

You can do THAT with SAS Software?

Using the socket access method to unite SAS with the Internet

David L. Ward, DZS Software Solutions, Inc., Bound Brook, NJ

ABSTRACT

The addition of the socket access method to version 6.11 of the SAS System provided the Base, Macro, and SCL languages with a simple way to use the standard TCP/IP network protocol. TCP/IP is the protocol that is used to access virtually all types of network servers, like FTP, Telnet, Web, etc. In the years since version 6.11 was released, the Internet has grown exponentially and become a vital part of virtually all technology-based businesses. Using the Socket Access Method, the SAS system can interact with the servers that make up the Internet, as well as become a server itself.

This paper will present simple implementations of the following TCP/IP protocols and discuss practical uses for each: HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), POP3 (Post Office Protocol), FTP (File Transfer Protocol), Telnet, and SAS to SAS client/server processing. Uses of these protocols will include building a SAS-based web server, dynamic publishing of SAS data on the Internet without SAS/Intrnet, sending email directly from the SAS system without the need to use a third-party's email software, and reading web pages from the Internet without using the URL access method.

NETWORKS AND SOCKETS AND PORTS, OH MY!

A socket is a software object that provides a way for two processes (either on the same computer or different computers) to communicate. There are connection-oriented sockets, which allow data to be transmitted in both directions, or connectionless sockets that allow only one message at a time to be transmitted. It is not too bold to say that the Internet is made up of sockets, among other things. Each computer that functions as a server must establish connections to the clients that are making requests, and by and large, that connection is made through the use of sockets.

Two crucial terms to understand before jumping head-first into socket programming are TCP/IP and port. TCP refers to Transmission Control Protocol, and IP refers to Internet Protocol. TCP/IP is the network protocol of the Internet and many private networks as well and is the predominant protocol used when making socket connections. TCP handles the breaking up of data into packets and re-assembling them at their destination, while IP is responsible for getting the message to its destination. Hence the term "IP Number" or "IP Address" refers to the unique, 4 byte, number of a computer on a network (such as the Internet). The IP number is like a computer's mailing address. Without it, the Internet Protocol does not know how to find a computer to give it the desired message.

A port refers to a number, on Windows-based computers between 0 and 65,535, that serves to provide multiple sockets on the same computer. Think of a socket as your street address and the port as the room within your house. Wouldn't it be nice if the post office did the same and delivered mail to individual rooms! In order to connect to a socket you must know the port that it is bound to. For common servers, like web servers, mail servers, etc., there exist conventions called "well-known ports". Thus, you are fairly safe to assume that a web server will be bound to port 80, a mail server to port 25, etc. See appendix III for more examples and appendix II for the link to a complete list of well-known ports.

THE SOCKET ACCESS METHOD

SAS has seen fit to write an access method that will allow the various SAS languages to read and write to sockets. That access method is appropriately named the "Socket Access Method", added in version 6.11 of the SAS System. Following is the abbreviated syntax of the Socket Access Method as defined in the version 6.12 on-line help. See the help file for more complete documentation.

For client mode:

```
FILENAME fileref SOCKET 'hostname-or-ip-  
address:portno' <tcpip-options>;
```

For server mode:

```
FILENAME fileref SOCKET ':portno' SERVER  
<tcpip-options>;
```

Where <tcpip-options> are any of the following options:

LRECL=lrecl, logical record length (256)
BLOCKSIZE=blocksize, size of socket data
buffer (8192)

RECFM=recfm, record format (V)

-F is fixed record format

-S is stream record format

-V is variable record format

RECONN=conn-limit, maximum number of
connections server will accept (1)

TERMSTR='eol-char', end-of-line character
when RECFM=V (LF)

-CRLF is a carriage return (CR) followed
by line feed (LF)

-LF is a line feed only

-NULL is the NULL character (0x00)

Once a filename has been defined using the Socket Access Method, the file can be treated, in most respects, like a local file. You can use the data step and fopen family of SCL functions to read and write to your socket.

THE WONDERFUL WORLD OF HTTP

The hypertext transfer protocol (HTTP), not to be confused with hypertext markup language (HTML), has emerged as the definitive protocol of what is now known as the world wide web. Each time you use a web browser to view a web page, your browser sends a request to the appropriate web server using HTTP, that is, the message that it sends is formatted according to the HTTP specifications. Let's take a look at a typical HTTP Request and the corresponding HTTP Response. Each contains HTTP header fields followed by the message itself.

To view an HTTP request you can use the SAS system to mimic a web server and use your web browser to connect to it. Run the following SAS code to mimic a web server:

```
* CREATE A SOCKET SERVER CONNECTION;  
filename web socket ':80' server  
termstr=CRLF;  
data _null_;  
  infile web;  
  ** READ AND PRINT DATA RECEIVED **;  
  input;  
  put _infile_;  
run;
```

Next, open your favorite web browser (Microsoft Internet Explorer in my case) and navigate to <http://my-ip-address>. Look at your SAS log and you should see something like this:

```
GET / HTTP/1.0
```

```
Accept: image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, application/vnd.ms-
excel, application/msword,
application/vnd.ms-powerpoint, */*
Accept-Language: en-us
Host: my-ip-address
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE
5.01; Windows NT)
Via: proxy (Proxy/5.0) for my-ip-address
```

You will need to break the data step manually due what is either a flaw in the Socket Access Method, or a limitation in my understanding of it. Once the web browser sends the HTTP request, it should send a message termination character (not a line termination character) so that the server will know that it is finished and is expecting results. SAS seems not to recognize the web browser's termination character thus continues to wait for more data.

The text that you see above is what your browser is sending out to every web server it gets web pages from. I'll bet you didn't know how much information is being sent! The only required fields to get a response from a web server are:

```
GET / HTTP/1.0
Host: my-ip-address
```

The GET directive tells the server what page you would like and what protocol is being used to send the request. The Host directive tells the server where to send the page back to. Leaving off the host parameter is like sending a letter with no return address to someone you do not know and expecting a letter back.

BUILDING YOUR OWN BROWSER

The SAS system does have another file access method to read information from a web page, called the URL Access Method. This method simplifies the task of programmatically retrieving information from web servers. See your version of SAS OnlineDoc® for more information. However, the URL Access method does have some limitations:

- 1) Only the GET method of retrieving a page is supported. GET means that all of the data you will be submitting to a web server is contained in the URL. Ever noticed all those funny percents and ampersands at the end of some web pages? Those characters actually make up parameters you have submitted, like when performing a search. The POST method allows the client to submit much more data to the server, and in a diversity of formats (like uploading files). If you need to get a web page that requires parameters using the POST method, you can not use the URL access method.
- 2) The URL access method does not let you read or write any HTTP headers. This is on purpose, to simplify the task of getting a web page. If you don't have to write or even think about HTTP headers, why do it? But there are times when you may want access to the headers, as they reveal the operating system, web server software, cookies, etc.

Downloading a simple page from a web server:

```
filename browser socket 'www.sas.com:80'
termstr=LF;
data _null_;
  infile browser;
  file browser;
  if _n_=1 then put
    'GET / HTTP/1.0' /
    'Host: my-ip-address' /;
  input;
  file log;
  put _infile_;
run;
```

And the results, printed to the SAS log window:

```
HTTP/1.1 200 OK
Date: Fri, 12 May 2000 19:01:46 GMT
Server: Apache/1.3.12 (Unix) PHP/4.0b4pl1
Last-Modified: Fri, 12 May 2000 18:29:46 GMT
```

```
ETag: "109c-413e-391c4d9a"
Accept-Ranges: bytes
Content-Length: 16702
Connection: close
Content-Type: text/html
```

```
<html>
<head>
<title>
  SAS ... The power to know
</title>
... rest of document follows
```

This example gives you a taste of what is possible using the socket access method to read data from web servers. You can form your own HTTP requests, according to the HTTP specifications (find the link to the document in appendix II), that POST data to web pages and read search results or the output of other queries. You can read cookies sent by the web server and send them back in order to do things like log in to web sites. Anything that can be done with a web browser could, theoretically, be done with SAS using the socket access method. In the next section we will extend the socket-HTTP relationship one step further...

A SAS POWERED WEB SERVER

A web server is nothing more than a software application running on a computer that is connected to the Internet (or a local Intranet). The software waits for clients to ask for pages, then serves them up. Web servers can also typically execute CGI scripts by sending a file through a CGI interpreter first. Using the Socket Access Method, it is possible to build a simple, limited web server using pure SAS code. It may not be terribly useful in the real world, but is certainly an interesting application of using sockets with SAS.

The idea is fairly simple: open a socket server connection and wait for a web browser to connect and ask for a page. Read the desired page from the browser, and return it, making sure to send an HTTP header before sending the actual data. Let's use SCL as the development language because of the ability to use submit blocks to run SAS code while the socket is open. This is not possible if a data step is used because once the data step is closed the socket connection is automatically closed.

The example code is the simplest possible web server. It only receives a file with the GET parameter, can't process any parameters, and can't handle binary files for use with images. It also does not create log files, or allow all *.SAS files to be submitted and the results returned, instead of just the code, a natural and obvious application for such a server. Such enhancements are left to the reader. Also note a rather severe limitation of this server – the Socket Access Method can only support one connection at a time. If multiple users requested pages from the server they would be forced to wait.

```
length file basepath line $200;
init:
  ** SET THE WWW ROOT DIRECTORY **;
  basepath='d:\dward\sasweb';
  ** CREATE THE SOCKET FILENAME **;
  rc=filename('s','80','socket',
    'server termstr=CRLF');
  ** LOOP INDEFINITELY **;
do while (1);
  fid=fopen('s','w',200,'v');
  rc=fread(fid);
  rc=fget(fid,file,200);
  file=scan(file,2,' ');
  put 'REQUESTED FILE=' file;
  rc=fput(fid,'HTTP/1.1 OK'||
    '0A'x||'Content-Type: text/html'||
    '0A'x);
  rc=fwrite(fid);
  ** DETERMINE DEFAULT FILE **;
```

```

if file='' then file='/default.htm';
if fileexist(trim(basepath)||
  translate(file,'\\','/')) then do;
  rc=filename('in',trim(basepath)||
    translate(file,'\\','/'));
  fidin=fopen('in','I');
  do while(fread(fidin)=0);
    rc=fget(fidin,line,200);
    rc=fput(fid,line);
    rc=fwrite(fid);
  end;
  rc=fclose(fidin);
  rc=filename('in','');
end;
else do;
  rc=fput(fid,'File '||trim(file)||
    ' not found.');
```

```

  rc=fwrite(fid);
end;
rc=fclose(fid);
end;
rc=filename('s','');
return;
```

DYNAMIC WEB PUBLISHING WITHOUT SAS/INRNET

It is possible to use the Socket Access Method to create a SAS session that will dynamically publish SAS data to users navigating to a web page. In order to understand how to accomplish this the concept of a CGI program must first be explained. The Common Gateway Interface (CGI) is a standard method of a web server invoking a program to process or execute a file instead of just returning output from a file. Typically, a directory such as cgi-bin/ exists that has execute only permissions (thus you cannot download the files). CGI programs can be written in any language and receive their input via environment variables. Their task is to process the parameters, usually look up information in some kind of a database, then return text-based output for the client web browser.

In our case, we would like to create a simple CGI script that connects to a SAS session via TCP/IP sockets, passes it the parameters via SAS/Macro variables, and waits for a response back. Of course, an alternative method of accessing the SAS system would be to invoke SAS in batch and return the output generated, but for the sake of example, let's use sockets.

A natural choice for developing a CGI script is to use Perl, the Practical Extraction and Reporting Language. You can find out more information about Perl at <http://www.perl.org>. Included below is a sample CGI script that simply passes parameters to SAS and waits for output back. An explanation of the Perl code will not be included not only to save space but because it is outside of the scope of this paper. Also, line breaks have been flowed over due to fitting the code in one column. You may need to do some reformatting to get it to compile properly.

```

#!d:\perl\bin\perl.exe
use Socket;
# Open a TCP/IP Socket to the SAS Session
$addr= inet_aton('191.168.1.189'); #IP
Address of SAS machine
$paddr= sockaddr_in('4000',$addr); #Port
socket(SOCK,PF_INET,SOCK_STREAM,6) || die
'die on socket';
connect(SOCK,$paddr) || die 'die on connect';

# Send either post or get buffer to SAS as
%let statements
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
if (length($buffer) < 5) {$buffer =
$ENV{'QUERY_STRING'}};
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
  ($name, $value) = split(/=/, $pair);
  $value =~ tr/+/ /;
```

```

  $value =~ s/%([a-fA-F0-9][a-fA-F0-
9])/pack("C", hex($1))/eg;
  # Mask special SAS characters
  $value =~ s/%/%/g;
  $value =~ s/\(/%\( /g;
  $value =~ s/\)/%)/g;
  $value =~ s/'/%'/g;
  $value =~ s/"/%"/g;
  $value =~ s/\n/ /g; # CR/LF replacement
  with a space
  print SOCK "%let $name=%nrstr($value);\n";
}
print SOCK "%let webip=$ENV{REMOTE_ADDR);\n";
print SOCK "%let
browser=%nrstr($ENV{HTTP_USER_AGENT});
close (SOCK);

# RECEIVE DATA BACK FROM SAS
$t = pack('S n C4 x8',2,'4000',0,0,0,0);
#Listen on port 4000
socket(S,2,1,6) || die 'die on socket';
bind(S,$t) || die 'die on bind';
listen(S,5) || die 'die on listen';
accept(NS,S) || die 'die on accept';
print <NS>; #Print results to browser
close (NS);
```

The corresponding SAS code could be something like:

```

** SET UP FILENAMES **;
filename _webout 'c:\temp\output.htm';
filename recv socket ':4000' server;
filename send socket '192.168.1.240:4000';
%macro loop;
  /** LOOP INDEFINITELY **/
  %do %while(1);
    data _null_; ** RECEIVE MACRO VARS **;
    infile recv;
    file 'c:\temp\parms.sas';
    input; put _infile_;
    run;
    ** SET MACRO VARS **;
    %include 'c:\temp\parms.sas';
    ** INCLUDE NAMED PROGRAM **;
    %if %length(&_program)>0 %then %include
"&_program";
    %else %do;
      data _null_;
      file _webout;
      put 'Content-Type: text/html' //
'File not found';
      run;
    %end;
    data _null_; ** SEND RESULTS BACK **;
    infile _webout;
    file send;
    input;
    put _infile_;
    run;
  %end;
%mend;
%loop;
```

All that is required for your web browsing users to run a SAS program is to make sure that the _program variable is set when calling the CGI script. The corresponding SAS program that will be executed should write output to the filename _webout and can make use of macro variables set to the input parameters coming from the web. Obviously, the above code is a bare minimum, and in real practice you would need to add error checking and some kind of security so that arbitrary programs cannot be executed. Consider this a foretaste of what is possible using sockets and SAS with a

web server.

E-MAIL, E-MAIL, AND MORE E-MAIL

Electronic mail has become one of the data delivery backbones for business and personal communications. There are many, especially in the corporate environment, who have e-mail access and no internet access, thus it is important to explore ways in which the SAS system can send and process e-mail messages. Assume that your job is to generate 1500 reports, one for each sales representative in a company's sales force, and email them to each person individually. You are supplied with the email addresses and names of all users, but no additional hands to click on the "send mail" button in your favorite e-mail program 1500 times. Clearly, you need an automated method.

SENDING E-MAIL

The SAS System was thoughtful enough to include the Email Access Method as a filename statement option that allows you to send email using the data step or any other process that can write to a filename. See the OnlineDoc® help files for information regarding this method. This may be useful for some quick and dirty e-mail jobs, but you may run into problems like:

- 1) The email access method can be difficult to configure right, particularly if using non-standard email clients or servers.
- 2) Email is sent according to your default MAPI client's configuration. Testing with Microsoft Outlook on Windows 98, I experienced the unpleasant feature of e-mail messages not actually being sent immediately, but instead being placed into the "Outbox". The messages were successfully sent after manually triggering the send command, and the scores of messages were placed in the "Sent Items" folder.

The Email Access Method makes use of your default mail *client*. You have much more flexibility if you can communicate directly with the *servers* that these clients eventually send their messages to. The predominant e-mail server protocol is called SMTP, Simple Mail Transfer Protocol. This protocol defines commands that a mail server will accept when processing incoming mail messages. A full listing of commands can be found by following the link in Appendix II.

You can obtain a free, fully functional SMTP server for windows based machines that is very helpful in understanding what SMTP servers do at <http://www.argosoft.com>. The example code uses Argosoft's Mail Server, version 1.12 running on Windows NT 4.0. The first and simplest example of sending email is to send a one-line message. Multiple short statements have been combined into one line for readability.

```
filename smtp socket '192.168.1.189:25'
termstr=CRLF;
data _null_;
  length word $40;
  infile smtp truncover;
  file smtp; input line $200.;
  word=scan(line,2);
  select (word);
    when ('ArGoSoft') put 'HELO';
    when ('Welcome') put 'MAIL FROM: <dward>';
    when ('Sender') put 'RCPT TO: <dward>';
    when ('Recipient') put 'DATA';
    when ('Enter') put
      'From: "David Ward" <dward@sashelp.com>' /
      'Subject: My Subject' /
      'To: "David Ward" <dward@sashelp.com>' //
      'E-mail Body' /
      '.';
    when ('Message') put 'QUIT';
    otherwise stop;
  end;
  file log; put _infile_;
run;
```

The above code does the following:

- 1) Establish the socket filename by giving the IP address of the SMTP server and specifying the port of the SMTP server (25)
- 2) Read lines of data from the server, looking at the first word to know when to send certain SMTP commands
- 3) Send the MAIL FROM and RCPT TO commands so that the server will agree to send our message
- 4) Send the body of the message, along with mail headers (like From, To, and Subject) that are parsed by the user's email program
- 5) Terminate the message by issuing the QUIT command and stopping the data step.

Inspecting the server messages in the SAS log window shows that the process worked correctly. You may wish to write these responses to a file so that you have documented proof of delivered messages. Here is the response from my SMTP server:

```
220 ArGoSoft Mail Server Plus, Version 1.2
(1.2.1.1) [UNREGISTERED]
250 Welcome [192.168.1.189], pleased to meet
you
250 Sender "dward" OK...
250 Recipient "dward" OK...
354 Enter mail, end with "." on a line by
itself
250 Message accepted for delivery.
<11rdc5j8djmig0v.180520001157@workstation>
```

A NOTE ON MIME

MIME (Multi-purpose Internet Mail Extensions) is the protocol that defines e-mail messages. Instructions on mail headers and how to send attachments can be found in the MIME specifications, the link to which is found in appendix II. If you want to send binary attachments, they will need to be encoded into plain text since e-mail is and has always been text based. The most common encodings are base64 and octet-stream, the details of which are outside of the scope of this paper. The best way to learn how to construct e-mail with attachments is to look at messages generated by typical e-mail client programs and inspect their original source code. With a little experimenting, you can send messages with multiple binary attachments, or send messages in HTML and include such things as embedded images.

RECEIVING E-MAIL WITH THE SAS SYSTEM

The standard format for receiving electronic mail is POP3, Post Office Protocol, version 3. By now, most of us have become familiar with this term because our Internet service providers give us one or more "POP3 mailboxes". POP3 defines the commands used to log on to a mail server, select messages to retrieve, and optionally delete messages. Anything you can do in an e-mail program's mail retrieval options settings you can do directly via POP3, in fact, your e-mail program is converting the point and click options into POP3 instructions.

Interacting with a POP3 server is very similar to interacting with an SMTP server. Once again, there are very simple commands used to get a list of messages and to retrieve individual ones. The example code below logs into the mail server and retrieves message 1, printing it to the log. Obviously, you will want to modify the code to write the e-mail message to a file, parse header fields, etc. More advanced functionality is left up to you. Multiple short statements have been combined into one line for readability.

```
filename pop3 socket '192.168.1.189:110'
termstr=crlf;
data _null_;
  length line $200;
  infile pop3 truncover;
  link input;
  put 'USER Dward'; link input;
  put 'PASS password'; link input;
  put 'RETR 1';
  do while(line^='.'); link input; end;
```

```

    stop;
    return;
input:
    input line $char200.;
    file log; put _infile_; file pop3;
run;

```

And the result, written to the SAS log:

```

+OK ArGoSoft Mail Server, Version 1.13
(1.1.3.2)
+OK Password required for Dward
+OK Mailbox locked and ready
+OK 548 octets
Received: from [192.168.1.189] by 58zgu
(ArGoSoft Mail Server, Version 1.13
(1.1.3.2)); Fri, 30 Jun 2000 10:15:36 -0400
Message-ID:
<000701bfe29d$a8c2ba20$bd01a8c0@DZSNT>
From: "David Ward" <dward>
To: "David Ward" <Dward>
Subject: test
Date: Fri, 30 Jun 2000 10:15:35 -0400
MIME-Version: 1.0
Content-Type: text/plain;
    charset="iso-8859-1"
Content-Transfer-Encoding: 7bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express
5.00.2314.1300
X-MimeOLE: Produced By Microsoft MimeOLE
V5.00.2314.1300

```

```
test
```

```
.
```

Using this same technique, you can send POP3 commands to your mail server to retrieve a list of messages, download individual messages (as in our example), delete messages, mark them as read/unread, and much more.

FILES, FILES – EVERYWHERE!

FTP has been around in some form or another since 1971. The simple File Transfer Protocol has proven to be the most reliable and most widely used method of transferring files on the Internet to this day. Given its usefulness and popularity, SAS has seen fit to include yet another file access method for using FTP, the "FTP Access Method". The FTP access method is actually quite extensive and includes commands for retrieving directory listings, reading and writing files, even using %include'ing a file via FTP! Using the socket access method to directly interact with FTP servers is not necessarily *better* than the FTP access method, but you gain much more control over what is actually happening on the server and can customize your FTP sessions way beyond what SAS has provided in their method.

FTP works a little differently than HTTP in that the client and server actually share TWO connections when making transactions, a "control connection" and a "data connection". The control connection is in charge of sending and receiving commands and command status, and the data connection receives and sends files or data. So to use FTP we must actually run TWO SAS sessions, one to communicate with the control, one with the data connection. The data connection is the simplest, so let's build a simple data step that will wait for a file to be sent to it via FTP:

```

filename recv socket ':20' server
termstr=crlf;
data _null_;
    infile recv;
    input; put _infile_;
run;

```

The control connection is a little more complex, but is very similar to our previous examples of interacting with a mail server. Again, multiple short statements have been combined into one line for readability, as well as defining reusable labels to print results:

```

data _null_;
    length cmd $200;
    infile ftp;
    ** SEND USERNAME AND PASSWORD;
    cmd='user dward'; link send; link get;
    ** SET TYPE TO ASCII **;
    cmd='type a'; link send; link get;
    cmd='pass password'; link send; link get;
    ** SET IP/PORT TO RECEIVE DATA ON;
    cmd='port 192,168,1,189,0,20'; link send;
    link get;
    ** SEND THE LIST COMMAND TO SEE FILE LIST;
    cmd='list'; link send; link get;
    ** OPTIONALLY SEND RETRIEVE COMMAND TO
    GET A FILE;
/*cmd='retr boot.ini'; link send; link get;*/
    ** GET RESULTS OF COMMAND;
    link get; link get; stop;
return;
send:
    file ftp; put cmd;
return;
get:
    input; file log; put _infile_;
return;
run;

```

After running first the data connection (to wait for the data being sent), then the control connection, the SAS log shows this:

```

220 ArGoSoft FTP Server, Version 1.2
(1.2.0.6) [UNREGISTERED]
331 User name OK, need password
200 Type set to ASCII
230 User dward logged in successfully **
200 Port command successful
150 Opening ASCII data connection
226 Transfer complete

```

And the log for the data connection shows the listing of files requested. Switching the retr command with the list command will print the contents of the requested file to the SAS log.

WRITE AN FTP CLIENT?

Though the example shown is simple, it outlines the basic procedure for writing your own SAS-based FTP client to interact with FTP servers. You can customize the code to do very complex tasks via FTP, like sending and receiving many files or just requesting information from an FTP server. If you have FTP needs that go beyond what the FTP access method has to offer, this method is one possible solution.

TO TELNET OR NOT TO TELNET?

Telnet is the protocol for remotely logging in to a computer, typically a Unix-like machine, and creating a terminal via TCP/IP. It is an integral part of the access and maintenance of many servers, thus developing a way to programmatically telnet to a machine, execute commands, and log off is invaluable to any network professional who uses SAS. Telnet works differently than the other protocols we have looked at so far and is much more difficult to use directly. Telnet consists of single byte commands given back and forth between the client and server where all data is transmitted in binary mode. This proves to be a rather cryptic and difficult process for the layman (like the author) to understand. The following example shows how far the author got, simply getting a message back from a Linux server stating that the server exists and will listen to my requests. The Telnet RFC (see appendix II) command abbreviations have been used to make the code at least a little more readable.

```
filename telnet socket '192.168.1.250:23'
termstr=crlf;
data _null_;
  iac=byte(255); ayt=byte(246);
  infile telnet;
  file telnet recfm=N;
  put (iac ayt) ($1.); input;
  file log; put _infile_;
  stop;
run;
```

FLYING SOLO: SAS TO SAS

Let's not forget that SAS can act as both a socket client and a socket server, therefore two SAS sessions, possibly on separate machines, can communicate with one another. You could set up a SAS session that waits for data and/or commands from another and bypass the need for another protocol (like HTTP) to get in the way. To see this in action, let's create a simple socket server using the data step. It will wait for SAS code to be sent to it, execute it, and then send the log back to the client.

```
filename server socket ':4000' server;
filename client socket 'my-ip-address:4000';
data _null_;
  infile server;
  file 'c:\temp\prog.sas';
  input; put _infile_;
run;
proc printto log='c:\temp\temp.log' new; run;
%inc 'c:\temp\prog.sas';
proc printto log=log;
data _null_;
  infile 'c:\temp\temp.log';
  file client;
  input; put _infile_;
run;
```

The corresponding client:

```
filename server socket ':4000' server;
filename client socket 'my-ip-address:4000';
data _null_;
  file client;
  put 'data _null_; run;';
run;
data _null_;
  infile server;
  input; put 'SERVER:' _infile_;
run;
```

After running the server first, then the client, here are the results, written to the client's log:

```
SERVER:NOTE: The PROCEDURE PRINTTO used 0.0
seconds.
SERVER:
SERVER:
SERVER:248 %inc 'c:\temp\prog.sas';
SERVER:
SERVER:NOTE: The DATA statement used 0.04
seconds.
SERVER:
SERVER:
SERVER:250 proc printto log=log;
SERVER:
```

CONCLUSION

The ability to use TCP/IP sockets in SAS programming opens up a world of possibilities for accomplishing Internet-related tasks. We have learned how to retrieve and serve web pages, send and receive email, use FTP, Telnet, and even communicate between two SAS sessions. May these ideas stimulate your SAS creativity and lead you to further develop some of these rather academic and basic examples!

APPENDIX I: GLOSSARY

ACRONYMS:

FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IMAP	Internet Message Access Protocol
IP	Internet Protocol
MIME	Multipurpose Internet Mail Extensions
NNTP	Network News Transfer Protocol
POP3	Post Office Protocol - Version 3
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol

TERMS:

Socket	A software object that connects an application to a network protocol.
Port	An endpoint to a logical connection. Windows supports up to 65,535 ports. Other operating systems support a varying number of ports.
TCP/IP	A set of protocols developed to allow cooperating computers to share resources across a network.

APPENDIX II: INTERNET RESOURCES

HTTP 1.1: <http://www.cis.ohio-state.edu/htbin/rfc/rfc2616.html>

FTP: <http://www.cis.ohio-state.edu/htbin/rfc/rfc959.html>

MIME: <http://www.cis.ohio-state.edu/htbin/rfc/rfc1341.html>

NNTP: <http://www.cis.ohio-state.edu/htbin/rfc/rfc977.html>

POP3: <http://www.cis.ohio-state.edu/htbin/rfc/rfc1939.html>

IMAP: <http://www.cis.ohio-state.edu/htbin/rfc/rfc2060.html>

SMTP: <http://www.cis.ohio-state.edu/htbin/rfc/rfc821.html>

TELNET: <http://www.cis.ohio-state.edu/htbin/rfc/rfc854.html>

Well-Known Port Numbers: <http://www.cis.ohio-state.edu/htbin/rfc/rfc1010.html>, <http://www.isi.edu/in-notes/iana/assignments/port-numbers>

APPENDIX III: WELL-KNOWN PORT NUMBERS

HTTP	80	IMAP	143
FTP	21	SMTP	25
NNTP	119	TELNET	23
POP3	110		

REFERENCES

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David Ward

DZS Software Solutions, Inc.

1661 Rt. 22 West

Bound Brook, NJ 08805

P: (732) 356-6961

F: (732) 764-6755

dward@dzs.com, dward@sashelp.com

<http://www.dzs.com>, <http://www.sashelp.com>