AllAboutEE
Electrical Engineering Website

ESP8266 Android Application to Control Arduino Digital Pins and Toggle LEDs

by Miguel on January 20, 2015

in ESP8266

In this post you'll find the Arduino and Android app code you need to control the digital pins of your Arduino from an Android phone through the ESP8266. The video below shows the app and explains the code, the code is below the video.

Arduino Code:

The Arduino code looks for the string +IPD whenever there is data available in the ESP8266, if the string is there then that means that a device (in this case the Android phone) is connecting to the ESP. The software then looks for the string pin=, then reads the pin number, toggles the pin, and sends back the string "Pin x is ON/OFF".

The code is below and works with every pin number, but don't use pins 0-3 since those are used for serial communication and the ESP8266 communication. Also, to get your ESP8266 and/or debug simply open the serial comm window in your Arduino IDE.

```
#include <SoftwareSerial.h>

#define DEBUG true

```

```
SoftwareSerial esp8266(2,3); // make RX Arduino line is pin 2, make TX Arduino line is pin 3.
                             // This means that you need to connect the TX line from the esp to the Arduino's pin 2
                             // and the RX line from the esp to the Arduino's pin 3
void setup()
{
  Serial.begin(9600);
  esp8266.begin(9600); // your esp's baud rate might be different

  pinMode(11,OUTPUT);
  digitalWrite(11,LOW);

  pinMode(12,OUTPUT);
  digitalWrite(12,LOW);

  pinMode(13,OUTPUT);
  digitalWrite(13,LOW);

  pinMode(10,OUTPUT);
  digitalWrite(10,LOW);

  sendCommand("AT+RST\r\n",2000,DEBUG); // reset module
  sendCommand("AT+CWMODE=1\r\n",1000,DEBUG); // configure as access point
  sendCommand("AT+CWJAP=\"mySSID\",\"myPassword\"\r\n",3000,DEBUG);
  delay(10000);
  sendCommand("AT+CIFSR\r\n",1000,DEBUG); // get ip address
  sendCommand("AT+CIPMUX=1\r\n",1000,DEBUG); // configure for multiple connections
  sendCommand("AT+CIPSERVER=1,80\r\n",1000,DEBUG); // turn on server on port 80

  Serial.println("Server Ready");
}

void loop()
{
  if(esp8266.available()) // check if the esp is sending a message
  {


    if(esp8266.find("+IPD,"))
    {
     delay(1000); // wait for the serial buffer to fill up (read all the serial data)
     // get the connection id so that we can then disconnect
     int connectionId = esp8266.read()-48; // subtract 48 because the read() function returns
                                           // the ASCII decimal value and 0 (the first decimal number) starts at 48

     esp8266.find("pin="); // advance cursor to "pin="

     int pinNumber = (esp8266.read()-48); // get first number i.e. if the pin 13 then the 1st number is 1
     int secondNumber = (esp8266.read()-48);
     if(secondNumber>=0 && secondNumber<=9)
     {
      pinNumber*=10;
      pinNumber +=secondNumber; // get second number, i.e. if the pin number is 13 then the 2nd number is 3, then add to the first number
     }

     digitalWrite(pinNumber, !digitalRead(pinNumber)); // toggle pin

     // build string that is send back to device that is requesting pin toggle
     String content;
     content = "Pin ";
     content += pinNumber;
     content += " is ";

     if(digitalRead(pinNumber))
     {
       content += "ON";
```

```
      }
      else
      {
        content += "OFF";
      }

      sendHTTPResponse(connectionId,content);

      // make close command
      String closeCommand = "AT+CIPCLOSE=";
      closeCommand+=connectionId; // append connection id
      closeCommand+="\r\n";

      sendCommand(closeCommand,1000,DEBUG); // close connection
     }
   }
}

/*
* Name: sendData
* Description: Function used to send data to ESP8266.
* Params: command - the data/command to send; timeout - the time to wait for a response; debug - print to Serial window?(true = yes, false = no)
* Returns: The response from the esp8266 (if there is a reponse)
*/
String sendData(String command, const int timeout, boolean debug)
{
    String response = "";

    int dataSize = command.length();
    char data[dataSize];
    command.toCharArray(data,dataSize);

    esp8266.write(data,dataSize); // send the read character to the esp8266
    if(debug)
    {
      Serial.println("\r\n====== HTTP Response From Arduino ======");
      Serial.write(data,dataSize);
      Serial.println("\r\n========================================");
    }

    long int time = millis();

    while( (time+timeout) > millis())
    {
      while(esp8266.available())
      {

        // The esp has data so display its output to the serial window
        char c = esp8266.read(); // read the next character.
        response+=c;
      }
    }

    if(debug)
    {
      Serial.print(response);
    }

    return response;
}

/*
* Name: sendHTTPResponse
* Description: Function that sends HTTP 200, HTML UTF-8 response
*/
```

```
void sendHTTPResponse(int connectionId, String content)
{

     // build HTTP response
     String httpResponse;
     String httpHeader;
     // HTTP Header
     httpHeader = "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=UTF-8\r\n";
     httpHeader += "Content-Length: ";
     httpHeader += content.length();
     httpHeader += "\r\n";
     httpHeader +="Connection: close\r\n\r\n";
     httpResponse = httpHeader + content + " "; // There is a bug in this code: the last character of "content" is not sent, I cheated by adding this extra space
     sendCIPData(connectionId,httpResponse);
}

/*
* Name: sendCIPDATA
* Description: sends a CIPSEND=<connectionId>,<data> command
*
*/
void sendCIPData(int connectionId, String data)
{
   String cipSend = "AT+CIPSEND=";
   cipSend += connectionId;
   cipSend += ",";
   cipSend +=data.length();
   cipSend +="\r\n";
   sendCommand(cipSend,1000,DEBUG);
   sendData(data,1000,DEBUG);
}

/*
* Name: sendCommand
* Description: Function used to send data to ESP8266.
* Params: command - the data/command to send; timeout - the time to wait for a response; debug - print to Serial window?(true = yes, false = no)
* Returns: The response from the esp8266 (if there is a reponse)
*/
String sendCommand(String command, const int timeout, boolean debug)
{
    String response = "";

    esp8266.print(command); // send the read character to the esp8266

    long int time = millis();

    while( (time+timeout) > millis())
    {
      while(esp8266.available())
      {

        // The esp has data so display its output to the serial window
        char c = esp8266.read(); // read the next character.
        response+=c;
      }
    }

    if(debug)
    {
      Serial.print(response);
    }

    return response;
}

```

200.

Android Code:

The Android code below was compiled with Android studio and should work with Android devices that have Android 2.2 or above.

Activity File

The activity file processes the user's interaction with the UI. When a button a clicked the IP address and port number are saved so you don't have to type them again, then the HTTP request is sent to the IP address specified with the parameter pin and the value of the pin.

```
1. package com.allaboutee.httphelper;
2.
3. import android.app.Activity;
4. import android.app.AlertDialog;
5. import android.content.Context;
6. import android.content.SharedPreferences;
7. import android.os.AsyncTask;
8. import android.os.Bundle;
9. import android.view.View;
10. import android.widget.Button;
11. import android.widget.EditText;
12.
13. import org.apache.http.HttpResponse;
14. import org.apache.http.client.ClientProtocolException;
15. import org.apache.http.client.HttpClient;
16. import org.apache.http.client.methods.HttpGet;
17. import org.apache.http.impl.client.DefaultHttpClient;
18.
19. import java.io.BufferedReader;
20. import java.io.IOException;
21. import java.io.InputStream;
22. import java.io.InputStreamReader;
23. import java.net.URI;
24. import java.net.URISyntaxException;
25.
26. public class HomeActivity extends Activity implements View.OnClickListener {
27.
28.     public final static String PREF_IP = "PREF_IP_ADDRESS";
29.     public final static String PREF_PORT = "PREF_PORT_NUMBER";
30.     // declare buttons and text inputs
31.     private Button buttonPin11,buttonPin12,buttonPin13;
32.     private EditText editTextIPAddress, editTextPortNumber;
33.     // shared preferences objects used to save the IP address and port so that the user doesn't have to
34.     // type them next time he/she opens the app.
35.     SharedPreferences.Editor editor;
36.     SharedPreferences sharedPreferences;
37.
38.     @Override
39.     protected void onCreate(Bundle savedInstanceState) {
40.         super.onCreate(savedInstanceState);
41.         setContentView(R.layout.activity_home);
42.
43.         sharedPreferences = getSharedPreferences("HTTP_HELPER_PREFS",Context.MODE_PRIVATE);
44.         editor = sharedPreferences.edit();
45.
46.         // assign buttons
47.         buttonPin11 = (Button)findViewById(R.id.buttonPin11);
48.         buttonPin12 = (Button)findViewById(R.id.buttonPin12);
49.         buttonPin13 = (Button)findViewById(R.id.buttonPin13);
50.
51.         // assign text inputs
52.         editTextIPAddress = (EditText)findViewById(R.id.editTextIPAddress);
```

```
        editTextPortNumber = (EditText)findViewById(R.id.editTextPortNumber);

        // set button listener (this class)
        buttonPin11.setOnClickListener(this);
        buttonPin12.setOnClickListener(this);
        buttonPin13.setOnClickListener(this);

        // get the IP address and port number from the last time the user used the app,
        // put an empty string "" is this is the first time.
        editTextIPAddress.setText(sharedPreferences.getString(PREF_IP,""));
        editTextPortNumber.setText(sharedPreferences.getString(PREF_PORT,""));
    }


    @Override
    public void onClick(View view) {

        // get the pin number
        String parameterValue = "";
        // get the ip address
        String ipAddress = editTextIPAddress.getText().toString().trim();
        // get the port number
        String portNumber = editTextPortNumber.getText().toString().trim();


        // save the IP address and port for the next time the app is used
        editor.putString(PREF_IP,ipAddress); // set the ip address value to save
        editor.putString(PREF_PORT,portNumber); // set the port number to save
        editor.commit(); // save the IP and PORT

        // get the pin number from the button that was clicked
        if(view.getId()==buttonPin11.getId())
        {
            parameterValue = "11";
        }
        else if(view.getId()==buttonPin12.getId())
        {
            parameterValue = "12";
        }
        else
        {
            parameterValue = "13";
        }



        // execute HTTP request
        if(ipAddress.length()>0 && portNumber.length()>0) {
            new HttpRequestAsyncTask(
                    view.getContext(), parameterValue, ipAddress, portNumber, "pin"
            ).execute();
        }
    }

    /**
     * Description: Send an HTTP Get request to a specified ip address and port.
     * Also send a parameter "parameterName" with the value of "parameterValue".
     * @param parameterValue the pin number to toggle
     * @param ipAddress the ip address to send the request to
     * @param portNumber the port number of the ip address
     * @param parameterName
     * @return The ip address' reply text, or an ERROR message is it fails to receive one
     */
    public String sendRequest(String parameterValue, String ipAddress, String portNumber, String parameterName) {
        String serverResponse = "ERROR";
```

```
118.
119.         try {
120.
121.             HttpClient httpclient = new DefaultHttpClient(); // create an HTTP client
122.             // define the URL e.g. http://myIpaddress:myport/?pin=13 (to toggle pin 13 for example)
123.             URI website = new URI("http://"+ipAddress+":"+portNumber+"/?"+parameterName+"="+parameterValue);
124.             HttpGet getRequest = new HttpGet(); // create an HTTP GET object
125.             getRequest.setURI(website); // set the URL of the GET request
126.             HttpResponse response = httpclient.execute(getRequest); // execute the request
127.             // get the ip address server's reply
128.             InputStream content = null;
129.             content = response.getEntity().getContent();
130.             BufferedReader in = new BufferedReader(new InputStreamReader(
131.                     content
132.             ));
133.             serverResponse = in.readLine();
134.             // Close the connection
135.             content.close();
136.         } catch (ClientProtocolException e) {
137.             // HTTP error
138.             serverResponse = e.getMessage();
139.             e.printStackTrace();
140.         } catch (IOException e) {
141.             // IO error
142.             serverResponse = e.getMessage();
143.             e.printStackTrace();
144.         } catch (URISyntaxException e) {
145.             // URL syntax error
146.             serverResponse = e.getMessage();
147.             e.printStackTrace();
148.         }
149.         // return the server's reply/response text
150.         return serverResponse;
151.     }
152.
153.
154.     /**
155.      * An AsyncTask is needed to execute HTTP requests in the background so that they do not
156.      * block the user interface.
157.      */
158.     private class HttpRequestAsyncTask extends AsyncTask<Void, Void, Void> {
159.
160.         // declare variables needed
161.         private String requestReply,ipAddress, portNumber;
162.         private Context context;
163.         private AlertDialog alertDialog;
164.         private String parameter;
165.         private String parameterValue;
166.
167.         /**
168.          * Description: The asyncTask class constructor. Assigns the values used in its other methods.
169.          * @param context the application context, needed to create the dialog
170.          * @param parameterValue the pin number to toggle
171.          * @param ipAddress the ip address to send the request to
172.          * @param portNumber the port number of the ip address
173.          */
174.         public HttpRequestAsyncTask(Context context, String parameterValue, String ipAddress, String portNumber, String parameter)
175.         {
176.             this.context = context;
177.
178.             alertDialog = new AlertDialog.Builder(this.context)
179.                     .setTitle("HTTP Response From IP Address:")
180.                     .setCancelable(true)
181.                     .create();
182.
```

```
            this.ipAddress = ipAddress;
            this.parameterValue = parameterValue;
            this.portNumber = portNumber;
            this.parameter = parameter;
        }

        /**
         * Name: doInBackground
         * Description: Sends the request to the ip address
         * @param voids
         * @return
         */
        @Override
        protected Void doInBackground(Void... voids) {
            alertDialog.setMessage("Data sent, waiting for reply from server...");
            if(!alertDialog.isShowing())
            {
                alertDialog.show();
            }
            requestReply = sendRequest(parameterValue,ipAddress,portNumber, parameter);
            return null;
        }

        /**
         * Name: onPostExecute
         * Description: This function is executed after the HTTP request returns from the ip address.
         * The function sets the dialog's message with the reply text from the server and display the dialog
         * if it's not displayed already (in case it was closed by accident);
         * @param aVoid void parameter
         */
        @Override
        protected void onPostExecute(Void aVoid) {
            alertDialog.setMessage(requestReply);
            if(!alertDialog.isShowing())
            {
                alertDialog.show(); // show dialog
            }
        }

        /**
         * Name: onPreExecute
         * Description: This function is executed before the HTTP request is sent to ip address.
         * The function will set the dialog's message and display the dialog.
         */
        @Override
        protected void onPreExecute() {
            alertDialog.setMessage("Sending data to server, please wait...");
            if(!alertDialog.isShowing())
            {
                alertDialog.show();
            }
        }

    }
}
```

Layout File

This file simply creates the UI (buttons and textboxes with hints).

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
     >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical">


        <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="IP Address:"
        android:id="@+id/textView" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="e.g. 192.168.0.10"
        android:id="@+id/editTextIPAddress" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Port Number:"
        android:id="@+id/textView2" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:ems="10"
        android:hint="e.g. 80"
        android:id="@+id/editTextPortNumber" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pin 11"
        android:id="@+id/buttonPin11" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pin 12"
        android:id="@+id/buttonPin12" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pin 13"
        android:id="@+id/buttonPin13" />
    </LinearLayout>

</ScrollView>


```

Manifest File

The only change I had to make to the manifest file was to add the Internet permission to allow for HTTP requests.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.allaboutee.httphelper" >

    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".HomeActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```


Related posts:

- **ESP8266 Arduino LED Control (Control The Digital Pins Via WiFi, Send Data From Webpage to Arduino)**
- **How To Use the ESP8266 and Arduino as a Webserver**
- **ESP8266 Arduino Code and Schematic to Send AT Commands and Print Output**
- **Make a Phone Call With Voice Using Your ESP8266 and Arduino With Twilio's API**
- **How To Send Text Messages From The ESP8266 and Twilio Using NodeMcu**

Previous post: ESP8266 Arduino LED Control (Control The Digital Pins Via WiFi, Send Data From Webpage to Arduino)

Next post: Make a Phone Call With Voice Using Your ESP8266 and Arduino With Twilio's API

- **Search:**

  To search, type and hit enter

- **AllAboutEE Facebook Page:**

- **Categories**
  - Circuits
  - ESP8266
  - LabVIEW
  - Microcontrollers
    - Arduino
    - ARM
    - AVR
    - PIC
      - PIC18 Explorer Board
  - Phone Development
    - Android
  - Uncategorized