



**CCE60220**

# Perangkat Bergerak (TKOM)



Fakultas Ilmu Komputer Universitas Brawijaya

# Agenda Perkuliahan

1. ~~Intro dan overview perkuliahan~~
2. ~~Sejarah dan perkembangan teknologi perangkat bergerak~~
3. ~~Komponen perangkat keras dan perangkat lunak~~
4. ~~Pengenalan dan instalasi android studio serta aplikasi sederhana~~
5. ~~Intent dan passing data pada Android Studio~~
6. ~~Android Studio: Sensor reading~~
7. ~~Android Studio: Storage & shared preference~~
8. =====**UTS**
9. ~~Pengenalan dan aplikasi sederhana dengan MIT AppInventor~~
10. ~~Appinventor: variable, conditional, tinyDB, file~~
11. ~~appinventor: sensor reading & persiapan project~~
12. ~~Appinventor: Akuisisi gambar dan suara~~
13. Appinventor: komunikasi bluetooth, Wifi to control device (http get)
14. Appinventor: basic animation
15. **Presentasi kelompok**
16. =====**UAS**

## Using Bluetooth









# Bluetooth



Bluetooth sender  
Bluetooth Server side ✓



Bluetooth receiver  
Bluetooth client side ✓

Palette		
	User Interface	
	Layout	
	Media	
	Drawing and Animation	
	Sensors	
	Social	
	Storage	
	Connectivity	
	ActivityStarter	
	BluetoothClient	
	BluetoothServer	
	Web	
	LEGO® MINDSTORMS®	
	Experimental	
	Extension	

## Properties

- Available: boolean, Tell whether Bluetooth is available on the Android device.
- CharacterEncoding: text, The character encoding to use when sending and receiving text.
- DelimiterByte: number, The delimiter byte to use when passing a negative number for the numberOfBytes parameter when calling ReceiveText, ReceiveSignedBytes, or ReceiveUnsignedBytes.
- Enabled: boolean, Tell whether Bluetooth is enabled.
- HighByteFirst: boolean, Whether 2 and 4 byte numbers should be sent and received with the high (or most significant) byte first. Check the documentation for the device with which your app will be communicating for the appropriate setting. This is also known as big-endian.
- IsAccepting: boolean, Tell whether this BluetoothServer component is accepting an incoming connection.
- IsConnected: boolean, Tell whether a Bluetooth connection has been made.

## Events


- ConnectionAccepted(): Indicates that a bluetooth connection has been accepted.

## Methods

- `AcceptConnection(text serviceName)`: Accept an incoming connection with the Serial Port Profile (SPP).
- `AcceptConnectionWithUUID(text serviceName, text uuid)`: Accept an incoming connection with a specific UUID.
- `number BytesAvailableToReceive()`: Returns an estimate of the number of bytes that can be received without blocking
- `Disconnect()`: Disconnect from the connected Bluetooth device.
- `number ReceiveSigned1ByteNumber()`: Receive a signed 1-byte number from the connected Bluetooth device.
- `number ReceiveSigned2ByteNumber()`: Receive a signed 2-byte number from the connected Bluetooth device.
- `number ReceiveSigned4ByteNumber()`: Receive a signed 4-byte number from the connected Bluetooth device.
- `list ReceiveSignedBytes(number numberOfBytes)`: Receive multiple signed byte values from the connected Bluetooth device. If `numberOfBytes` is less than 0, read until a delimiter byte value is received.
- `text ReceiveText(number numberOfBytes)`: Receive text from the connected Bluetooth device. If `numberOfBytes` is less than 0, read until a delimiter byte value is received.
- `number ReceiveUnsigned1ByteNumber()`: Receive an unsigned 1-byte number from the connected Bluetooth device.
- `number ReceiveUnsigned2ByteNumber()`: Receive a unsigned 2-byte number from the connected Bluetooth device.
- `number ReceiveUnsigned4ByteNumber()`: Receive a unsigned 4-byte number from the connected Bluetooth device.
- `list ReceiveUnsignedBytes(number numberOfBytes)`: Receive multiple unsigned byte values from the connected Bluetooth device. If `numberOfBytes` is less than 0, read until a delimiter byte value is received.
- `Send1ByteNumber(text number)`: Send a 1-byte number to the connected Bluetooth device.
- `Send2ByteNumber(text number)`: Send a 2-byte number to the connected Bluetooth device.
- `Send4ByteNumber(text number)`: Send a 4-byte number to the connected Bluetooth device.
- `SendBytes(list list)`: Send a list of byte values to the connected Bluetooth device.
- `SendText(text text)`: Send text to the connected Bluetooth device.
- `StopAccepting()`: Stop accepting an incoming connection.

# BluetoothServer

Display hidden components in Viewer  
 Check to see Preview on Tablet size.



Screen1

Ready to Accept Connection

DISCONNECT

DatePicker

Send data

Non-visible components

- BluetoothServer1
- Notifier1

Screen1

- VerticalArrangement3
  - Button2
  - Button3
- VerticalArrangement1
  - DatePicker1
- VerticalArrangement2
  - Button1
  - BluetoothServer1
  - Notifier1

Rename Delete

Media

Upload File ...

# BluetoothServer

```
when Screen1.Initialize
do
  if not BluetoothServer1.Enabled
  then
    call Notifier1.ShowAlert
      notice "please enable bluetooth first!"

when Button2.Click
do
  call BluetoothServer1.AcceptConnection
    serviceName ""

when BluetoothServer1.ConnectionAccepted
do
  set Button2.Text to "Connection accepted"
  set Button2.BackgroundColor to #00FF00

when Button3.Click
do
  call BluetoothServer1.Disconnect
  set Button2.Text to "Ready to Accept Connection"
  set Button2.BackgroundColor to #CCCCCC

when Button1.Click
do
  if BluetoothServer1.IsConnected
  then
    call BluetoothServer1.SendText
      text join DatePicker1.Day
        " / "
        DatePicker1.Month
        " / "
        DatePicker1.Year
```



## Properties

- `AddressesAndNames`: The addresses and names of paired Bluetooth devices
- `Available`: Whether Bluetooth is available on the device
- `CharacterEncoding`
- `DelimiterByte`
- `Enabled`: Whether Bluetooth is enabled
- `HighByteFirst`
- `IsConnected`
- `Secure`: Whether to invoke SSP (Simple Secure Pairing), which is supported on devices with Bluetooth v2.1 or higher. When working with embedded Bluetooth devices, this property may need to be set to `False`. For Android 2.0-2.2, this property setting will be ignored.


## Events

## Methods

- `number BytesAvailableToReceive()`: Returns an estimate of the number of bytes that can be received without blocking
- `boolean Connect(text address)`: Connect to the Bluetooth device with the specified address and the Serial Port Profile (SPP). Returns true if the connection was successful.
- `boolean ConnectWithUUID(text address, text uuid)`: Connect to the Bluetooth device with the specified address and UUID. Returns true if the connection was successful.
- `Disconnect()`: Disconnect from the connected Bluetooth device.
- `boolean IsDevicePaired(text address)`: Checks whether the Bluetooth device with the specified address is paired.
- `number ReceiveSigned1ByteNumber()`: Receive a signed 1-byte number from the connected Bluetooth device.
- `number ReceiveSigned2ByteNumber()`: Receive a signed 2-byte number from the connected Bluetooth device.
- `number ReceiveSigned4ByteNumber()`: Receive a signed 4-byte number from the connected Bluetooth device.
- `list ReceiveSignedBytes(number numberOfBytes)`: Receive multiple signed byte values from the connected Bluetooth device. If `numberOfBytes` is less than 0, read until a delimiter byte value is received.
- `text ReceiveText(number numberOfBytes)`: Receive text from the connected Bluetooth device. If `numberOfBytes` is less than 0, read until a delimiter byte value is received.
- `number ReceiveUnsigned1ByteNumber()`: Receive an unsigned 1-byte number from the connected Bluetooth device.
- `number ReceiveUnsigned2ByteNumber()`: Receive a unsigned 2-byte number from the connected Bluetooth device.
- `number ReceiveUnsigned4ByteNumber()`: Receive a unsigned 4-byte number from the connected Bluetooth device.
- `list ReceiveUnsignedBytes(number numberOfBytes)`: Receive multiple unsigned byte values from the connected Bluetooth device. If `numberOfBytes` is less than 0, read until a delimiter byte value is received.
- `Send1ByteNumber(text number)`: Send a 1-byte number to the connected Bluetooth device.
- `Send2ByteNumber(text number)`: Send a 2-byte number to the connected Bluetooth device.
- `Send4ByteNumber(text number)`: Send a 4-byte number to the connected Bluetooth device.
- `SendBytes(list list)`: Send a list of byte values to the connected Bluetooth device.
- `SendText(text text)`: Send text to the connected Bluetooth device.

# Bluetooth Client

Display hidden components in Viewer  
 Check to see Preview on Tablet size.



Screen1

Choose BT device

DISCONNECT

Text for Label1

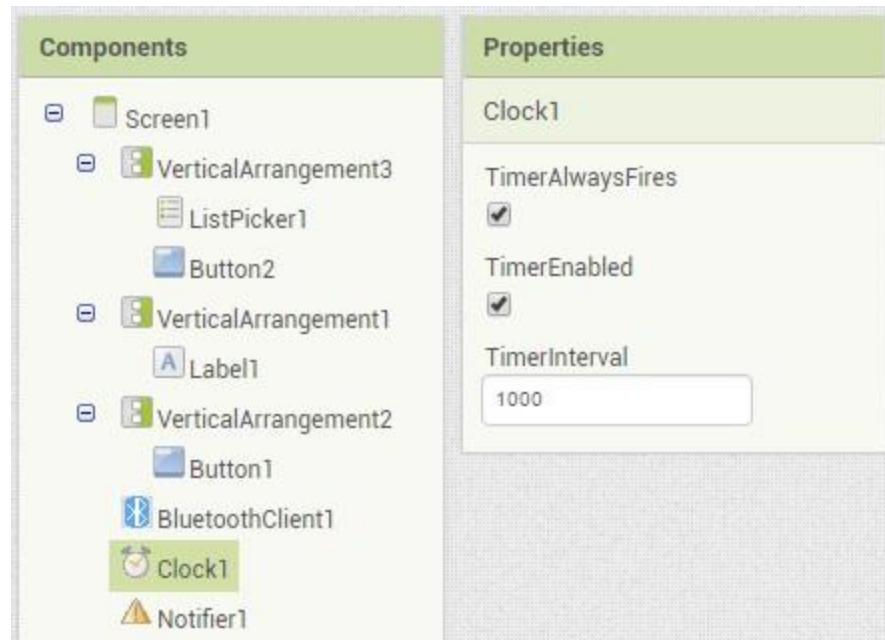
BluetoothClient1 Clock1 Notifier1

- Screen1
  - VerticalArrangement3
    - ListPicker1
    - Button2
  - VerticalArrangement1
    - Label1
  - VerticalArrangement2
    - Button1
    - BluetoothClient1
    - Clock1
    - Notifier1

Rename Delete

Media

Upload File ...



The screenshot displays a software development interface with two main panels: Components and Properties.

**Components Panel:**

- Screen1
  - VerticalArrangement3
    - ListPicker1
    - Button2
  - VerticalArrangement1
    - Label1
  - VerticalArrangement2
    - Button1
  - BluetoothClient1
  - Clock1** (highlighted)
  - Notifier1

**Properties Panel:**

The Properties panel is currently showing the properties for the selected **Clock1** component.

- Clock1
- TimerAlwaysFires:
- TimerEnabled:
- TimerInterval:

# Bluetooth Client

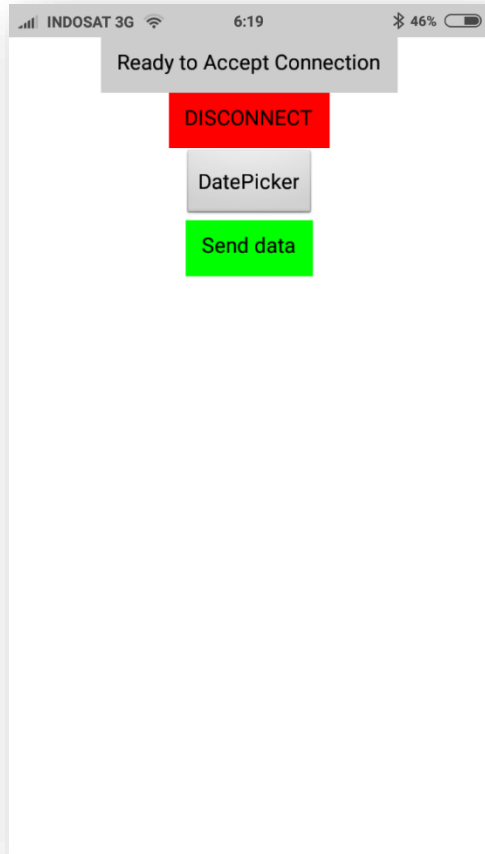
```
when ListPicker1 .BeforePicking
do
  if not BluetoothClient1 .Enabled
  then
    call Notifier1 .ShowAlert
      notice "Please enable bluetooth first "
    set ListPicker1 .Elements to BluetoothClient1 .AddressesAndNames

when ListPicker1 .AfterPicking
do
  set ListPicker1 .Selection to call BluetoothClient1 .Connect
    address ListPicker1 .Selection

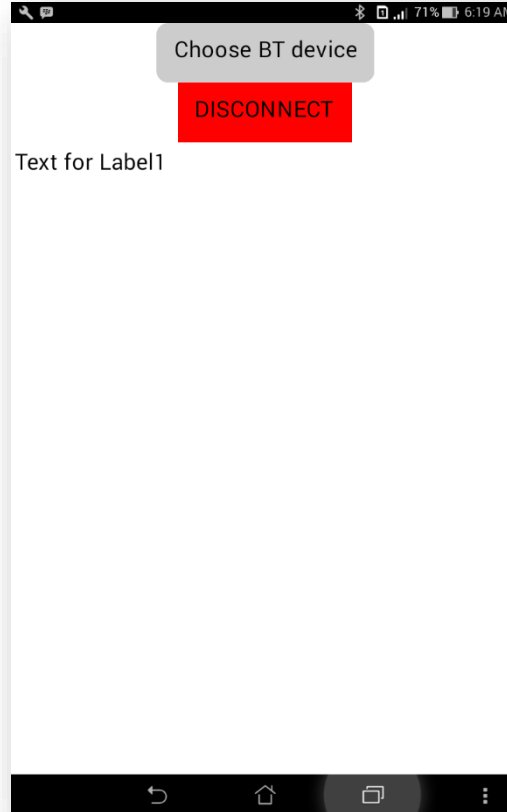
when Button2 .Click
do
  call BluetoothClient1 .Disconnect

when Clock1 .Timer
do
  if BluetoothClient1 .IsConnected
  then
    if call BluetoothClient1 .BytesAvailableToReceive > 0
    then
      set Label1 .Text to call BluetoothClient1 .ReceiveText
        numberOfBytes call BluetoothClient1 .BytesAvailableToReceive
```

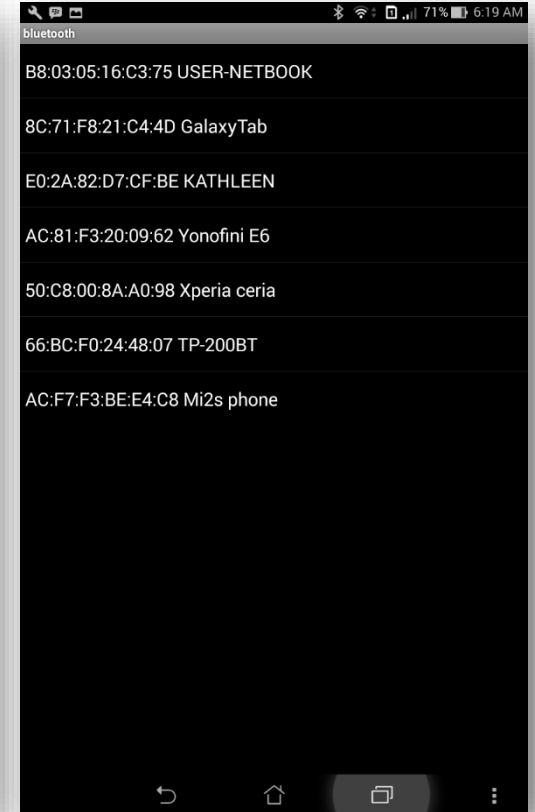
# Result

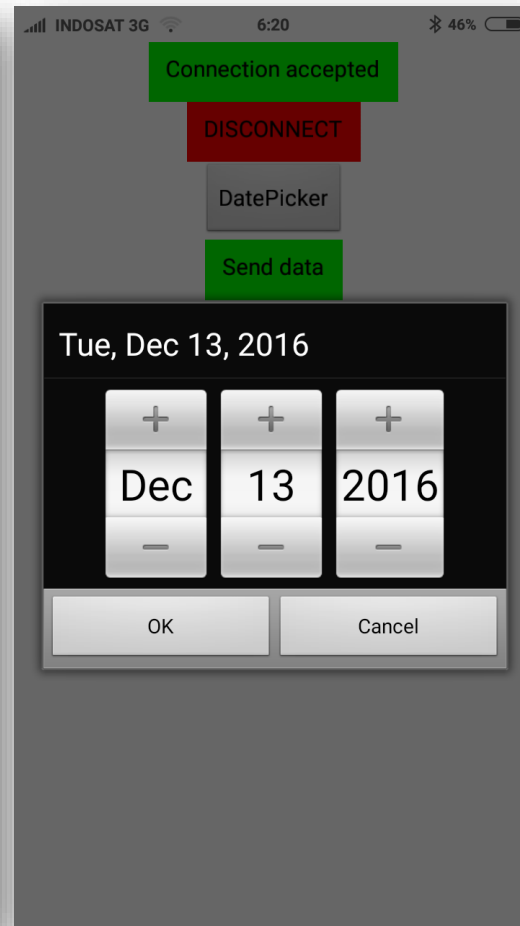
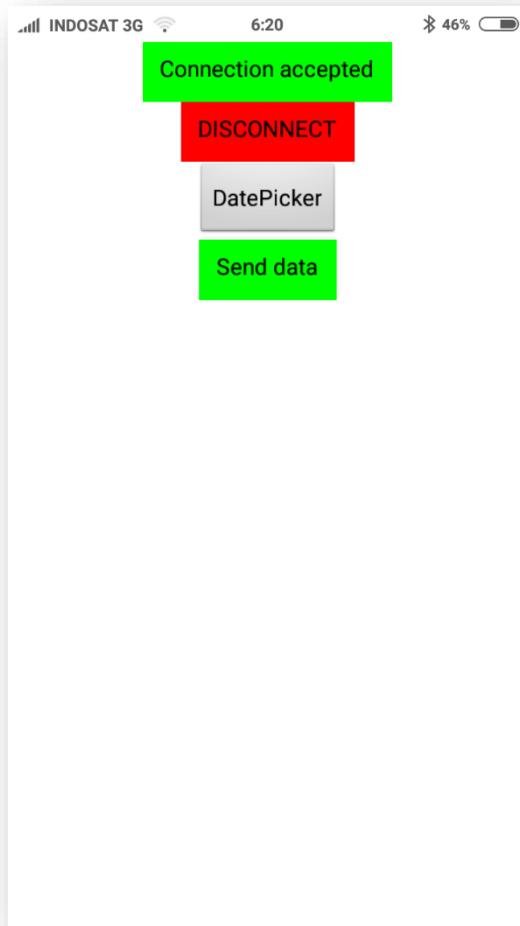


Server side

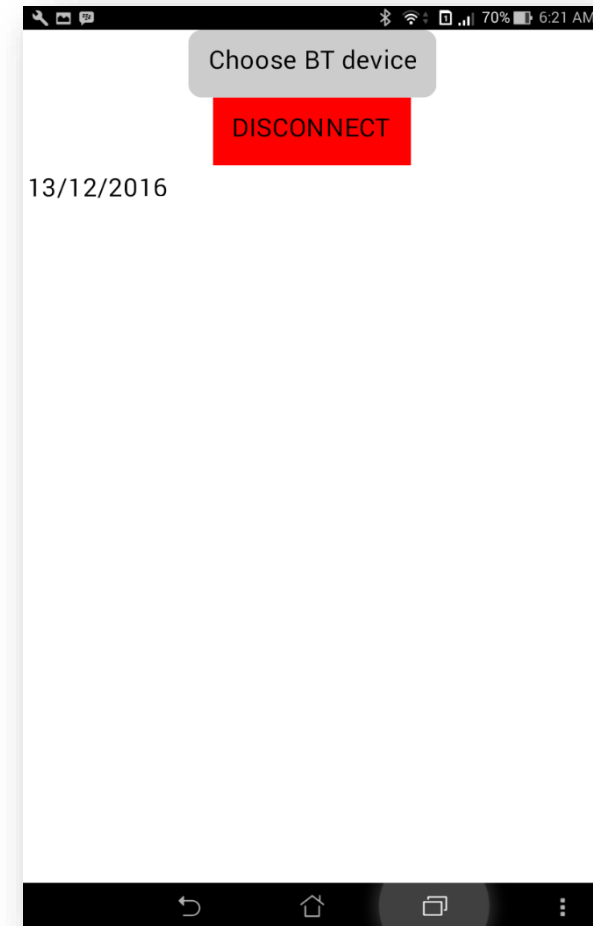


Client side





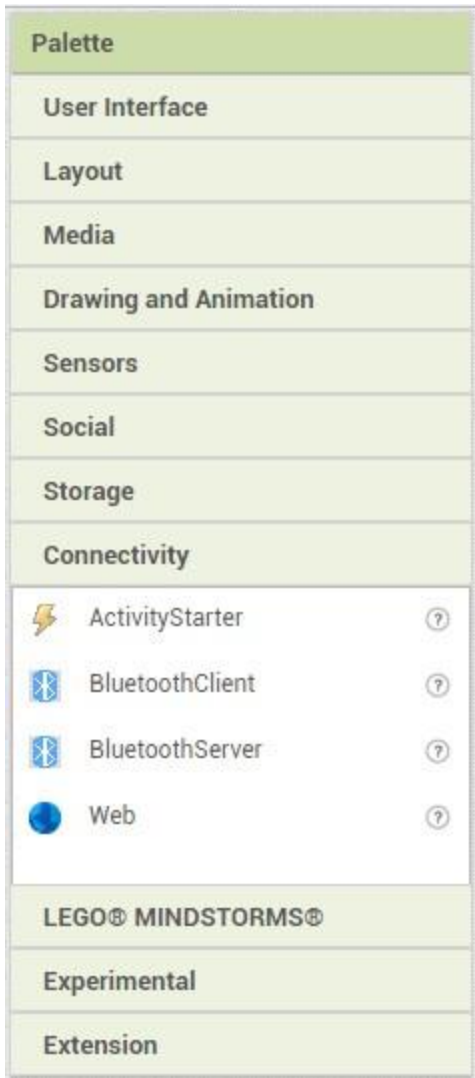
Server side



Client side

**Using Wifi to control device (http get)**

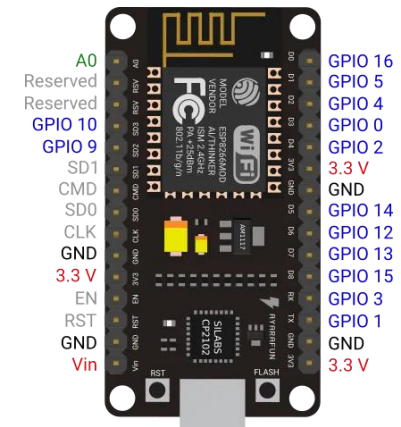




Both wifi-connected



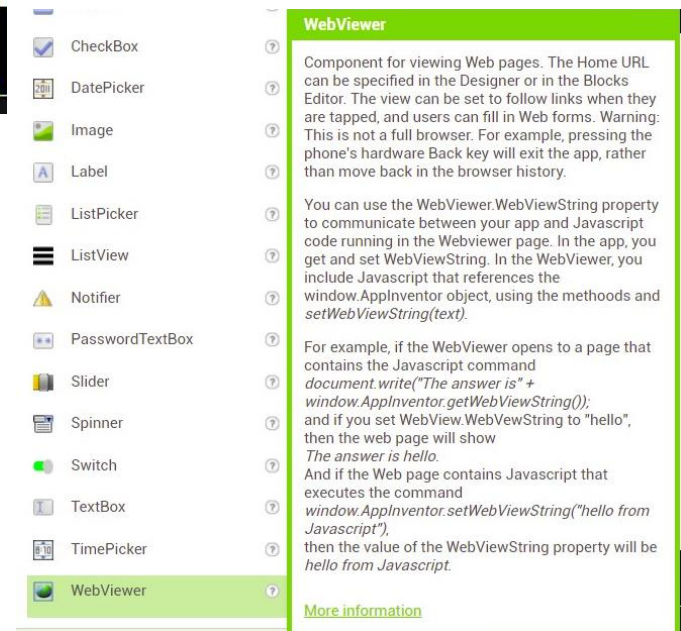
→  
command

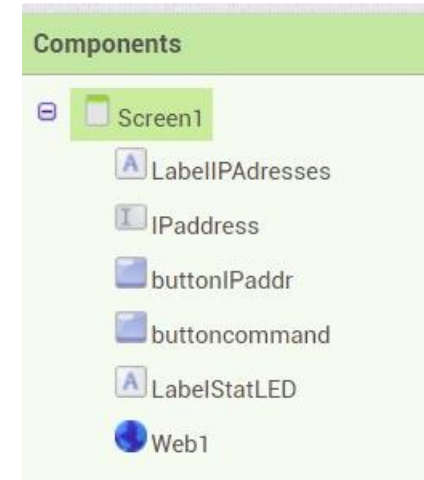


Alternative: http get



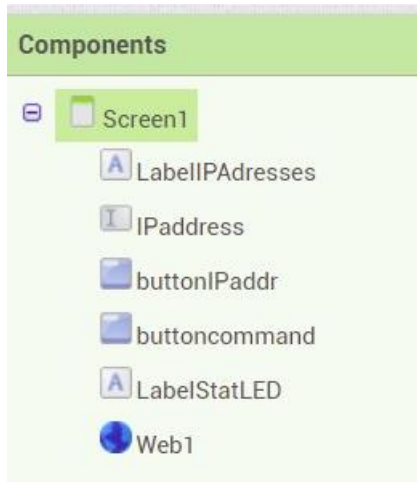
Use web (connectivity) **not webviewer**





**Note:**

Kode berikut belum diuji coba, namun secara teori semestinya bisa bekerja mungkin dengan beberapa penyesuaian





```
initialize global IPAddress to "http://192.168.0.26"
```

```
when Screen1.Initialize  
do  
  set buttoncommand.Visible to false  
  set LabelStatLED.Visible to false  
  set IPAddress.Text to get global IPAddress
```

```
when buttonIPAddr.Click  
do  
  set global IPAddress to IPAddress.Text  
  set buttoncommand.Visible to true  
  set LabelStatLED.Visible to true  
  set buttonIPAddr.Visible to false  
  set IPAddress.Visible to false  
  set LabelIPAddresses.Visible to false
```

2

```
when buttoncommand .Click
do
  if compare texts buttoncommand . Text = " command turn on "
  then
    set Web1 . Url to join get global IPAddress
      ( "/gpio/1 "
    set buttoncommand . Text to " command turn off "
    set buttoncommand . BackgroundColor to 
    set LabelStatLED . Text to " The LED is on "
    set buttoncommand . Image to on.png
  else
    set Web1 . Url to join get global IPAddress
      ( "/gpio/0 "
    set buttoncommand . Text to " command turn on "
    set buttoncommand . BackgroundColor to 
    set LabelStatLED . Text to " The LED is off "
    set buttoncommand . Image to off.png
  call Web1 .Get
```

```
#include <ESP8266WiFi.h>

#ifndef STASSID
#define STASSID "namassid"
#define STAPSK "passwordssid"
#endif

const char* ssid = STASSID;
const char* password = STAPSK;

// Create an instance of the server
// specify the port to listen on as an argument
WiFiServer server(80);
```

```
void setup() {
    Serial.begin(115200);

    // prepare LED
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, 0);

    // Connect to WiFi network
    Serial.println();
    Serial.println();
    Serial.print(F("connecting to "));
    Serial.println(ssid);

    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(F("."));
    }
    Serial.println();
    Serial.println(F("WiFi is connected"));

    // Start the server
    server.begin();
    Serial.println(F("server is started"));

    // Print the IP address
    Serial.println(WiFi.localIP());
}
```

```
void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }
  Serial.println(F("new client"));

  client.setTimeout(5000); // default is 1000

  // Read the first line of the request
  String req = client.readStringUntil('\r');
  Serial.println(F("request: "));
  Serial.println(req);

  // Match the request
  int val;
  if (req.indexOf(F("/gpio/0")) != -1) {
    val = 0;
  } else if (req.indexOf(F("/gpio/1")) != -1) {
    val = 1;
  } else {
    Serial.println(F("invalid request"));
    val = digitalRead(LED_BUILTIN);
  }

  // Set LED according to the request
  digitalWrite(LED_BUILTIN, val);
  // read/ignore the rest of the request
  // do not client.flush(): it is for output only, see below
  while (client.available()) {
    // byte by byte is not very efficient
    client.read();
  }

  // Send the response to the client
  // it is OK for multiple small client.print/write,
  // because nagle algorithm will group them into one single packet
  client.print(F("HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>\r\nGPIO is now "));
  client.print((val) ? F("high") : F("low"));
  client.print(F("<meta charset='utf-8'>"));
  client.print(F("<br><br><h1> Connecting to ESP8266 Wifi server to turn on the LED of the module </h1>"));
  client.print(F("<br><br>Click <a href='http://'>"));
  client.print(WiFi.localIP());
  client.print(F("/gpio/1'>here</a> to turn on the LED, or <a href='http://'>"));
  client.print(WiFi.localIP());
  client.print(F("/gpio/0'>here</a> to turn off the LED.</html>"));

  // The client will actually be *flushed* then disconnected
  // when the function returns and 'client' object is destroyed (out-of-scope)
  // flush = ensure written data are received by the other side
  Serial.println(F("Disconnecting from client"));
}
```



**TERIMA KASIH**