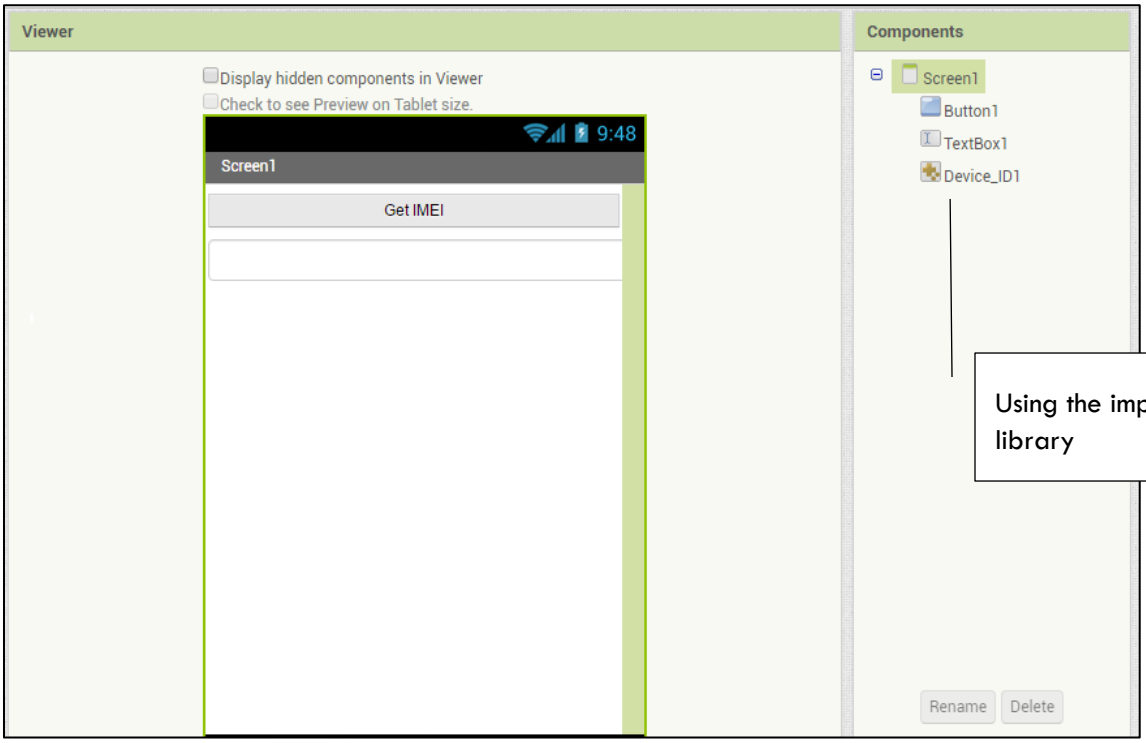
# Tutorial 3 - Solutions

## Getting a cell ID / IMEI

*Have a go yourself!*

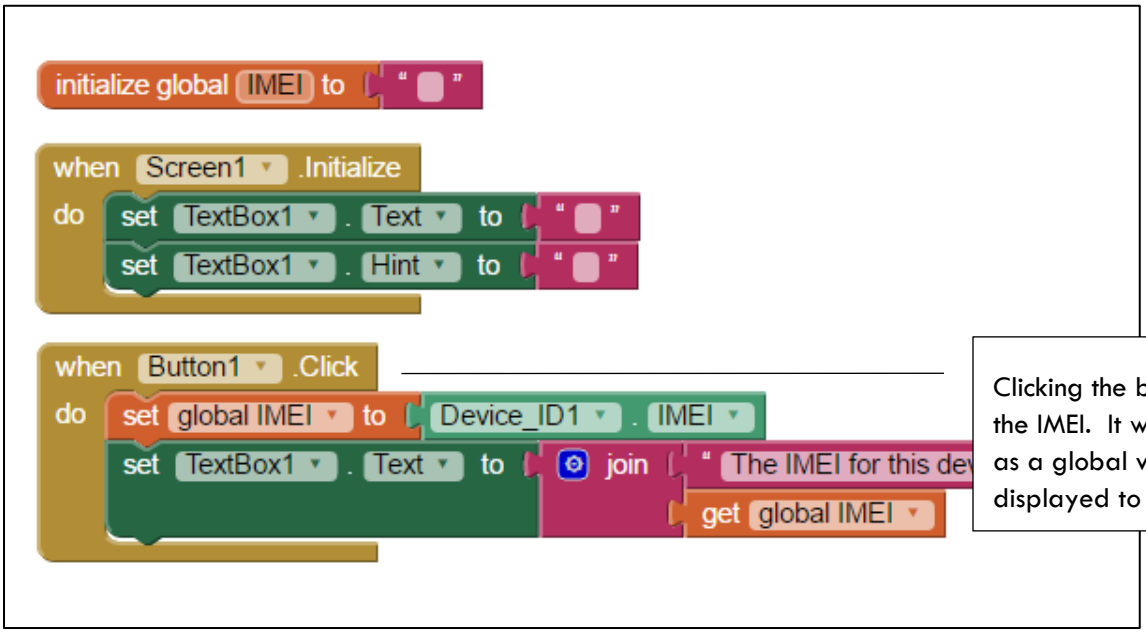
Create an app that displays the IMEI of your device.

**Designer with Components:**




Using the imported library

**Blocks**




Clicking the button will get the IMEI. It will be stored as a global variable, then displayed to the screen.

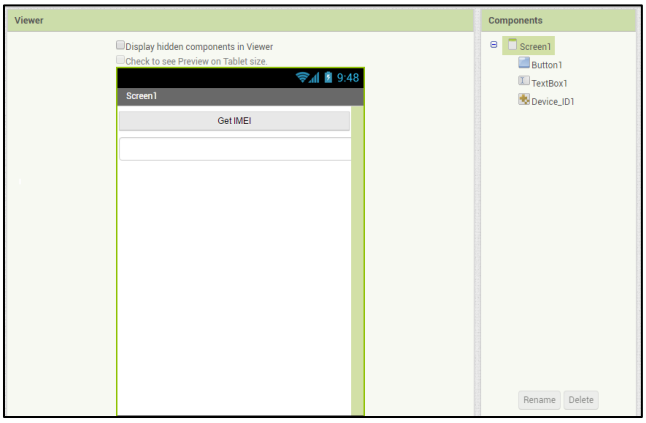
# Data Storage and Sharing

## TinyDB

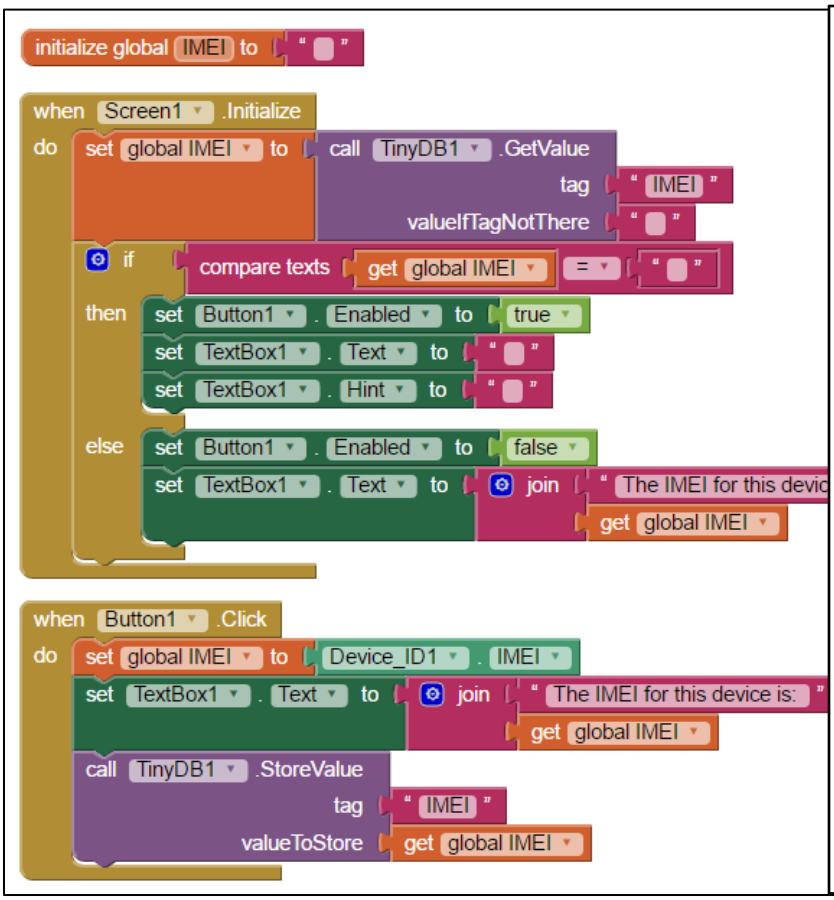
*Have a go yourself!*

Using the library from getting a cell ID/IMEI, create an app that holds the IMEI identification from your device. Store the code to the tinyDB then have the data persist so that if you open the application again, the value will stay loaded.

**Designer with Components:**




**Blocks:**




**How does this work?**

When the screen in initialised, the tinyDB will check if there is an entry for the IMEI. If the entry does not already exist, it will be created with the value of an empty string. In this solution, the tinyDB value for the IMEI will be stored as a global variable for the application. **So, if the entry does exist, the global variable will hold the value that is currently stored. If it does not, it will hold the empty string.**

This means we can test the empty string to decide how the program will work. If the value for the IMEI is the empty string, we know it did not have a record, so the button to find the IMEI will be enabled, and the text box will display nothing.

else, (if the IMEI was already stored in the database), the button will be disabled, and the text box will be filled with the details of the IMEI.

If the button is enabled, then when it is clicked, it will take a reference of the IMEI and print it to the text box on the screen. It will also store the details into the tinyDB for data persistence.

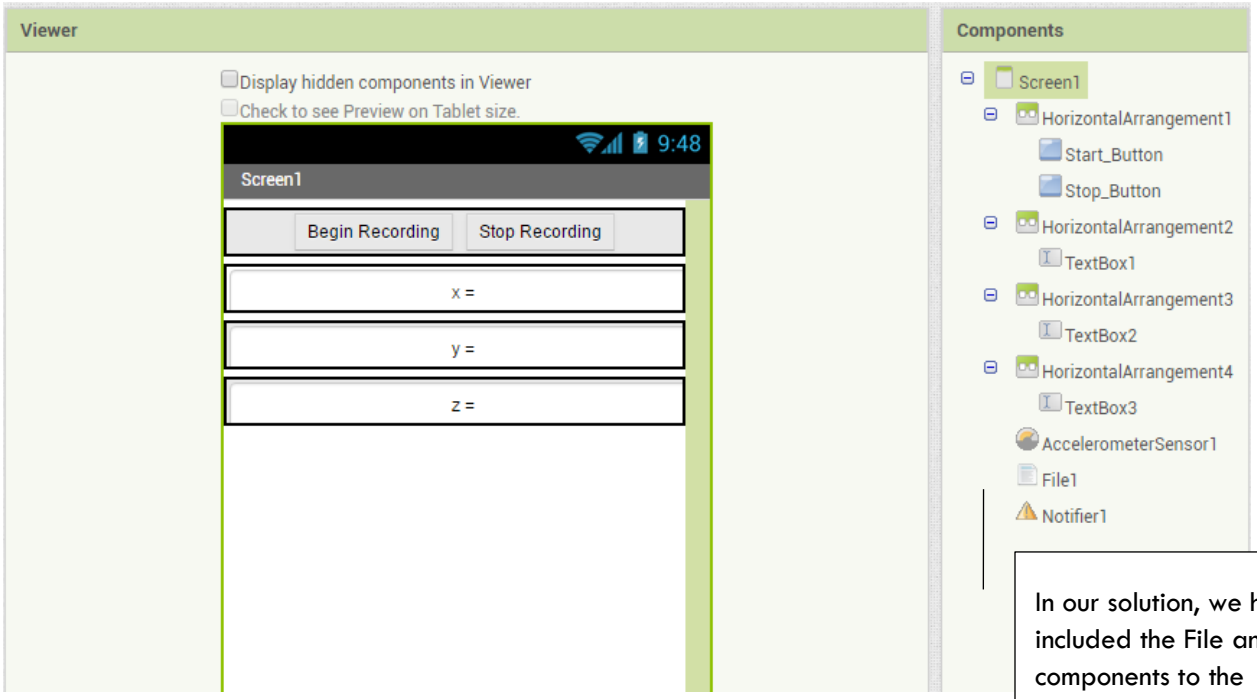
# File

*Have a go yourself!*

While this library is simple, there is no reason why we can't record data to useful file formats such as .csv. A .csv file is a simple file format, which displays tabular data such as a spreadsheet or data base.

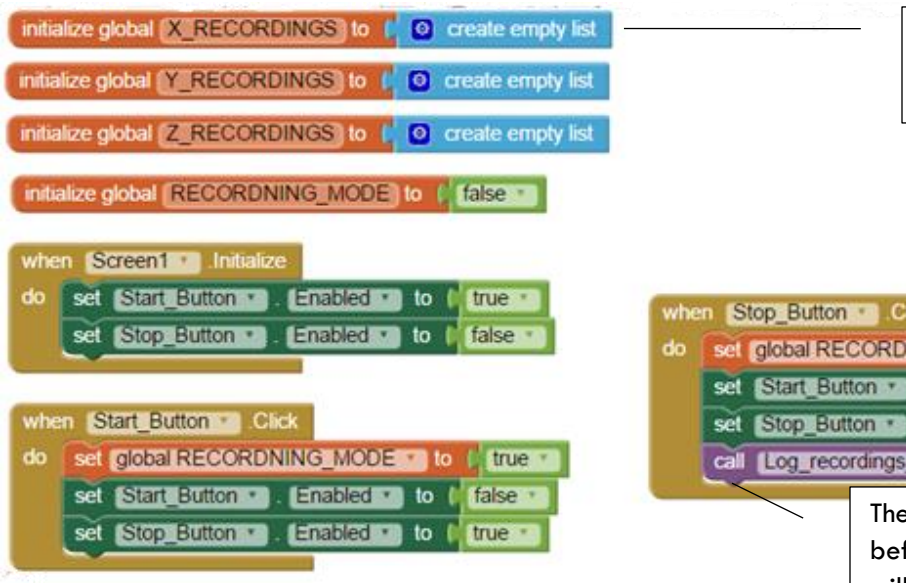
Make a copy of the application you made for the accelerometer last week. Amend this new program so that data from the recording mode will be saved to a .csv file locally on your phone.

**Designer**

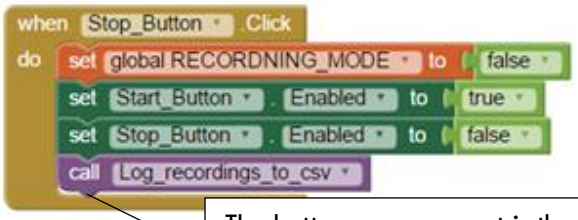



In our solution, we have included the File and Notifier components to the application

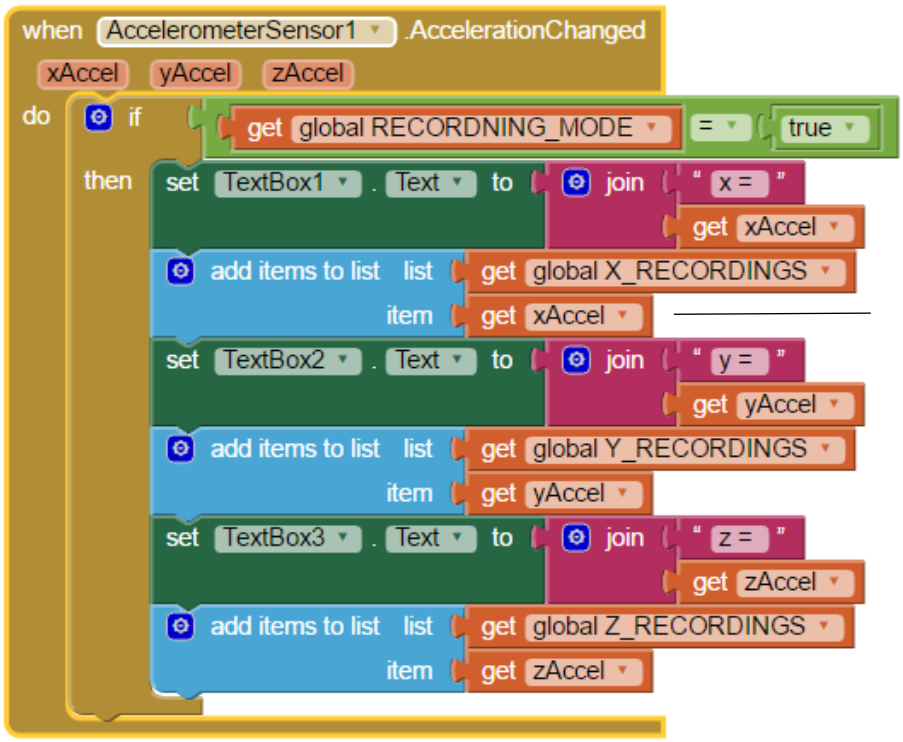
**Blocks**



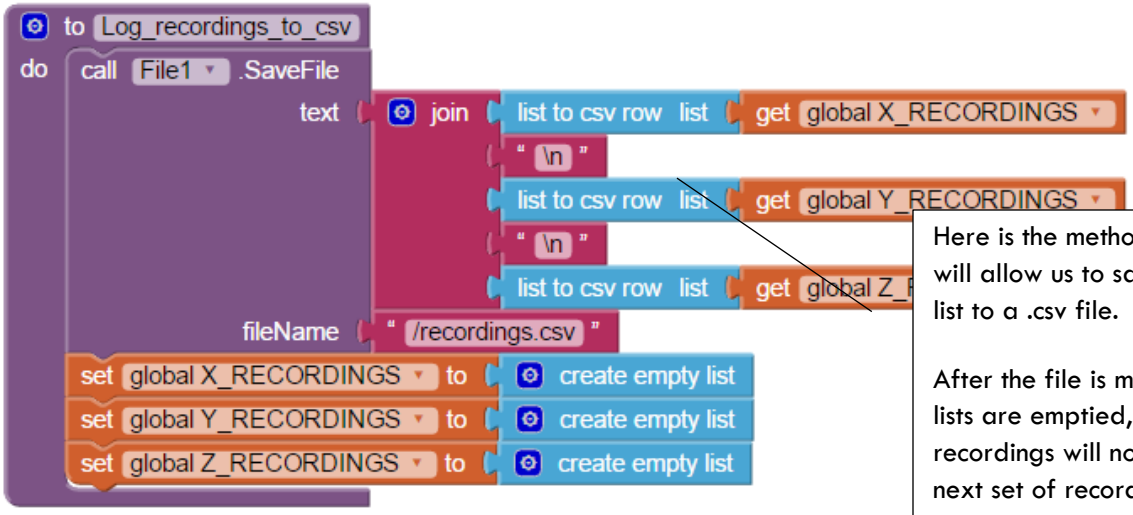

We are using global lists to keep track of x, y and z recordings




The button management is the same as before, except once we stop recording, we will log the details to a csv. We made this method ourselves, see below for details



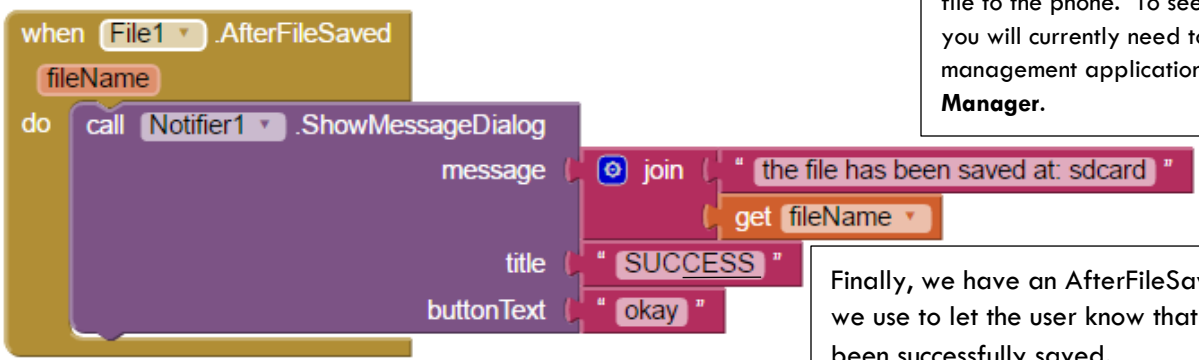

Now, when the accelerationChanged event occurs, we add the reading to the correct list, as well as updating the value on the screen.




Here is the method we made, which will allow us to save the details of the list to a .csv file.

After the file is made, we ensure the lists are emptied, so that the current recordings will not continue into the next set of recordings (i.e; if you were to click start again).

Note: This method/application saves the file to the phone. To see/access the file, you will currently need to use a file management application such as **File Manager**.




Finally, we have an AfterFileSaved event which we use to let the user know that the file has been successfully saved.

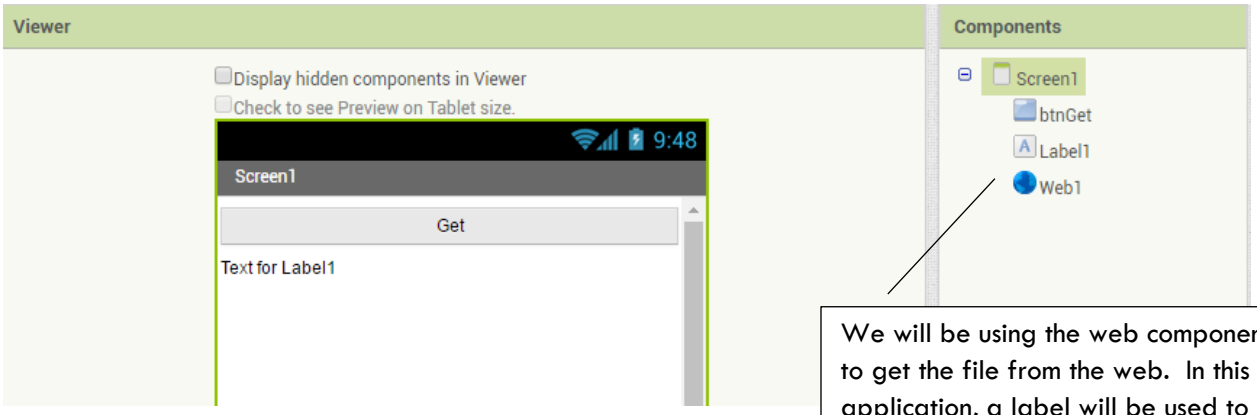
# File download

*Have a go yourself!*

Create an application which will download a file from the cloud to your phone:

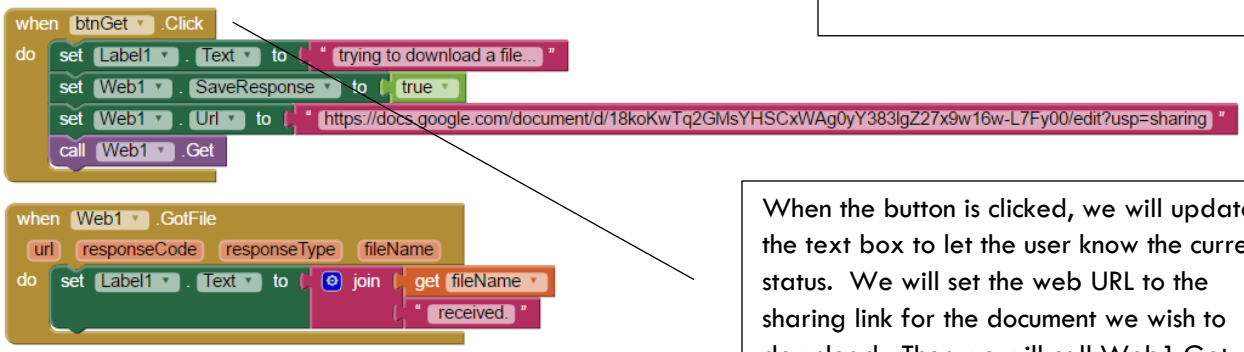
1. Setup a file (such as a google doc, or dropbox doc)
   *Hint: for this to work, the file will need to have sharing rights so that it can be downloaded. I.e: Look for an option that will create a shareable link to the file.*
2. Use the sharing link for the file in the web1 URL. Then use the web1.Get call to get a copy of the file.
3. Once the file has been downloaded to your phone, display the location of the file.

**Designer**




We will be using the web component to get the file from the web. In this application, a label will be used to let the user know the status of the file download

**Blocks**




When the button is clicked, we will update the text box to let the user know the current status. We will set the web URL to the sharing link for the document we wish to download. Then we will call Web1.Get

When the GotFile Event is activated, we can let the user know that the file has been downloaded by updating the text in the label1.Text field.

Note: To see/access the file, you will currently need to use a file management application.

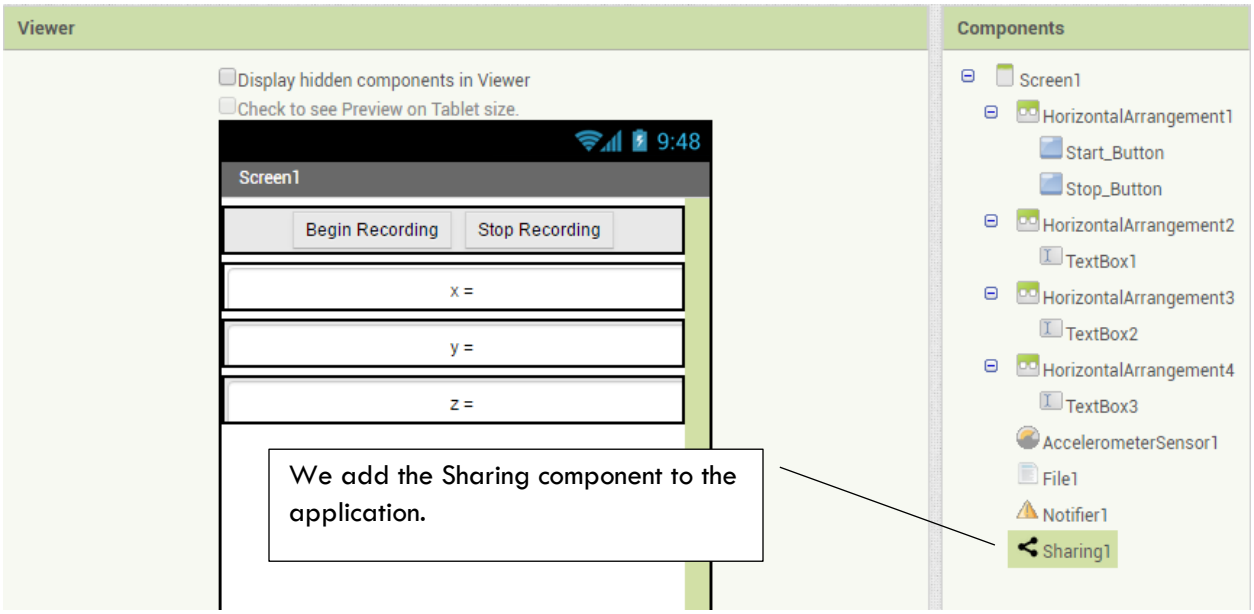
# FILE sharing

*Have a go yourself!*

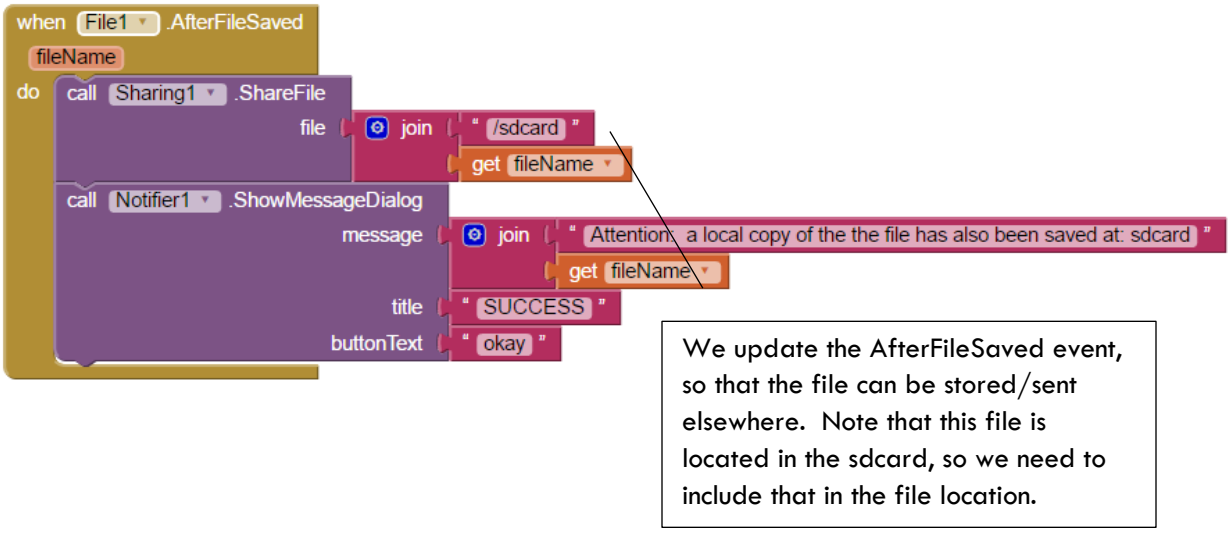
Make a copy of your solution from either **File** or **File Download**. Once the required event (AfterFileSaved or GotFile) has been activated, activate the fileSharing component so that the user can choose how they will store/share the file.

**Option 1: Sharing a File from local storage**

**Designer**

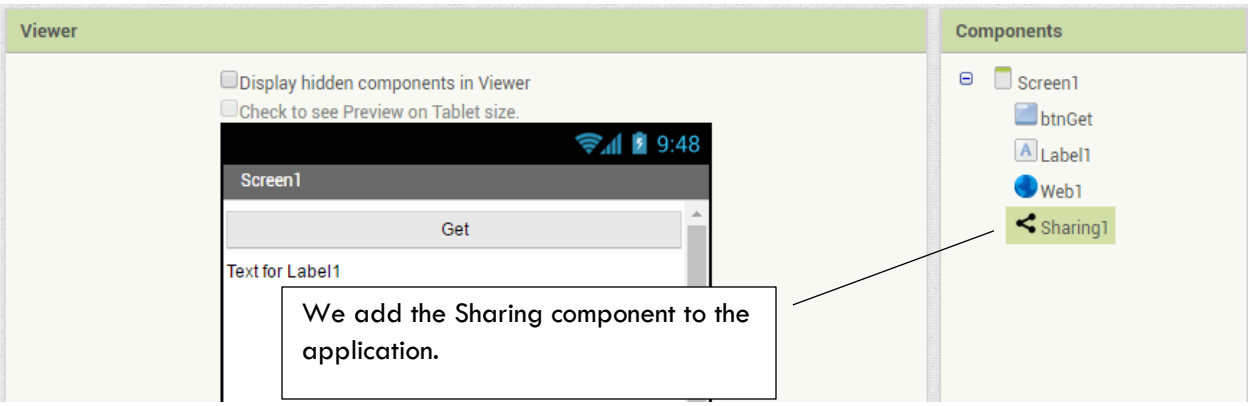



**Blocks**

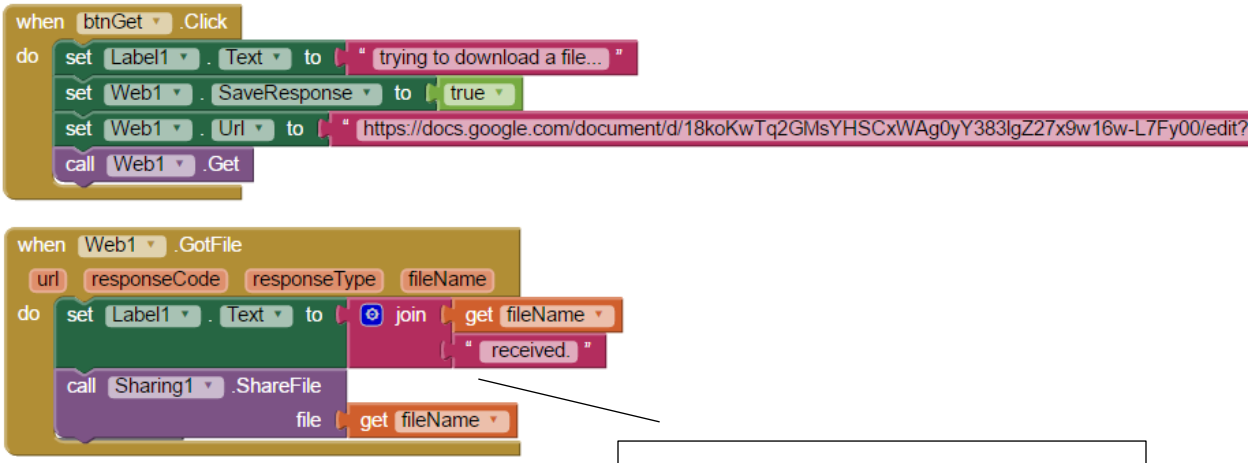


**Option 2: Sharing a File downloaded from the cloud**

**Designer**




**Blocks**




We update the GotFile event, so that the file can be stored/sent elsewhere. Note that in this case, the file location is included in the filename, so we only need to give a reference to the file name when sharing a web-based file.

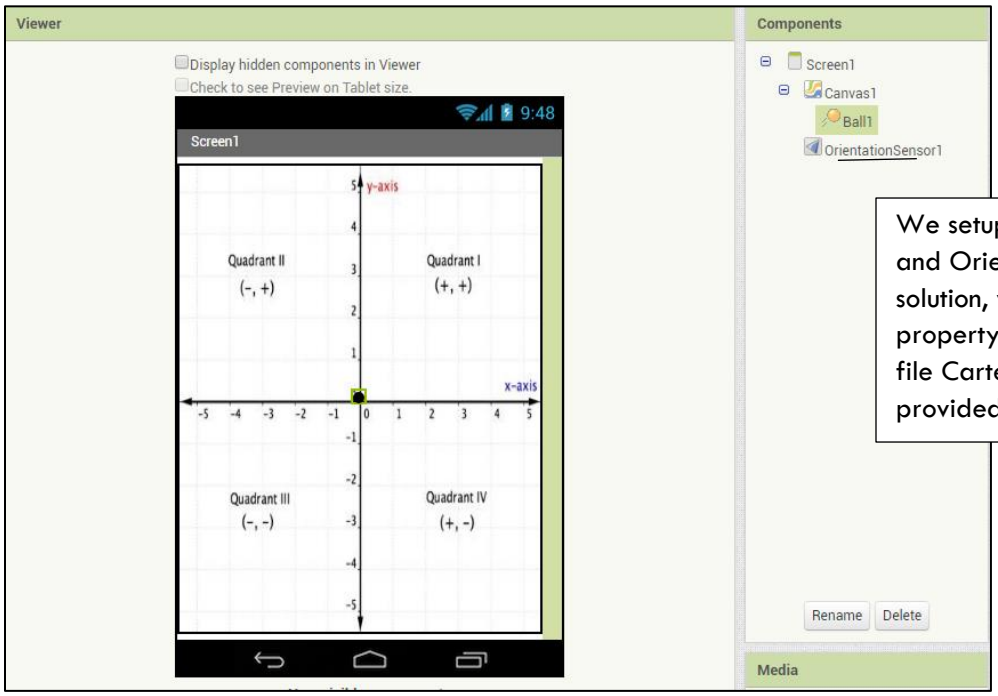
# Drawing and Animation Components

*Have a go yourself!*

Create a canvas displaying the Cartesian quadrants
*Hint: use the provided image file*

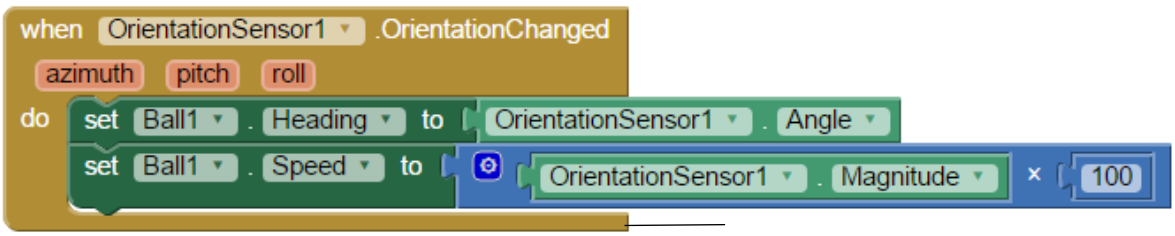
Using a ball sprint with orientation sensor, move the ball around on the screen.
*Hint: use the orientation sensor's angle and magnitude properties to control the ball*

**Designer**




We setup a screen with a Canvas, Ball and Orientation Sensor. In our solution, we will set the canvas width property to "fill parent" and load the file Cartesion_init.jpg (this image is provided with this week's material)

**Blocks**




Whenever the orientation changes, we will use the Angle from the Orientation Sensor to change the direction that the ball is heading. The ball will then be directed to move in the direction dictated by the orientation sensor.

The speed of the ball can also be adjusted using the Orientation Sensors Magnitude property. Note that the Magnitude will return a value between 0 and 1, so we multiply it by 100, as this will have a more visible effect.

# Multiple screens

*Have a go yourself!*

Create an initial screen manager which displays an initial set of Cartesian quadrants.
Then create another 4 screens, displaying a different image for Q1, Q2, Q3 and Q4.
*Hint: You can use the provided image files, or make your own*
*Hint: Setup a canvas on each screen, loaded with a corresponding image.*

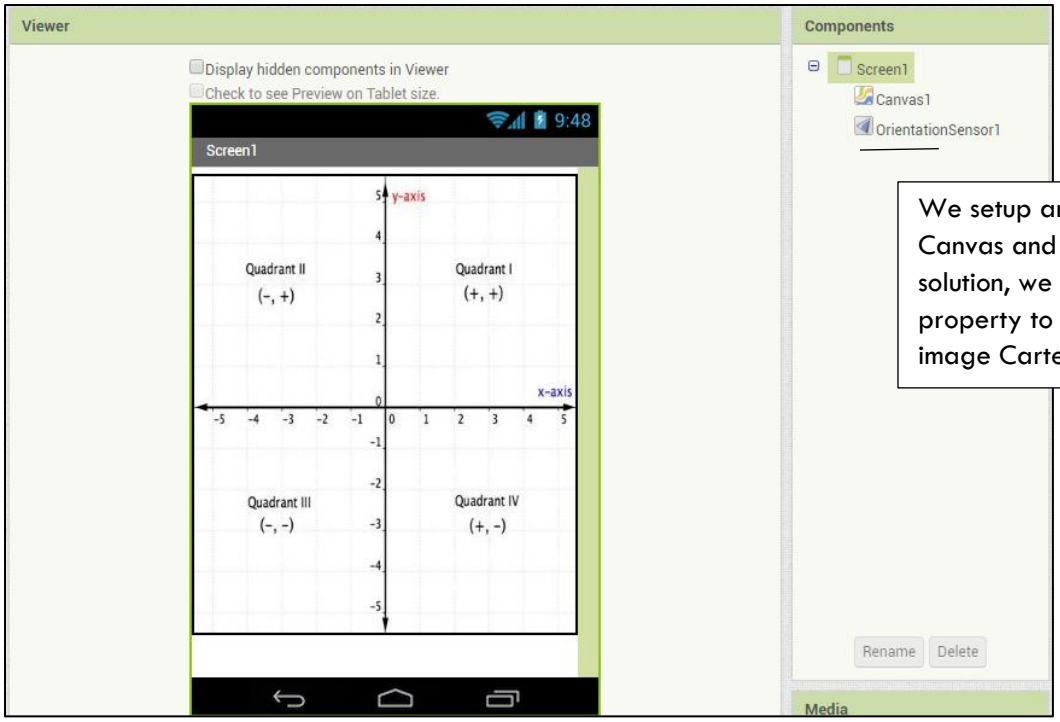
Using the orientation sensor's values of roll for the x-plane, and pitch for the y-plane, change the screen which is displayed:

For example:
From the initial screen manager, a reading of +y AND +x would direct to Q1.
From Q1, a reading of -y OR -x would cause the screen to close

and so on…
*Hint: All screens will require the orientation sensor.*

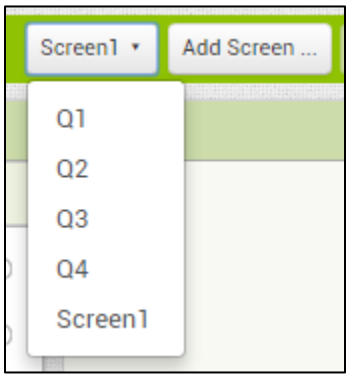
*Hint: Multiple screen application work best if a built application ( .apk file) is downloaded and installed to the phone.*

**Designer**




We setup an initial screen with a Canvas and Orientation Sensor. In our solution, we set the canvas width property to "fill parent" and load the image Cartesion_init.jpg.

We then used the add screen button to add screens for Q1, Q2, Q3 and Q4



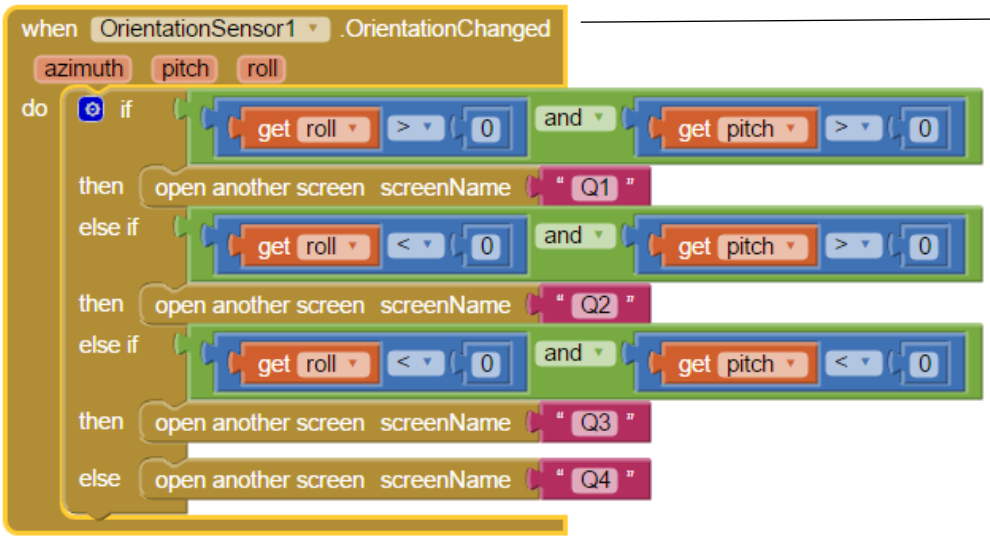

Note that all screens were loaded with a canvas, and orientation sensor. The canvas on all screens has the corresponding Cartesian image listed below, and the width property set to fill parent.

| Screen | Image |
|---|---|
| Q1 | Cartesion_I.jpg |
| Q2 | Cartesion_II.jpg |
| Q3 | Cartesion_III.jpg |
| Q4 | Cartesion_IV.jpg |

Note that you can use the images provided with the tutorial, OR find/make your own images for this solution.

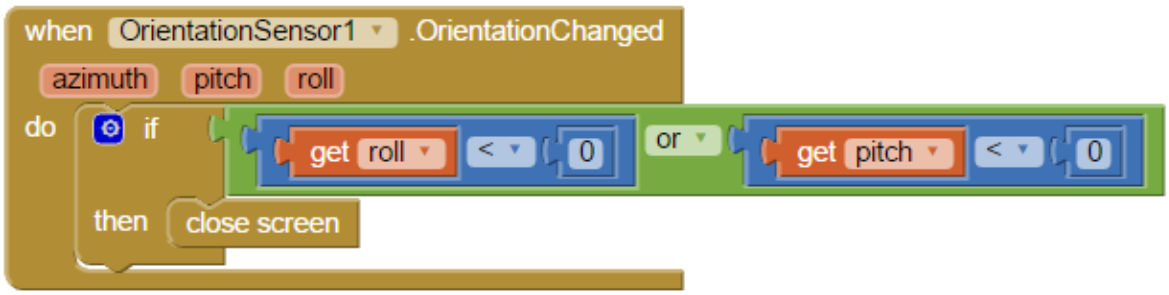
**Blocks**

**Screen 1**



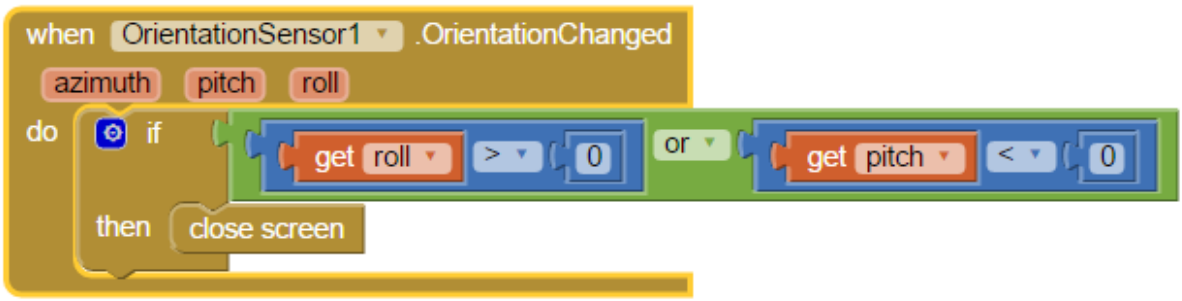

This is our manager screen, it will direct to Q1, Q2, Q3 or Q4 depending on the roll or pitch values.

**Q1**

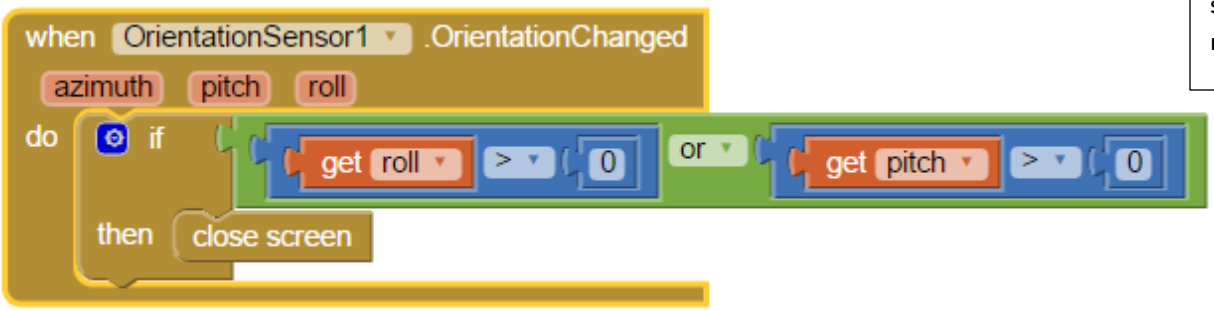



For the other screens, as soon as the orientation sensor detects a reading for roll or pitch which is not in range for the desired quadrant, the screen will be closed. This will force the application to go back to Screen 1, where the application will then direct to the updated quadrant screen as required.

**Q2**




**Q3**




**Q4**