

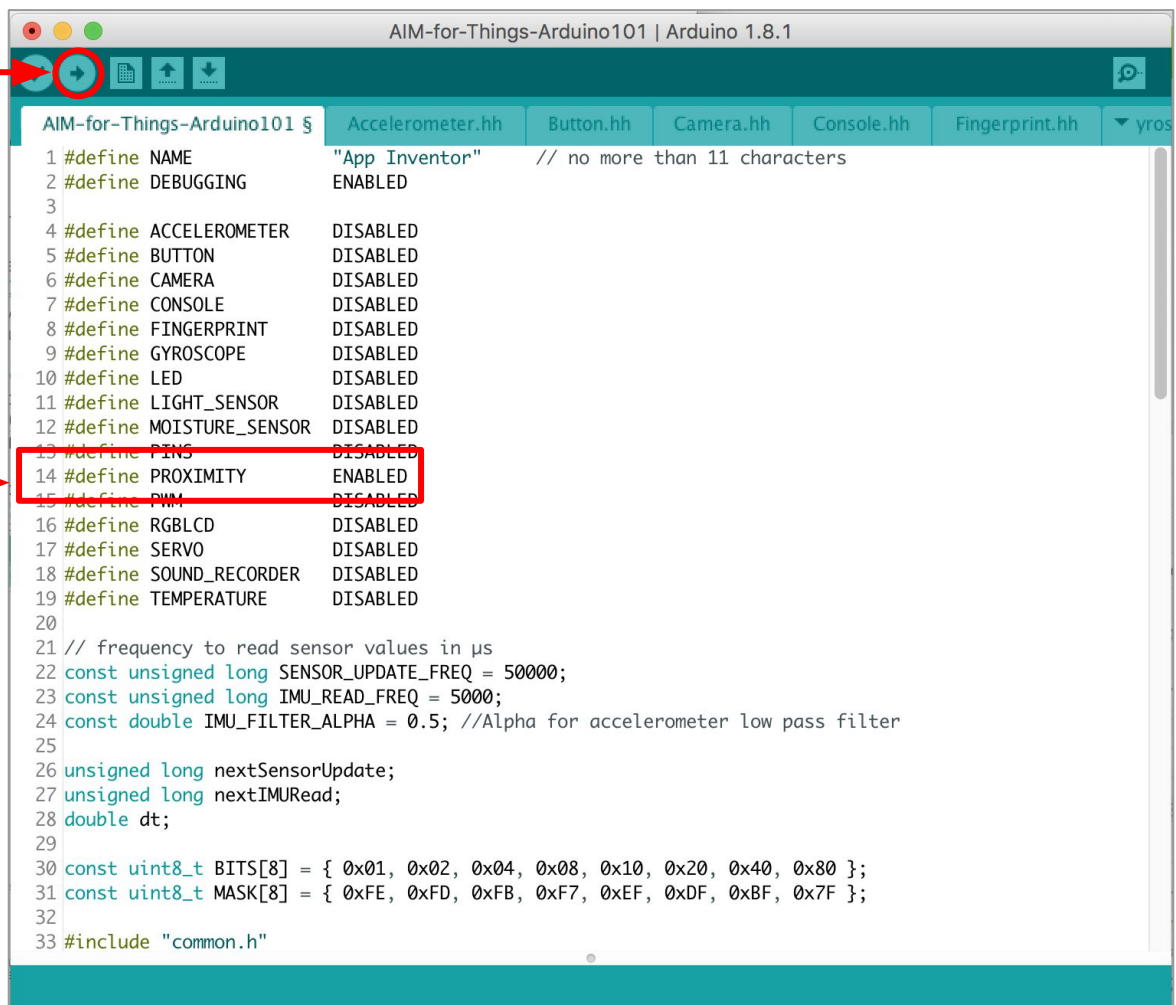
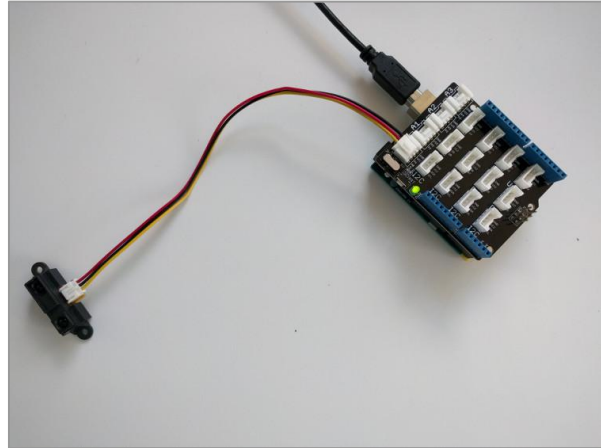


App Inventor + IoT: Proximity Sensor

This tutorial will help you get started with App Inventor + IoT and a proximity sensor on an [Arduino 101](#) controller. We are also using a [Seeed Grove](#) shield for this tutorial. You do not need to use this board, but it does make things easier. The proximity sensor we recommend is the [Grove Infrared Proximity Sensor](#).

Before you start you should first complete the [App Inventor + IoT Setup tutorial](#) to set up your Arduino device.

- Connect the proximity sensor to the Grove board in the A0 pin connector. 
- For this tutorial make sure **PROXIMITY** is set to **ENABLED** and all others are set to **DISABLED**.
- You should also click the arrow button in the top left to upload the code. 

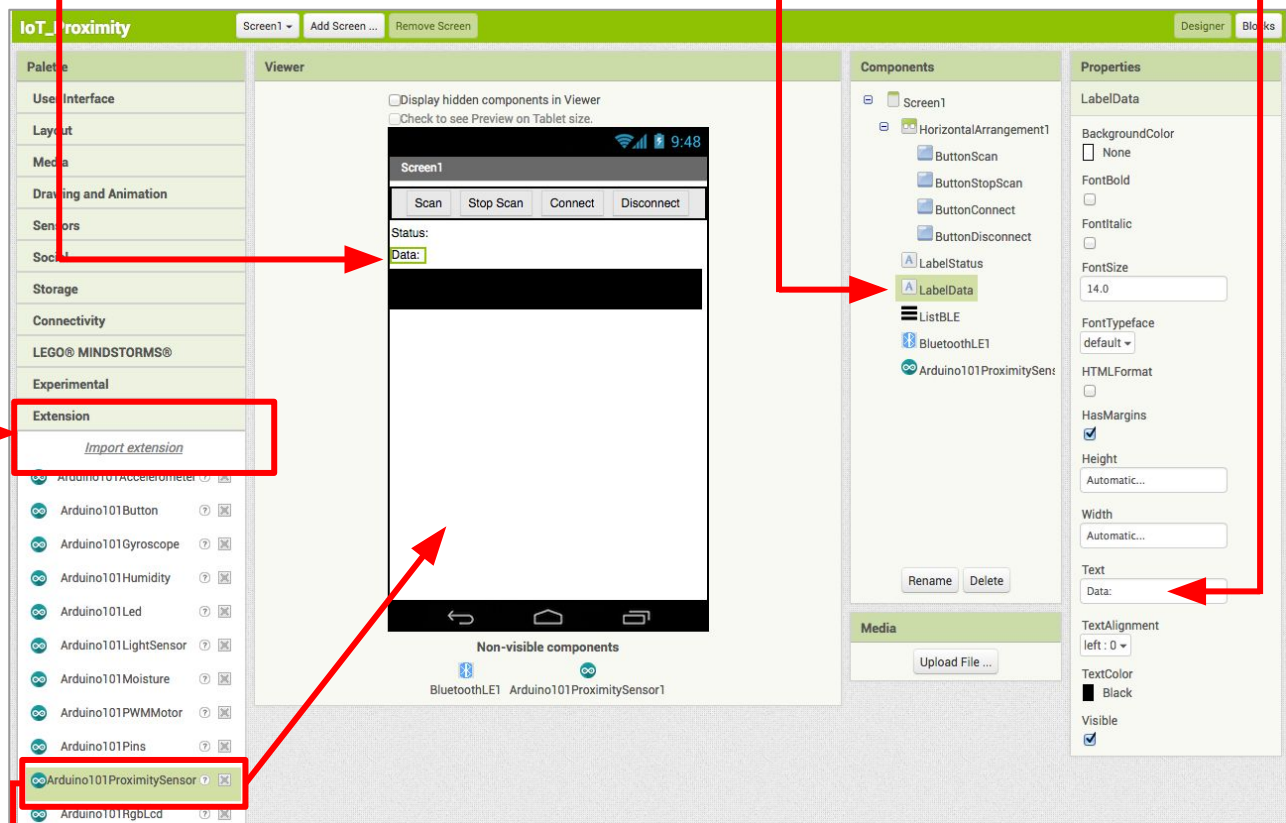


```
AIM-for-Things-Arduino101 | Arduino 1.8.1
AIM-for-Things-Arduino101 $ Accelerometer.hh Button.hh Camera.hh Console.hh Fingerprint.hh yros
1 #define NAME "App Inventor" // no more than 11 characters
2 #define DEBUGGING ENABLED
3
4 #define ACCELEROMETER DISABLED
5 #define BUTTON DISABLED
6 #define CAMERA DISABLED
7 #define CONSOLE DISABLED
8 #define FINGERPRINT DISABLED
9 #define GYROSCOPE DISABLED
10 #define LED DISABLED
11 #define LIGHT_SENSOR DISABLED
12 #define MOISTURE_SENSOR DISABLED
13 #define PINS DISABLED
14 #define PROXIMITY ENABLED
15 #define PWM DISABLED
16 #define RGBLCD DISABLED
17 #define SERVO DISABLED
18 #define SOUND_RECORDER DISABLED
19 #define TEMPERATURE DISABLED
20
21 // frequency to read sensor values in µs
22 const unsigned long SENSOR_UPDATE_FREQ = 50000;
23 const unsigned long IMU_READ_FREQ = 5000;
24 const double IMU_FILTER_ALPHA = 0.5; //Alpha for accelerometer low pass filter
25
26 unsigned long nextSensorUpdate;
27 unsigned long nextIMURead;
28 double dt;
29
30 const uint8_t BITS[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
31 const uint8_t MASK[8] = { 0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F };
32
33 #include "common.h"
```

Next, you should complete the [App Inventor + IoT Basic Connection](#) tutorial to make a basic connection to the Arduino device. If you prefer, you can download the completed .aia file [here](#).

The remaining steps all build off of the the starter code for Basic Connection tutorial and .aia:

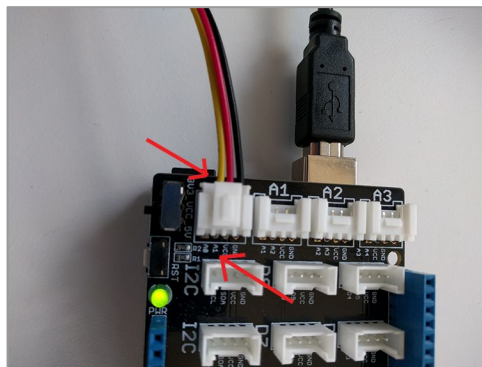
- Drag a **Label** from the User Interface Palette and drop it between **LabelStatus** and **ListBLE**
 - Rename the **Label** "LabelData".
 - Change its text to "Data: ".



- Download the [Arduino101](#) extension pack to your computer.
- In the Palette, click on Extension at the bottom and then on "Import extension" and then "Choose File". Find the file on your computer and upload it.
- Add the **Arduino101ProximitySensor** extension to your app by dragging it onto the Viewer.

Next, we need to let App Inventor know which pin on the Grove board the proximity sensor is connected to.

- Click on **Arduinio101ProximitySensor1** in the Components pane.
- In the Properties pane under **Pin**, write in the analog pin that matches the one the proximity sensor is plugged into on the Grove board, (in this case A0).
 - *Note: You only need to put the number (0), not the letter "A".*
 - *Another note: If your sensor wires look like the picture below, note that the yellow wire goes to the A1 pin, not A0 as you would think. Therefore, you should type the number 1 into the **Pin** property setting, not 0.*



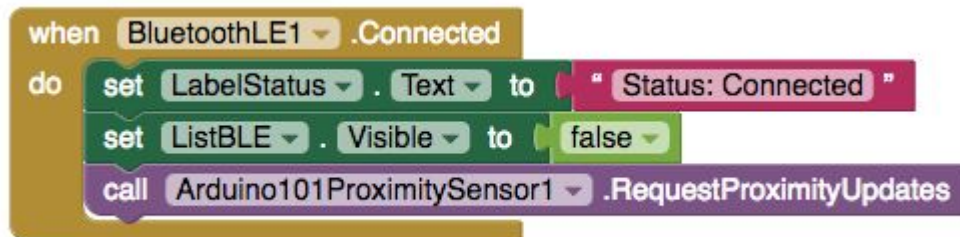
The screenshot displays the App Inventor web interface with three main panels: Viewer, Components, and Properties.

- Viewer Panel:** Shows a mobile app preview. At the top, there are buttons for "Scan", "Stop Scan", "Connect", and "Disconnect". Below these are labels for "Status:" and "Data:". A yellow tooltip labeled "Screen1" points to the "Disconnect" button. At the bottom, under "Non-visible components", both "BluetoothLE1" and "Arduinio101ProximitySensor1" are listed.
- Components Panel:** Lists the components on the screen. Under "HorizontalArrangement1", there are buttons for "ButtonScan", "ButtonStopScan", "ButtonConnect", and "ButtonDisconnect". Below these are "LabelStatus", "LabelData", "ListBLE", "BluetoothLE1", and "Arduinio101ProximitySensor1" (highlighted in green).
- Properties Panel:** Shows the properties for "Arduinio101ProximitySensor1". The "BluetoothDevice" property is set to "BluetoothLE1..." and the "Pin" property is set to "2". Red arrows from the text above point to these two properties.

Now switch to the Blocks Editor view

First, we want to request data updates when the sensor value on the Arduino changes.

- from the Arduino101ProximitySensor1 drawer in the Blocks pane, add **call Arduino101ProximitySensor1.RequestProximityUpdates** to the existing **when BluetoothLE1.Connected** block you made in the Basic Connection tutorial.



Next, we need to store the data we receive from the sensor. From the Variables drawer in the docs pane, drag an **initialize global name to** block and name it "Proximity". From the Math drawer add a number block and set it to "0". We'll use this to keep track of the sensor value.



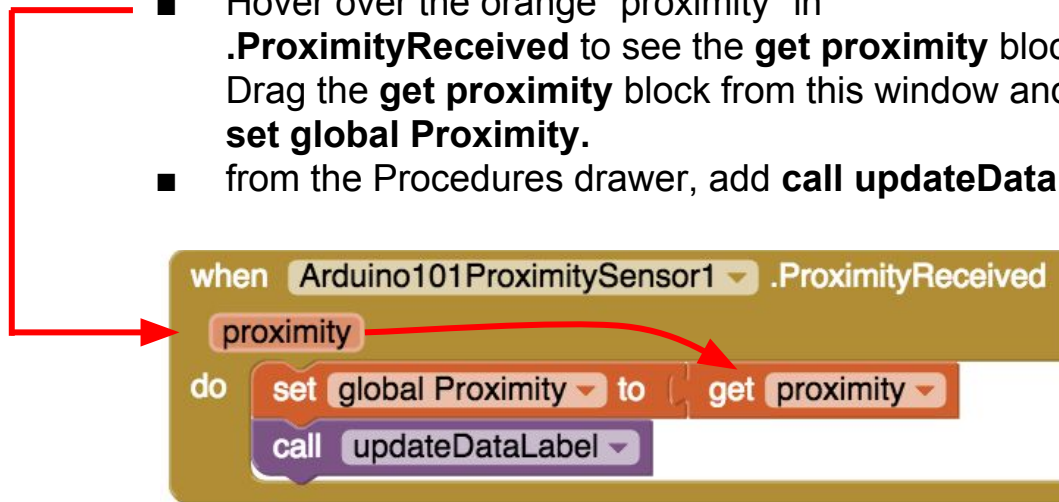
Let's make a new procedure to display the current readings in the **LabelData** when we get new data. You can create a procedure by dragging out a purple procedure block from the Procedures drawer in the Blocks pane. Let's rename it **updateDataLabel**.

- from LabelData in the Blocks pane, add **set LabelData.Text to**.
- from the Text drawer connect a **join** block.
 - From the Text drawer, connect a text block and type **"Distance: "**.
 - From the Variables drawer connect a **get global Proximity**.



Finally, we need to call the procedure when this data is received.

- From Arduino101ProximitySensor1 drag **when Arduino101ProximitySensor1.ProximitySensorDataReceived**.
 - from the Variables drawer, add **set global light**.
 - Hover over the orange "proximity" in **.ProximityReceived** to see the **get proximity** block. Drag the **get proximity** block from this window and snap to **set global Proximity**.
 - from the Procedures drawer, add **call updateDataLabel**.



Your app should now be working! Connect your Android device using the MIT AI2 Companion (if you haven't already). Test it out by moving your hand closer and farther away from the sensor. If it is working, you should see the data label change.

