MASTERs 2014
LAB Manual for 18020 EWN
Easy Wireless Networking Using the Arduino™ Compatible chipKIT™ Platform

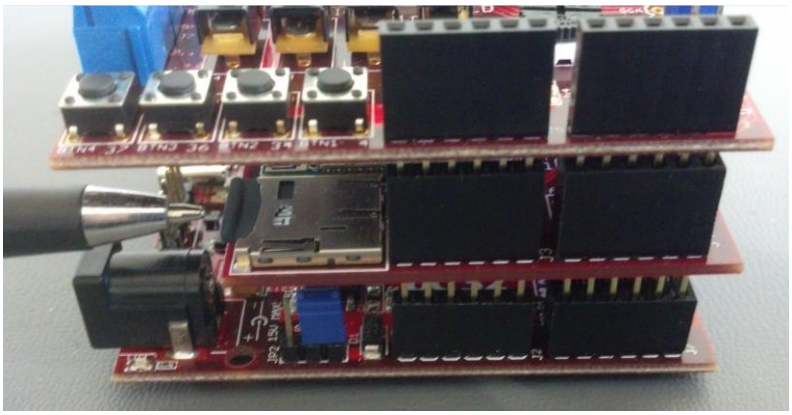
Table of Contents

LAB 1

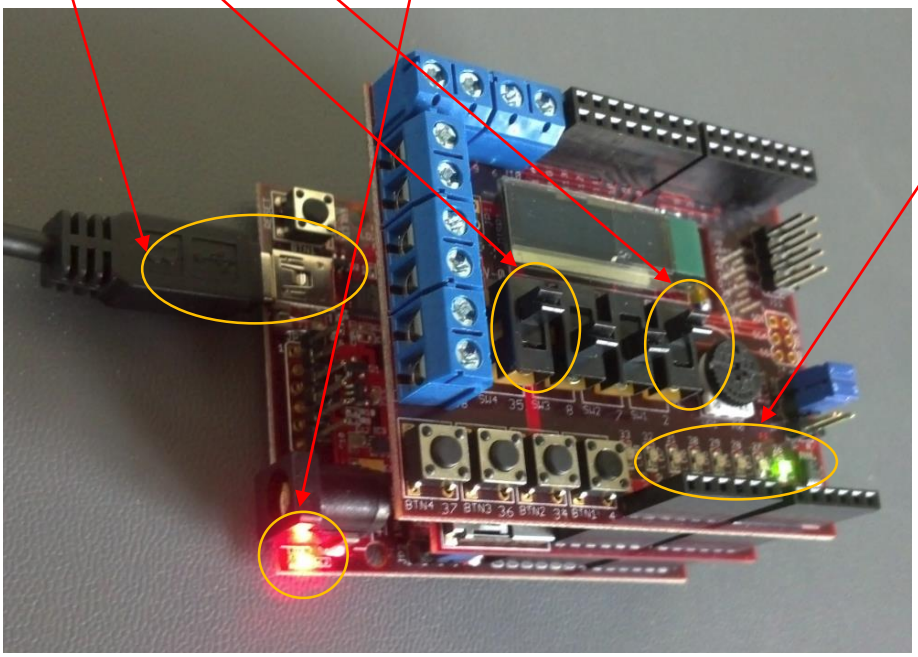
Build and Run the DEIPcK HTTP Example Server

1. In this exercise we will build the HTTP Example Server, load it, and run it without modification. Then you can look at some of the source files to see how the server uses the DEIPcK Network Stack.

2. Make sure the µSD card if fully inserted into the µSD reader.



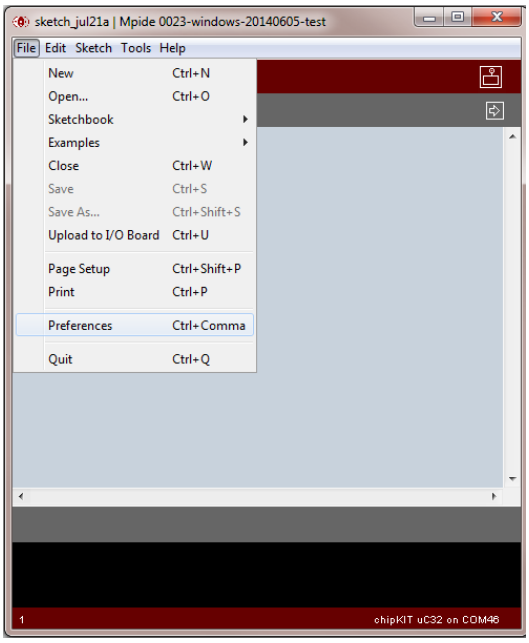
3. Plug the chipKIT™ uC32 into the USB cable and attach the USB cable to your computer. Set the switches SW4 and SW1 to ON. The red power light should come ON. Do not worry if any of the green LEDs come ON or not.



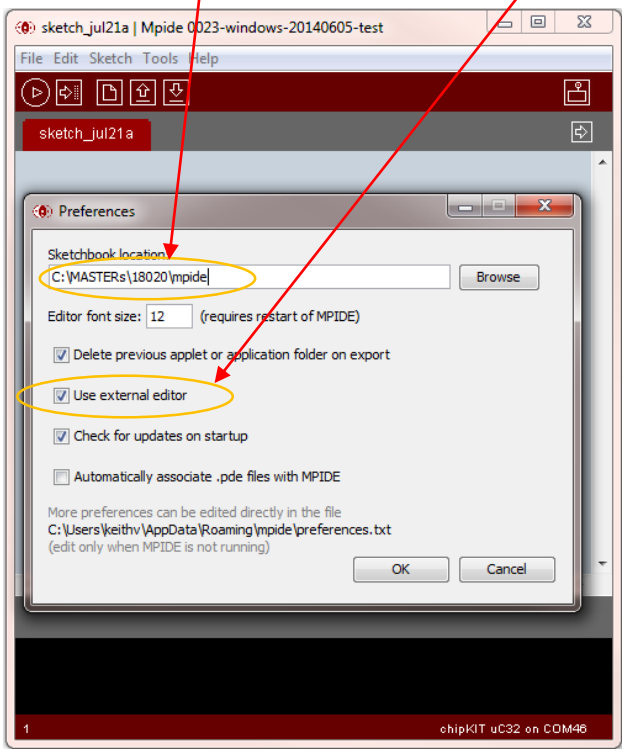
4. As a note, many of the signals on the Basic IO Shield conflict with the WiFi Shield. All buttons conflict and you should not press any of the buttons as that could reset the WiFi Shield. Likewise Switches 1 and 4 are shared with signals on the WiFi Shield and Switches 1 and 4 should remain ON at all times. We can use switches 2 and 3, and we will logically think of switch 2 as input 1, and switch 3 as input 2; this will be used later in this exercise. None of the LEDs conflict with the WiFi Shield and we can use all of the LEDs on the Basic IO Shield

5. On the computer desktop start MPIDE by clicking the MPIDE icon

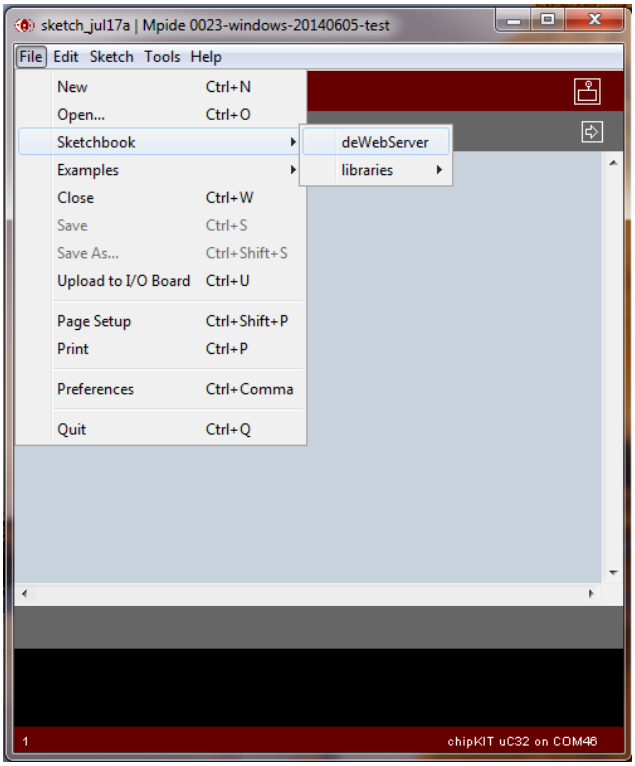
6. Ensure that you are looking at the correct sketchbook folder. These computers are used by another class and your sketchbook directory maybe set to the wrong location. Go into Preferences



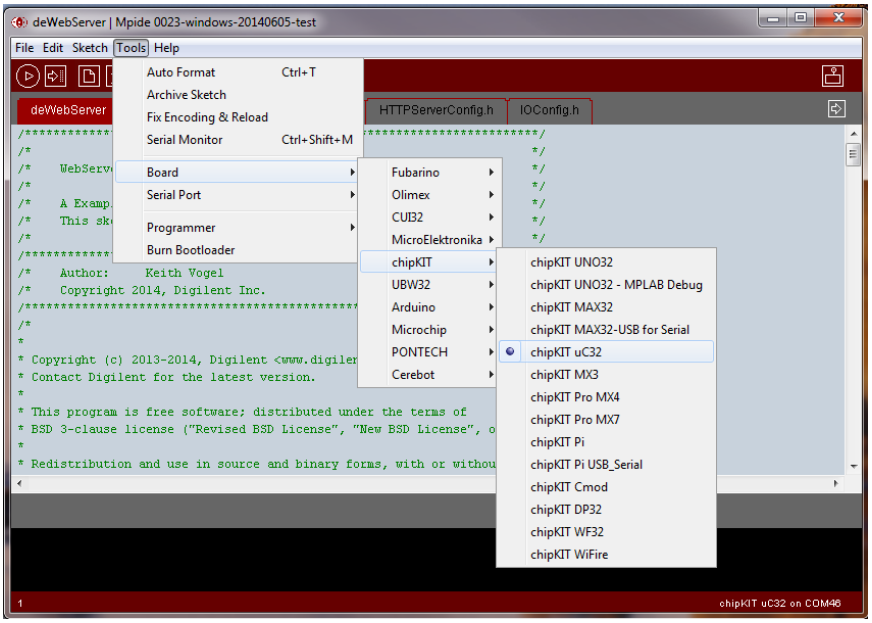
7. And make sure your sketchbook location is set to C:\MASTERs\18020\mpide; Verify that the class subdirectory is 18020! Also make sure the "Use external editor" is checked as we will be using Notepad for our editor.



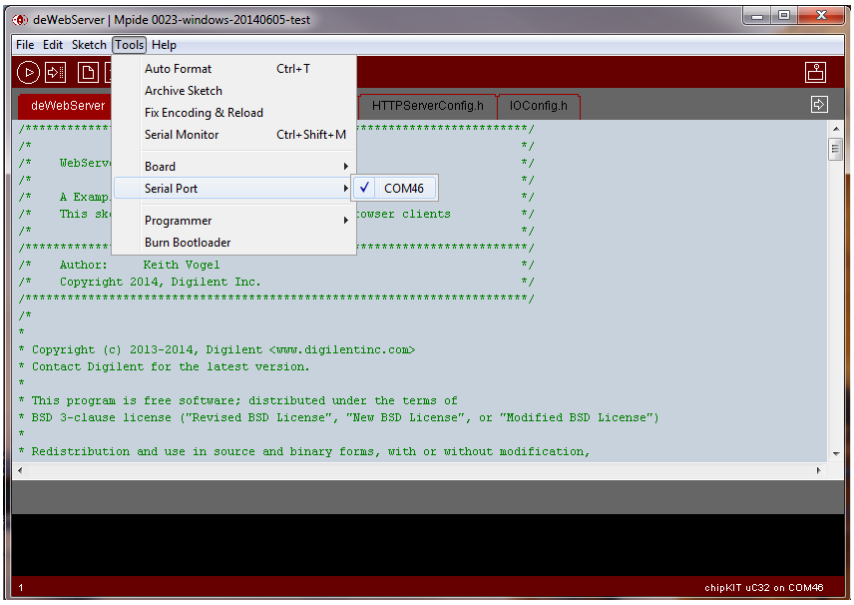
8. Now when you open the Sketchbook location you should see 2 directories, deWebServer and libraries. Select deWebServer. Two MPIDE windows will be open, you can close the first window leaving only deWebServer open.



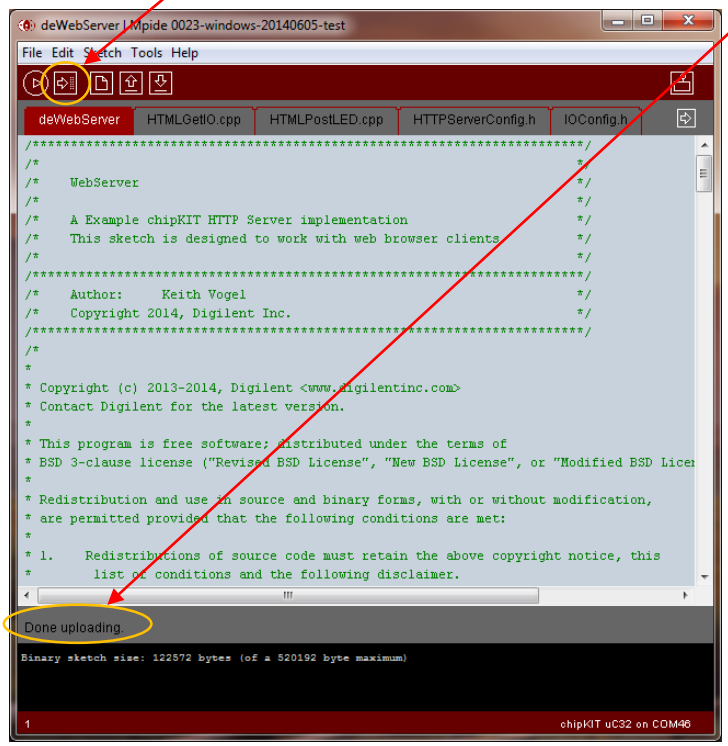
9. Ensure that the chipKIT™ uC32 board is selected under Tools->chipKIT



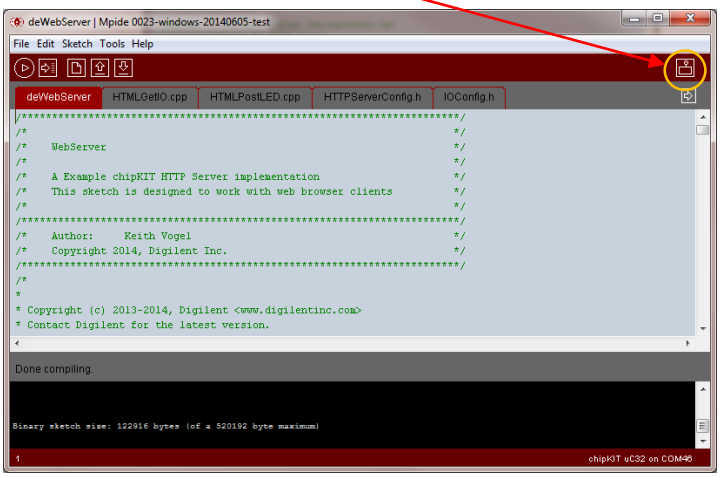
10. Ensure the correct serial port is selected. Serial ports vary from machine to machine, board to board. So if you have more than one serial port showing, unplug the chipKIT™ uC32 and see which serial port disappears, re-plug it in and see which one reappears; and that is the serial port the chipKIT™ uC32 is on. To see the changes you will have to close and reopen the Tools->Serial Port menu.



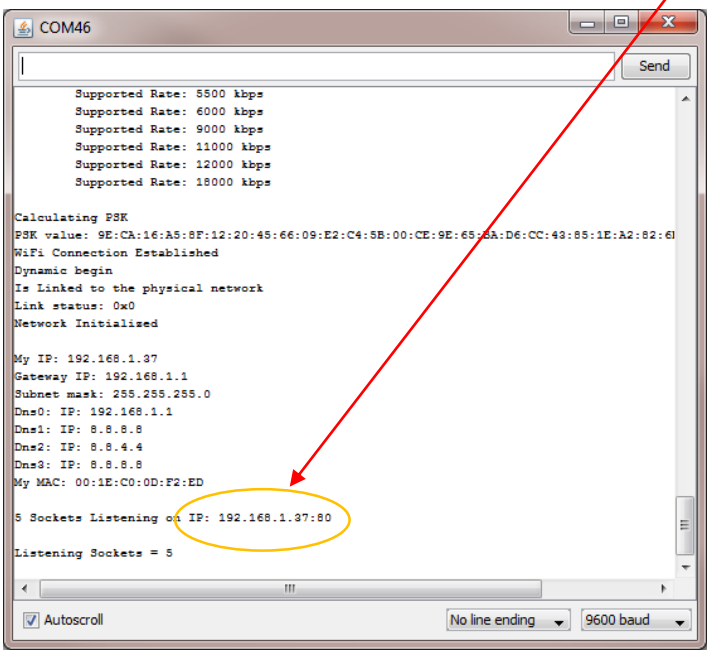
Compile and upload the sketch to your chipKIT™ uC32. Wait for the upload to finish.



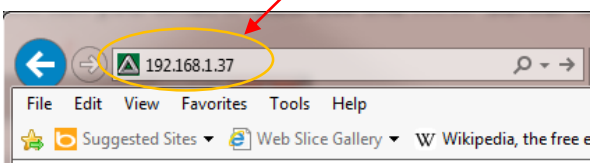
11. Open the Serial Monitor



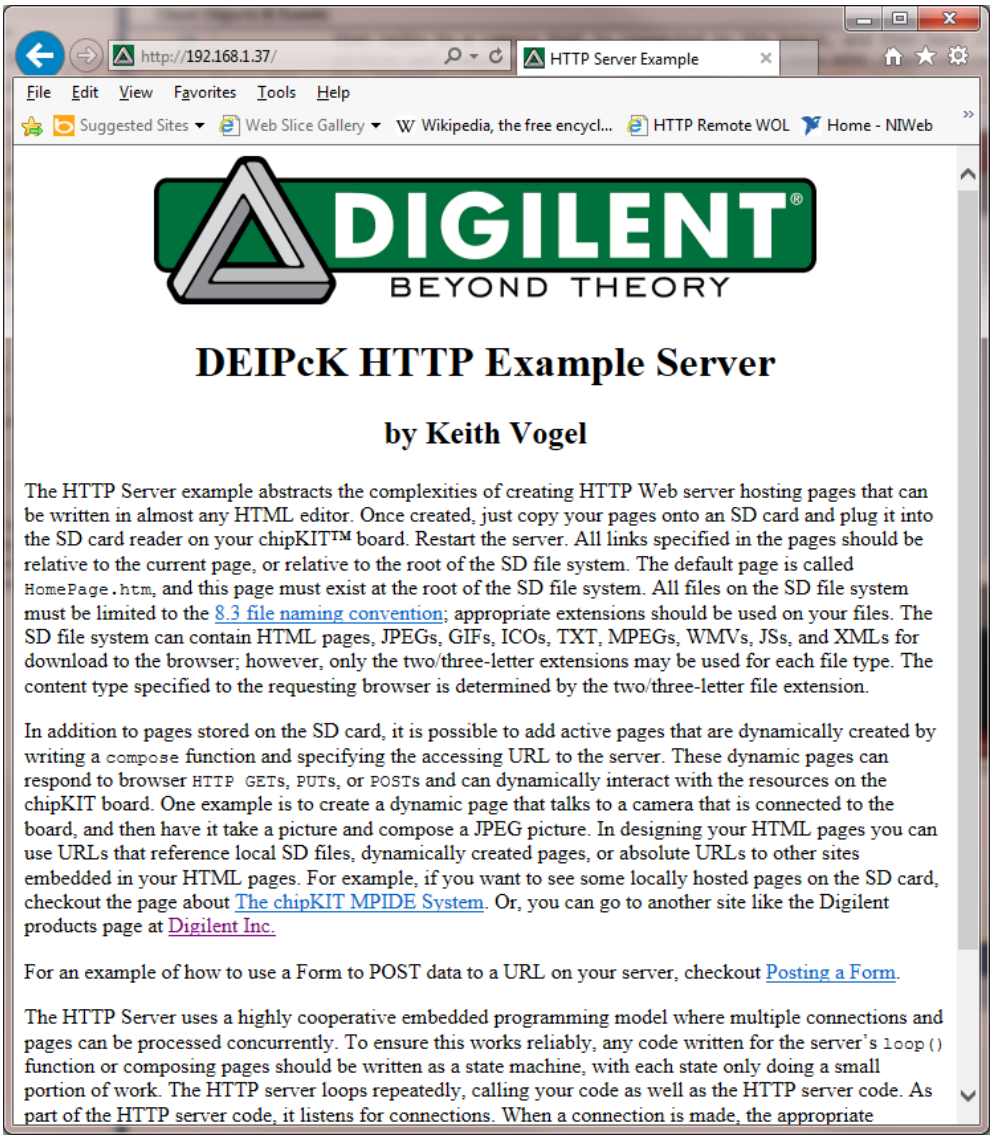
12. And observer the output. The output will tell you what IP address the HTTP Server is listening on; this was assigned to you by the DHCP server on this network. The exact IP address will vary.



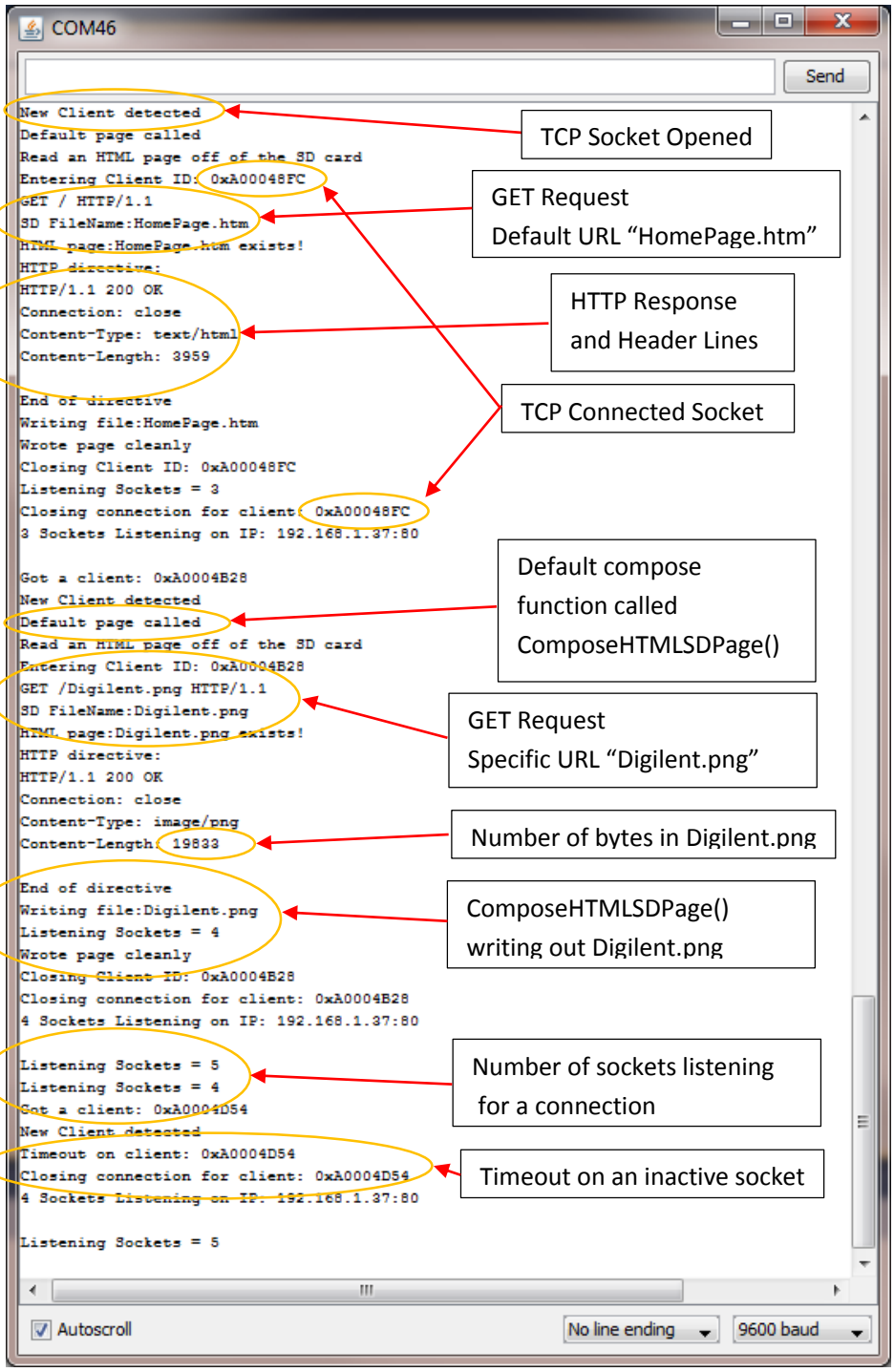
13. Start Internet Explorer and type in the IP Address in the Address line and enter. No need to enter the port number 80.



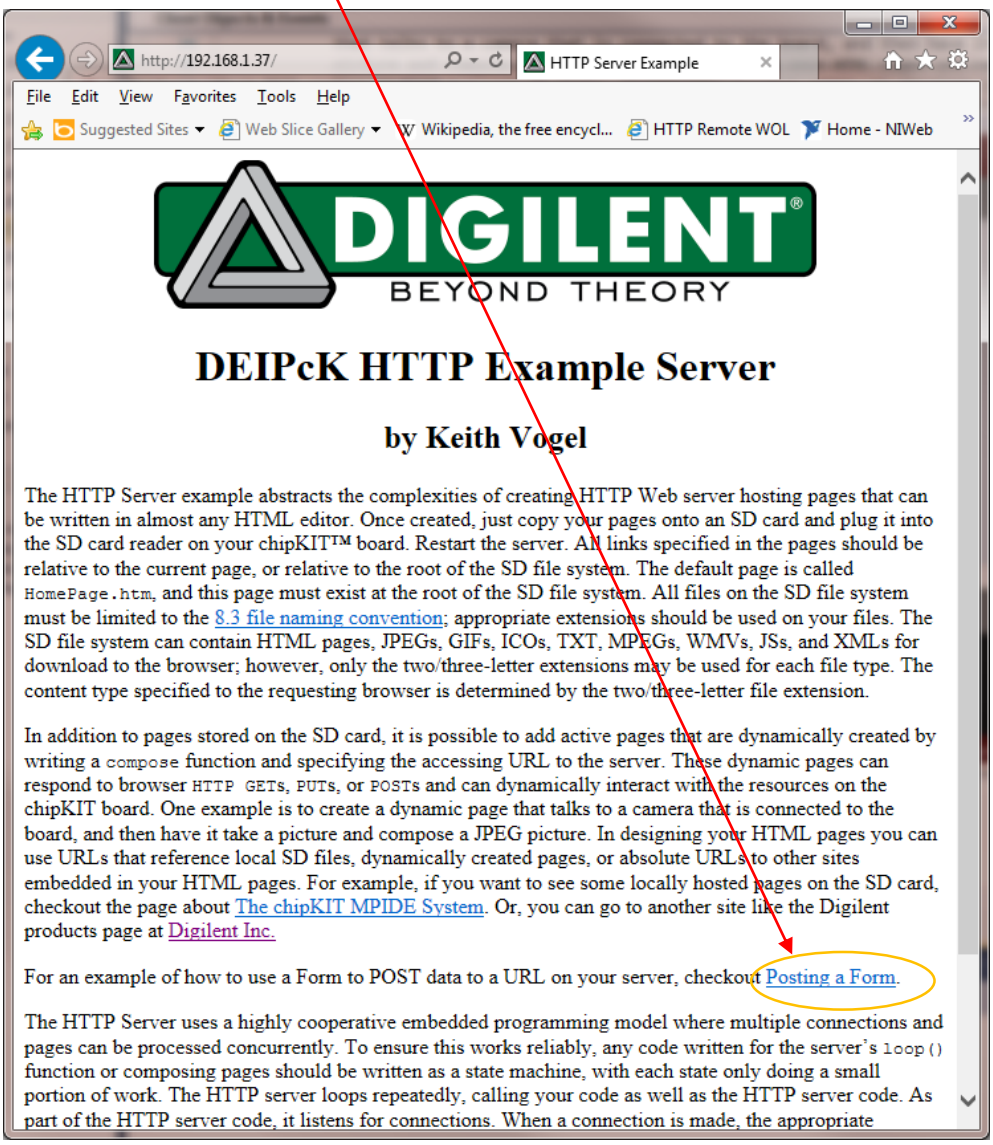
14. Internet Explorer should load the HomePage.htm as servered up off of the µSD card on the chipKIT™ uC32. You are now running the HTTP Server on the chipKIT™ uC32.



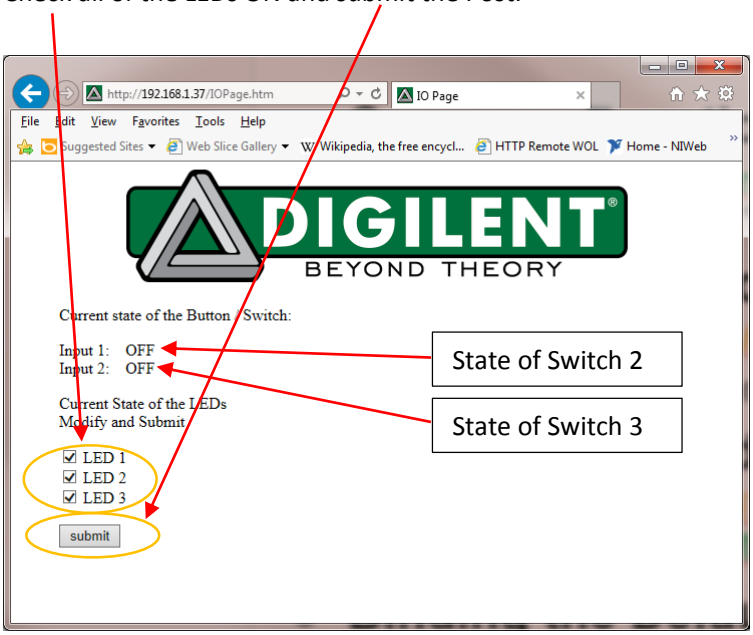
15. In your Serial Monitor window you will see the uC32 processing the connections to render the HomePage.htm.



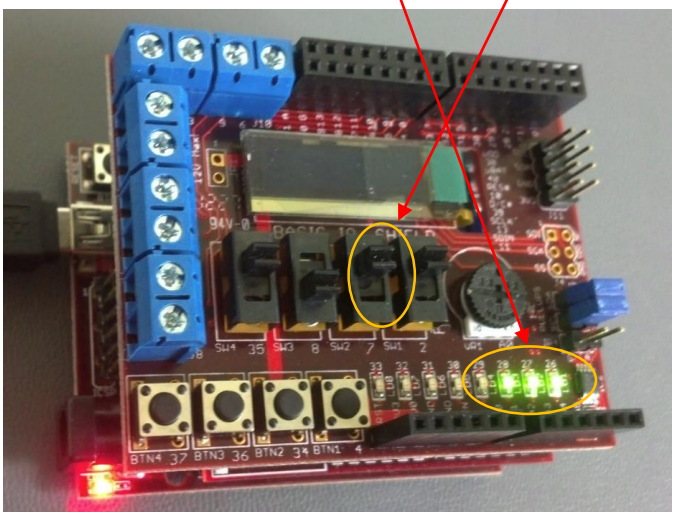
16. In Internet Explorer, follow the link to “Posting a Form”



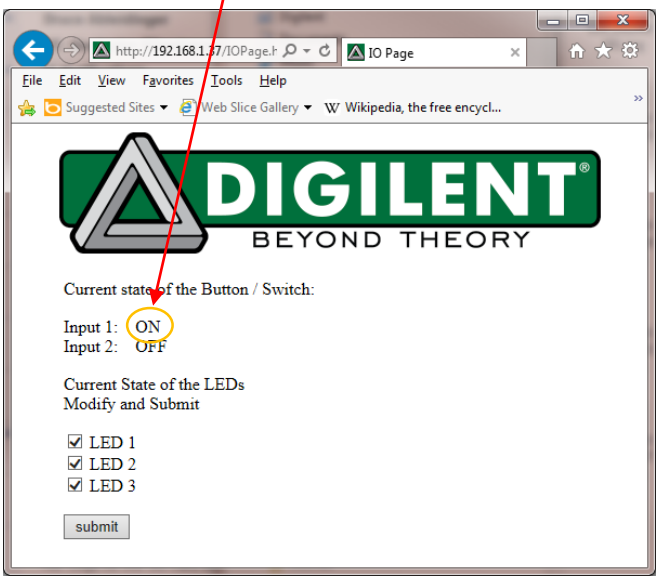
17. Check all of the LEDs ON and submit the Post.



18. Notice LEDs 1, 2, and 3 came ON. Turn Switch 2 ON and hit submit again in Internet Explorer and see how the state of Input 1 changes on the web page.

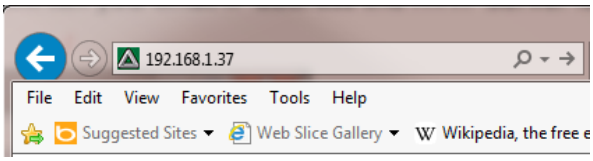


19. Input 1 updates to ON because switch 2 is ON. Likewise, if you turn switch 3 ON, Input 2 would update to ON; you must hit the submit button before the page will update.

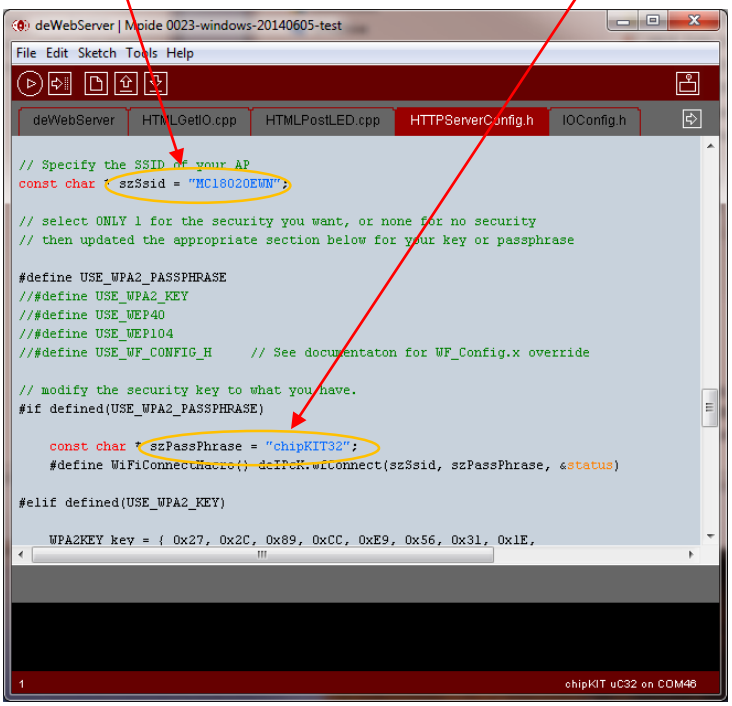


You can toggle switches 2 and 3 and/or change the state of LEDs 1,2,& 3 and the page and board will update each time you hit submit. This is a dynamically created page that is created on the fly by the HTTP Server based on the state of the switches on the uC32, or LEDs selected on the web page.

20. You can get back to the main page by typing in the original IP address in Internet Explorer's address line. From the main page you can explore the other pages hosted by the uC32 HTTP Server.

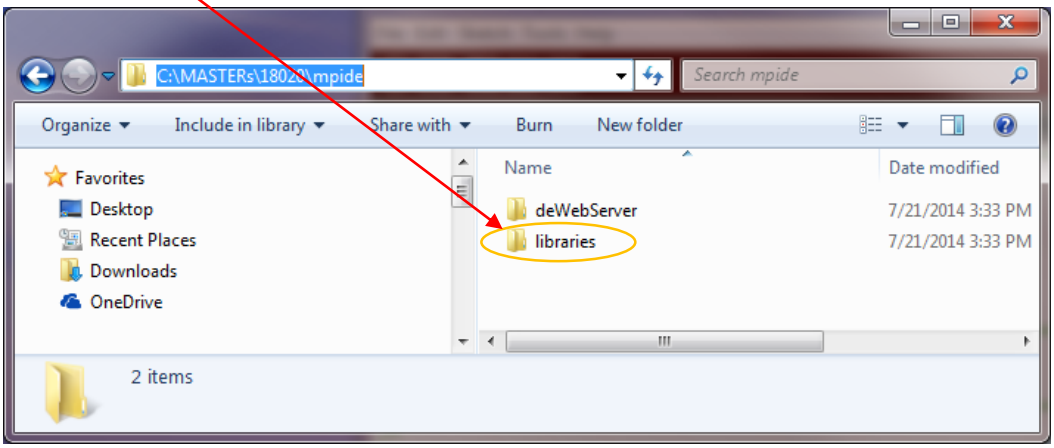


21. In HTTPServerConfig.h is where the network parameters are set. For this exercise we preset the SSID to “MC18020EWN” and the WPA2 passphrase to “chipKIT32” as this is how our router is set up.

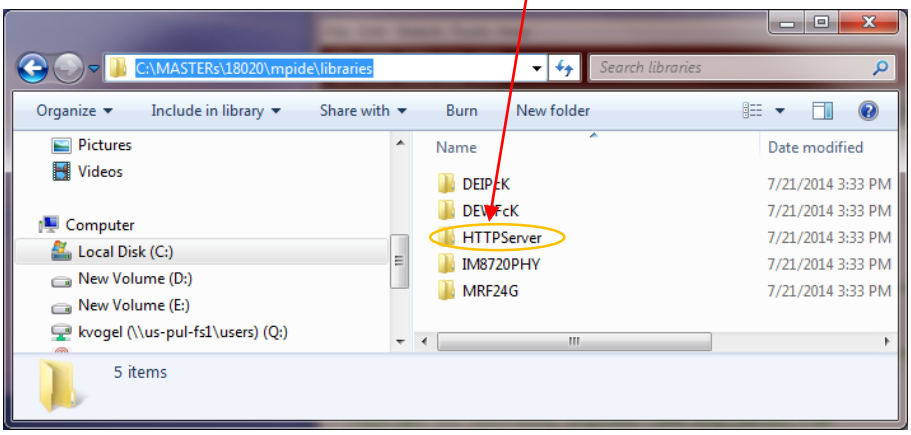


22. To see how the DEIPcK Network Stack code is used by the HTTP Server, you will need to look into the HTTPServer library source file ProcessServer.cpp. While looking at this source, please do not modify the source as it is part of the HTTPServer library code which we will be using throughout the class.

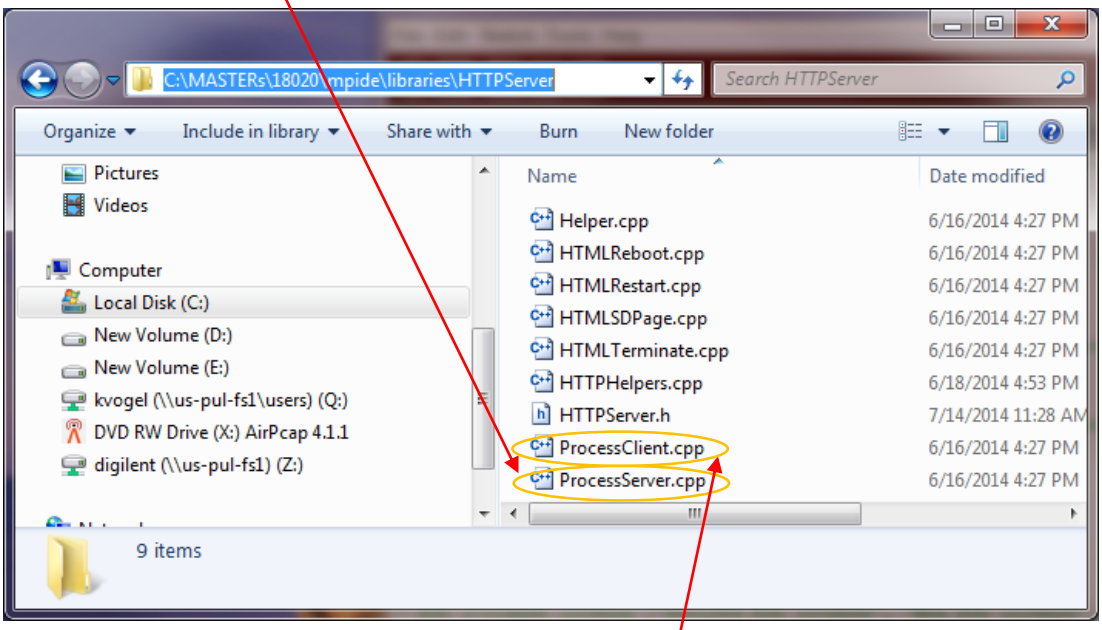
23. The HTTPServer library is a 3rd party library, you need to go to the libraries subdirectory under your Sketchbook directory; this is where 3rd part libraries are installed.



24. In the libraries subdirectory you will see the HTTPServer library, this is where all of the HTTP Server library code exists. The other libraries are part of the DEIPcK Network libraries.

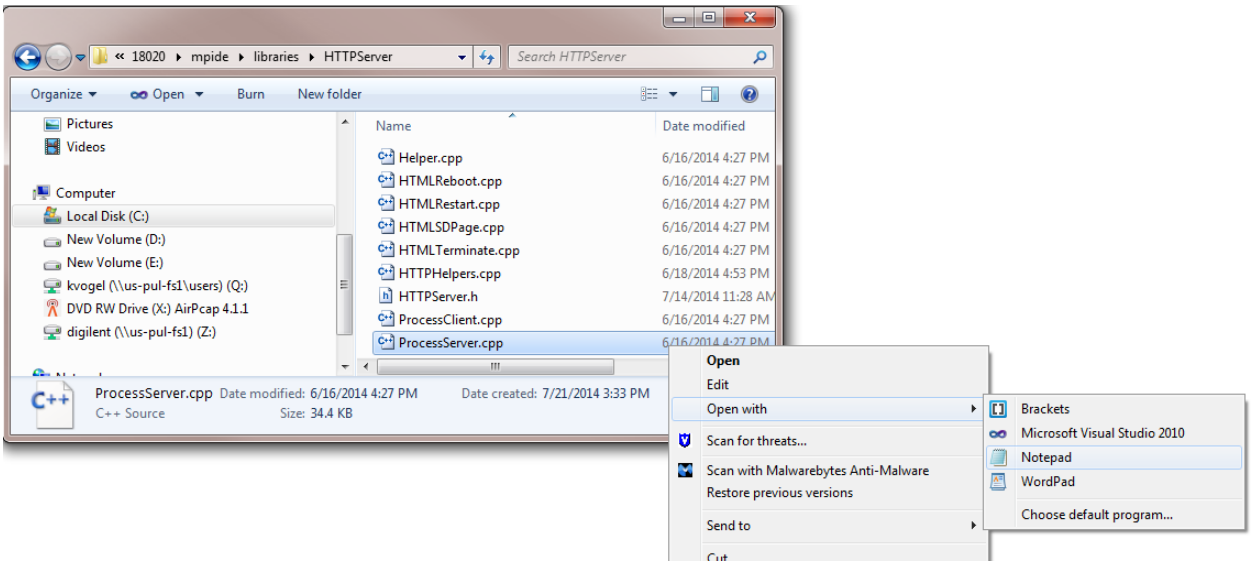


25. There you will find ProcessServer.cpp, this file contains the DEIPcK HTTP Server Network code.

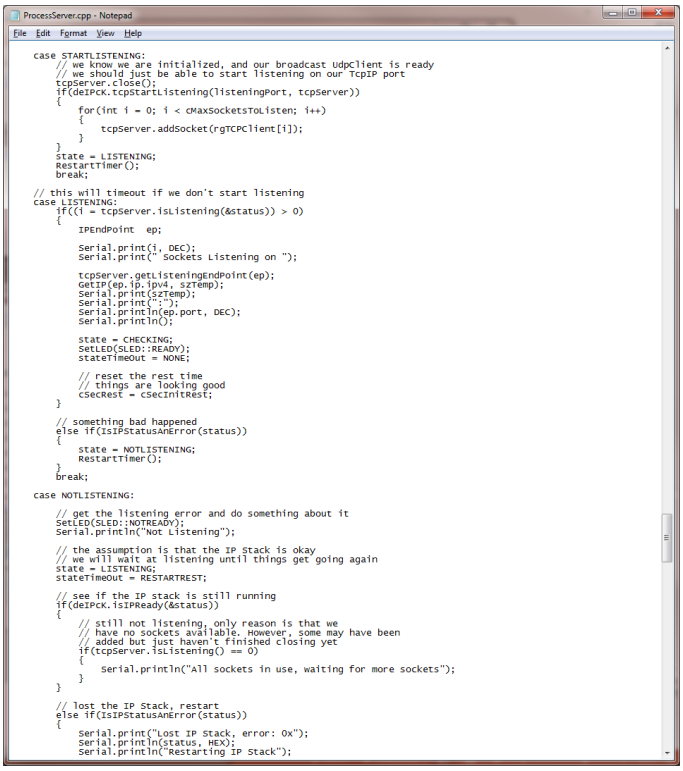


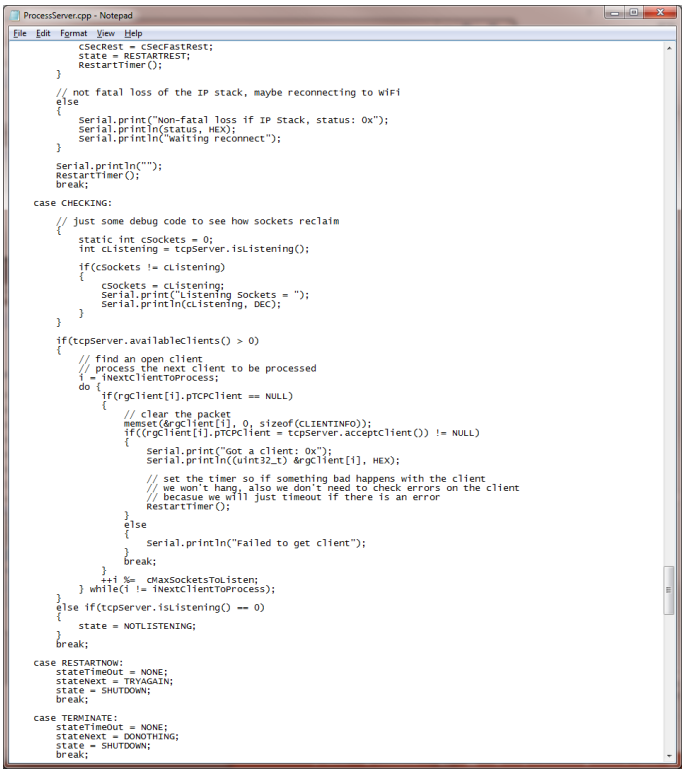
Also in ProcessClient.cpp is where the socket management code is implemented.

26. You can open these source files with notepad and take a look at the code. But please be careful not to edit the code as this is part of the library that we will be using.



27. In Process Server.cpp, review the states from STARTLISTENING to CHECKING; this is an example of how to use the DEIPcK server code.





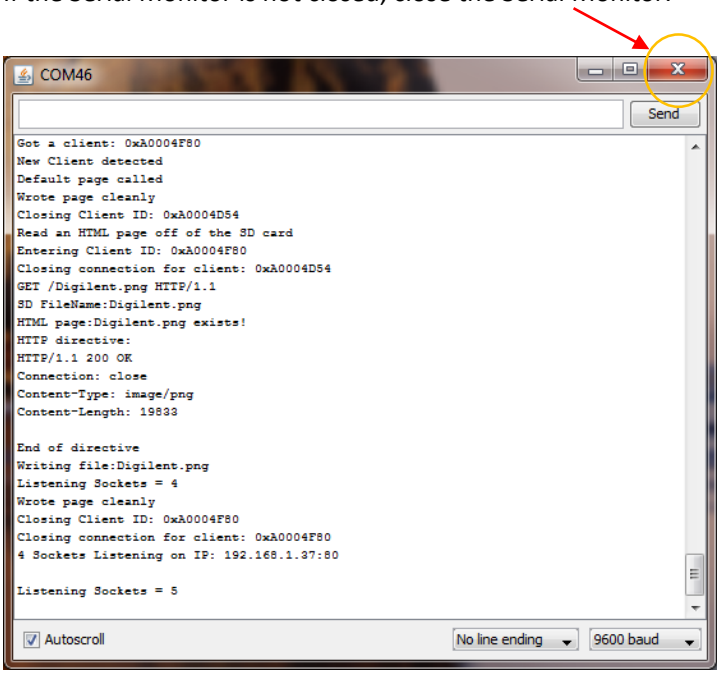
END LAB 1

LAB 2

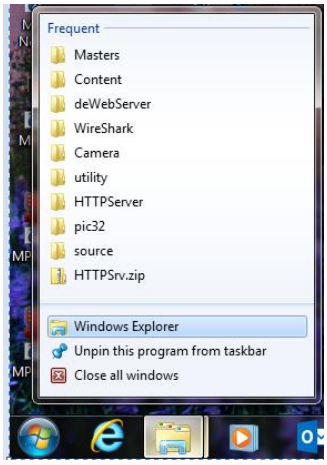
Working with Static HTML Pages

1. In this exercise we will update the HTML content on the µSD card, adding a new static HTML page to our server; but will not recompile or reload a new sketch on to the chipKIT™ uC32.

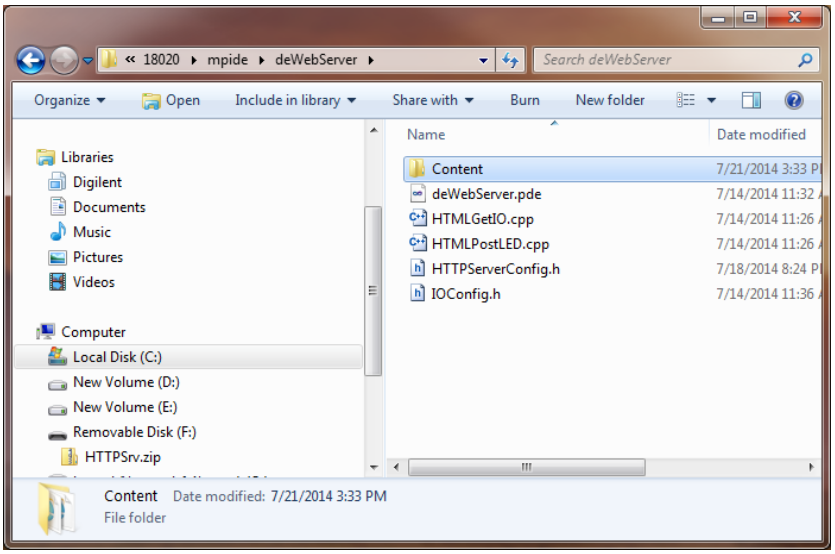
2. If the Serial Monitor is not closed, close the Serial Monitor.



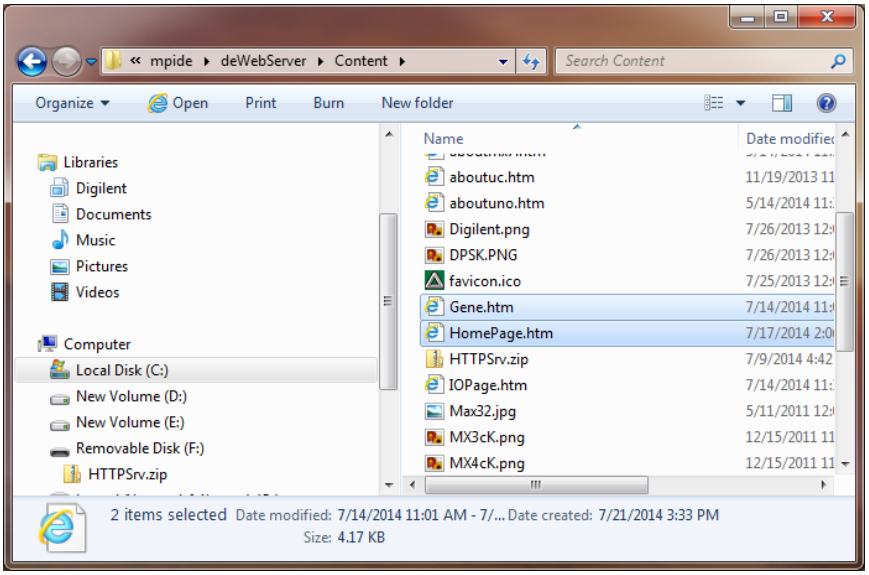
3. Open a Windows Explorer window



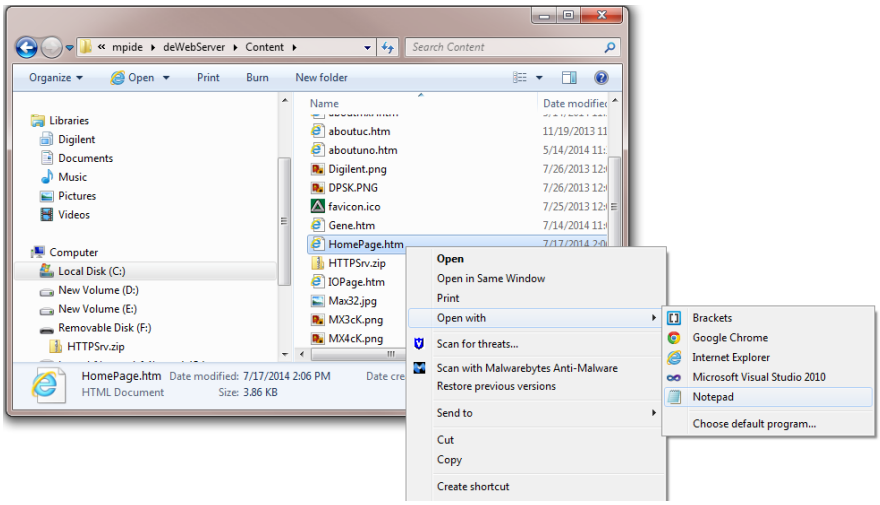
4. Go to the deWebServer Sketch Directory. Under that directory is a subdirectory “Content”. This subdirectory is not used by MPIDE or compiled in any way. It is only a subdirectory that contains the HTML content that is copied to the µSD card. Except for one file, Gene.htm, this is exactly what is currently on your µSD card.



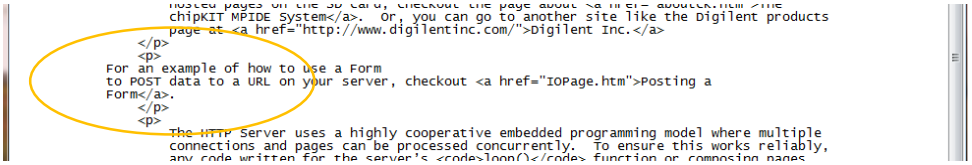
5. In that subdirectory we are going to modify HomePage.htm to reference Gene.htm. Gene.htm is exactly the html page as shown in the presentation. When your current µSD card was created, Gene.htm was not put on your card as nothing referenced it. However, now we are going to add a reference to that page from our Home Page, HomePage.htm.



6. Open HomePage.htm with Notepad.

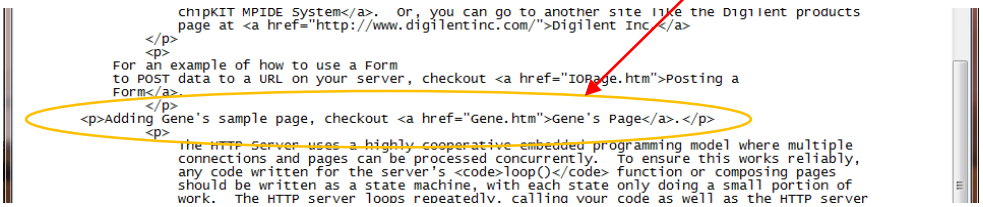


7. Scroll down into the file to where you see the break in the indentation of the text.

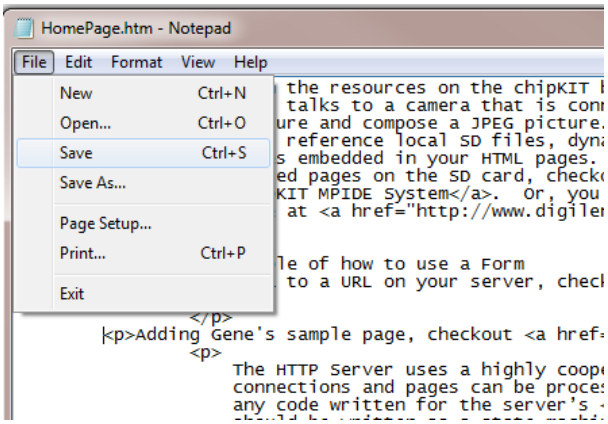


8. Modify HomePage.htm and add the href to Gene.htm page. You can use the following HTML to cut and paste into the text file. We are adding a new paragraph, so ensure you put it between the existing paragraph tags.

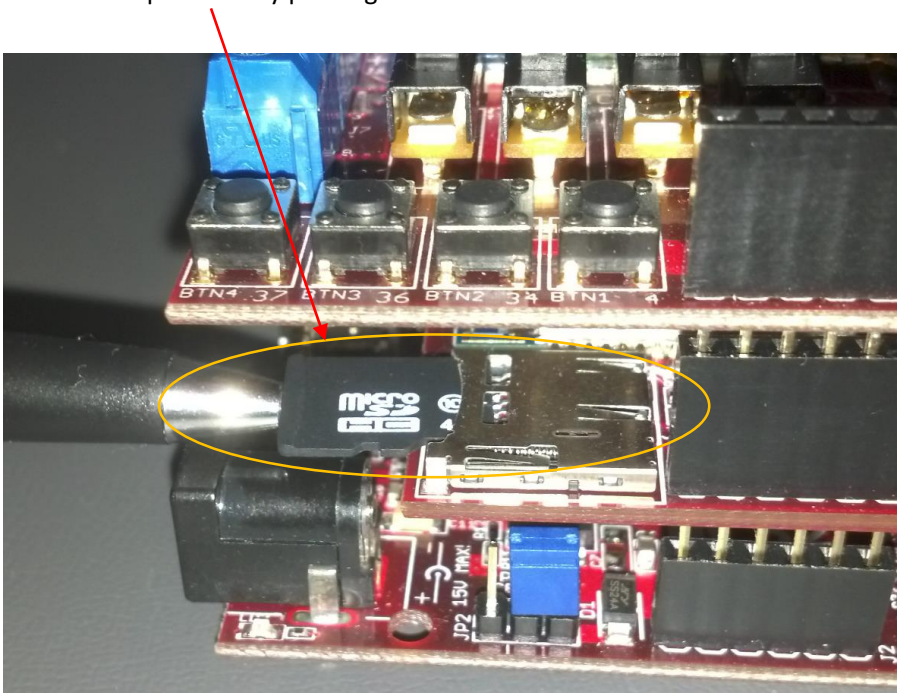
Adding Gene's sample page, checkout Gene's Page.



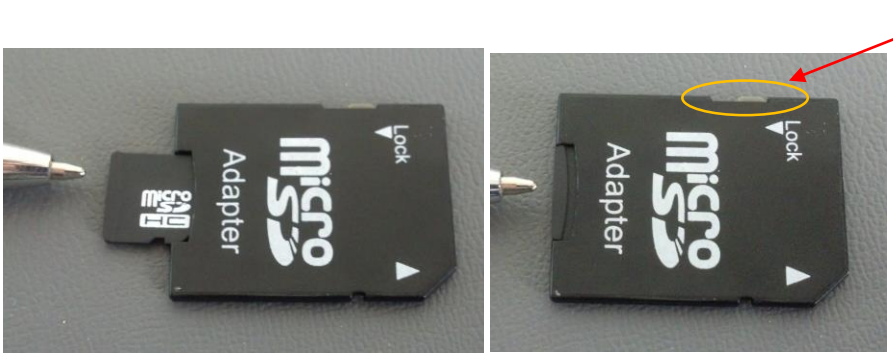
9. Save and exit from Notepad



10. Remove the µSD card by pushing in on the card until it clicks out.

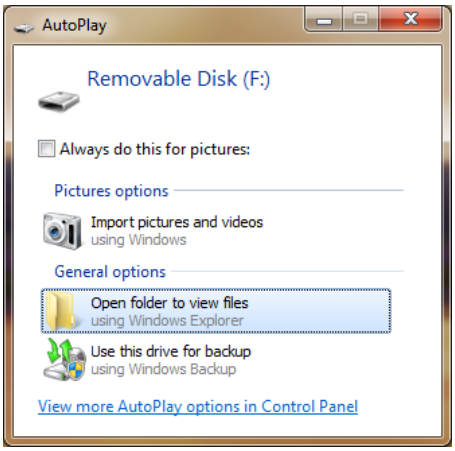


11. Insert the µSD card fully into the Micro SD Adapter. Also make sure your adapter is unlocked.

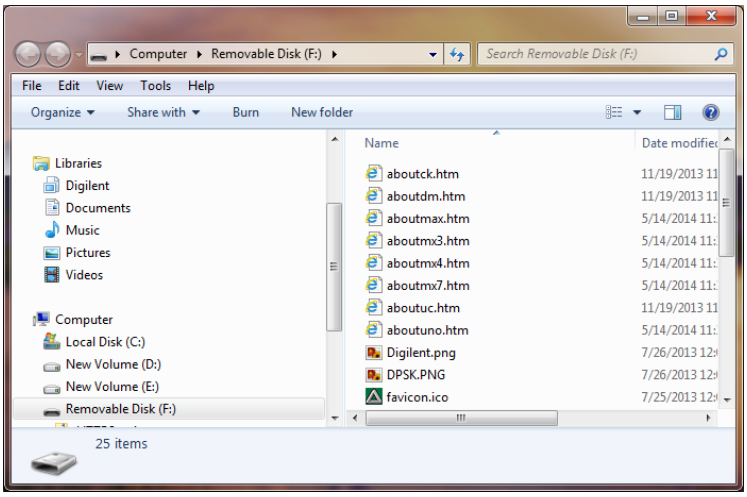


12. Insert the Micro SD Adapter fully into the SD slot on your computer.

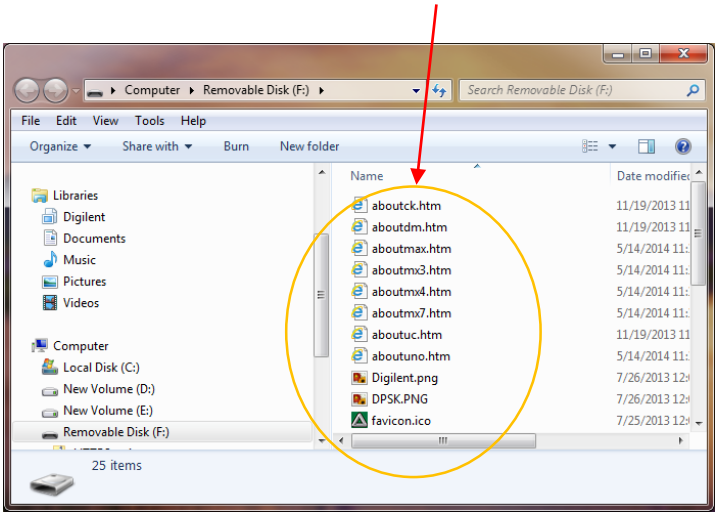
13. Windows should notify you of the Removable Disk, Open the folder and view the files.



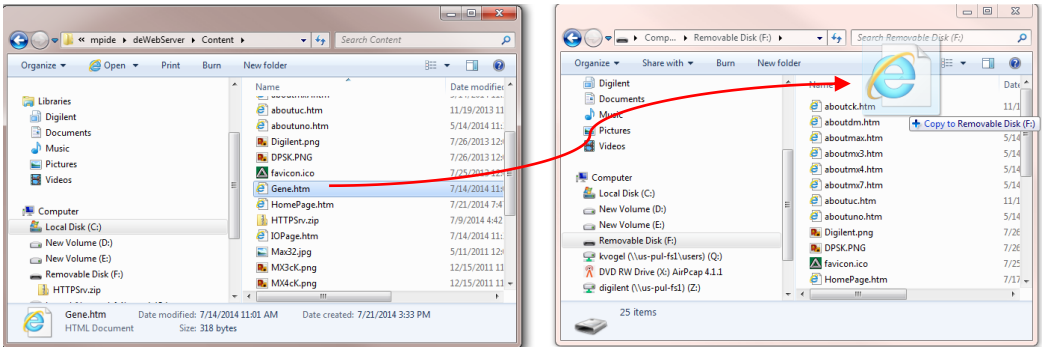
14. If windows does not AutoPlay to the above dialog, open the directory manually from Windows Explorer



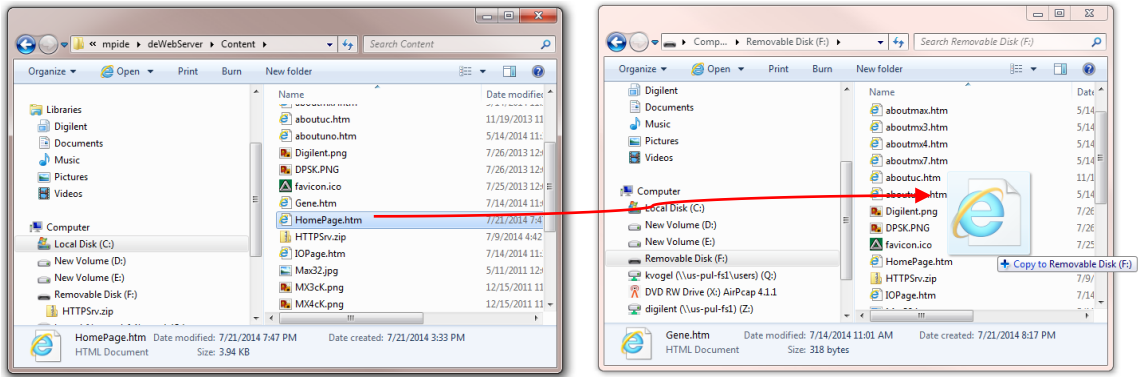
15. You should see the HTML files on the µSD card.



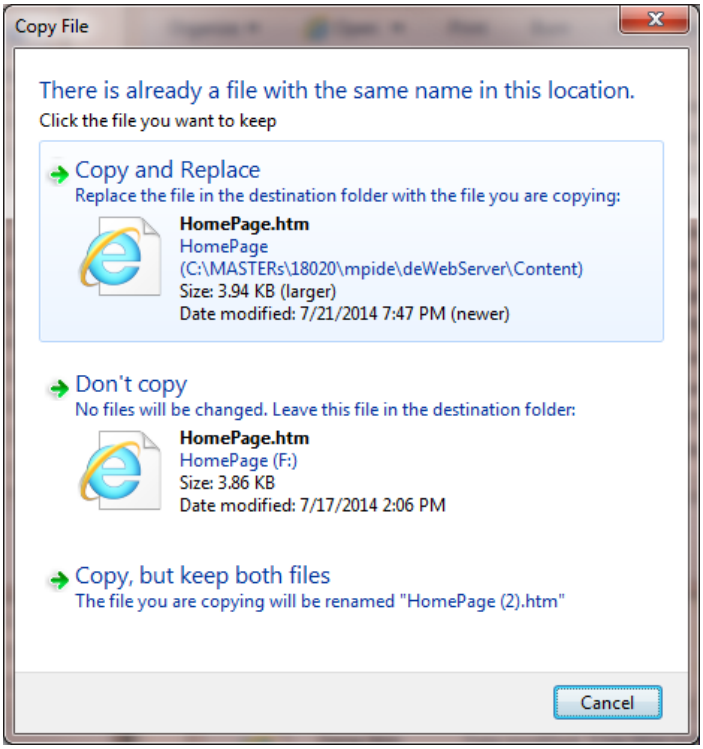
16. Now drag and drop Gene.htm from your Content directory to the root of the µSD card.



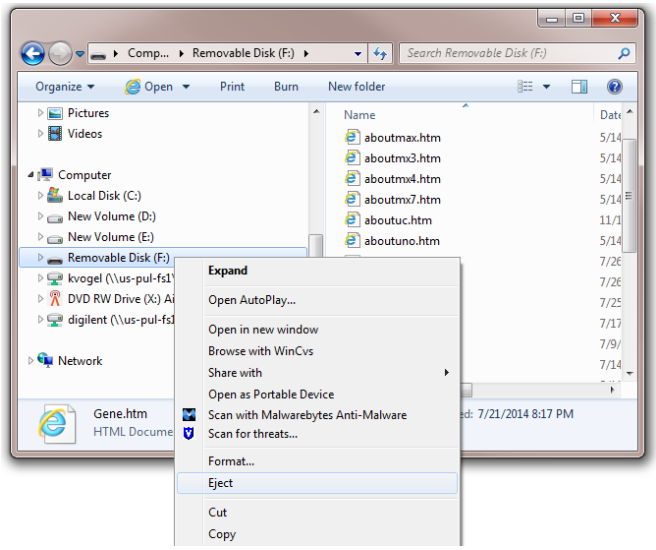
17. Do the same with HomePage.htm.



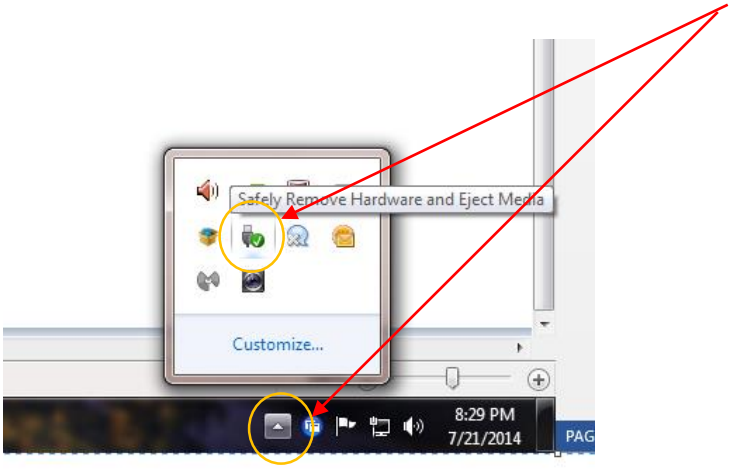
18. Remember that HomePage.htm already existed on the µSD card, so you will need to replace the file.



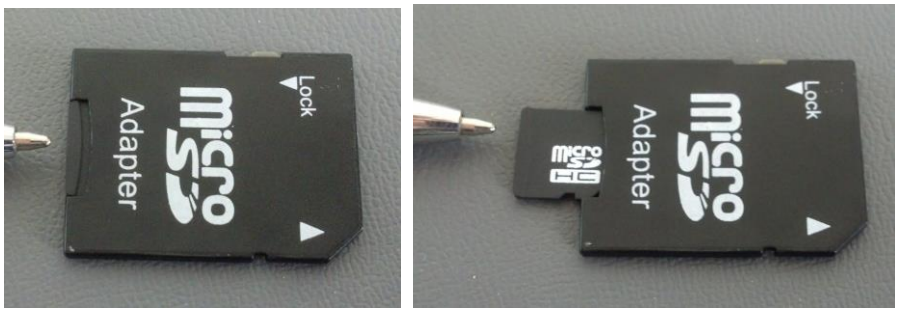
19. Remember to Eject the Removable µSD card to ensure all data was written to the card.



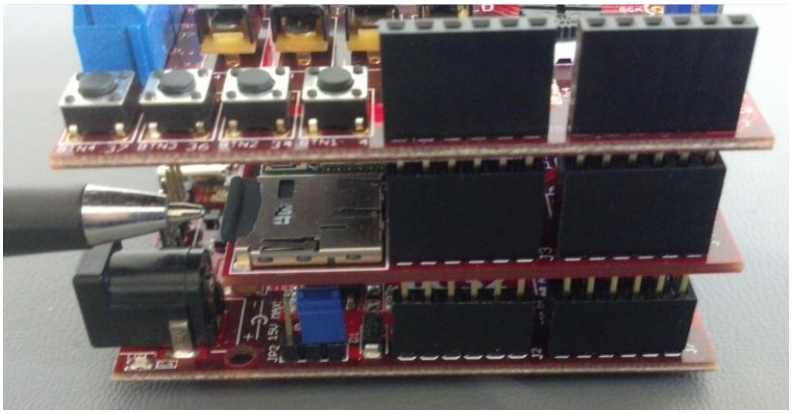
20. If you don't have an Eject menu item, then eject the hardware safely with the hardware manager.



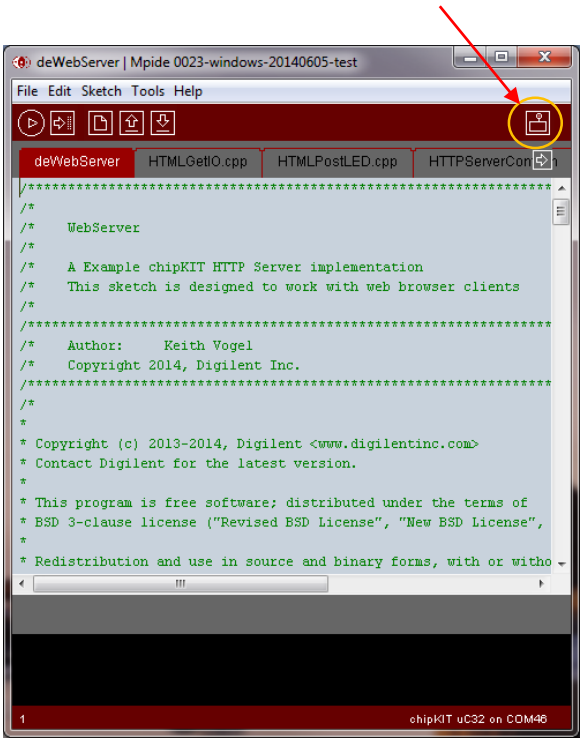
21. Remove the Micro SD Adapter from the computer and remove the µSD card from the Micro SD Adapter.



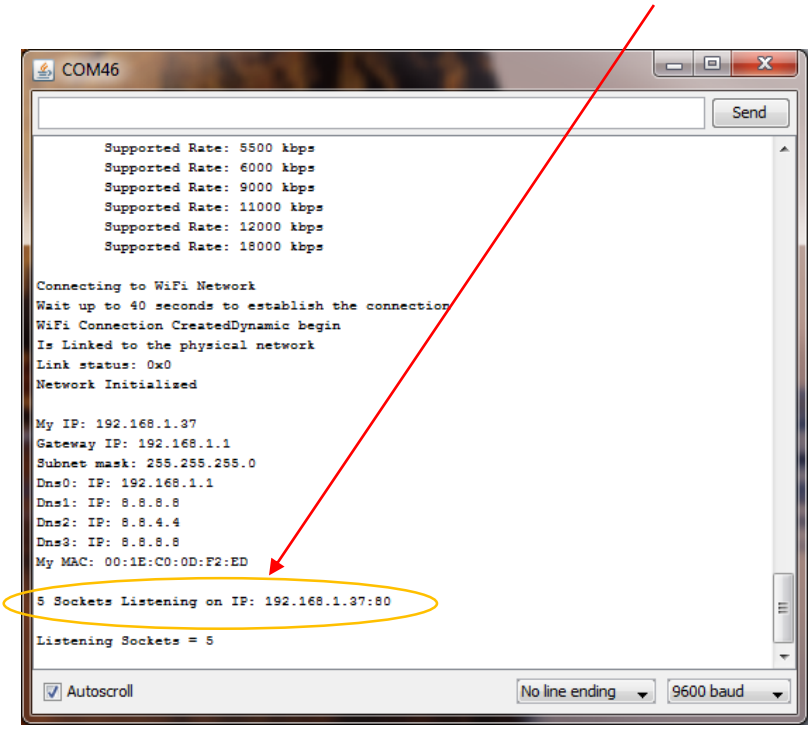
22. Fully reinserted the µSD card into the chipKIT™ uC32 µSD reader.



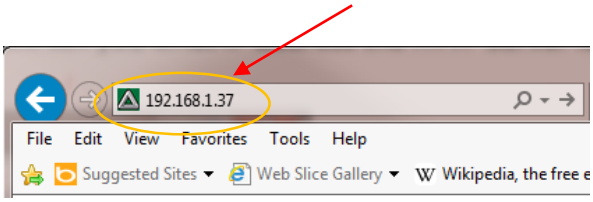
23. Restart the Serial Monitor. It is not necessary to recompile or upload the sketch as the HTTP Server code did not change. All that we did was update the static HTML pages on the µSD card.



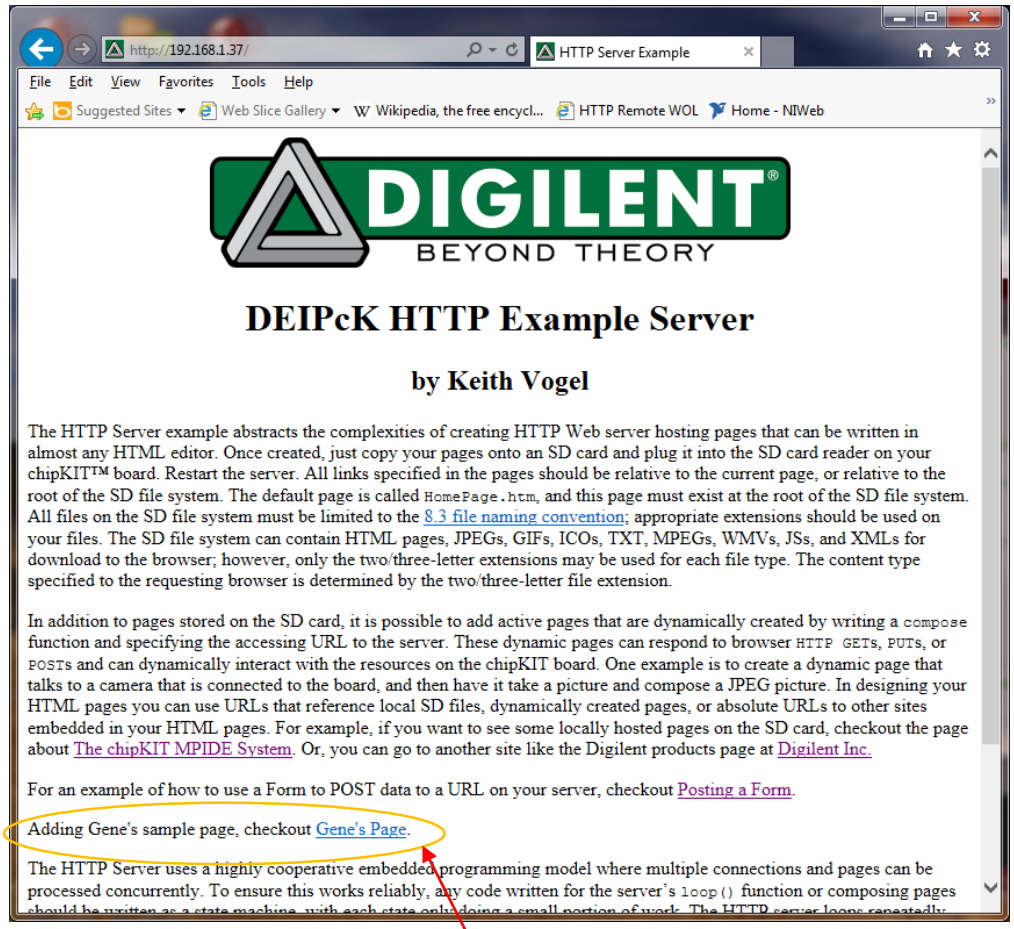
24. You should see the HTTP Server start as before, wait for the IP address so you know what to type into the browser address line; it should be the same address as before but does not have to be.



25. Start Internet Explorer and type in the IP Address in the Address line and enter.

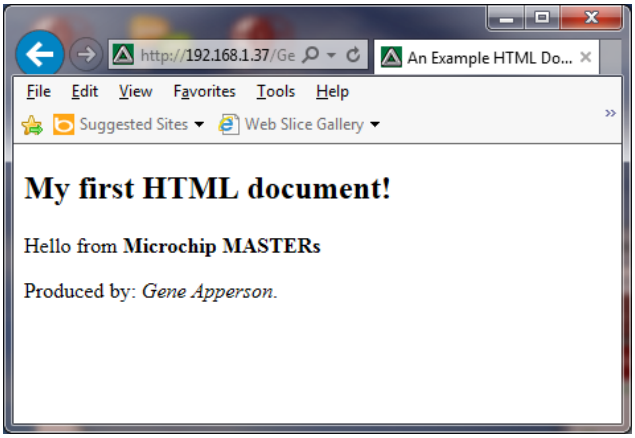


26. Internet Explorer should load the HomePage.htm as before.



But now we have a new reference to Gene's Page.

27. Click on the link and we get the new page.



28. We have added a new static HTML page without updating the HTTP Server code.

END LAB 2

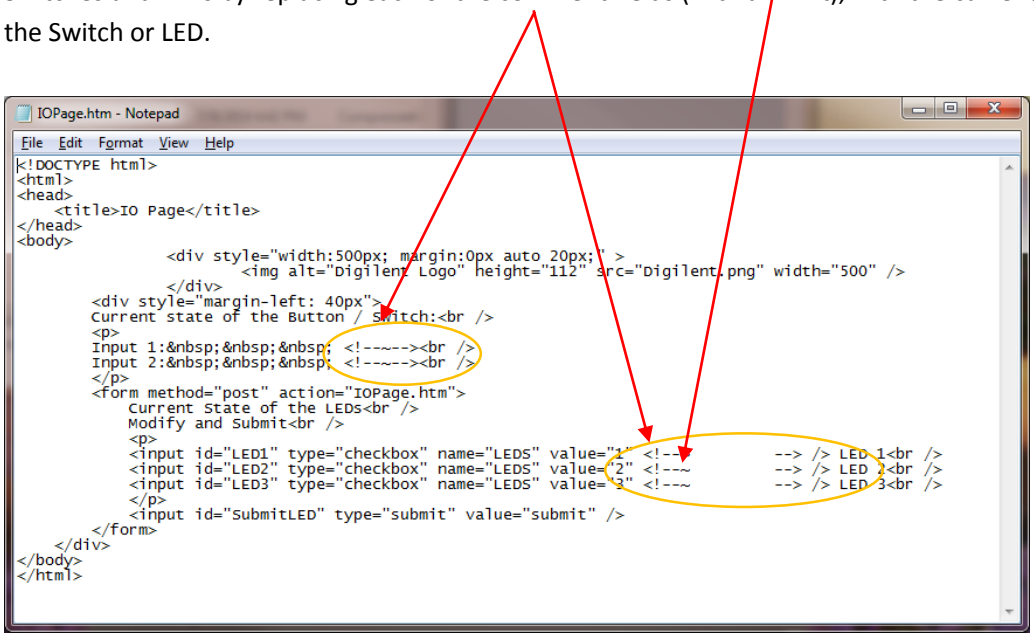
LAB 3

Working with Dynamic HTML Pages

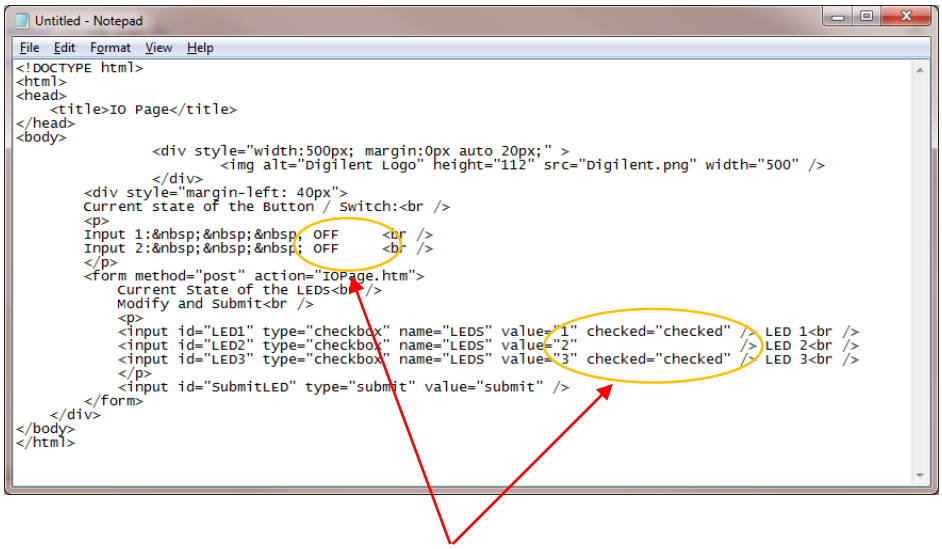
1. In this exercise we will see how to work with Compose Functions and we will update the web server to set the state of all 8 LEDs on the Basic IO Shield. There are 2 pages to deal with on the same URL. One page to POST an LED change to the URL (implemented in HTMLPostLED.cpp), and a second page to GET the state of the switches and buttons off of the URL (implemented in HTMLGetIO.cpp). Both of these pages operate on the same URL but different HTTP Methods, POST v.s. GET. There are many ways to implement a Compose Function, this is only an example.

2. IOPage.htm is our URL resource that is both the Page that displays the current state of the uC32 Switches and LEDS and is also the same resource URL that gets modified when you POST which LEDs to turn ON or OFF. We do a GET /IOPage.htm to see what the current state of the Switches and LEDs are, and a POST /IOPage.htm to set the LEDs.

3. IOPage.htm is a template HTML page read off of the µSD card by the ComposeHTMLGetIO() Compose Function. The Compose Function will dynamically update the current state of the Switches and LEDs by replacing each of the comment fields (with a ~ in it), with the current state of the Switch or LED.

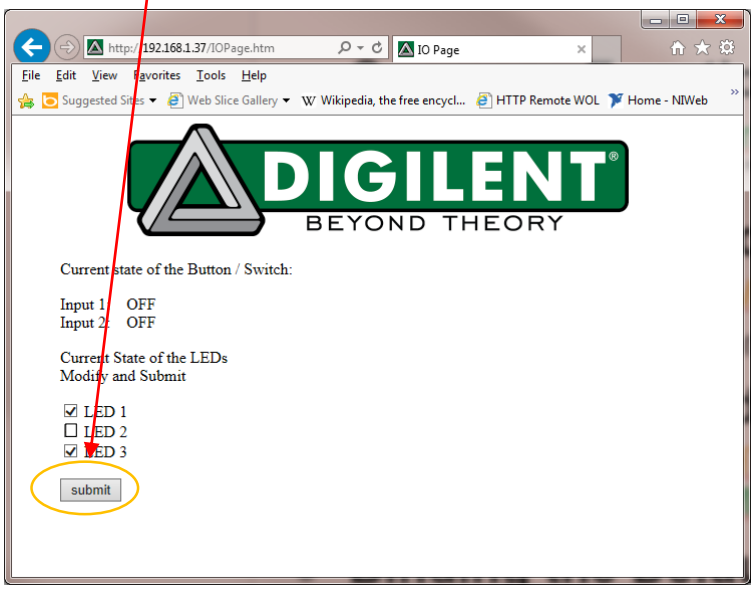


4. The actual HTML sent to the browser after HTMLGetIO renders it looks like:

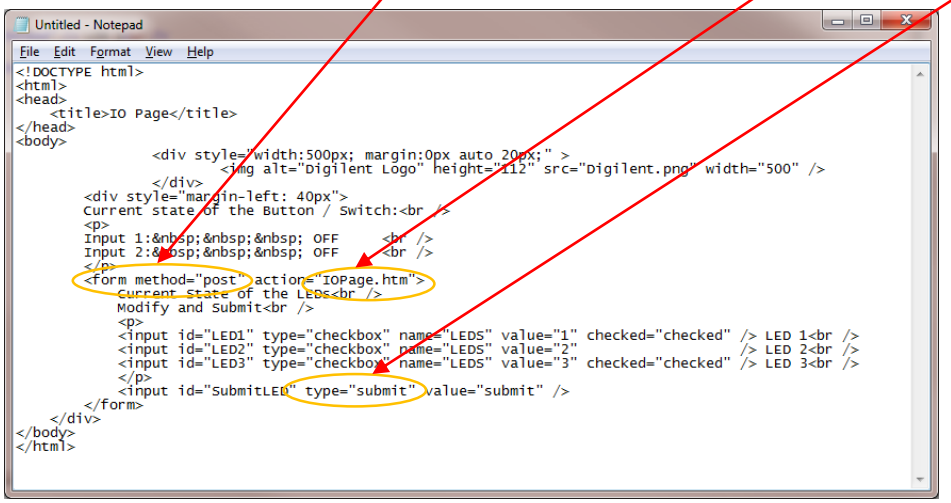


Notice how the comment fields were all replaced with the state of the Switches and LEDS. Also notice how the fields were all space filed so the actual size of the template file did not change. This allows the Compose Function to read the size of the IOPage.htm template file off of the µSD card and use the file size as the "Content Length" when generating the HTTP Header. The extra spaces are ignored and have no relevance in the HTML rendering.

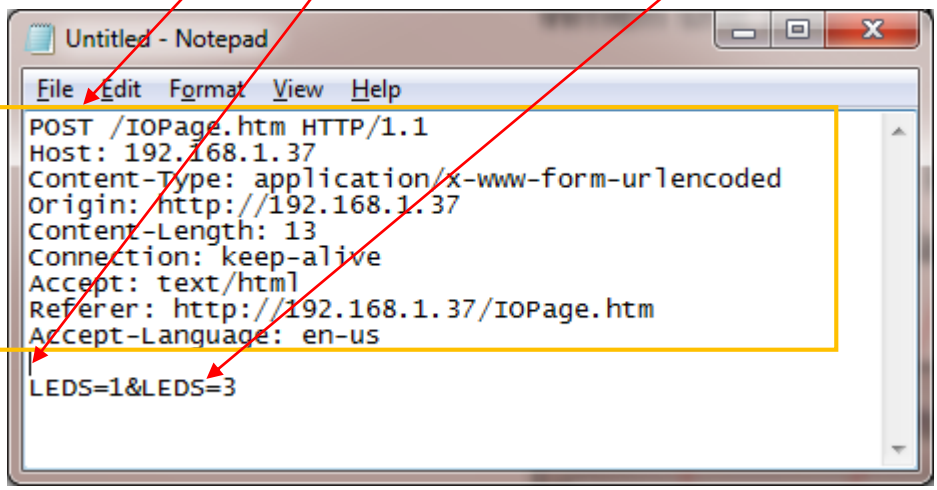
5. When we submit the page, we POST a FORM to the same URL resource, IOPage.htm. This tells the uC32 server to update that resource with the new LED status.



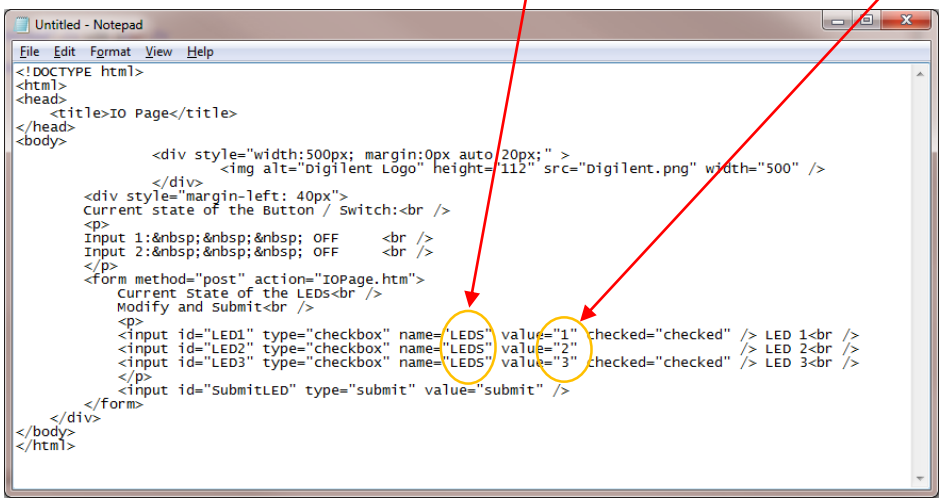
6. You can see in the HTML, the FORM tag and resource to update (IOPage.htm), and the submit.



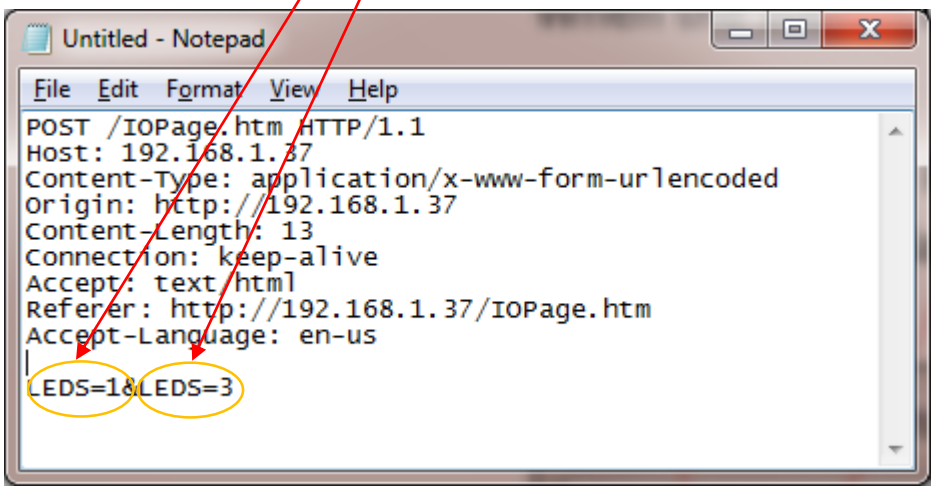
7. When the Form is POSTed, something like the following will be sent from the browser to the uC32. Notice the HTTP header, a blank line, and then the posted FORM data in the HTTP Body.



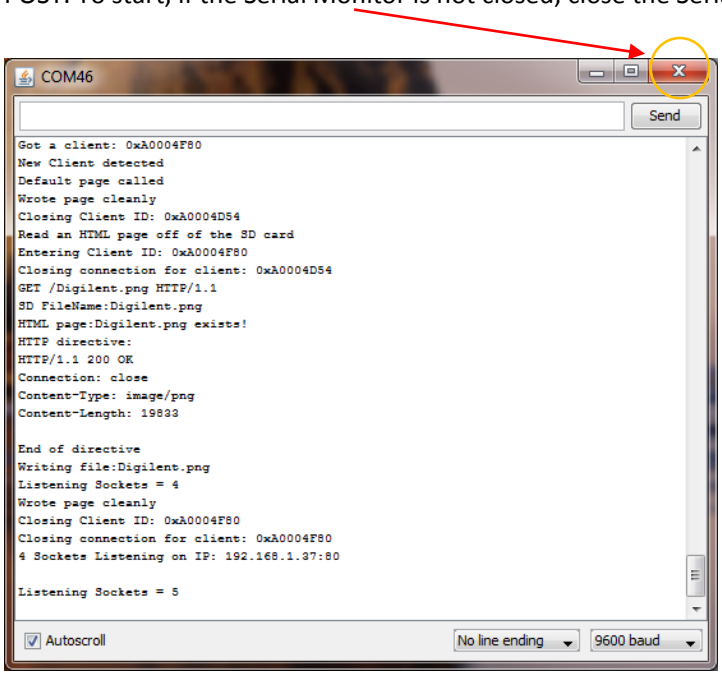
8. The POSTed data will give us all of the names of active items in the FORM, and their active values. In our case we only have 1 item in the FORM (LEDS), and their potential values (1, 2 and 3).



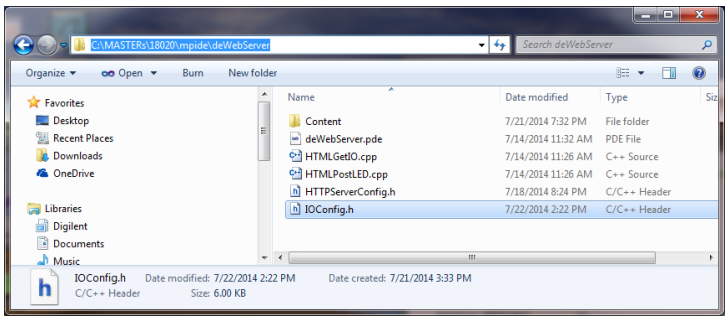
9. Here is what our Form POST looks like, we can see that the FORM POSTed that the “LEDS” item is active with active values of 1 & 3; or more specifically LED1 and LED3 are checked.



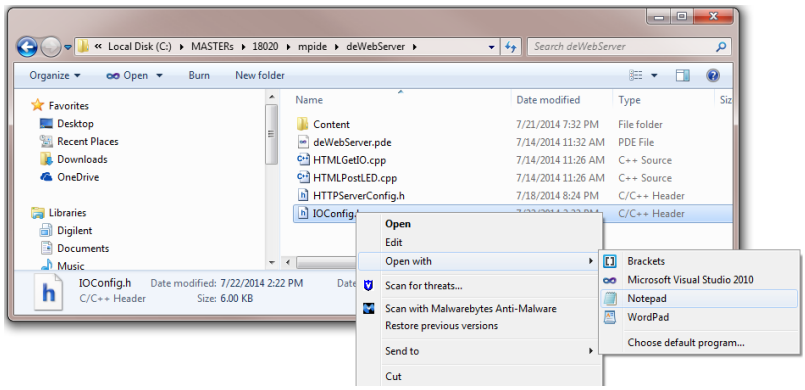
10. Now let's update our server to allow for all of the LEDs on the Basic IOShield to be modified by the POST. To start, if the Serial Monitor is not closed, close the Serial Monitor.



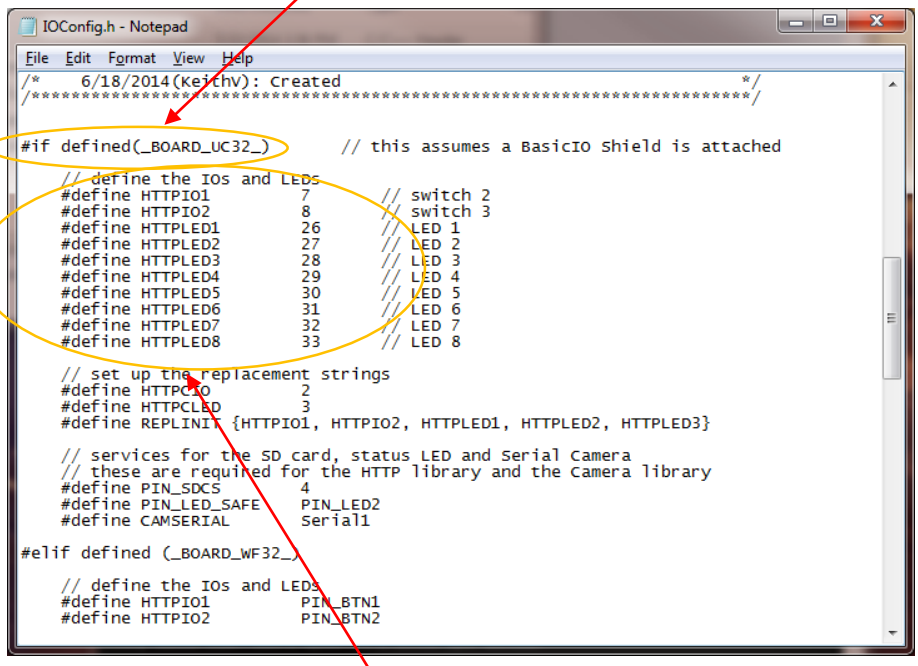
11. Navigate to the deWebServer Sketch folder, C:\MASTERs\18020\mpide\deWebServer



12. Open IOConfig.h with Notepad

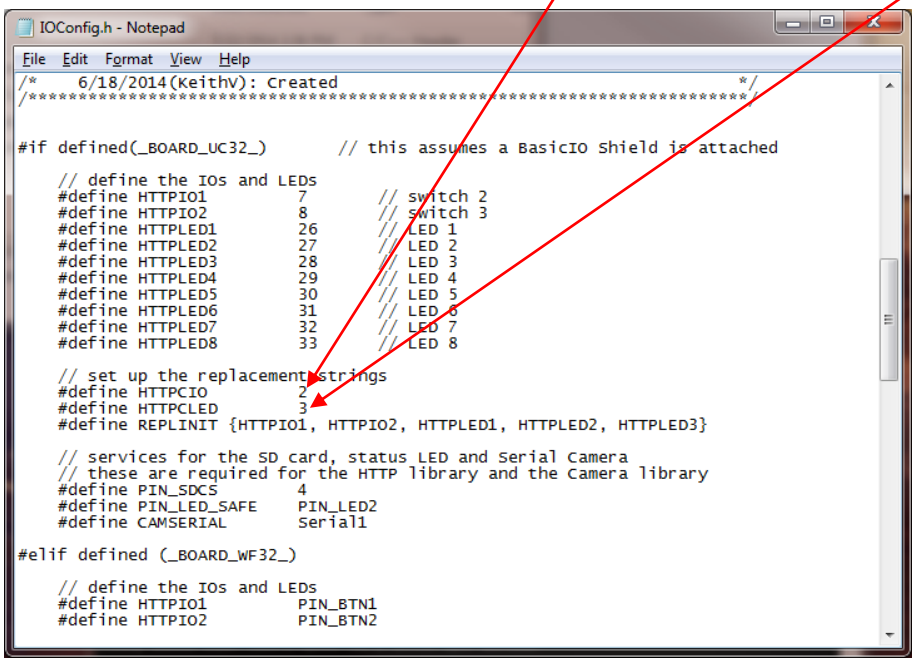


13. This file contains the chipKIT™ pin outs for the LEDs, Switches, and Buttons based on the specific board in use. Scroll to the section for the uC32, that is, "_BOARD_UC32_".

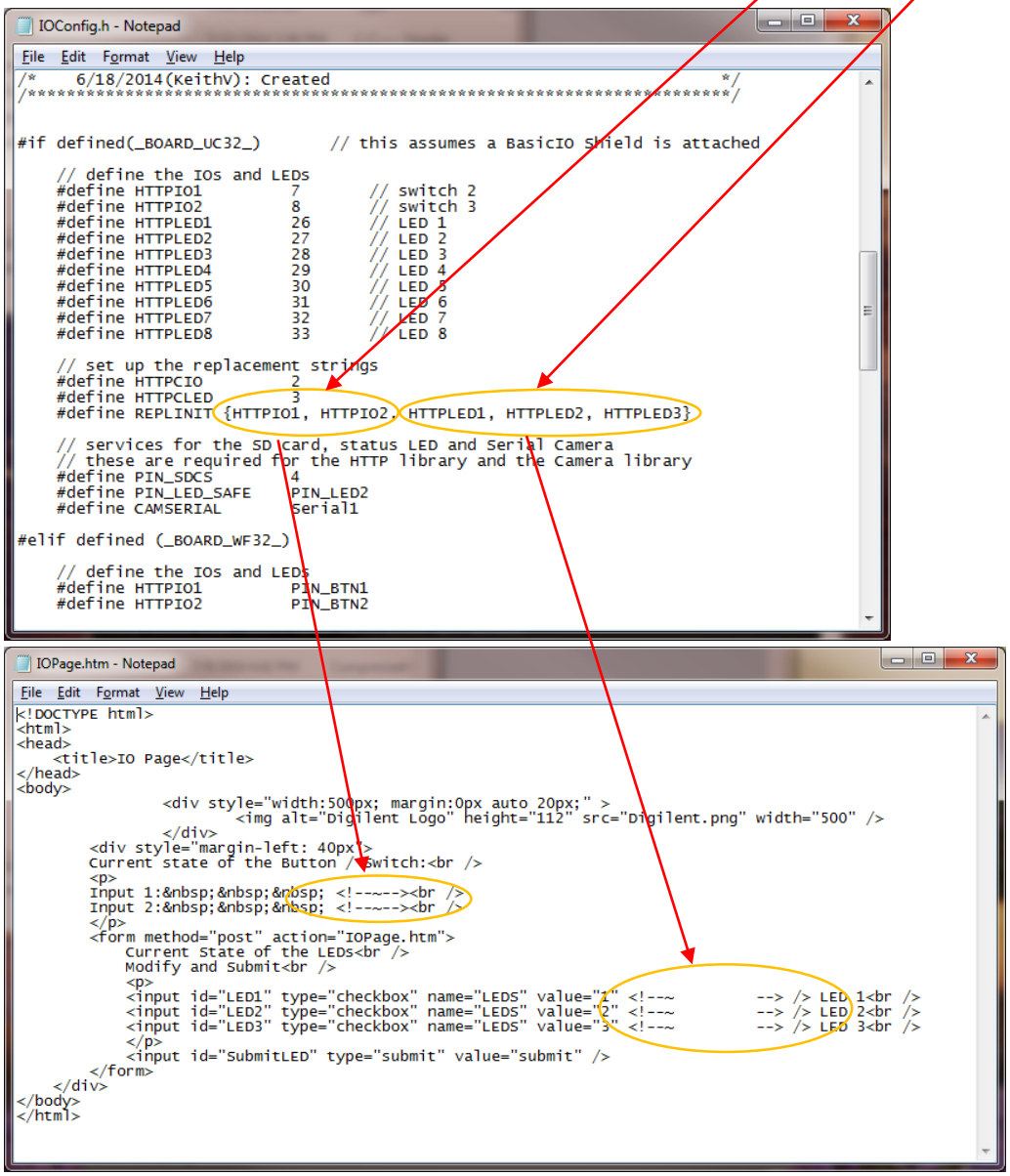


Here are the pin defines for the Switches and LEDs.

14. We must tell the Compose Functions how many switches we have, and how many LEDs we have.



15. The HTMLGetIO Compose Function is written to work off of an ordered list of switches and LEDs, in the same order as the comments in our template IOPage.htm file. We display the switches first and then the LEDs. We currently have 2 switches and 3 LEDs.



IOConfig.h - Notepad

File Edit Format View Help

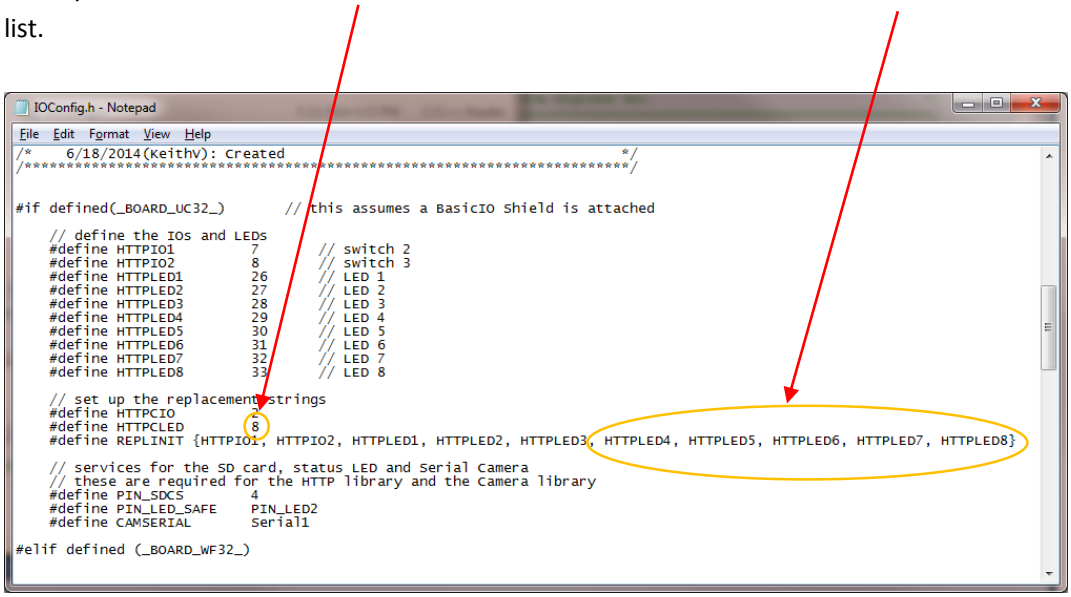
```
/*    6/18/2014(KeithV): Created                                              */
/*************************************************************************/


#if defined(_BOARD_UC32_)       // this assumes a BasicIO Shield is attached

    // define the IOs and LEDs
    #define HTTPIO1         7       // switch 2
    #define HTTPIO2         8       // switch 3
    #define HTTPLED1        26      // LED 1
    #define HTTPLED2        27      // LED 2
    #define HTTPLED3        28      // LED 3
    #define HTTPLED4        29      // LED 4
    #define HTTPLED5        30      // LED 5
    #define HTTPLED6        31      // LED 6
    #define HTTPLED7        32      // LED 7
    #define HTTPLED8        33      // LED 8

    // set up the replacement strings
    #define HTTPCIO         2
    #define HTTPCLED        3
    #define REPLINIT {HTTPIO1, HTTPIO2, HTTPLED1, HTTPLED2, HTTPLED3}

    // services for the SD card, status LED and Serial Camera
    // these are required for the HTTP library and the Camera library
    #define PIN_SDCS        4
    #define PIN_LED_SAFE    PIN_LED2
    #define CAMSERIAL       Serial1

#elif defined (_BOARD_WF32_)

    // define the IOs and LEDs
    #define HTTPIO1         PIN_BTN1
    #define HTTPIO2         PIN_BTN2
```

IOPage.htm - Notepad

File Edit Format View Help

```
<!DOCTYPE html>
<html>
<head>
    <title>IO Page</title>
</head>
<body>
                <div style="width:500px; margin:0px auto 20px;" >
                    <img alt="Digilent Logo" height="112" src="Digilent.png" width="500" />
                </div>
        <div style="margin-left: 40px;">
        Current state of the Button / Switch:<br />
        <p>
        Input 1:&amp;nbsp;&amp;nbsp;&amp;nbsp; <!--~~~><br />
        Input 2:&amp;nbsp;&amp;nbsp;&amp;nbsp; <!--~~~><br />
        </p>
        <form method="post" action="IOPage.htm">
            Current State of the LEDs<br />
            Modify and Submit<br />
            <p>
            <input id="LED1" type="checkbox" name="LEDS" value="1" <!--~~          --> /> LED 1<br />
            <input id="LED2" type="checkbox" name="LEDS" value="2" <!--~~          --> /> LED 2<br />
            <input id="LED3" type="checkbox" name="LEDS" value="3" <!--~~          --> /> LED 3<br />
            </p>
            <input id="SubmitLED" type="submit" value="submit" />
        </form>
    </div>
</body>
</html>
```

16. Modify the defines to indicate 8 LEDs, and add the additional Basic IO Shield LEDs to the ordered list.

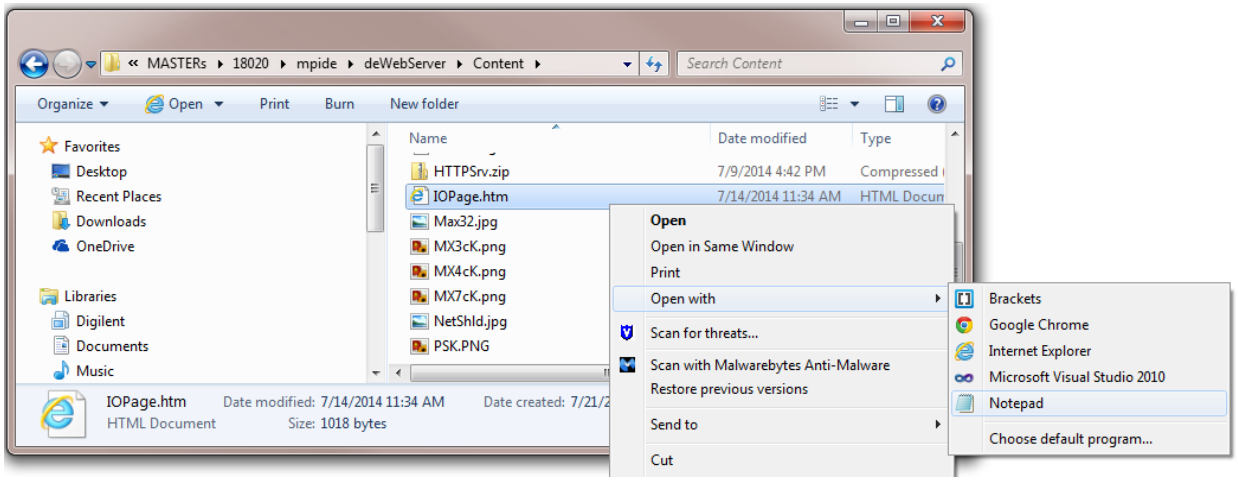


For your convenience, you can cut and paste the following text into the file.

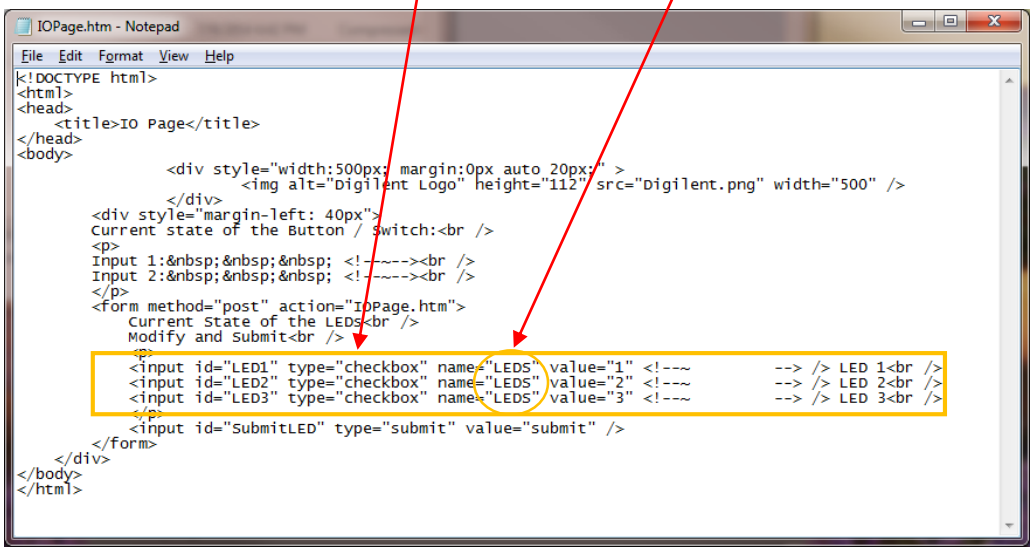
```
#define HTTPCLED      8
#define REPLINIT {HTTPIO1, HTTPIO2, HTTPLED1, HTTPLED2, HTTPLED3, HTTPLED4, HTTPLED5, HTTPLED6, HTTPLED7, HTTPLED8}
```

Save IOConfig.h and exit Notepad.

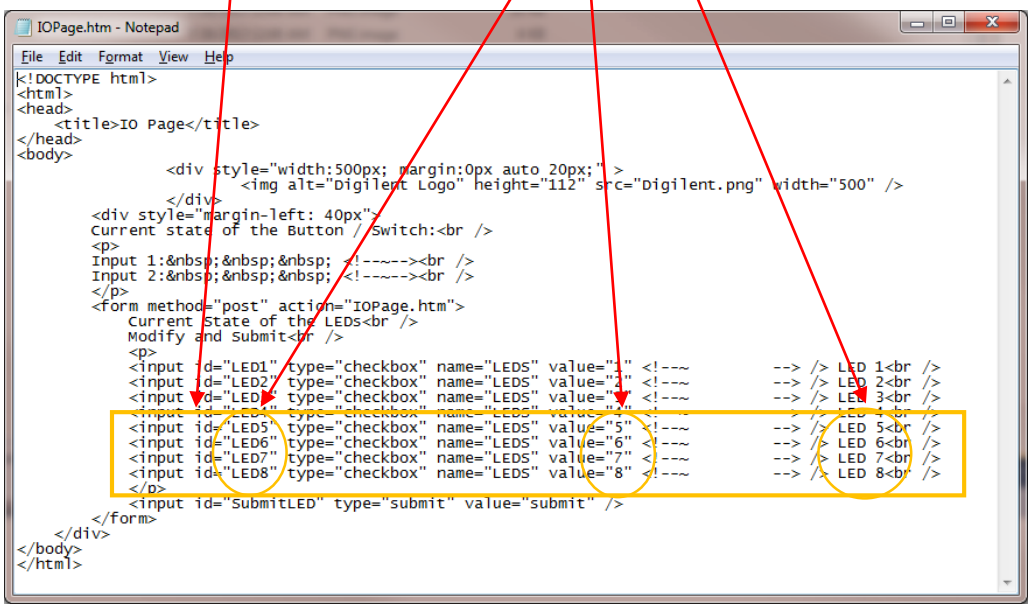
17. Go down into the “Content” subdirectory and open IOPage.htm with Notepad



18. In IOPage.htm we currently support 3 LEDs as part of the “LEDS” items.



19. We need to create 8 entries, and update the id, value, and text.

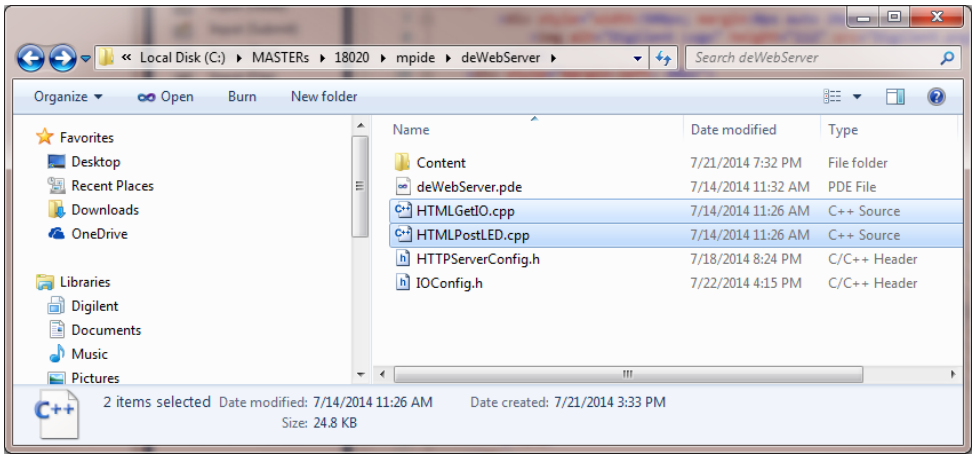


To help out, you can copy and paste the following lines

```
<input id="LED4" type="checkbox" name="LEDS" value="4" <!--~~ --> /> LED 4<br />
<input id="LED5" type="checkbox" name="LEDS" value="5" <!--~~ --> /> LED 5<br />
<input id="LED6" type="checkbox" name="LEDS" value="6" <!--~~ --> /> LED 6<br />
<input id="LED7" type="checkbox" name="LEDS" value="7" <!--~~ --> /> LED 7<br />
<input id="LED8" type="checkbox" name="LEDS" value="8" <!--~~ --> /> LED 8<br />
```

Save IOPage.htm and exit Notepad.

20. In your sketch directory HTMLGetIO.cpp implements the ComposeHTMLGetIO() callback function that renders the IOPage.htm from the template IOPage.htm. HTMLPostLED.cpp implements the ComposeHTMLPostLED() callback function that processes the POST IOPage.htm resource. These functions were written to be generic enough to handle the Switches and LEDs as enumerated in the REPLINIT ordered list in IOConfig.h without changing the code.



21. However, for your understanding of the code, you can see there are some strings defined that are exactly the same number of characters in length as the replacement comment strings in IOPage.htm; this allows for direct replacement without change the file size.

```
static const char szReplaceStr[]    = "<!--~";
static const char szChecked[]       = "checked=\"checked\"";
static const char szUnChecked[]     = "                    ";
static const char szON[]            = "ON        ";
static const char szOFF[]           = "OFF       ";
```

The code just loops through the full number of items in the "REPLINIT" ordered list, looking for the replacement comment string. When one is found the next chipKIT™ IO pin in the ordered list is read to get the current state (this works for Buttons, Switches, and LEDs) and determines if it is ON or OFF and then replaces the comment string with the appropriate HTML replacement string.

```
if(iReplace < cReplacementStrings)
{
    if((pchLast = strstr(pchLast, szReplaceStr)) == NULL)
    {
        // oh this is not good, something is wrong with the template file
        Serial.print("Unable able to find template replacemnt on index ");
        Serial.print(iReplace, DEC);
        Serial.print(" in template file ");
        Serial.println(szIOPage);
        pClientInfo->htmlState = JMPFILENOTFOUND;
    }
    else
    {
        bool isOn = digitalRead(rgReplacePins[iReplace]);

        if(iReplace < HTTPCIO)
        {
            if(isOn)
            {
                memcpy(pchLast, szON, sizeof(szON)-1);
            }
            else
            {
                memcpy(pchLast, szOFF, sizeof(szOFF)-1);
            }
        }
        else if((iReplace - HTTPCIO) < HTTPCLED)
        {
            if(isOn)
            {
                memcpy(pchLast, szChecked, sizeof(szChecked)-1);
            }
            else
            {
                memcpy(pchLast, szUnChecked, sizeof(szUnChecked)-1);
            }
        }

        // do the next replacement string.
        iReplace++;
    }
```

22. In HTMLPostLED.cpp, the input buffer is read until the content length in the HTTP Header is found. This length is saved away in the scratch buffer (rgbOut) for later use.

```
case CONTLEN:

    // if we hit the end of the header then there was no content length
    // and we don't know how to handle that, so exit with an error
    // File not found is probably the wrong error, but it does get out out
    // Fortunately all major browsers put in the content lenght, so this
    // will almost never fail.
    if(strlen((char *) pClientInfo->rgbIn) == 0)    // cbRead may be longer than just the line, so do a strlen
    {
        return(JumpToComposeHTMLPage(pClientInfo, ComposeHTTP404Error));
    }

    // found the content lengths
    else if(memcmp((byte *) szContentLength, pClientInfo->rgbIn, sizeof(szContentLength)-1) == 0)
    {

        *((uint32_t *) pClientInfo->rgbOut) = atoi((char *) &pClientInfo->rgbIn[sizeof(szContentLength)-1]);
        pClientInfo->htmlState = ENDHDR;
    }
    retCMD = GCMD::GETLINE;
    break;
```

23. The rest of the HTTP Header is skipped until the empty line at the end of the HTTP Header is found.

```
case ENDHDR:

    // the header is ended with a double \r\n\r\n, so I will get
    // a zero length line.  Just keep reading lines until we get to the blank line
    if(strlen((char *) pClientInfo->rgbIn) == 0)    // cbRead may be longer than just the line, so do a strlen()
    {
        uint32_t i = 0;

        // go to beyond the \0
        for(i = 0; i < pClientInfo->cbRead && pClientInfo->rgbIn[i] == '\0'; i++);

        // move the buffer to the front
        pClientInfo->cbRead -= i;
        if(pClientInfo->cbRead > 0)
        {
            memcpy(pClientInfo->rgbIn, &pClientInfo->rgbIn[i], pClientInfo->cbRead);
        }

        pClientInfo->htmlState = DATA;
    }
    else
    {
        retCMD = GCMD::GETLINE;
    }
    break;
```

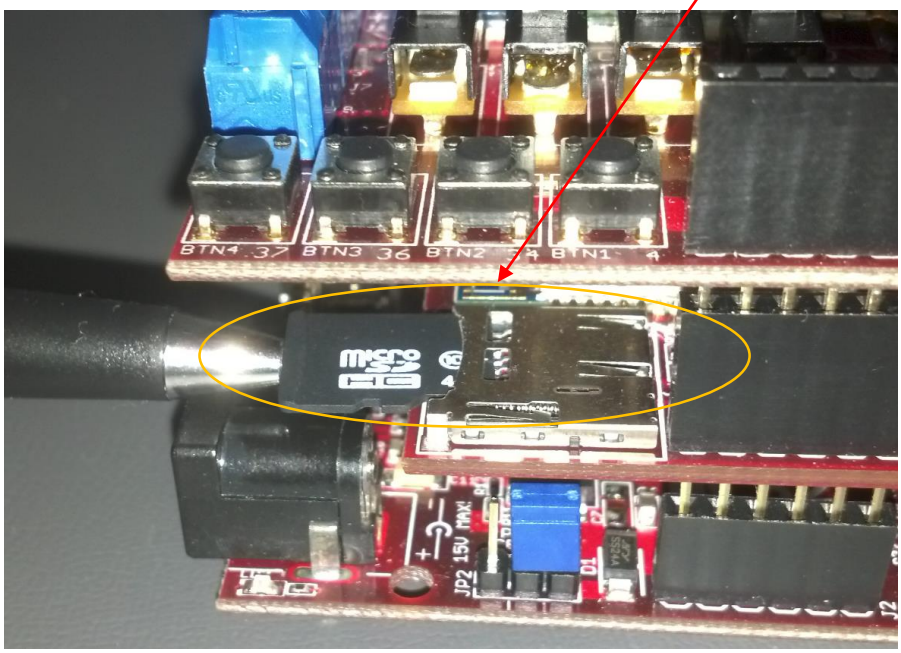
24. We continue to read until the full body of the HTTP message (as specified by our Content Length in the scratch buffer) is in the input buffer (rgbIn).

```
case DATA:

    // we know the body will be small, so just keep reading until we have the full
    // body of the HTTP message read into our input buffer.
    // remember we put the lenght in out output buffer using that as temp storage
    if(pClientInfo->cbRead < *((uint32_t *) pClientInfo->rgbOut))
    {
        pClientInfo->rgbIn[*((uint32_t *) pClientInfo->rgbOut)] = '\0';
        retCMD = GCMD::READ;
    }
    else
    {
        pClientInfo->htmlState = PARSE;
    }
    break;
```

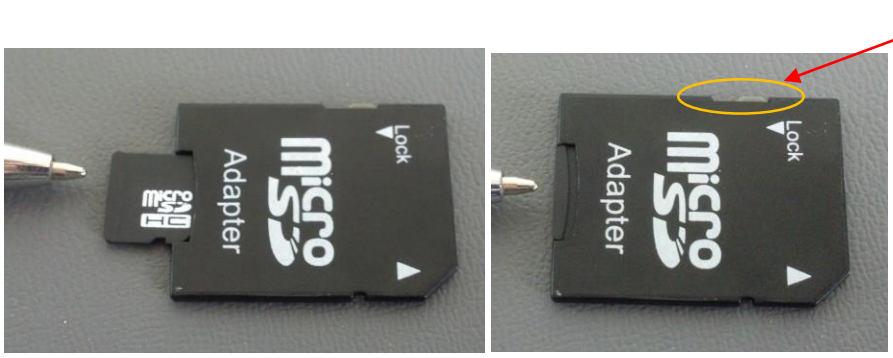
25. We look for the “LEDS=<nbr>” in the HTTP Body, adjust the value character to a zero based integer and index into the ordered list of chipKIT™ Pins and change the state of the LED.

```
case PARSE:
    {
        int ledState[HTTPCLED];
        pPattern = (char *) pClientInfo->rgbIn;

        // init all LEDS to LOW
        memset(ledState, LOW, sizeof(ledState));

        // read in the LED states
        while((pPattern = strstr(pPattern, szLEDS)) != NULL)
        {
            int i;
            pPattern += sizeof(szLEDS) - 1;
            i = (int) (*pPattern - '1');

            if(0 <= i  &&  i < HTTPCLED)
            {
                ledState[i] = HIGH;
            }
        }

        // set the LED state
        for(int i=0; i < HTTPCLED; i++)
        {
            digitalWrite(rgReplacePins[HTTPCIO + i], ledState[i]);
        }
    }
```

26. Now we must put our modified IOPage.htm on the µSD card. Remove the µSD card by pushing in on the card until it clicks out.



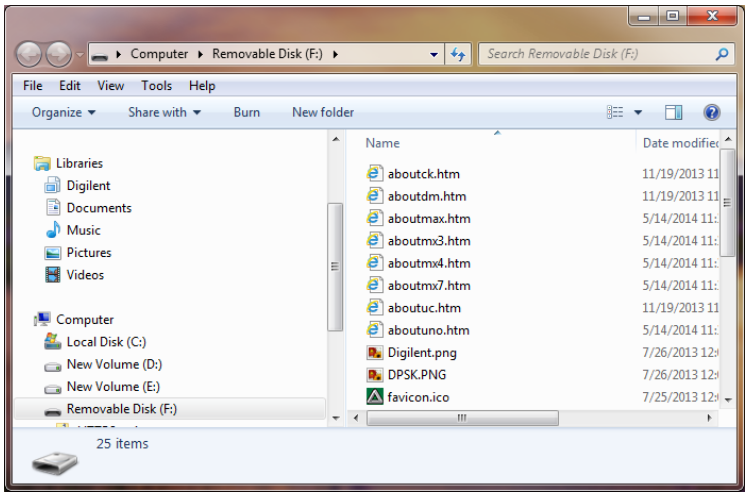
27. Insert the µSD card fully into the Micro SD Adapter. Also make sure your adapter is unlocked.



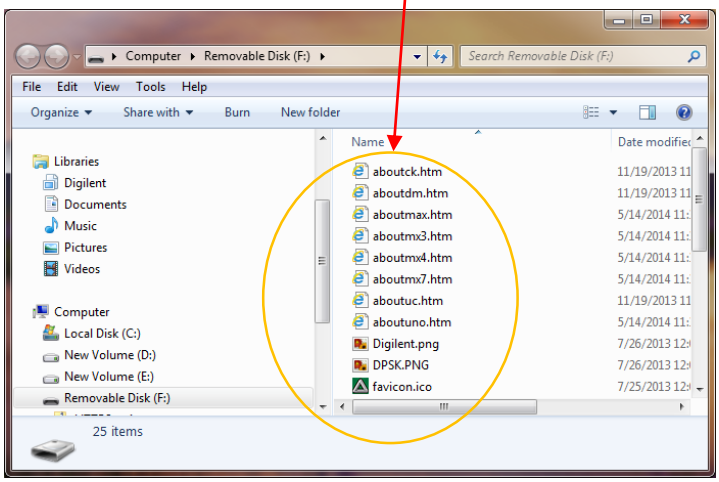
28. Insert the Micro SD Adapter fully into the SD slot on your computer.

29. Windows should notify you of the Removable Disk, Open the folder and view the files.

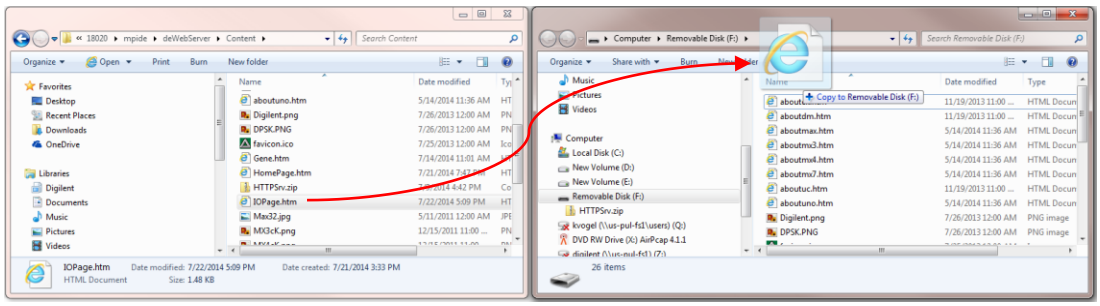
30. If windows does not AutoPlay to the above dialog, open the directory manually from Windows Explorer



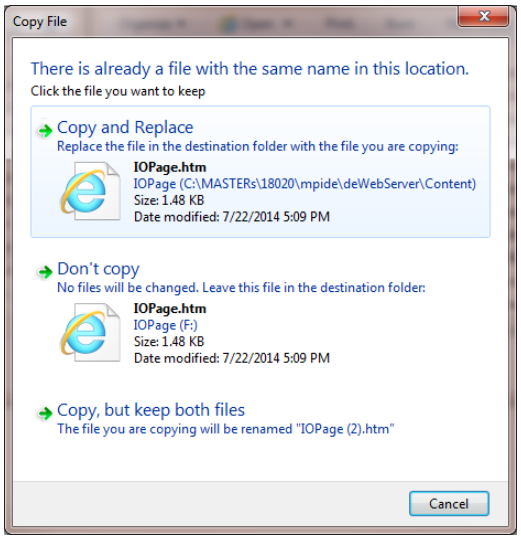
31. You should see the HTML files on the µSD card.



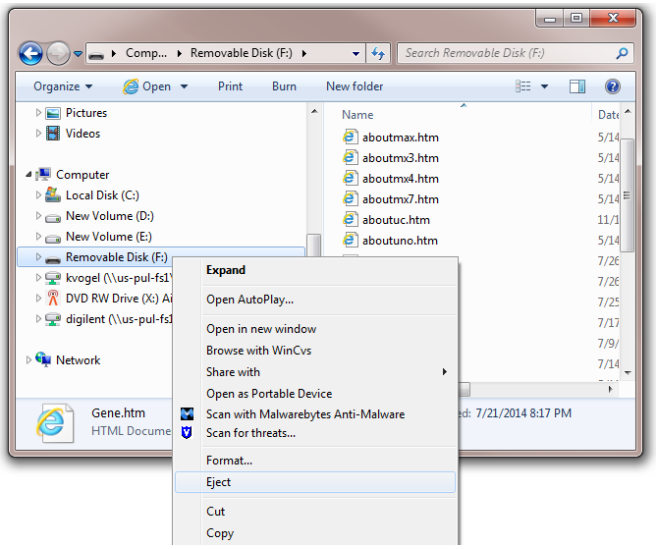
32. Now drag and drop IOPage.htm from your "Content" directory to the root of the µSD card.



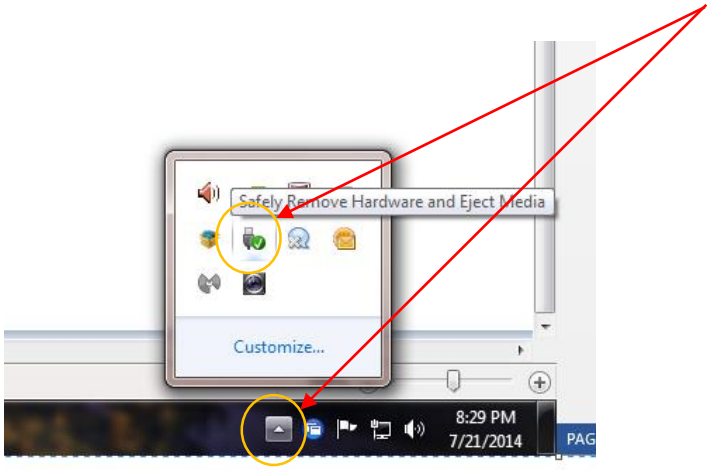
33. Remember that IOPage.htm already existed on the µSD card, so you will need to replace the file.



34. Remember to Eject the Removable µSD card to ensure all data was written to the card.

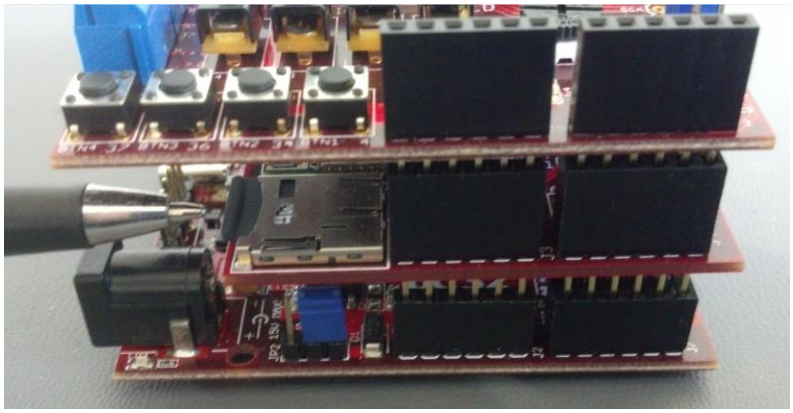


35. If you don't have an Eject menu item, then eject the hardware safely with the hardware manager.

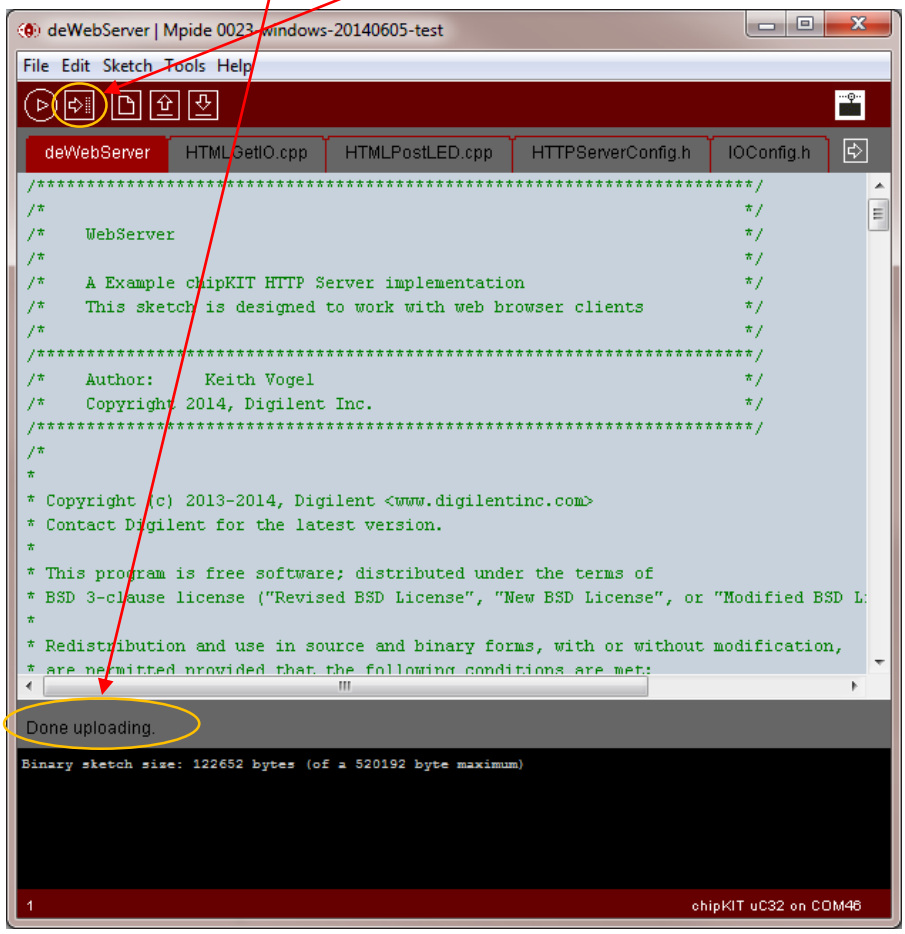


36. Remove the Micro SD Adapter from the computer and remove the µSD card from the Micro SD Adapter.

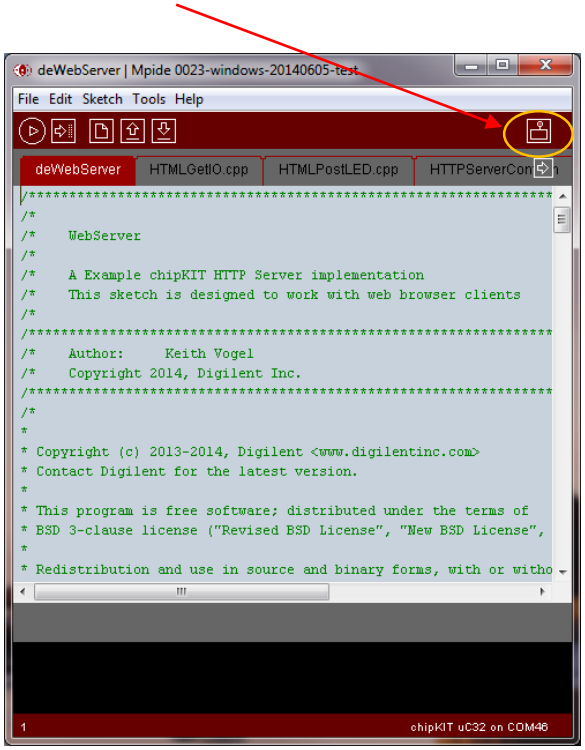
37. Fully reinserted the µSD card into the chipKIT™ uC32 µSD reader.



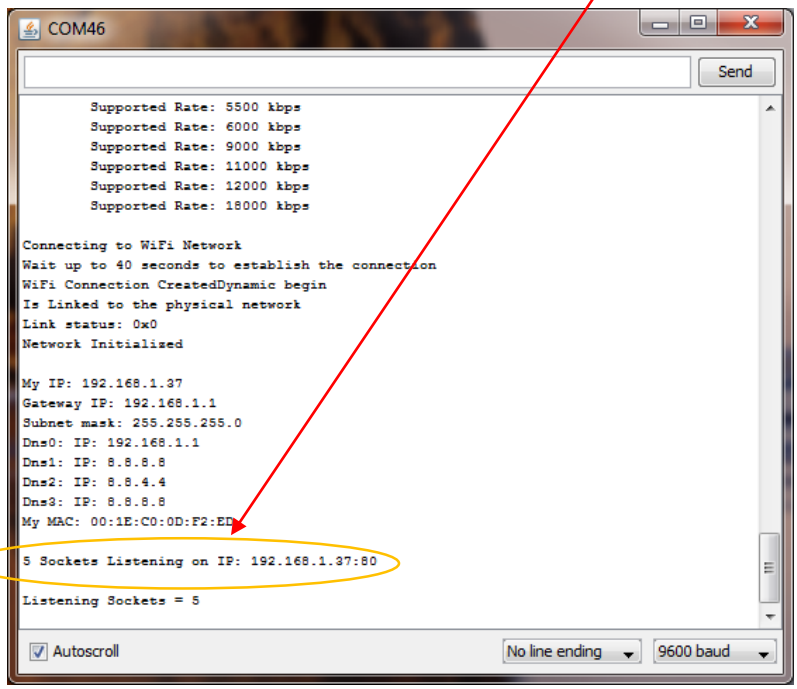
38. Since we modified code we will need to recompile and upload the sketch to the uC32. Remember to wait until it is done uploading.



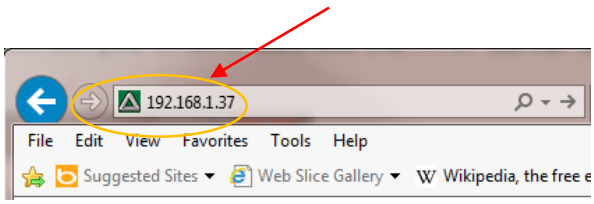
39. Restart the Serial Monitor.



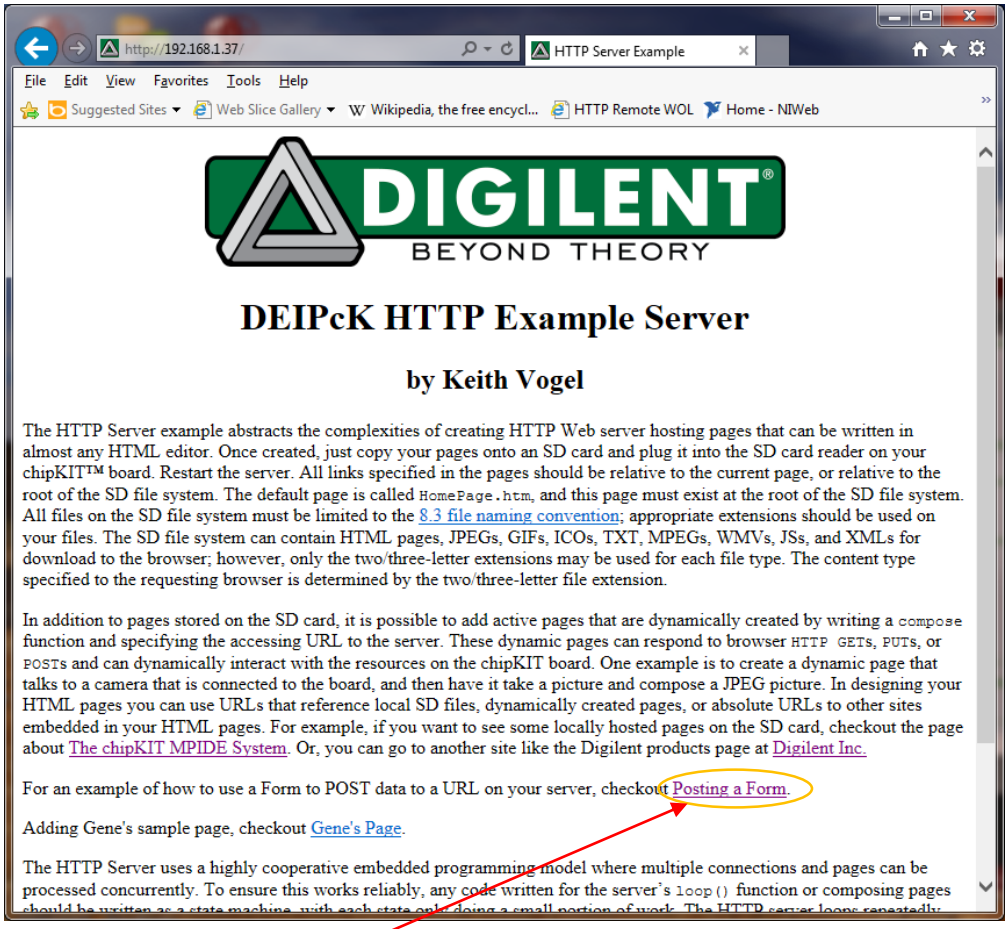
40. You should see the HTTP Server start as before, wait for the IP address so you know what to type into the browser address line; it should be the same address as before but does not have to be.



41. Start Internet Explorer and type in the IP Address in the Address line and enter.

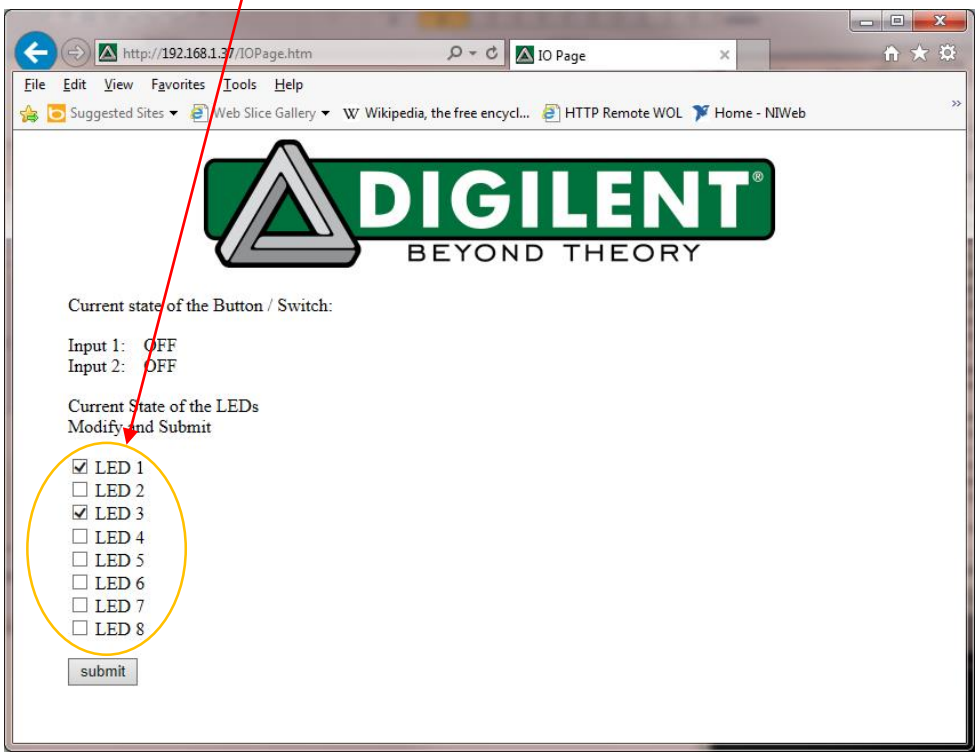


42. Internet Explorer should load the HomePage.htm as before.



Click on the "Posting a Form" link to get to your updated IOPage.htm.

43. Now you should see 8 LED entries instead of 3.



Change the state of the LEDs and hit submit. See how the uC32 LEDs respond.

44. You have modified a dynamic HTML page.

END LAB 3