How to Build a PicoWeb Project

Version 2.00

July 22, 2000

Lightner Engineering
8551 La Jolla Shores Drive
La Jolla, California 92037-3044
Phone: +1-858-551-4011 FAX: +1-858-551-0777
Email: support@lightner.net
URL: http://www.picoweb.net/

Table of Contents

Introduction

This document describes the steps needed to prepare a project for downloading into a PicoWeb server using the GNU-based PicoWeb development system (version 2). These steps are described by way of a sample PicoWeb project "WebLED". (Users of PicoWeb development system software released prior to July 2000 also should read the document "Converting PicoWeb Projects to the New Development System".)

The hardware of the PicoWeb server contains two integrated circuits that have volatile and non-volatile storage which needs to be configured before the PicoWeb server can be used. One is the Atmel 90S8515 microcontroller and the other is an I^2C serial EEPROM chip. The Atmel microcontroller chip has three kinds of memory: static RAM (512 bytes), flash-based program memory (8 Kbytes), and on-chip EEPROM (512 bytes). The I^2C serial EEPROM chip holds between 16 and 32 Kbytes of data, depending upon the PicoWeb version. The configuration of the "initialized" portion on the static RAM, the two EEPROM memory blocks (on-chip and external) and the microcontroller's flash-based program storage is defined completely by the PicoWeb *project file*.

The project file specifies exactly which HTML and image files are to be loaded into the PicoWeb's serial EEPROM-based "file system". The project file also specifies what firmware source code and object files are included as part of a particular PicoWeb project. The PicoWeb project file and its resultant output files are processed in multiple stages. These steps include the use of a "C-language style" macro preprocessor, a PicoWeb pcode preprocessor, an AVR instruction assembler, an object file linker, and finally a two-part firmware loader.

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

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

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

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

// linker directives (optional)
link add myobject.o
link search mylibrary.a

// 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"

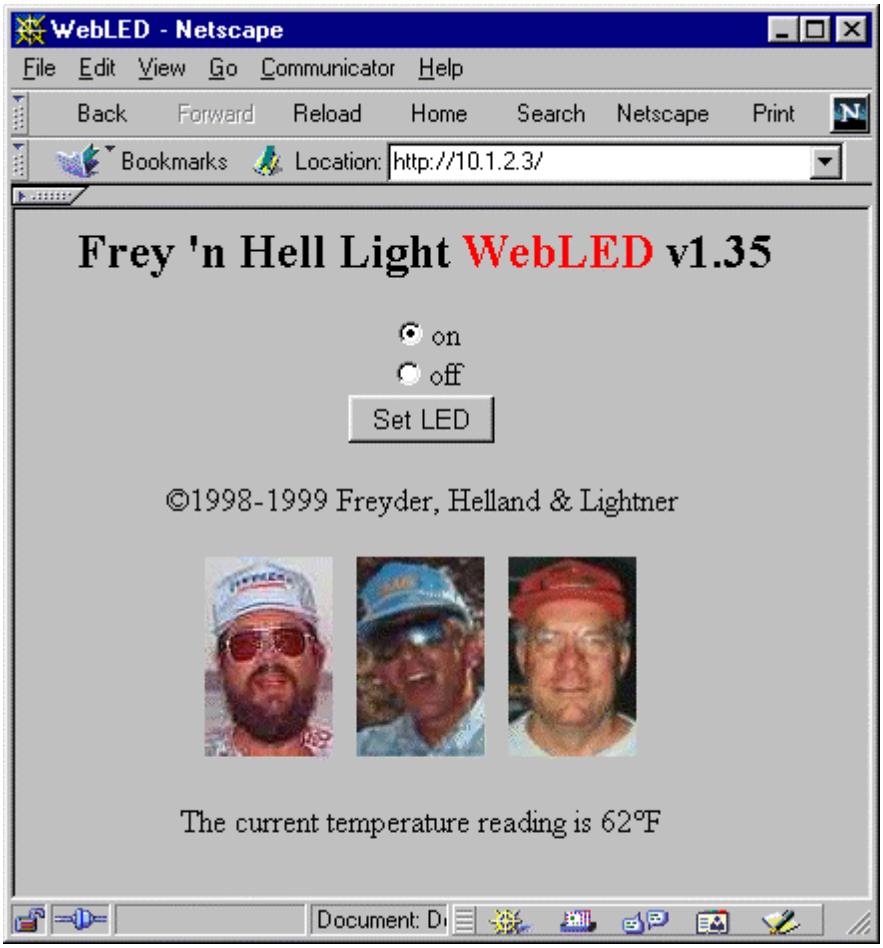

Figure 1 - WebLED Home Page

Project File

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 file names *must have* a three-character file extension of "`.pwp`". Sample PicoWeb projects can be found in the PicoWeb development system's "samples" directory. Each subdirectory in the "samples" directory normally contains 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.

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

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 cpp), C-style comments of the form "`/*...*/`" also are allowed wherever assembly language is allowed. (Note that this same preprocessor will issue a warning message whereever it finds single "`'`" characters within comments.)

PicoWeb Pcode
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 PicoWeb 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
BAUD_RATE	19200	Defines the serial port baud rate. Note that #define CLOCK must be correctly set. Not all baud rates are available at all processor clock rates. *(Consult Atmel's AT90S8515 datasheet for details.)*
CLOCK	7372000	Defines the clock rate of Atmel microcontroller (Hz). This is used to set the UART baud rate divisor and for other timing-related calculations.
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
NET_CONFIG_IP	not defined	Define this if to allow the PicoWeb's IP address to be changed over the Ethernet using the program `setip`
NO_ETHER_ADDR_ON_BOOT	not defined	Define this to inhibit printout of the PicoWeb's Ethernet address to the serial port on reset
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
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)
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 "defines section":

```
#define BANNER "\r\nPicoWeb WebLED\r\n"
#define EEPROM_IP         /* use file "ip" for 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 Hz) */
#define BAUD_RATE 19200   /* serial port baud rate */
```

User-specified defines also may be placed in the project file section. As part of the PicoWeb project "build" process, these `#define` statements are placed in a file named using the project name followed by "`.h`".

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 either must have one of the following file extensions:

`.htm` – HTML code (text file)
`.html` – HTML code (text file)
`.txt` – ASCII text file
`.jpg` – JPEG image
`.gif` – GIF image
`.png` – PNG image
`.js` – Javascript (text file)
`.cla` – Java applet (byte-codes)
`.class` – Java applet (byte-codes)

or they must be followed by a HTTP *content type* field specification. The following is a list of content type fields:

```
none
text/html
text/plain
image/jpeg
image/gif
image/png
application/octet-stream
```

If no file content type specification is given after the file name, then an appropriate content type will be assigned according to the file's extension. Whenever a file is supplied by the PicoWeb server in response to a Web browser's GET request, the following HTTP header information will proceed the delivery of the file's contents:

`HTTP/1.0 200¿`
`Content-type:` *content type*`¿`
`¿`

where ¿ indicates a new-line character. Those familiar with low-level CGI programming will recognize the significance of the HTTP header, which tells a Web browser exactly what kind of document is being returned from the Web server in response to an HTTP GET request.

Any PicoWeb-supplied file may be assigned an arbitrary HTTP *content type* by providing a content type field after the file's name in the project file. The *content type* fields are not checked and can be any arbitrary string of characters.

Note that the special *content type* field "`none`" specifies that the PicoWeb server should not proceed the file delivered in response to an HTTP GET request with anything. In this case, the file itself (or the user-supplied CGI code that executes) must supply the needed HTTP header. (Note that certain Web browsers (e.g., Netscape) will display, without error, retrieved Web pages which are missing the HTTP header. However, this is not the case with all Web browsers.)

The following is a sample PicoWeb project "file list":

```
//
// application-specific HTML and image file names
//
webled.htm                // home page (because it is listed first)
bruce.jpg
dave.jpg
steve.jpg   image/jpeg   // specifies JPEG content type field
```

Everything after the //'s is comment text, and is ignored by the software which processes the project file.

File Size Limit

No image or HTML code file should 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 8 Kbytes 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 Public CGI Routine Section

The next (optional) section of the project file contains a list of user-supplied CGI routines which will be used by the Web pages listed in the previous section. This section lists zero or more public *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 CGI pcode routine named "`temperature`" exists somewhere in the PicoWeb servers firmware. Because this routine is listed here in the project file, it can be accessed externally with an HTTP GET request using the following URL:

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

where `x.y.z.w` is the IP address assigned to 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 logically 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 an HTML file containing that tag is retrieved. Note that this tag processing applies to any global pcode subroutine label, not just those routines listed in this section.

Linker Directive Section

The next section of the project file is reserved for optional directives to the PicoWeb development system's linker. This section lists zero or more linker directives from the following list:

```
link add object.o
link search library.a
```

where *object.o* is the name of a PicoWeb object file and *library.a* is the name of a PicoWeb library archive. The "`link add`" directive causes the linker to unconditionally include the specified object file. The "`link search`" directive causes the linker to search the specified archive file to resolve any pending unresolved global symbol references.

The following is a sample project Linker Directive section:

```
// linker directives (optional)
link add myobject.o
link search mylibrary.a
```

The "`link add`" directive causes the linker to unconditionally include the object file "`myobject.o`" as part of the PicoWeb firmware image. The "`link search`" directive will cause the linker to search the library file "`mylibrary.a`" in order to resolve any global symbol references.

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. If present, this section must begin with the text line "`#avr_reset`".

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 call a routine that initializes an 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 its "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 its "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
    pprintswi [buf]                 ; output as decimal ASCII
    pret
.
.    (much source code removed)
.
```

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 when this routine is called the *standard output stream* will be directed at the HTML code stream. As a result, the latest temperature reading (as ASCII text) will be automatically inserted into the HTML code stream in place of this routine's tag. Note the `.eseg` directive which causes this pcode routine to reside in the PicoWeb server's external serial EEPROM memory.

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!)
<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.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>
&#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 Tags

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 (`).

The special tag `` `temperature.cgi` `` at the end of the 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.

Table 2 - PicoWeb HTML Tags

Tag	Meaning
`` `*routine*.cgi` ``	Call pcode routine "*routine*".
`` `*routine*.cgi?*parm*` ``	Set global parameter psetparm_parm to "*param*", then call pcode routine "*routine*".
`` `?*routine*.cgi?*parm*` *texteq* { *textneq* } ``	Set global parameter psetparm_parm to "*param*", then conditionally call pcode routine "*routine*". Conditionally output text according to the state of the pcode "Z flag". If Z=1, output text "*texteq*"; if Z=0, output text "*textneq*".
`` `?*routine*.cgi?*parm*@*label*.cgi` ``	Set global parameter psetparm_parm to "*param*", then conditionally jump to the label named "*label*" according to the state of the pcode "Z flag". If Z=1 then the jump is taken.
`` `=*label*` ``	Place a label named "*label*" in the pcode which outputs the HTML code which follows. May be "called" from other HTML code using `` `*label*.cgi` ``.
`` `@*label*.cgi` ``	Unconditional jump to pcode "*label*". Normally this is a location in an HTML code file labeled using the tag `` `*label*.cgi` ``.
`` `t ``	Removed if first item in a PicoWeb HTML file (compatibility)

Notes:

1. The affect of the "Z flag" can be reversed in the conditional call/jump tags (i.e., `?...`) by following the leading "?" with a "!" (i.e., `?!*routine*.cgi?*parm@label*.cgi`).
2. The value "*param*" shown in the table can take any of the following forms:
   - 0x*hhhh* A 16-bit hexadecimal value "*hhhh*" (e.g., 0x8000)
   - *ddddd* A 16bit decimal value "*ddddd*" (e.g., 32767)
   - "*string*" A text string delimited by double-quotes (e.g., "hello")

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

Table 2 lists all of the supported PicoWeb HTML tags. Only tags in HTML files are recognized (i.e., .htm and .html files)

PicoWeb HTML FORMs

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>
```

also 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 `PD`*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. This is a very special case of URL parameter parsing. The next subsection discusses a more general way of scanning for URL parameters from FORMs.

Processing URL FORM Parameters

A number of pcode instructions provide for simple processing of parameters passed as part of the HTTP GET "URL line". These parameters are typically passed to the PicoWeb server when an HTML FORM "submit" button is pressed. If you have a project that needs to access parameters of this kind, then we strongly suggest that you take a look at the PicoWeb samples and see how this can be done with these pcode instructions. (The sample project "serdev" makes use of these new instructions.)

Here is a sample CGI pcode routine that will print the string which follows the parameter "`S=`" anywhere in a URL line:

```
print_string:
   purlparm buf,"S="      ; search for command string (buf gets offset)
    pjumpne 1f            ; skip ahead if not found
    pprinturl buf,0       ; write text after M= (start at offset in buf)
1:
    pret
```


And, here is sample code that will convert to binary, then print the value of the decimal integer value after the URL parameter "I=" anywhere in a URL line:

```
print_int:
    purlparm buf,"I="     ; search for command string (buf gets offset)
    pjumpne 1f            ; exit if not found
    purl2int buf+2,buf    ; found...convert to integer (store at buf+2)
    pprintswi [buf+2]     ; print signed integer as text
1:
    pret
```

PicoWeb HTML Special Processing

Version String
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".

Comments
PicoWeb HTML code files with any lines beginning with "`//`" *(in column one)* will be replaced by a blank line before the HTML code is processed for download into the PicoWeb server.

Note that the Javascript coding "trick":

```
<script>
<!-- //
...
  // -->
</script>
```

therefore will not work unless the line beginning with "//" is proceeded by one or more spaces as shown.

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 LAN 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's "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
```

Beware! The "`set PATH`" statement extends the length of your current PATH by adding the string "`;C:\PWDEV\bin`" to the end of your current PATH. If your current PATH is unusually long, the added PATH information needed by the PicoWeb development software may be truncated. You can check for this condition by typing "PATH" to get a printout of the modified PATH.

Using AUTOEXEC.BAT
If you want to setup your PC 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
setting IP address
10.1.2.3
setting Ethernet address
0.1.2.3.4.5
making command table.
processing project file webled.pwp
Version: 2.03
4813 bytes of HTML code/images total
assembling webled.asm
avr-as -W  -alms=webled.lst -o webled.o webled.s
linking project webled.elf
making load files
3695 words in flash (90% used)
12 bytes in EEPROM (2% used)
5686 bytes in SEEPROM (38% used)
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 addresses, 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

Listing 2 is sample console 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:143(webled.s:1745) Error: unknown opcode
```

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

```
C:>pwavrld webled
Loading PicoWeb program memory with 'webled.rom'
Reading PicoWeb's current firmware
Found LPT1 (I/O base 0x378)
Program memory file webled.crm written (0-3735)
EEPROM file webled.cee written (0-511)
3651 changed locations in file webled.rom (@0x0001,0xc369->0xc311)
C:\pwdev\bin\avr   -ce -lp webled.rom
Found LPT1 (I/O base 0x378)
Sending the chip ERASE command
writing Flash memory (0-3694)
.................................
Loading PicoWeb EEPROM memory with 'webled.ep'
C:\pwdev\bin\avr   -le webled.ep
Found LPT1 (I/O base 0x378)
writing EEPROM memory (0-11)
.
Enabling PicoWeb server
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
load..........................
read..........................
Resetting 8515 EEPROM 1FF to 0.
PicoWeb network load complete.
```

Listing 3 - PicoWeb Sample Download Session

Downloading the Project

Downloading firmware into the PicoWeb server is a two-step process. The first step downloads data into the Atmel microcontroller using a PicoWeb programming cable attached to a PC parallel port. The second step uses the Ethernet to download Web pages and external pcode into the PicoWeb's serial EEPROM chip. Listing 2 is sample console output from a PicoWeb development system "firmware download" session using the sample project "WebLED".

Downloading the Microcontroller

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. Note that if a PicoWeb project that makes use of any of the Port B I/O pins which are shared with the PicoWeb programming cable, then the programming cable should be disconnected after this firmware download process is complete.

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.

Downloading the Web Pages
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".

Optimized Downloading
The two download steps, `PWAVRLD` and `PWNETLD` often can be run in as single step using the command `PWLOAD`. Note that if a PicoWeb project does not make use of any of the port B I/O pins which are shared with the PicoWeb programming cable, then the one-step downloading using `PWLOAD` is possible. This is because the programming cable can remain connected to the PicoWeb after the firmware download process is complete.

Also, the downloading of firmware into the PicoWeb over the PC parallel port can often times be sped up when using the `PWAVRLD` and `PWLOAD` commands. Adding the parameter "`-x`*ddd*" *after* the project file name on the command line, where *ddd* is a decimal integer in the range of 500-2000, controls the parallel port timing. Smaller values for *ddd* will speed up the firmware download. Note however that too small a value for *ddd* will cause the firmware download to fail.

Specifying PC Programming Port
By default, the PicoWeb programmer (`AVR.EXE`) will search all possible PC parallel ports for a PicoWeb programming cable (i.e., LPT0, LPT1 and LPT2). The programming software locates the cable by looking for a "loop-back" wire in the PicoWeb's programming cable. If you want to prevent this search from happening, you can specify which parallel port to use by adding the command line parameter "`-lpt`*N*" (where *N* is 0, 1, or 2) *after* the project file name on the `PWAVRLD` or `PWLOAD` command lines.

Table 3 - PicoWeb Server URLs

URL	Description
/	Return the first document listed in the project file's "file list section" (i.e., the "home page")
/*file*	Return the document named "*file*" which was listed in the project file's "file list section"
/*routine*.cgi	Call firmware pcode routine named "*routine*" and return any results sent to the *standard output stream*. (The routine must have been listed in the project file's "public CGI routine section")
/*file*?*n=value[&n=value...]*	Return the document named "*file*" as part of an HTML FORM's "ACTION=GET". One or more FORM parameters/value pairs follow the ? and are separated by &'s

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 software 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.

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 in fact 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 standard 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 in range *XXXX...XXXX+nn-1*
`de` *XXXX nn*	dump on-chip EEPROM in range *XXXX...XXXX+nn-1*
`ds` *XXXX nn*	dump serial EEPROM in range *XXXX...XXXX+nn-1*
`wm` *XXXX YY*	write SRAM address *XXXX* with byte *YY*
`we` *XXXX YY*	write on-chip 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*	set p-code debug trace flag to *n* (0=off; 1=on)
`PC` *XXXX*	call p-code routine at address *XXXX*
`R`	reset processor (no Web page returned!)
`^C` (control-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 can 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 PicoWeb 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 directory in assembly language files (.asm) which begin with "cmd". A custom debugger command set for a given project can be created by copying the file "`cmd.txt`" from the library directory to the project directory and then editing the local copy of "`cmd.txt`".

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.

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".

PicoWeb Pcode
A complete description of pcode can be found in the document "PicoWeb Pcode", including information on creating new pcode instructions written in native AVR assembly language. Because user-supplied CGI routines are written in pcode, a basic understanding of PicoWeb pcode is required in order to add and/or change these routines. A listing of all predefined pcode routines can be found in the library file `pcode.def`. This file is consulted as part of pcode preprocessing when "building" a PicoWeb project.

AVR Assembly Language
In addition, a basic understanding of the Atmel 90S8515's AVR assembly language is needed. The Atmel datasheet titled "AT90S4414/AT90S8515 - 8-bit AVR Microcontroller with 4K/8K Bytes In-System Programmable Flash" and the document "AVR Instruction Set" provide detailed information about programming the microcontroller in AVR assembly language. Another source of information is firmware source code which can be found in the sample project directories and in the PicoWeb library directory (`lib`).

AVR Assembly Macros
The PicoWeb firmware also makes use of a number of predefined AVR assembly language macros. These definitions can be found in the library file `picoweb.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.

Table 6 – PicoWeb Linker Sections

On-Chip Flash Program Memory (8 Kbytes)	
Assembler Directive	**Meaning**
`.text`	AVR program code and data
`.section reset`	PicoWeb `#avr_reset` section
`.section fast`	PicoWeb `#avr_fast` section
`.section slow`	PicoWeb `#avr_slow` section
`.section strings`	`.ascii` and `.ascz` strings in program memory
On-Chip SRAM (512 bytes)	
Assembler Directive	**Meaning**
`.data`	initialized data
`.section .bss`	uninitialized data (zeroed at reset)
`.section COMMON`	named common blocks (alignment penalty!)
`.section uninitialized_data`	uninitialized data (not cleared at reset)
`.dseg`	mapped to "`.data`" by preprocessor
On-Chip EEPROM (512 bytes)	
Assembler Directive	**Meaning**
`.section eeprom*`	initialized data in EEPROM
`.section .eeprom*`	initialized data in EEPROM
External Serial EEPROM (16-32 Kbytes)	
Assembler Directive	**Meaning**
`.section seeprom`	pcode and data in serial EEPROM
`.section seeprom_strings`	strings in serial EEPROM
`.section seeprom_cgi`	pcode and data in serial EEPROM (linked high)
`.section seeprom_cgi_strings`	strings in serial EEPROM (linked high)
`.eseg`	mapped to "`.section seeprom_cgi`" in project file (error elsewhere)

PicoWeb Assembler and Linker
Version 2 of the PicoWeb development system makes use of a number of GNU software tools which have been adapted to target Atmel's AVR machine code. This includes a special version of the GNU assembler (`avr-as`) which products relocatable object files (.o) and assembly listings (.lst), the GNU library (archive) tool (`avr-ar`) which manages object file libraries (.a), and the GNU linker (`avr-ld`) which produces ELF binary files (.elf) and "link maps" (.map). The ELF binary files are created by `avr-ld` from relocatable object modules under control of a linker script (`picoweb.ld`)..

The resulting ELF binary file is then post-processed to produce the data files needed to download firmware into a PicoWeb server. The PicoWeb development system also makes use of the GNU make utility (`avr-make`) to manage the many object modules in the library directory.

Note that the PicoWeb development environment makes use of several complex GNU linker *sections* to control the placement of code and data in the many kinds of memory present in the PicoWeb server's hardware. This includes the SRAM (e.g., `.bss` and `.data` sections), flash-based program memory (e.g., `.text` and `.cseg` sections) and EEPROM (e.g., `.section eeprom` section) in the Atmel microcontroller, as well as the external serial EEPROM chip (e.g., `.cseg` and `.section seeprom` sections). The naming and order of these many linker sections is critical. The addition of new sections and/or modifications to the master linker control script (`picoweb.ld`) in not recommended without a thorough understanding of the issues involved.

The following Table 6 lists the common assembler `.section` directives that are used to control the placement of code and data in the PicoWeb server:

GNU C-Compiler

The version of the GNU C-compiler which targets the AVR processors (i.e., `avr-gcc`) has been found to be compatible with the PicoWeb development system. This compiler produces object modules which can be linked using `avr-ld` and included as part of a PicoWeb project. However, this version of gcc can only produce binaries for *native* AVR machine code. Because there is typically less than 2 Kbytes of free native machine code space available in the PicoWeb server for user-supplied application code, the utility of gcc is limited. When using gcc, in order to conserve the Atmel microcontroller's limited code (and SRAM) space, the use of `unsigned byte` variables in place of `int` is encouraged. Also, the `avr-gcc` code optimization switch `–O4` is recommended when running gcc.

Note that the AVR machine code that `avr-gcc` produces may use (i.e., destroy) *any* of the Atmel processor's 32 8-bit registers. Therefore special steps must be taken when mixing gcc-produced code with AVR assembly language and/or PicoWeb pcode. In particular, if a gcc-produced routine (i.e., C-code) is called from inside a pcode instruction, the pcode instruction pointer (r4 and r5) must be preserved. Also, any AVR assembly routine which is called from a C-code routine must save any registers that it uses (except for r0 and r1). This means that if a routine called by C-code needs to execute PicoWeb pcode, then *all* of the processor's registers (except for r0 and r1) will need to be preserved. This is because pcode routines are free to use any of the processor's registers, except for the pcode instruction pointer (i.e., r4 and r5).

PicoWeb Preprocessor
A Perl-based preprocessor is used to prepare PicoWeb pcode instructions for assembly. In addition this preprocessor will automatically map certain Atmel assembler directives before assembly language source is sent to the GNU assembler. These automatic preprocessor mappings are listed in the following table:

Assembler Directive	Preprocessor Mapping
`.device`	(line is deleted)
`.listmac`	(line is deleted)
`.dseg`	`.data`
`.cseg`	`.text`
`.eseg`	`.section cgi_seeprom` (in a project file only, otherwise an error)
`eseg "string"`	`.section seeprom_cgi_strings`
`.dw`	`.word`
`.db`	`.byte`

`.equ X=y`	`.equ X,y`
	`.equ x,y`
	`.equ X,Y`
	(Note: maps to both upper and lower case)

`.byte` *n**	not changed
`.word` *n**	not changed

*Note: This is guaranteed to be a bug if code was originally developed for pre-GNU assembler! With the old Atmel assembler these directives allocated *n* bytes (or words) of memory. Under the new GNU-assembler they initialize a *single* byte (or word) of storage with the value *n*.

Custom Build Environments
Most PicoWeb projects can be built using the standard development systems tools with a single project file, using the default object modules and source files found in the library directory (`lib`). In those cases where the default files need to be augmented and/or overridden, the user can simply copy a library file into the project directory, modify them as required, and then build their PicoWeb project. Object modules (i.e., .o and .a files) and certain source files (e.g., `cmd.txt` and `pcode.def`) located in the project directory will override those located in the library directory (`lib`).

You can place AVR assembly language code in separate source files and assemble these to create object modules which can be linked with your project. Note that all assembly language file names *must* end in ".asm". These files can be assembled using the following command:

```
gas myasm
```

where *myasm* specifies that the file "`myasm.asm`" should be preprocessed and assembled to produce an object module named "`myasm.o`" (and an assembly language listing file named "`myasm.lst`"). You can cause the file "`myasm.o`" to be linked with your project by adding the following line to your project file:

```
link add myasm.o
```

Note that in order access labels and routine entry points between separately compiled object modules, the assembler directive "`.global`" needs to be used to declare shared labels as globally accessible.

The many firmware source file in the PicoWeb library directory (`lib`) provide examples of how to use separately compiled object modules to assemble a complex PicoWeb project.

Interrupt Vector Table
By default, each entry in the Atmel microcontroller's interrupt vector table, located at location 0 in program memory, points to a dummy routine that consists of a single "`ret`" instruction. This is done using the assembler's "`.weak`" global directive.

The following shows the PicoWeb kernel's interrupt vector table at location 0:

```
.weak VEC_reset
.weak VEC_int0_isr
.weak VEC_int1_isr
.weak VEC_timer1cap_isr
.weak VEC_timer1compa_isr
.weak VEC_timer1compb_isr
.weak VEC_timer1ovf_isr
.weak VEC_timer0ovf_isr
.weak VEC_spistc_isr
.weak VEC_rx_isr
.weak VEC_udre_isr
.weak VEC_tx_isr
.weak VEC_ana_comp_isr
```

Any user-provided interrupt service routines need to override these table entries.

Here is a sample code fragment from the serial port's receive interrupt service routine (see firmware source code in the file "`lib/serial.asm`"):

```
.global VEC_rx_isr
VEC_rx_isr:
    push    r18
    in      r18,sreg
    push r18
    .
    .
    .
```

The assembler directive "`.global VEC_rx_isr`" overrides the default "weak" dummy interrupt service routine for the serial port's receive interrupt with a new user-supplied handler (i.e., label "VEC_rx_isr").

Sample PicoWeb Projects

A number of example 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 a `README.txt` file or an Adobe Acrobat document describing the project in some detail. The Acrobat documents usually 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 shows 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.

PicoWeb Ethernet Addresses

The MAC address (i.e., Ethernet address) of a PicoWeb server is set using the text file "`ether`" as part of the project build process. By default, this 6-byte address is stored in the EEPROM memory of the Atmel microcontroller. Technically, this address should be unique, a different 6-byte value for each Ethernet device in the world. In practice, this address only needs to be unique within a given (local) network of Ethernet devices, up to and including the first router(s) in that network. Beyond the first router, the MAC address does not need to be unique. Therefore, one can pick a random Ethernet address for PicoWeb testing purposes. The precise value does not matter, *as long as the first byte is even*. The odds of sharing the same randomly assigned MAC address with another Ethernet card on your network is extremely small.

Lightner Engineering owns a block of 16 million unique Ethernet addresses, called an Organizationally Unique Identifier (OUI). This block of MAC addresses was assigned by the IEEE. Lightner Engineering's unique MAC addresses begin with 00-30-A2 (in hexadecimal). In PicoWeb project "ether" files this would be a *decimal* address of the form:

0.48.162.*x.y.z*

where *x*, *y*, and *z* are each decimal numbers in the range 0 to 255. Refer to the following IEEE Web page for more information on using IEEE assigned MAC addresses:

http://standards.ieee.org/regauth/oui/tutorials/lanman.html

If you are going to manufacture your own devices with the PicoWeb technology, then you will need to purchase a block of MAC addresses from someone. The minimum investment with the IEEE is $1,250 for 16 million addresses. You can find more information on purchasing your own unique OUI at:

http://standards.ieee.org/regauth/oui/index.html

Lightner Engineering has reserved a block of its IEEE-assigned MAC addresses for unrestricted use by PicoWeb customers:

00-30-A2-00-00-*xx*

where *xx* is a hexadecimal number in the range of 00 to FF. In a PicoWeb project "ether" file, this would be an address of the form:

0.48.162.0.0.*ddd*

where *ddd* is a decimal value in the range of 0 to 255.

Upon request, at no extra charge, PicoWeb customers will be assigned a unique MAC address for each PicoWeb purchased from Lightner Engineering.

PicoWeb IP Addresses

The IP address of a PicoWeb server is normally set using the text file "`ip`" as part of the project build process. Ideally, the PicoWeb server should be assigned an IP address in the same *subnetwork* as the PC that will be used for PicoWeb development and downloading. For example, if the IP address assigned to the PC is 10.1.2.3 and the network mask is 255.255.255.0, then any IP address in the range 10.1.2.1 through 10.1.2.254 is in the same subnetwork as your PC. If you need assistance in determining a proper IP address for your PicoWeb server, please consult your local network administrator.

Using Static Routes

PicoWeb servers are normally shipped loaded with a project and set to an IP address of 10.1.2.3. You *may be* able to use this default IP address by performing the following test using your Windows-based PC. You will

need a PicoWeb server with a project that has been previously built and the firmware downloaded using this default IP address. Then perform the following steps:

1. Plug the PicoWeb server's 10BaseT cable into the same network as the Windows PC. (The PC must have an Ethernet adapter with TCP/IP installed.)

2. Find the IP address of the PC. (You can use the Windows GUI program `WINIPCFG` or issue a "`ROUTE PRINT` " command in an MS-DOS Prompt window. Make sure you have found the IP address assigned to the Ethernet adapter, *not* that of the "dial-up" adapter!)

3. Enter the following command in an MS-DOS Prompt window:

   ```
   ROUTE ADD 10.1.2.3 x.y.z.w
   ```

   where "*x.y.z.w*" is the IP address of the Windows PC.

1. Ping the PicoWeb server with the following command:

   ```
   PING 10.1.2.3
   ```

If you get no errors, you are can use that IP address. Note however that you will have to re-issue the above "`ROUTE ADD`" command again each time the Windows PC is rebooted.

DHCP-Assigned IP Addresses
Assigning an IP address in the same subnetwork as your development PC may be tricky if your PC is being assigned an IP address using DHCP. If you cannot determine a "safe" IP address to assign to your PicoWeb server, consult your network administrator. Ask to be assigned an IP address for your PicoWeb which can be directly addressed by your PC, ideally on the same LAN as your PicoWeb server, with no intervening routers or gateways.

Remote IP Address Configuration

You can change the IP address of a PicoWeb server over the Ethernet using the Windows program "`setip.exe`", providing that the PicoWeb's firmware was build with `#define NET_CONFIG_IP`.

Here is the "usage" information for the program "`setip`":

```
usage: setip [options] MAC IP
where:
   MAC       - Ethernet address assigned to PicoWeb as follows:
                  hexadecimal: xx:xx:xx:xx:xx:xx
                      decimal: ddd.ddd.ddd.ddd.ddd.ddd
   IP        - new IP address for PicoWeb as follows:
                      decimal: ddd.ddd.ddd.ddd
                         name: host
where options:
  -d         - set debug mode
  -b IPaddr  - broadcast to IP address "IPaddr"
  -N num     - number of repeats
```

To use "setip", make sure the PicoWeb is on the same LAN as the host you will be using to run "`setip`" and issue a command like the following in an "MS-DOS Prompt Window":

```
setip 00:30:A2:01:00:01 10.1.2.3
```


Note that "setip" uses standard "BOOTP reply" UDP broadcast packets. Being UDP-based, packets can be lost. So you need to check that the PicoWeb actually "saw" the packets and did indeed change its IP address. "Ping" is a good way to verify that the IP address was changed.

The new IP address set using "`setip`" is no different than the IP address you set and load into the PicoWeb using the project file "ip". The IP address is stored in the PicoWeb's EEPROM where it should remain unaltered, even if the power is cycled.

The firmware that implements this feature can be found in the library source file named "`bootp.asm`".

PicoWeb Development System Caveats

This section lists some of the known problems and/or limitations of the PicoWeb development system.

No Firmware Download Under 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 use under Windows 95 and/or Windows 98. We plan to support PicoWeb firmware downloading in the future through the use of a PicoWeb server which has been specially configured to be able to download a second PicoWeb server. In this case, the firmware to be downloaded into the Atmel microcontroller would be supplied via the Ethernet.

Linux Support
At the current time we do not support development and download under Linux. As a result of our recent switch to the GNU software tools, we expect this situation to change.

Firmware Download Hang-Ups
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.

Random EEPROM Corruption
Certain PicoWeb servers built prior to hardware revision level v4.1 are subject to occasional random corruption of the Atmel processor's on-chip EEPROM during PicoWeb power-cycling. Because the PicoWeb's Ethernet address and IP addresses are usually stored in on-chip EEPROM, the symptom of this EEPROM corruption event is typically an inability to contact the PicoWeb server over the Ethernet. Newer PicoWeb servers include a "reset controller" chip which holds the Atmel microcontroller in reset during power "brown-out" conditions.

Parallel Port Hangs PicoWeb After PC Reboot
If you leave your PicoWeb server plugged into its programming cable, which is in turn plugged into a PC parallel port, you will likely discover that the PicoWeb server will go dead if the PC is rebooted. This is because the outputs lines on most PC parallel ports are driven low after the PC is rebooted. This will hold the PicoWeb in reset if it is still attached to the PC parallel port via its programming cable. You can clear this condition with software by issuing the following command:

```
avr –en
```

This will cause the PicoWeb programmer to send a reset sequence to the PicoWeb programming cable, thereby de-asserting the reset signal to the PicoWeb.

Download Fails with Parallel Ports on Certain PCs

If when attempting to download firmware into a PicoWeb server using `PWAVRLD` (or `PWLOAD`) you get repeated errors of the form:

```
Chip can not be enabled...
```

when the programming cable is attached to the PicoWeb server then you need to check your PC's parallel port. Verify that the parallel port is "enabled", that it is properly configured, and that it is not being used (and/or shared) by another program. On some PC's, the parallel port must be taken out of "bi-directional mode" (sometimes called ECP or EPP mode) before it can be used to program a PicoWeb server. On most PC's this "mode" is a BIOS setting. On other PC's (typically laptops) a Windows GUI application is used to set the parallel port "mode".

Note that certain programs and/or Windows drivers can interfere with the PicoWeb programmer (i.e., `AVR.EXE`). Iomega Zip drives and external CD-ROM drives which use the PC parallel port have been known to cause problems of this kind. One way to quickly "disable" any such installed drivers and test this theory is to reboot your PC in "safe mode". You can select "safe mode" by pressing the F8 function key at the beginning of the Windows 9x boot process. You will see a menu if you do this right. Select "safe mode" to complete the Windows boot-strap sequence.

7/22/00 10:26 PM

