Table of Contents

Sensors

Using the Location Sensor

GPS

Sensing Location with App Inventor

Checking Boundaries

Location Information Providers: GPS, Wi-Fi, and Cell ID

Querying and Setting the Current Provider

Using the Orientation Sensor

Moving a Shooter by Tilt

Using the Phone as a Compass

Using the Accelerometer

Sensors

Point your phone at the sky and Google SkyMap tells you what stars you’re looking at. Tilt your phone and you can control the game your playing. Take your phone on your walk and a breadcrumb app records your route. All of these apps are possible because the mobile devices we carry have high-tech sensors for detecting our location, orientation, and acceleration.

In this chapter, you’ll be introduced to the App Inventor components LocationSensor, OrientationSensor, and AccelerometerSensor. Along the way, you’ll learn about the Global Positioning System (GPS), orientation measures like pitch, roll, and yaw, and some math for processing accelerometer readings.
Creating Location-Aware Apps

Until the popularization of the smart-phone, computing was on desktop lock-down. Yes, laptops are mobile, but not in the same sense as the tiny devices we now carry around in our pockets. Computing has left the lab, left the office, and is now taking place out in the world.

One significance of carrying our computing with us is a new, very interesting piece of data for every app: a current location. Knowing where people are as they move about the world has far-reaching implications and the potential to help us greatly in our lives. It also has the potential to invade our privacy and be a detriment to humanity.

The Android Where’s My Car? app (chapter X) provides an example of a location-aware app of a personal benefit. It lets you remember a previous location so you can get back to it at a later time. That app is private: your location information is stored only in your phone’s database.

The same idea can be applied to groups. For instance, a group of hikers might want to keep track of each other’s whereabouts in the wilderness, or a group of business associates might want to find each other at a large conference. Such apps are starting to appear in the market place, with two of the most popular being Google’s Latitude (www.google.com/latitude), and Facebook’s Places (www.facebook.com/places). Due to the privacy concerns people have, all of these apps faced criticism on their launch.

Another type of location-aware app uses augmented reality tools. These apps use your location and the phone’s orientation to provide overlay information that augments the natural setting. So you might point a phone at a building and see its price on the real estate market or you might walk near an exotic plant in the botanical gardens and have an app tell you the species. Early players in this genre include Wikitude, Layar, and Google Skymap.

Wikitude even allows users to add data to the mobile cloud through its website wikitude.me. At wikitude.me you pull up a map to geotag the information you post. Later, when you or someone else uses Wikitude’s mobile app at that location, your information appears!

GPS

To create a location-aware app, you first need to understand how the GPS system works. GPS data comes from a satellite system maintained by the US government. As long as you have an unobstructed sight line to at least three of the satellites in the system, your phone can get a reading. A GPS reading consists of your latitude, longitude, and altitude. Latitude is how far north or south you are of the equator, with values for north being positive and south negative. The range is -90 to 90. Figure 23-2 shows a Google map of a spot near Quito, Ecuador. The latitude shown on the map is -0.01, just barely south of the equator!
Figure 23-2. Quito, Ecuador is on the equator.

Longitude is how far east or west you are of the Prime Meridian; east coordinates have positive values and west coordinates are negative. The most well-known place it runs through is Greenwich, a town near London which is the home of the Royal Observatory. The map in Figure 23-3 shows Greenwich and its longitude of 0.0.

Figure 23-3. Greenwich is on the Prime Meridian

Longitude values range from -180 to 180. Figure 23-4 shows a spot in Russia, very close to Alaska, which has a 180.0 longitude. You might say that a place like this (180.0 longitude) is half way around the world from Greenwich (0.0 longitude).
Sensing Location with App Inventor

App Inventor provides the LocationSensor component for accessing GPS information. The component has properties for Latitude, Longitude, and Altitude. It also communicates with Google Maps, so you can get a reading for your current street address.

The key event for the LocationSensor component is the **LocationChanged** event, pictured in figure 23-5.

![LocationSensor event handler](image)

*Figure 23-5. The **LocationChanged** event handler.*

This event is triggered both the first time the sensor gets a reading, and each time the phone is moved enough so that new data is read. There's often a delay of quite a few seconds before an app's first reading, and sometimes the device can't get a reading at all. For instance, if you're indoors and not connected to WiFi, the device might not get a reading. Your phone also has settings which allow you to turn GPS reading off to save battery life--this would be another reason the component can't get a reading. For these reasons, you shouldn't assume that the LocationSensor properties have a valid setting until the **LocationChanged** event occurs.

One way to deal with the unknowns concerning location sensing is to keep a variable **lastKnownLocation**, initialize it to “unknown”, then have the **LocationChanged** event change
the value of that variable, as shown in figure 23-6.

![Diagram](image)

*Figure 23-6. The value of the *lastKnownLocation* variable changes whenever the location changes.*

By programming the *LocationChanged* event in this way, you can always display the current location or record it in a database, with “unknown” appearing until the first reading. This strategy is used in No Text While Driving (Chapter X)—that app auto-responds to SMS texts and includes either “unknown” or the last reading taken in the response.

You can also ask explicitly whether the sensor has a reading using the *HasLatitudeLongitude* block pictured in figure 23-7.

![Diagram](image)

*Figure 23-7. The *HasLatitudeLongitude* block tests whether the sensor has a reading.*

**Checking Boundaries**

One common use of the *LocationChanged* event is to check whether the device is within a *boundary*, or a set area. For example, consider this code in figure 23-8 that vibrates the phone each time a new reading shows that a person is further than 0.1 longitude from the Prime Meridian.
Such boundary checking has numerous applications, e.g., warning parolees if they are nearing the legally specified distance from their home, or parents or teachers if a child leaves the playground area. If you’d like to see a slightly more complex example, see Chapter 21 on Conditional blocks.

**Location Information Providers: GPS, Wi-Fi, and Cell ID**

An Android device can determine its own location in a number of ways. The most accurate method—up to within a few meters—is through the satellites that make up GPS system maintained by the US Government. You won’t get a reading, however, if you’re inside and there are skyscrapers or other objects in the way: you need a clear path to at least three satellites in the system.

If GPS isn’t available or the user has disabled it, the device can obtain its position through a wireless network. You have to be near a Wi-Fi router, of course, and the position reading you’ll get is the latitude/longitude of that Wi-Fi station.

A third way a device can determine positioning is through Cell ID. Cell ID provides a location for the phone based on the strength of signals from nearby cell phone towers. It is not generally very accurate unless you have numerous cell towers near you. However, it does use the least amount of battery power compared to GPS or Wi-Fi connectivity.

**Using the Orientation Sensor**

The OrientationSensor is used for game-like apps in which the user controls the action by tilting the device. It can also be used as a compass to find out the direction (North/South, East/West) the phone is pointing.
The OrientationSensor has five properties, all of which are unfamiliar to most people other than aeronautical engineers:

- **Roll**: *Left-Right* The Roll is how much the device is tilted left or right. It is 0 degrees when the device is level, increases to 90 degrees as the device is tilted up onto its left side, and decreases to −90 degrees when the device is tilted up onto its right side.
- **Pitch**: *Up-Back* 0 degrees when the device is level, increasing to 90 degrees as the device is tilted so its top is pointing down, further increasing to 180 degrees as it gets turned over. Similarly, as the device is tilted so its bottom points down, pitch decreases to −90 degrees, then down to −180 degrees as it gets turned all the way over.
- **Yaw**: *Compass* 0 degrees when the top of the device is pointing north, 90 degrees when it is pointing east, 180 degrees when it is pointing south, 270 degrees when it is pointing west, etc.
- **Magnitude**: *Speed of a rolling ball roll* Returns a number between 0 and 1 that indicates how much the device is tilted. It gives the magnitude of the force that would be felt by a ball rolling on the surface of the device.
- **Angle**: *Angle of a rolling ball* Returns an angle that tells the direction in which the device is tiled. That is, it tells the direction of the force that would be felt by a ball rolling on the surface of the device.

The OrientationSensor also provides the **OrientationChanged** event which is triggered every time the orientation changes. To explore the meaning of these properties, write an app that shows you how the properties change as the user tilts the devices. Just add five heading labels, and five other labels to show the current values of the properties above. Then add the following blocks shown in figure 23-9.
Figure 23-9. These blocks output the OrientationSensor properties to text labels each time the orientation changes.

**Using the Roll Parameter**

Now let’s try to move an image left or right on the screen based on the user tilting the device, like you might do in a shooting or driving game. Drag out a Canvas and set the Width to “Fill Parent” and the Height to 200 pixels. Then add an ImageSprite or Ball within the Canvas, and a Label named RollLabel under it to display a property value: as shown in figure 23-10.

![ImageSprite](image.png)

**Figure 23-10. The image moves as the phone is tilted and the roll property shown in the RollLabel.**

The OrientationSensor’s roll property will tell you if the phone is tilted left or right, i.e., if your are holding the phone upright and tilt it slightly to the left, you’ll get a positive reading for the roll, if you tilt it slightly right, you’ll get a negative reading. Thus, you can let the user move an object with an event-handler like the following in figure 23-11.

![Code Blocks](code_blocks.png)

**Figure 23-11. The OrientationChanged event blocks to respond to changes in the roll property.**

The blocks multiply the roll by -1 as tilting left gives a positive roll, and tilting left should move the object left (thereby making x-coordinate smaller--for a review of how the coordinate system
works in animated apps, see Chapter X).

Note that this app works only when the device is in portrait mode (upright) and not in landscape mode. As is, if you tilt the phone too far, the screen will auto-rotate into landscape mode and the image will stay marooned onto the left of the screen. The reason is that if the device is on its side, it is tilted left and thus will always get a positive reading for the roll. A positive roll reading, in the blocks above, will always make the x-coordinate smaller.

If App Inventor provided the capability, you could either 1) lock the phone so it didn’t auto-rotate for this app, or 2) find out which mode the phone was in and modify your formula for moving the object based on that setting. Such capabilities will certainly be added to the system, but as of now you can only instruct your users on how the app will work.

**Moving Any Direction with Heading and Magnitude**

The sample above moves the image left or right. If you want to allow for movement in any direction you can use the *Angle* and *Magnitude* properties of the OrientationSensor. These are the properties used to move the LadyBug in the game described in Chapter X.

In figure 23-12 you can see the blocks for a test app that lets the user tilt the device to move a character in any direction (you need two labels and an image sprite for this one).

![Blocks to move a character in any direction.](image)

Try this one out: the *Magnitude* property, a value between 0 and 1, denotes how much the device is tilted. In this test app, the image moves faster as the magnitude gets bigger.

**Using the Phone as a Compass**

Compass apps and apps like Google’s Skymap need to know the phone’s orientation in the
world, east/west and north/south. Skymap uses the information to overlay information about the constellations at which the phone is pointing.

The yaw reading is useful for this type of orientation. Yaw is always between 0 and 360 degrees, with 0 north, 90 east, 180 south, and 270 west. So a reading of 45 means the phone is pointing north-east, 135 means the phone is pointing south-east, 225 means south-west, and a reading of 315 means north-west.

The blocks in figure 23-13 are for a simple compass: it displays in text which direction the phone is pointing (e.g., NorthWest).

![Figure 23-13. Blocks to program a simple compass.](image)

As you may have noticed, the blocks only show one of four possibilities: NorthWest, NorthEast, SouthWest, and SouthEast. As a challenge, see if you can modify it to show just a single direction (North, South, East, or West) if the reading specifies that you are within a few degrees
of directly pointing that way.

**Using the Accelerometer**

Acceleration measures the rate of change of velocity over time. If you press your foot to the pedal of your car, the car accelerates, its velocity increases at a particular rate.

An accelerometer, like the one in your Google device, measures acceleration, but its frame of reference is not the device at rest, but the device in free-fall: if you drop the phone, it will register an acceleration reading of zero. Simply put, the readings take gravity into account.

If you want to know more about the physics of the matter than that, you’ll have to consult your Einstein-related books. But in this section we’ll explore the accelerometer enough to get you started. We’ll even examine an app that could help save lives!

**Responding to the Device Shaking**

If you’ve been going through the chapters and completed the app of chapter 1 (HelloPurr), you’ve already used the AccelerometerSensor. In that app, you used the `Accelerometer.Shaking` event to make the kitty meow when the phone was shaken, as shown in Figure 23-14.

![Figure 23-14. These blocks play a sound when the phone is shaken.](image)

**Using the AccelerometerSensor’s Readings.**

Similarly to the other sensors, the Accelerometer has an event for when the readings change, `AccelerometerSensor.AccelerationChanged`. That event has three arguments corresponding to the acceleration in 3 dimensions:

- **xAccel**: positive when the device is tilted to the right (that is, its left side is raised), and negative when the device is tilted to the left (its right size is raised).
- **yAccel**: positive when its bottom is raised, and negative when its top is raised.
- **zAccel**: positive when the display is facing up, and negative when the display is facing down.

**Detecting Free Fall**

We know that if all of the acceleration readings are near zero, the device is free falling to the ground. With this in mind, we can mimic a “freefall” event by checking the readings in the
**AccelerometerSensor.AccelerationChanged** event. Such blocks, with lots of testing, could be used to detect when an elderly person has fallen and automatically send an SMS message out in response.

Figure 23-15 shows the blocks for an app that simply reports that a freefall has occurred (and lets the user click a reset button to check again).

![Image of blocks for freefall detection](image)

**Figure 23-15. Blocks that report when a freefall has occurred.**

Each time the sensor gets a reading, the blocks check the x-, y-, and z- dimensions to see if they’re near zero (if they’re absolute value is less than 1). If all three are, the app changes a status label to denote that the phone is in freefall. When the user clicks the ResetButton, the status label is reset to its original state (“Device has NOT been in freefall”).

If you’d like to try this app, you can download it at [book URL].

**Detecting Acceleration using Calibrated Values**

The AcclerometerSensor’s readings are calibrated to the free-fall state. If you want to instead measure the acceleration with respect to the phone lying inert on a table, you need to *calibrate* the readings to that standard. To calibrate means to check, adjust, or determine by comparison with a standard; in this case the standard you want is the readings when the device is lying flat.
To do this, you need the user to help you by lying the device flat on a table, then clicking a “Calibrate” button. When the button is clicked, the app records the readings for the flat surface. Those readings can then be used later, in `AccelerationChanged` events, to offset the new readings and tell you if the device is moved rapidly in some dimension.

Figure 23-16 shows a sample app that lets the user calibrate the readings, then detects acceleration.

![Detecting Acceleration app](image)

*Figure 23-16. This app lets the user calibrate the acceleration readings.*

You can download and install this app from [book URL]. Run it, then set the phone on a table and click “Calibrate.” The readings will appear in the “readings when flat” area. If you then raise the phone slowly, the readings in the “Significant accelerations” area won’t change. But if you raise the phone up rapidly, the “No” reading for Z will change to “Yes”, as shown in Figure 23-16. Similarly, if you move the phone rapidly across the table, you’ll get a significant acceleration for X or Y. Figure 23-17 shows the blocks for getting the initial calibration.
These blocks just take the readings and place them in three labels, XCalibLabel, YCalibLabel, and ZCalibLabel. The blocks also initialize the labels that will be used to report accelerations later, after this calibration step.

The accelerometer should get a reading of zAccel around 9.8 when the phone is flat, and xAccel and yAccel readings of around 0. But the calibration step tells us exactly how the Accelerometer is working. Once the calibration readings are set, your app can detect changes in the x-, y-, or z-dimension by measuring new readings offset from the old (similar to the boundary checking app covered in Chapter 18). Figure 23-18 provides the blocks for detecting acceleration using the calibrated readings.
These blocks will be triggered if the device is moved. The blocks check the new accelerometer readings to see if they are significantly different (within 3) of those taken when the phone was lying flat. Suppose that our calibration step had put a 9.0 in ZCalibText. If you slowly lift the phone up, the new readings will remain close to 9 and no change will be reported. But if you rapidly lift the phone, the reading will become significantly higher and the blocks will report a change.

**Summary**

Sensors are of great interest in mobile apps because they allow your users to truly interact with your environment. By taking computing mobile, you are opening up (almost literally) a whole world of opportunities in user experiences and app development. You’ll need to think carefully about how, where, and when you use sensors in your apps—there’s lots of privacy concerns out there and some people won’t use your app if they’re worried about what you’re doing with their sensor data—but with all the options in games, social networking, travel, and more, the possibilities are nearly endless.