PROJECTS ■ THEORY ■ APPLICATIONS ■ CIRCUITS ■ TECHNOLOGY

NUTS AND VOLTS

NUTS AND VOLTS

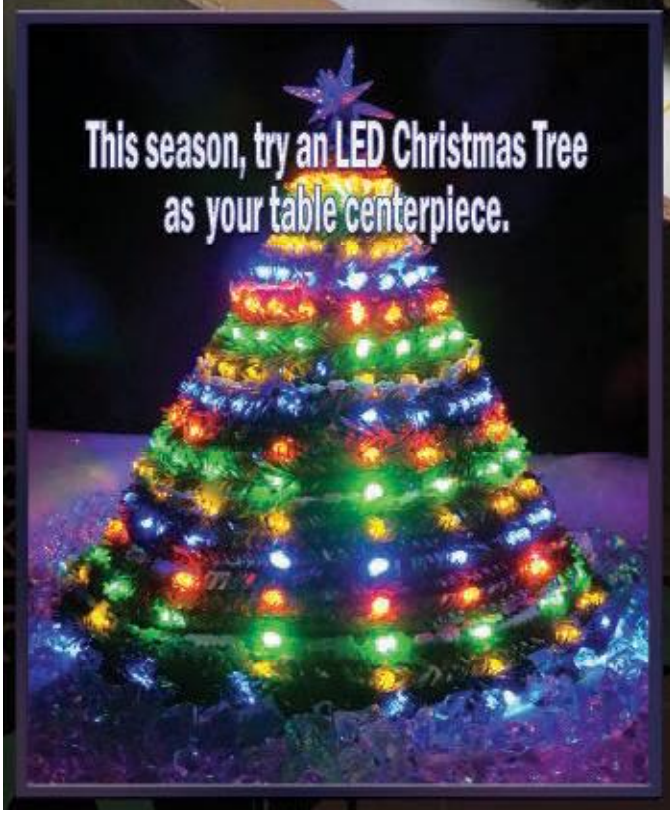
EVERYTHING FOR ELECTRONICS

www.nutsvolts.com
October 2011

Live Action
Flasher, Gate, And Bell
For Your Model Railroad

Legacy Communication With The 32-Bit Micro Experimenter

- Meet the new PICAXE M2-Class microcontrollers
- Web browser control of your home thermostat
- Whatever happened to CB radio?



U.S. $6.50 CANADA $7.50
10
0 71486 02421 7

Nuts & Volts
October 2011
www.nutsvolts.com

Columns

Projects & Features

You Asked and We Listened — New Forums are ONLINE!

This issue marks the first time that *Nuts & Volts* content will be directly linked to our online presence! We have completely rearchitected the forums so they now closely reflect the content of the magazine. Every regular column now has its own dedicated forum with the column author as your host! In addition, there are also forums to support project articles for reader projects and even an area where you can learn about what it takes to become a writer for the magazine! This new forum design invites you to interact with the author(s), read about corrections or modifications to projects, post questions, or just find and interface with lots of other folks who enjoy the same things you do. As an added bonus, we have populated the forums with example articles by each author so you can read them there if you don't have time to read them here. We hope you find the new forum layout exciting and useful. Please come by, have a look around, and introduce yourself! To explore the new forums, just point your browser to: **http://forum.servomagazine.com**.

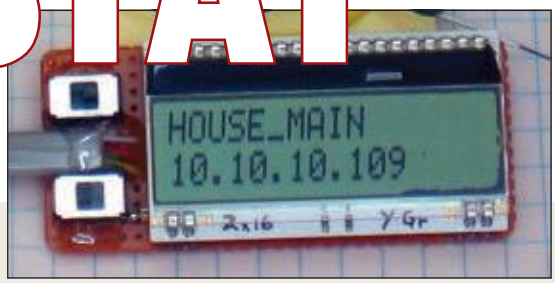
Hope to see you online soon! :)
Vern Graner, Forum Moderator, *Nuts & Volts*

Departments

Nuts & Volts (ISSN 1528-9885/CDN Pub Agree #40702530) is published monthly for $26.95 per year by T & L Publications, Inc., 430 Princeland Court, Corona, CA 92879.
PERIODICALS POSTAGE PAID AT CORONA, CA AND AT ADDITIONAL MAILING OFFICES.
POSTMASTER: Send address changes to **Nuts & Volts, P.O. Box 15277, North Hollywood, CA 91615** or Station A, P.O. Box 54, Windsor ON N9A 6J5; cpcreturns@nutsvolts.com.

WEB BROWSER CONTROL OF YOUR HOME THERMOSTAT

Part 2

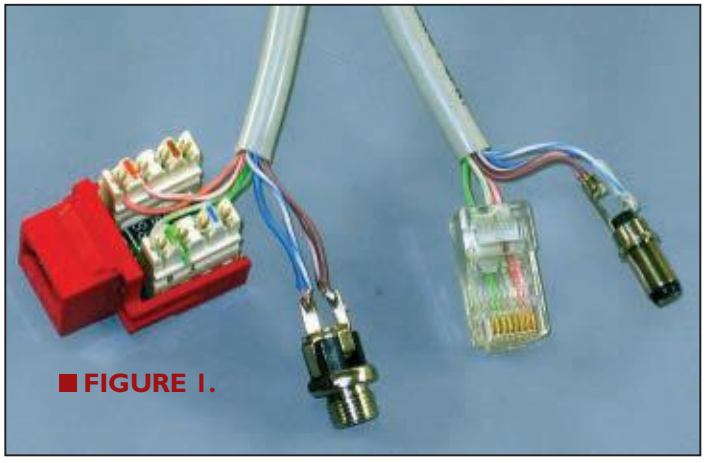


By Jürgen G. Schmidt

Last month, I described the basic hardware for implementing a thermostat with an embedded web server so it can be controlled from a browser. This consists of a TCP/IP Base board using a Microchip 16-bit PIC24 processor to provide a general-purpose web server, with a daughterboard that implements the thermostat hardware and interface to the home heating, ventilation, and air conditioning (HVAC) equipment. In this article, I will describe the physical connections, some construction details, and the software that makes it all work.

Network, Power, and HVAC Connections

While I had the foresight to install CAT-5 cable to the thermostats when I built my house, I did not run any power lines. I assumed at the time that I would get power from the thermostat itself. Alas, while some thermostats have a fifth wire to provide power, mine did not. Instead, power for the new thermostat is provided via the unused pairs of the CAT-5 networking cable. Only pins 1, 2, 3, and 6 are used for the network connection. If you look at the Wikipedia article on CAT-5 wiring (see **References**), you will see there are two ways to arrange the wires in the connectors. I used the T568B arrangement, since that matched my existing cables. The Wikipedia article on Power over Ethernet shows which wires to use for the power connection. The table at the end of that article shows that pins 4 and 5 are used for the positive terminal and pins 7 and 8 are used for the negative terminal.



■ FIGURE 1.

When I made my cables, I separated the wires and fed only the network-related pairs into the RJ45 connectors and soldered the other pairs to coaxial 2.1 mm power connectors. To verify that I had everything figured out correctly, I made a short test cable, the two ends of which are shown in **Figure 1.**

The red connector mounts on my patch panel, from which it is connected via a jumper to my network switch. A regulated five volt power supply plugs into the 2.1 mm jack. The male RJ-45 connector and the 2.1 mm plug connect to the TCP/IP base board. The live thermostat end of this can be seen in **Figure 2.**

The green connector coming out of the wall is attached to the HVAC control cable. This is a Phoenix pluggable terminal block that plugs onto a header on the thermostat board (**J1** on the circuit diagram). These connectors are handy because they can be oriented three different ways when plugged into the header and are convenient for testing and installation. If you mount screw terminals directly on the thermostat board, you have to juggle the thermostat, screwdriver, and cable while you attach and detach the wires. Instead, you attach the terminal block directly to the cable and then plug and unplug as needed.

Construction Issues

Eventually, the thermostat would be mounted in an enclosure and attached to the wall in place of the original thermostat. To simplify the overall assembly, I wanted to limit the connections to the circuit boards to power, network, and HVAC control. All the switches, the display, and connectors should be on the circuit boards. Finding the right switches was a challenge, along with mounting the LCD so that everything could be mounted at the right height behind and through the faceplate. The connectors between the circuit boards needed to be the right length to leave just enough clearance for the MagJack.

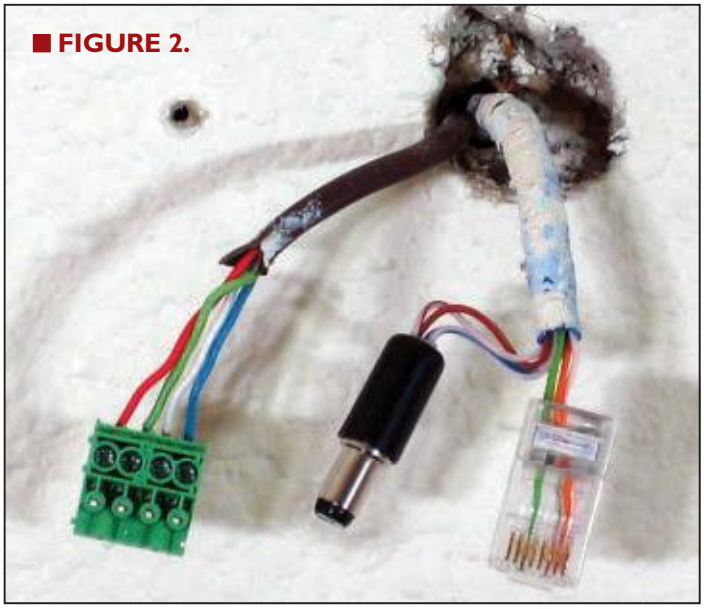
Plastic enclosures do not come in an infinite array of sizes and colors. The BUD CU-389 enclosure was just the right size but only available in black. After drilling the ventilation holes in the base, I spray-painted it off-white to match the walls. I custom-made my own faceplates from some scrap plastic. You can see these in **Figure 3**. The translucent gray cover is made from acrylic and the almond colored cover is made from an old ABS plastic enclosure. These were made on a ZenBot CNC router. The routed lettering is filled in with oil pastel.

Some of the components for this project are only available in surface-mount (SMD) format. That and size limitations dictated a SMD implementation for all but the connectors, LCD, relays, and switches. Some of the connectors are available in SMD format, but I prefer to use the through-hole version for mechanical sturdiness. This was my largest SMD project to date. I've used a fine-tipped soldering iron in the past, but for this project I decided to invest in a compressed air-powered solder paste dispenser and got a convection toaster oven to use as a reflow oven.

Initially, I used a thermocouple attached to my multimeter to monitor the oven temperature, but now I do so much surface-mount work that I built an automated reflow oven controller, based on the thermostat daughterboard. It provides the display, pushbuttons, and relay driver. (See the **sidebar**: Flexible Designs.) My website — **www.jgscraft.com** — has additional information about my surface-mount soldering process and how these boards were made.

Everything worked well, including soldering some of the bypass capacitors on the underside of the TCP/IP board. I moved these to the back to reduce clutter on the top of the PCB. As the flux heats up, there is enough surface tension to hold the parts in place, so only a single baking cycle is needed. I used 805-sized components for the resistors, LEDs, and most of the capacitors. The bypass 0.1 µf capacitors are 603s.

After baking the boards, the only cleanup I needed to do was around the PIC 44-pin TQFP part. There were some solder bridges that were easy to fix by running a fine-tipped soldering iron between the leads. The corollary to solder bridges is unsoldered pins which again can be fixed with a fine-tipped soldering iron and .025 inch solder. I test all the connections on the multi-pin parts for shorts and opens just to avoid headaches later on.

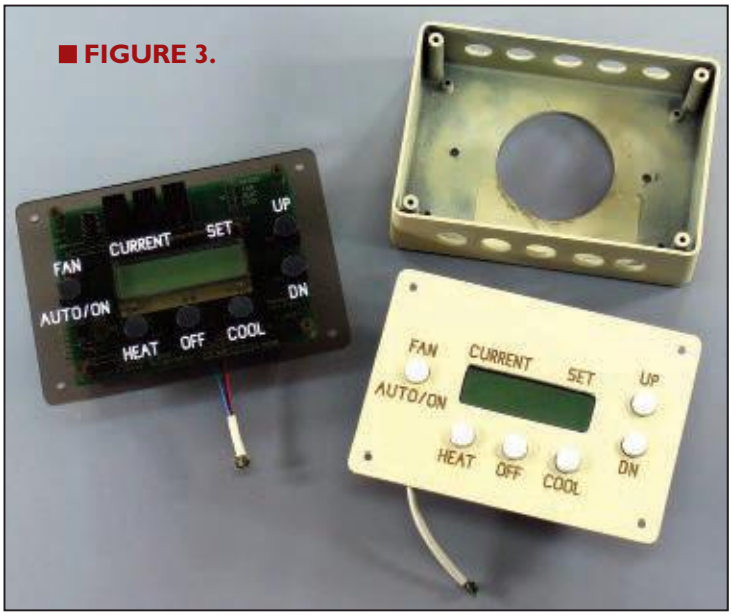


■ FIGURE 2.

For the fine-pitched parts, there is a fine line between applying enough solder paste to avoid dry connections and too much that results in bridged connections. I also found that some parts had slightly bent pins that did not make contact with their pads, so now I check the parts on the flat side of an old CPU heatsink to be sure all the pins touch the surface.

Software

Developing the software for the thermostat requires Microchip's C30 compiler which is now called "MPLAB C Compiler for PIC24 MCUs" and the TCP/IP stack version 5.25 (or later) which is part of the Microchip Application Libraries. You also need the MPLAB IDE to manage the project and its many files. The student or demo version of



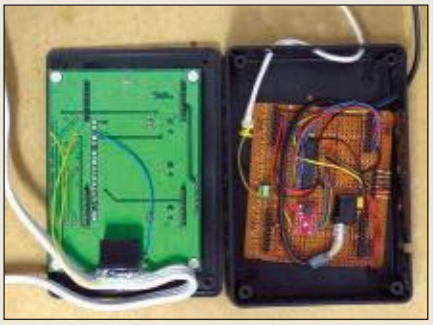
■ FIGURE 3.

Flexible Designs

The initial design work for the browser-controlled thermostat started out as a single, monolithic circuit and board design. Space limitations suggested a double-decker design which led to a separation of functions. This ultimately led to the design of two boards which could be applied to much more than just a thermostat application.

I isolated the hardware for the network connection and embedded webserver on one board and added some additional support circuitry to make it more flexible. The original design using a 28-pin processor had just enough I/O pins to support the thermostat application. Switching to a 44-pin part allows flexibility with other applications. I was able to develop the TCP/IP stack, bootloader, and webserver software without needing the thermostat board. All the unused I/O lines, power, and UART2 connections are available on PCB pads that will support standard 0.1" pitch headers.

The thermostat interface board has an LCD, some pushbuttons, relay driver circuitry, and some relays. Many of the projects I have seen in *Nuts & Volts* have similar interface requirements. The board does have dedicated circuitry for attaching the temperature and humidity sensors. However, these — as well as all the other connections — are just links via headers to I/O lines on the processor — they could be connected to anything.



There are lots of projects in my backlog that need a browser interface to hardware systems. The TCP/IP base board will work well for those. Other projects require a display and pushbutton interface. I have already repurposed one of the thermostat daughterboards as a reflow oven controller. Instead of the two-line LCD, I used a three-line version from the same manufacturer. I only needed one heavy-duty relay and three of the pushbuttons. The headers attach to a perfboard that mounts the PIC16F690 controller I used for this, a thermocouple interface, and power regulator. All of this I mounted in the same enclosure I used for the thermostat. I just deleted some of the holes from the existing CNC router program to give me the right faceplate. The results can be seen in the photos.

The added benefit of flexible designs is that I make use of the extra PCBs I end up with. Some boards I order have design errors, but if the design is flexible, they can still be used for other projects. Even for the final boards there is usually a minimum area per order, resulting in extra boards. Ideally I would sell them, but if I can't, I will eventually find another use for them.

the C30 compiler works fine, so all your software development tools are free. The TCP/IP stack is fairly easy to use, provided you know the basics of the TCP/IP protocols and you read the documentation for the stack. Studying the source code and posts on the Microchip forums related to Ethernet and the processor family you are using is also a good idea. The forums are well supported and provide a wealth of information that might be hard to find elsewhere.

The first step in using the stack is to take one of Microchip's sample applications as a starting point and modify it for your hardware. This is usually the "TCP/IP Demo App" and is referred to regularly in forum postings. This implements a webserver, as well as most of the common TCP/IP components such as UDP, TCP, Client/Server, email, DHCP, and more. The Demo App is configured to run on Microchip's evaluation and demonstration boards but modifications to the hardware configuration files will adapt it to run with your own TCP/IP hardware or — in this case — the thermostat project. The TCP/IP base board comes with detailed instructions for configuring Microchip's TCP/IP stack version 5.31 to run the Demo App.

This is not my first embedded TCP/IP project and I don't recommend anyone start a project like this from scratch. Get a development kit from Microchip or other hardware that comes with working software and start experimenting from there. I started out working with TCP/IP on a small development board, a sample application, and a series of tutorials that were known to work. Over time, my software and hardware evolved from this in gradual steps. If something failed, I could go back a step to find out if the software or the hardware was at fault. One incorrectly assigned pin in the software or one loose solder joint is all it takes to hang up the system, and it's hard to tell if hardware or software is the culprit. So, before adding my own code, I make sure any new hardware will work with the demo application.

The files that typically need to be modified are HardwareProfile.h and TCPIPConfig.h. For the TCP/IP base board, I also needed to modify ENC28J60.c to enable the clockout signal that drives the CPU clock. The main.c file contains some hardware initialization routines that need to be modified to match the hardware, as well. Once the board tests okay for the basic TCP/IP applications, I can add support modules for any additional hardware. Then, I add modules to main.c for my specific application.

The HVAC control software consists of a loop that: 1) Pauses a while and checks the system clock to see if certain timeout events have occurred; 2) Reads the sensor(s) and updates variables with the results; 3) Checks if a button has been pressed and updates variables corresponding to the thermostat settings; 4) Checks to see if any of the relays need to be turned on or off; and 5) Updates the display. There is no rocket science here, but there is a control detail that needs attention.

If a heater is set to turn on when the temperature gets below 60 degrees F, it will do so when the sensor is at that temperature. However, once the sensor is at 60 or above again, the heater will turn off and when the temperature falls again, the heater will turn on. This can result in the heater trying to turn on and off rapidly as the temperature fluctuates. To avoid this, we include a dead-band or hysteresis into the control loop.

Once the heater turns on, it will not turn off again until the heat is a certain amount, maybe one degree

```
switch( myAppState )
{
  case smMY_IDLE:              //------ don't work too hard ---------------
    //-- check if we need to turn off backlight
    if( TickGet() - ticks3 >= TICK_SECOND * 5uL ) LEDB_OFF();

    //-- check if we need to turn off alternate LCD display
    if( TickGet() - ticks4 >= TICK_SECOND * 3uL ) showLCD = LCD_MAIN;

    if( TickGet() - ticks1 >= TICK_SECOND/20ul )
    {
      ticks1 = TickGet();
      myAppState = smMY_STEP1;
    }
    break;

  case smMY_STEP1:          //------ get temp and humidity -----------------
    //-- get temp and humdity periodically
    if( TickGet() - ticks2 >= TICK_MINUTE/12ul )
    {
      ticks2 = TickGet();
      sht_rd(&tempC, &humidity);        // SHT11 library
    }
    humidity = (int)(humidity);         // round to integer value
    tempF = (int)(tempC * 9 / 5 + 32);
    myAppState = smMY_STEP2;
    break;

  case smMY_STEP2:          //------ check for button press ----------------
    .
    .
    .
    myAppState = smMY_STEP3;
    break;

  case smMY_STEP3:          //------ check if we need to turn anything on or off
    .
    .
    .
    myAppState = smMY_STEP9;
    break;


  case smMY_STEP9:          //------ check for any configuration changes and save them
    .
    .
    .
    myAppState = smMY_IDLE;
    break;


} //switch( myAppState )
```

LISTING 2.

```
  while( 1 )
  {
    switch( smSysState )
      {
        //-------------------------------------------
        //------ begin of main TCP/IP processing loop ------

        case RUN_IDLE:
          //-- toggle heartbeat LED
          if( TickGet() - t1 >= TICK_SECOND/2ul )
          {
            t1 = TickGet();
            LED1_IO ^= 1;
          }
          smSysState = RUN_NET_TASKS;
          break;

        case RUN_NET_TASKS:
          StackTask();               // HTTPServer runs here as another state machine
          smSysState = RUN_NET_APPS;
          break;

        case RUN_NET_APPS:
          StackApplications();
          smSysState = RUN_MY_APPS;
          break;

        case RUN_MY_APPS:
          MyTasks();
          smSysState = RUN_UPDATES;
          break;

        case RUN_UPDATES:
          if( AppConfig.MyIPAddr.Val != eth0_ip_addr )
          {
            eth0_ip_addr = AppConfig.MyIPAddr.Val;
            IPAddressToString( &AppConfig.MyIPAddr, buffer);
            printf("New IP Address = %s\n\r", buffer);
#if defined(STACK_USE_ANNOUNCE)
            AnnounceIP();
#endif
          }
          smSysState = RUN_IDLE;
          break;

      //------ end of main processing loop ------
      //-------------------------------------------

      case SYS_ERROR:
        printf("ERROR STATE\n\r");
        while(1);

      case SYS_RESET:
        printf("%s System Reset\n\r", PRODUCT_NAME);
        Reset();

    } //switch( smSysState )
  } //while( 1 )
```

LISTING 1.

higher than the set point; in this case, 61 degrees. This temperature difference is the dead-band in which no changes will take place. In the thermostat software, there is a dead-band of two degrees Fahrenheit. The heater will turn on at one degree below the set point and will turn off at one degree above the set point. You can see the details of this in the thermostat.c file that is included with this article's downloads.

Multitasking

For a webserver to work properly, the TCP/IP software needs to monitor the network connection for incoming traffic. While the incoming data is processed, it still needs to continue checking for incoming data so none is lost. In order to accomplish this, the TCP/IP stack is implemented as a series of nested loops. Each of these loops has multiple steps in it. Each step performs a little bit of work, exits the loop to let the outer loop do a little work, and then the inner loop resumes with the next little step, exits, and so on. This is called "cooperative multitasking" and is described in detail in the TCP/IP stack documentation. This is important in the context of the thermostat application because it must also follow this pattern. **Listing 1** shows the case statement that runs the TCP/IP stack in the demo application. This is running in an infinite while() loop. The blue code highlights where the user application, MyTasks(), has been added. This gets called every fifth time through the loop.

The MyTasks() function in the file thermostat.c implements the thermostat as described above and is shown in abbreviated form in **Listing 2**. Whenever it is called, the state variable, myAppState, is checked and the corresponding portion of the case statement is processed. Then, the state variable is updated with the next value and the function exits to allow the main loop to do some work. Five iterations of the main loop later, we're back in MyTasks() to process the next section of code, and so on. The application programmer is responsible for making sure nothing is interrupted for long — or worse — blocked while waiting for something to happen such as a button press or serial communication.

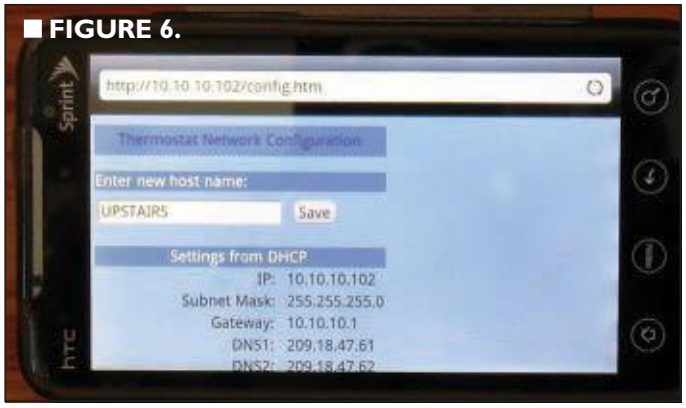
The Embedded WebServer

The usual software process for microcontroller development involves compiling a program, loading the HEX file, and testing the result. With an embedded webserver, there are some additional steps. You need to develop the web pages you want displayed and then you need to load them onto the server.

Web pages for the TCP/IP stack are created just like any other website. You can include pictures, JavaScript, CSS, and any other files that your target browser will support. The webserver stores the files in the serial Flash memory, up to four megabytes worth. The primary job of the webserver is to just deliver the files to the browser. Dynamic variables in the web files will be filled in by the

■ FIGURE 4.

■ FIGURE 5.



■ FIGURE 6.

webserver before the files are sent to the browser. This is fine for relatively static content, but for an interactive application such as a thermostat, we need something more. We want to be able to see the temperature change without having to hit the refresh button on the browser. This is accomplished via an AJAX interface.

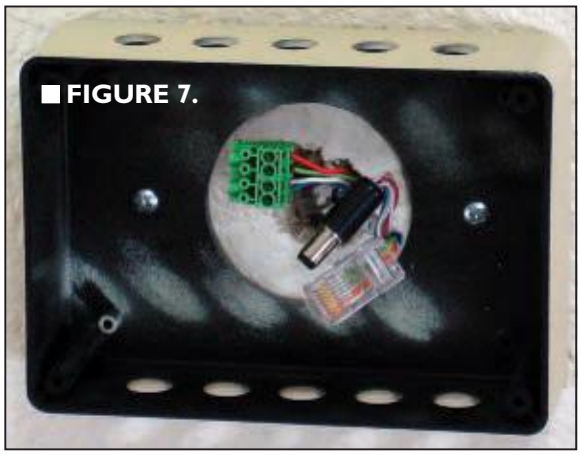
AJAX is basically a library of JavaScript functions that leverage certain browser features for interactive applications. The AJAX routines communicate with the corresponding functions in CustomHTTPApp.c which is part of the TCP/IP stack. A detailed explanation of how this works would take at least another article on its own. For now, you'll have to settle for reading about it in the documentation and sample code from Microchip. The JavaScript code runs in the user's browser, periodically sending updates from the web page to the webserver and retrieving data from the webserver for updating the browser display.

For example, the thermostat web page has a button for lowering the set point. Clicking this button activates a piece of JavaScript code that sends a message to the server that the decrease button has been clicked. On the server, the set point value is decremented and transmitted back to the web browser so the new set point can be displayed.

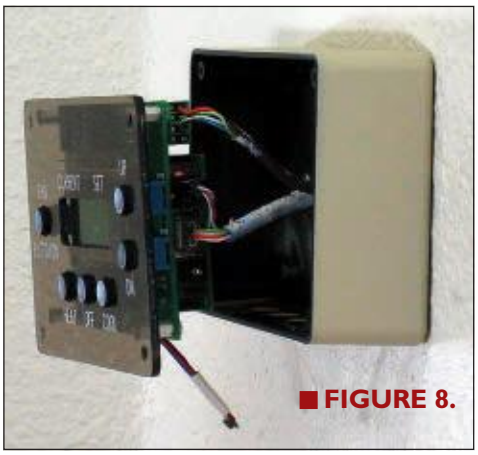
Another piece of code updates the LCD display. Even if there aren't any user-initiated changes on the web page, the JavaScript code periodically queries the webserver for updates so that the web page is kept current. That will happen if the decrease pushbutton on the physical thermostat is pressed. The set point value is decremented and displayed on the LCD. The JavaScript code will retrieve the updated value for display on the web page.

The web page design can be changed and uploaded to the webserver without making any changes to the application code. The only time you would need to change the application and recompile it is if you want to exchange more data between the web page and the application. If you don't like the way the thermostat web page is laid out, change it and upload it using the MPFS2.EXE utility. You can even add additional pages, as long as they fit into the memory.

Figure 4 shows the default thermostat page. **Figure 5** shows an alternative page that is accessed by changing the web address. **Figure 6** shows a configuration page that changes the host name for the thermostat. Most computer-based browsers allow you to access the web page by the host name. Unfortunately, the browser on my EVO doesn't do that and I have to type in the IP address and save it as a bookmark.



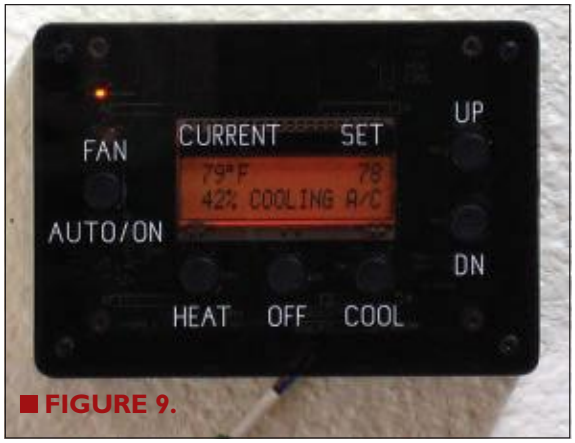
■ FIGURE 7.



■ FIGURE 8.

Installation

The hardest part of



■ FIGURE 9.

the installation is probably arranging for a network cable to come out where you plan to mount the physical thermostat. In most cases, that should be a matter of dropping the cable down inside the wall from the attic or feeding it up from the basement or crawl space. Attach the connectors as described in Wikipedia articles and here.

You do need a DHCP server on your network to provide the thermostat with an IP address. Most cable or DSL routers provide this by default. Disconnect the old thermostat from the wall and identify the function of the wires. The diagram at **http://wiki.xtronics.com/index.php/Image:Thermostat.gif** should help you in identifying them if it's not obvious. Connect the thermostat wires to the Phoenix terminal block in the sequence shown on the thermostat PCB. For now, connect only the network cable and power.

The heartbeat LED should blink once per second and the LCD should display the current status.

Test the various buttons to be sure they change something. If you did not connect a terminal to see the debug output, you can press the "UP" and "DN" buttons at the same time to display the IP address and host name. If all this is working, connect the thermostat connector, making sure it is oriented correctly.

Sequencing the final assembly of the pieces on the wall was a puzzle in itself. The enclosure needs to be attached to the wall. Wires need to be plugged in. The PCB assembly needs to be mounted in the enclosure and then the faceplate goes on. After playing with some mockups, I decided to attach the thermostat daughterboard to the faceplate with some standoffs. The TCP/IP base is socketed to the daughterboard, held firmly in place by the four eight-pin headers.

After plugging in all the connectors, you can attach the faceplate to the enclosure that has already been attached to the wall. It's a tight fit so you need to be sure you can push any excess wire into the wall. **Figures 7**, **8**, and **9** show this sequence. You are now ready to control your thermostat from your recliner. **NV**

REFERENCES

CAT-5 Wiring
http://en.wikipedia.org/wiki/Cat-5

Power Over Ethernet
http://en.wikipedia.org/wiki/Power_over_Ethernet

ZenBot CNC Router
www.zenbotcnc.com

Hysteresis Explained
http://wiki.xtronics.com/index.php/Dead-band_hysteresis

AJAX
http://en.wikipedia.org/wiki/Ajax_%28programming%29

Thermostat Circuit
http://wiki.xtronics.com/index.php/Image:Thermostat.gif

Brush Electronics
http://brushelectronics.com/

Microchip Application Libraries
www.microchip.com/mal

Microchip Bootloader Application Note
http://ww1.microchip.com/downloads/en/AppNotes/01094a.pdf

You can contact me at jurgen@jgscraft.com.