SensorManager

Added in API level 1


```
public abstract class SensorManager
extends Object
```

java.lang.Object

↳ android.hardware.SensorManager

SensorManager lets you access the device's sensors. Get an instance of this class by calling Context.getSystemService() with the argument SENSOR_SERVICE.

Always make sure to disable sensors you don't need, especially when your activity is paused. Failing to do so can drain the battery in just a few hours. Note that the system will *not* disable sensors automatically when the screen turns off.

> Note: Don't use this mechanism with a Trigger Sensor, have a look at TriggerEventListener. TYPE_SIGNIFICANT_MOTION is an example of a trigger sensor.

```
public class SensorActivity extends Activity implements SensorEventListener {
    private final SensorManager mSensorManager;
    private final Sensor mAccelerometer;

    public SensorActivity() {
        mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    protected void onResume() {
        super.onResume();
```

```
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }

    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    public void onSensorChanged(SensorEvent event) {
    }
}
```

See also:

SensorEventListener

SensorEvent

Sensor

Summary

Nested classes	
`class`	SensorManager.DynamicSensorCallback Used for receiving notifications from the SensorManager when dynamic sensors are connected or disconnected.

Constants	
`int`	AXIS_MINUS_X see remapCoordinateSystem(float[], int, int, float[])
`int`	AXIS_MINUS_Y

	see remapCoordinateSystem(float[], int, int, float[])
`int`	AXIS_MINUS_Z see remapCoordinateSystem(float[], int, int, float[])
`int`	AXIS_X see remapCoordinateSystem(float[], int, int, float[])
`int`	AXIS_Y see remapCoordinateSystem(float[], int, int, float[])
`int`	AXIS_Z see remapCoordinateSystem(float[], int, int, float[])
`int`	DATA_X *This constant was deprecated in API level 3. use Sensor instead.*
`int`	DATA_Y *This constant was deprecated in API level 3. use Sensor instead.*
`int`	DATA_Z *This constant was deprecated in API level 3. use Sensor instead.*
`float`	GRAVITY_DEATH_STAR_I Gravity (estimate) on the first Death Star in Empire units (m/s^2)
`float`	GRAVITY_EARTH Earth's gravity in SI units (m/s^2)
`float`	GRAVITY_JUPITER Jupiter's gravity in SI units (m/s^2)
`float`	GRAVITY_MARS Mars' gravity in SI units (m/s^2)
`float`	GRAVITY_MERCURY Mercury's gravity in SI units (m/s^2)
`float`	GRAVITY_MOON The Moon's gravity in SI units (m/s^2)

`float`	GRAVITY_NEPTUNE Neptune's gravity in SI units (m/s^2)
`float`	GRAVITY_PLUTO Pluto's gravity in SI units (m/s^2)
`float`	GRAVITY_SATURN Saturn's gravity in SI units (m/s^2)
`float`	GRAVITY_SUN Sun's gravity in SI units (m/s^2)
`float`	GRAVITY_THE_ISLAND Gravity on the island
`float`	GRAVITY_URANUS Uranus' gravity in SI units (m/s^2)
`float`	GRAVITY_VENUS Venus' gravity in SI units (m/s^2)
`float`	LIGHT_CLOUDY luminance under a cloudy sky in lux
`float`	LIGHT_FULLMOON luminance at night with full moon in lux
`float`	LIGHT_NO_MOON luminance at night with no moon in lux
`float`	LIGHT_OVERCAST luminance under an overcast sky in lux
`float`	LIGHT_SHADE luminance in shade in lux
`float`	LIGHT_SUNLIGHT luminance of sunlight in lux
`float`	LIGHT_SUNLIGHT_MAX

	Maximum luminance of sunlight in lux
`float`	LIGHT_SUNRISE luminance at sunrise in lux
`float`	MAGNETIC_FIELD_EARTH_MAX Maximum magnetic field on Earth's surface
`float`	MAGNETIC_FIELD_EARTH_MIN Minimum magnetic field on Earth's surface
`float`	PRESSURE_STANDARD_ATMOSPHERE Standard atmosphere, or average sea-level pressure in hPa (millibar)
`int`	RAW_DATA_INDEX *This constant was deprecated in API level 3. use Sensor instead.*
`int`	RAW_DATA_X *This constant was deprecated in API level 3. use Sensor instead.*
`int`	RAW_DATA_Y *This constant was deprecated in API level 3. use Sensor instead.*
`int`	RAW_DATA_Z *This constant was deprecated in API level 3. use Sensor instead.*
`int`	SENSOR_ACCELEROMETER *This constant was deprecated in API level 3. use Sensor instead.*
`int`	SENSOR_ALL *This constant was deprecated in API level 3. use Sensor instead.*
`int`	SENSOR_DELAY_FASTEST get sensor data as fast as possible
`int`	SENSOR_DELAY_GAME rate suitable for games
`int`	SENSOR_DELAY_NORMAL rate (default) suitable for screen orientation changes

int	SENSOR_DELAY_UI rate suitable for the user interface
int	SENSOR_LIGHT *This constant was deprecated in API level 3. use Sensor instead.*
int	SENSOR_MAGNETIC_FIELD *This constant was deprecated in API level 3. use Sensor instead.*
int	SENSOR_MAX *This constant was deprecated in API level 3. use Sensor instead.*
int	SENSOR_MIN *This constant was deprecated in API level 3. use Sensor instead.*
int	SENSOR_ORIENTATION *This constant was deprecated in API level 3. use Sensor instead.*
int	SENSOR_ORIENTATION_RAW *This constant was deprecated in API level 3. use Sensor instead.*
int	SENSOR_PROXIMITY *This constant was deprecated in API level 3. use Sensor instead.*
int	SENSOR_STATUS_ACCURACY_HIGH This sensor is reporting data with maximum accuracy
int	SENSOR_STATUS_ACCURACY_LOW This sensor is reporting data with low accuracy, calibration with the environment is needed
int	SENSOR_STATUS_ACCURACY_MEDIUM This sensor is reporting data with an average level of accuracy, calibration with the environment may improve the readings
int	SENSOR_STATUS_NO_CONTACT The values returned by this sensor cannot be trusted because the sensor had no contact with what it was measuring (for example, the heart rate monitor is not in contact with the user).
int	SENSOR_STATUS_UNRELIABLE The values returned by this sensor cannot be trusted, calibration is needed or the environment doesn't allow readings

`int`	SENSOR_TEMPERATURE *This constant was deprecated in API level 3. use Sensor instead.*
`int`	SENSOR_TRICORDER *This constant was deprecated in API level 3. use Sensor instead.*
`float`	STANDARD_GRAVITY Standard gravity (g) on Earth.

Public methods	
`boolean`	cancelTriggerSensor(TriggerEventListener `listener`, Sensor `sensor`) Cancels receiving trigger events for a trigger sensor.
`boolean`	flush(SensorEventListener `listener`) Flushes the FIFO of all the sensors registered for this listener.
`static float`	getAltitude(`float p0, float p`) Computes the Altitude in meters from the atmospheric pressure and the pressure at sea level.
`static void`	getAngleChange(`float[] angleChange, float[] R, float[] prevR`) Helper function to compute the angle change between two rotation matrices.
Sensor	getDefaultSensor(`int type`) Use this method to get the default sensor for a given type.
Sensor	getDefaultSensor(`int type, boolean wakeUp`) Return a Sensor with the given type and wakeUp properties.
List<Sensor>	getDynamicSensorList(`int type`) Use this method to get a list of available dynamic sensors of a certain type.
`static float`	getInclination(`float[] I`) Computes the geomagnetic inclination angle in radians from the inclination matrix **I** returned by getRotationMatrix(float[], float[], float[], float[]).
`static float[]`	getOrientation(`float[] R, float[] values`) Computes the device's orientation based on the rotation matrix.

static void	getQuaternionFromVector(float[] Q, float[] rv) Helper function to convert a rotation vector to a normalized quaternion.
static boolean	getRotationMatrix(float[] R, float[] I, float[] gravity, float[] geomagnetic) Computes the inclination matrix **I** as well as the rotation matrix **R** transforming a vector from the device coordinate system to the world's coordinate system which is defined as a direct orthonormal basis, where: • X is defined as the vector product **Y.Z** (It is tangential to the ground at the device's current location and roughly points East).
static void	getRotationMatrixFromVector(float[] R, float[] rotationVector) Helper function to convert a rotation vector to a rotation matrix.
List<Sensor>	getSensorList(int type) Use this method to get the list of available sensors of a certain type.
int	getSensors() *This method was deprecated in API level 3. This method is deprecated, use* getSensorList(int) *instead*
boolean	isDynamicSensorDiscoverySupported() Tell if dynamic sensor discovery feature is supported by system.
void	registerDynamicSensorCallback(SensorManager.DynamicSensorCallback callback) Add a DynamicSensorCallback to receive dynamic sensor connection callbacks.
void	registerDynamicSensorCallback(SensorManager.DynamicSensorCallback callback, Handler handler) Add a DynamicSensorCallback to receive dynamic sensor connection callbacks.
boolean	registerListener(SensorEventListener listener, Sensor sensor, int samplingPeriodUs) Registers a SensorEventListener for the given sensor at the given sampling frequency.
boolean	registerListener(SensorEventListener listener, Sensor sensor, int samplingPeriodUs, int maxReportLatencyUs) Registers a SensorEventListener for the given sensor at the given sampling frequency and the given maximum reporting latency.
boolean	registerListener(SensorEventListener listener, Sensor sensor, int samplingPeriodUs, Handler handler) Registers a SensorEventListener for the given sensor.
boolean	registerListener(SensorListener listener, int sensors) *This method was deprecated in API level 3. This method is deprecated, use* registerListener(SensorEventListener, Sensor, int) *instead.*
boolean	registerListener(SensorListener listener, int sensors, int rate)

	This method was deprecated in API level 3. This method is deprecated, use registerListener(SensorEventListener, Sensor, int) *instead.*
`boolean`	registerListener(SensorEventListener `listener, Sensor sensor, int samplingPeriodUs, int maxReportLatencyUs,` Handler `handler)` Registers a SensorEventListener for the given sensor at the given sampling frequency and the given maximum reporting latency.
`static boolean`	remapCoordinateSystem`(float[] inR, int X, int Y, float[] outR)` Rotates the supplied rotation matrix so it is expressed in a different coordinate system.
`boolean`	requestTriggerSensor(TriggerEventListener `listener, Sensor sensor)` Requests receiving trigger events for a trigger sensor.
`void`	unregisterDynamicSensorCallback(SensorManager.DynamicSensorCallback `callback)` Remove a DynamicSensorCallback to stop sending dynamic sensor connection events to that callback.
`void`	unregisterListener(SensorEventListener `listener)` Unregisters a listener for all sensors.
`void`	unregisterListener(SensorEventListener `listener, Sensor sensor)` Unregisters a listener for the sensors with which it is registered.
`void`	unregisterListener(SensorListener `listener)` *This method was deprecated in API level 3. This method is deprecated, use* unregisterListener(SensorEventListener) *instead.*
`void`	unregisterListener(SensorListener `listener, int sensors)` *This method was deprecated in API level 3. This method is deprecated, use* unregisterListener(SensorEventListener, Sensor) *instead.*

Inherited methods

From class java.lang.Object

Constants

AXIS_MINUS_X

Added in API level 3

```
int AXIS_MINUS_X
```

see remapCoordinateSystem(float[], int, int, float[])

Constant Value: 129 (0x00000081)

AXIS_MINUS_Y

Added in API level 3

```
int AXIS_MINUS_Y
```

see remapCoordinateSystem(float[], int, int, float[])

Constant Value: 130 (0x00000082)

AXIS_MINUS_Z

Added in API level 3

```
int AXIS_MINUS_Z
```

see remapCoordinateSystem(float[], int, int, float[])

Constant Value: 131 (0x00000083)

AXIS_X

Added in API level 3

```
int AXIS_X
```

see remapCoordinateSystem(float[], int, int, float[])

Constant Value: 1 (0x00000001)

AXIS_Y

Added in API level 3

```
int AXIS_Y
```

see remapCoordinateSystem(float[], int, int, float[])

Constant Value: 2 (0x00000002)

AXIS_Z

Added in API level 3

```
int AXIS_Z
```

see remapCoordinateSystem(float[], int, int, float[])

Constant Value: 3 (0x00000003)

DATA_X

Added in API level 1

```
int DATA_X
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Index of the X value in the array returned by onSensorChanged(int, float[])

Constant Value: 0 (0x00000000)

DATA_Y

Added in API level 1

```
int DATA_Y
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Index of the Y value in the array returned by onSensorChanged(int, float[])

Constant Value: 1 (0x00000001)

DATA_Z

Added in API level 1

```
int DATA_Z
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Index of the Z value in the array returned by onSensorChanged(int, float[])

Constant Value: 2 (0x00000002)

GRAVITY_DEATH_STAR_I

Added in API level 1

```
float GRAVITY_DEATH_STAR_I
```

Gravity (estimate) on the first Death Star in Empire units (m/s^2)

Constant Value: 3.5303614E-7

GRAVITY_EARTH

Added in API level 1

```
float GRAVITY_EARTH
```

Earth's gravity in SI units (m/s^2)

Constant Value: 9.80665

GRAVITY_JUPITER

Added in API level 1

```
float GRAVITY_JUPITER
```

Jupiter's gravity in SI units (m/s^2)

Constant Value: 23.12

GRAVITY_MARS

Added in API level 1

```
float GRAVITY_MARS
```

Mars' gravity in SI units (m/s^2)

Constant Value: 3.71

GRAVITY_MERCURY

Added in API level 1

```
float GRAVITY_MERCURY
```

Mercury's gravity in SI units (m/s^2)

Constant Value: 3.7

GRAVITY_MOON

Added in API level 1

```
float GRAVITY_MOON
```

The Moon's gravity in SI units (m/s^2)

Constant Value: 1.6

GRAVITY_NEPTUNE

Added in API level 1

```
float GRAVITY_NEPTUNE
```

Neptune's gravity in SI units (m/s^2)

Constant Value: 11.0

GRAVITY_PLUTO

Added in API level 1

```
float GRAVITY_PLUTO
```

Pluto's gravity in SI units (m/s^2)

Constant Value: 0.6

GRAVITY_SATURN

Added in API level 1

```
float GRAVITY_SATURN
```

Saturn's gravity in SI units (m/s^2)

Constant Value: 8.96

GRAVITY_SUN

Added in API level 1

```
float GRAVITY_SUN
```

Sun's gravity in SI units (m/s^2)

Constant Value: 275.0

GRAVITY_THE_ISLAND

Added in API level 1

```
float GRAVITY_THE_ISLAND
```

Gravity on the island

Constant Value: 4.815162

GRAVITY_URANUS

Added in API level 1

```
float GRAVITY_URANUS
```

Uranus' gravity in SI units (m/s^2)

Constant Value: 8.69

GRAVITY_VENUS

Added in API level 1

```
float GRAVITY_VENUS
```

Venus' gravity in SI units (m/s^2)

Constant Value: 8.87

LIGHT_CLOUDY

Added in API level 1

```
float LIGHT_CLOUDY
```

luminance under a cloudy sky in lux

Constant Value: 100.0

LIGHT_FULLMOON

Added in API level 1

```
float LIGHT_FULLMOON
```

luminance at night with full moon in lux

Constant Value: 0.25

LIGHT_NO_MOON

Added in API level 1

```
float LIGHT_NO_MOON
```

luminance at night with no moon in lux

Constant Value: 0.001

LIGHT_OVERCAST

Added in API level 1

```
float LIGHT_OVERCAST
```

luminance under an overcast sky in lux

Constant Value: 10000.0

LIGHT_SHADE

Added in API level 1

```
float LIGHT_SHADE
```

luminance in shade in lux

Constant Value: 20000.0

LIGHT_SUNLIGHT

Added in API level 1

```
float LIGHT_SUNLIGHT
```

luminance of sunlight in lux

Constant Value: 110000.0

LIGHT_SUNLIGHT_MAX

Added in API level 1

```
float LIGHT_SUNLIGHT_MAX
```

Maximum luminance of sunlight in lux

Constant Value: 120000.0

LIGHT_SUNRISE

Added in API level 1

```
float LIGHT_SUNRISE
```

luminance at sunrise in lux

Constant Value: 400.0

MAGNETIC_FIELD_EARTH_MAX

Added in API level 1

```
float MAGNETIC_FIELD_EARTH_MAX
```

Maximum magnetic field on Earth's surface

Constant Value: 60.0

MAGNETIC_FIELD_EARTH_MIN

Added in API level 1

```
float MAGNETIC_FIELD_EARTH_MIN
```

Minimum magnetic field on Earth's surface

Constant Value: 30.0

PRESSURE_STANDARD_ATMOSPHERE

Added in API level 9

```
float PRESSURE_STANDARD_ATMOSPHERE
```

Standard atmosphere, or average sea-level pressure in hPa (millibar)

Constant Value: 1013.25

RAW_DATA_INDEX

Added in API level 1

```
int RAW_DATA_INDEX
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Offset to the untransformed values in the array returned by onSensorChanged(int, float[])

Constant Value: 3 (0x00000003)

RAW_DATA_X

Added in API level 1

```
int RAW_DATA_X
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Index of the untransformed X value in the array returned by onSensorChanged(int, float[])

Constant Value: 3 (0x00000003)

RAW_DATA_Y

Added in API level 1

```
int RAW_DATA_Y
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Index of the untransformed Y value in the array returned by onSensorChanged(int, float[])

Constant Value: 4 (0x00000004)

RAW_DATA_Z

Added in API level 1

```
int RAW_DATA_Z
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Index of the untransformed Z value in the array returned by onSensorChanged(int, float[])

Constant Value: 5 (0x00000005)

SENSOR_ACCELEROMETER

Added in API level 1

```
int SENSOR_ACCELEROMETER
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing an accelerometer. See SensorListener for more details.

Constant Value: 2 (0x00000002)

SENSOR_ALL

Added in API level 1

```
int SENSOR_ALL
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant that includes all sensors

Constant Value: 127 (0x0000007f)

SENSOR_DELAY_FASTEST

Added in API level 1

```
int SENSOR_DELAY_FASTEST
```

get sensor data as fast as possible

Constant Value: 0 (0x00000000)

SENSOR_DELAY_GAME

Added in API level 1

```
int SENSOR_DELAY_GAME
```

rate suitable for games

Constant Value: 1 (0x00000001)

SENSOR_DELAY_NORMAL

Added in API level 1

```
int SENSOR_DELAY_NORMAL
```

rate (default) suitable for screen orientation changes

Constant Value: 3 (0x00000003)

SENSOR_DELAY_UI

Added in API level 1

```
int SENSOR_DELAY_UI
```

rate suitable for the user interface

Constant Value: 2 (0x00000002)

SENSOR_LIGHT

Added in API level 1

```
int SENSOR_LIGHT
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing an ambient light sensor See SensorListener for more details.

Constant Value: 16 (0x00000010)

SENSOR_MAGNETIC_FIELD

Added in API level 1

```
int SENSOR_MAGNETIC_FIELD
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing a magnetic sensor See SensorListener for more details.

Constant Value: 8 (0x00000008)

SENSOR_MAX

Added in API level 1

```
int SENSOR_MAX
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Largest sensor ID

Constant Value: 64 (0x00000040)

SENSOR_MIN

Added in API level 1

```
int SENSOR_MIN
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

Smallest sensor ID

Constant Value: 1 (0x00000001)

SENSOR_ORIENTATION

Added in API level 1

```
int SENSOR_ORIENTATION
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing an orientation sensor. See SensorListener for more details.

Constant Value: 1 (0x00000001)

SENSOR_ORIENTATION_RAW

Added in API level 1

```
int SENSOR_ORIENTATION_RAW
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing an orientation sensor. See SensorListener for more details.

Constant Value: 128 (0x00000080)

SENSOR_PROXIMITY

Added in API level 1

```
int SENSOR_PROXIMITY
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing a proximity sensor See SensorListener for more details.

Constant Value: 32 (0x00000020)

SENSOR_STATUS_ACCURACY_HIGH

Added in API level 1

```
int SENSOR_STATUS_ACCURACY_HIGH
```

This sensor is reporting data with maximum accuracy

Constant Value: 3 (0x00000003)

SENSOR_STATUS_ACCURACY_LOW

Added in API level 1

```
int SENSOR_STATUS_ACCURACY_LOW
```

This sensor is reporting data with low accuracy, calibration with the environment is needed

Constant Value: 1 (0x00000001)

SENSOR_STATUS_ACCURACY_MEDIUM

Added in API level 1

```
int SENSOR_STATUS_ACCURACY_MEDIUM
```

This sensor is reporting data with an average level of accuracy, calibration with the environment may improve the readings

Constant Value: 2 (0x00000002)

SENSOR_STATUS_NO_CONTACT

Added in API level 20

```
int SENSOR_STATUS_NO_CONTACT
```

The values returned by this sensor cannot be trusted because the sensor had no contact with what it was measuring (for example, the heart rate monitor is not in contact with the user).

Constant Value: -1 (0xffffffff)

SENSOR_STATUS_UNRELIABLE

Added in API level 1

```
int SENSOR_STATUS_UNRELIABLE
```

The values returned by this sensor cannot be trusted, calibration is needed or the environment doesn't allow readings

Constant Value: 0 (0x00000000)

SENSOR_TEMPERATURE

Added in API level 1

```
int SENSOR_TEMPERATURE
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing a temperature sensor See SensorListener for more details.

Constant Value: 4 (0x00000004)

SENSOR_TRICORDER

Added in API level 1

```
int SENSOR_TRICORDER
```

> **This constant was deprecated in API level 3.**
> use Sensor instead.

A constant describing a Tricorder See SensorListener for more details.

Constant Value: 64 (0x00000040)

STANDARD_GRAVITY

Added in API level 1

```
float STANDARD_GRAVITY
```

Standard gravity (g) on Earth. This value is equivalent to 1G

Constant Value: 9.80665

Public methods

cancelTriggerSensor

Added in API level 18

```
boolean cancelTriggerSensor (TriggerEventListener listener,
                Sensor sensor)
```

Cancels receiving trigger events for a trigger sensor.

Note that a Trigger sensor will be auto disabled if onTrigger(TriggerEvent) has triggered. This method is provided in case the user wants to explicitly cancel the request to receive trigger events.

Parameters	
`listener`	`TriggerEventListener`: The listener on which the onTrigger(TriggerEvent) is delivered.It should be the same as the one used in requestTriggerSensor(TriggerEventListener, Sensor)
`sensor`	`Sensor`: The sensor for which the trigger request should be canceled. If null, it cancels receiving trigger for all sensors associated with the listener.

Returns	
`boolean`	true if successfully canceled.

Throws	
IllegalArgumentException	when sensor is a trigger sensor.

flush

Added in API level 19

```
boolean flush (SensorEventListener listener)
```

Flushes the FIFO of all the sensors registered for this listener. If there are events in the FIFO of the sensor, they are returned as if the maxReportLantecy of the FIFO has expired. Events are returned in the usual way through the SensorEventListener. This call doesn't affect the maxReportLantecy for this sensor. This call is asynchronous and returns immediately. onFlushCompleted is called after all the events in the batch at the time of calling this method have been delivered successfully. If the hardware doesn't support flush, it still returns true and a trivial flush complete event is sent after the current event for all the clients registered for this sensor.

Parameters

`listener`	`SensorEventListener`: A SensorEventListener object which was previously used in a registerListener call.

Returns	
`boolean`	`true` if the flush is initiated successfully on all the sensors registered for this listener, false if no sensor is previously registered for this listener or flush on one of the sensors fails.

Throws	
IllegalArgumentException	when listener is null.

See also:

registerListener(SensorEventListener, Sensor, int, int)

getAltitude

Added in API level 9

```
float getAltitude (float p0,
                   float p)
```

Computes the Altitude in meters from the atmospheric pressure and the pressure at sea level.

Typically the atmospheric pressure is read from a TYPE_PRESSURE sensor. The pressure at sea level must be known, usually it can be retrieved from airport databases in the vicinity. If unknown, you can use PRESSURE_STANDARD_ATMOSPHERE as an approximation, but absolute altitudes won't be accurate.

To calculate altitude differences, you must calculate the difference between the altitudes at both points. If you don't know the altitude as sea level, you can use PRESSURE_STANDARD_ATMOSPHERE instead, which will give good results considering the range of pressure typically involved.

```
float altitude_difference = getAltitude(SensorManager.PRESSURE_STANDARD_ATMOSPHERE, pressure_at_point2) -
getAltitude(SensorManager.PRESSURE_STANDARD_ATMOSPHERE, pressure_at_point1);
```

Parameters	
`p0`	`float`: pressure at sea level

`p`	`float`: atmospheric pressure

Returns	
`float`	Altitude in meters

getAngleChange

Added in API level 9

```
void getAngleChange (float[] angleChange,
                float[] R,
                float[] prevR)
```

Helper function to compute the angle change between two rotation matrices. Given a current rotation matrix (R) and a previous rotation matrix (prevR) computes the intrinsic rotation around the z, x, and y axes which transforms prevR to R. outputs a 3 element vector containing the z, x, and y angle change at indexes 0, 1, and 2 respectively.

Each input matrix is either as a 3x3 or 4x4 row-major matrix depending on the length of the passed array:

If the array length is 9, then the array elements represent this matrix

```
   /  R[ 0]   R[ 1]   R[ 2]   \
   |  R[ 3]   R[ 4]   R[ 5]   |
   \  R[ 6]   R[ 7]   R[ 8]   /
```

If the array length is 16, then the array elements represent this matrix

```
   /  R[ 0]   R[ 1]   R[ 2]   R[ 3]  \
   |  R[ 4]   R[ 5]   R[ 6]   R[ 7]  |
   |  R[ 8]   R[ 9]   R[10]   R[11]  |
   \  R[12]   R[13]   R[14]   R[15]  /
```

See getOrientation(float[], float[]) for more detailed definition of the output.

Parameters

`angleChange`	`float`: an an array of floats (z, x, and y) in which the angle change (in radians) is stored
`R`	`float`: current rotation matrix
`prevR`	`float`: previous rotation matrix

getDefaultSensor

Added in API level 3

Sensor `getDefaultSensor (int type)`

Use this method to get the default sensor for a given type. Note that the returned sensor could be a composite sensor, and its data could be averaged or filtered. If you need to access the raw sensors use getSensorList.

Parameters	
`type`	`int`: of sensors requested

Returns	
Sensor	the default sensor matching the requested type if one exists and the application has the necessary permissions, or null otherwise.

See also:

getSensorList(int)

Sensor

getDefaultSensor

Added in API level 21

```
Sensor getDefaultSensor (int type,
                boolean wakeUp)
```

Return a Sensor with the given type and wakeUp properties. If multiple sensors of this type exist, any one of them may be returned.

For example,

- getDefaultSensor(TYPE_ACCELEROMETER, true) returns a wake-up accelerometer sensor if it exists.
- getDefaultSensor(TYPE_PROXIMITY, false) returns a non wake-up proximity sensor if it exists.
- getDefaultSensor(TYPE_PROXIMITY, true) returns a wake-up proximity sensor which is the same as the Sensor returned by getDefaultSensor(int).

> Note: Sensors like TYPE_PROXIMITY and TYPE_SIGNIFICANT_MOTION are declared as wake-up sensors by default.

Parameters	
`type`	`int`: type of sensor requested
`wakeUp`	`boolean`: flag to indicate whether the Sensor is a wake-up or non wake-up sensor.

Returns	
Sensor	the default sensor matching the requested type and wakeUp properties if one exists and the application has the necessary permissions, or null otherwise.

See also:

isWakeUpSensor()

getDynamicSensorList

Added in API level 24

List<Sensor> `getDynamicSensorList (int type)`

Use this method to get a list of available dynamic sensors of a certain type. Make multiple calls to get sensors of different types or use Sensor.TYPE_ALL to get all dynamic sensors.

> NOTE: Both wake-up and non wake-up sensors matching the given type are returned. Check isWakeUpSensor() to know the wake-up properties of the returned Sensor.

Parameters

`type`	`int`: of sensors requested

Returns	
List<Sensor>	a list of dynamic sensors matching the requested type.

See also:

Sensor

getInclination

Added in API level 3

```
float getInclination (float[] I)
```

Computes the geomagnetic inclination angle in radians from the inclination matrix **I** returned by getRotationMatrix(float[], float[], float[], float[]).

Parameters	
`I`	`float`: inclination matrix see getRotationMatrix(float[], float[], float[], float[]).

Returns	
`float`	The geomagnetic inclination angle in radians.

See also:

getRotationMatrix(float[], float[], float[], float[])

getOrientation(float[], float[])

GeomagneticField

getOrientation

Added in API level 3

```
float[] getOrientation (float[] R,
                float[] values)
```

Computes the device's orientation based on the rotation matrix.

When it returns, the array values are as follows:

- values[0]: *Azimuth*, angle of rotation about the -z axis. This value represents the angle between the device's y axis and the magnetic north pole. When facing north, this angle is 0, when facing south, this angle is π. Likewise, when facing east, this angle is π/2, and when facing west, this angle is -π/2. The range of values is -π to π.
- values[1]: *Pitch*, angle of rotation about the x axis. This value represents the angle between a plane parallel to the device's screen and a plane parallel to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the top edge of the device toward the ground creates a positive pitch angle. The range of values is -π to π.
- values[2]: *Roll*, angle of rotation about the y axis. This value represents the angle between a plane perpendicular to the device's screen and a plane perpendicular to the ground. Assuming that the bottom edge of the device faces the user and that the screen is face-up, tilting the left edge of the device toward the ground creates a positive roll angle. The range of values is -π/2 to π/2.

Applying these three rotations in the azimuth, pitch, roll order transforms an identity matrix to the rotation matrix passed into this method. Also, note that all three orientation angles are expressed in **radians**.

Parameters	
`R`	`float`: rotation matrix see getRotationMatrix(float[], float[], float[], float[]).
`values`	`float`: an array of 3 floats to hold the result.

Returns	
`float[]`	The array values passed as argument.

See also:

getRotationMatrix(float[], float[], float[], float[])

GeomagneticField

getQuaternionFromVector

Added in API level 9

```
void getQuaternionFromVector (float[] Q,
                float[] rv)
```

Helper function to convert a rotation vector to a normalized quaternion. Given a rotation vector (presumably from a ROTATION_VECTOR sensor), returns a normalized quaternion in the array Q. The quaternion is stored as [w, x, y, z]

Parameters	
`Q`	`float`: an array of floats in which to store the computed quaternion
`rv`	`float`: the rotation vector to convert

getRotationMatrix

Added in API level 3

```
boolean getRotationMatrix (float[] R,
                float[] I,
                float[] gravity,
                float[] geomagnetic)
```

Computes the inclination matrix **I** as well as the rotation matrix **R** transforming a vector from the device coordinate system to the world's coordinate system which is defined as a direct orthonormal basis, where:

- X is defined as the vector product **Y.Z** (It is tangential to the ground at the device's current location and roughly points East).
- Y is tangential to the ground at the device's current location and points towards the magnetic North Pole.
- Z points towards the sky and is perpendicular to the ground.


World coordinate-system diagram.

By definition:

[0 0 g] = **R** * **gravity** (g = magnitude of gravity)

[0 m 0] = **I** * **R** * **geomagnetic** (m = magnitude of geomagnetic field)

R is the identity matrix when the device is aligned with the world's coordinate system, that is, when the device's X axis points toward East, the Y axis points to the North Pole and the device is facing the sky.

I is a rotation matrix transforming the geomagnetic vector into the same coordinate space as gravity (the world's coordinate space). **I** is a simple rotation around the X axis. The inclination angle in radians can be computed with getInclination(float[]).

Each matrix is returned either as a 3x3 or 4x4 row-major matrix depending on the length of the passed array:

If the array length is 16:

```
/  M[ 0]   M[ 1]   M[ 2]   M[ 3]  \
|  M[ 4]   M[ 5]   M[ 6]   M[ 7]  |
|  M[ 8]   M[ 9]   M[10]   M[11]  |
\  M[12]   M[13]   M[14]   M[15]  /
```

This matrix is ready to be used by OpenGL ES's glLoadMatrixf(float[], int).
Note that because OpenGL matrices are column-major matrices you must transpose the matrix before using it. However, since the matrix is a rotation matrix, its transpose is also its inverse, conveniently, it is often the inverse of the rotation that is needed for rendering; it can therefore be used with OpenGL ES directly.

Also note that the returned matrices always have this form:

```
/  M[ 0]   M[ 1]   M[ 2]   0  \
|  M[ 4]   M[ 5]   M[ 6]   0  |
|  M[ 8]   M[ 9]   M[10]   0  |
\      0       0       0   1  /
```

If the array length is 9:

```
/  M[ 0]   M[ 1]   M[ 2]  \
|  M[ 3]   M[ 4]   M[ 5]  |
\  M[ 6]   M[ 7]   M[ 8]  /
```

The inverse of each matrix can be computed easily by taking its transpose.

The matrices returned by this function are meaningful only when the device is not free-falling and it is not close to the magnetic north. If the device is accelerating, or placed into a strong magnetic field, the returned matrices may be inaccurate.

Parameters	
`R`	`float`: is an array of 9 floats holding the rotation matrix **R** when this function returns. R can be null.
`I`	`float`: is an array of 9 floats holding the rotation matrix **I** when this function returns. I can be null.
`gravity`	`float`: is an array of 3 floats containing the gravity vector expressed in the device's coordinate. You can simply use the values returned by a SensorEvent of a Sensor of type TYPE_ACCELEROMETER.
`geomagnetic`	`float`: is an array of 3 floats containing the geomagnetic vector expressed in the device's coordinate. You can simply use the values returned by a SensorEvent of a Sensor of type TYPE_MAGNETIC_FIELD.

Returns	
`boolean`	`true` on success, `false` on failure (for instance, if the device is in free fall). Free fall is defined as condition when the magnitude of the gravity is less than 1/10 of the nominal value. On failure the output matrices are not modified.

See also:

getInclination(float[])

getOrientation(float[], float[])

remapCoordinateSystem(float[], int, int, float[])

getRotationMatrixFromVector

Added in API level 9

```
void getRotationMatrixFromVector (float[] R,
                float[] rotationVector)
```

Helper function to convert a rotation vector to a rotation matrix. Given a rotation vector (presumably from a ROTATION_VECTOR sensor), returns a 9 or 16 element rotation matrix in the array R. R must have length 9 or 16. If R.length == 9, the following matrix is returned:

```
/  R[ 0]   R[ 1]   R[ 2]   \
|  R[ 3]   R[ 4]   R[ 5]   |
\  R[ 6]   R[ 7]   R[ 8]   /
```

If R.length == 16, the following matrix is returned:

```
/  R[ 0]   R[ 1]   R[ 2]   0  \
|  R[ 4]   R[ 5]   R[ 6]   0  |
|  R[ 8]   R[ 9]   R[10]   0  |
\  0       0       0       1  /
```

Parameters	
`R`	`float`: an array of floats in which to store the rotation matrix
`rotationVector`	`float`: the rotation vector to convert

getSensorList

Added in API level 3

List<Sensor> `getSensorList (int type)`

Use this method to get the list of available sensors of a certain type. Make multiple calls to get sensors of different types or use Sensor.TYPE_ALL to get all the sensors.

> NOTE: Both wake-up and non wake-up sensors matching the given type are returned. Check isWakeUpSensor() to know the wake-up properties of the returned Sensor.

Parameters	
`type`	`int`: of sensors requested

Returns	
List<Sensor>	a list of sensors matching the asked type.

See also:

getDefaultSensor(int)

Sensor

getSensors

Added in API level 1

```
int getSensors ()
```

> **This method was deprecated in API level 3.**
> This method is deprecated, use getSensorList(int) instead

Returns	
`int`	available sensors.

isDynamicSensorDiscoverySupported

Added in API level 24

```
boolean isDynamicSensorDiscoverySupported ()
```

Tell if dynamic sensor discovery feature is supported by system.

Returns	
`boolean`	`true` if dynamic sensor discovery is supported, `false` otherwise.

registerDynamicSensorCallback

Added in API level 24

```
void registerDynamicSensorCallback (SensorManager.DynamicSensorCallback callback)
```

Add a DynamicSensorCallback to receive dynamic sensor connection callbacks. Repeat registration with the already registered callback object will have no additional effect.

Parameters	
`callback`	`SensorManager.DynamicSensorCallback`: An object that implements the DynamicSensorCallback interface for receiving callbacks.

Throws	
IllegalArgumentException	when callback is null.

See also:

ERROR(/#addDynamicSensorCallback(DynamicSensorCallback, Handler))

registerDynamicSensorCallback

Added in API level 24

```
void registerDynamicSensorCallback (SensorManager.DynamicSensorCallback callback,
                Handler handler)
```

Add a DynamicSensorCallback to receive dynamic sensor connection callbacks. Repeat registration with the already registered callback object will have no additional effect.

Parameters	
`callback`	`SensorManager.DynamicSensorCallback`: An object that implements the DynamicSensorCallback interface for receiving callbacks.
`handler`	`Handler`: The Handler the sensor connection events will be delivered to.

Throws	
IllegalArgumentException	when callback is null.

registerListener

Added in API level 3

```
boolean registerListener (SensorEventListener listener,
                Sensor sensor,
                int samplingPeriodUs)
```

Registers a SensorEventListener for the given sensor at the given sampling frequency.

The events will be delivered to the provided `SensorEventListener` as soon as they are available. To reduce the power consumption, applications can use registerListener(SensorEventListener, Sensor, int, int) instead and specify a positive non-zero maximum reporting latency.

In the case of non-wake-up sensors, the events are only delivered while the Application Processor (AP) is not in suspend mode. See isWakeUpSensor() for more details. To ensure delivery of events from non-wake-up sensors even when the screen is OFF, the application registering to the sensor must hold a partial wake-lock to keep the AP awake, otherwise some events might be lost while the AP is asleep. Note that although events might be lost while the AP is asleep, the sensor will still consume power if it is not explicitly deactivated by the application. Applications must unregister their `SensorEventListener`s in their activity's `onPause()` method to avoid consuming power while the device is inactive. See registerListener(SensorEventListener, Sensor, int, int) for more details on hardware FIFO (queueing) capabilities and when some sensor events might be lost.

In the case of wake-up sensors, each event generated by the sensor will cause the AP to wake-up, ensuring that each event can be delivered. Because of this, registering to a wake-up sensor has very significant power implications. Call isWakeUpSensor() to check whether a sensor is a wake-up sensor. See registerListener(SensorEventListener, Sensor, int, int) for information on how to reduce the power impact of registering to wake-up sensors.

> Note: Don't use this method with one-shot trigger sensors such as TYPE_SIGNIFICANT_MOTION. Use requestTriggerSensor(TriggerEventListener, Sensor) instead. Use getReportingMode() to obtain the reporting mode of a given sensor.

Parameters	
`listener`	`SensorEventListener`: A SensorEventListener object.
`sensor`	`Sensor`: The Sensor to register to.
`samplingPeriodUs`	`int`: The rate sensor events are delivered at. This is only a hint to the system. Events may be received faster or slower than the specified rate. Usually events are received faster. The value must be one of SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, or SENSOR_DELAY_FASTEST or, the desired delay between events in microseconds. Specifying the delay in microseconds only works from Android 2.3 (API level 9) onwards. For earlier releases, you must use one of the `SENSOR_DELAY_*` constants.

Returns	
`boolean`	`true` if the sensor is supported and successfully enabled.

See also:

registerListener(SensorEventListener, Sensor, int, Handler)

unregisterListener(SensorEventListener)

unregisterListener(SensorEventListener, Sensor)

registerListener

Added in API level 19

```
boolean registerListener (SensorEventListener listener,
                Sensor sensor,
                int samplingPeriodUs,
                int maxReportLatencyUs)
```

Registers a SensorEventListener for the given sensor at the given sampling frequency and the given maximum reporting latency.

This function is similar to registerListener(SensorEventListener, Sensor, int) but it allows events to stay temporarily in the hardware FIFO (queue) before being delivered. The events can be stored in the hardware FIFO up to `maxReportLatencyUs` microseconds. Once one of the events in the FIFO needs to be reported, all of the events in the FIFO are reported sequentially. This means that some events will be reported before the maximum reporting latency has elapsed.

When `maxReportLatencyUs` is 0, the call is equivalent to a call to registerListener(SensorEventListener, Sensor, int), as it requires the events to be delivered as soon as possible.

When `sensor.maxFifoEventCount()` is 0, the sensor does not use a FIFO, so the call will also be equivalent to registerListener(SensorEventListener, Sensor, int).

Setting `maxReportLatencyUs` to a positive value allows to reduce the number of interrupts the AP (Application Processor) receives, hence reducing power consumption, as the AP can switch to a lower power state while the sensor is capturing the data. This is especially important when registering to wake-up sensors, for which each interrupt causes the AP to wake up if it was in suspend mode. See isWakeUpSensor() for more information on wake-up sensors.

Note: Don't use this method with one-shot trigger sensors such as TYPE_SIGNIFICANT_MOTION. Use requestTriggerSensor(TriggerEventListener, Sensor) instead.

Parameters	
`listener`	`SensorEventListener`: A SensorEventListener object that will receive the sensor events. If the application is interested in receiving flush complete notifications, it should register with SensorEventListener2 instead.
`sensor`	`Sensor`: The Sensor to register to.
`samplingPeriodUs`	`int`: The desired delay between two consecutive events in microseconds. This is only a hint to the system. Events may be received faster or slower than the specified rate. Usually events are received faster. Can be one of SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, SENSOR_DELAY_FASTEST or the delay in microseconds.
`maxReportLatencyUs`	`int`: Maximum time in microseconds that events can be delayed before being reported to the application. A large value allows reducing the power consumption associated with the sensor. If maxReportLatencyUs is set to zero, events are delivered as soon as they are available, which is equivalent to calling registerListener(SensorEventListener, Sensor, int).

Returns	
`boolean`	`true` if the sensor is supported and successfully enabled.

See also:

registerListener(SensorEventListener, Sensor, int)

unregisterListener(SensorEventListener)

flush(SensorEventListener)

registerListener

Added in API level 3

```
boolean registerListener (SensorEventListener listener,
                Sensor sensor,
```

```
int samplingPeriodUs,
Handler handler)
```

Registers a SensorEventListener for the given sensor. Events are delivered in continuous mode as soon as they are available. To reduce the power consumption, applications can use registerListener(SensorEventListener, Sensor, int, int) instead and specify a positive non-zero maximum reporting latency.

Note: Don't use this method with a one shot trigger sensor such as TYPE_SIGNIFICANT_MOTION. Use requestTriggerSensor(TriggerEventListener, Sensor) instead.

Parameters	
`listener`	`SensorEventListener`: A SensorEventListener object.
`sensor`	`Sensor`: The Sensor to register to.
`samplingPeriodUs`	`int`: The rate sensor events are delivered at. This is only a hint to the system. Events may be received faster or slower than the specified rate. Usually events are received faster. The value must be one of SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, or SENSOR_DELAY_FASTEST or, the desired delay between events in microseconds. Specifying the delay in microseconds only works from Android 2.3 (API level 9) onwards. For earlier releases, you must use one of the `SENSOR_DELAY_*` constants.
`handler`	`Handler`: The Handler the sensor events will be delivered to.

Returns	
`boolean`	`true` if the sensor is supported and successfully enabled.

See also:

registerListener(SensorEventListener, Sensor, int)

unregisterListener(SensorEventListener)

unregisterListener(SensorEventListener, Sensor)

registerListener

Added in API level 1

```
boolean registerListener (SensorListener listener,
                int sensors)
```

> **This method was deprecated in API level 3.**
>
> This method is deprecated, use registerListener(SensorEventListener, Sensor, int) instead.

Registers a listener for given sensors.

Parameters	
`listener`	`SensorListener`: sensor listener object
`sensors`	`int`: a bit masks of the sensors to register to

Returns	
`boolean`	`true` if the sensor is supported and successfully enabled

registerListener

Added in API level 1

```
boolean registerListener (SensorListener listener,
                int sensors,
                int rate)
```

> **This method was deprecated in API level 3.**
>
> This method is deprecated, use registerListener(SensorEventListener, Sensor, int) instead.

Registers a SensorListener for given sensors.

Parameters	
`listener`	`SensorListener`: sensor listener object
`sensors`	`int`: a bit masks of the sensors to register to

`rate`	`int`: rate of events. This is only a hint to the system. events may be received faster or slower than the specified rate. Usually events are received faster. The value must be one of SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, or SENSOR_DELAY_FASTEST.

Returns	
`boolean`	`true` if the sensor is supported and successfully enabled

registerListener

Added in API level 19

```
boolean registerListener (SensorEventListener listener,
                Sensor sensor,
                int samplingPeriodUs,
                int maxReportLatencyUs,
                Handler handler)
```

Registers a SensorEventListener for the given sensor at the given sampling frequency and the given maximum reporting latency.

Parameters	
`listener`	`SensorEventListener`: A SensorEventListener object that will receive the sensor events. If the application is interested in receiving flush complete notifications, it should register with SensorEventListener2 instead.
`sensor`	`Sensor`: The Sensor to register to.
`samplingPeriodUs`	`int`: The desired delay between two consecutive events in microseconds. This is only a hint to the system. Events may be received faster or slower than the specified rate. Usually events are received faster. Can be one of SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, SENSOR_DELAY_FASTEST or the delay in microseconds.
`maxReportLatencyUs`	`int`: Maximum time in microseconds that events can be delayed before being reported to the application. A large value allows reducing the power consumption associated with the sensor. If maxReportLatencyUs is set to zero, events are delivered as soon as they are available, which is equivalent to calling registerListener(SensorEventListener, Sensor, int).
`handler`	`Handler`: The Handler the sensor events will be delivered to.

Returns	
`boolean`	`true` if the sensor is supported and successfully enabled.

See also:

registerListener(SensorEventListener, Sensor, int, int)

remapCoordinateSystem

Added in API level 3

```
boolean remapCoordinateSystem (float[] inR,
                int X,
                int Y,
                float[] outR)
```

Rotates the supplied rotation matrix so it is expressed in a different coordinate system. This is typically used when an application needs to compute the three orientation angles of the device (see getOrientation(float[], float[])) in a different coordinate system.

When the rotation matrix is used for drawing (for instance with OpenGL ES), it usually **doesn't need** to be transformed by this function, unless the screen is physically rotated, in which case you can use Display.getRotation() to retrieve the current rotation of the screen. Note that because the user is generally free to rotate their screen, you often should consider the rotation in deciding the parameters to use here.

Examples:

- Using the camera (Y axis along the camera's axis) for an augmented reality application where the rotation angles are needed:

  ```
  remapCoordinateSystem(inR, AXIS_X, AXIS_Z, outR);
  ```

- Using the device as a mechanical compass when rotation is Surface.ROTATION_90:

  ```
  remapCoordinateSystem(inR, AXIS_Y, AXIS_MINUS_X, outR);
  ```

  Beware of the above example. This call is needed only to account for a rotation from its natural orientation when calculating the rotation angles (see getOrientation(float[], float[])). If the rotation matrix is also used for rendering, it may not need to be transformed, for instance if your Activity is running in landscape mode.

Since the resulting coordinate system is orthonormal, only two axes need to be specified.

Parameters	
`inR`	`float`: the rotation matrix to be transformed. Usually it is the matrix returned by getRotationMatrix(float[], float[], float[], float[]).
`X`	`int`: defines the axis of the new cooridinate system that coincide with the X axis of the original coordinate system.
`Y`	`int`: defines the axis of the new cooridinate system that coincide with the Y axis of the original coordinate system.
`outR`	`float`: the transformed rotation matrix. inR and outR should not be the same array.

Returns	
`boolean`	`true` on success. `false` if the input parameters are incorrect, for instance if X and Y define the same axis. Or if inR and outR don't have the same length.

See also:

getRotationMatrix(float[], float[], float[], float[])

requestTriggerSensor

Added in API level 18

```
boolean requestTriggerSensor (TriggerEventListener listener,
                Sensor sensor)
```

Requests receiving trigger events for a trigger sensor.

When the sensor detects a trigger event condition, such as significant motion in the case of the TYPE_SIGNIFICANT_MOTION, the provided trigger listener will be invoked once and then its request to receive trigger events will be canceled. To continue receiving trigger events, the application must request to receive trigger events again.

Parameters	
`listener`	`TriggerEventListener`: The listener on which the onTrigger(TriggerEvent) will be delivered.
`sensor`	`Sensor`: The sensor to be enabled.

Returns

`boolean`	true if the sensor was successfully enabled.

Throws	
IllegalArgumentException	when sensor is null or not a trigger sensor.

unregisterDynamicSensorCallback

Added in API level 24

```
void unregisterDynamicSensorCallback (SensorManager.DynamicSensorCallback callback)
```

Remove a DynamicSensorCallback to stop sending dynamic sensor connection events to that callback.

Parameters	
`callback`	`SensorManager.DynamicSensorCallback`: An object that implements the DynamicSensorCallback interface for receiving callbacks.

unregisterListener

Added in API level 3

```
void unregisterListener (SensorEventListener listener)
```

Unregisters a listener for all sensors.

Parameters	
`listener`	`SensorEventListener`: a SensorListener object

See also:

unregisterListener(SensorEventListener, Sensor)

registerListener(SensorEventListener, Sensor, int)

unregisterListener

Added in API level 3

```
void unregisterListener (SensorEventListener listener,
                Sensor sensor)
```

Unregisters a listener for the sensors with which it is registered.

Note: Don't use this method with a one shot trigger sensor such as TYPE_SIGNIFICANT_MOTION. Use cancelTriggerSensor(TriggerEventListener, Sensor) instead.

Parameters	
`listener`	`SensorEventListener`: a SensorEventListener object
`sensor`	`Sensor`: the sensor to unregister from

See also:

unregisterListener(SensorEventListener)

registerListener(SensorEventListener, Sensor, int)

unregisterListener

Added in API level 1

```
void unregisterListener (SensorListener listener)
```

> **This method was deprecated in API level 3.**
> This method is deprecated, use unregisterListener(SensorEventListener) instead.

Unregisters a listener for all sensors.

Parameters	
`listener`	`SensorListener`: a SensorListener object

unregisterListener

Added in API level 1

```
void unregisterListener (SensorListener listener,
                int sensors)
```

> **This method was deprecated in API level 3.**
>
> This method is deprecated, use unregisterListener(SensorEventListener, Sensor) instead.

Unregisters a listener for the sensors with which it is registered.

Parameters	
`listener`	`SensorListener`: a SensorListener object
`sensors`	`int`: a bit masks of the sensors to unregister from