Paper 39-27

A Look at the Development Process for a SAS/IntrNet® Application

Andrew Rosenbaum, Venturi Technology Partners, Kalamazoo, MI

ABSTRACT

Many sources of information are available to assist a developer learning the basics of Web development with SAS/IntrNet. However, once the developer becomes proficient with the basics, little information exists of how to put an application together. The SAS/IntrNet developer must know what questions to ask and what choices need to be made in order to construct an efficient and effective SAS/IntrNet application.

INTRODUCTION

Developing an application that is intended to be deployed on the Internet requires a different approach than one intended for a local network. A developer familiar with SAS/AF® Frames must learn to get along without his Frame object and the close association between Frames and SCL. He must contend with the limited HTML visual 'objects' and also learn to think of an application as a series of batch programs instead of one program.

As the developer makes the transition from conventional application development to SAS/IntrNet development, he will come across many issues that are unique to Web-based development. How should the application be written? DATA steps or SCL? What services should be created? Launch, socket, or pool? Should SAS Sessions be employed? What kind and level of security is needed? What browser will be used? These are some of the questions that will be discussed. There are other issues involving hardware, business concerns, and more. However, the scope of this paper will be limited to applications intended to run with the use of the Application Dispatcher over an intranet or the Internet. There are SAS/IntrNet products other than Application Dispatcher but issues involving those products will not be discussed in this paper.

The specifics in this paper refer to the SAS® System version 8.2 running in a Windows environment. Many of the issues discussed will be the same for other versions and operating systems. It is strongly recommended that if a different version or operating system is used that the information in this paper is verified for that version or operating system.

DATA STEP OR SCL?

SAS programs can be written in one of two methods:

- Base SAS (DATA steps, procedures, macros)
- SAS Component Language (SCL)

For an Internet application, which is better? While one is not necessarily better than the other is, the main question is with which one is the developer more comfortable. Although a developer should use the method he is most familiar with, the SCL option offers several advantages.

SCL IS COMPILED

SCL code is compiled once by the developer and it can be run any number of times. DATA steps are compiled every time they are run. Since SCL is compiled only once, this makes SCL programs more efficient. There are also security advantages when using SCL. (More on this in the SECURITY section).

ORGANIZATION

SCL entries lend themselves to organization. For example, if there are 20 SCL entries, they can all be placed within one catalog. If DATA steps are being used and there are 20 programs, then 20 separate files are required. Again, SCL offers an advantage over DATA steps. (However, this is not a significant advantage because catalog entries are possible with base SAS by placing code in SOURCE entries).

HTML

It is very simple to write HTML code using SUBMIT blocks. (See "Using HTML in SUBMIT blocks") This technique allows the developer to write HTML code without using PUT statements which is a tedious method and one that results in code that is not easily readable. A static HTML program can be written and tested outside of the SAS environment and then copied into a SUBMIT block without any editing changes, saving significant time.

OBJECT-ORIENTED PROGRAMMING (OOP)

The most significant advantage to SCL is that it allows the use of Object-Oriented Programming. OOP has many documented advantages over standard programming. These advantages become more pronounced as the size and complexity of the application increase.

The main disadvantage to using SCL is that non-SCL programmers cannot easily maintain the application. If it is going to be used in an organization that has no SCL programmers, then base SAS may be the better choice simply for ease of maintenance. Of course, in this situation, job security would be ensured for the SCL programmer by using SCL.

DEBUGGING

CODES

Debugging a SAS/IntrNet program can be a serious challenge. When things go wrong, sometimes there is not a coherent error message to offer clues. To aid the debugging process, codes are available to direct the SAS system to send information back to the browser. The debugging codes are simply numbers that are included in the URL (i.e. _DEBUG=128 sends a copy of the log to the browser). These codes can be used in conjunction with each other by simply adding the numbers of the debug functions that are needed and using the sum in the URL.

OPTIONS

When using debug codes to view information, the SAS system option SOURCE must be set. If NOSOURCE is set, then base SAS code will not appear in the log. Additionally, if macro language components are used, MPRINT, MLOGIC, SYMBOLGEN, and MERROR may be useful.

After a program has been thoroughly tested and debugged, it is advisable for security reasons to restrict the use of DEBUG codes to keep the SAS code from being seen by the user. This is accomplished by using debug masks in the BROKER.CFG file.

NOTE: The SAS system option NOSOURCE must be set so that the source code is not viewable in the log.

LOGS

Even without the use of DEBUG codes, two logs are written out to files during each session. They are sent to the same folder that contains the APPSTART.SAS file for the specified service. These logs contain information regarding the SAS session that is run on the server. These logs sometimes contain useful information but in general are not very helpful for program debugging.

A developer can use the ALTLOG function to direct a copy of the SAS log to a file. This is a benefit because it allows the developer to see program information and errors without cluttering up the browser window with extraneous information.

There are also the usual techniques such as the use of PUT and CALL PUTLIST statements within the program code. However, these must be used in conjunction with the appropriate DEBUG code in order to see the values in the browser.

DEBUGGERS

The DATA step debugger and the SCL debugger are also available for debugging an Internet application. To use the DATA step debugger, the Application Server is started in full-screen mode. Then `/debug` is added to the DATA step that is to be debugged.

```
   DATA WORK.DEAN / DEBUG;
      SET WORK.RICK;
      . . .;
   RUN;
```

The program is run and when it gets to this DATA step, the debugger window will open and the DATA step can be stepped through one line at a time.

To use the SCL debugger, the Application Server is started with the option:

```
   AFPARMS='DEBUG=YES'
```

Also, the SCL source code must be compiled with the DEBUG option turned on.

DEBUGGING PROCESS

Sometimes errors occur before the program even starts to run. The problem is somewhere in the system that calls the SAS program. When this happens, it is time to start at the beginning and systematically determine where the problem lies. First, verify that the Web server is running. Then the following command is entered in the browser's command line. (*localhost* refers to the name of the Web server)

```
   //localhost
```

If the server is running, there should be some kind of indication that a Web server was found. While using Microsoft Personal Web Server, there are two possibilities; a listing of files from the home directory or a page entitled "Welcome to Microsoft Personal Web Server will be seen. If the Web server is not running, there will be some kind of error indicating that the page could not be found. If using the Microsoft Personal Web Server, then a search page may appear.

If the Web server is running, then the next check will be to see if the path to the Application Broker is correct. The following is entered onto the browser's command line: (`cgi-bin` may be different on your system. It is the name of the directory containing the Broker executable file.)

```
   //localhost/cgi-bin
```

If this is correct, then a listing of files in that directory will appear (if the server allows browsing of the directory). If the listing does not appear, then there is something wrong with the path to the directory containing the Broker file.

Next, the Broker itself can be tested. The following is entered into the browser's command line:

```
   //localhost/cgi-bin/broker.exe
```

If correct, then a page will appear that says, "SAS/IntrNet Application Dispatcher". There are three links below it. One is for Application Dispatcher administration, the next one is a link to samples of SAS/IntrNet programs, and the last one is a link to SAS/IntrNet documentation. If this page does not appear, then there is something wrong with the path to the Broker file.

Next, the Application Server is checked by specifying a service:

```
//localhost/cgi-bin/broker.exe?_service=default
```

If correct, then a page will appear that says, "Application Error. The required field _PROGRAM was not specified. Please retry." Although this is an error message, it means that the Application Server is running and everything is normal. If the message "Error connecting to the SAS server" appears, then there is something wrong with the service configuration, or that the Application Server has not been started.

Finally, the Application Dispatcher is checked by using the PING test program:

```
//localhost/cgi-
bin/broker.exe?_service=default&_program=ping
```

If correct, then the Application Server is functioning correctly. If not, then the message will be similar to the one for testing the service.

If these tests are successful and an application still will not start, then the libnames should be checked to verify that they are properly set in the APPSTART.SAS file and that the appropriate SAS programs and data are in their proper place.

SERVICES

Setting up the application's environment involves deciding which service to use. The main criterion is user load. How many users will be active at one time? Will more users be added to the system? What kind of hardware is available for high user demand? These are some of the questions that need to be asked in order to select the optimal service.

The three services (launch, socket, and pool) have their own advantages and disadvantages. A brief discussion of each service follows. Further information is available in "Choosing a Service Type" in the SAS/IntrNet documentation.

The launch service starts a SAS session upon request from a user. This session is referred to as the Application Server. After the application has processed the request, the SAS session is shut down. The launch service is an appropriate choice if response time is not a big concern and if there will not be many users on the system. Since the SAS session is launched only when a user accesses the application, the service does not tie up resources when it is not in use. There is also no need to reboot the service if the Web server goes down. The launch service can be used on all platforms except CMS, OpenVMS, and OS/390 operating systems.

In order to reduce the overhead involved with the starting of a SAS session, the socket service can be used. A SAS session is started and sits idle until requests are made. After processing a request, either the session processes the next request or waits for a new request. It is also the only service that works on all platforms. The socket service provides a better response time since the overhead of starting a SAS session has already been done.

One of the disadvantages is that if the server that is hosting the socket service stops working, then the socket service might need manual restarting. Depending on the operating system, this could be an automatic function. Another downside to the socket service is that if there are too many simultaneous users, the response time will suffer.

This leaves the pool service to use if there will be heavy demands on the server. The pool service is used in conjunction with the Load Manager to handle potentially heavy demands. The Load Manager will start up a service if there are many users and will shut down services if there is little or no demand on the server. This is the best service choice to make if a heavy demand is expected on the application. However, the pool service and the Load Manager are more difficult to configure.

For the beginner, it is advisable to use the socket or launch service for development purposes. For an application deployed to a Web server, the socket service will handle a modest number of requests with very little work involved in configuring it.

PERSISTENCE

One of the inherent problems of Internet application development is the lack of persistence. The application does not keep any information between users' requests. For example, a user sends a request that includes the user's name. The application processes and returns the results to the browser. When the same user makes another request, he must send his name again. The application has deleted all information from the first request. All variables, data sets, and SCL lists are gone.

The use of sessions will solve this problem. A session allows the application to save data between a user's requests. The application assigns a session ID to each user so that when requests come in from more than one user, the application can distinguish among them. Each session has its own storage space so that each user can save their data and keep it private.

A session is started with the APPSRV procedure.

```
   rc=appsrv_session('create');
```

A session ID is created and is used to keep track of which user is making a request.

A session is automatically deleted after the amount of idle time equals the timeout setting. The session can also be deleted manually.

Once a session has been established, data can persist between users' requests. For example, a user sends in a request and sends his first name as one of the parameters. The application can save the name in a macro variable that will persist as long as the session exists. Now, when the user makes a second request his name does not need to be sent again. The application can read it from the saved macro variable.

To save a value in a persistent macro variable the developer must precede the macro variable name with SAVE_.
Although the following example is very simplistic, it serves to demonstrate the technique. The SAS/IntrNet developer will find many uses for this function.

```
   call symput('SAVE_NAME','CHUCK')
```

This statement will store the value CHUCK in the macro variable SAVE_NAME. That macro variable will exist as long as the session exists and will be available anywhere and anytime in the application.

```
   call symput('NAME','CHARLES')
```

This statement will store CHARLES in the macro variable NAME. This value will exist as long as the current request is being processed. As soon as the request has been processed, the macro variable will be deleted and will no longer be available.

When a session is created a SAS library called SAVE is created. This library will persist between requests and will only be deleted when the session ends.

To save a work data set, the developer copies it to the SAVE library:

```
   DATA SAVE.FRED;
      SET WORK.FRED;
   RUN;
```

An SCL list (as well as any other type of catalog entry) can be saved to the SAVE library also.

```
rc=savelist('catalog','save.list.names.slist',namelst);
```

The ability to save data between requests can be a huge factor in making an application more efficient. Data sets and SCL lists need only be created once per session and not regenerated for every request.

USING OOP WITH SAS/INTRNET

The application developer will want to consider seriously the benefits of Object-Oriented programming techniques for a SAS/IntrNet application. There are many advantages to using OOP in SCL programming and since there are many similarities among SAS/IntrNet programs, a library of reusable methods will come in handy.

A start for the library would be three basic classes: PAGE, GRAPH, and DATA.

The PAGE class would be created as the parent to contain the common methods that one would need to create Web pages. One of the methods would be called HEADER. This method would be called before any page is created. It would be responsible for writing out code that is common to all of the pages in an application. For example, if a logo or a menu were needed at the top of each page. The same would be true for a method called FOOTER. Any code that is common to the end of each page would be written out by this method. The code that is necessary to write out the HTML code from the preview buffer to the _WEBOUT filename would also be in the FOOTER method.

The DATA class would store methods that allow a program to retrieve and manipulate data. Methods such as GetVars would get a list of the variables from a data set. GetValues would retrieve all of the values for a specified variable from a specified data set.

The GRAPH class could contain all of the code necessary to create a graph using PROC GPLOT. It would also allow GOPTIONS to be set using methods. For example, the device would be set with the SetDevice method. The type of graph would be specified with the SetGraphType method. A hierarchy of graph type could be created using the GRAPH class as the parent. BARCHART would be a child of GRAPH and VERTICAL BARCHART and HORIZONTALBARCHART would be children of BARCHART.

BROWSER ISSUES

SAS/IntrNet is designed to work with the two most popular browsers, Netscape Navigator and Internet Explorer. If the developer is not sure which browser will be used to access the application, then thorough testing of the application in both browsers is imperative. It is also advised that different versions of the browsers be used for testing, not just the most recent version.

If an application needs to accomplish a task and the syntax for that task is different for these two browsers, then it is possible to determine at runtime which browser is in use so that the appropriate code can be called.

One significant difference between the two browsers is that the ActiveX device driver can only be used with Internet Explorer browsers running on a Windows platform.

SECURITY

The Internet application development process is complicated by the need for security to keep unauthorized users from gaining access to private data as well as keeping malicious individuals from doing their evil deeds.

Several security measures need to be addressed before starting a SAS/IntrNet project. The level of security needed is obviously dependent upon the potential threat. An application running on an internal intranet with trusted users will need far less security than one that is exposed to the Internet.

Some security measures may be addressed by systems personnel and not the developer. Therefore, only a brief mention will be made of those functions.

CODE

As mentioned earlier, SCL code is compiled. Therefore, the source code never appears in the log as it does with base SAS code. Once compiled, the SCL source code can be deleted from the Application Server so that if someone did gain access to the files, only the compiled code would be visible and not the readable source code.

FIREWALLS

There are different configurations for using firewalls with SAS/IntrNet's Application Dispatcher. The first option is to have the firewall placed between the browser and the Web server. Using this method, the firewall is easy to configure and the SAS/IntrNet application has no special requirements. However, it is not always a good idea to have the Web server inside the firewall because now it has access to the entire network. If someone were able to breach the Web server's security, the entire application and its data would be in jeopardy.

Another option is to have the firewall placed between the Web server and the Application Server. This configuration is more difficult to configure but it keeps the Web server outside the firewall so that it is not exposed to the entire network. Some care must be taken by the developer to protect data sources from unauthorized access. This is because the Application Server is exposed to the client. It is possible to access the Application Server directly, bypassing the Broker, and accessing data sources. Password protection and/or access control is necessary to protect data sources in this configuration.

The third firewall configuration is to have the firewall between the Application Server and the internal network. In this situation, the firewall is easy to configure and the internal network is not exposed to the Web server. The problem with this setup is that data on the internal network must be copied to an area outside of the firewall in order for the application to use it. SAS/SHARE® can be configured to allow the Application Server to access data directly.

WEB SERVER SECURITY

Using a secure Web server ensures that information passed between the Web server and the browser is not subject to unauthorized access.

APPLICATION BROKER SECURITY

Since the Broker configuration file has information that unauthorized users should not see, it is important to protect the Broker configuration file so that it cannot be read or changed. The access level on the file is set for users to read-only. If possible, the access level is set so that it can only be executed and not edited or browsed.

Another measure that can be taken to protect the Broker configuration file is to set the DebugMask so that `_debug=4` is not possible. This is important because `_debug=4` allows a user to see the contents of the Broker configuration file.

APPLICATION SERVER SECURITY

It is possible for someone running his or her own Broker to access your Application Server. Therefore, in order to protect your application from this kind of threat, it is necessary to ensure that the Broker connecting with your Application Server is your Broker. This is accomplished by setting the ServiceSet parameter in the service definition to a secret value.

```
SocketService Socket1 "Application Service"
 server server.company.com
 port 5001
 ServiceSet confirm "secret value goes here"
```

This value must be checked in the application. The value "confirm" that is passed to the program must be equal to "secret value goes here". If it is, then the Broker that is connecting to the Application Server is the correct one. There are two points to take note of when using this method: First, the `_debug=1` must be disabled. This would allow a user to see the value of "confirm". Second, the source code for the application must be secure because the secret value of "confirm" is hard-coded there.

Another security measure that should be considered is to password protect the Application Server. If the server is not password protected then anyone could issue the STOP command to halt the server.

UTILITIES

Several tools are included with the SAS/IntrNet software that can help with some routine activities.

HTML FORMATTING TOOLS

These tools make it simple to display data or output to the browser. A few of them are:

- HTML Data Set Formatter
  Converts SAS data sets into HTML

- HTML Output Formatter
  Converts procedure output into HTML

- HTML Tabulate Formatter
  Converts Proc Tabulate output into HTML

- GraphApplet HTML Generator
  Generates graphs from SAS data and using the GraphApplet or SAS/GRAPH Control for ActiveX the graphics can be displayed.

APPLICATION UTILITIES

These utilities serve to perform data set conversions, serve files, and encode and decode character strings.

- DS2CSV - converts SAS data sets to comma-separated variable files

- FILESRV - controls which files are served and also controls the HTTP and MIME headers that are served with a file

- HTMLDECODE - decodes a string that contains HTML character entity references or numeric character references

- HTMLENCODE - uses character entity references to encode characters that would normally be interpreted as HTML syntax

- URLDECODE - decodes a string that is encoded with URL escape syntax

- URLENCODE - encodes a string using URL escape syntax

Utilizing the tools that the fine folks at SAS Institute have given us is an excellent way to save time when creating an application.

JAVASCRIPT

JavaScript (no relation to the Java language) is used when executable code is needed in a Web page. For example, if there is a required field on an HTML form, with JavaScript, field validation can be carried out by the client instead of submitting the page and having SAS code do the validation. By running such tasks on the client, a great deal of processing time can be avoided.

In order to learn JavaScript, it is necessary to go to a JavaScript Web site, get a book, or attend a class. There is nothing in the SAS documentation about JavaScript.

USING HTML IN SUBMIT BLOCKS

If the developer is using base SAS to create a SAS/IntrNet application, then the main method of writing HTML code to the browser is:

```
DATA _NULL_;
   FILE _WEBOUT;
   PUT "<HTML>";
   PUT "HELLO WORLD!";
   PUT "</HTML>";
RUN;
```

(NOTE: FPUT can also be used.)

However, when using SCL two methods may be employed. The first is similar to the base SAS method:

```
file_id=fopen('_webout','o');
rc=fput(file_id, "<HTML>");
rc=fput(file_id, "HELLO WORLD!");
rc=fput(file_id, "</HTML>");
rc=fwrite(file_id);
rc=fclose(file_id);
```

The second (and far superior) method is to use the preview buffer. This is accomplished by putting the HTML code in SUBMIT blocks. As long the keyword CONTINUE is not used, then the HTML code stays in the preview buffer and is not sent to the base SAS processor. When all of the necessary HTML code has been accumulated the code is sent to _WEBOUT.

```
SUBMIT;
   <HTML>
      HELLO WORLD!
   </HTML>
ENDSUBMIT;

RC=PREVIEW('FILE','_WEBOUT');
RC=PREVIEW('CLEAR');
```

The advantages here are significant. First, no PUT statements and quotes are used. Second, a static Web page can be developed and tested outside of the SAS environment and then simply copied into an SCL entry. This is not possible when using PUT statements.

There is one note of caution that must be observed when using this method. Base SAS code cannot be interspersed with HTML code. For example,

```
SUBMIT;
   <HTML>
     HELLO
ENDSUBMIT;

SUBMIT CONTINUE;
   PROC SORT DATA=WORK.FRED;
      BY NAME;
   RUN;
ENDSUBMIT;


SUBMIT;
      WORLD
   </HTML>
ENDSUBMIT;

RC=PREVIEW('FILE','_WEBOUT');
RC=PREVIEW('CLEAR');
```

This will result in an error since the base SAS processor will try to process the HTML code that is in the preview buffer when it is submitted by the CONTINUE command. HTML SUBMIT blocks must be separated from SAS code SUBMIT blocks. The SAS blocks must be processed before the HTML blocks.

CONCLUSION

This paper has discussed some of the issues that the application developer must face before creating a SAS/IntrNet application. A paradigm shift is required to enable a program to become a thin-client application. Some research, thought, and trial and error on the part of the applications developer will be necessary in order to create a successful, robust, and easy-to-use application.

ACKNOWLEDGMENTS

Many thanks to Dean Clous, Charles Kincaid, Rick Schmalzried, Charles Bininger, and Susan Stejskal for their inspiration and valuable assistance.

CONTACT INFORMATION

Contact the author:

Andrew Rosenbaum
Venturi Technology Partners
A division of Personnel Group of America
5278 Lovers Lane
Kalamazoo, MI 49002
Telephone: 616.344.4100
Email: arosenbaum@venturipartners.com
Web: www.venturipartners.com