Paper 19-25

Client/Server Setup and Implementation: Web and non-Web Environments

John Laing, SAS Institute (Canada) Inc., Toronto, ON

ABSTRACT

Client/server setup is becoming more and more challenging. This presentation provides some practical guidelines when using SAS® software in a client/server environment.

INTRODUCTION

Client/server requirements are driven by a number of factors:

- Large data warehouses, distributed data marts, “knowledge warehouses”.
- Mergers/acquisitions consolidating some staff and facilities, while distributing others.
- The web used for both information delivery and analysis.

Design issues include

- Thin or fat client?
- Web-based?
- Processing requirements of client.
- What kind of users?
- Security
  - data and applications.
  - internal or external users?
- Automation
  - push or pull?

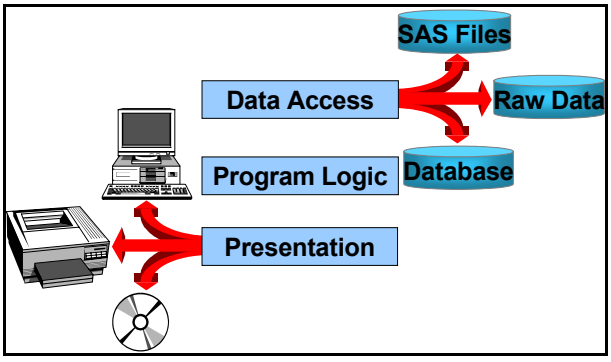
OVERVIEW

Client/server applications involve co-operative processing among 2 or more computers.

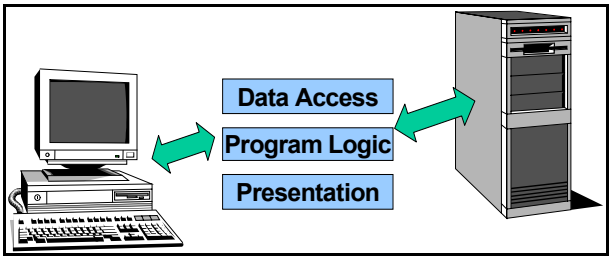
- The *client* is a process that makes a request for a service.
- The *server* is a process that fulfils the request from the client.
- The *network* is the medium used to transmit the requests and the results.

Applications can be thought of as a series of functional components. This is not news - remember HIPO charts?

Consider an application to be made up of three basic functions:



In a single-platform application, all components execute on the same computer. In a client/server application, you can choose among multiple computers for each functional component:



To design a flexible client/server application, you need to

- execute programs on any computer (***compute services***).
- transfer data among computers (***data transfer services***).
- process remote data on the local computer without transferring the entire file (***remote library services***).

TRADITIONAL CLIENT/SERVER SYSTEMS

In a traditional client/server system, SAS software is installed on both the client and the server. A SAS session runs on each computer, and the two sessions cooperate with each other while the connection is active.

Getting the two SAS sessions to cooperate is a lot like making a phone call. If Joe wants to call Jane on the phone, a number of events and conditions have to happen:

- both Joe and Jane must have a phone.
- there must be a physical line between the phones.
- Joe has to know Jane’s phone number, or be able to find it.
- Joe and Jane must be able to speak the same language.
- both people need to know how to use the phone - Joe has to know how to dial, and Jane has to know how to answer.

To establish a client/server environment between two SAS sessions

- the required SAS software products must be installed
  - SAS/CONNECT® on the client
  - SAS/CONNECT® or SAS/SHARE® on the server
  - SAS/SECURE™ if encryption is required
  - SAS/ACCESS® if the data access will be to non-SAS databases.
- the underlying network must be functional (outside of SAS)
- the client must be able to identify the server
- both client and server must use the same communications protocol
- the client must be able to start a SAS session on the server.

USING SAS/CONNECT

SAS/CONNECT uses the system options **COMAMID=** to specify the communications protocol to be used in the session, and **REMOTE=** to identify the server. A *script file* is often used to start the SAS session on the server. On the server side, the *spawner* program can be used on some systems to start the SAS session.

To set up for a connection to a server named MYUNIX from a Microsoft® Windows® client, submit

```
options comamid=tcp remote=myunix;
filename rlink
'!sasroot\connect\saslink\tcpunix.scr';
```

THE REMOTE= SYSTEM OPTION

- Identifies the remote (server) computer.
- Must be a valid SAS name.
- For TCP/IP, must be a recognized address or alias.
- Can be a macro variable.

IP ADDRESSES

Since the value of the REMOTE= system option must be a valid SAS name, an ordinary IP address would be invalid. To create an acceptable value

- use an alias in the TCP/IP HOSTS file
- use a SAS macro variable.

USING THE HOSTS FILE

Most companies use a TCP/IP feature called Domain Name Services to share a hosts file. If you use a Domain Name Server,

then your network administrator must assist you in creating an alias.

A local HOSTS file can also be used (in the \WINDOWS directory in Windows 95/98, in WINNT\SYSTEM32\DRIVERS\ETC in Windows NT™). The HOSTS file is a text file that can be viewed and changed with the SAS Program Editor, or any other text editor. Each line in the HOSTS file contains an IP address, and one or more aliases:

```
127.0.0.1       myunix
```

USING A MACRO VARIABLE

A SAS macro variable can be used if you can't or don't want to change your HOSTS file. Create a macro variable whose value is the IP address of the remote computer:

```
%let myunix=127.0.0.1;
```

Use the name of the macro variable as the value of the REMOTE= system option:

```
options remote=myunix;
```

> **TIP**: Don't make it a normal macro variable reference by coding &myunix - if you do, the actual IP address will become the value of the REMOTE= option.

HOW SAS/CONNECT USES THE REMOTE= SYSTEM OPTION

1. SIGNON asks TCP/IP to perform a lookup.
2. TCP/IP looks first for an actual IP address.
3. If an alias, TCP/IP looks for it in the local HOSTS file.
4. If not found, TCP/IP tries Domain Name Services.
5. If not found, TCP/IP returns an error to SAS.
6. If SAS gets an error back from TCP/IP, it checks for a macro variable.
7. If found, SAS substitutes the value and again asks TCP/IP to perform a lookup (back to step 1). If TCP/IP again returns an error, the signon fails.
8. If no macro variable exists, the signon fails.

> **TIP**: Since using a macro variable causes TCP/IP to do two lookups, signing on will be very slow. In a DNS environment, the shared HOSTS file can be very large, so you can really speed up the signon process if you use a local HOSTS file.

THE COMAMID= SYSTEM OPTION

- means COMmunications Access Method ID
- is used by SAS to determine the protocol to use when communicating with the server.
- also dictates which script file to use.

USING THE SCRIPT FILE

The script file is used by the SIGNON command.

- A fileref of RLINK is used by default.
- You can specify either a different fileref, or a file name as an option.
- Sample script files are provided in the !SASROOT\CONNECT\SASLINK folder.
- Script files (WIN) have an extension of SCR.
- Script file names are often a combination of the COMAMID and the target platform:

WHAT DOES THE SCRIPT FILE ACTUALLY DO?

The main purpose of the script file is to send a SAS startup command to the remote system. It can also log you on to the remote system.

- Some access methods do not require a script file (APPC, NETBIOS, SPX, CPIC).
- Using the spawner may also mean no script is required.

Programming a script file is very similar to writing a data step, with extensions so that you can send an receive data from the remote system. With these extensions, a script file can

- prompt you for your user id and password, and sign you on to the remote system.
- send a SAS startup command to the remote system.
- sign you off when you're finished.

The SAS session that runs on the server uses some special startup options:

DMR (Display Manager Remote) sets up a link between the LOG and OUTPUT windows on the client and the server.
NOTERMINAL tells the remote SAS session that there is no display output device attached.
- Means you can't invoke a command or procedure that opens a window (like DIR or FSVIEW)

NO$SYNTAXCHECK allows continuation of statement processing at the remote host regardless of syntax error conditions.
NONEWS can reduce network traffic.
CLEANUP specifies that SAS is to attempt to perform automatic continuous clean-up of resources that are not essential for execution. When CLEANUP is in effect, and an out-of-resource condition is encountered (except for disk full), no intervention is required by the user.

WHAT IS THE SPAWNER AND WHAT DOES IT DO?

The spawner is a process that runs on the server and listens for requests from SAS/CONNECT client sessions. When it "hears" a request, it starts a SAS session on the server. The spawner

- Encrypts data flow between client and server (starting with 6.09E and 6.11 TS040), including signon.
- Can authenticate users against an existing user database.
- Is available for NETBIOS, SPX, and TCP/IP on Windows 95, Windows NT and OS/2®. Required if using NETBIOS or SPX.
- Is available for TCP/IP on UNIX®.
- Is available for TCP/IP on OS/390® and Open VMS™ (Version 8).

INVOKING THE SPAWNER

The spawner is executed either from the command line, or as an operating system service (Windows NT only). Since the spawner needs to be able to locate the SAS executable , it normally uses the directory containing the SAS executable as its working directory. The syntax is

Windows, OS/2:	`SPAWNER <options>`
UNIX, VMS™:	`SASTCPD <options>`
OS/390:	invoked as a started task.

SPAWNER COMMAND-LINE OPTIONS (ALL PLATFORMS)

COMAMID communications access method. Tells the spawner what protocol to use (same as SAS system option on the client).
NOCLEARTEXT prevents access by clients that don't support encryption (i.e. SAS versions prior to 6.09E or 6.11 TS040). Ensures an encrypted data flow.
NOSCRIPT prevents a connection from a client using a script. Requires that the client set the TCPSEC macro variable to *userid.password* or _PROMPT_.

SPAWNER COMMAND-LINE OPTIONS (WINDOWS)

FILE indicates the name of a batch file used to start the remote SAS session. Use with NOSCRIPT.
FILEPROMPT will prompt the client for the name of the batch file to be used to start SAS. If the client does not specify a batch file name, the batch file specified in the FILE option is used, or of no FILE option is provided, SAS.EXE is used.
AUTHSERVER specifies the name of an NT server or domain that can be used to validate userids and passwords.

SECURITY tells the spawner to use native Windows NT security. To run the spawner in a secured mode, the user invoking the spawner must have administrator privileges, plus the "Act as part of the operating system" and "Replace a process level token" rights. To connect to the spawner program, a user must have the "Log on as a batch job" right.
PROTECTION tells the spawner to use userids and passwords to restrict access to the remote host. For TCP/IP, the userids and passwords are entered by the client either by using a script file, or by creating the TCPSEC macro variable. The userid and password entered by the user are validated against the contents of the ACI.DAT file, which must reside on the remote host.
INSTALL on Windows NT, this option causes the spawner to run as an NT service, meaning it can execute without a user being logged on. Typically used in an server environment only. Use the DELETE option to remove the service.

TIP: When you install SAS on Windows NT, the installation dialog contains an option to install the SAS Job Spawner as a service. Unless you are planning to use the machine as a server, you should NOT take this option. If you need to install the spawner as a service later on, you can always do it from the command line.

TIP: Use SECURITY and AUTHSERVER= instead of PROTECTION. Your users will be able to use their normal userids and passwords. Don't forget to set them up with the appropriate privileges.

SPAWNER COMMAND-LINE OPTIONS (UNIX)
SASCMD specifies the name of an executable file that starts a SAS session when signing on without a script.
BACKGROUND specifies that the spawner program should run as a background process. The default is for the spawner program to run in the foreground.
AUTHPROG specifies the name of the executable that is used to authenticate users.
SERVICE specifies the name of the service that the UNIX spawner program uses to listen for incoming requests. Must be specified, and is used by clients in the REMOTE= option.
SHELL allows the SAS session that is invoked by the UNIX spawner program to create a shell. Necessary if the remote host will execute commands.
USER allows the UNIX spawner program to run without root privileges. SAS assumes the security status of the user or the administrator who started the spawner program. The default action is to assume the privileges of the user whose username and password are given to the UNIX spawner program. This option may not work on all UNIX systems.

TIP: Using the NOSCRIPT option together with the FILE or SASCMD opions of the spawner means that the user doesn't need to know the actual location of the SAS executable on the server. For systems support personnel, it means that SAS can be moved to a new location without having to worry about failed connections.

SIGNING ON TO THE SERVER
To kick off the process of establishing the cooperative processing environment, you need to sign on to the server by using the

- Run ⇨Signon menu selection (V8)
- Locals ⇨Signon menu selection (V6)
- SIGNON command
- SIGNON statement.

The syntax of the SIGNON statement (V6) is

```
signon <script fileref|script filename
|NOSCRIPT> | <remote name> <TBUFSIZE=value>;
```

For V8, the SIGNON statement has been significantly enhanced:

```
signon <options>;
```

where any combination of options can be used.

SIGNON STATEMENT OPTIONS
Script Fileref specifies a previously defined fileref for the script file to be used to establish the link. DEFAULT: RLINK
Script Filename specifies the file name (fully qualified) of the script file to be used to establish the link.
CSCRIPT=value (V8) specifies the script file to be used during sign on. It may either be a fileref or a quoted, fully-qualified pathname. If the fileref, the filespec, and the CSCRIPT= option are specified, the last specification overrides and takes precedence over the others.
NOSCRIPT
NOCSCRIPT (V8) specifies that no script should be used to sign on because the user is running an access method (such as APPC) that does not use a script file.
remote name
CONNECTREMOTE=remote name (V8)
CREMOTE=remote name (V8)
REMOTE=remote name (V8) specifies the session to be the target of the connection. This may be a port number, a 3270 emulation defined session name (long or short), etc.
TBUFSIZE=value specifies the buffer size that SAS/CONNECT should use for transmitting data. TBUFSIZE should only be specified with program-to-program communications access methods, such as APPC, DECnet, NETBIOS and TCP/IP. The default buffer size is 32K.
CONNECTWAIT=YES|NO
CWAIT=YES|NO
WAIT=YES|NO specifies whether remote submits are to be executed synchronously or asynchronously by default during a SAS session. The default setting can be overridden by specifying the CONNECTWAIT= option in subsequent RSUBMIT statements for a specific remote submit.
In synchronous processing (the default), you must wait for the remote processing to complete before control in the local SAS session is returned to you. In asynchronous processing, after the RSUBMIT block begins to execute on the remote host, you will regain control of your local SAS session to continue local processing or to continue to RSUBMIT to other remote sessions.
CMACVAR=value specifies the name of the macro variable to associate with this remote session. The macro variable will NOT be set if the SIGNON command fails due to incorrect syntax. Other than this one exception, the macro variable is set when the SIGNON command is completed. The variable will have one of the following values:
0 indicates that the SIGNON was successful.
1 indicates that the SIGNON failed.
2 indicates that you have already signed on to this remote session.
If the SIGNON is successful, the macro variable is set, and it becomes the default macro variable for this remote session. This default can only be overridden by a subsequent successful RSUBMIT command that has the CMACVAR= option specified.
CONNECTSTATUS=YES|NO
CSTATUS=YES|NO
STATUS=YES|NO specifies the default setting for the display of the status window.

TIP: Set STATUS=NO in batch or non-interactive jobs that perform many data transfer steps. Setting the option on the SIGNON statement means you don't have to remember to code it on each procedure step.

USING ENCRYPTION SERVICES
If you have a license for SAS/SECURE™, you can use encryption

services of the RSA BSAFE Toolkit or the Microsoft CryptoAPI. SAS proprietary encryption services are available with base SAS.

The RSA BSAFE Toolkit is supported on the following types of Open VMS, OS/2 and UNIX platforms:

- Open VMS Alpha
- Open VMS VAX
- OS/2
- AIX®
- Digital UNIX
- HP-UX®
- Solaris 2.

CryptoAPI from Microsoft is an application programming interface that provides access to the cryptographic services that are provided by:

- Windows 95 (as part of Internet Explorer 3.0+)
- Windows NT 4.0+ (as part of the operating system). Service Pack 3 or a subsequent release must be installed.

You must have either of the following packages installed on your Windows host to use CryptoAPI:

- Microsoft Base Cryptographic Service Provider, which supports weak encryption.
- Microsoft Enhanced Cryptographic Service Provider, which supports strong encryption.

Encryption services are supported by the following access methods:

- TCP/IP
- DECNet®
- NETBIOS

ENCRYPTION ALGORITHMS

Encryption services supports these encryption algorithms:

RC2 :A proprietary algorithm developed by RSA Data Security, Inc., RC2 is an alternative to DES. The algorithm expands a single message by up to 8 bytes. RC2 is a block encryption algorithm that encrypts data in blocks of 64 bits. The size of the output of the algorithm is always a multiple of the block size. The RC2 key size can range from 8 to 256 bits.

RC4 :A proprietary algorithm developed by RSA Data Security, Inc., RC4 is a stream encryption algorithm. A stream encryption algorithm encrypts one byte at a time. The RC4 key size can range from 8 to 2048 bits.

DES :An acronym for Data Encryption Standard, DES was developed by IBM. The algorithm expands a single message by up to 8 bytes. DES is a block encryption algorithm that encrypts data in blocks of 64 bits by using a 56-bit key.

Triple DES :Triple DES executes DES three times on the data in order to exploit a key size that is three times that of DES. The algorithm expands a single message by up to 8 bytes. DES is a block encryption algorithm that encrypts data in blocks of 64 bits.

SAS Proprietary :This provides basic encryption services on all platforms and requires no additional product licenses. The algorithm expands a single message by approximately one-third. It uses a 32-bit key.

> **NOTE:** There are U.S. export regulations regarding encryption technology. Not all encryption algorithms or key lengths are available in all countries.

SYSTEM OPTIONS FOR ENCRYPTION

Several system options are used to control encryption services:

NETENCRYPT = YES | NO

NETENCRYPT | NONETENCRYPT Set this option at both the local and remote hosts. At the remote host, this option specifies that encryption is required for each connection from a local host SAS session. At the local side, this option specifies that the local host must connect only to a remote host that supports encryption.

NETENCRYPTALGORITHM=("algorithm1", "algorithm2", ...)
NETENCRALG=("algorithm1", "algorithm2", ...) If you specify more than one algorithm, enclose the algorithm names in parenthesis and use commas to separate them. If there are embedded blanks in the algorithm name, enclose each algorithm with quotation marks.

Set this option at the remote host and, optionally, at the local host to specify one or more encryption algorithms to use in a SAS session. However, the local and remote hosts must have an algorithm in common. If you specify the option in the remote host session only, the local side attempts to select an algorithm that was specified at the remote host. If you also set the option at the local host and specify an algorithm that is not specified at the remote host, the attempt by the local host to connect to that remote host fails.

Valid values for this option are

- RC2
- RC4
- DES
- TripleDES
- SASProprietary.

NETENCRYPTKEYLEN = n

NETENCRYKEYLEN=n Set this option in either the local or the remote host SAS session. It specifies the key length to be used by the encryption algorithm.

Valid values for this option are

- 128 specifies strong encryption (1024-bit RSA and 128-bit RC2 and RC4 key algorithms).
- 40 specifies weak encryption (512-bit RSA and 40-bit RC2 and RC4 key algorithms).
- 0 no value is set. This is the default.

If you leave the option at the default value, sessions will attempt to encrypt at the longest key length they have in common.

NETMAC | NONETMAC This option controls the use of Message Authentication Codes (MACs) on network communications. A Message Authentication Code is the equivalent of a checksum that is used to ensure that the original message has not been modified. The MAC integrity checking adds an extra 16 bytes to RC4 encrypted messages and an extra 24 bytes to RC2, DES, and TripleDES encrypted messages.

> **TIP**: You need to weigh the need for security against resource consumption when using more sophisticated encryption algorithms.

USING COMPUTE SERVICES

When you use compute services, you can write programs on the client, and execute them on the server. The DMR option links the server's LOG and OUTPUT windows to the client's LOG and OUTPUT windows, so that you can immediately see the results of your program's execution.

To link the server's GRAPH window to the client's GRAPH window

- You must have SAS/GRAPH® on both systems.
- Use GOPTIONS DEVICE=GRLINK on the server.

> **TIP**: Add this option to your script file, or to the AUTOEXEC file on the server.

Execute a program on the server by using the Remote Submit facility:

- Locals ➪ Remote Submit from the menu bar (V6).
- Run ➪ Remote Submit from the menu bar (V8).
- RSUBMIT command.
- `rsubmit;…. endrsubmit;` statements.

The RSUBMIT command and the Remote Submit menu selection will submit the entire contents of the program editor to the server.

You can distribute processing between client and server using the `rsubmit;… endrsubmit;` statements.

Execute SQL statements on the server using Remote SQL Pass-through.

WHY USE COMPUTE SERVICES?

- Data may be in a form not accessible on the client (e.g. database data).
- Files may be too large to store locally.
- Security requirements may prevent copying the data.
- It's generally faster to run the program on the same platform as the data.

CONSIDERATIONS FOR REMOTE SUBMIT

- Since each SAS session has its own set of system options, reporting options like LINESIZE and PAGESIZE may differ.
- Page numbers will not be synchronized.
- Titles and footnotes need to be set on both systems.
- Any user-defined formats and macro programs must exist in a library on the server.
- System clocks will probably be different (TODAY() and TIME() functions return different results).
- Time zones may be different.
- SAS libraries allocated on the remote host cannot be viewed interactively on the client (DIR window, VAR window, etc.).
  - Use utility procedures like DATASETS and CONTENTS!
- For procedures that open windows, if coded in an remote submit block
  - some procedures (e.g. FSEDIT, FSVIEW) will fail.
  - Some procedures (e.g. REPORT) will automatically switch to non-interactive mode.
- Errors are sometimes harder to detect:
  - If you omit a RUN statement, there is no indication in the title bar that a procedure is running on the server.
  - You have two sets of macro error variables - &SYSINFO, &SYSRC, &SYSERR. Use %SYSRPUT to transfer the values of these variables to the client.

USING REMOTE LIBRARY SERVICES

Remote Library Services (RLS) is a form of data services that can allow you to process a server library as if it resides on the client computer. Subsetting options and statements such as WHERE, DROP and KEEP are passed to the server and executed as close to the data as possible. Using RLS means that you can

- use the client GUI to work with members of the library (DIR/VAR windows, VIEWTABLE, SAS/FSP® Software, SAS/AF® Software).
- Execute programs locally to process remote data.

Convenience comes at a cost, however:

- if the character sets of the client and the server are different, catalogs can not be processed.
- Observations are processed individually, or in small sets, rather than in large blocks, which will slow down processing of large data sets.

To use RLS, add the SERVER= option to a LIBNAME statement:

```
libname libref 'library location' server=<remote
ID>;
```

or

```
libname libref 'library location' server=<remote
ID.SAS/SHARE server ID>;
```

USING REMOTE SQL PASS-THROUGH

One way to improve RLS performance is to use Remote SQL Pass-through. Using this technique, you embed an SQL query inside another one, and the embedded query is executed on the server.

Using RLS, you could code

```
libname mydata 'library' server='server';
proc sql;
  select destination, sum(passengers)
    from mydata.counts
    group by destination;
quit;
```

All observations from mydata.counts would be transferred to the client, which would perform all the summarization.

With Remote SQL Pass-through, you would code

```
libname mydata 'library' server='server';
proc sql;
  connect to remote(server='server');
  select *
    from connection to remote
      (select destination, sum(passengers)
                from mydata.counts
                group by destination);
quit;
```

Only the summarized rows from the embedded SQL would be transferred to the client. Network traffic would be considerably less.

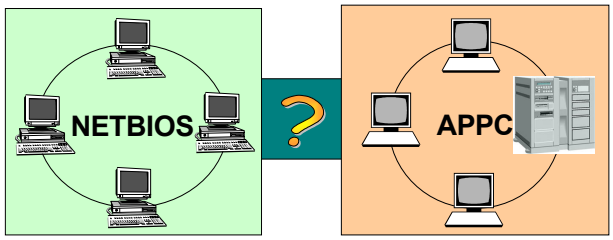
> **TIP**: Use Remote SQL Pass-through when you need to use RLS, and you also need to perform data combining or summarizing tasks that can be coded in SQL.

MORE COMPLEX NETWORKS

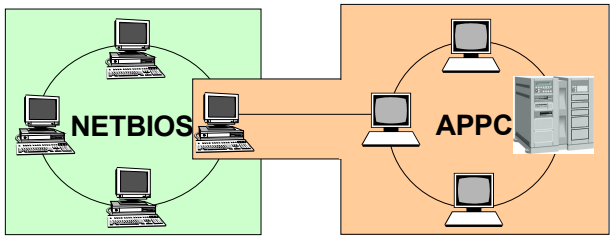
Your environment may include separate network segments that use entirely different protocols, or may include a class 1 or class 2 ODS (Operational Data Store). These environments require special processing that can be performed by the SAS/CONNECT Domain Server. The Domain Server provides

- indirect messaging services
- direct messaging services
- agent scheduling services
- protocol gateway services.

CONNECTING NETWORKS WITH DIFFERENT PROTOCOLS



The Domain Server can act as a protocol gateway between network segments that use different protocols. Only one machine in the network needs to have both protocols installed.



To use the Domain Server as a protocol gateway, you need a dedicated SAS session running on OS/2 or Windows NT.

The Domain Server is invoked by executing PROC DOMAIN using the *protocol* option. If one of the protocols is TCP/IP, you will need an entry for the Domain Server in the SERVICES file. The COMAMID, COMAUX1 and COMAUX2 options should be specified on the command line or in your configuration file.

```
proc domain protocol serverid=domsrv;
```

In the client session, use macro variables GWHOST to identify the node name of the Domain Server, TCPGW to identify the Domain Server itself, and TCPSEC to hold your userid and password, _PROMPT_, or _NONE.

```
%let GWHOST=ABC.COM;
%let TCPGW=GWHOST.DOMSRV;
%let TCPSEC=_prompt_;
options comamid=tcp remote=RMTNODE;
signon;
```

In this case, RMTNODE is the name of the remote computer you wish to sign onto, NOT the name of the Domain Server.

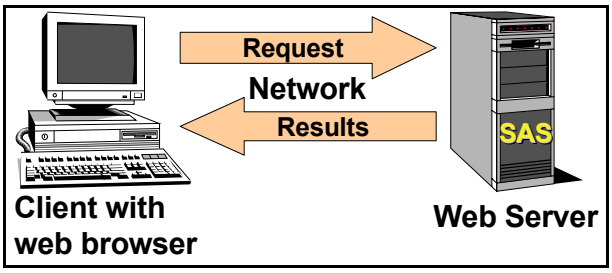
CLASS 1 AND CLASS 2 OPERATIONAL DATA STORES
An Operational Data Store is similar to a Data Warehouse, in that the information it contains is subject-oriented and integrated. However, an ODS usually doesn't contain as much historical information as a Data Warehouse, and may contain more detail. The reason for the differences is that an ODS is used for tactical decision-making, rather than strategic decision-making. There are generally considered to be three types of ODS:

- Class 1 - updated from the production systems within seconds.
- Class 2 - updated fro the production systems on a regular basis, usually hourly or several times a day.
- Class 3 - updated overnight or on the same schedule as the Data Warehouse.

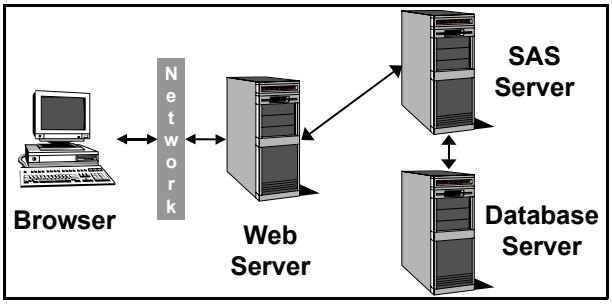
Class 1 and 2 ODS's can benefit from the Domain Server's messaging services, as well as agent scheduling services. A Class 1 ODS could be updated using direct messaging, and a class 2 ODS by a combination of indirect messaging and agent scheduling.

CLIENT/SERVER AND THE INTERNET

In a web-based client/server environment, SAS Software is installed on the server, and the client uses a web browser.



In fact, web-based configurations can be quite complex, with separate servers for both SAS Software and databases.



Results sent to the client can be

- Static information
  - HTML residing on a web server
- Dynamic information
  - HTML generated by the SAS server and passed back to the web server on demand.
- Java applets residing on a web server.

STATIC REPORTING ON THE WEB
Static reports are created offline and stored in a location accessible from the web server. There are a number of ways to use SAS Software to create static web reports:

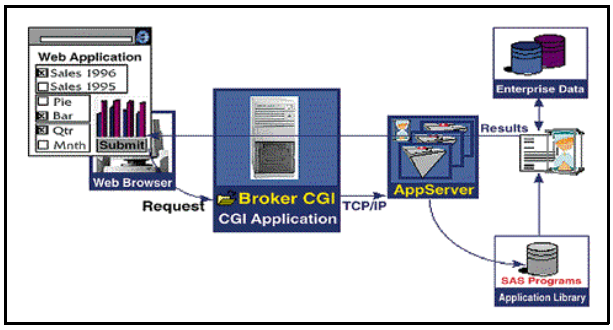
- Data step with FILE and PUT statements to write HTML.
- Web publishing macros (V6).
  - %OUT2HTM, %DS2HTM, %TAB2HTM, %DS2GRAF.
- Output Delivery System (ODS) (V8).
- HtmSQL® in batch mode.

There is no special client/server setup required when using these methods. They are mentioned because a large number of users need only this type of report.

DYNAMIC WEB APPLICATIONS USING SAS/INTRNET™
Dynamic web applications are executed in response to a user request, and need to generate web pages by combining data values and HTML code. You can use

- the Application Dispatcher feature of SAS/Intrnet
- htmSQL (part of SAS/Intrnet)
- webAF (part of Appdev Studio)
- webEIS (part of Appdev Studio).

USING THE SAS/INTRNET APPLICATION DISPATCHER



The SAS/Intrnet Application Dispatcher uses two components to create dynamic web content.

- The Application Broker.
- The Application Server.

Optionally, the Load Manager can be used to balance resources across multiple Application Servers.

The first component, called the Broker, is a Common Gateway Interface (CGI) program that resides on the web server. CGI is a standard for interfacing external applications with information servers. It enables the execution of real-time programs to create dynamic web content. When a user presses the Submit button or clicks on a hypertext link, the Web browser sends the form information to the Web server, which immediately invokes the Broker using the CGI protocol. The Broker then contacts the Application Server.

The second component, the Application Server, is a SAS session running the APPSRV procedure. When the Server receives a request from the Broker, it defines macro variables and an SCL list containing the name/value pairs passed from the HTML page. Next, the Server examines the program name to determine the type and location of the program being run. If the Server does not find the program, it generates an error page that displays in the user's browser. Next the Server defines a special SAS fileref for program output and runs the requested Dispatcher program.

Output from the program is sent to the Broker using the predefined fileref, and the Broker checks the HTTP header for consistency. The output is then sent back to the web server, which streams it back to the web browser.

INVOKING THE BROKER FROM A WEB PAGE
You can invoke the Broker using one of several methods:

- using the Broker in the HREF attribute of a hypertext link.
- using the Broker an ACTION attribute of an HTML form.
- creating an inline image whose source is a reference to the Broker.
- using a reference to the Broker in a Java Applet, ActiveX control, or Plug-in.

Example:
```
<a
href=http://websrv/cgi-bin/broker.exe?_progra
m=mylib.mysascode.sas&_service=default>
```

In addition to providing the user interface, the HTML document provides the following information:

- the location of the Application Broker, as defined in the ACTION attribute of the HTML FORM tag
- data contained in the _PROGRAM and _SERVICE required fields that identify the name of the Dispatcher program and the service that will be used to process it
- name and value information that is necessary for processing and is contained in optional fields such as INPUT fields.

TIP: The syntax for coding the location of the Broker depends on the operating system of the web server's computer. For example, if the web server is UNIX, then the location of the Broker is simply ***broker***, but if the web server is Windows, the location is ***broker.exe***. Your webmaster is the best source of information about the web server.

SPECIFYING THE SERVICE

The Broker uses the _SERVICE parameter together with its configuration file to direct your request to an Application Server.

Three types of services are available:

- Socket Service: The Broker sends the data via a TCP/IP socket to a pre-started Application Server. The Application Server must be running on a machine connected to the Web server machine via a TCP/IP network. Multiple servers can belong to the service and spread the processing load across multiple processes or systems.
- Pool Service: The Broker contacts the Load Manager to find an available Application Server. If no server is available, the Load Manager may start a new server to handle the request. Once the request is complete, the new server is added to the service pool and is available for future requests.
- Launch Service: The Broker launches a new Application Server for each new request. The Application Server runs on the Web server machine and terminates when the request is complete.

TIP: To use a Launch Service, the web server and the SAS server must be on the same machine. Check with your webmaster to find out if this is permitted. Many webmasters feel that this places too much load on the web server.

The Broker configuration file defines all the available services. A service normally corresponds to one Application Server, but it can be more than one. For socket services, the configuration file also specifies the machine name or IP address and the TCP/IP port name or number that should receive the request.

The Broker attempts to communicate with the Application Server. If the server is using a Socket Service, it accepts requests from the Broker on a defined TCP/IP socket. If the Application Server does not respond, the Broker performs one of the following actions:

- If the service defines more than one server, the Broker selects another Application Server.
- If the service defines only a single server or if none of the Application Servers respond, the Broker generates an error page that displays in the user's browser.

SPECIFYING THE PROGRAM TO EXECUTE

The program to execute is specified in the _PROGRAM parameter passed to the Broker. There are four types of programs you can invoke:

- SAS programs
- SCL entries
- Macro programs
- SOURCE entries.

The specific entry is coded using a three or four-level name. The first level is a libref pointing to the location of the program.

SAS PROGRAMS

SAS programs are specified

`_program=`*`libref.program`*`.sas`

SAS programs are stored in source form in external files. These files

- must have a `.sas` filename extension on directory-based platforms. The filename must match the combined second and third levels in the value of `_program`.
- must be contained in a partitioned data set (PDS) on OS/390 systems. The PDS member name must match the second level in the value of `_program`.

TIP: If the Application Server's operating system is case sensitive in terms of file names, ***so is the SAS program name***.

SCL ENTRIES

SCL entries are specified

`_program=`*`libref.catalog.entryname`*`.scl`

SCL entries are stored in SAS catalogs, and must have an entry type of SCL. Names are not case sensitive.

TIP: Make sure you use only non-visual objects and functions in your SCL code. The Application Server is a non-windowing environment, much like the remote SAS session in a SAS/CONNECT environment.

MACRO PROGRAMS

Macro programs are specified

`_program=`*`libref.catalog.program`*`.macro`

Macro programs must be stored in a SAS catalog and have an entry type of MACRO. Names are not case sensitive.

SOURCE ENTRIES

SOURCE entries are specified

`_program=`*`libref.catalog.program`*`.source`

SOURCE entries can contain the same code as SAS Programs, and are stored in a SAS catalog with an entry type of SOURCE. Names are not case sensitive.

USING ADDITIONAL PARAMETERS IN YOUR SAS CODE

You can provide additional input parameters to your SAS programs using name/value pairs in your HTML code. These can be provided on the hyperlink, or as fields in an HTML form. Name/value pairs are converted to SAS macro variables and an SCL list.

Macro variable values can be obtained within the application by

- direct reference (&var)
- using the %SUPERQ macro function
- using the SYMGET function (DATA step)
- using the SYMGET, SYMGETC, or SYMGETN functions (SCL).

The Application Server will strip any unsafe characters (as defined by the UNSAFE option on PROC APPSRV). These are normally

- single quote
- double quote
- ampersand
- semicolon.

TIP: If you need to retrieve all characters in macro variable, including unsafe characters, use the APPSRV_UNSAFE function:

```
var=appsrv_unsafe('macrovar');
```

TIP: You may have to use conversion functions, since all macro variables are stored in character format.

```
Age=input(symget('age'),3.);
```

SETTING UP THE APPLICATION SERVER

You can control the behavior of PROC APPSRV using a number of options and statements. The general syntax is

```
PROC APPSRV PORT=n <options>;
  ADMINLIBS statement;
  ALLOCATE statemnt(s);
  DATALIBS statement;
  LOG statement;
  PROGLIBS statement;
  REQUEST statement(s);
  SESSION statement(s);
RUN;
```

PROC APPSRV OPTIONS

PROC APPSRV has a number of options. PORT= is the only required option, all others are optional.

ADMINPW=password allows the user to restrict access to certain administrator programs. The server has several built-in programs such as STATUS and STOP. If ADMINPW is enabled, the password must be supplied in the request (using the _ADMINPW variable) in order to run the administrator program.

AFPARMS='string' is a quoted string that is appended to the SAS/AF command when invoking user programs written in SCL. It can be used to pass a variety of parameters to the SAS/AF environment, but the primary use will be to enable the SCL debugger. To invoke the SCL debugger, you should compile your SCL program with debug on and then start the server with:

```
AFPARMS='debug=yes'
```

AUTH=scheme
GUESTUSER
GUESTPASS specifies the authentication scheme. The default scheme (AUTH=NONE without GUESTUSER being specified) is similar to previous releases of the Application Server and causes all requests to be run under the username under which PROC APPSRV was started. Specifying GUESTUSER (and the corresponding GUESTPASS) will cause all requests to be run under the GUESTUSER username. All access to catalogs, data sets and external files will be checked against this username. The AUTH=HOST scheme that denotes a secure Application Server requires a username and password with each request.

The username and password may be specified with the reserved variables _USERNAME and _PASSWORD (and optionally _PASSWORD2). GUESTUSER and GUESTPASS (and optionally GUESTP2) may be used to specify default values if they are not specified with the request. If the username is not specified by either the _USERNAME variable or by the GUESTUSER option, the request will be rejected (unless the LOGIN option is used.) Usernames and passwords are saved with sessions, so requests connecting to an existing session do not need to and cannot specify a new username and password.

GUESTP2 This is used only in Open vms environments because Open vms can accept two passwords.

LOCALIP=ip-address allows you to manually override the local IP address used by the Application Server. In rare cases, the local IP address returned by the operating system is not usable and a manual override is necessary.

LRECL=n is the logical record length for _WEBOUT filerefs. The default is 65535.

PORT=n specifies the request socket for the Application Server.

- If a numeric value is supplied, the value is used as the TCP/IP port number on which the server will listen for requests.
- If an alpha or numeric value is supplied, it is assumed to be a network service name.
- If zero is supplied, PROC APPSRV will choose an available port. Used only for launch or pool services.

PROGRAMS=n specifies the maximum number of requests that can execute concurrently. The default is 1. This option is experimental in this release of the Application Server and should not be used for a production environment.

UNSAFE='string' specifies a quoted string containing a list of characters that should be stripped from values in the request data (the name/value pairs). Normally used to strip characters from input values that may cause unwanted SAS Macro language processing, usually the

- single quote
- double quote
- ampersand
- semicolon.

Because this list is enclosed by single quotes you can represent a single quote by placing two single quotes within the quoted string in the following manner:

```
UNSAFE='&";'''
```

PROC APPSRV STATEMENTS

ALLOCATE STATEMENT

```
ALLOCATE LIBRARY libref <engine>
  'library-path' <options>;
ALLOCATE FILE fileref <device>
  'directory-or-PDS-path' <options>;
```

Defines a file or library that the Application Server will assign. Options to these statements are the same as the LIBNAME and FILENAME statements.

TIP: Use the ACCESS=READONLY option on the ALLOCATE statement for program libraries and data libraries that do not need to be updated during the session. This will protect your data and programs from intentional and non-intentional corruption.

ADMINLIBS STATEMENT

```
ADMINLIBS libref1|libref.catalog1|fileref1
  <...>;
```

Declares which libraries, filerefs, and catalogs contain programs that can be run by an administrator using the _ADMINPW password.

DATALIBS STATEMENT

```
DATALIBS libref1|fileref1 <...>;
```

These logical libraries must be assigned in an ALLOCATE statement in the same server procedure. Any libraries defined externally to SAS (such as in JCL code) are automatically permanent data libraries and should not be listed in the DATALIBS statement. In previous versions of the Application Server, global data libraries or files were allocated in the `permdata.sas` file. The Application Server now enables you to:

- assign the logical libraries externally to SAS
- allocate them with an ALLOCATE statement and then list them in the DATALIBS statement.

LOG STATEMENT

```
LOG DISPLAY=NONE|ERRORS|ALL
  SYMBOLS=NONE|ERRORS|ALL FILE=fileref
```

```
APPEND|REPLACE;
```

The Version 8.0 Application Server has several options to control the content and operation of the SAS log. The SAS log can be redirected to a new file based on the date, day of week or time. The log contains information about each client request. The log can be limited to a brief note for each request or the complete SAS log for the request can be captured.

PROGLIBS STATEMENT

```
PROGLIBS libref1|libref.catalog1|fileref1
  <...>;
```

Declares which libraries, filerefs, and catalogs contain programs that can be run on an Application Server.

When a request is received by the Application Server, the PROGLIBS list is scanned for a match on the first one or two levels in the program name supplied in the special request variable `_program`. If a match is found then the program is executed.

REQUEST STATEMENT

```
REQUEST INIT=program-name TERM=program-name
  LOGIN=program-name TIMEOUT=n MAXTIMEOUT=n
  READ=n FROMADR=("ipaddress1" <...
  "ipaddressn">);
```

Controls how a request should be processed by the Application Server.

SESSION STATEMENT

```
SESSION INIT=program-name TERM=program-name
  TIMEOUT=n MAXTIMEOUT=n VERIFY=(var1
  <...varn>) COOKIE=( <VALUE=varname>
  <PATH=varname|"string">
  <DOMAIN=varname|"string"> );
```

Controls how a session should be administered by the Application Server.

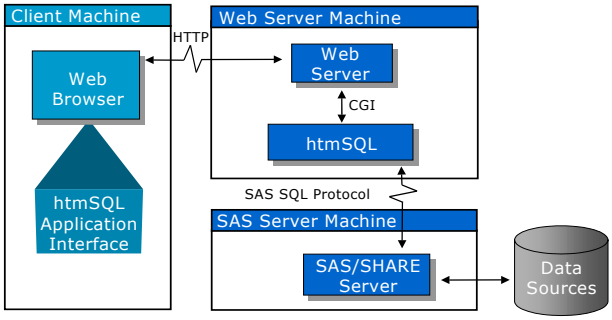
USING HTMSQL

htmSQL is a CGI program that is invoked like the Broker. However, htmSQL contains its own directives that enable you to

- specify the SAS/SHARE server to be used
- send an SQL statement to the SAS/SHARE server
- process each row that is returned (if a SELECT)
- handle error conditions
- handle a query that returns no rows.

htmSQL uses remote SQL pass-through, so it requires the use of a SAS/SHARE server. It does not use the Application Server.

HOW HTMSQL WORKS



An htmSQL file contains three components:

- HTML code
- SAS SQL code to query or manipulate data
- htmSQL directives.

htmSQL files by convention use an extension of HSQL. The htmSQL program reads the .HSQL file, searching for htmSQL directives. As these directives are encountered, they are processed.

INVOKING HTMSQL

htmSQL is invoked by

- creating a reference on a web page using
  - an HTML form with an ACTION tag
  - a hyperlink containing the URL for the htmSQL program
- from the command line
  - redirect the output to a file to generate the web page.

TIP: Like the Broker, the htmSQL CGI program is invoked differently, depending on the web server's operating system. However, if the web server supports a feature called script mapping, a reference to the .HSQL file is all that's needed in the invocation. Script mapping works like a file association in Windows, and allows the automatic execution of a program based on a file extension. Talk to your webmaster about setting up script mappings for .HSQL files.

TIP: You can use the features of htmSQL to generate static web pages by running htmSQL from the command line.

HtmSQL directives are similar to HTML tags:

- there are paired and standalone directives
- for paired directives, there is a beginning and an ending, and the ending directive begins with a /.

All htmSQL directives are enclosed in curly brackets or braces. In general, an htmSQL file contains at least one QUERY or UPDATE section with

- an SQL section containing the SQL statement to be passed to the SAS server.
- optional SUCCESS, ERROR, NOROWS sections.
- an EACHROW section (for queries only) containing instructions on how to format the results generated by the SQL section immediately preceding that EACHROW section.

htmSQL can also process symbolic references, which can be passed from the invoking web page as name/value pairs.

GENERAL LAYOUT OF AN HTMSQL FILE

For a query application, an htmSQL file is usually laid out like this:

```
<html>
<head>
  ...HTML tags...
</head>
<body>
  ...HTML tags...
  {query server="host:port"}
    {sql}
     ...SQL query...
    {/sql}
    {eachrow}
    ...HTML formatting for returned data...
    {/eachrow}
  {/query}
</body>
</html>
```

DEFINING SERVER LIBRARIES

The source of data for your queries or updates must be either a SAS library, or a DBMS. To define a SAS library to be used in an SQL statement, you can

- predefine it with a LIBNAME statement in the SAS/SHARE server session. From an administration point of view, this may be the easiest method. Libraries become administrator-defined, and remain available for the duration of the SAS/SHARE session, unless deallocated by the server administrator using the OPERATE procedure.
- predefine it using the DSDEF utility. Using this method

means that htmSQL dynamically allocates the library, and then frees it when the query is completed.
- define it in the htmSQL source file using the {LIBRARY} directive. Dynamic allocation is performed.

WHICH TECHNIQUE SHOULD YOU USE?
There are no functions available in htmSQL that cannot be performed using the Application Server. So, the decision on which technique to use can be based on other factors:
- If your application updates SAS data sets, and multiple users will be accessing it concurrently, use htmSQL, because the SAS/SHARE server will ensure the integrity of your data.
- If your program requires multiple steps, use the Application Server.
- If your application can be coded using the SQL procedure, consider using htmSQL. The resulting application may be written in fewer, less complex, lines of code (HTML does not have to be written out by PUT statements).

CONCLUSION

There are a number of methods available to implement client/server processes using SAS software. One of your major decisions is whether to use thin-client or fat-client setups.

Advantages of fat client solutions are
- The full power of SAS software is available to all clients.
- Fewer layers exist between the client and the data.
- Wide variety of services available (compute services, data services, remote library services). Lots of flexibility.

Considerations for fat-client solutions are
- All users may not be able to use SAS software effectively (use SAS Training® services to address this issue!).
- Maintenance and deployment of applications is more complex and time-consuming.

Advantages of thin-client solutions are
- Deployment of applications is easier.
- Users need to know how to use only the browser and the application.

Considerations for thin-client solutions are
- Complex web server and application server setup.
- More complex security issues.

Sometimes, a mixture of thin and fat-client environments is best, depending on the type of user. For the knowledge workers that are working with data in a largely unstructured fashion, the fat-client setup gives them the most flexibility. For other, more casual users of the data, who access only specific applications, the thin-client solution would apply.

REFERENCES

SAS Institute, Inc. (1999), *SAS Online Doc™, Version 7-1*, Cary, NC:SAS Institute, Inc.

SAS Institute, Inc. (1999), *SAS/Intrnet Software, Version 8*, Cary, NC:SAS Institute, Inc.

CONTACT INFORMATION

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

John Laing
SAS Institute (Canada) Inc.
181 Bay Street, Suite 2220
Toronto, Ontario, Canada M5J 2T3
Work Phone: 1-416-307-4556
Fax: 1-416-363-5399
Email: John.Laing@sas.com