public class

SensorManager

extends Object

java.lang.Object

↳android.hardware.SensorManager

Class Overview

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.

```
 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

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 is deprecated. use `Sensor` instead.*
int	DATA_Y	*This constant is deprecated. use `Sensor` instead.*
int	DATA_Z	*This constant is deprecated. 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 is deprecated. use `Sensor` instead.*

int	RAW_DATA_X	*This constant is deprecated. use* `Sensor` *instead.*
int	RAW_DATA_Y	*This constant is deprecated. use* `Sensor` *instead.*
int	RAW_DATA_Z	*This constant is deprecated. use* `Sensor` *instead.*
int	SENSOR_ACCELEROMETER	*This constant is deprecated. use* `Sensor` *instead.*
int	SENSOR_ALL	*This constant is deprecated. 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 is deprecated. use* `Sensor` *instead.*
int	SENSOR_MAGNETIC_FIELD	*This constant is deprecated. use* `Sensor` *instead.*
int	SENSOR_MAX	*This constant is deprecated. use* `Sensor` *instead.*
int	SENSOR_MIN	*This constant is deprecated. use* `Sensor` *instead.*
int	SENSOR_ORIENTATION	*This constant is deprecated. use* `Sensor` *instead.*
int	SENSOR_ORIENTATION_RAW	*This constant is deprecated. use* `Sensor` *instead.*
int	SENSOR_PROXIMITY	*This constant is deprecated. 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 needed

int	SENSOR_STATUS_ACCURACY_MEDIUM	This sensor is reporting data with an average level of accuracy, calib environment may improve the readings
int	SENSOR_STATUS_UNRELIABLE	The values returned by this sensor cannot be trusted, calibration is environment doesn't allow readings
int	SENSOR_TEMPERATURE	*This constant is deprecated. use `Sensor` instead.*
int	SENSOR_TRICORDER	*This constant is deprecated. use `Sensor` instead.*
float	STANDARD_GRAVITY	Standard gravity (g) on Earth.

Public Methods	
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.
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	getRotationMatrix(float[] R, float[] I, float[] gravity, float[] geomagnetic)

boolean	Computes the inclination matrix **I** as well as the rotation matrix **R** transforming a vector from the device 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 loca 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 is deprecated. This method is deprecated, use* `getSensorList(int)` *instead*
boolean	registerListener(SensorListener listener, int sensors, int rate) *This method is deprecated. This method is deprecated, use* `registerListener(SensorEventListe` `int)` *instead.*
boolean	registerListener(SensorListener listener, int sensors) *This method is deprecated. This method is deprecated, use* `registerListener(SensorEventListe` `int)` *instead.*
boolean	registerListener(SensorEventListener listener, Sensor sensor, int rate, Handler handler) Registers a `SensorEventListener` for the given sensor.
boolean	registerListener(SensorEventListener listener, Sensor sensor, int rate) Registers a `SensorEventListener` for the given sensor.
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.
void	unregisterListener(SensorListener listener)

	This method is deprecated. This method is deprecated, use `unregisterListener(SensorEventLi`
void	unregisterListener(SensorListener listener, int sensors) *This method is deprecated. This method is deprecated, use* `unregisterListener(SensorEventLi` `Sensor)` *instead.*
void	unregisterListener(SensorEventListener listener, Sensor sensor) Unregisters a listener for the sensors with which it is registered.
void	unregisterListener(SensorEventListener listener) Unregisters a listener for all sensors.

[Expand]

Inherited Methods

▶ From class java.lang.Object

Constants

public static final int ***AXIS_MINUS_X***

Since: API Level 3

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

Constant Value: 129 (0x00000081)

public static final int ***AXIS_MINUS_Y***

Since: API Level 3

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

Constant Value: 130 (0x00000082)

public static final int ***AXIS_MINUS_Z***

Since: API Level 3

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

Constant Value: 131 (0x00000083)

public static final int ***AXIS_X***

Since: API Level 3

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

Constant Value: 1 (0x00000001)

public static final int ***AXIS_Y***

Since: API Level 3

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

Constant Value: 2 (0x00000002)

public static final int ***AXIS_Z***

Since: API Level 3

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

Constant Value: 3 (0x00000003)

public static final int ***DATA_X***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 0 (0x00000000)

public static final int ***DATA_Y***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 1 (0x00000001)

public static final int ***DATA_Z***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 2 (0x00000002)

public static final float **GRAVITY_DEATH_STAR_I**

Since: API Level 1

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

Constant Value: 3.5303614E-7

public static final float **GRAVITY_EARTH**

Since: API Level 1

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

Constant Value: 9.80665

public static final float **GRAVITY_JUPITER**

Since: API Level 1

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

Constant Value: 23.12

public static final float **GRAVITY_MARS**

Since: API Level 1

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

Constant Value: 3.71

public static final float **GRAVITY_MERCURY**

Since: API Level 1

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

Constant Value: 3.7

public static final float **GRAVITY_MOON**

Since: API Level 1

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

Constant Value: 1.6

public static final float **GRAVITY_NEPTUNE**

Since: API Level 1

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

Constant Value: 11.0

public static final float **GRAVITY_PLUTO**

Since: API Level 1

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

Constant Value: 0.6

public static final float ***GRAVITY_SATURN***

Since: API Level 1

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

Constant Value: 8.96

public static final float ***GRAVITY_SUN***

Since: API Level 1

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

Constant Value: 275.0

public static final float ***GRAVITY_THE_ISLAND***

Since: API Level 1

Gravity on the island

Constant Value: 4.815162

public static final float ***GRAVITY_URANUS***

Since: API Level 1

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

Constant Value: 8.69

public static final float ***GRAVITY_VENUS***

Since: API Level 1

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

Constant Value: 8.87

public static final float ***LIGHT_CLOUDY***

Since: API Level 1

luminance under a cloudy sky in lux

Constant Value: 100.0

public static final float ***LIGHT_FULLMOON***

Since: API Level 1

luminance at night with full moon in lux

Constant Value: 0.25

public static final float **LIGHT_NO_MOON**

Since: API Level 1

luminance at night with no moon in lux

Constant Value: 0.0010

public static final float **LIGHT_OVERCAST**

Since: API Level 1

luminance under an overcast sky in lux

Constant Value: 10000.0

public static final float **LIGHT_SHADE**

Since: API Level 1

luminance in shade in lux

Constant Value: 20000.0

public static final float **LIGHT_SUNLIGHT**

Since: API Level 1

luminance of sunlight in lux

Constant Value: 110000.0

public static final float **LIGHT_SUNLIGHT_MAX**

Since: API Level 1

Maximum luminance of sunlight in lux

Constant Value: 120000.0

public static final float **LIGHT_SUNRISE**

Since: API Level 1

luminance at sunrise in lux

Constant Value: 400.0

public static final float **MAGNETIC_FIELD_EARTH_MAX**

Since: API Level 1

Maximum magnetic field on Earth's surface

Constant Value: 60.0

public static final float **MAGNETIC_FIELD_EARTH_MIN**

Since: API Level 1

Minimum magnetic field on Earth's surface

Constant Value: 30.0

public static final float ***PRESSURE_STANDARD_ATMOSPHERE***

Since: API Level 9

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

Constant Value: 1013.25

public static final int ***RAW_DATA_INDEX***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 3 (0x00000003)

public static final int ***RAW_DATA_X***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 3 (0x00000003)

public static final int ***RAW_DATA_Y***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 4 (0x00000004)

public static final int ***RAW_DATA_Z***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 5 (0x00000005)

public static final int ***SENSOR_ACCELEROMETER***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

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

Constant Value: 2 (0x00000002)

public static final int ***SENSOR_ALL***

Since: API Level 1

> **This constant is deprecated.**
>
> use `Sensor` instead.

A constant that includes all sensors

Constant Value: 127 (0x0000007f)

public static final int ***SENSOR_DELAY_FASTEST***

Since: API Level 1

get sensor data as fast as possible

Constant Value: 0 (0x00000000)

public static final int ***SENSOR_DELAY_GAME***

Since: API Level 1

rate suitable for games

Constant Value: 1 (0x00000001)

public static final int ***SENSOR_DELAY_NORMAL***

Since: API Level 1

rate (default) suitable for screen orientation changes

Constant Value: 3 (0x00000003)

public static final int ***SENSOR_DELAY_UI***

Since: API Level 1

rate suitable for the user interface

Constant Value: 2 (0x00000002)

public static final int ***SENSOR_LIGHT***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

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

Constant Value: 16 (0x00000010)

public static final int ***SENSOR_MAGNETIC_FIELD***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

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

Constant Value: 8 (0x00000008)

public static final int ***SENSOR_MAX***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

Largest sensor ID

Constant Value: 64 (0x00000040)

public static final int ***SENSOR_MIN***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

Smallest sensor ID

Constant Value: 1 (0x00000001)

public static final int ***SENSOR_ORIENTATION***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

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

Constant Value: 1 (0x00000001)

public static final int ***SENSOR_ORIENTATION_RAW***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

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

Constant Value: 128 (0x00000080)

public static final int ***SENSOR_PROXIMITY***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

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

Constant Value: 32 (0x00000020)

public static final int ***SENSOR_STATUS_ACCURACY_HIGH***

Since: API Level 1

This sensor is reporting data with maximum accuracy

Constant Value: 3 (0x00000003)

public static final int ***SENSOR_STATUS_ACCURACY_LOW***

Since: API Level 1

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

Constant Value: 1 (0x00000001)

public static final int ***SENSOR_STATUS_ACCURACY_MEDIUM***

Since: API Level 1

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

Constant Value: 2 (0x00000002)

public static final int ***SENSOR_STATUS_UNRELIABLE***

Since: API Level 1

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

Constant Value: 0 (0x00000000)

public static final int ***SENSOR_TEMPERATURE***

Since: API Level 1

> **This constant is deprecated.**
>
> use Sensor instead.

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

Constant Value: 4 (0x00000004)

public static final int ***SENSOR_TRICORDER***

Since: API Level 1

> **This constant is deprecated.**
> use Sensor instead.

A constant describing a Tricorder See SensorListener for more details.

Constant Value: 64 (0x00000040)

public static final float ***STANDARD_GRAVITY***

Since: API Level 1

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

Constant Value: 9.80665

Public Methods

public static float ***getAltitude*** *(float p0, float p)*

Since: API Level 9

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 usePRESSURE_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	pressure at sea level
p	atmospheric pressure

Returns

- Altitude in meters

*public static void **getAngleChange** (float[] angleChange, float[] R, float[] prevR)*

Since: API Level 9

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 rotation around the x,y, and z axes which transforms prevR to R. outputs a 3 element vector containing the x,y, and z 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]  /
```

Parameters

angleChange	an array of floats in which the angle change is stored
R	current rotation matrix
prevR	previous rotation matrix

*public Sensor **getDefaultSensor** (int type)*

Since: API Level 3

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	of sensors requested

Returns

- the default sensors matching the asked type.

See Also

- `getSensorList(int)`
- `Sensor`

public static float ***getInclination*** *(float[] I)*

Since: API Level 3

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

Parameters

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

Returns

- The geomagnetic inclination angle in radians.

See Also

- `getRotationMatrix(float[], float[], float[], float[])`
- `getOrientation(float[], float[])`
- `GeomagneticField`

public static float[] ***getOrientation*** *(float[] R, float[] values)*

Since: API Level 3

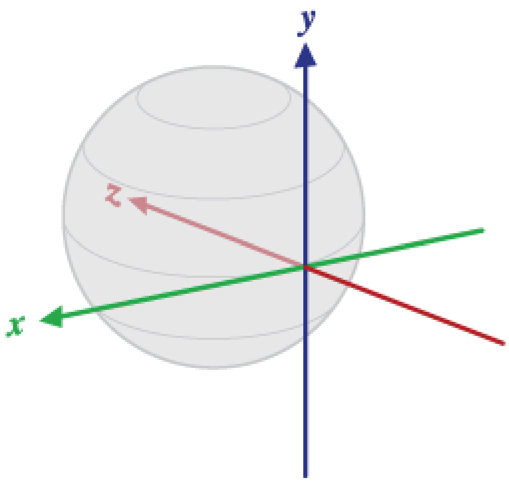
Computes the device's orientation based on the rotation matrix.

When it returns, the array values is filled with the result:

- values[0]: *azimuth*, rotation around the Z axis.
- values[1]: *pitch*, rotation around the X axis.
- values[2]: *roll*, rotation around the Y axis.

The reference coordinate-system used is different from the world coordinate-system defined for the rotation matrix:

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



All three angles above are in **radians** and **positive** in the **counter-clockwise** direction.

Parameters

R rotation matrix see getRotationMatrix(float[], float[], float[], float[]).

values an array of 3 floats to hold the result.

Returns

- The array values passed as argument.

See Also

- getRotationMatrix(float[], float[], float[], float[])
- GeomagneticField

*public static void **getQuaternionFromVector** (float[] Q, float[] rv)*

Since: API Level 9

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 an array of floats in which to store the computed quaternion

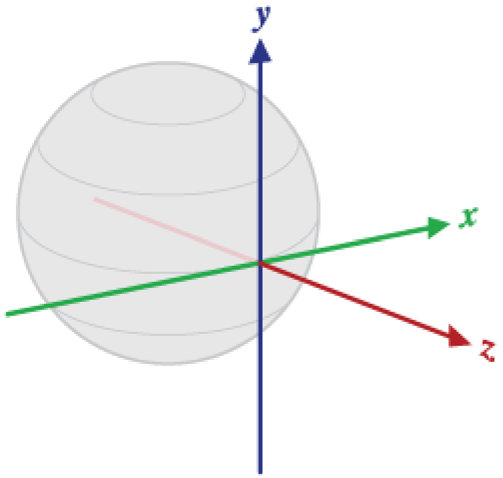
rv the rotation vector to convert

*public static boolean **getRotationMatrix** (float[] R, float[] I, float[] gravity, float[] geomagnetic)*

Since: API Level 3

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.



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 is an array of 9 floats holding the rotation matrix **R** when this function returns. R can be null.

I is an array of 9 floats holding the rotation matrix **I** when this function returns. I can be null.

gravity 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 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

- `true` on success, `false` on failure (for instance, if the device is in free fall). On failure the output matrices are not modified.

See Also

- `getInclination(float[])`
- `getOrientation(float[], float[])`
- `remapCoordinateSystem(float[], int, int, float[])`

*public static void **getRotationMatrixFromVector** (float[] R, float[] rotationVector)*

Since: API Level 9

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	an array of floats in which to store the rotation matrix
rotationVector	the rotation vector to convert

*public List<Sensor> **getSensorList** (int type)*

Since: API Level 3

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.

Parameters

type	of sensors requested

Returns

- a list of sensors matching the asked type.

See Also

- `getDefaultSensor(int)`

- Sensor

public int ***getSensors*** *()*

Since: API Level 1

> **This method is deprecated.**
>
> This method is deprecated, use `getSensorList(int)` instead

Returns

- available sensors.

public boolean ***registerListener*** *(SensorListener listener, int sensors, int rate)*

Since: API Level 1

> **This method is deprecated.**
>
> This method is deprecated, use `registerListener(SensorEventListener, Sensor, int)` instead.

Registers a SensorListener for given sensors.

Parameters

listener	sensor listener object
sensors	a bit masks of the sensors to register to
rate	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

- `true` if the sensor is supported and successfully enabled

public boolean ***registerListener*** *(SensorListener listener, int sensors)*

Since: API Level 1

> **This method is deprecated.**
>
> This method is deprecated, use `registerListener(SensorEventListener, Sensor, int)` instead.

Registers a listener for given sensors.

Parameters

listener sensor listener object

sensors a bit masks of the sensors to register to

Returns

- `true` if the sensor is supported and successfully enabled

public boolean ***registerListener*** *(SensorEventListener listener, Sensor sensor, int rate, Handler handler)*

Since: API Level 3

Registers a `SensorEventListener` for the given sensor.

Parameters

listener A `SensorEventListener` object.

sensor The `Sensor` to register to.

rate 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 microsecond.

handler The `Handler` the `sensor events` will be delivered to.

Returns

- true if the sensor is supported and successfully enabled.

See Also

- `registerListener(SensorEventListener, Sensor, int)`
- `unregisterListener(SensorEventListener)`
- `unregisterListener(SensorEventListener, Sensor)`

public boolean ***registerListener*** *(SensorEventListener listener, Sensor sensor, int rate)*

Since: API Level 3

Registers a `SensorEventListener` for the given sensor.

Parameters

listener A `SensorEventListener` object.

sensor The `Sensor` to register to.

rate 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 microsecond.

Returns

- `true` if the sensor is supported and successfully enabled.

See Also

- `registerListener(SensorEventListener, Sensor, int, Handler)`
- `unregisterListener(SensorEventListener)`
- `unregisterListener(SensorEventListener, Sensor)`

*public static boolean **remapCoordinateSystem** (float[] inR, int X, int Y, float[] outR)*

Since: API Level 3

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	the rotation matrix to be transformed. Usually it is the matrix returned by `getRotationMatrix(float[], float[], float[], float[])`.
X	defines on which world axis and direction the X axis of the device is mapped.
Y	defines on which world axis and direction the Y axis of the device is mapped.
outR	the transformed rotation matrix. inR and outR can be the same array, but it is not recommended for performance reason.

Returns

- `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[])`

*public void **unregisterListener** (SensorListener listener)*

Since: API Level 1

> **This method is deprecated.**
>
> This method is deprecated, use `unregisterListener(SensorEventListener)` instead.

Unregisters a listener for all sensors.

Parameters

listener	a SensorListener object

*public void **unregisterListener** (SensorListener listener, int sensors)*

Since: API Level 1

> **This method is deprecated.**
>
> This method is deprecated, use `unregisterListener(SensorEventListener, Sensor)` instead.

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

Parameters

listener	a SensorListener object
sensors	a bit masks of the sensors to unregister from

public void ***unregisterListener*** *(SensorEventListener listener, Sensor sensor)*

Since: API Level 3

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

Parameters

listener	a SensorEventListener object
sensor	the sensor to unregister from

See Also

- `unregisterListener(SensorEventListener)`
- `registerListener(SensorEventListener, Sensor, int)`

public void ***unregisterListener*** *(SensorEventListener listener)*

Since: API Level 3

Unregisters a listener for all sensors.

Parameters

listener	a SensorListener object

See Also

- `unregisterListener(SensorEventListener, Sensor)`
- `registerListener(SensorEventListener, Sensor, int)`