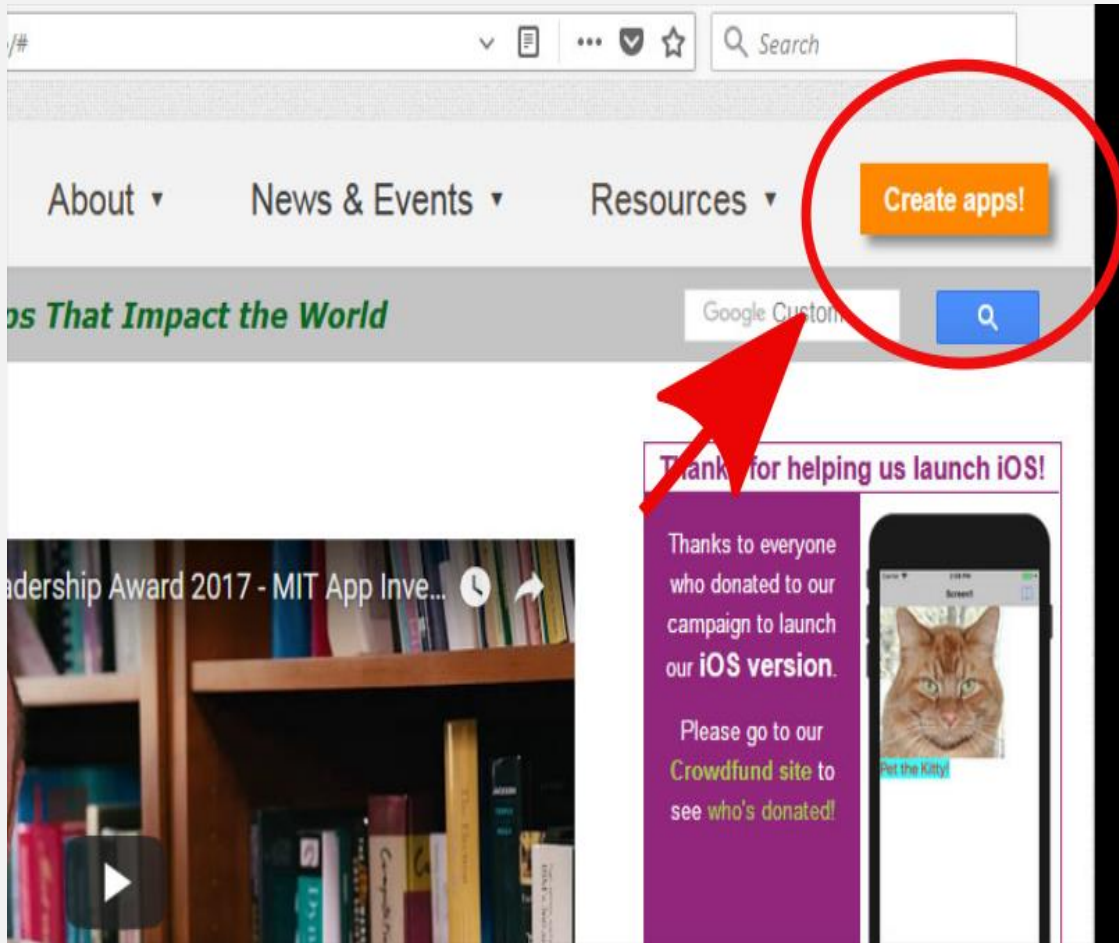


# **MIT APP INVENTOR - Interfacing Cretile**

# INTRODUCTION

- App Inventor is a cloud-based tool, by which you can build android and iOS applications using web browser.
- MIT App Inventor website offers all the support you'll need to learn how to build your own apps.
- *Let's build an app!*



*www.ai2.appinventor.mit.edu.*

By clicking “Create Apps!” button from any page on this website you can get MIT app login page.



## Sign in

with your Google Account

Email or phone

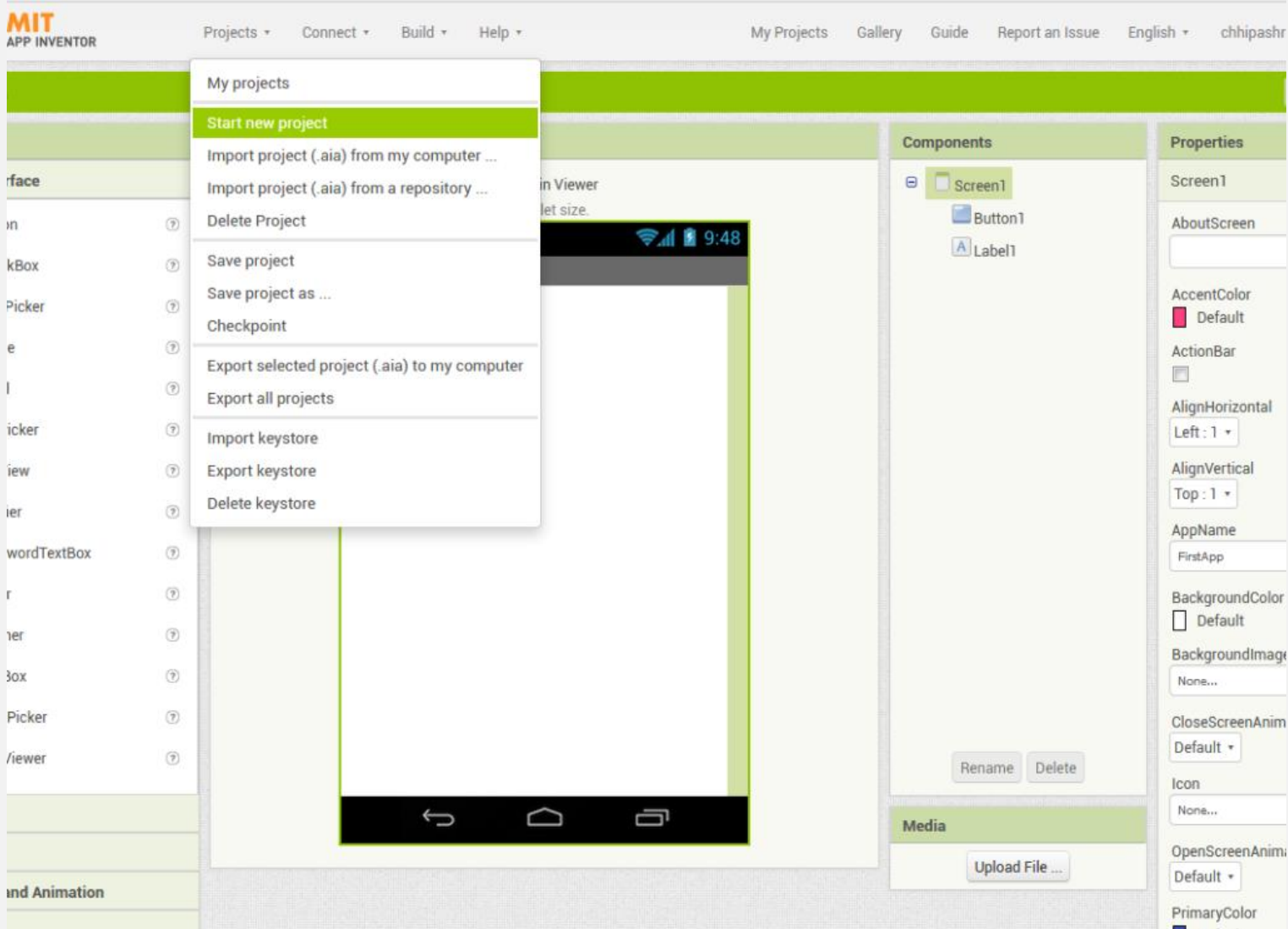
---

[Forgot email?](#)

[More options](#)

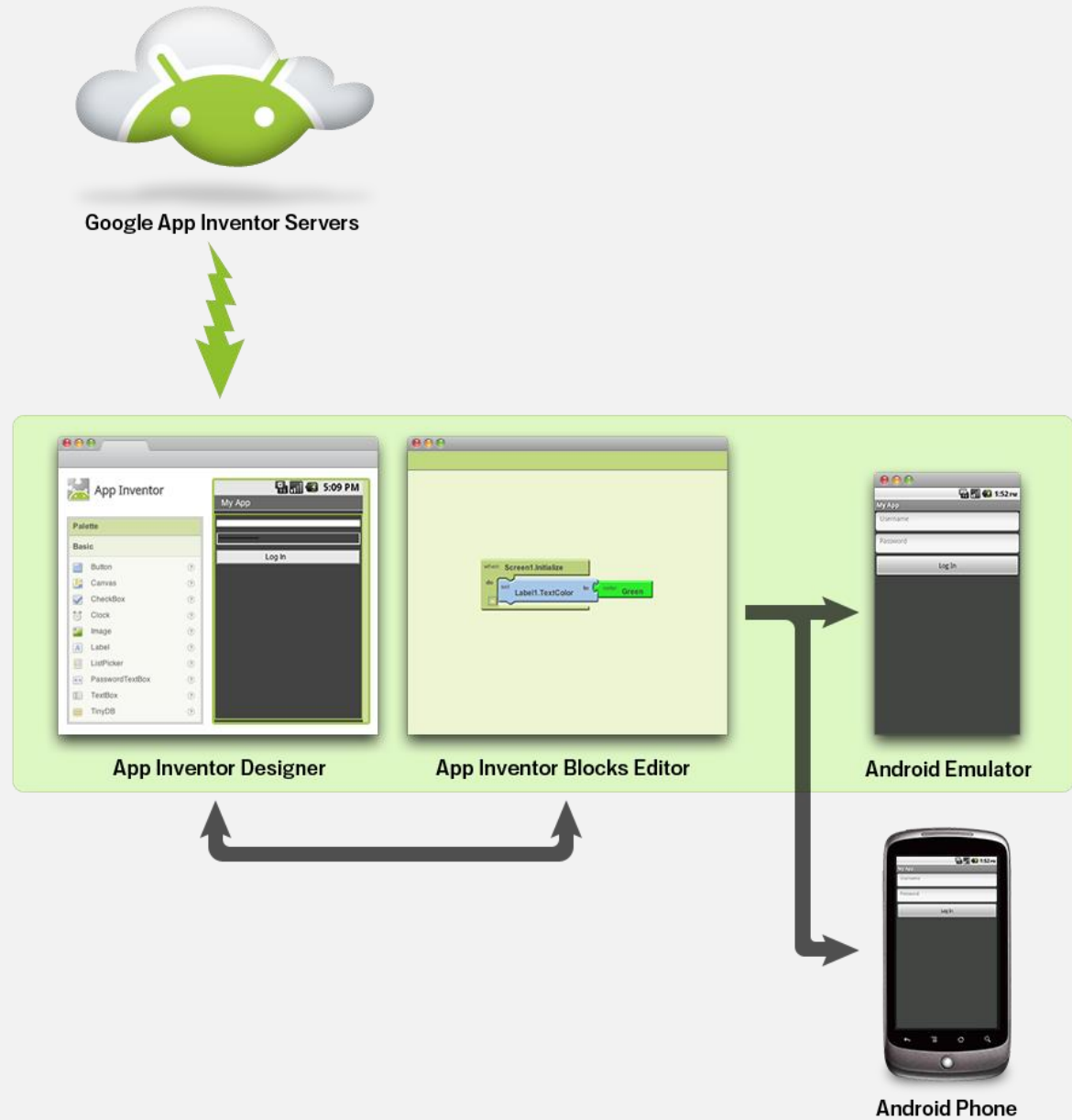
NEXT

Log in to App Inventor using an existing Google account.



To start making your application, click on the Projects menu on the top left of the screen and select the “Start new project” and then give the name of your project.

# How it all works?



You build apps by working with:

- **Designer** - select the components for your app.
- **Blocks Editor** - assemble program blocks that specify how the components should behave. You assemble programs visually, fitting pieces together like pieces of a puzzle.

- The app appears on the phone step-by-step as you add pieces to it, so you can test your work as you build.
- Once the app is complete, we can package the app and produce a stand-alone application to install.
- The App Inventor development environment is supported for Mac OS X, GNU/Linux, and Windows operating systems and Android.



The screenshot displays the Android Studio IDE interface during a component rename operation. The central 'Viewer' pane shows a mobile application preview with a button labeled 'Text for Button1' and a label 'Text for Label1'. A 'Rename Component' dialog box is overlaid on the preview, with 'Old name' set to 'Button1' and 'New name' set to 'Text'. The 'Components' pane on the right shows a tree view with 'Screen1' containing 'Label1' and 'Button1'. The 'Properties' pane on the far right shows the properties for the selected 'Button1' component, including 'Text' set to 'Text for Button1'. The 'Palette' on the left lists various UI components like Button, CheckBox, DatePicker, etc.

- The designer panel appears with five palette:

- **USER INTERFACE PALETTE**

We choose things for the user interface things like Buttons, Images, Text boxes to the palette where we can layout the “user interfaces” of our app.

- **VIEWER PALETTE**

In viewer palette we will be able to arrange the outlook of our app.

- **COMPONENTS PALETTE**

Component palette displays all the components placed in the app in an order

- **MEDIA PALETTE**

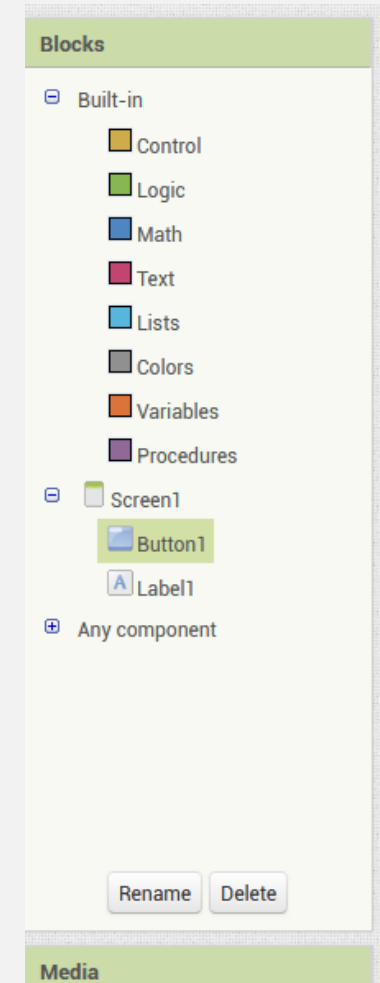
Media palette is used to insert any external media to the app.

- **PROPERTIES PALETTE**

We can change the properties of the components like height, width, text, color to the properties panel.

## BLOCK EDITOR

- The Blocks Editor is where you program the behavior of your app.
- There are built-in blocks that handle things like math, logic, and text with each component you have added.

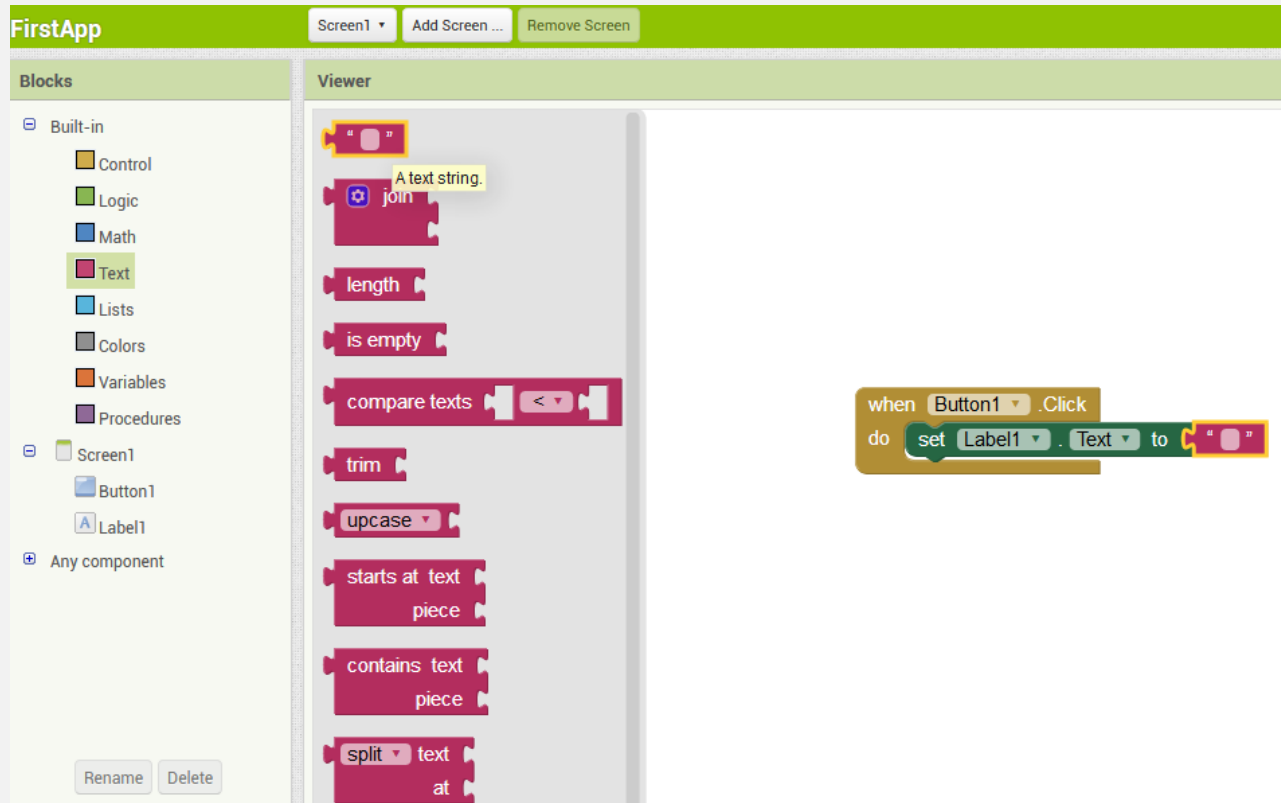


- Click on the Button1 drawer.
- Click and hold the “when Button1.Click do” block.
- Drag it over to the workspace and drop it there.
- This is the block that will handle what happens when the button on your app is clicked.
- It is called an “Event Handler”.



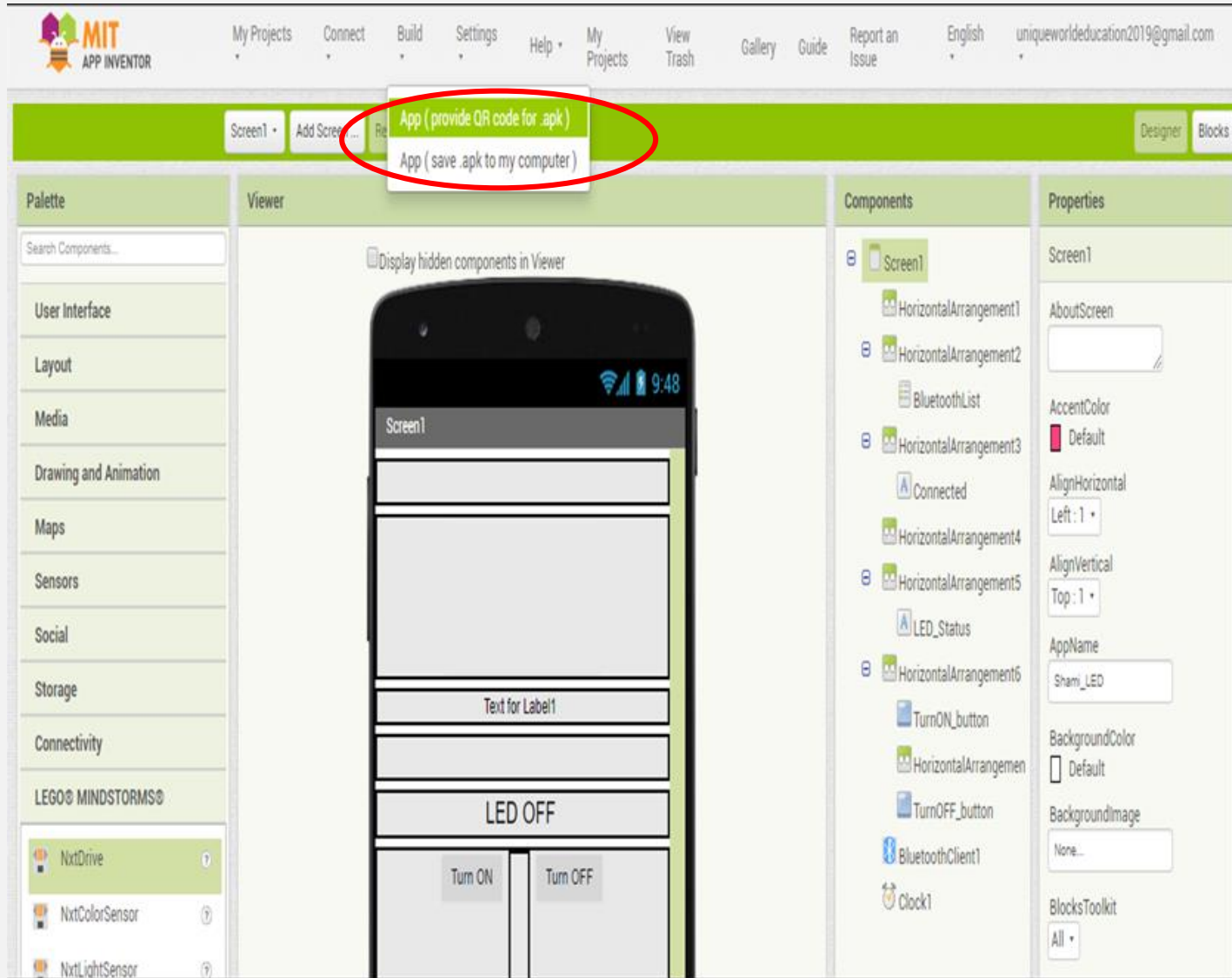
The image shows a visual programming environment with two main panels: 'Blocks' on the left and 'Viewer' on the right. The 'Blocks' panel is organized into categories: 'Built-in' (Control, Logic, Math, Text, Lists, Colors, Variables, Procedures), 'Screen1' (Button1, Label1), and 'Any component'. The 'Label1' block is highlighted in the 'Screen1' category. Below the 'Blocks' panel are 'Rename' and 'Delete' buttons, and a 'Media' section with an 'Upload File ...' button. The 'Viewer' panel displays a vertical stack of blocks for 'Label1', including 'BackgroundColor', 'FontSize', 'HasMargins', 'Height', 'HeightPercent', 'Text', 'TextColor', and 'Visible'. A 'when Button1 Click' event block is shown in the center, containing a 'set Label1 Text to' block. The 'set Label1 Text to' block is highlighted with a yellow border.

- Now click on the Label1.
- Click and hold the “set Label1 text to” block.
- Drag it inside the button click, it will run when the button is pressed.



- At last, click on the text drawer, drag out a text block and plug it into the socket labelled to and write anything that you want to display.
- Click on the text block and write anything.

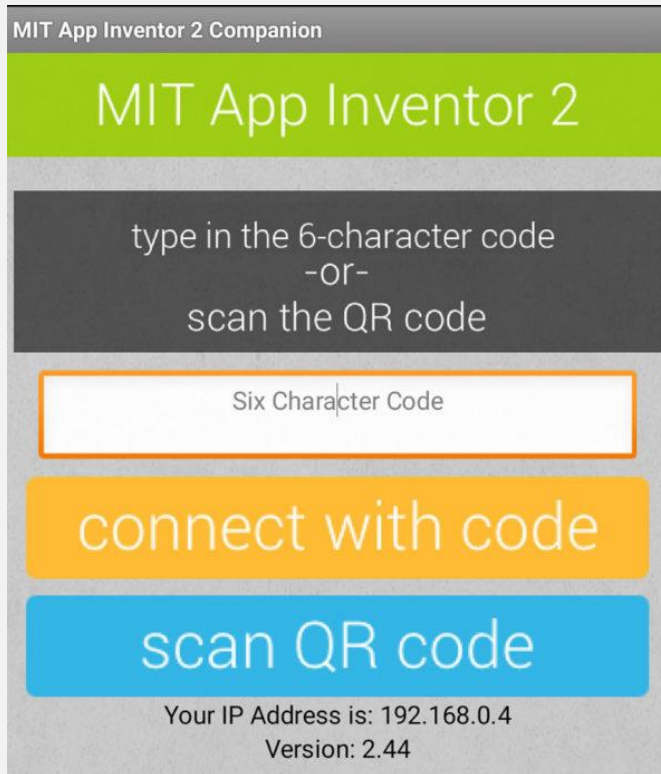




- Once we program the behaviour of our app we can build our app.

We can choose to provide QR code for .apk file of the app which can be installed in our android phone.





- Download and install the MIT App Inventor 2 Companion on the phone.

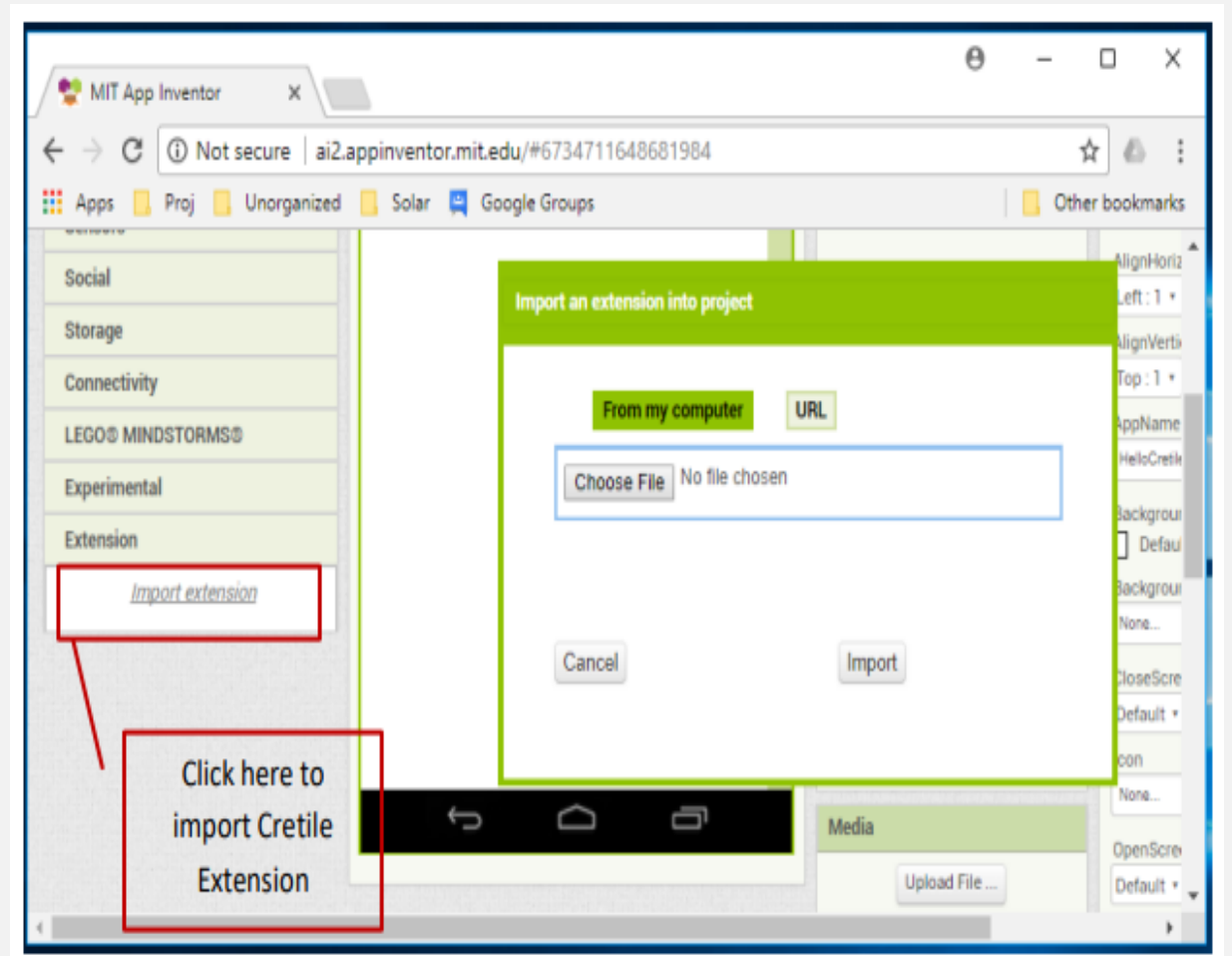
Open the QR Code from the App inventor website.

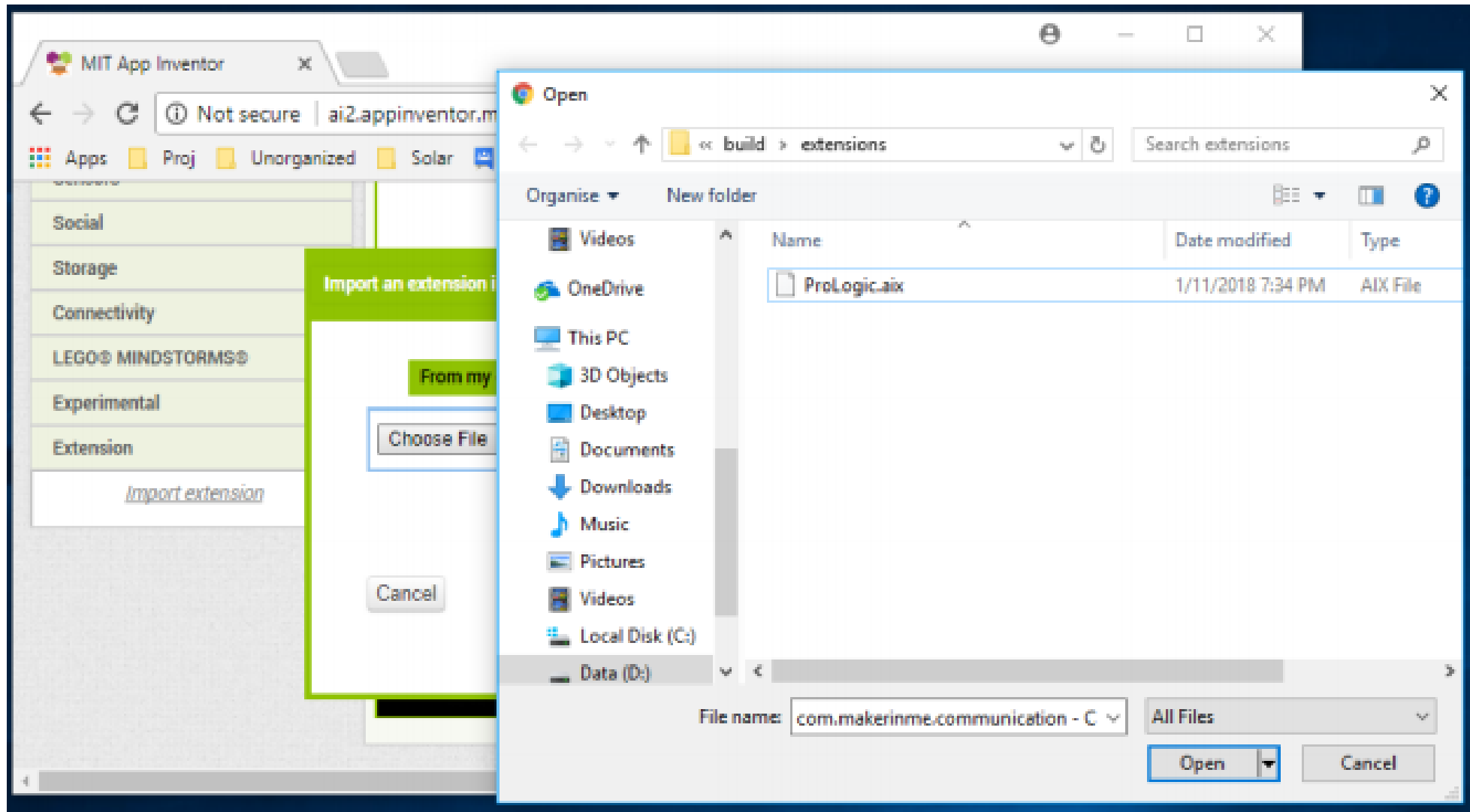
# CRETILE WITH MIT APP INVENTOR

- Cretile provides an App Inventor extension for communicating with ProLogic using Android phone's USB port.
- This communication works **only** when phone has USB-OTG support.
- ProLogic local INputs and remote INputs are sent to Android phone, which can read these values.
- Android phone can send values of ProLogic local OUTputs and remote OUTputs over USB.

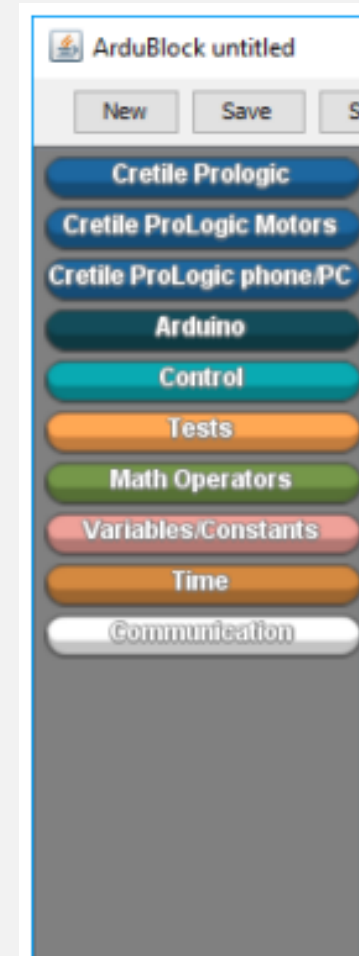
- In ProLogic program you can choose to write values sent by Android phone on respective OUTputs.
- There are blocks provided in App Inventor as well as Ardublock for this communication.

- Download the Cretile folder and unzip it to access extension using this link:
- <https://www.cretile.com/blog/download-software-and-documentation>

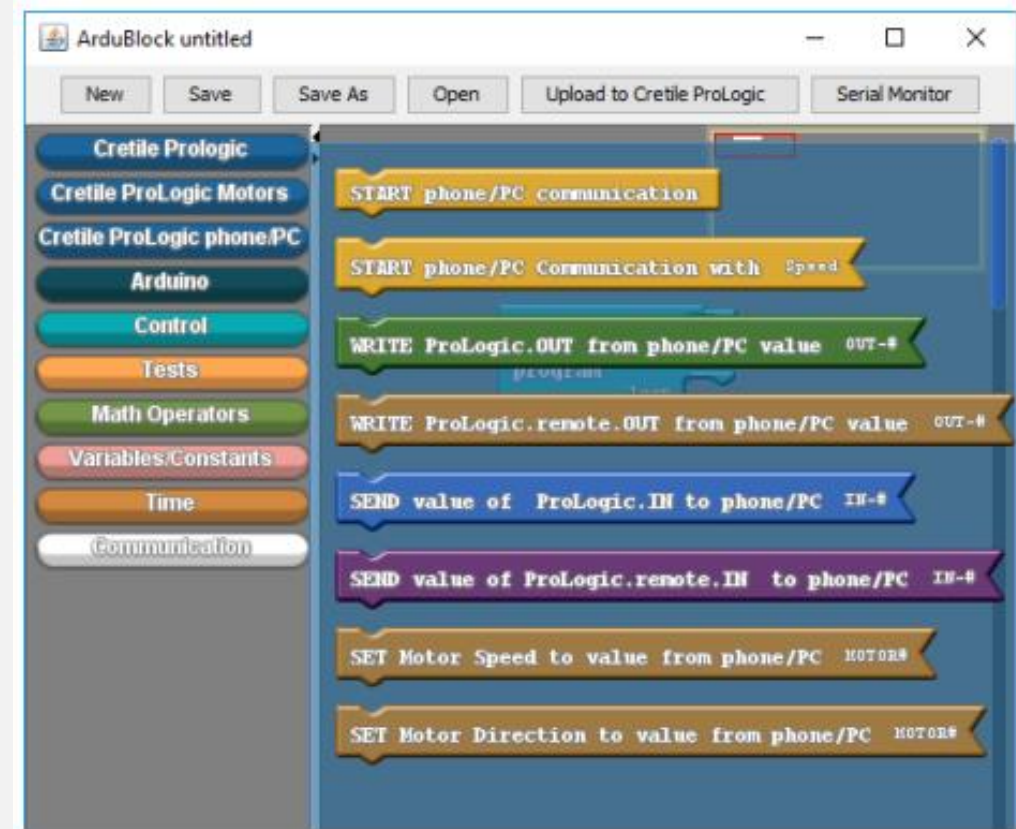




- After we designing our app and programming it we have to develop ProLogic program to work with the Android application
- ArduBlock for Cretile has block drawer named Cretile ProLogic phone/PC.

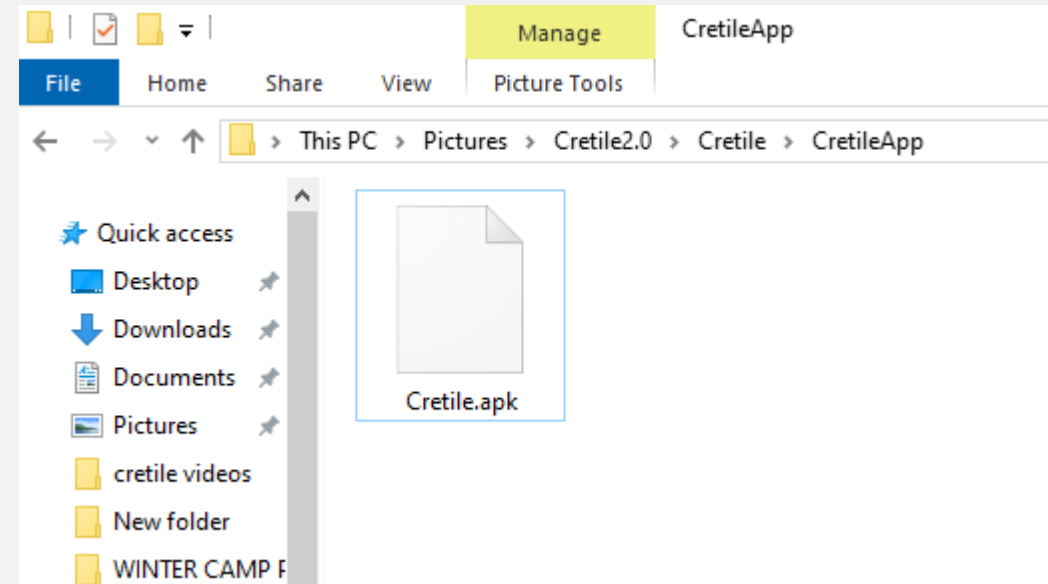


- This block drawer contains blocks to send data to phone/PC and use data from Android phone/PC.
- Open Arduino->Ardublock for Cretille as explained.
- There is Block drawer named Cretille ProLogic Phone/PC which provides all the blocks needed for communication between phone/PC and ProLogic (using USB cable).



# Setting up your Android phone

- We need to install Android application provided by Cretile to enable communication between ProLogic and Android application developed.
- We don't have to open the Cretile application (Cretile.apk) as it automatically runs in background when needed by Application developed by you in App Inventor.





# Connecting Android with ProLogic using USB-OTG

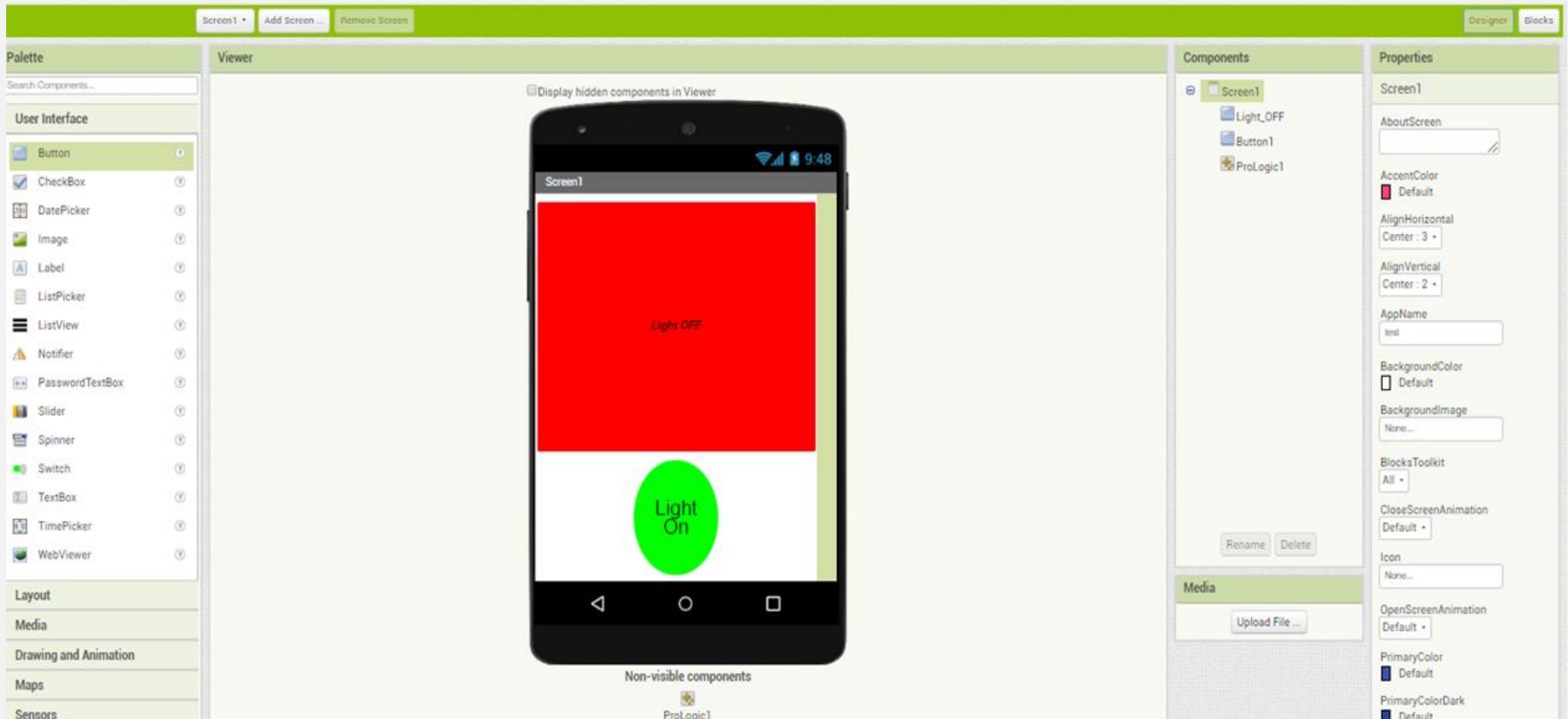
- Start application created in MIT App Inventor in android phone and connect the phone with ProLogic which is programmed using Ardublock.
- We will need USB-OTG adapter/cable to connect ProLogic to our USB-OTG capable android phone. The setup is shown in the image.



- Once phone is connected to ProLogic it will be recognized by the App and permission request for accessing USB device will appear.
- Grant the permission and communication with ProLogic will be established.

# Cretile Interfacing - 01

- Create an app for Cretile to switch light on/off.



Screen1 • Add Screen ... Remove Screen Designer Blocks

Blocks

- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists
  - Colors
  - Variables
  - Procedures
- Screen1
  - Light\_OFF
  - Button1
  - ProLogic1
- Any component

Rename Delete

Media

Upload File ...

Viewer

The viewer displays three event-driven logic blocks:

- when Light\_OFF .Click**  
do call ProLogic1 .WRITE\_OUT  
output\_number 1  
value 0
- when Screen1 .Initialize**  
do call ProLogic1 .Start
- when Button1 .Click**  
do call ProLogic1 .WRITE\_OUT  
output\_number 1  
value 99

Show Warnings

Designer tools: eye, zoom in (+), zoom out (-), trash.

Privacy Policy and Terms of Use

The screenshot shows the ArduBlock software interface. The window title is "ArduBlock untitled \*". The menu bar includes "New", "Save", "Save As", "Open", "Upload to Cretile ProLogic", and "Serial Monitor". The left sidebar contains a list of categories: "Cretile ProLogic", "Cretile ProLogic Motors", "Cretile ProLogic phone/PC", "Arduino", "Control", "Tests", "Math Operators", "Variables/Constants", "Time", and "Communication". The main workspace displays a code block with the following structure:

```
program
  setup
  loop
    WRITE ProLogic.OUT from phone/PC value OUT-# 1
    delay MILLIS milliseconds 10
```

The "WRITE ProLogic.OUT from phone/PC value OUT-# 1" block has a dropdown menu open, showing "1" as the selected value. The "delay MILLIS milliseconds 10" block has a dropdown menu open, showing "10" as the selected value. A "Main" label is visible in the top right corner of the workspace.

# Cretile Interfacing - 02

- Create an app for Cretile dimmer by providing Analog values.

Screen1 • Add Screen ... Remove Screen Designer Blocks

Search Components...

**User Interface**

- Button
- CheckBox
- DatePicker
- Image
- Label
- ListPicker
- ListView
- Notifier
- PasswordTextBox
- Slider
- Spinner
- Switch
- TextBox
- TimePicker
- WebView

Layout

Media

Drawing and Animation

Maps

Sensors

Viewer

Display hidden components in Viewer

Screen1

ProLogic Input 1  
00

Non-visible components

- ProLogic1

**Components**

- Screen1
  - textN1
  - valueN1
  - ProLogic1

Rename Delete

**Media**

Upload File ...

**Properties**

Screen1

AboutScreen

AccentColor  
Default

AlignHorizontal  
Center : 3

AlignVertical  
Center : 2

AppName  
test

BackgroundColor  
Default

BackgroundImage  
None...

BlocksToolkit  
All

CloseScreenAnimation  
Default

Icon  
None...

OpenScreenAnimation  
Default

PrimaryColor  
Default

PrimaryColorDark  
Default



Screen 1 • Add Screen ... Remove Screen Designer Blocks

**Blocks**


- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists
  - Colors
  - Variables
  - Procedures
- Screen1
  - textN1
  - valueN1
  - ProLogic1
- Any component

Rename Delete

**Media**

Upload File ...

**Viewer**

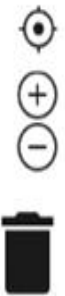


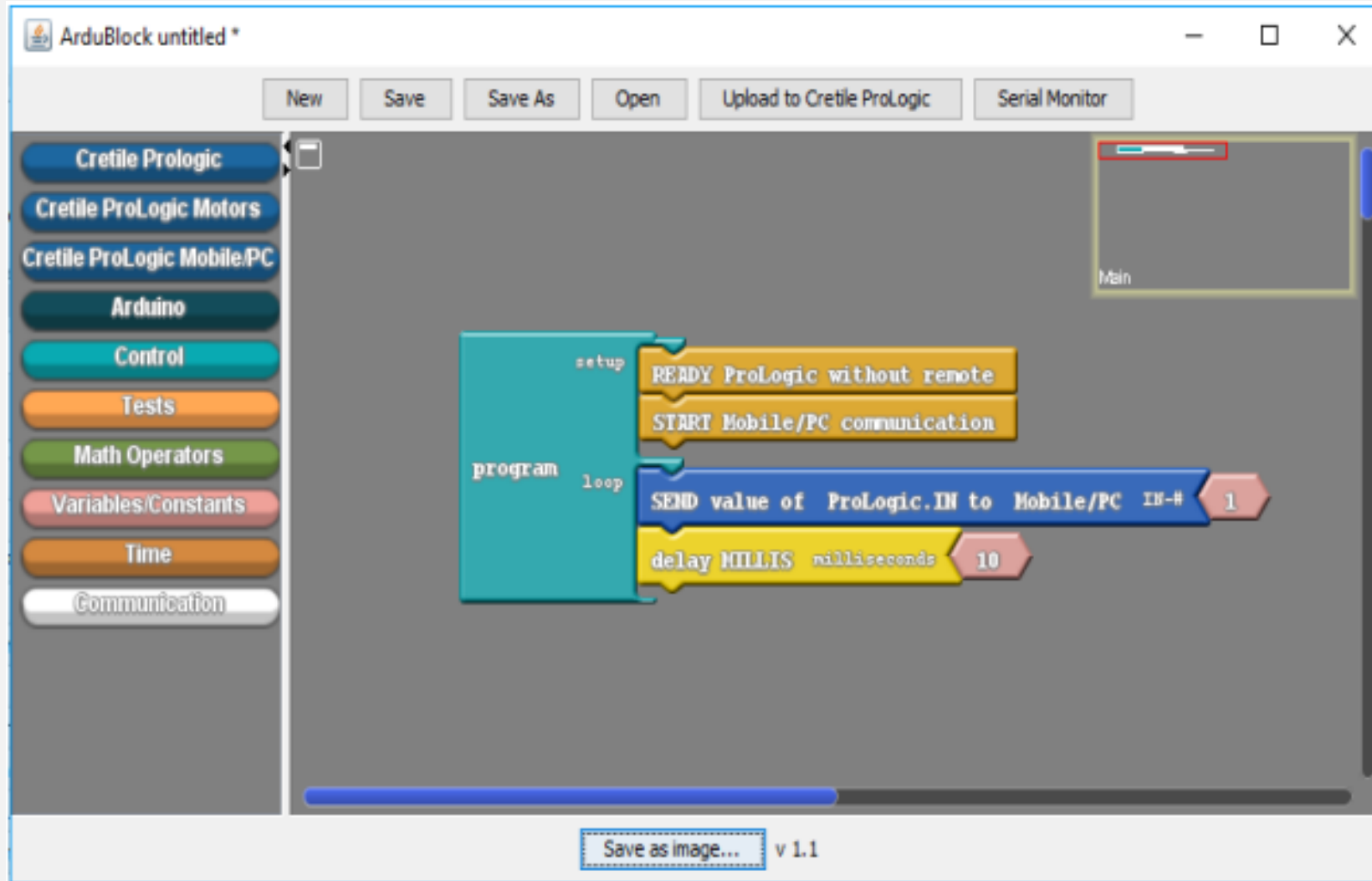
```
when Screen1 Initialize
do call ProLogic1 Start

when ProLogic1 InputValuesReceived
do set valueN1 Text to call ProLogic1 READ_IN
input_number 1
```

Show Warnings

0 0





**THANK YOU!**