|  |  |
| --- | --- |
| This tutorial shows how to use the *Orientation* and *Location* sensors to create a simple compass app. Objectives: In this lesson you will learn to  * create an app that   + uses the *Global Positioning System* (GPS) to display your latitude and longitude;   + uses the *OrientationSensor* to point to North;   + uses the *Canvas* component for drawing;   + uses an *ImageSprite* component to animate the arrow; * think abstractly about the problem of placing labels on various sized screens. | ***[Click to watch Preview Video](http://www.youtube.com/watch?v=3Mvjt3ZgjKU)*** |

## 

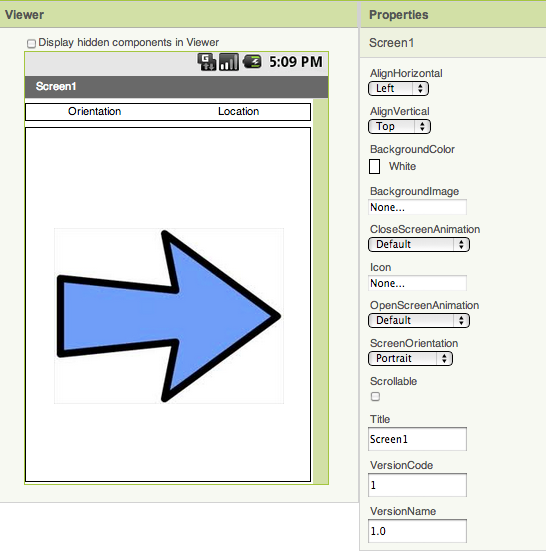
## Opening the Where is North Template in App Inventor

Open the [Where is North Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit2/templates/WhereIsNorthTemplate/WhereIsNorthTemplate.asc) in App Inventor. It opens in a separate tab. After the project opens, use the *SaveAs* option to rename it *WhereIsNorth*.

# Where Is North Tutorial

([Video Tutorial](http://www.youtube.com/watch?v=bMRuVNmdvgg))

## The Where Is North UI



**Setting up the UI**

1. The template contains a Canvas whose width and height have been set to *FIll Parent.* (NOTE how the Screen’s orientation has been set to *Portrait* mode and its *scrollable* property has been set to false. This is necessary to allow the Canvas to fill the screen.)
2. The Canvas contains an *ImageSprite* whose picture has been set to a blue arrow.
3. Two labels at the top of the screen will be used for displaying the phone’s orientation and location.

## 

## Basic Idea

The basic idea for this app is that the image sprite will ***always point due North*** and will be animated to turn in sync with the device’s orientation as the device is moved. The Orientation label will report the device’s current orientation with respect to due north as a number between 0 and 360 degrees, where 0 and 360 represent *North*, 90 degrees represents *East,* 180 degrees represents *South* and 270 degrees represents *West.*

In addition, the app will also display the device’s location in terms of *latitude* and *longitude.*

We will use the sensors’ event handlers to report this information.

## Canvas Component

The *Canvas* is App Inventor’s graphics component. It is used for drawing, painting, and displaying *ImageSprites.*  The Canvas component has a coordinate system that is similar to the *Cartesian* coordinate system that you might have learned about in geometry, but it has some important differences. Its main characteristics are summarized in the following diagram.

Origin

(0,0)

x-axis

y-axis

Canvas.Width

Canvas.Height

(width,height)

The Canvas’s *origin*, the point (0,0) is at its top-left. So its *horizontal* x-axis grows positively from left to right. Its *vertical* y-axis grows positively from top to bottom. Coordinate values on the Canvas are represented as *pixels*, which is short for *picture elements.* So, for example, the Width of the Canvas might be 300 pixels and its height might be 450 pixels.

## Completing the UI

### The Orientation Sensor

Drag and drop an Orientation Sensor from the Palette’s *Sensors* drawer onto the Viewer.

App Inventor’s *Orientation Sensor* is a non-visible component that provides information about the device’s orientation in 3 dimensions:

* **Roll**: 0 degrees when the device is level, increases to 90 degrees as the device is tilted up on its left side, and decreases to -90 degrees when the device is tilted up on its right side.
* **Pitch**: 0 degrees when the device is level, up to 90 degrees as the device is tilted so its top is pointing down, up 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 further decreases to -180 degrees as it gets turned all the way over.
* **Azimuth**: 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.

For this compass app we will only be using the device’s *azimuth,* which tells us the device’s position relative to due North. Note that the azimuth is reported as 0 degrees when the top of the devices is pointing North. You can [read more about the orientation sensor](http://ram8647-appinventor.appspot.com/learn/reference/components/sensors.html#OrientationSensor) in App Inventor’s glossary.

### The Location Sensor

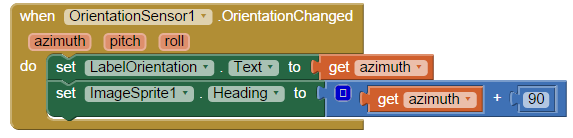
Drag and drop a Location Sensor from the Palette’s *Sensors* drawer onto the Viewer.

App Inventor’s *Location Sensor* (in the Sensor drawer) is a non-visible component that provides location about the device’s longitude, latitude, altitude and street address. It can also perform ***geocoding***. You can [read more about the Location Sensor](http://ram8647-appinventor.appspot.com/learn/reference/components/sensors.html#LocationSensor) in App Inventor’s glossary.

## 

## Coding the Compass

As noted above we want the arrow to move in sync with the phone’s azimuth. Therefore, we will use the Orientation Sensor’s *OrientationChanged* event handler to set the ImageSprite’s *Heading* property.



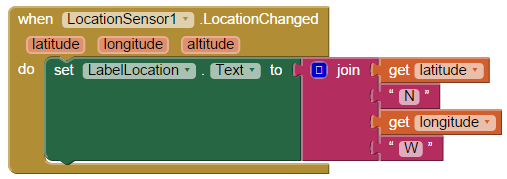
Note that we have set the sprite’s heading to (azimuth + 90). The reason for this is that we want the arrow pointing at the top of the screen when the azimuth is 0. But we need to make this adjustment here because the image sprite uses a different coordinate system than the orientation sensor. For an image sprite, its heading is 90 when it is pointing at the top of the screen (and 0 when it is pointing right, 180 when it is pointing left, and 270 when it is pointing to the bottom of the screen). Thus, by adding 90 to the orientation sensor’s azimuth value, we can get the arrow to point to the top of the screen when the azimuth is 0 (North).

## Coding the Location

A mobile device can detect its location in one of three ways:

* Using its built-in GPS sensor. This is the most accurate but, ideally, requires that the phone have a clear shot of the sky so that it can receive readings from at least 3 GPS satellites. This is accurate within a few meters but uses the most battery power.
* Using a Wifi signal from surrounding Wifi router. The phone’s location would be the latitude and longitude of the router. This might work indoors and uses less battery power.
* Using the Cell ID -- i.e., signals from surrounding cell towers. This is least accurate but uses the least power.

For our app we will just report the phone’s latitude and longitude in the *LabelLocation* whenever the phone’s location changes:



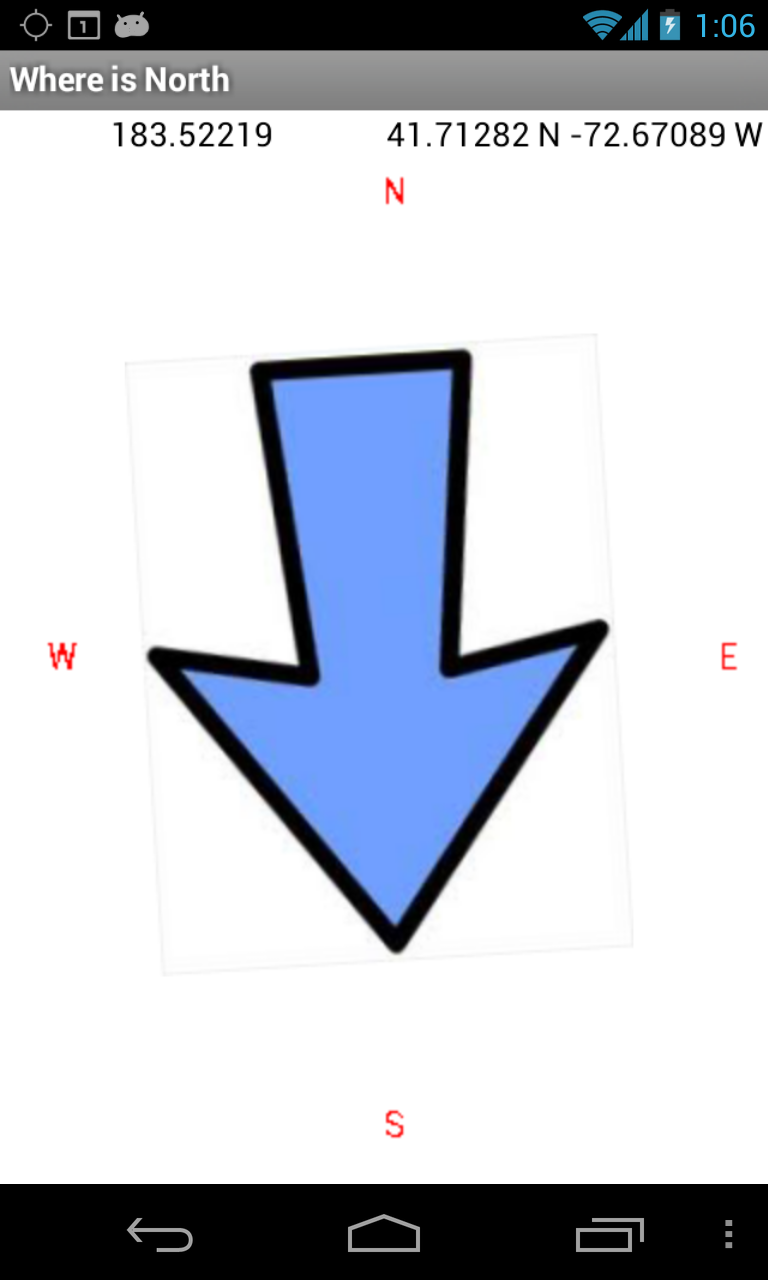
In the template we have changed the Location Sensor’s default *TimeInterval* from 60,000 milliseconds (every minute) to 10,000 milliseconds (every 10 seconds). That means that the phone will check for a location change every 10 seconds -- hence using more power.

## Running and Testing the App

After you have made these code implementations, run the app on your mobile device and test how it works. You’ll probably want to go outside to perform some of your tests.

# Enhancing the UI

To help the user find their direction, it would be useful to put some directional markers on the canvas as shown in this screenshot:

The red directional letters, N, E, S, and W, help orient the user by identifying where East, South, and West are when the arrow is pointing to the N -- i.e., due north. 

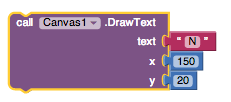
The idea is that the user would turn the phone until the arrow points to the N and then East would be to the user’ right, West to the left and South would be behind the user.

Using the***Canvas.drawText*** method, draw the red letters in the proper positions on the screen.

Because you want the directional letters to always be visible, you will want to do the drawing in the *When* *Screen1.Initialize* block.

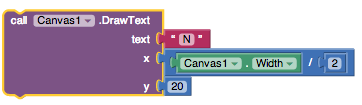
# Designing for All Devices

One important design issue that you need to consider as you work on this task is that your app will be run on different devices, with different size screens. Therefore, you shouldn’t draw the directional letters, N, E, S, W, at absolute coordinates. For example, one way to draw the letter N along the top middle of the screen, would be to set the *x*-parameter to 150 and the *y*-parameter to 20:



However, this solution would not be ***portable*** -- it will only work for a device that has a screen that happens to be 300 pixels wide. For example, it might run on my Nexus phone, but it won’t run on my Nexus tablet, which has a much wider (and higher) screen.

A better -- a more general, and therefore more portable -- way to do this would be to use the Canvas’ ***width*** property to specify the x coordinate:



The ***Canvas1.Width*** is a property of this app’s Canvas. As long as the Canvas has its Width set to ***fill parent***, the value of this property will be the same width as the screen -- ***no matter what device the app is run on.*** Because it is equal to the Screen’s Width, the Canvas’ Width property will be set by the device’s operating system. So it will be the right value for any device.

## Important Principles

There are a couple of important principles involved in this design choice:

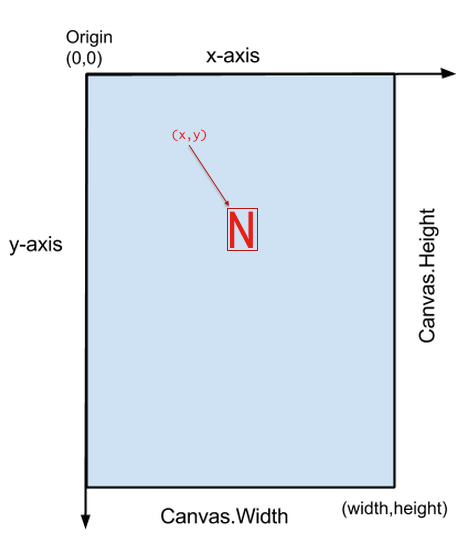
1. **Design for Shareability**: Design your apps with sharing in mind -- i.e., so they will work on a wide variety of devices. This is especially important if you plan to share your app with friends or market your app.
2. **Abstraction:** Design for generality. That is, try to make your solutions to problems as general as possible. In this case, by using a variable, the Canvas’ Width property, instead of a constant like 150, we’ve made our app more general. That’s just a better solution.

Now try to apply those principle in figuring out how to position the other directional labels, E, S, and W. Some hints:

* E has to be on the right edge of the Canvas (Width). And it has to be halfway between the top and bottom of the screen (Height).
* S has to be on the bottom edge of the Canvas (Height). And halfway across (Width).

## Other Canvas Issues

When you draw a letter on the canvas using ***Canvas.Drawtext***, you are positioning the letter by its x and y coordinates. But what do those refer to, exactly? The answer is that they refer to ***the top-left corner of the rectangle that encloses the letter.*** Here’s an example:



***Nice work! Complete the Self-Check Exercises and Portfolio Reflection Questions as directed by your instructor.***