How to Build a PicoWeb Project

Introduction

A PicoWeb development system *project file* is needed in order to build the data files needs to load a PicoWeb server with firmware and Web pages. PicoWeb project files must have a three-character file extension of "`.pwp`". Sample PicoWeb projects can be found in the PicoWeb development system "samples" directory. Each subdirectory in the "samples" directory should contain a single project file, plus any additional files needed to build the sample PicoWeb application. The sample project "WebLED", located in the directory `samples\webled` will be used in the examples which follow.

```
// application-specific preprocessor definitions (normally required)
#define XXXXXX ...

// application-specific HTML and image file names (at least one required)
index.htm

// application-specific CGI routines (optional)
mypcode.cgi

// application-specific assmbly language files (optional)
myasm.asm

// included application-specific pcode and/or AVR assembly language (optional)
#avr_reset
; this code is executed each time microcontroller is reset

#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

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

Listing 1 – PicoWeb Project File "Skeleton"

Listing 1 shows a skeleton PicoWeb project file. The various parts of the WebLED project file will be described individually in the following sections.

Comments on the project file are denoted by "//". All text after (and including) the "//" characters will be ignored by the PicoWeb project file pre-processor. In project file section where AVR assembly language is expected, comments also can be specified with a ";", in which case everything after and including the ";" is ignored by the AVR assembler. Because all PicoWeb source code is passed through a C-language preprocessor (namely gcc), C-style comments of the form "/*...*/" also are allowed wherever assembly language is allowed.

Note that some of the firmware source code used in the examples which follow makes use of PicoWeb *pcode*, a kind of assembly language for an interpreted "virtual machine". Please refer to the document "PicoWeb Pcode" for a complete description of pcode.

Table 1 – PicoWeb Project Preprocessor Defines

Preprocessor Define	Default Value	Description
BANNER	"\r\nPicoWeb Server\r\n"	Text string that is printed to the serial port each time the PicoWeb server is reset
CGI_INDEX_BASE	0x3800	Base address in serial EEPROM of CGI mapping table and user-supplied pcode
DEBUGGER	not defined	If defined, includes the serial port debugger firmware as part of the project build process
EEPROM_IP	not defined	If defined, specifies that the PicoWeb's IP address is stored in on-chip EEPROM (address will come from text file `ip`)
ENABLE_WATCHDOG	not defined	Enables the watchdog timer hardware on reset if defined
NO_ETHER_ADDR_ON_BOOT	not defined	Define this if you don't want a printout of your Ethernet address to the serial port on boot
PKT_WATCHDOG_MAX	250	Idle network packet watchdog timer limit in "slow idle loop" trip counts (recommended range: 100-254)
SEEPROM_IP	not defined	Defines address in serial EEPROM which holds the PicoWeb's IP address. Define if you want the IP address stored in serial EEPROM memory
SER_EEPROM_BASE_ADDR	0xA0	Base address of 24LC128 128 Kbit serial EEPROM chip
SERIAL_BAUD_DIVISOR	25	Serial port's baud rate clock divisor (varies with Atmel processor clock rate): **Divisor 8MHz 4MHz** 25 19200 9600 51 9600 4800 103 4800 2400 207 2400 1200
STATIC_IP_LSB STATIC_IP_MSB	not defined	Define these two 16-bit constants if you want the PicoWeb's IP address to be "hard coded" into program memory (defines high and low part of 32-bit IP address)
TCPPORT_CMD	911	TCP/IP port used debugger to receive commands via the Ethernet (not used unless DEBUGGER is defined)
URL_IN_FLASH	not defined	Define if you want your Web pages in on-chip flash (otherwise Web pages are stored in serial EEPROM)
USE_BOOTP	not defined	Use BOOTP protocol to get PicoWeb IP address

Preprocessor Defines Section

The preprocessor defines section is used to set certain parameters that tailor the PicoWeb firmware build process. Table 1 lists the standard PicoWeb Project preprocessor defines. The following is a sample project file define section:

```
#define BANNER "\r\nPicoWeb WebLED\r\n"
#define EEPROM_IP               /* use file "ip" for IP address */
#define ENABLE_WATCHDOG         /* use Atmel watchdog timer hardware */
#define DEBUGGER                /* include debugger firmware */
#define SERIAL_BAUD_DIVISOR 25  /* 19200 baud @ 8 MHz */
```

User-specified defines also may be placed in the project file section.

Web Page File List Section

The next section of the project file contains a list of Web pages which will be included in the project. This section lists one or more files which contain HTML code and/or images which will be stored in the PicoWeb's serial EEPROM memory as part of the Web server's "file system". The first file in the list is the default Web server "home page". Any files listed must have one of the following file extensions:

`.htm` – HTML code (text file)
`.html` – HTML code (text file)
`.jpg` – JPEG image
`.gif` – GIF image

The following is a sample project file list:

```
//
// application-specific HTML and image file names
//
webled.htm      // ii00  (default Web page)
bruce.jpg       // ii01
dave.jpg        // ii02
steve.jpg       // ii03
```

Everything after the //'s is comment text, and is ignored by the software which processes the project file. In the above example, the text string of the form `ii`*hh* after each file indicates the URL "file name" that will be assigned to each file after it is placed in the PicoWeb server's file system.

File Size Limit - No image or HTML code file can be larger than 8000 bytes. Files larger than this size will be truncated by the PicoWeb development system at the time the PicoWeb server's "file system" data image is prepared. Note that in cases where an image larger than 8K bytes must be displayed on a Web page, the larger image can be broken up into multiple pieces (or *tiles*), and these tiles can be displayed as a single seamless image using HTML "coding tricks", such as borderless tables. In the case of unusually large HTML code files, HTML *frames* can be used to provide virtually unlimited HTML code size by using multiple smaller HTML files in place of a single large HTML file. (Note that dynamic PicoWeb Web pages which use CGI routines to create HTML text "in the fly" must not deliver a single Web page that exceeds the 8000-byte limit.)

Web Page CGI Routine Section

The next section of the project file contains a list of Web user-supplied CGI routines which will be used by the Web pages listed in the previous section. This section lists zero or more *pcode entry point labels* for pcode routines which are normally provided in the following code sections of the project file. The extension ".cgi" must be appended to each pcode entry point label listed in this section.

The following is a sample project Web Page CGI Routine section:

```
//
// application-specific CGI routines
//
temperature.cgi // iu00 (returns ASCII temperature reading (deg-f)
```

The above line indicates that a user-supplied CGI pcode routine named "`temperature`" needs to be loaded into the PicoWeb server. Also, wherever the string `` `temperature.cgi` `` is found in any of the HTML source code files listed in the Web Page File List section as part of the project, a special *tag* will be inserted into those files. This special tag will cause the pcode routine "`temperature`" to be called at that point in the HTML code stream whenever a HTML file containing that tag is retrieved.

Processor Reset Code Section

The next section of the project file contains user-supplied, application-specific AVR assembly code which needs to be executed each time the PicoWeb processor is reset. This user-supplied code section is optional.

The following is a sample project Processor Reset Code section:

```
#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., pin PD4). The next two assembly language instructions initialize a I^2C digital thermometer chip.

Slow Idle Loop Code Section

The next section of the project file contains user-supplied, application-specific AVR assembly code which needs to be executed each time the PicoWeb processor traverses it's "slow idle" loop. The "slow idle loop" is executed about once per second. This user-supplied code section is optional. If present, this section must begin with the text line "`#avr_slow`".

The following is a sample project Slow Idle Loop Code section:

```
#avr_slow
        rcall check_heater    ; check heater (set LED as required)
```

The above AVR assembly language code calls the routine `check_heater` about once every second.

Fast Idle Loop Code Section

The next section of the project file contains user-supplied, application-specific AVR assembly code which needs to be executed each time the PicoWeb processor traverses it's "fast idle" loop. The "fast idle loop" is executed continuously, as fast as possible, when the PicoWeb processor has no other tasks to perform. The "fast idle loop" is where the PicoWeb server firmware polls the Ethernet controller chip. This user-supplied code section is optional. If present, this section must begin with the text line "`#avr_fast`".

The following is a sample project Processor Reset Code section:

```
#avr_fast
        rcall check_input   ; check for input from data port
```

The above AVR assembly language code calls the routine `check_input` as part of the PicoWeb processor's fast idle loop.

CGI/Assembly Code Section

The next section of the project file contains user-supplied, application-specific AVR assembly code and CGI pcode routine which are needed as part of the project, but which do not need to be executed at reset time, nor every time through one of the "idle loops". This section is where user-supplied CGI routines are placed. This user-supplied code section is optional. If present, this section must begin with the text line "`#avr_asm`".

The following is a sample project CGI/Assembly Code section (extracted from the sample PicoWeb project named "WebLED"):

```
#avr_asm
.eseg
temperature:
    pcall tempf_buf                    ; read temp in deg-F
    pcall decconv                      ; convert to decimal and output
    pret
.
.    (much source code removed)
.
.cseg
#include "muldiv.asm"
```

The above sample code section is a PicoWeb CGI pcode routine named `temperature` which, when called, will read the current temperature from the digital thermometer chip and output that reading in ASCII text to the *standard output stream*. If this routine is called because an HTML page is being retrieved which contains a special CGI tag pointing to this routine, then the *standard output stream* will be pointed at the HTML code stream. Therefore the latest ASCII temperature reading will be automatically 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.

The last two sample project file lines shown above are the last two lines in the project file. These lines cause firmware which implements integer multiply and divide, stored in the source code library file `muldiv.asm`, to be loaded into the program memory (`.cseg`) of the Atmel microcontroller as part of the project build process.

This concludes the description of PicoWeb *project files*. The next section discusses PicoWeb HTML coding, an important part of the PicoWeb server configuration process.

```
// This is a comment line which will be deleted
// Comments lines must begin with "///" on column 1 (i.e., no leading white-space!)
`t<html>
<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="/x">
<input type=radio NAME=4 VALUE=0 `?04 CHECKED{}>on<br>
<input type=radio NAME=4 VALUE=1 `?04{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" (`hello.htm`)

PicoWeb HTML Coding

Listing 1 shows the contents of the "home page" from the sample project "WebLED". (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 (`). For example, the first line contains **`t**, not something one normally finds in an HTML document. This PicoWeb *tag* causes an HTTP text header of the form:

```
HTTP/1.0 200
Content-type: text/html
```

to be output when this document is retrieved from the PicoWeb server. Those familiar with low-level CGI programming will recognize the significance of this. This *HTTP header* tells a Web browser exactly what kind of document is being returned from the Web server in response to an HTTP *GET request*. Every proper HTML document loaded into the PicoWeb server should begin with the string "`t". (Note that many Web browsers (e.g., Netscape) *will* display without error retrieved Web pages which are missing this HTTP text header. However, this is not always the case!)

The special tag `temperature.cgi` was discussed 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.

The following lines in Listing 1:

```
<input type=radio NAME=4 VALUE=0 `?04 CHECKED{}>on<br>
<input type=radio NAME=4 VALUE=1 `?04{CHECKED}>off<br>
```

need to be explained. A tag of the form:

`?*hh text0* { *text1* }

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

- If port D output bit *hh* is low, output the text up to the next {, then discard the text up to the next }. The {}'s are discarded.
- If port D output bit *hh* is high, discard the text up to the next {, then output the text up to the next }. The {}'s are discarded.

In other words, if bit *hh* is low, output the first *text0* string, else output the second *text1* 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.

Table 2 - PicoWeb HTML Tags

Tag	Meaning
`t	Output HTML header for text/html
`?*hh text0* { *text1* }	Conditional text output according to the state of PicoWeb pin DB*n* where *n* is decimal equivalent of the two hex digits *hh*. If pin DB*n* is 0, output text "*text0*"; if pin DB*n* is 1, output text "*text1*".
`*routine*.cgi`	Call user-supplied CGI routine "*routine*" ("*routine*" must be listed in the project file's Web Page CGI Routine Section)
`7*hh*	Call user-supplied CGI routine number *hh* in the CGI mapping table (*hh* is exactly two hex digits)

The HTML form shown in Listing 1:

```
<FORM name=mfrm method=GET action="/x">
<input type=radio NAME=4 VALUE=0 `?04 CHECKED{}>on<br>
<input type=radio NAME=4 VALUE=1 `?04{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/x?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 which one is "checked", cause a *value* named "4" to be sent to the PicoWeb server.) Firmware in the PicoWeb server recognizes HTTP *GET requests* of the form "x?*p*=*s*", where *p* is a digit in the range of 0-7, and will set the state of output pin DB*p* to *s*.

Table 2 lists the supported PicoWeb HTML tags.

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".

Any lines beginning with "//" will be deleted from HTML code files before the HTML code is processed for download into the PicoWeb server.

Setting Up to Build 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. Also, the examples below assume that the PicoWeb server is connected to the same network as the Windows PC used for PicoWeb development and that the PC and the PicoWeb server are both assigned IP addresses in the same Ethernet network subnetwork.

Before PicoWeb development can begin, two parameters in the MS-DOS command line environment needs to be properly configured. The environment variable PWDEV *must 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
```

Using AUTOEXEC.BAT - If you want to setup for PicoWeb development in a more permanent way, you can add the following command lines to the file `C:\AUTOEXEC.BAT`:

```
set PWDEV=XXXXXX
set PATH=%PATH%;%PWDEV%\bin
```

where `XXXXXX` is the full path name of the PicoWeb development system "base directory", for example "C:\PWDEV". Reboot your computer to cause the new environment variable (PWDEV) and path to take effect.

Using a Windows Shortcut Icon - Alternatively, under Windows 95/98, you can set up a special Ms-DOS Prompt Window icon on your desktop which is specially configured to do PicoWeb development.

To make such a desktop shortcut, first create a new BATCH file (i.e., a text file ending in `.BAT`) with the above command lines. Then create a new "shortcut" icon on your Desktop which points to the file DOSPRMPT.PIF (located in your Windows directory). Then change the "Properties" on the newly created shortcut icon, under the "Program" tab, to specify your new "batch file" containing the above commands.

You also will probably want to change the "Working" directory entry under the same shortcut Properties "Program" tab to specify the directory where you will be doing your firmware development.

Warning: In order to provide for additional environment string space needed to store the PWDEV environment variable, you may need to change "Initial environment" under the "Memory" Properties tab from "Auto" to a value like 4096.

Using the new short-cut icon you should now be able to start a "MS-DOS Prompt" Window for the purposes of PicoWeb development.

Warning: None of the PicoWeb development batch files will work unless the above environment variables are set properly when running under the required "MS-DOS Prompt" Window.


```
C:>pwbuild webled
Deleting output files
Making command table.
Setting IP address
10.1.2.3
Setting Ethernet address
0.1.2.3.4.5
Processing HTML/images/CGI.
Version: 1.35
ii00: webled.htm (685 bytes)
ii01: bruce.jpg (1306 bytes)
ii02: dave.jpg (1304 bytes)
ii03: steve.jpg (1383 bytes)
4627 bytes of HTML code/images total
Preprocessing.
Warning: import import_udp is being stubbed.
Warning: import import_unknown_ipaddr is being stubbed.
Assembling.
X:\avr\firmware\v36\bin\avrasm -g -w webled.pi webled.lst webled.rom
AVRASM: AVR macro assembler version 1.21  (Mar  5 1998 01:21:00)
Copyright (C) 1995-1997 ATMEL Corporation
Assembling 'webled.pi'
Including  'webled.asi'
Program memory usage:
Code             : 1216 words
Constants (dw/db): 2435 words
Unused           :    0 words
Total            : 3651 words
Assembly complete with no errors.
0 warnings/errors.
Cracking webled.eep into webled.ep and webled.el
SEEPROM base B800
Done.
```

Listing 2 - Sample PicoWeb Build Session

Building the Project

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`, as well as four 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 CGI pcode binary data, to be loaded into the external serial EEPROM
`webled.dat` – PicoWeb "Web file system" data (i.e., HTML code and images)


```
C:>pwavrld webled
Loading PicoWeb program memory with 'webled.rom'
X:\avr\firmware\v36\bin\avr    -ce -lp webled.rom
Found LPT1 (I/O base 0x378)
Sending the chip ERASE command
writing Flash memory (0-3650)
................................
Loading PicoWeb EEPROM memory with 'webled.ep'
X:\avr\firmware\v36\bin\avr    -le webled.ep
Found LPT1 (I/O base 0x378)
writing EEPROM memory (0-10)
.
Enabling PicoWeb server'
X:\avr\firmware\v36\bin\avr    -en
Found LPT1 (I/O base 0x378)
PicoWeb AVR firmware load complete.

C:>pwnetld webled
Setting 8515 EEPROM 1FF to FF.
Loading PicoWeb CGI pcode into EEPROM via network
.
Loading PicoWeb HTML/images into EEPROM via network
.......................
Resetting 8515 EEPROM 1FF to 0.
PicoWeb network load complete.
```

Listing 3 - PicoWeb Sample Download Session

Listing 2 is sample output from a PicoWeb development system "build" using the sample project "WebLED".

If any errors are encountered during the build process, and those errors are caused by problems with the firmware source files, then the AVR assembler will emit error lines with the name of the offending source file along with a source file line number. The following is a sample error line:

```
webled.pwp(128):(5931) Undefined variable referenced
```

Note that the second number in ( )'s is the line number in the file `webled.pi`, the source code file that is sent to the AVR assembler after all the PicoWeb development system's pre-processing is complete.

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 EEPOM 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.

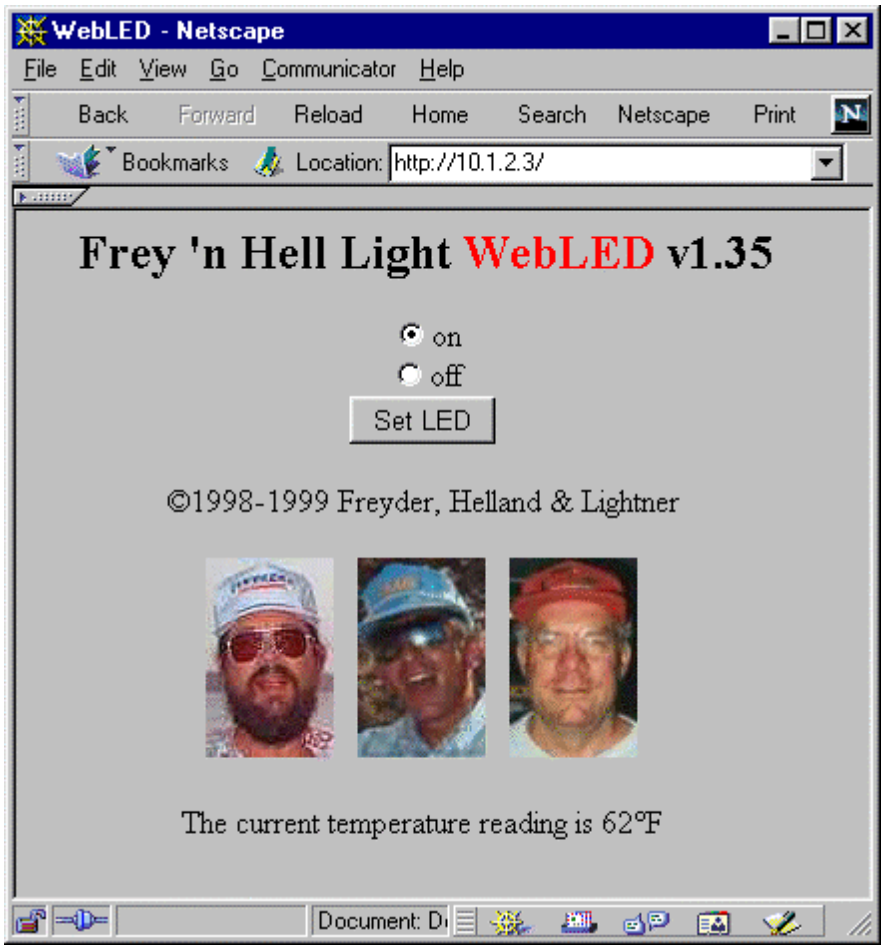

Figure 1 - WebLED Home Page

At this point, the Web pages and CGI pcode routines store in the files `webled.dat` and `webled.el` need 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`.

Retrieving Web Pages

Assuming all goes well with the PicoWeb load procedure, the PicoWeb server can now be accessed over the network as a Web server. If you used the sample project "WebLED", 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.

Table 3 - PicoWeb Server URLs

URL	Description
/	Return document ii00.(i.e., the "home page")
/ii*hh*	Return document number *hh*. Documents numbers are 2-digit hex values. (anything after *hh* in the URL is ignored.)
/iuhh	Call firmware routine number *hh*. (mostly useful for testing user-supplied CGI routines)
/x?*n*=*value*	Set digital I/O port DB*n* to *value*, where *value* is either 0 or 1

The PicoWeb Server's prime function is to return Web pages and images in response to HTTP GET requests directed to URLs targeting its HTTP server. The PicoWeb Server's firmware responds to such URLs directed to TCP port 80 in a conventional manner. Because the PicoWeb Server does not have a true "file system", URLs trigger dedicated routines in the firmware, as opposed to simply returning the contents of a disk file. A summary of the standard URL's implemented in the PicoWeb Server's demonstration firmware are shown in Table 3. (Note that the "`http://x.y.z.w`" part of the URL does not appear in the table.)

Table 4 – PicoWeb Server Debug Commands

Command	Description
dm XXXX nn	dump SRAM from XXXX...XXXX+nn-1
de XXXX nn	dump EEPROM from XXXX...XXXX+nn-1
ds XXXX nn	dump serial EEPROM from XXXX...XXXX+nn-1
wm XXXX YY	write SRAM address XXXX with byte YY
we XXXX YY	write EEPROM address XXXX with byte YY
ws XXXX YY	write serial EEPROM address XXXX with byte YY
l	toggle TCP packet logging on/off
pd n	control p-code debug trace (0=off; 1=on)
PC XXXX	call p-code routine at address XXXX
R	reset processor
^C	reset processor (serial port only)

PicoWeb Debugger

The PicoWeb server firmware library contains a simple, extensible debugger which provides for things like memory dumps, EEPROM memory alteration, pcode and network tracing control, etc. The debugger firmware will be included in a PicoWeb build by adding "`#define DEBUGGER`" to the beginning of the PicoWeb project file. Debugger commands can be entered via the serial port, or via the network using a Web browser and a URL which references a special TCP port (i.e., port 911). The built-in PicoWe debugger commands are listed in Table 4.

The format of a debugger command URL is as follows:

`http://x.y.z.w:911/command`[[+*parameter1*]+*parameter2*]

where `x.y.z.w` is the IP address previously assigned to the PicoWeb server and parameter1 and parameter2 are optional, depending upon the debugger command. Any results from executing a debugger command will be returned as a Web page.

For example, the URL:

```
http://x.y.z.w:911/dm+60+80
```

will list the contents of the first 128 bytes of the microcontroller's SRAM.

New debugger commands easily can be added (or deleted to save program code space). The list of active debugger commands is maintained in the PicoWeb library file `cmd.txt`. The firmware source code for most debugger commands can be found in the library file `cmd.asm`.

Table 5 - Predefined AVR Assembly Macros

Macro	Operands	Description	Operation	Flags
`addw`	Wd, Wr	Add Words without Carry	Wd ← Wd + Wr	Z,C,N,V,H
`addwi`	Wd, K	Add Immediate to Word	Wd ← Wd + K	Z,C,N,V
`andwi`	Wd,K	Logical AND Word with Immediate	Wd ← Wd · K	Z,N,V
`clrw`	Wd	Clear Word	Wd ← 0	None
`cmpw`	Wd,Wr	Compare Words without Carry	Wd - Wr	Z,C,N,V,H
`cmpwi`	Wd,K	Compare Word with Immediate	Wd - K	Z,C,N,V,H
`decw`	Wd	Decrement Word	Wd ← Wd - 1	Z,C,N,V,H
`incw`	Wd	Increment Word	Wd ← Wd + 1	Z,C,N,V,H
`ldsbw`	Wr	Load Byte from SRAM into Word	Wd ← (k)	None
`ldsw`	Wd,k	Load Word Direct from SRAM	Wd ← (k+1,k)	None
`movw`	Wd,Wr	Move Words	Wd ← Wr	None
`movwi`	Wd,K	Move Immediate into Word	Wd ← K	None
`popw`	Wd	Pop Word from Stack	Wd ← STACK	None
`pushw`	Wr	Push Word onto Stack	STACK ← Wr	None
`shlw`	Wd	Logical Sift Left Word	Wd ← Wd << 1	Z,C,N,V,H
`shrw`	Wd	Logical Sift Right Word	Wd ← Wd >> 1	Z,C,N,V,H
`stsw`	Wd,k	Store Word Direct to SRAM	(k+1,k) ← Wr	None
`subw`	Wd, Wr	Subtract Words without Carry	Wd ← Wd - Wr	Z,C,N,V,H
`subwi`	Wd, K	Subtract Immediate from Word	Wd ← Wd - K	Z,C,N,V

Notes:

- Wd, Wr and K represent 16-bit values.
- Wd and Wr are one of the *even-numbered* AVR registers (i.e., r0, r2, r3, ..., r30) or registers X, Y, or Z). These registers are paired with the next higher register number to make a 16-bit value.
- k is a constant SRAM address.

PicoWeb Assembly Language

The PicoWeb server firmware is written in a mixture of standard AVR assembly language and PicoWeb *pcode*, a kind of assembly language for an interpreted 16-bit "virtual machine". A complete description of pcode can be found in the document "PicoWeb Pcode".

The PicoWeb firmware also makes use of a number of predefined AVR assembly language macros. These definitions can be found in the library file `picbin.inc`, which is included as part of every PicoWeb project build. A number of the predefined macros are used to allow the convenient manipulation of 16-bit data, by making use of adjacent pairs of even/odd 8-bit AVR registers. Table 5 lists a number of the predefined "16-bit word-oriented" AVR assembly macros used as part of the PicoWeb server build processing.

Sample PicoWeb Projects

A number of sample PicoWeb projects can be found in the "samples" directory. The various subdirectories in the "samples" directory each should contain a single project file, plus any additional files needed to build the

sample PicoWeb application. For example, the sample project “WebLED” used in the discussions above is located in the directory `samples\webled`. Note that a number of the samples subdirectories also contain an Adobe Acrobat document describing the project in some detail. These documents will have the same name as the project, with a file extension of `.pdf` (e.g., `webled.pdf`).

The sample PicoWeb projects are a good source of practical information on how to use the PicoWeb server in a real application. For example, the `hello` project show how a basic “hello world” Web page (and associated JPEG image) can be built and loaded into the PicoWeb server. The `webled` project shows how to use HTML FORMs to control hardware attached to the Atmel microcontroller’s data ports (i.e., the on-board LED). The `serdev` project is an example of how to access an external device using the PicoWeb’s serial port.

Editing PicoWeb Source Files

The PicoWeb build process will produce an error listing if and when errors are detected during the assembly process using the AVR assembler. The PicoWeb development system processes the raw error stream from the AVR assembler and produces error messages which should reflect the line number in the original source file that is the true source of the assembly error. This means that in order to locate and correct a source line which is causing an error, a Windows-based text editor which can go to a specific line number is probably required. The standard release of Windows 95/98 does not include such a text editor.

The PicoWeb development system includes a text editor called “PICOIDE” which can be used as a source file editor for PicoWeb firmware development. This editor can be commanded to seek to a particular line number in a text file. Also, this editor, if given a file with the error stream from the PicoWeb build process, can automatically seek to the source files and source file lines which are causing assembly errors. Please refer to the documentation on the “PICOIDE” editor.

Development System Caveats

This section lists some of the known problems with the PicoWeb development system.

AVR Assembler - Version 1.21 of Atmel's AVR assembler *must be used* with the supplied PicoWeb Server software package. That version (`AVRASM.EXE`) is supplied with this software package. That version seems to have problems with "`.db`" AVR assembly directives in "`.eseg`" sections. Do not use "`.db`" directives! Instead use “`.dw`” directives to allocate storage in the EEPROM segment.

Windows NT - At the current time the PicoWeb PC parallel port programming tool (i.e., `AVR.EXE`) does not function correctly under Windows NT. Therefore, the PicoWeb firmware download utility (i.e., `PWAVRLD.BAT`) is restricted to Windows 95 and/or Windows 98.

Firmware Download - We have had reports from the field of the Atmel microcontroller occasionally entering a "confused" state when a PicoWeb Server is power-cycled with the programming cable(s) attached. In this state the Atmel microcontroller will repeatedly fail to download new firmware. To correct this problem, unplug *all* of the cables from the PicoWeb server, wait a few seconds, re-apply DC power to the PicoWeb Server, then re-attach the other cables. The source of this problem appears to be "leakage" current supplied to the PicoWeb via the cables attached to the PC, preventing a "clean" power-up sequence, as required by the Atmel flash programming circuitry.

