

# Tutorial 3

**Disclaimer:** In your life as a software engineer, it will be important to know how to use third party software platforms and their materials. As such, **many of the materials we recommend for this course will be third party.** This will differ from some of your other programming courses, where you will need to write most of the software yourself. It is important to develop both sets of skills, that is, using third party materials as well as knowing how to write your own programs.

Last week, we took a look at the sensors on an android phone, and the MIT app inventor libraries for using these sensors. This week, we will be introducing some other miscellaneous features which can be used with MIT app inventor. These features should help you with your project later down the track.



## Other Useful Features

Apart from the android sensors, there are some other useful features which you might find useful. These include things like:

- Getting a cell ID / IMEI
- Data storage and data sharing components
- Drawing and animation components
- Multiple screens

Note that some sections in this tutorial are presented in grey. While you are free to learn as much as you like about these features (if they are of interest to you), they have been marked in grey because we feel they may not be the most important features to learn in the context of this course.

As with last week's tutorial, some of these requests we make will be classified based on the technique we are trying to teach:

	Tasks related to data collection or data maintenance can be identified with Yoshi.
	Tasks related to conditional statements and user interactions can be identified with a mushroom.

## Getting a cell ID / IMEI

Every mobile phone has a unique International Mobile Equipment Identity code associated with it, known as the IMEI code. Obtaining this code from a phone is relatively simple, and may assist you with the design of your application for this project.

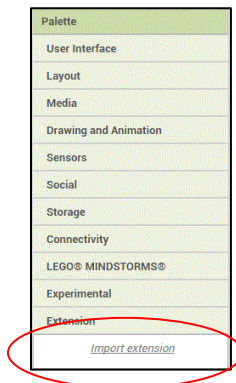
We have created a library which will find/locate the IMEI of any device. To use this library, you will need to import it into the project you wish to use.

### *Load new library to your project*

From the blackboard materials released this week, download the file:  
au.edu.newcastle.engg1500.PseudoDevice.aix

From the MIT app inventor build environment, select the project you wish to add this feature to

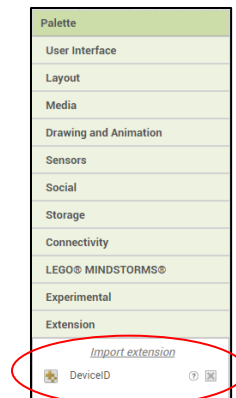
In the project design page, select import extension:



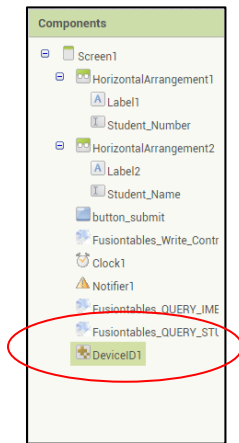
When the import box opens, select “from my computer”, select the file you just downloaded and click on import.

Give the extension package a name such as “DeviceID”.

The library will be available for use. You will find it listed in the extension panel.



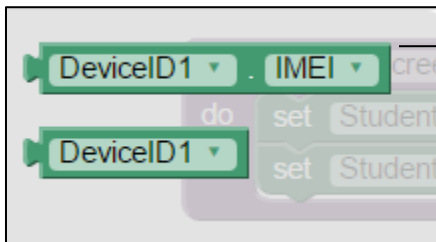
To add the library to your application, drag and drop the library to the Viewer of the screen you wish to add this component to. The Library should now appear under components for this screen.



### Using the Library

Navigate to the blocks view for the screen using this library.

You will notice that this library offers two features. **You will only need to use the first method.**



This method will return a String containing the IMEI from a phone. If you are using the emulator, it will return "SIMU1234".

### Have a go yourself!



Create an app that displays the IMEI of your device.

### For students using the emulator

Note that this feature will work for students using the emulator, returning the value "SIMU1234"

## Data Storage and Sharing

You may find that you need to store data or share it for your application to work. The MIT app inventor platform offers a couple of libraries which will help with data storage. The documentation about these libraries can be found [here](#)

In this section, we'll take a look at the most useful of these libraries.

### TinyDB

An android phone comes with a small data-base. Androids TinyDB can be used to provide persistent local storage of data for an application.

**What is local storage:** This refers to a disc which is directly attached to the device. Data stored on this disc is stored locally.

**What is persistent data:** In this case, this refers to data which persists over different runs of an application. i.e; you can open an application many times, and the data will still be available. It is usually used for data components which will not change during different runs of the application (such as login codes, etc).

MIT tutorials using TinyDB can be found [here](#) and [here](#)

*Note that these are the same tutorials listed for the location sensor in last week's tutorial sheet. If you have already completed these tutorials, you only need to review the applications with regard to how they use the tinyDB for persistent data storage of the GPS co-ordinates.*

### 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.

### For students using the emulator

Note that this feature is only available in the emulator, if the data to be stored in the tinyDB comes from a working component or sensor in the emulator. This component will work for students using a phone via a wifi connection or USB connection

## Fusion tables

The MIT app inventor platform provides a library which will enable you to write data to a fusion table.

A fusion table is a cloud based storage spreadsheet.

**This is an advanced feature and use of this feature is not recommended during this course.**

## File

The MIT app inventor platform provides a File library which will enable you to read, or write files locally to a phone.

There is no MIT tutorial on this feature, however, a simple YouTube tutorial can be found [here](#).

### 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.

Blocks

- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists**
  - Colors
  - Variables
  - Procedures

Viewer

Hint: Use some global lists (from the Built-in blocks) to store the readings before writing them to file.

create empty list

make a list

add items to list list item

is in list? thing

## File download

The MIT app inventor platform is also able to download cloud based files using the web component. More details about the web component library can be found [here](#).

There is no MIT tutorial on this feature, however, a simple tutorial can be found [here](#). Note that the provided tutorial uses a cloud based *image*.

### 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.

### For students using the emulator

Note that this feature is not available for students using the emulator. This sensor will work for students using a phone via a wifi connection or USB connection.

## FILE sharing

The MIT app inventor supports file sharing. That is, sharing a file or text message to another phone application such as g-mail, google drive or messenger.

An MIT tutorial about file sharing can be found [here](#)

### *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.

### *For students using the emulator*

Note that this feature is not available for students using the emulator. This sensor will work for students using a phone via a wifi connection or USB connection



## Drawing and Animation Components

The MIT app inventor includes libraries with drawing and animation components. These components allow you to control images on the screen, so that they can be controlled by either user interaction, or by image interaction with each other. The documentation about these libraries can be found [here](#).

MIT have included a number of tutorials using these components which can be found [here](#).

Note that in last week's tutorial, for the Clock sensor, we recommended you review the [MoleMash](#) tutorial. For more experience with the drawing and animation components, we now recommend you look at the [MoleMash2](#) tutorial. We can also recommend the [SpacInvaders](#) tutorial.

### *Have a go yourself!*

Create a canvas displaying the Cartesian quadrants

*Hint: use the provided image file*

Using a ball sprite with orientation sensor, move the ball around on the screen.

*Hint: use the orientation sensor's angle and magnitude properties to control the ball*

### *For students using the emulator*

Note that this feature is not available for students using the emulator. This sensor will work for students using a phone via a wifi connection or USB connection.

## Multiple screens

Some applications work better with multiple screens. The MIT app inventor platform allows you to create applications which will have more than one screen.

Building an app with multiple screens is a like creating several individual apps. Every screen that you create has its own components in the Designer window.

When you are working in the Blocks Editor for a screen, the variables and details will be local to the screen. That is, you will only be able to see only the components of the screen you are working with, and cannot easily refer to blocks of code from in another screen.

If required, you can enable your screens to communicate/share information with each other, using one of the following approaches:

**Option 1:** passing and returning values when the screens open and close.

**Option 2:** using androids tinyDB to store information in one screen, which another can retrieve later.

For practice using multiple-screens, we recommend running through the MIT tutorial which can be found [here](#). This tutorial uses Option 1 to pass information between the screens. Note that this is an advanced tutorial and may take several hours to complete.

### 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.*

### For students using the emulator

Applications using multiple screens should work with the emulator, if the driver for changing screens is a component or sensor which also works with the emulator.