



Design Team Six

Sending Data via HTTP Requests with the Arduino Yún

ABSTRACT

The Arduino Yún is able to connect to Wi-Fi networks via onboard hardware. Using this hardware and open-source libraries provided by Arduino, it is possible to make Hyper Text Transfer Protocol (HTTP) GET requests without great investment of time. While the typical data transmission method for HTTP is via POST requests, all major webserver applications support data transfer via GET requests using the query string of a URI defined in the HTTP standard.

This document describes a method for leveraging the Arduino libraries to send blocking GET requests to poll a webserver for information. Possible uses for this concept would be a microcontroller that fetches weather forecasts from a webserver and takes various actions based on the predictions received.

A method for building HTTP GET query strings from equal length arrays of keys and values is described. This allows for the Arduino Yún to transmit data using the built in libraries for HTTP connectivity without needing to provide code for creating the HTTP headers needed for a POST request.

KEYWORDS

Microcontroller, Arduino, Wi-Fi, HTTP, GET.

1. Arduino Yún Hardware Interface

The Arduino Yún is able to connection to various Wi-Fi networks using on board hardware. Two processors are present on the board, an ATmega32u4 processor which runs the bulk of a user's code as well as an Atheros AR9331 which runs a customized Linux distribution. In order to initiate an HTTP request, the ATmega32u4 processor (the "Arduino processor") must send messages to the AR9331 (the "Linux processor") and wait for the response back from the Linux processor with the result.

In order to send messages between the two processors, one must first initiate the communication channel between the two proccessors. This channel is refered to as the Bridge and much of the code needed to use the Bridge is provided by Arduino in the Bridge library. To include the library in the current program, which the Arduino IDE refers to as a sketch, one needs only add the following line of C code to the top of their sketch:

```
#include <Bridge.h>
```

Once this has been done, establishing the Bridge between the two processors is relatively simple. In the Arduino's `setup()` function, we need to add a single line of code. However, for debugging and user feedback purposes it is often desirable to use an LED to indicate whether the Bridge has been established, a process that takes about 2 seconds to complete. The required lines of code to accomplish both the LED status light and the Bridge initiation are:

```
void setup(){
    // ...
    pinMode(13,OUTPUT);
    digitalWrite(13, LOW); //Turn ON the LED
    Bridge.begin();        //Initiation Bridge
    digitalWrite(13, HIGH); //Turn OFF the LED
    // ...
}
```

Having now established the Bridge between the processors, HTTP requests can be made by the Arduino processor.

2. Sending Synchronous HTTP GET Requests

The Arduino IDE provides a library with two mechanisms for sending HTTP GET Requests. The simpler technique is to send the request and block execution of later code until the response has been received from the server; this is known as a “synchronous” request. Asynchronous requests allow the microcontroller to perform other tasks while waiting for the server's response to arrive. Synchronous requests are discussed here as this type of request is much more commonly used; typically, microcontrollers only proceed after processing the response from the server.

The library that provides for simple HTTP GET requests can be included with the following line at the top of the sketch:

```
#include <HttpClient.h>
```

This library implements the `HttpClient` class. This class's `get()` method will send an HTTP GET request to a given URL and block until the response has been received or the request times out. The code below fetches the page at <http://www.company.com/weather/> every 15 minutes.

```
void loop(){
    //Initialize an HttpClient object
    HttpClient client;
    //long unsigned int to keep track of time
    long unsigned int last_check = 0;
    //1 KB buffer to store response in
    char buffer[1024];
    //Used to prevent buffer overflows
    int i;

    //Compare the current time to the last time the uController
    //checked the weather. 15 minutes = 900 seconds
    //900 seconds = 900,000 milliseconds
```

```

        if(millis() - time > 900000){
            time = millis();
            //Make request to company's server
            client.get("http://www.company.com/weather/");

            //Get all the data and move it to a buffer that
            //is more accessible... one byte at a time.
            i = 0;
            while(client.available() && i < 1024){
                buffer[i] = client.read();
                i++;
            }
            //uController should now act on the data it received
            //Perhaps it updates an LCD display with a picture of
            //Sun, Clouds, Rain, Snow, etc.

        }else{
            //Just keep waiting. Possibly enter low-power/sleep
            //mode for specific time interval to save
            //battery power.
        }
    }
}

```

The while loop above simply pulls bytes one by one out of the internal buffer of the HttpClient object and moves them into a local buffer until either the buffer runs out of space or the end of the response is reached. At this point, the response would be processed and the microcontroller would take action based on the data received.

3. Sending Data via HTTP GET Requests

While HTTP GET requests are typically used to ask that a webserver deliver data, it is possible to use this form of request to transmit data to a server. Due to the nature of HTTP, the server will then deliver a response back to the Yún.

Data is sent in an HTTP GET request by appending it onto the end of the URL string; when the server receives the request, it parses the string to determine which page is being requested and what data, if any, is being sent. Data is passed as a series of key-value pairs and separated from the resource URL by a question mark. What follows the question mark is referred to as the query string in the HTTP specification.

The function below will take two arrays of the same length, one of which represents the data keys and the other the data values, as well as the length of these arrays and builds the data string for the HTTP GET request. This string is then returned so that a GET request can be easily created from two arrays of respective key-value pairs. Here Arduino's String objects are chosen over traditional C character arrays for their simplicity and because the String object manages the dynamic sizing of the resulting string automatically and efficiently.

```

String build_get_data(String[] keys, String[] values, int len){
    i = 0; //for loop variable
    String URI_string = ""; //Initialize to empty
    String encoded_string;
    boolean firstKeyValuePair = true; //Different first loop

```

```

//Loop over key-value pairs and add to the URI_string
for(i = 0; i < len; i++){
    //If first key-value pair, skip what follows but
    //unset firstKeyValuePair so it triggers next time
    if(firstKeyValuePair){
        firstKeyValuePair = false;
    }else{
        //Need to add & character to separate value
        //of one pair from key of another
        URI_string = URI_string + '&';
    }
    //Replace any reserved URL characters in key. Without
    //this step, an '=' character in the key could cause
    //the server to parse the data in unwanted ways.
    encoded_string = url_encode(keys[i]);
    URI_string = URI_string + encoded_string;

    //Add '=' character to indicate what follows is
    //value. Value must be checked for any reserved
    //characters before being concatenated onto
    //URI_string.
    URI_string = URI_string + '=';
    encoded_string = url_encode(values[i]);
    URI_string = URI_string + encoded_string;

    //key-value pair added, move on to the next one
}
//All key-value pairs added, so return the resulting string
return URI_string;
}

```

The above function relies on another function, `url_encode`, not shown here. This function searches for any characters in the parameter string which are reserved in URLs such as a space character, equal sign, or ampersand. This function has been omitted because it is very simple and numerous implementations can be found online. Without sanitizing the keys and values, reserved characters could cause the server to interpret the microcontroller's data in undesired ways, as stated in a comment within the code.

Sending the data via HTTP GET request is relatively simple once this function has been written. A relatively simple example is included below where a GET request is made every time the value of analog input 0 differs from its previous values, as a rolling average, by a certain threshold. The analog input in this case represents the output from a load cell.

```

void loop(){
    float avgScaleValue = 0.0;
    float curScaleValue;
    HttpClient client;
    String URL;
    String keys[1];    //1-element array so build_get_data
    String values[1];  //can be used without modification.
                        //This uses just as much space as a
                        //standard String variable so there
                        //is no reason not to;

    int pairs = 1;
}

```

```

char response[8]; //Stores response from server.
int i;             //useful little counter variable

//Get analog value, compare to rolling average.
curScaleValue = analogRead(0);
//Check if change exceeds threshold
if(abs(curScaleValue - avgScaleValue) > 300){

    //Send weight to server at IP address 192.168.1.1
    keys[0] = "weight";
    values[1] = curScaleValue;
    URL = "http://192.168.1.1/updateWeight.php?";
    URL = URL + build_get_data(keys, values, pairs);
    client.get(URL);

    //Move response into local buffer. We expect the
    //response "CONFIRM" all else indicates a failure.
    i = 0;
    while(client.available() && i < 7){
        response[i] = client.read();
        i++;
    }
    //NULL terminate the response string
    response[i] = 0;

    //if strcmp == 0, then the strings are equal
    if(strcmp(response, "CONFIRM") == 0){
        //Any code to run on successful scale update
    }else{
        //Any code to run on failed scale update
        //Perhaps illuminate a red LED to indicate the
        //error at a glance.
    }
}
//Add new value to running average
avgScaleValue = .75*avgScaleValue + .25*curScaleValue;
}

```

The basic error checking performed in the above code could be easily extended. If the server is configured to return different messages indicating different error conditions, the microcontroller could take various actions in response. For instance, if the server replies with a message indicating it's under a heavy load and cannot properly handle the request, the microcontroller could wait for a specified time interval before attempting to resend the data to ease network traffic.

4. Conclusions

The Arduino Yún microcontroller provides a number of libraries which allow for easy implementation of HTTP GET requests. While most data sent over HTTP is submitted via a POST request it is nonetheless possible to submit data via GET requests. As the libraries provided by Arduino do not include implementation of POST requests it is desirable to leverage the libraries to submit data via GET.

This can be done by manually building the GET query string and concatenating it onto the end of the URL where the server expects to receive data. All major webserver implementations can interpret data passed in this way. The expected format of the data is not complex, but care must be taken to properly encode any HTTP reserved characters present in either the keys or the values being submitted to the server as data. If these reserved characters are not encoded the server may parse the data in undesired ways, corrupting the data.

Because of the nature of HTTP's polling requests, a webserver is expected to return some information to the microcontroller after each request. This lends itself well to checking that the server was able to interpret the data successfully because a server can be expected to return a simple message if all is well: for example, "CONFIRM" or "SUCCESS". Additionally, information regarding to a failure can also be sent so that debugging can be performed by examining the data received by the microcontroller if the server is not accessible.

5. Additional Reading

Further information on the Arduino Yún's hardware can be found on Arduino's website at:

<http://arduino.cc/en/Main/ArduinoBoardYun?from=Products.ArduinoYUN>

A basic overview of the component and their interaction can be found on that page, as well as links to the board's schematics.

Arduino is an Open Source platform and provides excellent documentation for all their libraries. Documentation on the Bridge library and a basic description of the interaction between the processors can be found at:

<http://arduino.cc/en/Reference/YunBridgeLibrary>

The HttpClient library is a sublibrary of the Bridge library. Its documentation can be found on that page as well.

The IETF has defined the HTTP Standard through a series of Requests for Comments (RFCs). The current RFCs defining the HTTP protocol, including the GET request method, are 7230, 7231, 7232, 7233, 7234, and 7235. These are available on the IETF's website for viewing free of charge. The archive of RFCs can be accessed at:

<http://tools.ietf.org/html/>