Paper 280-27

Energizing End Users with a Slice of SAS® and a Cup of Java™

Randy Curnutt, Solutions Plus, Inc., Indianapolis, IN
Michael Pell, Solutions Plus, Inc., Indianapolis, IN
John LaBore, Eli Lilly and Company, Indianapolis, IN
Mitch Mitchell, Eli Lilly and Company, Indianapolis, IN

ABSTRACT

Many corporate information systems have evolved to integrate a diverse mixture of hardware platforms. Although commonplace, the mainframe has been joined by a plethora of UNIX and NT servers as well as an army of personal computers often connected by Local Area Networks (LAN) or Wide Area Networks (WAN). Further, software tools in this environment are diverse. Users now wish to analyze their mainframe data with PC and UNIX tools in addition to using mainframe tools. It is often necessary to migrate data to new platforms because some tools require local data access. Users may become intimidated and stymied by the prospect of finding an easy and reliable method for moving mainframe data to remote platforms. We explore how the flexibility of SAS system architecture, combined with Java programming capabilities, provides instant access in an easy-to-use tool for point-and-click data migration and conversion (OS/390 SAS, DB2 views, Oracle views) to a PC or UNIX/NT servers. We also discuss methods to automatically convert the resulting dataset to the desired format (SAS dataset, SAS transport file, Excel, CSV, tab delimited, space delimited, or dBASE) on the target platform. Our tool implements a verification process that ensures integrity of transferred data. Additionally, meta-data is available before and after the transfer.

INTRODUCTION

BUSINESS PROBLEM

In today's IT environment many different hardware and software platforms are often thrown together in an attempt to meet diverse business needs. This hodgepodge approach forces the user to learn and work with each different system in order to perform the steps necessary to obtain the needed data or programs. Often users spend more time learning an operating system and/or protocol than doing the job they were hired to do. For example, if the specific operating system commands needed to logon and traverse a directory structure are very different, this significantly increases user knowledge/time requirements. However, even subtle differences between platforms can have an impact. For example, floating-point numbers are stored differently on various operating systems. This may result in discrepancies in the significant digits in high precision decimal numbers that may greatly discomfort a systems analyst (Klenz, 1992). Conversely, a statistician who understands the net effect of this type of difference may not perceive it as a problem.

The DATAccess Tool facilitates clinical data analysis by providing an easy way to transport and convert SAS data on the OS/390 mainframe to the platforms and formats desired for analysis. Instead of learning mainframe or communications programming commands, or waiting for a systems analyst to move the data for them, the user is free to focus on analyzing the data.

GOALS

The goal of the DATAccess Tool was to provide an intuitive graphical user interface that gave non-technical users a simple and repeatable process for copying (and converting) data from any platform to any supported target platform. Data integrity was of the utmost importance, especially since this tool may be used to create SAS Transport files that are submitted to the regulatory agency for review. The DATAccess Tool facilitates the verification process to ensure the transferred data is accurate. Since the tool brought together so many disparate technologies and platforms, it was imperative that it go through a rigorous validation process to ensure that it functioned as desired. It was also important for the tool to provide instantaneous transfers since the users had previously experienced lengthy delays waiting on technical analysts to move and convert the data for them.

ENVIRONMENT

Our environment is very diverse, both from a user perspective as well as from a technological standpoint. The end-users span a very broad base; from non-technical to very technical. This user base includes statisticians, systems analysts, data analysts, research scientists, and physicians; these staff are located at both US and overseas locations.

The hardware encompasses an OS/390 mainframe, Sun Solaris systems, Windows NT servers and Windows based client machines.

The software environment is comprised of a Java applet, Windows NT server, Windows 95/2000/NT clients, PC SAS 6.1/8.2, UNIX SAS 6.12/8.2 and OS/390 SAS 6.09/8.2, JCL, FTP. The data is a mixture of SAS, SAS Transport, DB2, Oracle, CSV, dBase, tab delimited text files, and Excel. Communications techniques utilize both SAS/Connect® and FTP. The DATAccess Tool also generates and submits OS/390 JCL jobs, as well as dynamically generating SAS programs that run on the most appropriate platform via SAS/Connect (i.e., the Windows client, OS/390, UNIX, or NT Server).

SOLUTION

One solution to the above problem is to develop an application that uses SAS, SAS/Connect and Java to provide functionality to the users so that they can accomplish their desired tasks without learning the different nuances of multiple hardware and software environments.

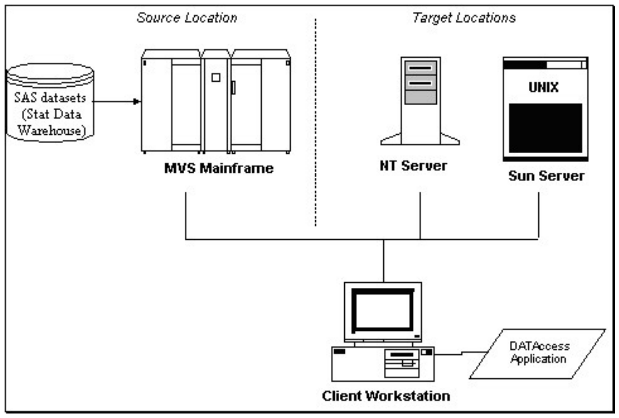
All of these multi-platform tasks can be performed by SAS/Connect itself; however anyone who chooses to do such a task must be fairly knowledgeable of the entire hardware and software environments in which they are working. This includes technology differences between the platforms being used, networking and communications issues between the platforms, operating systems, protocols, multiple languages, etc. Solving all of these issues at any one time can be difficult, even for an experienced SAS user. Optimally, the SAS user should not be sidetracked from their primary job with all of these diverse issues. Some of the tasks that are commonly performed on a routine basis yet may be difficult to complete include:

- Moving data to other platforms
- Converting data to other formats
- Analyzing and reporting
- Data manipulation
- Input/output
- Consistency, validation, repeatability
- Formally publishing results

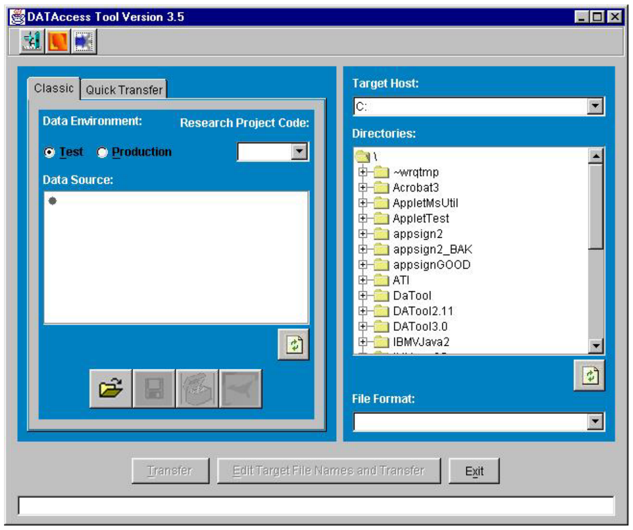
The driving force for creating the suggested application is that it provides a custom solution for all users and prevents them from reinventing the wheel. Furthermore, with this approach the solution is built and validated once, instead of requiring retesting every time it is used.

SAS and Java provide the glue necessary to tie all of the various technologies together. Architecturally, the DATAccess Tool is viewed as a pipe that connects a source to a target, and manages the transfer and conversion process. Realistically, the initial design was for the source to always be SAS data residing on an OS/390 platform, but we did not wish to architect to such a rigid standard. The basic idea is that it shouldn't matter what platform or data structure resides at either end of the pipe.

The following diagram depicts the physical architecture for the tool:



The focus of this technique is to optimally integrate multiple hardware platforms using SAS/Connect with Java so that maximum convenience is provided to the end user. The application itself is developed in Java and it can be deployed as either a stand-alone application or a Web based applet. SAS programs and macros are embedded within the Java code, so that Java acts as a program generator for SAS programs and macros.



Based on the user's selections (see screen shot above), Java is used to dynamically generate PC SAS, OS/390 SAS, UNIX SAS programs or OS/390 JCL jobs to accomplish the data transfer and conversion. The tool submits the generated program to the appropriate SAS engine depending on the client and server configuration. For example, if the user has PC SAS, then most programs generated by the application would leverage the local SAS software. If the user does not have PC SAS, then all generated programs are submitted to OS/390 SAS, which would make use of SAS/Connect as needed. If the user desires a transfer to the UNIX platform, this would always be accomplished by generating an OS/390 SAS program that runs SAS/Connect on the OS/390 to spawn a job on the desired UNIX host.

Java, SAS programs and SAS macros are highly standardized and portable. SAS programs, macros and datasets are not binary compatible across platforms but the programs themselves are highly compatible in terms of commands and syntax. There are minor distinctions in syntax for OS/390 SAS versus PC SAS, but for the most part the majority of programs can be used across platforms with only minor modifications.

STATIC OR DYNAMIC PROGRAMS

Combining Java and SAS provides an architecture capable of generating either static, predefined SAS programs, or dynamically generated custom programs based upon selections a user has made. To understand the static and dynamic generation of SAS programs, let's examine how this can be accomplished with Java. Java programs contain classes, a term describing a combination of attributes and methods that may be executed. One approach to generating SAS programs or macros is to have a Java class that contains methods, similar to functions in other languages, dedicated to concatenating a number of Strings together to build the desired program. Specifically, the method actually uses StringBuffer objects, since this is a far more efficient manner for concatenation within the Java programming language.

Here's an example of appending two StringBuffers together.

```
StringBuffer sb1 = new StringBuffer(
  "SAS and Java ");
StringBuffer sb2 = new StringBuffer(
  "make a good team.");
StringBuffer sb3 = new StringBuffer();
sb3.append(sb1);
sb3.append(sb2);
```

```
sb3.toString();// will return "SAS and Java
make a good team."
```

This approach can append together any number of predefined StringBuffer objects to build one long String that contains a whole SAS program. Here's an example of building a static SAS program which is run on the OS/390, and returns the output of a PROC CONTENTS in a text file:

```
StringBuffer sasPgm = new StringBuffer();
sasPgm.append("//USERID JOB (,8305,S),
   'PROC_CONTENTS',MSGCLASS=T\r\n");
sasPgm.append("//*\r\n");
sasPgm.append("//SAS     EXEC
SAS,SOUT=*,OPTIONS='LS=80 MPRINT SGEN'\r\n");
sasPgm.append("//SYSOUT    DD SYSOUT=*\r\n");
sasPgm.append("//SYSIN     DD *\r\n");

sasPgm.append("%MACRO CONTENTS(
   PRINTTO=,) ;\r\n");
sasPgm.append("%LET LIB=%UPCASE(&LIB);\r\n");
sasPgm.append("LIBNAME LNAME \"ABC_LIB\"
   DISP=SHR;\r\n");
sasPgm.append("FILENAME PRT \"&PRINTTO\"
    DISP=(OLD,DELETE);\r\n");
sasPgm.append("RUN;\r\n");

sasPgm.append("FILENAME PRT \"&PRINTTO\"
    DISP=(,CATLG) ");
sasPgm.append("SPACE=(CYL,(1,1))\r\n");
sasPgm.append("       RECFM=FB
   LRECL=80;\r\n");
sasPgm.append("RUN;\r\n");

sasPgm.append("PROC PRINTTO PRINT=PRT
   NEW;\r\n");
sasPgm.append("PROC CONTENTS
   DATA=LNAME._ALL_;\r\n");

sasPgm.append("%MEND CONTENTS;\r\n");
sasPgm.append("%CONTENTS(PRINTTO=
   USERID.TMP);\r\n");

sasPgm.append("/*\r\n");
sasPgm.append("//IFGOOD    IF (^ABEND &
   SAS.SAS.RC <= 4) THEN\r\n");
sasPgm.append("//RENAME    EXEC
   PGM=IDCAMS\r\n");
sasPgm.append("//SYSPRINT  DD
   SYSOUT=*\r\n");
sasPgm.append("//SYSIN     DD  *\r\n");
sasPgm.append("   ALTER USERID.TMP -\r\n");
sasPgm.append("   NEWNAME(USERID.TXT)\r\n");
sasPgm.append("//ELSE      ELSE\r\n");
sasPgm.append("//DELETE    EXEC
   PGM=IEFBR14\r\n");
sasPgm.append("//SASDEL    DD
   DSN=USERID.TMP\r\n");
sasPgm.append("//ENDIF     ENDIF\r\n");
```

The Java code above creates:

```
//USERID JOB (,8305,S),'PROC_CONTENTS',
   MSGCLASS=T
//*
//SAS     EXEC SAS,SOUT=*,OPTIONS='LS=80
   MPRINT SGEN'
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
%MACRO CONTENTS(PRINTTO=,) ;
%LET LIB=%UPCASE(&LIB);
LIBNAME LNAME "ABC_LIB" DISP=SHR;
FILENAME PRT "&PRINTTO" DISP=(OLD,DELETE);
RUN;
FILENAME PRT "&PRINTTO" DISP=(,CATLG)
   SPACE=(CYL,(1,1))
       RECFM=FB LRECL=80;
RUN;
PROC PRINTTO PRINT=PRT NEW;
PROC CONTENTS DATA=LNAME._ALL_;
%MEND CONTENTS;
%CONTENTS(PRINTTO= USERID.TMP);
/*
//IFGOOD IF (^ABEND & SAS.SAS.RC <= 4) THEN
//RENAME    EXEC PGM=IDCAMS
//SYSPRINT  DD  SYSOUT=*
//SYSIN     DD  *
   ALTER USERID.TMP -
        NEWNAME(USERID.TXT)
//ELSE      ELSE
//DELETE    EXEC PGM=IEFBR14
//SASDEL    DD DSN=USERID.TMP
//ENDIF     ENDIF
```

Dynamic generation is only slightly more complicated. It extends the previous approach by appending the value of variables between the StringBuffer objects. When the program is executed the method builds a single String that appends the StringBuffer value with the value of the given variable. For example, on a user interface a field is used to provide for the entry of a userId. In this situation presume that a value of “RandyC” was entered into the user interface as the value of userId. Following is a code snippet for doing dynamic generation:

```
public String buildSasMacro(String userId,
String password, String body)
{
  StringBuffer sasPgm = new StringBuffer(
   "%macro myMacro(userId=");
  sasPgm.append(", pswrd=);");

  sasPgm.append("\r\n\r\n");//2 returns
  sasPgm.append(body);
  sasPgm.append("\r\n\r\n");

  sasPgm.append("%mend myMacro;");
  sasPgm.append("%myMacro(userId=\"");
  sasPgm.append(userId);
  sasPgm.append("\",pswrd=\"");
  sasPgm.append(password);
  sasPgm.append("\");");

  sasPgm.toString();
}
```

The output of this method is:

```
%macro myMacro(userId=, pswrd=);

the body of the macro would be generated
here...

%mend myMacro;
%myMacro(userId="RandyC",pswrd="myPass");
```

The SAS code that is generated can easily be saved to a file. Doing this allows the code to later be edited by the user or it could be used to execute the program again without needing to regenerate the program.

Conversely, it would also be simple enough to have Java read the ‘template’ SAS program in from a file, and only alter the variable portions of it. This would reduce the amount of SAS code that is embedded within Java. It would also allow for an easier technique of altering the SAS program (i.e. not having to change Java code and re-deploy).

BENEFITS

Not only can Java be used to generate the SAS scripts, it can also submit them to either a remote SAS server or to the local PC-SAS software.

Benefits of this approach include:

- A means to easily develop web-enabled platform independent applications
- A graphical user interface that provides an intuitive, easy to use interface for the user, shielding them from the behind-the-scenes details and protocols
- Enhanced security since the protocols and access methods are embedded within the application itself. The user can still be required to enter a password via the user interface but all of the secured functions are performed internally by the application, thus denying a would-be hacker the chance of altering the flow of the application

IMPLEMENTATION

LOCAL BATCH SUBMISSIONS

To submit a SAS program to PC SAS from Java, the SAS program is written to a temporary file on the local machine, then PC SAS is invoked using the temporary file as a parameter. This is equivalent to the following DOS command:

```
SAS.exe -NOSPLASH -ICON -SYSIN tempfile.sas
```

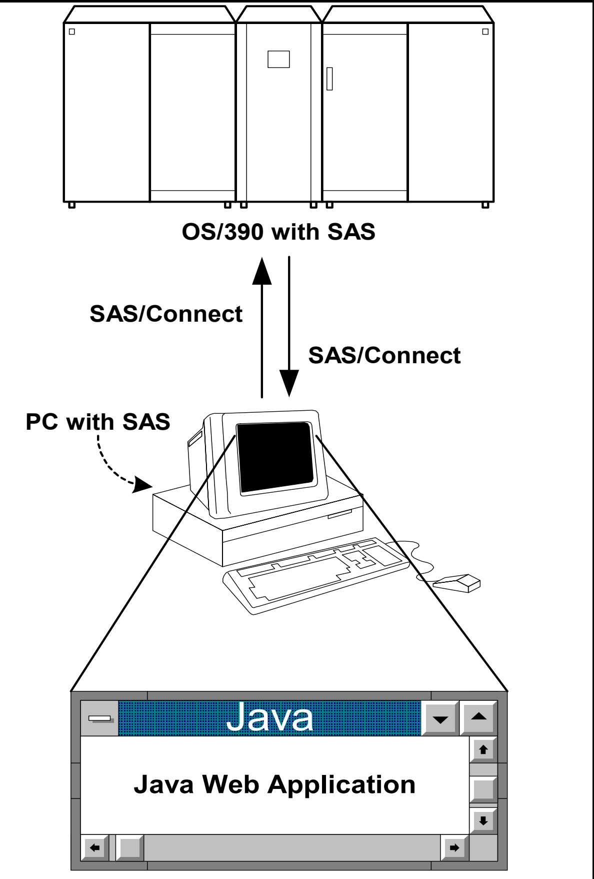
The appendix contains an example of how Java can submit the generated SAS program to PC SAS. See Example 1 for an overview.

There are a couple of issues to consider when using SAS/Connect. First, there must be an understanding of how PC-SAS connects to the remote server. It uses a sign on/sign off script file, such as tcptso.scr, to connect and sign on to the remote host. The networking environment and the desired remote host platform are the driving factors as to which script file should be used. The default scripts provided with the SAS software may need to be altered to accommodate the specifics of a remote host. One important change that is necessary when modifying the scripts is that there will be a need to develop a script that can be used for batch submissions by the application. The scripts come developed for an interactive session and are written to query the user for their user id and password. For the application to use such a script, it must be altered to pass in the user id and password and not query the user for these values (the querying is done by Java). Secondly, one must consider that a server might restrict a user to a limited number of sign-ons. This process will likely account for a sign-on, so it may be an important factor if the server's limit is one sign-on.

SAS/Connect provides the communication mechanism within a SAS framework to execute SAS commands on a remote host. SAS/Connect requires that the user login to the remote host. Once this occurs the user is free to write code to run on this remote host. This is done with the "rsubmit" and "endrsubmit" commands. When the user is ready to execute code on a remote host an "rsubmit" is issued in the SAS program. All SAS commands after the rsubmit are then run on the remote host until an "endrsubmit" command is issued.

For reporting errors or the SAS log back to the user via the Java code, the PC SAS program should use a PROC PRINTTO to output the errors or the SAS log to a temporary file:

```
PROC PRINTTO PRINT='C:\temp\saslog.log';
run;
```



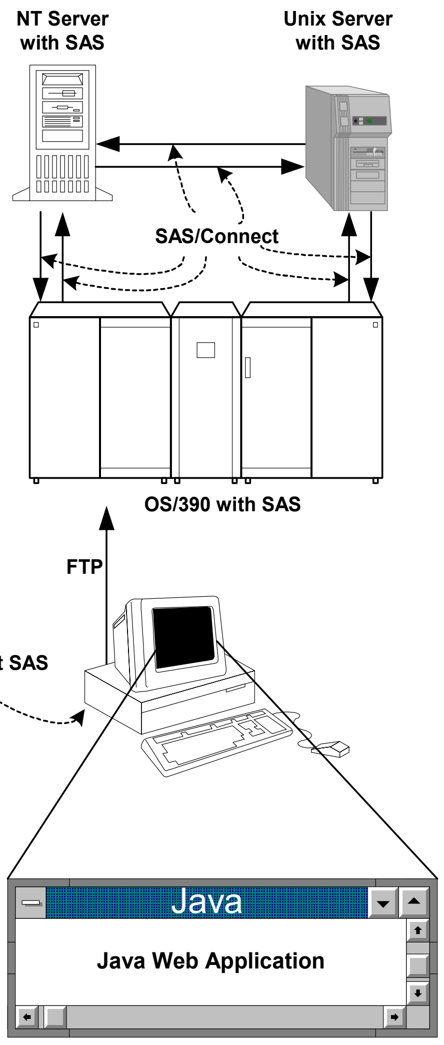
Example 1. Completing a task on a remote platform via Java, PC SAS, and SAS/Connect

When the PC SAS program completes, the Java code will look for the temporary file, read its contents into memory, then delete the temporary file. It is important to remember that the output will not show up in the PC SAS Log window when developing and testing the SAS program with this PROC PRINTTO. It only shows up in the temporary file. Therefore the programmer may want to be selective about when that file is deleted. This will not be an issue once the SAS program is tested and validated.

REMOTE BATCH SUBMISSIONS

There are several approaches when submitting a SAS program to a remote host such as OS/390 or UNIX. This approach uses FTP to submit a SAS program via a JCL job. This can all be done in Java as well, however the process is a bit more involved than submitting a SAS program to PC SAS. (Java implementations of the FTP protocol are available for purchase or as freeware. For an example of developing an FTP class see "Hacking Java" by Mark Wutka).

The first step is to create a JCL program that runs a SAS job. The static SAS program shown in the 'Static or Dynamic Programs' section above is a good example. See Example 2 for an overview.



Example 2. Completing a task on a remote platform Via Java and SAS using FTP and then using SAS/Connect to complete subsequent tasks on other remote platforms

The next step is to submit the JCL to the OS/390. In this application, a Java class is responsible for handling all FTP commands for communication to the remote server. Some of the commands are specific to the remote system's operating system, so it is important to design in this flexibility. The FTP Java class will use a Socket to connect to a port on the remote machine. In this case, set up the data connection using the PASV command rather than the PORT command. This causes the server to establish a listen socket. This is important because a Java applet may not allow an incoming socket connection from the remote server.

The FTP class will provide a user id and password (collected from the user) to establish a valid session. From there, this class performs typical FTP types of functions, such as list, get, put, delete, etc.

Once there is an FTP class that can communicate with the remote server, one must create a class that uses the FTP class to submit a JCL job. This class knows the required behavior of the specific remote server. For example, to submit a JCL job to the OS/390, this class would first prepare the OS/390 session by telling it that the next command it receives will be a specific file type, a JCL job.

```
        "site filetype=jes"
```

It would then send the JCL job via the FTP class, using the STOR command. The STOR command tells OS/390 that a file is about to be sent. The file is actually sent from Java to the OS/390 through the socket using a Java OutputStream class. With each command sent to the remote server, the remote server will return a response code, so there will also be a need to check for valid response codes and handle any errors.

Now that the file is on the remote server, the next step is to run the job. This is also done by sending a command using the FTP class:

```
        "site filetype=seq"
```

At this point, the JCL job should start and run. Per the JCL example above, the Java code should start monitoring the remote server for a file named 'USERID.TXT'. Once the file shows up, Java can then use the FTP class to GET the file. The file will be read in as a String, which is then presented to the user.

Again, this is a very simplified summary of submitting a JCL job from Java. It is recommended that programmers find a book such as "Hacking Java" to get started. Additionally, there are FTP classes now available for download from various sites on the Internet.

ADVANTAGES & DISADVANTAGES

There are several advantages when using SAS/Connect on OS/390 to submit a job to a remote host (which also has SAS/Connect installed).

- User is not required to have PC SAS on their local machine.
- Data doesn't have to be moved; instead SAS/Connect allows execution of the SAS program on the host where the data resides.
- This approach leverages the power of the remote host, which is likely far more powerful than the local machine. For example, if the SAS program runs a CPU intensive procedure that will process a large number of rows then it is far more efficient to run this program on the powerful remote host instead of the user's PC.

Some disadvantages of this approach are:

- The developer who codes this piece of the application must be knowledgeable of the FTP protocol, JCL, SAS and Java programming.
- The application will experience delays due to connection time if there are frequent connections and disconnections, depending on the architecture.
- Once a job is submitted the application must wait until the job is actually run by the remote host.

Minor obstacles are to be expected and addressed when bringing together so many different hardware and software technologies.

Here are a few items worthy of consideration:

- Do set options = nocaps on OS/390 jobs. Problems may be experienced with SAS/Connect from OS/390 to Unix due to the password being uppercased by SAS/Connect, thus causing logins to fail. With Version 8 of SAS this may no longer be an issue, but watch out for it.
- Be aware of remote host connection limits. OS/390 only allows one direct user connection and this cannot be set by a parameter. Therefore, if a user is logged on via a telnet session they will be unable to connect via SAS/Connect. Note that FTP sessions are not counted in this area and a user can be connected to OS/390 via FTP and still successfully connect via SAS/Connect.
- Consider how to monitor remote jobs for completion or failure. One approach is to always FTP either a validation log or error log to the target destination. The Java code can wait until one of these files exists before continuing. See the JCL code of Appendix B for an example of this approach.
- Take the extra step of capturing error messages on the remote host and handle these in an appropriate manner.

A variety of SAS Procedures are used within the generated programs. For example, PROC SQL is used because it allows the tool to dynamically limit which rows and columns to include in the target dataset (per the user's specification). PROC DOWNLOAD is used to copy the dataset from a source to a target host, and PROC COMPARE is used when possible to assist with the verification that the transferred data is exactly the same as the source data. There are also cases where a SAS PROC was not available, such as for creating text files from the SAS datasets. Buchecker (1996) describes SAS macros that effectively automate the creation of flat files from SAS datasets.

When it is necessary to execute an OS/390 based program, the tool will generate a JCL job. It then uses the FTP protocol to connect to the OS/390 mainframe, send a command to alert the host that the next command should be auto-submitted to the JES2 job stream.

REVIEW OF THE CODE SAMPLE IN APPENDIX B

As stated earlier, this code sample was generated by a Java program based upon the selections made by a user. Note that in this case the file is an OS/390 JCL file ready to be executed on the mainframe. After the job statement is an exec command that launches SAS.

Notice that a SAS macro is used in the job and is passed in following the JCL statement:

```
//SYSIN DD *
```

In the macro, an options line is used to define the remote system that will be accessed:

```
options remote=unix comamid=tcp nocaps;
```

The next line to note is the filename command that identifies the location of the SAS/Connect script file that will be used. That is immediately followed by the signon command that will trigger SAS to connect to the identified remote server.

```
filename rlink
'syst.sas.sas608.ctmisc(datcpunx)';
signon unix;
```

Lastly, the code between the `rsubmit` and `endrsubmit` accesses the remote server. In the example, the first occurrence of the `rsubmit` then does a `PROC` UPLOAD to transfer the file from OS/390 “client” to the UNIX system. In this case, the remotely submitted code is running on the UNIX host, which is viewed as the server.

Additional tasks are also included in Appendix B.

CONCLUSION

This paper highlights some of the advantages and disadvantages of creatively combining and leveraging the unique features of SAS, SAS/Connect and Java. The techniques presented allow extremely powerful and flexible systems to be built and deployed. Useful aspects of Java and SAS are introduced, and various methods of integration with SAS/Connect are explored, including dynamic and static SAS program generation. Additionally, techniques are discussed for submitting the generated programs in both local (PC) and remote (OS/390 mainframe, Unix, and NT servers) batch jobs. Advantages and disadvantages of each approach are reviewed. The techniques presented allow the creative systems developer to build timesaving applications that may not be easily possible with any single one of the discussed languages. Some additional benefits of the DATAccess Tool are:

- Provides a verification report that addresses the integrity of the transferred data and that can be submitted to a regulatory authority.
- Provides transfer progress monitoring (color-coded)
- Provides users instant access to SAS datasets on the platform they prefer and in the format they choose
- Adheres to all corporate security standards
- Provides metrics reports on the number of transfers, what data was transferred and where, who transferred the data, what formats were used, etc.
- Uses batch mode for multiple transfers
- 30,000+ files transferred with no data corruption
- 37 teams have used the DATAccess Tool
- Improves the clinical analysis process by allowing the users more flexibility to choose the software tools that provide the analytical capabilities they need
- Provides an easy to use graphical user interface for transferring SAS datasets from OS/390 to the NT or Unix servers, or the local machine.
- Shields the user from the complexities of OS/390, JCL, SAS/Connect and FTP
- Leverages capabilities of OS/390 and UNIX
- Allows user to specify subsetting criteria by columns and rows

REFERENCES

Buchecker, M. M. 1996. "%FLATFILE, and Make Your Life Easier," Proceedings of the Twenty-First Annual SAS Users Group International Conference, 21, 178-180.

Klenz, B. W., 1992. “Handling Numeric Representation Error in SAS Applications,” Observations, Vol. 1, Number 3,

SAS Institute Inc. (1996), *SAS Companion for the Microsoft Windows Environment*, Cary, NC: SAS Institute Inc.

SAS Institute, Inc. (1996*), SAS Companion for the MVS Environment, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993), *SAS Companion for Unix Environments: Language, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1994*), SAS/Connect Software: Usage and Reference, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS Guide to Macro Processing, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989*), SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

Wutka, M.,1997, "Hacking Java: The Java Professional's Resource Kit", Que Publishing, Indianapolis, IN

TRADEMARK NOTICE

ABOUT THE AUTHORS

Randy Curnutt, Solutions Plus, Inc. (http://www.sol-plus.com) Randy Curnutt is the president of Solutions Plus, Inc., a software consulting company that specializes in applying leading edge technologies in order to provide comprehensive solutions to its clients. He focuses on client/server solutions, especially object oriented technology, and relational database management systems.   He has experience with Java, Smalltalk, Visual Basic, C, C++, Oracle, MS SQLServer, and numerous other development languages. Randy may be reached via email at rcurnutt@sol-plus.com.

Michael Pell, Solutions Plus, Inc. (http://www.sol-plus.com) Michael Pell is a consultant at Solutions Plus, Inc., a software consulting company that specializes in applying leading edge technologies in order to provide comprehensive solutions to its clients. He focuses on the analysis, design, and implementation of object oriented technology client/server solutions. Michael has 2-3 years of Java development experience, and spent 6 years as an IBM consultant prior to joining Solutions Plus, Inc. Michael may be reached via email at mjpell@sol-plus.com.

John LaBore, Eli Lilly And Company (http://www.lilly.com) John LaBore is the SAS and JMP Coordinator for Eli Lilly and Company, a leading innovation-driven pharmaceutical corporation. He is responsible for supporting use of SAS and JMP by Lilly staff worldwide. John has been a SAS software user for more than 20 years, and has authored numerous SAS technical papers for SUGI, PharmaSUG, SEUGI, and other SAS user group conferences.

Mitch Mitchell, Eli Lilly and Company (http://www.lilly.com) is a Sr. Systems Analyst for Eli Lilly and Company, a leading innovation-driven pharmaceutical corporation. He is responsible for supporting global medical information systems used in clinical trials for Lilly worldwide. Mitch has over 15 years of systems experience using various technologies in the Pharmaceutical, Financial, Automotive, Manufacturing, Telecommunications, and Insurance industries.

CONTACT INFORMATION

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

Randy Curnutt
Solutions Plus, Inc.
10401 North Meridian, Suite 300
Indianapolis, IN 46217
(317) 848-3081
rcurnutt@sol-plus.com
http://www.sol-plus.com

Michael Pell
Solutions Plus, Inc.
10401 North Meridian, Suite 300
Indianapolis, IN 46217
(317) 848-3081
mjpell@sol-plus.com
http://www.sol-plus.com

John LaBore
Eli Lilly And Company
Lilly Corporate Center
Drop Code 6335
Indianapolis, IN 46285
(317) 277-6387
jml@lilly.com
http://www.lilly.com

Mitch Mitchell
Eli Lilly And Company
Lilly Corporate Center
Drop Code 6323
Indianapolis, IN 46285
(317) 277-9632
mitchell_mitch@lilly.com
http://www.lilly.com

APPENDIX A

The generated PC SAS program can be written to a temporary file and then PC SAS can be run using the generated SAS program (all done in the Java method as shown below).

```
  /** Writes the PC SAS program to disk, then
runs it.
  * @param sasProgramString String of the SAS
program to be executed
  * @return int the return code returned by
the executed SAS program
  */
public int runSasProgram(String
sasProgramString) throws
TransportFailedException
{
    int returnCode = 0;
    try
    {
            //Write the file to disk
            File sasProgramFile = new
File("C:\\SAS\\GenProg.sas");
            BufferedReader buffReader = new
BufferedReader(new
StringReader(sasProgramString));
            PrintWriter sasProgOut = new
PrintWriter(new FileOutputStream(sasProgramFile));
            printDIStreamOnPrintWriter(buffReader, sasProgOut);
            Runtime rt = Runtime.getRuntime();
            sasProgOut.close();

            //Now run the program using PC-SAS.  This is equivelant to starting PC-SAS from a
            //DOS command line
            StringBuffer programSb = new StringBuffer("C:\\SAS");
            programSb.append("\\SAS.exe -NOSPLASH -ICON -SYSIN \"");
            programSb.append(sasProgramFile.getPath());
            programSb.append("\"");
            Process myProc = rt.exec(programSb.toString());
            returnCode = myProc.waitFor(); //wait here until SAS is done running
            SasProgramFile.delete();  //be a good neighbor and clean up
    }
    catch (Exception e)
    {
            throw new TransportFailedException("Error communicating with PC-SAS software.");
    }
    return returnCode;
}
```

APPENDIX B

The following is an example of a JCL job that contains a SAS program that uses SAS/Connect to transfer a SAS dataset to a UNIX server. It also provides a summary report about the transferred data's integrity. If the job detects an error, then it will transfer an error log rather than the data requested.

```
//MYUSERID JOB (,8305,S),'DAT35TestSasToUnix',MSGCLASS=T,CLASS=T
//*--------------------------------------------------
//JS010      EXEC PROC=SAS609,SOUT=T
//*--------------------------------------------------
//MVSIN    DD DSN=ABC_DATA,
//            DISP=(SHR,KEEP,KEEP)
//VALLOG  DD DSN=MYUSERID.F0545615.VALLOG,
//            DISP=(NEW,PASS,DELETE),
//            SPACE=(10796,(250,500),RLSE),
//            UNIT=WKDISK,
//            BLKSIZE=0,
//            LRECL=120,
//            RECFM=FB
//ERRNAME  DD DSN=MYUSERID.F0545615.ERRORS.TXTT,
//            DISP=(NEW,PASS,DELETE),
//            SPACE=(10796,(250,500),RLSE),
//            UNIT=WKDISK,
//            BLKSIZE=0,
//            LRECL=120,
//            RECFM=FB
//SYSIN    DD *
%macro sasToUx(userId=,
              pswrd=,
           inclause=,
           rowconst=,
               cols=,
               file=,
           targHost=,
           targDir=,);
%let unix=&targHost;
%let userId=&userId;
%let pswrd=&pswrd;
%let inclause=%upcase(&inclause);
%let rowconst=%upcase(&rowconst);
%let cols=%upcase(&cols);
%let file=%upcase(&file);
%let targHost=%upcase(&targHost);
%let targDir=%upcase(&targDir);
options remote=unix comamid=tcp nocaps;
options sgen mprint;
filename rlink 'syst.sas.sas608.ctmisc(datcpunx)';
signon unix;
```

```
PROC PRINTTO log=ERRNAME;
proc sql;
create table WORK.INV as
    select &cols
    from MVSIN.INV &rowconst ;

* Create the first comparison dataset  ;
create table WORK.valid1 as
  select nobs,obslen,nvar
  from dictionary.tables
  where libname = 'WORK'
  and memname = 'INV';

* Go to UNIX ;
rsubmit;

libname unixOut "/home/myUserId";

*Move the MVS work.&dsn file to Unix  ;
proc upload
  data=WORK.INV
  out=unixOut.INV;
run;

*Move the 1st comparison file to Unix  ;
proc upload data=WORK.valid1
  out=WORK.valid1;
run;

* move the transferred file back to MVS ;
PROC download data=unixOut.INV
     out=WORK.xfer;
run;

endrsubmit;

*Must be done on MVS to get the dictionary.tables;
proc sql;
create table WORK.valid2 as
  select nobs,obslen,nvar
  from dictionary.tables
  where libname = 'WORK'
  and memname = 'INV';

* Go back to UNIX ;
rsubmit;

* move the validation dataset to Unix ;
proc upload data=work.valid2
  out=work.valid2;

* Perform the comparison ;
proc compare  base=WORK.valid1
     compare=WORK.valid2
     out=WORK.vallog noprint;
run;

data _null_;
    set WORK.vallog;

(DATASTEP CODE TO COMPARE NUMBER OF OBSERVATIONS, OBSERVATION LENGTH, AND NUMBER OF VARIABLES DELETED
FOR BREVITY)

run;
endrsubmit;

proc printto print=VALLOG;

* Perform the comparison ;
proc compare  base=WORK.INV
     compare=WORK.xfer
     outstats=WORK.vallog;
```

```
run;

* Perform the PROC contents ;
proc contents data=WORK.xfer;

proc printto;

signoff unix;
%mend sasToUx;
%sasToUx(userId="myUserId",
pswrd="mypass",
inclause=
    and name in ('CLINVNO', 'COUNTRYC', 'GLBINVNO'),
rowconst=WHERE clinvno>"804";,
cols=%STR(CLINVNO, COUNTRYC, GLBINVNO),
file=inv.ssd01,
targHost=unixhostnamee@mycompany.com,
targDir=/home/myUserId);
run;
/*
//*******************************************************************
//*  IF JOB DOES NOT ABEND AND RETURN CODE <= 4 THEN RENAME
//*******************************************************************
//IFGOOD   IF (^ABEND & JS010.SAS.RC <= 4) THEN
//FTP      EXEC  PGM=FTP,PARM='(EXIT'
//OUTPUT   DD SYSOUT=*
//INPUT    DD *
  PASCAL
  myUserId
  mypass
  RENAME /home/myUserId/invval.logT invval.log
  PUT 'MYUSERID.F0545615.ERRORS.TXTT' /home/myUserId/invERRORS.TXT
QUIT
/*
//*******************************************************************
//*  ELSE TRANSFER THE ERROR LOG
//*******************************************************************
//ELSE     ELSE
//FTP      EXEC  PGM=FTP,PARM='(EXIT'
//OUTPUT   DD SYSOUT=*
//INPUT    DD *
  PASCAL
  myUserId
  mypass
  PUT 'MYUSERID.F0545615.ERRORS.TXTT' /home/myUserId/invERRORS.TXT
  DELETE /home/myUserId/invval.logT
  DELETE /home/myUserId/inv.ssd01
  QUIT
//ENDIF    ENDIF
//
```