PicoWeb “WebLED” Project

This PicoWeb project can be found in the Picoweb development system “samples” directory in a file named `samples\webled\webled.pwp`. This project consists of a single HTML document or Web page (`webled.htm`) which references three JPEG images (`bruce.jpg`, `dave.jpg` and `steve.jpg`).This project allows a PicoWeb server to display the “home page” shown in Figure 1.

A single user-supplied application-specific CGI routine, written in PicoWeb pcode, is also required to support this simple PicoWeb project. This CGI routine is used to read the current temperature from an I^2C digital thermometer chip attached to the PicoWeb server’s 25-pin I/O expansion connector. (You can still build and load this project’s firmware into a PicoWeb server without an attached digital thermometer. If the I^2C chip is missing, the PicoWeb’s firmware will always read -1 degrees Celsius from the missing chip.)

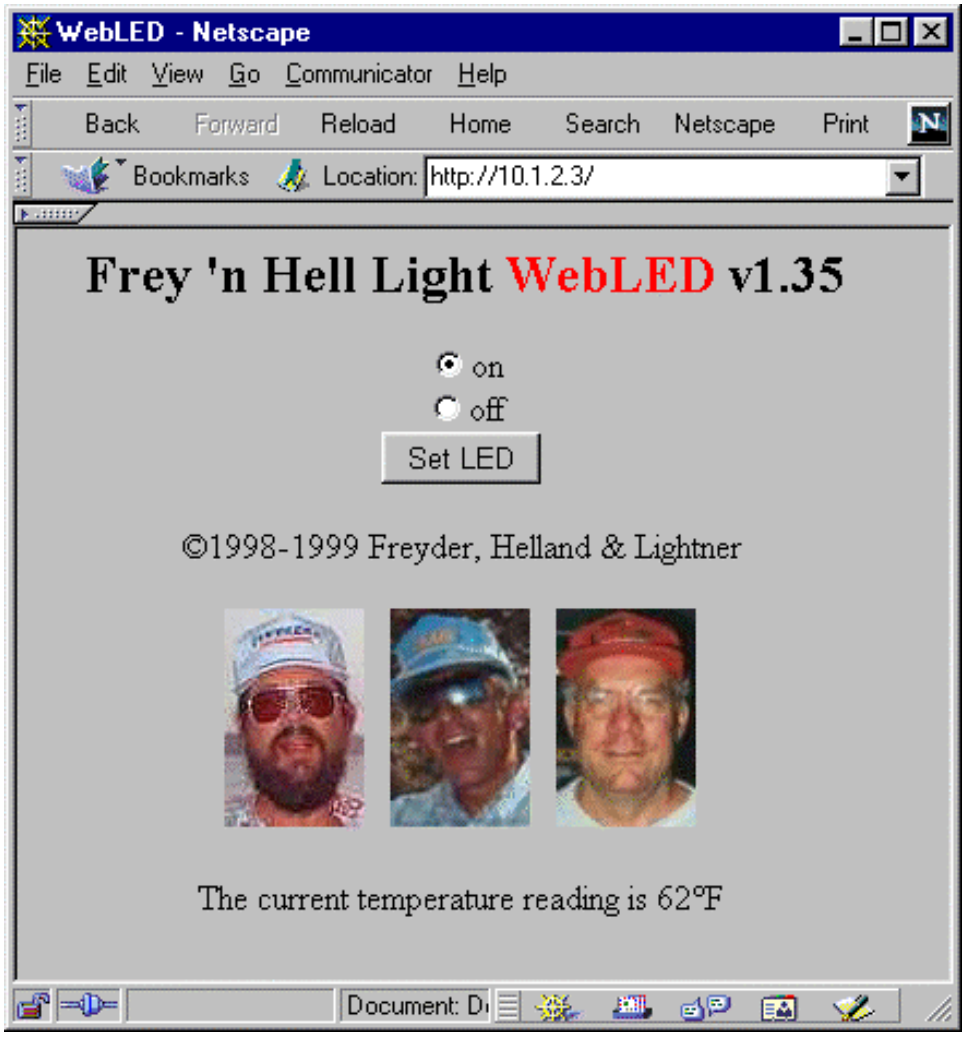

Figure 1- PicoWeb WebLED Project Web Page

Project File - The various parts of the WebLED project file (`webled.pwp`) will be described individually in the following sections:

```
//-------------------------------------------------------------------------
//
// PicoWeb Project File for WebLED
//
//-------------------------------------------------------------------------
//
// application-specific preprocessor definitions
//
#define BANNER "\r\nPicoWeb WebLED\r\n"
#define EEPROM_IP         /* use file "ip" for default IP address */
#define NET_CONFIG_IP     /* allow IP address reconfiguration via net */
#define ENABLE_WATCHDOG   /* use Atmel watchdog timer hardware */
#define DEBUGGER          /* include debugger firmware */
#define CLOCK 7372000     /* clock rate of microcontroller (in MHz) */
#define BAUD_RATE 19200   /* serial port baud rate */

link add i2cprog_I2C_PORT1_PRESENT.o  // use special version of I2C routine
```

The project file `webled.pwp` also contains several preprocessor #define statements, as follows:

- `BANNER` – specifies a text string that is printed each time the PicoWeb server is reset
- `EEPROM_IP` – specifies that the PicoWeb IP address is stored in on-chip EEPROM (the address will come from the file `ip`)
- `NET_CONFIG_IP` - allow IP address reconfiguration via net using the program `setip`
- `ENABLE_WATCHDOG` – enables the watchdog timer hardware on reset
- `DEBUGGER` – includes the serial port debugger firmware
- `SERIAL_BAUD_DIVISOR` – sets the baud rate for the serial port (used by the debugger)
- `CLOCK` – defines the clock rate of the Atmel microcontroller (in MHz)
- `BAUD_RATE` – sets the baud rate of the serial port UART

A special PicoWeb *linker directive* is also specified (i.e., `link add i2cprog_I2C_PORT1_PRESENT.o`). This will cause the PicoWeb development system to load a version of the PicoWeb's I^2C port driver (`lib/i2cprog.asm`) which is assembled with the macro `I2C_PORT1_PRESENT` defined. This object file version adds support for a Dallas DS1621 digital thermometer chip and exposes a number of added subroutines for using this chip. (Refer to the file `lib/Makefile` to see how this object file is created.)

```
//
// application-specific HTML and image file names
//
webled.htm      // default Web page (i.e., "home page")
bruce.jpg
dave.jpg
steve.jpg
```

This section of the project file lists the files which contain HTML code and images which will be stored in the PicoWeb serial EEPROM memory as part of the Web server's "file system". The first file in the list must be the default Web server "home page". Everything after the //'s is comment text, and is ignored by the software which processes the project file.

```
//
// application-specific public CGI routines
//
temperature.cgi // (returns ASCII temperature reading (deg-f)
```


The above line indicates that a user-supplied CGI pcode routine named "`temperature`" is a "public" CGI routine that can be referenced as the PicoWeb URL "`/temperature.cgi`". Note also that wherever the special PicoWeb *tag* `` `temperature.cgi` `` appears in any of the HTML source code files loaded into the PicoWeb server file system as part of this project something special happens. This *tag* will cause the pcode routine "`temperature`" to be called at that point in the HTML code stream whenever a HTML file with that tag is retrieved.

```
#avr_reset
;-------------------------------------------------------------------------
; this code is executed each time microcontroller is reset
;-------------------------------------------------------------------------
;
    sbi     ddrd,LED_BIT      ; make LED driver pin an output

    ldi     yl,0xEE           ; start DS1621 temperature conversion
    rcall   ds1621_write
```

The above AVR assembly language will be inserted into the PicoWeb firmware build such that that code is executed each time the Atmel microcontroller is reset. The first line of code makes the microcontroller pin which drives the single user-controlled LED on the PicoWeb server an output pin (i.e., the I/O pin PD4). The next two assembly language instructions shown above initialize the I^2C digital thermometer chip.

```
#avr_slow
;-------------------------------------------------------------------------
; this code is executed each trip through "slow idle" loop (~1 sec period)
;-------------------------------------------------------------------------
;

#avr_fast
;-------------------------------------------------------------------------
; this code is executed each trip through "fast idle" loop
;-------------------------------------------------------------------------
;
```

The #avr_slow and #avr_fast code sections are empty. In fact, those directives could be removed from the project file. These sections allow the user to insert code in the inner (fast) and outer (slower) idle loops in the PicoWeb server "kernel". The "fast idle loop" is where the PicoWeb server firmware polls the Ethernet controller chip. The outer "slow idle loop" is executed about once per second, and is used to check on less demanding tasks.

```
#avr_asm
;-------------------------------------------------------------------------
; application-specific CGI pcode and AVR assembly routines go here
;-------------------------------------------------------------------------
;

;-------------------------------------------------------------------------
;+
; **-temperature-show the temperature (deg-F) in decimal.
;-
;-------------------------------------------------------------------------
.eseg
temperature:
    pcall tempf_buf                 ; read temp in deg-F
    pprintswi [buf]                 ; output as decimal ASCII
    pret
```


The #avr_asm project file directive specifies the beginning of user-supplied, application-specific routines which do not need to be executed at reset time, nor every time through one of the "idle loops". This is where user-supplied CGI routines which are activated by Web pages are normally placed.

The above pcode section is a PicoWeb pcode routine named temperature which, when called, will read the current temperature from the digital thermometer chip and output that reading as ASCII text to the *standard output stream*. If this routine is called because an HTML page is being retrieved which contains a special tag pointing to this routine, the *standard output stream* will be directed to the HTML code stream. Therefore the latest ASCII temperature reading automatically will be inserted into the HTML code stream in place of this routine's tag. Note the .eseg directive which causes this code to reside in the PicoWeb server's external serial EEPROM memory.

```
;---------------------------------------------------------------------------
;+
; **-tempf_buf-read temp (deg-C) from DS1621 and convert to deg-F
;
; inputs:
;   none
;
; outputs:
;   buf = latest temperature reading in degrees Fahrenheit
;-
;---------------------------------------------------------------------------
.cseg
tempf_buf:
    pclrw buf
    pread_temp buf                  ; get temp
    pmovwi buf+4,0x80               ; set up for sign test
    pandwi buf+4,[buf]              ; check the sign
    pjumpeq temp_positive           ; it is positive
    pmovbi buf+1,0xff               ; extend the sign
    pnegw buf                       ; negate
temp_positive:
    pmul buf,[buf],9                ; C*9
    pdiv buf,[buf],5                ; /5
    pandwi buf+4,0x80               ; test original sign
    pjumpeq temp_norenegate         ; positive
    pnegw buf
temp_norenegate:
    paddwi buf,32                   ; +32 - buf now has temp (F)
    pret
```

This PicoWeb pcode routine reads the current temperature from the digital thermometer chip then converts that reading from degrees Celsius to degrees Fahrenheit. Temperature is read from the chip as a single signed 8-bit value in degrees Celsius. After multiplying that reading by 9, dividing by 5, and adding 32, the resultant 16-bit value is passed back to the caller in the SRAM location buf. Note the .cseg directive which causes this routine (and those which follows it) to reside in the PicoWeb server's internal program memory.

```
    pprintswi [buf]                 ; output as decimal ASCII
```

The above PicoWeb pcode instruction, part of the CGI routine "temperature" takes the signed 16-bit value stored at SRAM location buf (i.e., the current temperature), converts that number into ASCII text, and outputs the text to the *standard output stream*.


```
<html>
`pchk_port_url_parms.cgi`
<head><title>WebLED</title></head>
<body text=#000000 bgcolor=#c0c0c0>
<center>
<h2>Frey 'n Hell Light <font color="red">WebLED</font> v$$VERSION$$</h2>
<FORM name=mfrm method=GET action="/">
<input type=radio NAME=4 VALUE=0 `?testport.cgi?0x04`CHECKED{}>on<br>
<input type=radio NAME=4 VALUE=1 `?testport.cgi?0x04`{CHECKED}>off<br>
<input type=submit VALUE="Set LED">
</FORM>
&#169;1998-1999 Freyder, Helland & Lightner
<br><br>
<img src="steve.jpg" width=64 height=100 alt="Frey">  
<img src="dave.jpg" width=64 height=100 alt="Hell">  
<img src="bruce.jpg" width=64 height=100 alt="Light">
<br><br>
The current temperature reading is `temperature.cgi`&#176;F
</center>
```

Listing 1 – PicoWeb "Home Page" (`webled.htm`)

PicoWeb HTML Coding - Listing 1 shows the contents of the project's "Home Page" (`webled.htm`). This page will display the Web page shown in Figure 1. To those familiar with HTML, this file looks very much like an ordinary HTML document except for some odd syntax relating to tokens "tagged" with back-ticks (`).

The special tag `` `temperature.cgi` `` near the end of the HTML file was mentioned previously. It tells the PicoWeb server to call the user-supplied pcode routine "`temperature`" at that point in the HTML code stream where that tag is located. This pcode routine writes the current temperature reading as ASCII text to the standard output stream. As a result, that string will appear in the Web page at that point and the current temperature reading will be displayed by the Web browser.

The following lines in Listing 1:

```
<input type=radio NAME=4 VALUE=0 `?testport.cgi?0x04`CHECKED{}>on<br>
<input type=radio NAME=4 VALUE=1 `?testport.cgi?0x04`{CHECKED}>off<br>
```

need to be explained. Any tag of the form:

`` `?routine.cgi?`*0xhh*`` *texteq* `{` *textneq* `}`

indicates the following actions should be preformed when that tag is encountered as part of the HTML code output stream:

1. The global parameter `psetparm_parm` is first set to the value given after the second "?", in this case the hexadecimal value *hh*
2. The pcode subroutine `routine` is then called, which is expected to set the pcode Z flag (indicating zero or non-zero)
3. If the pcode Z flag is set (indicating *equal*), output the text up to the next `{`, then discard the text up to the following `}`. The `{}`'s are discarded. In other words the text *texteq* is output.
4. If the pcode Z flag is cleared (indicating *not equal*), discard the text up to the next `{`, then output the text up to the next `}`. The `{}`'s are discarded. In other words the text *textneq* is output.

In the example shown, the pcode routine "testport" sets the pcode Z flag according to the state of the I/O bit in port D specified by `psetparm_parm`. A Z=1 indicates that the LED is on (i.e., port bit is low) and Z=0 indicates that the LED is off. Therefore in the case of our example:

- If port D output bit *hh* is low, output the text up to the next `{`, then discard the text up to the next `}`.
- If port D output bit *hh* is high, discard the text up to the next `{`, then output the text up to the next `}`

In other words, if bit *hh* is low, then output the first *tetxeq* string, else output the second *textneq* string.

In the example of Listing 1, the text "CHECKED" is output on either the first or second "radio button" HTML code line, depending upon the state of PicoWeb output bit PD4, the output pin which drives the user-controlled LED. The net result is that the radio buttons which are part of this *dynamic HTML code* will always reflect the current state of the PicoWeb's user-controlled LED. The radio button that is "pressed" reflects the true state of that LED.

The HTML FORM shown in Listing 1:

```
<FORM name=mfrm method=GET action="/">
<input type=radio NAME=4 VALUE=0 `?testport.cgu?0x04`CHECKED{}>on<br>
<input type=radio NAME=4 VALUE=1 `?testport.cgi?0x04`{CHECKED}>off<br>
<input type=submit VALUE="Set LED">
</FORM>
```

allows the user of the "WebLED" PicoWeb Web page to change the state of the user-controlled LED by clicking on the Web page button labeled "Set LED". When this button is clicked, the Web browser will *submit* an HTTP *GET request* to the PicoWeb server with the following URL:

`http://`*x.y.z.w*`/?4=`*s*

where *x.y.z.w* is the IP address of the PicoWeb server and *s* is the desired state of the LED, either 0 or 1. (The "radio buttons" in the form, depending upon exactly which one is "checked", cause a *value* named "4" to be sent to the PicoWeb server.) Firmware in the PicoWeb server needs to recognize HTTP *GET requests* of the form "`?`*p*`=`*s*", where *p* is a digit in the range of 0-7, and set the state of output pin `DB`*p* to *s*.

We cause this to happen by embedding the PicoWeb tag `` `pchk_port_url_parms.cgi` `` at the beginning of the Web page's HTML code. This tag will cause the pcode routine "`pchk_port_url_parms`" to be executed. This pcode routine scans the URL parameter area for the above described string and sets the PicoWeb I/O port bit accordingly.

Also note that any text strings "`$$VERSION$$`" will be replaced with the current PicoWeb firmware *version string* when HTML files are processed for downloading into the PicoWeb server's "file system".

Building the Project - The WebLED project can be built using the PicoWeb development system as in the following examples. The examples that follow assume that commands are being entered into an MS-DOS Prompt Window under Windows 95/98. Note that the environment variable PWDEV needs to be set to the full path name of the PicoWeb development system "base directory", and the PicoWeb development system's "bin" subdirectory needs to be in the PATH. Assuming that the PicoWeb development system "base directory" is C:\PWDEV, then the following commands will set up the environment for PicoWeb development:

```
C:>set PWDEV=C:\PWDEV
C:>set PATH=%PATH%;%PWDEV%\bin
```

Before beginning building the WebLED project two text files may need to be edited:

`ip` – must contain the IP address assigned to the PicoWeb server (e.g., `10.1.2.3`)
`ether` – must contain the Ethernet address assigned to the PicoWeb server (e.g., `0.1.2.3.4.5`)

After assigning IP and Ethernet address, the WebLED project can be compiled and readied for download into the PicoWeb server. This can be accomplished with the following command:

```
C:>pwbuild webled
```


This will cause the generation of a number of files, including an assembly language listing file (`webled.lst`) and linker map file (`webled.map`), as well as three data files needed to download the firmware and Web pages into the PicoWeb server hardware:

`webled.rom`	- Atmel micocontroller program memory binary data (a.k.a., ROM file)
`webled.ep`	- Atmel microcontroller on-chip EEPROM program binary data
`webled.el`	- PicoWeb "file system" and CGI pcode binary data to be loaded into the external serial EEPROM

Downloading the Project - The project files `webled.rom` and `webled.ep` can be loaded into the PicoWeb server hardware using a PC parallel port by attaching the PicoWeb programming cable to the connector on the PicoWeb server and to the PC's parallel port, then entering the following command:

```
C:>pwavrld webled
```

This will erase the program and EEPROM memory in the PicoWeb server's Atmel microcontroller, then download new firmware and data into the chip.

At this point, if the PicoWeb server is plugged into the network, it should now be active on the Ethernet. In fact, one should be able to "ping" the PicoWeb server from the development system PC using the PicoWeb server's assigned IP address.

At this point, the Web pages and CGI pcode routines stored in the file `webled.el` needs to be loaded into the PicoWeb server. This can be done with the following command:

```
C:>pwnetld webled
```

which will download the data in these files over the network using the PC's network card. Listing 3 shows a sample download session for the sample project "WebLED".

Note that the two download steps, `PWAVRLD` and `PWNETLD` can be run in a single step using the single command `PWLOAD`. Note also that because this project does not make use of any of the I/O pins which are shared with the PicoWeb programming cable, this cable can remain connected to the PicoWeb after the firmware download process.

Retrieving Web Pages -Assuming all went well with the PicoWeb load procedure, the PicoWeb server can now be accessed over the network as a Web server. You should be able to retrieve the Web page shown in Figure 1 by using a Web browser to access the following URL:

```
http://x.y.z.w/
```

where `x.y.z.w` is the IP address previously assigned to the PicoWeb server.

Note that in order for this to work from a Windows PC, that PC must have TCP/IP installed and properly configured to use an Ethernet adapter attached to the same network as the PicoWeb server. If your Web browser is configured to use a "proxy server" any kind, you will probably have to turn off that option, or change your browser preferences to bypass the proxy server when accessing the PicoWeb server's IP address. If you are having trouble with your Web browser, first verify that you can "ping" the PicoWeb server. This will verify that TCP/IP is properly configured to access the PicoWeb server.

07/07/00 6:45 PM

