Paper 49-26

Develop in GUI, run with 'grunt'

David Johnson, DKV-J Consultancies, Holmeswood, England

ABSTRACT

Development and testing of code is usually simplified by working in a Windows or similar Graphical User Interface (GUI) environment. The results of individual steps can be checked, and additional analysis performed on the data before further manipulation is performed. Problems in the code can be resolved and a final code version developed that is both tuned for performance, and robust enough for the production environment.

However, the quantity of data that may be manipulated in a production environment is sometimes too large for the Windows platform. So the production process may need to be run on a mid range or mainframe platform. Is it then practical to develop a major application on one platform and expect it to function effectively on a larger scale data source on a different platform? What issues may arise in the development, and how may they be overcome?

Having processed the data in our batch environment, have we any options to simplify the reviewing of the logs and output generated by our batch?

INTRODUCTION

We'll look at these questions in the context of a Data Warehouse developed for a Management Information System in 1999/2000. As it is appropriate, we'll refer to a second MIS under development at the time of writing.

The processes for the first system, we'll call it '*Florida*', were developed on the SAS® System version 6.12 TS020 for Unix and the SAS® System version 6.12 TS065 for Windows.

The SAS/CONNECT® product was not initially available, but the impact of its later delivery is explored as well as the impact of Version 8e trials on Unix and Version 8.0 trials on Windows.

The second system, we'll call it '*California*', uses a front end developed in SAS/AF® and the SAS® System version 6.12 TS065 for Windows. Access to the DB2 Data Warehouse is through a SAS/CONNECT session to the SAS System version 6.09E TS470 on MVS.

THE ENVIRONMENT

The '*Florida*' Windows environment was a thin client running on a MicroSoft Terminal Server (MSTS). It accessed the SAS session on the Unix server through a Telnet application. Additional applications were not available for the MSTS platform due to year 2000 constraints, but a Web browser was available that was later used against a Web server set up on the Unix machine.

The Unix platform supported the production delivery of a data Warehouse of Sybase databases, viewed by clients through a Business Objects front end. 58GB of data space was initially made available for the SAS process. This received transport files built in a mainframe SAS process and copied to the Unix machine through a third party product. The Unix SAS processes created data sets and loaded these to the Sybase tables.

Because of Year 2000 constraints, the data structure and content provided from the mainframe was "(structurally) frozen" early on in the project. All data cleansing procedures were to be performed in the Unix 'format and load' processes.

Most of the data feeds came from a Data Warehouse feed built for an earlier (aborted) project. As a result, there was a strong but misguided expectation that the data being provided to the Warehouse had already been cleansed. Consequently, data reconciliation issues were expected to be minimal.

This was not the case and some 8% of the data were identified with one or more errors. The errors appeared in certain smaller segments of the business. Unfortunately, these areas were of particular business significance; both in financial impact and to the decision making that the Warehouse was expected to support.

While programs were tested with small segments of the data, these were in strata selected by year and product type. Processes that worked with test data failed for size and data type issues when the full scope of data was processed.

As protection against the anticipated failures, a series of test reports were to be provided with each run. These reported the distribution of data by key 'code-table' fields (including product type) and date ranges. These tabulations provided an immediate point of reference for data queries from the project 'business reconciliation' team. However, these were very substantial reports, and a method of indexing them was needed. The method also had to be suitable for 'thin client' access.

The solution was to create HTML pages from the log and list files. Customised HTML macros were created to facilitate the inclusion of formatting that would highlight errors and issues, as well as provide a simplified path into the output files.

BUT IT DOESN'T WORK THE SAME ON UNIX

The key issues with developing on Windows for Unix were:

- the different Operating system environments,
- the macros used within the programs,
- the availability of Unix data on Windows and
- the limited processing power available on the Windows machine.

The first two issues are closely related, since many of the macros used in the process interfaced with the Operating system. These surfaced such information as the names and location of files; the availability and creation of directories or folders for output files; the name of the user logged on; the resource availability for the process and the method for sending email notification of process milestones.

Two distinct sets of macros were maintained, one for each operating system. These had common naming and parameters and surfaced information in a similar manner. So for instance, a file search utility macro that used a Unix operating system shell call used the same parameter as the one using a Windows API. Each macro created a data set of file names and attributes that was similar in the structure of key fields.

We'll look at the file search macro that was central to some of our key processes. It was the key to the detection of the log and list files created by the elements of the batch process, and the identification of the appropriate target directory for HTML files created by the log analysis macros.

Significant effort was needed early on to identify the data that could be surfaced through the Windows API so that the smaller subset of data provided from Unix could be matched to it. While much of this was simple, the permission flags ('Read Only', 'Archive', 'System', 'Hidden', 'Compressed' etc) surfaced by the Windows API were substantially different to the Unix permissions ('Read', 'Write, 'Execute' independently set for each of 'Owner', 'Group member' and 'Global user').

Ultimately, the 'permission mapping' problem was solved when these decisions were made:

- Create a ‘SASUSER’ user permissions group on Unix,
- Create a 'SASADMIN' user and make it the principal member of this group
- Attribute ownership of all SAS data and files to ‘SASADMIN’
- Attribute group membership of all SAS data and files to SASUSER.
- Deny global access to any 'SASUSER' files
- Run all production processes when logged on as 'SASADMIN'

This solved the problem, since the most important permissions became of the owner. The Read/Write/Execute attributes could be mapped to equivalent status using the 'Read', 'System' and 'Hidden' file flags originally defined from the DOS 'File Allocation Table' model. (This FAT model is 'upwardly compatible' with the NTFS model.)

Some other file attribute data was also lost, since the file creation, modification and access dates that could be retrieved from the Windows NT Operating System could not be matched on the Unix system. The modification date was chosen as the best matching data set parameter that also was the most descriptive and useful.

Most macros that were run on the Windows Operating System used Windows APIs. The process is shown in Figure 1.

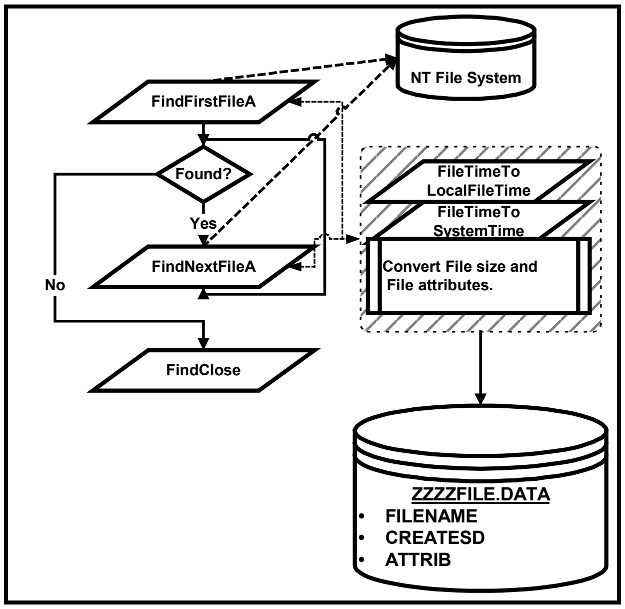

Figure 1

Aside from creating data sets, they would sometimes populate SAS macro tokens. Consequently, a policy was created very early on to define the naming conventions for all Macros, macro parameters, macro tokens and data sets to ensure naming contentions did not occur.

The Unix macros (see figure 2) mostly used commands 'Piped' to the Operating System. This required definition of a filename for the Pipe, and a similar convention was created for file and library references. These standards were rigidly enforced. Failure to do so would have threatened the whole process.

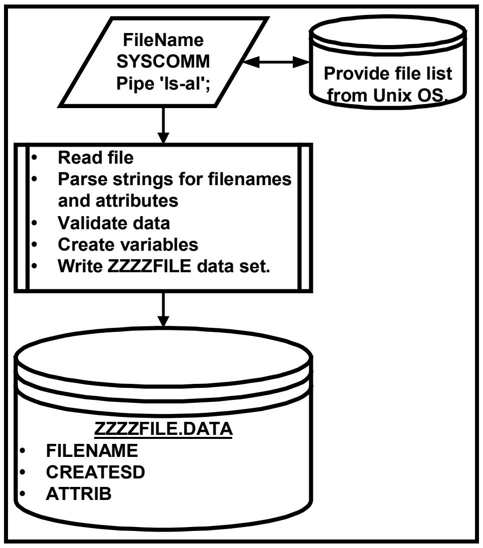
FileName SYSCOMM Pipe 'ls-al';
Provide file list from Unix OS.
Read file
Parse strings for filenames and attributes
Validate data
Create variables
Write ZZZZFILE data set.
ZZZZFILE.DATA
FILENAME
CREATESD
ATTRIB

Figure 2

The macros are discussed in greater detail in the SUGI 26 paper ‘*Coding across the boundaries’*, paper number 280-26. Samples of the macros are also available on the author's company website at http://www.dkvj-cons.com/sugi_26_p280.

IS CODE DEVELOPMENT EASIER IN WINDOWS?

The benefits of the GUI interface for SAS development are most strongly demonstrated when the alternative is a limited interface. The original method of coding for Unix involved a 'Windows Telnet' session. This was initiated on a MicroSoft Terminal Server and connected to a SAS session on the Unix machine.

The Telnet session was then made available to a thin client desktop, with very limited functionality. Bugs in the system were common. For instance, the Telnet session would always crash with an 'invalid memory address error' if more than one screen was copied and pasted from the Unix display to a Windows application.

Browsing of output was on a 25 row by 80 column screen, which produced some very hard to read print output on the wide data sets in the data warehouse. Function keys were also limited.

When one machine with a full version of Windows became available, development moved ahead quite a lot faster. A SAS for Windows session could be used to read and edit programs and output, although transfer of files between the two platforms was by FTP scripts written in a text editor.

The SAS System Viewer® was installed on the standalone machine. This provided multiple edit and browse windows, making comparison of multiple files as easy as with ISPF on the mainframe. (Most of the SAS team had a mainframe development background and equivalent expectations.)

HOW DO YOU SAFELY SYNCHRONISE THE PROGRAM VERSIONS?

Increased use of the Windows platform for code testing and development meant files had to be synchronised between the Unix and Windows platforms with a process more sophisticated than hand crafted FTP script files.

A SAS program was written which included parameters for the source and target directories. This allowed a directory to be selected from among the individual directories maintained for the project.

The program names were stored as rows in a SAS table, and FTP scripts were then written by a program reading the program table and embedding the file names in a series of command strings. As the need for more frequent synchronisation of different libraries, an application in SAS/AF was built with the programs stored as Source entries behind the push button widgets. The screen layout is represented in Figure 3.

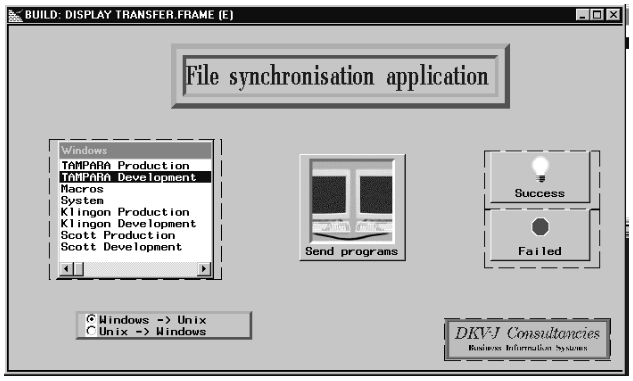

Figure 3

The list box on the left represents the following directory structures available for synchronisation:

- SAS System files (Multiple autoexec and configuration files)
- Macros stored in the SAS System 'SASAUTOS' directory
- Development environments for each of three projects
- Production environments for the same three projects.

The Radio box below it allows for the transfer direction (To or From Unix) to be selected.

The Icon in the centre initiated the file transfer.

The two icons beside it have been moved for clarity. In the Production application the two were overlaid. A 'successful' file transfer would initiate the '_swap_in_' method for the icon representing 'success'. Alternatively, the 'Failed' icon would appear on the desktop.

LOGGING THE PROCESS.

Building the data warehouse was taking upwards of 40 hours in the early stages of the project. (This rose to just over 50 hours on occasions later on.) This presented a series of problems including:

- extended contention with enquiries on the production system,
- delays in starting the build process compounded in delays in delivery of the data warehouse,
- serious problems with data or the process that required a process restart set the delivery of data back by two days at a time.
- Identifying the cause of the problems may take many hours as data analysis was performed on data sets up to 8GB in size.
- Similarly, reading the results of the analysis was difficult on the Telnet screens.

Indeed, the decision was made on a number of occasions to completely rerun the batch. This happened because the time to fully analyse the successful components and synthesise a partial rerun was likely to extend beyond the batch time. In addition, there was a risk of not correctly setting up the interdependent data sets required for some parts of the build. A failure may then cost three or more days rather than two.

To reduce the impact of data or process problems, the batch was modelled with the following criteria;

- all Notes, Source and Source2 messages were retained,
- explicit data tests were inserted within the process log,
- 'critical error' traps were placed at key points of the process to halt the process if faulty data was found, and
- test prints of data and frequency analysis of all codetable fields were routinely performed and captured to the batch output file.

Retention of all process Errors, Warnings and Notes meant the logs could be reviewed in detail for any indication of data faults.

The inclusion of data tests meant 'program note' messages could be generated and reviewed along with the SAS System generated messages. Program notes were generated for many reasons, some of them were:

- data set is unexpectedly empty
- n% of data has been removed from the warehouse load due to dates missing in critical fields
- the total number of rows apportioned between three data sets does not equal the number of rows in the source data set.

If these were critical problems, then identifying the error early meant the process could be stopped quickly. This saved time being wasted on a warehouse build that may need to be abandoned. It also flagged the error point to the supervisor of the batch process.

At the core of the data set tests was the ability to retrieve the number of observations in a data set. The following macro was used for this application. Note that the macro is not suitable for SAS/AF applications, where a deletion flag may be set against an observation, but the observation is not physically deleted immediately. In this case, the data set attribute (ATTRN) to test for is the number of logical observation: NLObs.

```
%Macro GetDNobs( MDSName = ,
                  /* NObs in this data set */
                 MMacVar =  )
                  /* Place NObs in this macro */
     / Des = 'Get DSN obs count to macro';

  %Local  DSid;
  %Global &MMacVar;

  %Let &MMacVar = -1;
  /* If dsn does not exist, force value=-1 */

  %Let DSid = %SysFunc( Open( &MDSName) );

  %If &DSid  %Then
     %Let &MMacVar  =
          %SysFunc( Attrn( &DSid, NObs) );
  %Else %Put
        Macro Call error:
        The specified data set
        &MDSName cannot be opened;

%If &DSid  %Then
   %Let SysRc  = %SysFunc( Close( &DSid) );
%If &DSid  And  &SysRc  %Then %Put
        Macro Run Time Error:
        Data Set &MDSName
        was not closed properly;

%Mend GetDNobs;
```

Now counts could be retrieved and tested within the code. Unlike our earlier example of cross platform macros, this particular macro works on any platform where the SAS System version is equivalent to The SAS System version 6.12. On MVS, that is

Version 6.09E. Earlier versions did not have the %SysFunc() function available.

Here is an example of two tests on data sets. In the first, the data set does not exist.

```
48   %GetDNobs( MDSName = SASUSER.UKIRE,
MMacVar = NObsDS2 );
Macro Call error: The specified data set
SASUSER.UKIRE cannot be opened
49
50   %Put How many Obs did we find? &NobsDS2;
How many Obs did we find?  -1
```

In this call, the log explicitly reports the problem; that we have either coded a non-existent data set into the program, or a data set we expected, has not been created.

Now let's try that macro on a data set that does exist.

```
51   %GetDNobs( MDSName = MAPS.UKIRE, MMacVar
= NObsDS2 );
52
53   %Put How many Obs did we find? &NobsDS2;
How many Obs did we find?  12986
```

In our second call, we surface the number of observations. We can now test one or more others and similarly test and compare their observation counts.

DATA ANALYSIS

We also routinely created test prints on all data sets that would be populating the Data Warehouse. These provided a means of quickly checking a random selection of the data. Our test prints were created from another cross-platform macro that included the following code:

```
Proc Print Data = LIB.DSN( Where =
           (RanUni(0) < 40/&MNObs) );
Run;
```

The number of observations was retrieved from the macro described above. Then up to 40 observations were randomly selected from the data set. Naturally, a smaller report was produced if the data set had fewer than 40 observations. (In fact, the number of observations does vary slightly around 40 because of the distribution of the numbers provided by the pseudo random generator. But an adequate number of data set observations were presented for visual assessment of the data. A very informative explanation (by Dr John Whittington) of this method can be found on SAS/L.)

The reason that the first 40 observations weren't selected instead is that the sorted data set may have observations with missing values in key fields. Similarly, the last observations may have all fields populated. Either is misleading in evaluating the data. In this manner, larger data sets were likely to have different observations reported each month because of the random selection. Reviewing the samples over a number of months could be very informative.

Data quality was also evaluated through frequency analyses, which included summary tests of numeric data:

- dates formatted to month level and
- currency values formatted with scientific notation (Ew.) formats

These resulted in summary tables of a manageable size, which were immediately available to test the 'reasonableness' of the data.

With such large data sets, writing and running data test code could take an excessive amount of time. Similarly, since the test code was usually being written on the Windows platform, either the code and output had to be exchanged between the Windows and Unix platform, or sample data had to be made available to the Windows SAS session. Having such summary tables available routinely meant basic analysis of data problems could start immediately a problem was identified.

In addition, since log and output files were being retained for many months at a time, it was possible to compare current results with the saved results of previous months. Understanding the timing of the emergence of a problem also proved very valuable in narrowing down the source of a data issue.

Indeed an unusual gross monthly revenue total that emerged in February 2000 was immediately suspected of being a Year 2000 error. The (routinely generated) cross tabulations of revenue amounts by product type and month highlighted the source of a problem. The table might have looked like figure 4 below. This table has an abnormally low claim rate for 'Household' policies in 'March 2001'.

Claims * Lodgement	Household	Marine	Automotive
Jul2000	327	12	635
Aug2000	483	4	702
Sep2000	395	9	598
Oct2000	344	13	674
Nov2000	396	11	639
Dec2000	465	8	716
Jan2001	503	14	685
Feb2001	288	12	648
Mar2001	2	16	627

Figure 4

We might suspect that the extract data from the claims system dealing with household insurance was at fault. The real issue was not unlike this table and demonstrated the problem had arisen in a particular production system for a small set of products. Thus, the new MI system proved its worth very early in its life in an unexpected area.

The tabulations also proved of assistance when data preparation in the SAS process was complete, but loading and indexing of the MI tables was some days from completion. An urgent user question on 'revenue by product over time' was answered from the summary tables.

WHEN ‘REVIEW’ WAS AN ‘E-VIEW’

The log and list files from the batch were deployed in a directory structure with folders identified by the batch run date. These had been moved from the Unix environment to the Windows environment with an FTP process written by the SAS batch from an analysis of available files. However, interpretation of the output was still a time consuming task with such large files.

A decision was taken to restructure these files into HTML. This would allow their deployment in an Intranet hierarchy, and allow them to be browsed by software other than a text editor, or the SAS viewer. (The deployment of the SAS Viewer generally was still held up due to the year 2000 change freeze.)

For users logging on remotely to the corporate server, the HTML files would be readily visible. In time they could also be made available to users outside the SAS team of the project.

The process of building the Web pages is covered in greater detail in the SUGI 26 paper '"*Just the facts Ma'am" meets "Is it Safe*"'. It is available from the conference proceedings as paper P168-26. Supplementary material is also available from the author's website at http://www.dkvj-cons.com/sugi_26_p168.

The solution discussed in that paper was to write macros that read and analysed the batch files. Custom formatting was applied by detecting characteristics that included the following:

- page titling and footnoting in output,
- Error messages in logs,
- Program generated notes on data irregularities
- Remote system specific from the DBLoad Procedure and
- Data/Proc step boundaries in program files.

The SAS programs were also being copied back to the web server so that a snapshot was available of any programs that may have been changed between batches. The formatting was applied through the use of cascading style sheets, which allowed better formatting to be developed over time, while not delaying the urgent task of delivering the batch logs.

After some eight months, an additional data mart was developed for the project. The existing 'web-log' processes were found to be of such value and utility that the new project used the same code for its reporting. Needless to say, this satisfied the client who was very keen for consolidation of processes and software for their corporate projects.

The HTML build process is depicted in figure 5 below.

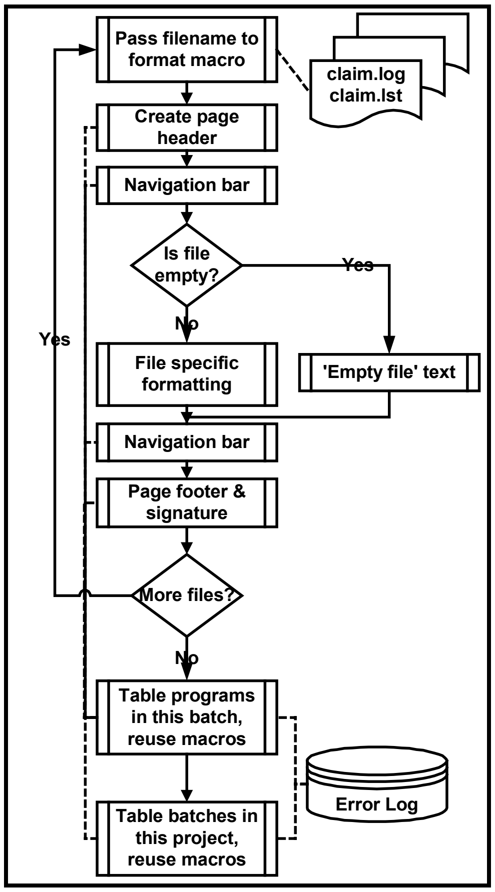

Figure 5

The conversion of the text files into HTML pages involved a SAS process to read and parse the files. This parsing of files was a very early characteristic of the batch. Through the file parsing, counts could be maintained of Errors and Warnings in the logs. Once all files had been parsed, a condition code could be created based on the log analysis.

This condition code was used to write a file on the Unix server. The name of the file immediately alerted any user on the Unix server to the success of the process. Reading the file would also disclose a summary of the problem(s) identified in the batch. This allowed very quick decisions to be made on whether to hand over the SAS generated data to the MI database team.

Later in the process, this was extended. The condition code was used to populate an email message sent to the people most concerned with the MI batch process. This included information on the start and end times, the total run time and the verdict on the batch from the log analysis. When the testing of 'control totals' was added to the batch, and record counts and revenue values were being reported to the SAS team, considerable interest was aroused within the business team. These values were available 2 to 3 days prior to the final availability of the MI and could provide an early 'confidence indicator' to the business manager.

The overview of the batch process is depicted in figure 6.

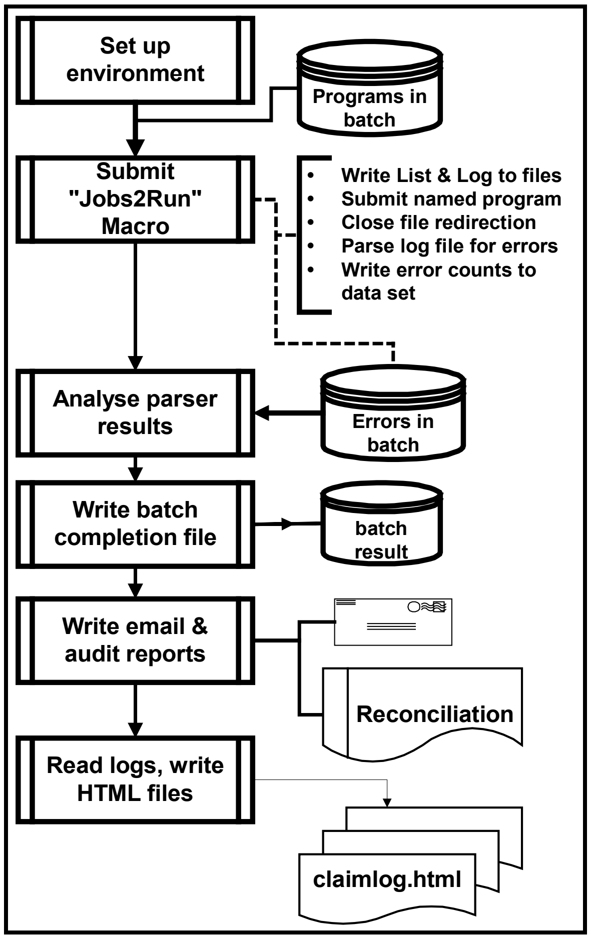

Figure 6

Email processes were then added to the batch roundup to report current totals to the business team. These values were compared to previous months and a basic analysis provided on the reasonableness of the values produced. In this way, the business team was predisposed to accept the data when it was finally available, and their tests of the values were completed more quickly. This ensured the delivery of the MI at an earlier date to the company.

WHAT DID SAS/CONNECT ADD TO THE PROJECT?
Shortly after Phase 2 of the project commenced, SAS/CONNECT was licensed for the Unix machine, and standalone machines deployed to replace the MSTS machines the SAS team had been using. The availability of 'RSUBMIT' and 'Remote Library Services' meant Unix data could be tested with code written on, and submitted from the Windows platform.

The old process for FTP transfer of (program and batch) files through the AF screen was also modified to use the UPLOAD and DOWNLOAD procedures. This also meant that any problem with transferring a file could be identified immediately. This had not been possible with the FTP script process.

The process was not without its' cost; the licensing fee being the first hurdle. However, the productivity improvements demonstrated during the product evaluation very quickly overcame that problem.

In testing code, a piece of code may have been modified with an 'RSUBMIT' command and these were occasionally not removed before the code was saved. This lack of diligence meant the program would fail when it was run from the Unix machine.

It was also found that if a piece of code was highlighted and the RSUBMIT command issued from the command box or a PF Key, then the whole program editor would be submitted.

Finally, the use of Connect on Version 6 meant that the host (remote) and client (local) sessions ran synchronously. If the user passed code without due care, a long host process would lock the client SAS session until the process finished. (Accidentally submitting all the code for a 15 hour batch was usually only done once, but it was easy to forget to include an 'Obs=' constraint on a Proc PRINT against a data set with 20,000 observations.)

WOULD VERSION 8 HAVE CHANGED THE APPROACH?
With
- change freezes,
- a more urgent need for SAS/CONNECT,
- considerable unexpected data problems and
- a focus on development,

the project was slow in looking at SAS Version 8.

When this happened it was late in the second phase as a result of trying to tune the batch process and bring run times down. A second Unix server had been commissioned as a development platform, and the SAS installation was restructured to better utilise the machine resources.

Consequently, first tests of the batch on the Development server brought run times down from almost fifty hours (at that time) to slightly more than 20. (Much of this was due to better-defined DASD structures, and the implementation of large file structures to break the 2GB-file size limit.)

In the closing parts of the project then, the need was to trim as much time as was possible from the overall batch. The program modules that made up the batch were analysed and the six longest running pieces identified for attention.

Of these the second largest was the process for loading the Sybase transfer tables. The SAS System version 8 changes to the Libname engine to allow direct access to non-SAS tables looked promising. It was expected that this would allow faster appending of data, over the DBLOAD procedure.

Unfortunately this wasn't to prove true. The source tables for the preceding data creation processes were all version 6, and changes to those at that time was impractical. Consequently, the new load process made a copy of a Version 6 table from the existing library in a Version 8 library.

Then the variable names were altered with a macro that read the file header and applied the longer variable labels to the variable names. (These variable labels matched the names of the variables on the Sybase table.) Here is an extract from the log where the copy and rename took place.

```
238  Proc Copy  InLib = FLORIDA
239            OutLib = LOAD;
240    Select A01;
241  Run;

NOTE: Copying FLORIDA.A01 to LOAD.A01
(memtype=DATA).
NOTE: There were 432293 observations read
from the data set FLORIDA.A01.
NOTE: The data set LOAD.A01 has 432293
observations and 37 variables.
NOTE: PROCEDURE COPY used:
      real time           4.51 seconds
      user cpu time       2.47 seconds
      system cpu time     1.66 seconds
      Memory                            130k
      Page Faults                       0
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        30
      Involuntary Context Switches      77
      Block Input Operations            3
      Block Output Operations           14437


242  Proc DataSets Lib = LOAD  NoList;
243    Contents Data = A01 NoPrint
Out=WORK.CONTLOAD( Keep = NAME LABEL);
244  Quit;

NOTE: The data set WORK.CONTLOAD has 37
observations and 2 variables.
NOTE: PROCEDURE DATASETS used:
      real time           0.04 seconds
      user cpu time       0.01 seconds
      system cpu time     0.01 seconds
      Memory                            74k
      Page Faults                       0
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        4
      Involuntary Context Switches      2
      Block Input Operations            0
      Block Output Operations           6


245  Data _NULL_;
246    Set CONTLOAD End = LAST;
247    If _N_ = 1  Then Call
248      Execute(' Proc DataSets Lib = LOAD
NoList; Modify A01;');
249    Call Execute(' Rename ' || NAME || ' =
' || LABEL || ';');
250    If LAST  Then Call Execute('Quit;');
251  Run;

NOTE: There were 37 observations read from
the data set WORK.CONTLOAD.
NOTE: DATA statement used:
      real time           0.01 seconds
      user cpu time       0.01 seconds
      system cpu time     0.00 seconds
      Memory                            68k
      Page Faults                       0
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        0
      Involuntary Context Switches      2
      Block Input Operations            0
      Block Output Operations           0


NOTE: CALL EXECUTE generated line.
1   +  Proc DataSets Lib = LOAD NoList;
Modify A01;

12   +  Rename ADDR1
= address_1
13   +
;
NOTE: Renaming variable ADDR1 to address_1.

76   + Quit;

NOTE: PROCEDURE DATASETS used:
      real time           0.03 seconds
      user cpu time       0.02 seconds
      system cpu time     0.00 seconds
      Memory                            117k
      Page Faults                       0
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        1
      Involuntary Context Switches      1
      Block Input Operations            0
      Block Output Operations           5
```

Finally, the APPEND procedure could be performed, since the variable names now agreed. Here is a log excerpt where version 8 tests were run on the load process.

```
899  RSubmit Host8dev
NOTE: Remote submit to HOST8DEV commencing.
899                ;
372  Libname  MYDBLIB   Sybase     User =
sasadmin
373          Password = XXXXXXXX  DataBase =
florida_download_test
374          Server = MYSERVER    SCHEMA =
dbo
375          Max_Connects = 25    PacketSize
= 4096
376          Read_buffer = 100;
NOTE: Libref MYDBLIB was successfully
assigned as follows:
      Engine:        SYBASE
      Physical Name: MYSERVER
377
378  Proc Append Base = MYDBLIB.agency_ins
379             Data = LOAD.A01( Obs =
1000);
380  Run;

NOTE: Appending LOAD.A01 to MYDBLIB.agency.
NOTE: There were 1000 observations read from
the data set LOAD.A01.
NOTE: 1000 observations added.
NOTE: The data set MYDBLIB.agency_ins has .
observations and 37 variables.
NOTE: PROCEDURE APPEND used:
      real time           10.32 seconds
      user cpu time       1.05 seconds
      system cpu time     0.10 seconds
      Memory                            108k
      Page Faults                       0
      Page Reclaims                     0
      Page Swaps                        0
      Voluntary Context Switches        1034
      Involuntary Context Switches      146
      Block Input Operations            0
      Block Output Operations           9

NOTE: Remote submit to HOST8DEV complete.
```

Unfortunately, while the copy and rename process took no more than 60 seconds on even the largest tables, the append process was taking a lot longer than the equivalent DBLOAD procedure. This test of 1000 rows, and a series of some forty similar ones with parameter and row count changes, demonstrated almost exactly 100 rows would be added each second. That scaled to over 72 minutes for this table. The DBLOAD Procedure was loading the table in less than 15 minutes.

No amount of work within the team or by SAS Customer Support in England and Europe ever established the cause for the unexpectedly (and unacceptably) long times. Version 8 for the batch process was abandoned since it was not seen to contribute any significant value to the project at that time.

Tests of the version 6 code run under version 8 were generally good, although a couple of issues emerged. The major one was the file format produced by the DATASETS procedure. The output was used for updating other management data sets that held record counts and update values.

However, the length of a variable name increased from 8 bytes to 32 bytes in Version 8, and the structure of the output data set changed accordingly. Appending incompatible data sets was not possible, and making the change to Version 8 would involve code changes that were a threat to the project and infrastructure changes that would be difficult to back out.

There were better results achieved when the development processes were examined. The ability of the remote submit processes to run asynchronously was a significant gain, and the colour coding of the new editor promised some benefits in better code production.

The Telnet processes were included in the project documentation to provide an alternative means of amending code and reviewing data if all else failed. It was also a reasonably uncomplicated method for dial-in access to the LAN.

Unfortunately however, this method would not function with the SAS System version 8. This meant a dial up session from a machine without SAS installed could not access a version 8 interface to the Unix machine. Tests of XWindows and similar emulators that were supported by version 8 also had installation and performance problems and they were abandoned when the costs of continuing threatened to exceed the benefits.

CONCLUSION

This paper summarises parts of the ***Florida*** project that encompassed a year. It began in challenging times, with limited equipment and flexibility. But the commitment of the project team, and of the company, to the benefits of the MIS made it the success that the company earned.

I think it demonstrates that a flexible approach to the challenges that arise will always find some solution to the difficulties that are encountered. At times, the development of the project was akin to modelling a motor car with a model railway electric motor, and then expecting the full sized version to function like a scaled up version of the model.

In data warehousing especially, there are so many variables and unknowns hidden in legacy system data, that expecting scaling of sample data to the final product is often unrealistic. Yet the need remains to deliver the project benefits to the business, and build a robust system that can easily be handed on. To achieve this goal, a cautious 'model- test- refine- scale- test- deliver' strategy is probably still the best approach.

In the absence of some of the elements that eventually produced a very powerful SAS development environment, the team produced and integrated a set of pieces into a robust system. It demonstrated that you can 'develop your warehouse with a GUI front end', and then move the process to another platform to 'run it with grunt'.

The value of the object-oriented approach was proven, when the lessons of ***Florida*** were of almost immediate relevance to the second MIS required by the company. Objects built for one could be used on a second and third project, with the benefits of faster delivery time and lower cost.

Indeed, as the postscript below discusses, the original basic 'object toolkit' proved its value again and showed that SAS programs can be developed on one platform and run on another.

It is possible to design SAS objects that perform similar functions, each on entirely different operating systems. These can be coded into programs that will work as reliably on one platform as on another. This opens the way to develop an application on the most suitable development platform, often a GUI one, and then run it on a higher-powered server.

POSTSCRIPT: THE *CALIFORNIA* PROJECT?

On the surface, the new project could not have been more different.

- Extensive use was made of SAS/AF in the development of the user screens.
- With the Data Warehouse moved from a Unix mid range server to MVS 'big iron', many of the performance problems of the earlier project will not be seen.
- The Data Warehouse too is updated on a daily basis, unlike the monthly process seen previously.

Yet one of its major processing functions is to build code dynamically from A/F screen selections, and pass the code in a remote submit block to the mainframe. In the absence of a Data Warehouse until the very late stages of the project, early testing of this code was performed on sample data sets created on the NT servers. This meant we again faced the potential problem of code running on one environment, and then needing to be run on another. Once more, the cross platform macros came into play.

The new demands, and a new set of problems, meant the Windows API macros were deployed, and a series of additional ones written, as well as old ones modified for new demands. The authors' original MVS macros, not seen in three years, were dusted off, loaded and worked with almost no changes. Similarly, the moved code worked on its new home.

One of the possibly surprising outcomes though was the need for log parsing again. The A/F application was designed to function without the need for the user to read logs, so a reasonable piece of work was performed in extending the log parser process. The new functionality reads the log and then selects from a standard set of error messages, to deploy the message through a Windows Message Box. The user of the front-end screens doesn't even see a log.

The use of email also came to the fore when it was identified that certain critical conditions with the system needed to be reported by email to users. These conditions included:

- reports of users locked out (an audit initiative),
- check reports of data invalidation to prevent false reports being accidentally used,
- summary reports of financial values moved day to day
- summary reports of 'accounts' flagged for errors overnight
- Successful Data Warehouse update and batch completion

TRADEMARK CITATION

REFERENCES

Barron, David L & Hemedinger, Chris (1996), "Accessing Dynamic Link Library Routines with the SAS System for Windows", *Observations: The Technical Journal for SAS Software Users*, First Quarter 1996.

Johnson, David H (1997), "DLLs, APIs and SAS", SAS Melbourne User Group.

SAS Institute Inc (2000 et al), "SAS Companion for the Microsoft Windows environment", SAS On Line documentation, Cary, NC, SAS Institute Inc.

Johnson, David H (2000), "Have SAS, will travel", SAS European User Group International, Dublin, SAS Institute Inc.

ACKNOWLEDGEMENTS

My clients who have asked for the robustness and functionality that required this development to take place.

The SUGI committee and especially the Applications Development section chair, Greg Barnes-Nelson, for supporting the production of this paper.

SAS for their documentation and the Observations publications which provide a forum for sharing ideas and experiences.

The helpful people who post ideas and assistance to SAS/L, the global SAS discussion list you can access at http://www.listserv.uga.edu.

Last but not least, my family for their support and patience.

CONTACT INFORMATION

Your comments, suggestions and questions are valued and encouraged. Please contact the author:

David Johnson
DKV-J Consultancies
C/- 'Bonds Cottage,
Holmeswood Rd
Holmeswood nr Rufford
Lancashire England L40 1UA

Work Phone:	+44 (0)7080 81 8399
Fax:	+44 (0)7092 25 9556
Email:	sugi49@dkvj-cons.com
Web:	http://www.dkvj-cons.com