Client/Server Application Design Strategies for Small Development Teams

Ray L Ransom and Sharon Mosley-Hixon, Centers for Disease Control and Prevention

Abstract

This paper will describe and discuss a real world application development experience. It is meant to provide guidance as well as elicit discussion. As with most development teams, it was much easier to list our resources than the vast needs of our target audience. We will discuss here what our needs were and how our constraints influenced the design and contents of the application. As with any endeavor with the SAS System®, this is but one of many ways in which we could have proceeded. This paper will attempt to document our options and why we chose the routes we have.

Introduction

Our development team qualifies as small. We do not have to quote Merriam-Webster or provide relative comparisons to make you believe us. As far as we know, a team must be made up of more than one person. With that one definition out of the way, we can say our development team, consisting only of the two authors listed above, is as small as they get.

Our needs are seemingly infinite. With computers becoming more common place in everyday life, computer phobia is beginning to give way to information dependency. Supporting researchers and managers through traditional programming and reporting methods is becoming impossible. Application development is a necessity for routine operation and decision making and is becoming more viable as a solution for data exploration and research.

If an organization's director has a programming staff of several SAS programmers, why would she consider hiring more staff for applications development? Why not just send them to a few courses and pick someone to be project manager? Although this may seem absurd to some, this really is the more likely approach that many of us are finding ourselves facing.

Ray Ransom has been programming with the SAS System for 10 years. He has never managed a project before this one and has no formal training in applications development, object-oriented design or systems architecture. Sharon Hixon has only used SAS for 2 years. This includes the year and a half since we started this project. Sharon has a very diverse background in support and training of many PC applications. Neither of us work on this project full time.

Needless to say, our jobs are not boring, but we also hope to demonstrate here that our goals are not insurmountable. The key to our success has been in design. We tempered every decision with an analysis of our needs, a research of the possibilities and the realities of our limitations. We hope that you benefit from our experience.

Background

The Division of Sexually Transmitted Disease Prevention (DSTDP) of the Centers for Disease Control and Prevention (CDC) collects data on occurrence of STD's and disseminates this information throughout the world public health community. The division also conducts, supports and funds millions of dollars in prevention and treatment activities including education, research, guidelines development and clinical care. These activities produce and are guided by data. Statistical and data management staff within the division are responsible for the reception, maintenance and analysis of these data.

The division has several components. There are branches dedicated to epidemiology and surveillance, STD program support, clinical and field staff training, behavioral research, health services evaluation, international health activities, and statistics and data management. The data needs of each of these branches share many similarities, but also are very diverse.

DSTDP datasets range in size from large files of weekly reports of all identified cases of certain STD's to smaller files of clinical trial data. Storage of these files is even more dispersed. Large surveillance data files are stored on an IBM MVS mainframe and smaller files are on a Novell LAN, local hard drives or even diskettes. Documentation of these files and associated programs is currently dependant on the organizational skills of their owner.

In the Fall of 1995, Ray Ransom decided to see if he could get funding to improve the division's data management strategy. With the assistance of other data managers in the division, he proceeded to write a proposal for funding that was to start the process. Two years later, our data warehousing effort is coming to fruition and the SAS/AF application front-end, STDINFO, is nearing beta test. Now, let us tell the story of how this process evolved. Hopefully this undertaking will be of use to others and open channels of communication through which we can share information that will benefit other small development teams as well as our own.

Planning

It was apparent that our needs required two related, but separate solutions. First of all, we needed a

system for identifying, organizing and cataloging our data. The completion of this task would require skills that were primarily organizational and not application development skills. The planning of how to accomplish this, however, required considerable thought and design. The solution for this task was the development of a data warehouse. I am not sure we could all agree on a definition of a data warehouse, but for this paper, our data warehouse will be little more than systematic storage of our division's data files and related documentation coupled with the process of normalizing existing data and defining standards for future development of data collection tools or database structures.

The next required solution was an application front-end to the data warehouse. This would also be a place where we could provide standard analysis tools that are inherent to analysis of public health data and STD program evaluation. These tools might include logistic regression, ROC curves, categorical modeling and source/spread analyses. SAS already offers the ability to perform these tasks, but considerable knowledge of programming is required to achieve them. We decided to name our application STDINFO.

Our next step was to create an advisory group to help guide us in our planning. We were careful to select people from each branch of the division and of varying professions. After explaining our plan so far, this group assisted us in deciding what functionality we should offer first in the application. The greatest interest seemed to be in providing a common interface to data files. A Windows interface seemed to be the preference. Another strong interest in the division was providing current surveillance data to decision makers and researchers in a timely manner. Currently, access to these files required interaction with a division data manager.

Since there were only 2 of us, we decided that whatever we created, it had to be stable, require minimal maintenance, and require little training. We will mention these qualities several times in the sections ahead. We attended all SAS/AF related courses and purchased the entire SAS documentation library.

Selecting Hardware

Prior to the onset of the project, the division's resources consisted of 386, 486 and Pentium PC's on a Novell 16MB token ring network and CDC's IBM MVS mainframe computer. SAS could be run interactively or in batch on the mainframe or by remote submitting from a SAS/CONNECT session from SAS 6.12 for Windows. Since our project was only funded for 2 years at $75,000 per year, we decided to make all hardware purchases up front.

Because of unacceptable job queuing times at peak hours and the non-graphical display of our mainframe environment, a midrange processing server was considered for STDINFO for more cpu intensive processing and larger surveillance tables. Both NT and UNIX were considered. As presented by Keith Humphrey at SUGI 22 (Humphrey 1997), we found that although the NT operating system posted impressive benchmark times against UNIX, UNIX's true multiprocessing and multithreading architecture surpassed NT performance as more concurrent processes were added. Since our application was planned to be an integral part of the division's daily operations, we needed the ability to provide concurrent access for several users.

After purchasing a SUN Ultra Enterprise (ES3000) server with 2 processors and 256MB RAM, we proceeded to evaluate a beta-test version of the SAS Scalable Performance Data Server (SPDS). SPDS is the result of a joint venture between SAS Institute and SUN Microsystems. SPDS is a software server that enables the SAS System to take advantage of the SUN multi-threading environment. As promised, this server results in 6 times improvement in processing and querying times on data managed by this server. In order to maximize the performance of the server, data files should be indexed on all key variables and WHERE clause and SQL programming should be used. The process of reviewing our data storage and programming conventions resulted in considerable performance improvements without the server. The combination of the high-end hardware and smarter data access conventions allowed us to enjoy querying times of under 6 seconds for our largest files. Since our data demands are not yet beyond our current hardware, we could not justify the expense of the server at this time, but expect to purchase it as our application's capacity grows. For more detail on this evaluation, please refer to the paper cited above.

Architecture Overview

The DSTDP LAN consists of Windows 3.1 and Windows 95 clients on a Novell 4.1 server. Connectivity is also provided to the CDC mainframe, a Microsoft NT server, and the new SUN server purchased for the data warehouse. Since the majority of the division's personnel time is spent in the Windows environment, the SAS System, version 6.12 for Windows was the chosen platform for development and execution of STDINFO. SAS 6.12 is also licensed for the UNIX server and SAS 6.09 for the mainframe. SAS/CONNECT is used to open a remote SAS session on UNIX from a host session in Windows for STDINFO processing. A batch SAS session in Unix opens a remote session with the mainframe for large data table building and retrieval as new data arrive at CDC. Since busy mainframe job queues result in unpredictable processing times, STDINFO does not initiate a remote session with the mainframe. At the time this paper was written, we were not utilizing the NT server, but we will discuss later in this paper potential uses of the server. For a detailed

description of our network topology please refer to Humphry, *et al*.

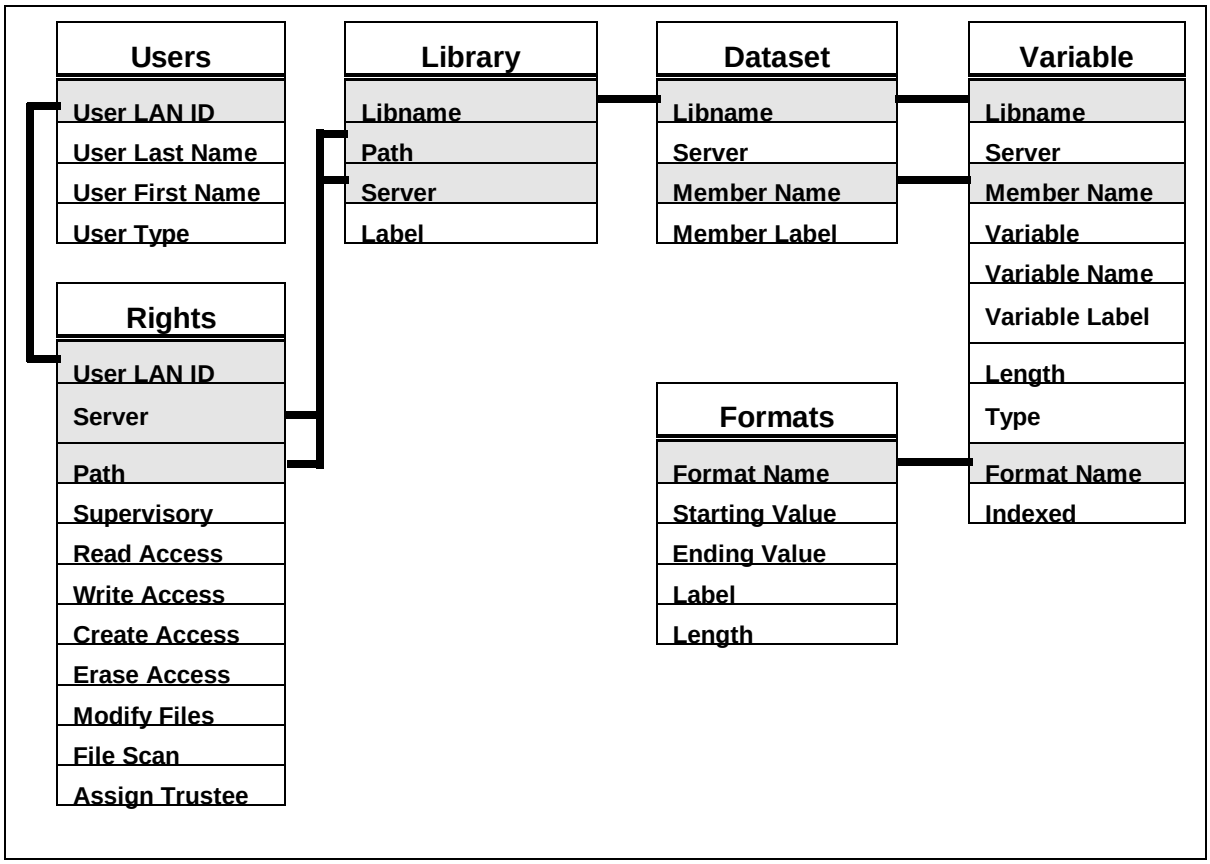
Phase I

For our first attempt at application frame development we selected a large surveillance dataset that would require UNIX processing, minimal "preparation", and would lend itself well to frame design. Since neither of us had AF or SCL experience and we had never been involved in an application development process, we did not have the luxury of conducting a design analysis. Our only option was to make an attempt at a frame realizing the we would not get it right the first time. Only when we began to understand SAS/AF and SCL could we make knowledgeable decisions about frame design. With the benefit of hindsight, we still believe that this approach was our only option. You are not going to read yourself into an application development role.

As our development progressed we began to realize that the frame needed data about the dataset prior to any querying or processing. Since we only wanted to open the remote session once and only when the user had specified what they wanted from the data, we needed to store some information about the data on the LAN side of the application. We realized that this client/server design was imperative for acceptable application performance. Data that the frame required included variables names, labels, and unique values or formats for non-continuous variables. We decided to use the CNTLOUT= option of PROC FORMAT to generate a SAS dataset of the formatted values and the OUT= option of PROC CONTENTS to create a SAS dataset of the variable names and labels. This improved application performance dramatically. Please refer to Sharon Hixon's SUGI 22 paper (Hixon 1997) for more details.

In the first phase of development of STDINFO it became apparent that the construction of the data warehouse was very important to the design and performance of STDINFO. By requiring that all SAS datasets and variables be labeled, and all categorical variables be formatted before data are "checked into" the data warehouse, we facilitated application development while improving the appearance of all data frames. From this point forward all design considerations for the data warehouse or STDINFO had to be considered with the other in mind.

Figure 1



Phase II

The second fiscal year we added two more processors and another 256MB RAM to the UNIX server. Anticipating large data storage needs and not being able

to anticipate future funding, we added drive space to the UNIX server to bring the capacity to 35 GB. Having completed the only two courses SAS offers for application development, we proceeded to look elsewhere for training. After identifying all SAS Quality Partners in the Metro-Atlanta area with application development expertise, we proceeded to contact these three vendors. Scott Groussman of Catalyst Technologies offered to provide additional SAS/AF consulting for training and code review.

It was now time to identify additional STD data and functionality to incorporate into STDINFO. The next dataset selected was the National Electronic Telecommunications Systems for Surveillance (NETSS) data file. This collection of line-listed and aggregate case counts of all reportable diseases is transmitted weekly by state epidemiologists to CDC. With many states in the process of moving to this electronic reporting system, the potential magnitude of this dataset is currently unmeasurable. With less then 10 states reporting line-listed data for more than a year there are still a total of over 900,000 individual cases of STDs reported.

Other functionality we wished to add to STDINFO included the ability to display the reporting status of this data by state and disease and the completeness of vairables over time. In addition, we also wanted to create a more generic utility that would provide users access to all datasets in the data warehouse for variable selection and querying to generate analysis datasets that could be stored locally, on the network, or the UNIX server for further analysis using SAS. This would be the standard data access tool for all division data for all operating systems. Yet another completely separate functionality was a SAS reference book check-out utility. We wished to include this in the first beta test of STDINFO to demonstrate some of the diverse ways in which application development could greatly benefit the day-to day function of the division.

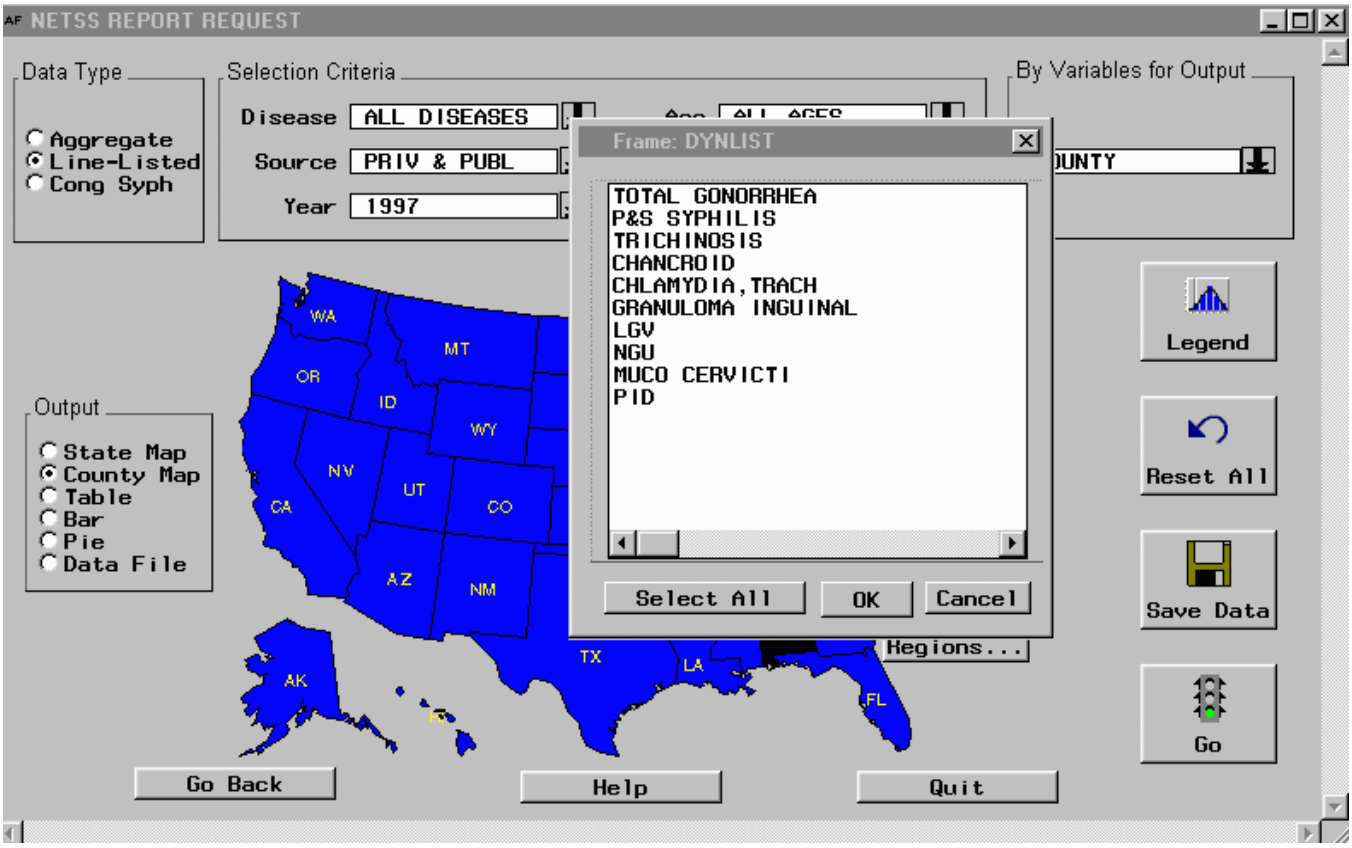
Figure 2

```
-------------------------------------------------------------------------------------------
** code from DOS batch file  **
-------------------------------------------------------------------------------------------
del n:\group\sasapp\maintain\users.txt
rem       List userid's and names for all LAN accounts
z:\public\nlist user show "Full Name" /c >n:\group\sasapp\maintain\users.txt
rem       Open a SAS session and run maintain.sas
n:\prog\sas\sas.exe -config c:\saswin\config.sas -sysin n:\group\sasapp\maintain\maintain
-------------------------------------------------------------------------------------------
** code from SAS program **
-------------------------------------------------------------------------------------------
data newusers;
attrib userid length=$4 name length=$25
       lname length=$20 fname length=$15;
infile 'n:\group\sasapp\maintain\users.txt' missover;
** check to see if line contains a user name **;
input check $ 1 @;
if check='U' then input @7 userid $ / @13 name & ;
else input ;
** keep only lines with user names **;
if userid=' ' then delete;
** convert userid to all upper case **;
userid=upcase(userid);
** keep only CDC-assigned userid's.  (must end in a number) **;
if '0'<=substr(userid,4,1)<='9';
**  extract last name and first name from name string **;
lname=left(scan(name,1,','));
fname=left(scan(name,2,','));
if fname=' ' then do;
  fname=left(scan(name,1," "));
  lname=left(scan(name,2," "));
end;
** drop unneeded variables **;
drop check name;
run;
-------------------------------------------------------------------------------------------
```

As we began to develop frames for NETSS data analysis and data warehouse queries we realized that once again we needed to create datasets describing our data (metadata). Under the guidance of Mr. Groussman we proceeded to build the metadata necessary for STDINFO (figure 1). Once again since all application interaction should result only in LAN side processing until the actual data analysis occurs, we needed this metadata to reside on the LAN server. By providing this data to the application in the form of SAS datasets instead of using AF objects to read this from the master tables or generating tables for every dataset by the methods we used in phase I, we provide optimal application performance with minimal overhead.

You will notice in our metadata tables that all of the variables except library names, paths and user rights are inherent to the structure of SAS datasets and catalogs. Since the SAS system does not allow extended labels for library names, we created this variable in our table. The only data here that must be maintained manually are the library dataset and a master STDINFO format catalog. The users dataset is generated using the Novell syntax and subsequent SAS input statement as seen in figure 2 . These data can now be used by application frames such as in figure 3 to populate SCL lists such as the one displayed. Building the rights table will be discussed later.

Figure 3



Automating Presentation of New Data

It is very important to document the process required to check routinely reported data into the data warehouse. Since our time would be better spent on building capacity into the application, all routine processes should be automated when possible. Data need to be updated at various intervals. CDC receives some data as infrequently as once a year while other data are reported much more frequently. Metadata including users, user rights and data warehouse libraries will be changing daily.

UNIX offers a very simply way to automate this process by scheduling jobs using the *cron* tool. The processes of retrieving STD data from the master NETSS dataset on the mainframe, formatting and labeling all variables, generating the necessary STD data files, downloading them to UNIX server, and building all necessary indexes can be achieved by running one SAS

program in batch SAS session communicating with the mainframe using SAS/CONNECT. This program would need to be run every Wednesday during off peak hours after the new weekly data had been added. This can be easily scheduled through an entry in a UNIX *crontab* as shown in figure 4.

figure 4

```
0 3 * * 4 /data/maintenance/cdc2638
15 3 * * 4 /data/maintenance/netss
```

There are only six necessary parameters in a *cron* statement.

```
Column 1 - minutes
Column 2 - hours (0-23)
Column 3 - day of month
Column 4 - month
Column 5 - day of week (0-7 Sunday = 0)
Column 6 - Command field
```

The first command in our crontab will be executed every Thursday morning at 3:00 A.M. The actual command that invokes SAS is stored in the file CDC2638 as shown in figure 5.

figure 5

```
/sas/sas.exe
-sysin /data/maintenance/cdc2638.sas
-log /data/maintenance/cdc2638.log
```

LAN side data and metadata are currently updated every morning by double clicking on an icon in Windows 95. Since a user must log in to a server with sufficient rights to generate this data, automation of this process is proving to be a little tricky. We have considered several options including Novell scripting and Windows scheduling. Since neither of us are members of the LAN staff, we would depend on others to reschedule or add processes if we used Novell scripting. Windows messaging on the other hand would also require either physically logging in or hard-coding of user ids and passwords, a potential security problem.

At the time of publication, we are investigating scheduling all processes on the UNIX side. If this is possible, all scheduled processes could be listed in one *crontab* greatly simplifying maintenance. We believe we can accomplish this by using SAS/CONNECT to allow a UNIX session to communicate with a secured computer running a SAS/SHARE server process. We will discuss how the SAS/SHARE server is being used later. We will give an up-to-date report of these efforts at SUGI.

Current Status

As the overall architecture of the data warehouse and STDINFO solidifies, we are currently trying to identify other SAS programmers who would be good candidates to become developers and other non-programming personnel for assisting us in the development and maintenance of the data warehouse. We refer to the process of preparing data for the warehouse as *normalizing* the data. This process requires much time consuming work including variable labeling and formatting. We are currently building a feature into STDINFO to give non-programmers the ability to perform these tasks.

There are other tasks that can be performed without application development experience. One glaring example for us is hot-spotting. Since all NETSS data are reported by state and county and as of version 6.12, SAS does not offer hot-spotted map objects by county, we need to create several such objects. We create our first hot-spotted state/county map and then show clerical staff how to do the rest.

We have identified a finite list of features that we wish to include in our beta test. It is very important to plan a beta test that will have enough functionality to insure continued support for development, but not so much that it cannot be completed before the next fiscal cycle. By the time this is presented at SUGI, we will have begun beta testing of STDINFO.

There are features that must always be considered in the first version of any application. We will now discuss some of these.

Receiving Feedback

There are two types of feedback that you can expect from your users. First, you have the requested and in-depth feedback from your beta testers. Secondly, all other users should have a mechanism through which to provide feedback to the development team regarding the current live application and it's functionality. Once again, adhering to our "keep it simple" philosophy, we found one solution for both of these needs. Every frame in STDINFO will contain a feedback button. This button will invoke a frame that will use the extended text entry object to allow a user to type an unlimited amount of text. The users id, the name of the current active frame, date, and time is stored in a SAS dataset. This dataset is keyed to another SAS dataset that contains one record for every line of text entered.

Since multiple users will open this dataset for update access, the SAS/SHARE server is required. As discussed before, a dedicated SAS/SHARE server process will remain running on a PC in a secured computer room. Since the main function of this machine will be serving and not processing, the location of this server in the network topology is much more important that the actual processing capacity of the machine on which it is running. Our SAS/SHARE server is located on the same

100 MB ring as the data server. It is as close to the data as possible.

The steps for starting a SAS/SHARE server are quite simple. First, the server must be listed in the services file of the computer on which the server session is running and all clients needing to use the server. Since STDINFO is a division-wide application, we edited the services file of all LAN clients. Second, an alias must be given for the IP address for that server. The three ways of accomplishing this are listed in figure 6.

figure 6

`% let stdshare=123.123.12.12`	Since this is the last place Windows will look for an alias, this is the least efficient option.
add the line: `123.123.12.12 stdshare` to the *hosts* file in the Windows directory:	This allows for faster resolution of the alias but would be required in the *hosts* file of all clients
register the address and server in your domain name server	This is optimal if you use a DNS.

The third necessary step for starting a SAS/SHARE session is to submit code similar to figure 7 on the SAS server machine. A password is entered to prevent unauthorized users from stopping the server.

figure 7

```
options comamid=tcp;
proc server id=sasshare oapw=doodles;
run;
```

All a user needs to do to access the SAS/SHARE server would be to assign a libname as in figure 8.

figure 8

```
Libname feedback 'n:\group\feedback'
server=stdshare.sasshare;
```

This libname is automatically assigned in the STDINFO *autoexec.sas* file.

The server is quiesced by using the syntax in figure 9. Quiescing means to gradually stop the server as all current accesses through it are closed. For instance, if a user is entering feedback, when they are finished they will no longer be able to access that server. When all users currently entering feedback finish, the server process will be stopped and no access will be allowed until it is restarted.

figure 9

```
options comamid=tcp;
proc operate;
  quiesce server sasshare / sapw=doodles;
run;
```

Help

At time of publication, we had not resolved how we would introduce help functionality into STDINFO. We would like for our help feature to serve as printable documentation for an STDINFO user's guide as well as on-line help. Because of our staffing limitations, we want to eliminate as much redundant work as possible. Ideally our help screens could be organized easily to form an STDINFO user manual or training materials.

There are three basic techniques we have considered. First, we considered using the _HELP_ method available in release 6.12 of SAS. Although this method could be included by developers at development time, we did not see how we could satisfy our desired needs described above with this technique. The next obvious consideration was third party software to provide help features through Winhelp. This of course would require some licensing costs and would result in compatibility issues should our operating system change. Finally, we decided we could deliver help to STDINFO simply by displaying HTML text in either a client-side browser or a frame object.

HTML help could provide several benefits. Since we plan to eventually move part of the application to the CDC intranet and the Internet, this would be a way

for us to begin to build an Internet skill set. HTML would allow for very nice help displays and could easily be used to generate the documentation discussed above. All help frame written in HTML would require no recoding when moving across operating systems or platforms. We will update you on our endeavors in this area at SUGI 23 or we could be contacted through email afterwards for more information.

Security

Chances are your data have some level of security defined already. In DSTDP our data on the network are protected through Novell trustee assignment rights. On UNIX, files are protected through read, right and execute rights assigned by the file's owner. This is definitely a place to make a small team decision. Why would we duplicate efforts? Since a user id and password are required to log onto the LAN server, we will not require one to use STDINFO. A user will only have access to data as defined by their rights assigned to them in Novell or UNIX. In order for this to work, we somehow must identify all rights of a user upon entry into STDINFO and assign appropriate LIBNAMES for data stored on the LAN. LIBNAMES pointing to data stored on UNIX will be assigned at run-time during the remote session.

We create a SAS dataset of the users by piping the output of a Novell NLIST command to a text file and subsequently reading this file with an input statement. This process is very similar to the creation of the USERS file (figure 2.) We create a SAS dataset of all directories in the data warehouse by piping the output of a DIR command to a text file and reading it using an input statement. We create a SAS dataset of all defined right objects by piping the output of the Novell RIGHTS command to an output file and reading it in as we did above. With these three files we now how have enough information to create a dataset with one record for every directory for every user where they have at least read access. This file, coupled with the library dataset can be used to assign a LIBNAME for all appropriate directories.

The process described above is not that simple. Since all Novell rights are inherited from parent directories to subdirectories, we must create a record for each subdirectory of a directory for which a user has rights. This can be achieved through a Cartesian product merge. This is an SQL join where all members of one table are matched with all members of another table resulting in N1 X N2 observations. We then identify all subdirectories of a subdirectory using the index function to search for the parent directory in the path of the subdirectory. We then select only matches for directories where rights were assigned. Surprisingly, this process takes only a few seconds and is repeated every morning to build the rights metadata. Once this functionality was put in place it required no further maintenance and security is still maintained by our LAN personnel.

Working Across Operating Systems

When you have more operating systems then members on your development team, then someone is going to do administration and development on multiple platforms. FTP software can be used in windows to allow LAN side writing and editing of SAS programs and copying of these programs to the UNIX server. Windows Telnet software can be used to log in to the UNIX server and perform all system administrative duties, including adding users and scheduling *cron* processes. Various free and shareware versions of these two applications can be downloaded from the Internet or are often supplied with Windows software. Existing Windows based 3270 emulation software is used for mainframe programming. This way we can develop and administer from our desktop while all application servers reside in a secured and climate controlled computer room.

Promoting the Application

DSTDP, with its organizational structure allows for logical planning of STDINFO demonstrations. Since this is a division project, it is very important that every branch recognize how it will benefit from the application. Our plan is to work with one branch member prior to their branches scheduled demo to check their data into the data warehouse. We will show this user the capabilities of STDINFO and then ask this user to lead the demonstration of the application to their branch. This way we build alliances, use branch-specific data, and demonstrate the facility and utility of the application. Next, we ask each branch to prioritize their needs for data warehousing and adding functionality to STDINFO. Finally, we ask for assistance whenever possible!

Conclusions

When all of the work for an application development project has to be done by a few people, it is very important that you scrutinize every design decision. No one will tell you that this process is easy, but it is definitely doable. We are succeeding because of solid design and obvious need. This application will benefit everyone, and we had to be able to show that on paper and then turn it into an application.

If you are a small team, than chances are team development is new to your organization as it was to ours. One of the first things you will learn as you create that second frame or prepare for another part of the application, is that by establishing and enforcing conventions, you can minimize redundancy and add application uniformity across developers. As your team grows, code review, object review and application testing will be necessary to see that conventions are being

adhered to. This is when you will recognize the benefit of a solid foundation of conventions.

Good projects never end. We will be working on STDINFO for years to come. SAS/AF may not be our future. We may decide on JAVA, HTML, C++ or some yet unnamed language. Because of the depth of statistical tools provided by the SAS System, we expect to use it for data analysis for years to come, so we will always be interested in participating in a forum of SAS application developers. Please feel free to email the authors if you would like to discuss anything here or make suggestions. We will welcome the interaction. Since we are employees of the United States federal government, we would like to make all code available upon request. We plan to have source code and metadata development code available on the DSTDP home page in the future. Visit www.cdc.gov for more information or contact the authors listed below.

Acknowledgments

The authors would like to acknowledge Scott Groussman of Catalyst Technologies for his constant wisdom and guidance and Van Munn of the Centers for Disease Control for being "THE" SAS reference at CDC. Without the assistance of either of these two, we would still be thinking like DATA STEP programmers.

References

Hixon S, Ransom R (1997), "Storage Strategies for Data, Formats, Catalogs and Other Information in Application Development Using the SAS System", Proceedings of the 22nd Annual SAS Users Group International Conference, pp 1405-1408

Humphrey K, Ransom R (1997), "Systems Architecture Solutions for an STD Information System", Proceedings of the 22nd Annual SAS Users Group International Conference, pp 1425-1430

Authors

Ray L Ransom
Epidemiology and Surveillance Branch
Division of STD Prevention
Centers for Disease Control and Prevention
mail stop E02
1600 Clifton Road
Atlanta, Georgia, USA 30333
email: rlr1@cdc.gov
phone: (001) 404-639-8369
FAX: (001) 404-639-8610

Sharon Mosley-Hixon
Statistics and Data Management Branch
Division of STD Prevention
Centers for Disease Control and Prevention
mail stop E63
1600 Clifton Road
Atlanta, Georgia, USA 30333
email: shm5@cdc.gov
phone: (001) 404-639-8882
FAX: (001) 404-639-8611