

Using SAS/IntrNet™ Software

Kevin Davidson
FSD Data Services, Inc.

ABSTRACT

This paper will demonstrate the best methods of running SAS programs with SAS/IntrNet software. Examples of SAS programs executed via the web will be shown. Installation issues will be discussed, such as the optional settings of the Application Dispatcher and the efficiencies of each (the Application Dispatcher is the intermediary between the web browser and the SAS programs that retrieve or update data.) One topic will be customizing the broker configuration file and PROC APPSRV (the procedure which invokes the Application Server). Another will be the options for invoking SAS programs.

INTRODUCTION

SAS/IntrNet software allows you to build dynamic Web applications and gives you the ability to dynamically query your data, generate reports on demand, and perform many of the same analysis that you would run in an interactive SAS session. Obviously, taking the power of SAS software to the Web has tremendous advantages for a wide variety of companies. The purpose of this paper is to discuss some of the options within SAS/IntrNet software that make for an efficient operation and show how you can use the built in options to keep a fairly easily maintained web server.

In the ensuing pages, the following issues will be discussed:

Customization options:

Decide on socket, pool, or launch services

Broker.cfg

- Timeout option
- Administrator name and e-mail address
- Fieldwidth for textarea boxes
- Exporting fields
- Creating fields using SET statement
- Load Manager

PROC APPSRV

- ADMINPW option
- Unsafe option

- Short macro for simplifying the ALLOCATE, DATALIBS, and PROGLIBS statements.
- Request init
- Statistics
- Using datalib to use aggregate naming
- Log statement

A brief review of the plusses and minuses of 4 different program submit styles

SAS/IntrNet SOFTWARE

SAS/IntrNet software is composed of a number of components including Application Dispatcher, HTMSQL, and the MDDDB Report Viewer. An Application Dispatcher server is a SAS session that serves out requests from thin clients and returns results via the browser. Parameters are passed to the Application Dispatcher and can be accessed in the form of macro variables. One of the parameters is '_program' which tells the Application Dispatcher which SAS program to run (usually either a standard .sas program or a compiled macro).

CUSTOMIZATION OPTIONS:

Deciding on type of service

The on-line documentation for SAS/IntrNet does an excellent job of presenting the pros and cons of the three different types of services. For discussion purposes, the following text on services is adapted from the on-line documentation.

A service can be a socket, pool, or launch service. The features, advantages, and disadvantages of each of these service types are discussed below.

Socket Services

Socket services consist of one or more Application Servers that run continuously, servicing client requests. Socket services are generally started whenever a machine is restarted (either manually or by an operating system mechanism for starting processes at boot or login time).

The service usually runs until the machine is shut down. Socket services are relatively simple to configure and manage and are adequate for most applications

Advantages

- Socket services are supported on all SAS/IntrNet platforms. Other service types are not supported everywhere.
- The server is already running by the time a client request appears, so clients do not have to wait for a server to start.
- The administrator has explicit control of resources allocated to the service: the administrator can control how many servers are run on each system and which resources are allocated to each server.
- Increasing load can be handled by adding more servers to the service.

Disadvantages

- Servers must be started and stopped manually or by the operating system. No automated start-up and shutdown service is provided by SAS/IntrNet software.
- No dynamic scaling to meet increasing loads is provided. A fixed number of servers are available to handle all client requests. A few long-running requests can slow the entire service for all clients.

Pool Services

Pool services consist of a pool of Application Servers shared by clients. Based on system loading the servers are started and stopped by the Application Load Manager. Numerous options are provided to fine-tune the operation of a pool service. Pool services combine some of the advantages of socket and launch services.

Advantages

- Servers are started as needed. If all servers in the service are busy, the Load Manager can start an additional server.
- Servers can be reused by new clients once they are started. A started server remains in the pool until an idle timeout is reached and the server is stopped.
- Unlike launch services, pool services can be on a different system than the Web server and can be distributed across multiple server systems.
- Using the SAS Spawner, servers can be started under specific usernames to control access to system resources.

Disadvantages

- Installation and configuration are more complex for pool services. The Application Load Manager

must be installed. The SAS Spawner must be installed in most cases.

- Client requests might have to wait for a new server to start, although this is typically no worse (and could be better) than waiting for currently executing requests to complete in a socket service.

Launch Services

A launch service starts a new Application Server for each client request. An existing server is reused only for applications that use sessions or the `_tmpcat` catalog for IDS output. Most of the features of launch services are better provided by pool services (a new feature for Version 8). Launch services are not generally recommended for new installations.

Advantages

- Server start-up is automatic for each request. Once the launch service is configured, little or no additional administration is necessary.
- Requests run in a separate server, so a long-running request will not block access to the service for other clients.
- Many requests can run in parallel, assuming that the system will support the load.
- Ill-behaved applications that "crash" or "hang" a server will not affect other client requests.

Disadvantages

- Launch services are started by the Application Broker and must run on the same system as the Web server.
- Each new request incurs the resource overhead and delay of starting a new server session.
- Launch services are not suitable for high user loads. There are no settable limits on the server load. The service will attempt to start a new server for each new client. In an extreme case, 200 simultaneous users could cause 200 servers to be started, likely causing extreme "memory thrashing" and very slow response for all users. Most Web servers have limits on the number of simultaneous CGI requests that could help to control this problem.
- Each launch service request must incur the additional time for starting a SAS session.
- Launch services are not supported on CMS, OpenVMS, and OS/390 platforms.
- Launched servers can be difficult to shut down. A launched server that creates a session or `_TMPCAT` catalog will continue running until an idle timeout is reached. These servers cannot be shut down other than by interrupting the server process.

There are a number of things that can be done to customize your SAS/IntrNet configuration. Here we will focus on customizations that can be made within two files, broker.cfg and PROC APPSRV. Broker.cfg is the main configuration file.

Broker.cfg - The broker configuration file is the configuration file for the Application Broker. Among the items that you might find useful to alter are:

1) One of the options that can be set is the 'timeout' period, after which the end-user will see an error message. To manually set this option, put a line similar to the following anywhere in your broker.cfg:

```
Timeout=60
```

This sets the timeout option to 60 seconds. You will want to carefully analyze the expected times that your programs may take before setting this value. You need to take into account the speed of the machine, speed of connectivity, and amount of traffic. It is a good idea to periodically check the run times of your web program submissions to determine the causes of any slowdowns. Should an end-user experience a timeout they will see something similar to this:

Timeout error

The program ran longer than its allotted timeout period (60 seconds). This could happen if the timeout is too short, if the server is unavailable or busy, or there was an error invoking the SAS server. Note: program may still be running.

Kevin Davidson, KevinD@FSD.nu

This request took 6.38 seconds of real time (v8.2 build 1391).

Note that the Timeout option can also be set as a PROC APPSRV option.

2) Administrator name and e-mail address

In case of an error this name and e-mail address will appear so that an end-user can alert you. This data can easily be altered by changing the lines that read:

```
Administrator "Kevin Davidson"  
AdministratorMail "KevinD@FSD.nu"
```

3) Field width of text area blocks

Any text area fields that you may set up in your HTML are chopped up into character block fields according to an option set in the configuration file. The default used to be 80 and although the SAS supplied comments in the 8.2 configuration file still indicate that the default is 80, it does not appear to be so (at least under Windows). Suppose your HTML page has an input field such as the following:

```
<textarea name=comment cols=80 rows=4 wrap=virtual>  
This is a pretty long text field. We want to see where it  
wraps. The zero in 80 is the eightieth character in this  
line. We want to see if it splits it given the default length  
that is specified in the broker.cfg file.  
</textarea>
```

There is an option in the broker.cfg called 'Fieldwidth'. If the Fieldwidth is set to 80, then upon submitting the above text to the Application Dispatcher, the symbols made available to SAS will look like the following:

```
comment=This is a pretty long text field. We want to see  
where it wraps. The zero in  
comment0=3  
comment1=This is a pretty long text field. We want to  
see where it wraps. The zero in  
comment2=80 is the eightieth character in this line. We  
want to see if it splits it  
comment3=given the default length that is specified in the  
broker.cfg file.
```

As you can see, the longer string is cut up into smaller fields of roughly 80 characters each. The 1st field listed above is the original input field name. The 'comment0' value of 3 indicates that the original variable was split into three different blocks. The three different blocks are represented by the numeric suffix being added to the original input field name.

You can alter this setting by entering a value such as the following:

```
Fieldwidth 32767
```

This will allow text area strings to be exported to SAS as one long character string (you will of course want to make use of the 'compress' data set option if you create extremely long character variables in your data set).

4) Exporting fields

If you examine the broker.cfg file, there is a series of potential export variables which you can make available to your SAS session (they will appear in the 'SYMBOLS' section of your logs). In version 8.2, the list of available export variables are as follows: Simply uncomment those that you want to make use of.

What is it?	Export variable name
CGI version	Export GATEWAY_INTERFACE GATEWAY
Web server hostname	Export SERVER_NAME SRVNAME
Web server name/version	Export SERVER_SOFTWARE SRVSOFT
HTTP version	Export SERVER_PROTOCOL SRVPROT
Web server port number	Export SERVER_PORT SRVPORT
GET, POST, etc.	Export REQUEST_METHOD REQMETH
Extra path info after script	Export PATH_INFO _PATHINF
Local filename of PATH INFO	Export PATH_TRANSLATED PATHTRN
Dup of _URL	Export SCRIPT_NAME SCRIPT
Directory from which Web documents are served [unreliable]	Export DOCUMENT_ROOT _DOCROOT
Query string for GET requests [duplicate of user macro parms]	Export QUERY_STRING _QRYSTR
User's DNS name if known	Export REMOTE_HOST RMTHOST
User's IP address	Export REMOTE_ADDR RMTADDR
Usually Basic	Export AUTH_TYPE AUTHTYP
Username if authenticated	Export REMOTE_USER RMTUSER
RFC931 id if supported	Export REMOTE_IDENT RMTID
HTTP POST type	Export CONTENT_TYPE CONTTYP
HTTP POST length	Export CONTENT_LENGTH CONTLEN
Email address of user making request [unreliable]	Export HTTP_FROM _HTFROM
MIME types UA likes [unreliable]	Export HTTP_ACCEPT _HTACPT
Cookies	Export HTTP_COOKIE HTCOOK
Browser name	Export HTTP_USER_AGENT HTUA
Referring page if known	Export HTTP_REFERER HTREFER

As an example, if we have the "_HTUA" set to export, then your SAS programs will be able to make use of the variable and you can also automatically store the value in the STATISTICS data set (see below). Near the top of your log, you would see something similar to:

```
Thu, 31 May 2001 11:18:23.687 Request 12 Program is
proglib.myprogram.sas
Thu, 31 May 2001 11:18:53.858 Request 12 ended okay
(30.50 seconds)
===== Request 12 Symbols =====
  _SRVNAME=www.fsddatasvc.com
  _RMTADDR=216.91.105.82
  _HTUA=Mozilla/4.0 (compatible; MSIE 5.5; Windows
NT 4.0)
```

5) Creating fields using a SET statement

In addition to the SAS supplied export variables, you can also create your own. For example, if you have multiple web servers running Application Dispatcher(s), you may want to display in the log (and/or in the STATISTICS data set) the name of the server. If you add a line to your configuration file such as:

Set machinename kevinspc

This statement will display the variable 'MACHINENAME' in the log with a value of 'kevinspc'.

6) Load Manager

The load manager is not required for socket services but is recommended. It is required for the use of pool services. It is a good idea to run the load manager so that some software intelligence is used to direct requests to an inactive port, rather than being shipped to a port that is already active and thus put into a queue. Another benefit is that if all dispatchers are busy, then the load manager will hold the request until a dispatcher is open. In the case of pool services, it will launch a new dispatcher. Once the load manager (an .exe file) is loaded, there is simply one line within the broker.cfg file which activates it:

LoadManager www.fsddatasvc.com:6301

PROC APPSRV

This procedure invokes the application dispatcher and contains a number of customization options which can be used to your benefit. The procedure can be run within a larger SAS program if you want to create macros and/or set macro variables. At our company, we create a file

called 'appstart.sas' which contains the PROC APPSRV procedure. Here are a few of the options and statements that you might find beneficial.

1) ADMINPW

The Adminpw=xxxxx. Allows you to run admin programs and restricts others from being able to shut down your dispatchers. There are several SAS supplied administrative programs such as PING, STATUS, and STOP (you can create your own as well). If you set up an administrative password, then these administrative programs cannot be run unless the password is provided. An example of how to implement this option is something like this:

```
proc appsrv port=6013
adminpw='MYADMINPASSWORD';
```

2) UNSAFE

The unsafe option strips unwanted characters from fields which end users submit. The most often stripped characters are ampersands, semicolons, and quotes. Use this with caution as you might be surprised where these values are needed.

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

There is an appsrv_unsafe function that allows you to pull the original value even if the unsafe option is utilized.

3) Short macro for simplifying the ALLOCATE, DATALIBS, and PROGLIBS statements.

If your applications dispatcher references a number of libraries, your PROC APPSRV can get quite long and cumbersome. Below is a short macro which can simplify the process.

```
%let drive_direct=f;
%let drive_share=c;;
```

```
%macro libfile(ref,path,server);
  allocate library d&ref "&drive_direct\&path";
  allocate library s&ref "&drive_share\&path" &server;
  allocate file p&ref "&drive_direct\&path";
  datalibs d&ref s&ref p&ref;
  proglibs p&ref;
%mend;
```

```
proc appsrv port=6013
adminpw='MYADMINPASSWORD';
```

```
%libfile(mylib,\mydir,%str(server=servname.myserver));
```

```
%libfile(mylib2,\mydir2,%str(server=servname.myserver)
);
```

4) REQUEST statement

If you wish to run a program either before or after each requested program is run, you can specify the REQUEST statement with either or both of the INIT and TERM arguments. An instance of where this might be useful is if you want to initialize a set of macro variables or run an options statement to apply to all requests. If you are using compiled macros across multiple dispatchers, you need to point the program to them using an options statement to avoid a lock being put on the catalog. To do so, you need to submit an options statement with the 'mstored' and 'sasmstore' options specified. You can then store that options statement in a .sas program (e.g. myinit.sas) and then have it submitted before each request with a statement such as:

```
request init=mylib.myinit.sas;
```

5) STATISTICS Statement

The STATISTICS statement is an extremely useful to for collecting statistics on individual requests made to your dispatchers. By default, the data set is set up to collect the following pieces of information:

Variable Name	Variable Type	Description
Obstype	Character length 1	R = request, I = Internal, U = startup, D = shutdown, T = trace
Okay	Character length 1	1 = request ran okay, 0 = error
Duplex	Character length 1	H = half duplex, F = full duplex
Http	Character length 1	1 = http request, 0 = normal broker request
Program	Character length 32	_PROGRAM variable
Peeraddr	Character length 16	Peer address
Hostname	Character length 20	Node name of the server
Username	Character length 12	_USERNAME variable, if any
Entry	Character length 32	_ENTRY variable, if any
Sessionid	Character length 12	_SESSIONID, if any
Service	Character length 12	Service name
Starttime	Number	Time the request started
Runtime	Number	Run time of the request
Port	Number	Server port number

Bytesin	Number	Number of input bytes (read from client)
Bytesout	Number	Number of output bytes (written to client)

As long as the variable is available when the request is made (either by having set it to be exported or creating your own variable), the variable gets added to the data set. There are options that you can set to control the frequency with which the request queue gets written to the data set. You can easily add or drop variables from the data set just as you would any other data set (assuming of course that the application dispatcher is closed as it puts a lock on the data set). You can use SAS/SHARE software to so that multiple dispatchers can write to a single data set. The following statement is a sample usage of the STATISTICS statement:

```
statistics data=mylib.mystats (DLDMGACTION=repair);
```

Note that the data set option is a useful option that we have found prevents manual intervention when the data set is damaged due to an unfortunate shutdown. One of the great utilities of this data set is that you can then write SAS utility programs as needed to analyze the data on a real time basis. For example, you can create reports to show you which programs are running slowly, what the frequency of the various programs is, what the usage by hour and by day of the week is, etc.

- 6) Using aggregate syntax (member name syntax) with %include

Note that since you are essentially setting up filerefs within your PROC APPSRV statement, you can utilize aggregate syntax for use with %include. This saves you the trouble of having to include the path name. For example, to %include a file named 'myfile.sas' you would simply use the following code:

```
%include mydir(myfile);
```

- 7) LOG statement

The LOG statement gives you some control over what gets written to your application dispatcher log as well as the naming of the log. If you are not using Web Hound and wish to be able to track happenings via the logs, it is highly suggested that you display as much information as possible. To do so, specify the following:

```
allocate file logfile "d:\mydir\%nrstr(%m%d%y)_1.log";
log display=all symbols=all file=logfile;
```

The 'display=all' option tells the dispatcher to write the log information from all requests to the log file.

'Display=error' would write the log information only when the request ended in an error. 'Display=none' would never write any log data. The same three options (all, error, and none) are available for the symbols argument which controls the listing of the client request passed values within the log. The syntax shown in the example above creates a file with a name of "mmddyy_1.log" where mmddyy is month, day and year. There are numerous other options available for naming the log file.

Running your SAS programs:

Whenever the application dispatcher is summoned, one of the pieces of information that it must receive is something that tells it what to do. This piece of information is in the form of an input field with a reserved name of '_program'. This information can be one of 4 types of programs:

You can run 4 different types of SAS programs. Three of the types reside in catalogs.

- 1) SAS programs - _program=*library.program.sas*

Pros – easily maintained
Cons – relatively slow as the code needs to be compiled
Generates larger dispatcher log files

- 2) SCL entries - program=*library.catalog.program.scl*

Pros – precompiled code
Cons - SCL code has its origins in the full-screen environment and thus much existing SCL code needs modification to run stand-alone.
Source code is not stored with the compiled macro

- 3) MACRO entries - compiled macros
_program=*library.catalog.program.macro*

Pros – precompiled code
Cons – Can be difficult to maintain as must compile them and copy them to a catalog
Source code is not stored with the compiled macro.

- 4) SOURCE entries - .sas programs stuck in a catalog with a different extension
_program=*library.catalog.program.source*

Pros – fairly easy to maintain although not as easy as running SAS programs
Cons - relatively slow as the code needs to be compiled
Generates larger dispatcher log files

CONCLUSION

Determining how you set up your Application Dispatcher within SAS/IntrNet can be a daunting task. As usual with SAS software, there are many ways to get things done. The administrator and programmer are given a wide range of options. A little forethought into what type of statistics you want to gather, how much security you want or need, and how you plan to maintain your SAS/IntrNet web site will go a long way in helping you make decisions that you are comfortable with for the long haul.

Kevin Davidson, Ph.D.
FSD Data Services, Inc.
2020 Southwest Freeway, Suite 206
Houston, Texas 77098
713-942-8436
kevind@fsd.nu
www.fsd.nu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.