infoArray[i].getStatus();
                    int accuracy = infoArray[i].getAccuracy();
                    long timestamp = infoArray[i].getTimeStamp();
                }
            }
        }
    };
}
```

If you want to receive the current data from the activity through the listener, call the `SmotionActivity.updateInfo()` method. To check if the device supports `updateInfo` method in batch mode, call the `isUpdateInfoBatchModeSupport()` method. And to check if the device supports each activity type (stationary, walk, run, vehicle, bike), call the `isActivitySupported()` method.

4.2.4. Using Activity Notifications

`SmotionActivityNotification` offers you the following methods to get specific activity data with notifications:

- `getStatus()` gets the user activity status.

- `getAccuracy()` gets the accuracy of the detected activity.
- `getTimeStamp()` gets the timestamp in milliseconds.

You can select a specific activity type for notifications using the `InfoFilter.addActivity()` method. If you start walking or running or getting in a vehicle after `SmotionActivityNotification.start()` is called, `SmotionActivityNotification` captures the specific activity status and accuracy of the activity. And to check if the device supports each activity type (stationary, walk, run, vehicle, bike), call the `isActivitySupported()` method.

The following sample code shows how to use a listener for receiving data.

```
public class MainActivity extends Activity {
    private Smotion mMotion;
    private SmotionActivityNotification mActivityNotification;
    private SmotionActivityNotification.InfoFilter mFilter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mActivityNotification =
            new SmotionActivityNotification(Looper.getMainLooper(), mMotion);
        mFilter = new InfoFilter();
        mFilter.addActivity(SmotionActivityNotification.Info.STATUS_VEHICLE);
        mActivityNotification.start(mFilter, changeListener);
    }

    @Override
    protected void onDestroy() {
        ...
    }

    // Update the activity data by listener callback.
    final SmotionActivityNotification.ChangeListener changeListener =
        new SmotionActivityNotification.ChangeListener() {
            @Override
            public void onChanged(Info info) {
                // TODO Auto-generated method stub
                int status = info.getStatus();
                int accuracy = info.getAccuracy();
                long timestamp = info.getTimeStamp();
            }
        };
}
```

Copyright

Copyright © 2014 Samsung Electronics Co. Ltd. All Rights Reserved.

Though every care has been taken to ensure the accuracy of this document, Samsung Electronics Co., Ltd. cannot accept responsibility for any errors or omissions or for any loss occurred to any person, whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

Samsung Electronics Co. Ltd. may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

No part of this document may be communicated, distributed, reproduced or transmitted in any form or by any means, electronic or mechanical or otherwise, for any purpose, without the prior written permission of Samsung Electronics Co. Ltd.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

For more information, please visit <http://developer.samsung.com/>