Paper 83

Using Sockets in SAS® Software for Internet Publishing

Chapman Gleason, US Environmental Protection Agency, Washington, DC
Hsiwei Yu (Michael), Digital Intelligence Systems, Centreville, VA

ABSTRACT

The SAS System's® Base product supports TCP/IP socket communication access method as a filename parameter since Release 6.11. Two processes running on different hosts, say PC and Unix, can set up ports to communicate by TCP/IP sockets. With the aid of the SAS Web Publishing Tools®, a Perl® program can be implemented for dynamic display of SAS data on the Internet.

INTRODUCTION

TCP/IP socket communication access method is an integral part of networking environment today. It has been implemented on many operating systems, such as on MVS®, Unix, and PC. Software languages and packages from many vendors, such as SAS, C/C++, Visual Basic®, Java®, and Perl, support it. This paper aims to demonstrate how to use SAS sockets in client/server applications.

OVERVIEW OF SOCKET, AS IMPLEMENTED IN SAS

A socket is like a telephone for two persons to talk and listen. Two processes can communicate by setting up a socket with a common port number. However if both processes simultaneously wanting to send i.e. talk, or to receive i.e. listen, deadlock can result. Base SAS supports TCP/IP socket communication access method as a filename parameter. As implemented in the SAS System, a socket can be used only to send or to receive, i.e. one direction only. The syntax clearly indicates for sending or for receiving:

```
/* A socket for receiving: */
Filename receiver SOCKET ':portno' SERVER <tcpip-
options>;

/* A socket for sending: */
Filename sender SOCKET 'host:portno' <tcpip-
options>;

Data _null_;
Infile receiver; /* Read, receive, as normal */
  File sender; * Write, send, like a normal file;
…
run;
```

A SAS process would need two socket ports, one for sending and one for receiving, for it to communicate with other process.

SEQUENCE OF EVENTS BETWEEN SERVER AND CLIENT

Model of events for a single request:

SAS server	Client (SAS or otherwise)
Opens a receiving socket on port number, say, 001	
	Opens a sending socket on port 001
Receives request on port 001	Sends request on port 001
Closes port 001	Closes port 001
Parses request before execution	Opens a receiving socket on port number, say, 002
Opens a sending socket on port number, say, 002	
Sends back the result on port 002	Receives result on port 002
Closes port 002	Closes port 002

The table above gives the event timing sequence between a client and its SAS server for a single request. Note a SAS server first opens a receiving socket (for request) then opens a sending socket (for result). The client does this in reverse, first opening a sending socket (for request) then a receiving socket (for result).

A SAS server, ready to receive on port number 001, could wait a long time before a request comes to it. However if without a server ready to receive, a client can never send any request. Also, the server and client must agree on, in advance, how to pick the new port number for transmitting the resulting file. For simplicity, this author adopts the convention that the new port number is the next sequential number. However, these two port numbers can be the same number.

SAMPLES TO HANDLE A SINGLE REQUEST

Below are SAS server program and client program in SAS and Perl capable of handling a single request. These can be the building blocks for constructing a client/server application, following common gateway interface (CGI) convention, for invoking SAS services on remote host to dynamically satisfy user requests from the Internet.

SAMPLE SAS SERVER

Here's a SAS server in macro language, looping forever to respond to requests. It's worth noting while waiting for the next request, the server is in a sleep state, consuming very little resource on its host machine.

```
/* Port as input parameter */
%let portno= &sysparm;
%* let portno= 001;

%do %while ( 1 );
filename in_msg SOCKET " :&portno"
SERVER;

filename temp_pgm
" %sysfunc(pathname(work))/a-ready.sas" ;

data _null_;
infile in_msg;
  file temp_pgm;
 input ;
   put _infile_;
run;

filename in_msg;

/* Before execution, server can choose to
parse it first */
%inc temp_pgm;
run;
filename temp_pgm;
%end;
```

We can start several SAS batch jobs each with its own socket port number, ready to respond to client requests.

CLIENT WRITTEN IN SAS

This client is supplied with a SAS server's IP address, port number to call on, and the program file for the server to execute. It is able to send one request to a known available SAS server. Also note the client is responsible for directing the server on which port to send back the results.

```
%let host= IP-address-for-server-machine;
```

```
%let portno= 001;

filename to_servr SOCKET " &host:&portno" ;
filename input 'SAS-program-to-be-run-on-
server.sas';

/* Simple scheme to come up with a different port
number to accept response from the server */
%let ret_port= %eval( &portno + 1 );

data _null_;
  file to_servr;
infile input end= EOF;

if _n_ = 1 then do;
/* Required info for the server to return results
back to this client */
   put '%let ret_port= ' " &ret_port;" ;

   put '%let client= IP-address-for-client-
machine;';
end;

input ;
  put _infile_;
run;

filename input;
filename to_servr;

/* Receiving socket for the results */
filename back_rst SOCKET " :&ret_port"  SERVER;

data null_;
infile back_rst;
  file print;

 input ;
   put _infile_;
run;

filename back_rst;
```

REQUEST FOR TEXTUAL REPORT

This file is stored on the client's machine, and is sent by the client for execution on the server's machine.

```
/* &CLIENT and &RET_PORT are set by the client
program */
Filename _webout SOCKET " &client:&ret_port" ;

Proc printto print= _webout;
Run;

Proc print …;
Run;

Proc printto;
Run;

Filename _webout;
```

REQUEST FOR GRAPHICAL REPORT

```
/* &CLIENT and &RET_PORT are set by the client
program */
Filename _webout SOCKET " &client:&ret_port" ;

Goptions device= gif gsfname= _webout;
Run;

Proc gchart …;
Run;

Filename _webout;
```

CLIENT WRITTEN IN PERL

Some of the Perl functions used: gethostbyname, getprotobyname, socket, connect, bind, listen, and accept. This client is supplied, in the first five parameters, both the client and SAS server's IP addresses, sending and receiving port numbers, and name of the program file for the server to execute. It is able to send one request to a known available SAS server. Note the client is responsible for directing the server on which port to send back the results. Error checking code is not shown for simplicity.

```
$Web_Server= IP-address-for-client-
machine;
$SAS_host= IP-address-for-server-machine;
$to_SAS_port= 001;
$from_SAS_port= 002;
$pgm_2_run= SAS-program-to-be-run-on-
server.sas;


# Part ONE: Ready to send to SAS

($d1,$d2,$prototype)
=getprotobyname("tcp");

($d1,$d2,$d3,$d4,$addr_Web_Server)
=gethostbyname( $Web_Server );
$packd_Web_Server=
pack("Sna4x8",PF_INET,0,$addr_Web_Server);

($d1,$d2,$d3,$d4,$addr_SAS_host)
=gethostbyname( $SAS_host );
$packd_SAS_host=
pack("Sna4x8",PF_INET,$to_SAS_port
,$addr_SAS_host);

$d1= socket(TO_SAS,PF_INET,SOCK_STREAM,
$prototype );
$d1= bind(TO_SAS,$packd_Web_Server);
$d1= connect(TO_SAS,$packd_SAS_host);


# Part TWO: Ready to receive results back
# from SAS

$packd_Web_Server=
pack("Sna4x8",PF_INET,$from_SAS_port
,$addr_Web_Server);

$d1= socket(FROM_SAS,PF_INET,SOCK_STREAM
,$prototype);
$d1= bind(FROM_SAS,$packd_Web_Server);
$d1= listen(FROM_SAS,1);


# Part THREE: Send request to the server

open(SEND_2_SAS,$pgm_2_run);

# Direct the server where to send back
# result!
print TO_SAS
"%let ret_port= $from_SAS_port;";
print TO_SAS "%let client= $Web_Server;";
while ( <SEND_2_SAS> ) {
   print TO_SAS $_;
}

close( SEND_2_SAS );
close( TO_SAS );


# Part FOUR: Receive from the server
# and show to the Internet

$d1= accept( RESULT_FI, FROM_SAS );
while ( <RESULT_FI> ) {
```

```
   print STDOUT $_;
}

close( RESULT_FI );
close( FROM_SAS );
```

OUTLINES FOR A CONTINUOUS SAS SERVER

By adding programming logic to the server and client code, the above single-request-event-model can be expanded into continuous responding server handling multiple requests,

1) Use a macro loop for the server program.
2) Make intelligent server program by parsing the incoming request prior to execution.
3) Start several SAS servers, so that when one server is busy responding, other servers are still waiting for new requests.
4) Set up a configuration file storing such information like, server's host IP address, number of available SAS servers, and port numbers associated with servers on that host. Client program, upon starting, reads this file, determines the first available SAS server then sends request to server on that port number.
5) Enhance the client program, following the common gateway interface (CGI) convention, to accept HTML form's input parameters and construct the corresponding name and value pairs as input to the program for execution on the server machine.

The enhanced client written in Perl, together with installation instructions, are available from three sources, on the Tools CD-ROM, from contacting authors' email address, or download from Washington D.C. SAS Users Group homepage, http://www.ita.doc.gov/industry/otea/dcsug/.

SOME CAUTIONS

1) Network system administrator can block port numbers used by TCP/IP sockets.
2) Consider using this utility as an education tool to familiarize with the SAS Web Publishing Tools and HTML technology in general. Implement it in a well-controlled environment, say, your PC at home.
3) This utility implements a very, very small subset of functions provided by the SAS/IntrNet® product. Applications based on this tool probably will never be adopted for production in your organization. However, we have designed it in such a way that your application programs can easily be adapted to run with the SAS/IntrNet product.

CONCLUSION

We have successfully implemented intelligent client written in Perl, and used it internally for dynamic display of SAS data on Intranet.

Since socket communication access method is part of the Base product, interested SAS developer can easily create client/server SAS applications, promoting the power and value of the SAS System.

REFERENCES

- "Use Socket in SAS" by Hsiwei Yu, 3rd quarter 1998, Washington DC SAS Users Group Newsletter.
- 60 Minute Guide to CGI Programming with Perl 5 by Robert Farrell (IDG Books Worldwide 1996).
- Teach Yourself Perl 5 for Windows NT in 21 Days by Tony Zhang and David Till (Sams.net Publishing 1997).
- "SAS Software and the Web: Creating a Common Gateway Interface" by David Shinn and John Hansen, Observation, SAS Institute Inc., First quarter 1997, p. 62-66.

ACKNOWLEDGMENTS

This author likes to thank Chapman Gleason of US Environmental Protection Agency and Myles Powers of Logicon for formulating this requirement, and for their support and encouragement for its solution.

CONTACT INFORMATION

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

Chapman Gleason
US Environmental Protection Agency
Washington, District of Columbia
Work Phone: 202-260-9006
Fax: 202-260-4968
Email: gleason.chapman@epa.gov

Hsiwei Yu (Michael)
Digital Intelligence Systems
Centreville, Virginia
http://www.disys.com
Work Phone: 202-260-5312
Fax: 202-260-4968
Email: vhyu@netkonnect.net