Integrating SAS™ Software into a legacy system or How to keep a foot in the 21st Century while dragging along the 19th.

Dave Hrivnak, Eastman Chemical Company, Kingsport TN

ABSTRACT

Acid Division at Eastman Chemical Company, integrated 4th Generation Languages (SAS™), data warehousing, concepts of Object Oriented Programming System (OOPS), and client-server concepts into our "legacy" Manufacturing Information System (MIS). This approach gave us a 5-fold improvement in productivity and substantial new functionality at a minimum cost. We focused on relatively low tech, but effective, uses of the tools to integrate newer technology with the old. This information would be helpful to managers and beginning SAS programmers.

THE CHALLENGE

The fast pace of technology is very much a dual edged-sword. It is great to utilize new technology for the productivity and other benefits it brings. However, it is frustrating to know that as soon as a system is in place it is obsolete. In the chemical industry, management plans to use capital assets for many years. We have manufacturing systems built in the early 30's still producing good product. So breaking the news to management that the seven year old computer and fourteen year old software are obsolete is not received well. The challenge

is to meet changing business needs, keep productivity up, and to keep the programming staff interested, without significant capital investment or major changes for the users. We believe we met that challenge by achieving a good balance between keeping the old and integrating the new.

COMPANY BACKGROUND

EASTMAN CHEMICAL is the 10th largest chemical company in the US with annual sales of about $5 billion. Our headquarters and largest plant site are located in Kingsport TN. The plant site is home to over 12,000 employees. For the first seventy-three years of our history we were a division of EASTMAN KODAK but two years ago we split to form our own company. Kingsport is a manufacturing site with eight manufacturing divisions. Each division has a VAX™ based custom written Manufacturing Information System (MIS) to control and monitor product quality. The MIS's were written in the 80's to enable us to implement statistical quality control, a cornerstone of our quality management system. This has proved very worth while as our high product quality keeps sales strong even in cyclical downturns. In 1993 we were the first and only chemical company to ever win the Malcolm Baldridge National Quality Award.
DIVISION/SYSTEM BACKGROUND

I work in the Acid division where we produce ten's of millions of pounds of 175 different industrial chemicals per week. Our division had the first MIS written at the company in 1982. It was and still is primarily a laboratory management system. It was based on VAX/PDP technology in the days when 200mb of storage was considered huge. Testing and storing more than 2,000,000 analytical results a year led the programmers to create very cryptic and non-normalized data structures. Here is a small example of the data compression used.

There is a history data file for each quarter of the year, so by knowing the file, one knows the year and quarter. The date was the sequential day of the quarter 1-92 and the time was kept to the hour with A being the start of our day, 7am. So with 3 bytes one could calculate the date and hour of the sample. So 31C would translate to May 1, 1995 at 9:00.

The logic is ingenious, but it is a nightmare to program, maintain and change. To make matters worse up to fifteen different data types were kept in the same physical file. Because of this one must read part of the record to find out how to read the rest of the record. I'm not sure why this was done but I know most older programmers have struggled with this. The file structure is the major reason program changes took so long to carry out. Seemingly simple changes could take months, as one had to program around data inflexibilities.

However, from the perspective of the users the system worked well. Using the information from the system, the quality of our products improved significantly, while lab and quality assurance productivity improved. Users enjoy and are well trained in the simple text interface and use the system extensively as we have over 1200 log-ins per day so we have little incentive to upgrade to a graphical interface. We have an extensive base of text based terminals as over 95% of our 750+ users are tied to "dumb terminals" or non-networked PCs. With over 350 such devices, and users spread over several square miles in twenty-two buildings, upgrading the desktop hardware to a graphical interface becomes a very expensive proposition.

Programming productivity was the only significant problem. For example in a popular data retrieval message, when one enters a sample point, and a start and end-date, it takes 2040 lines of Fortran code to read the data from the files and bring it into the computer's memory. That is a 36-page program and NO analysis is performed! So when the business situation requires a change it would often take weeks to implement what the users thought would be a simple change. This led to frustration to the point that many users simply would not ask for enhancements because the answer was, with the backlog of projects, it would be years before completion.

However, business situations are not static and new functionality is needed. The system lacked any summary functions to track quality on a department or division level. Therefore we employed additional labor to extract data from the system and summarize by hand. Another driving force for change was the need to access process information. With our production processes controlled by Honeywell digital control systems, tracking over 15,000 pressures, temperatures, flows and levels, we realized much of the key to understanding our processes was locked inside our process
control systems. The only way to analyze the data was a data dump to a report and then rekey the data into a spreadsheet, something many engineers would not take the time to do. Clearly the system needed improvement.

SOLUTIONS

We tried to take the best parts of many of today's ideas to build the new functionality into our system. This included SAS (what I consider a 4th generation language) Data Warehousing with user access, and a small sprinkling of OOPS and Client server ideas.

SAS™ SOFTWARE

There were several reasons we settled on SAS over other 4GL's. SAS had it's foot in the door due to its statistical roots, but we needed to go slow as in 1990 SAS was not considered a "real" programming language by many at Eastman Chemical. We found SAS broader in scope than other 4GL languages, there were better form generators but they could not do graphics. There were better charting and statistical packages but they did not have the control offered by SCL. Rather than learn, license and program with a variety of packages we wanted one solution. Another reason we settled on SAS was I was familiar with SAS and continually contemplated that SAS could accomplish with five lines what Fortran used 2040 lines of code to accomplish:

```
DATA ONE;
  SET DATABASE;
  WHERE SAMPLEPT = 'XXX' AND START_DATE<DATETIME<END_DATE;
RUN;
```

In reality the above five line program grew to 450 lines after packaging in SCL with all error checking, but this remained a significant improvement from 2040 lines used by Fortran.

The final reason for deciding upon SAS was the company decided it should move data from flat files to relational databases and SAS presented a seamless integration with most DBMS's. This gave us a good political footing of supporting and integrating with the company's I/T strategy by using SAS. So we decided upon SAS as the tool to significantly improve our programming productivity.

DATA WAREHOUSE

The foundation of any system is a solid data structure. We began by laying out a "normalized" data structure from our existing file structure. This meant we had to duplicate our data but with current prices of disk storage it is a small price to pay. Sometimes it was quite easy to move data from flat files to a DBMS as some of the RMS files were normalized. All it takes is a simple program like:

```
* Getting the test data file;
DATA TESTDATA;
  INFILE 'TESTDESCR.DAT' TRUNCOVER;
  INPUT @3 TESTCODE $ 8.
    @11 TESTNAME $ 25.
    @66 METHODD $ 3.
    @69 METHODNO $ 4.
    @82 TESTPREP VAXRB4.
    @92 TESTFACT IBI.
    @93 TSTNAME $ 12.
    @110 TEST_ACT $ 1.
    @167 Formula $ 100.;
  * Keeping only active tests;
  IF TEST_ACT = 'Y';
  RUN;

* Deleting old data from the table;
PROC SQL;
  DELETE FROM DATABASE.TESTDATA;
```

* Add new data to database;
PROC APPEND BASE=DATABASE.TESTDATA
  DATA=TESTDATA;

...
However, sometimes it required significant programming effort as we had to read several files and perform major data conversions to change and separate the data into a normalized format.

We initially copied the core part of the laboratory system with nightly batch jobs. Then we created views to provide the needed department and divisional summaries. For the first time a department superintendent could run a single command to show the quality for their area over the past day, week or month and pinpoint trouble areas. Eventually the data became so valuable we now update most data real time. Thus, we grew into a hybrid system of SAS/Database and Fortran/RMS Files to meet our needs. An example of a summary report follows and shows how all inprocess tests ran in an area over a weekend. DATA IS FOR ILLUSTRATIVE PURPOSES ONLY.

With a solid normalized data foundation and the programming speed and power offered by SAS we had a quick visible success that allowed us to proceed. After the lab data was moved into a database we started to integrate massive amounts of process data. Initial performance testing proved using a relational database a wise choice. While there is much overhead with DBMS's we found that RDB performed much better than SAS datasets for large tables and especially views. Once a table grew to over 400,000 records, or needed multiple indexes and/or be part of data views RDB, was a strong performer, often out performing SAS datasets by a factor of more than ten. As we grew very large however, we found some tables to be denormalize for reasonable performance. Initially our process data was stored in the following format:

<table>
<thead>
<tr>
<th>Datetime</th>
<th>Tag</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>01JAN95:00:00</td>
<td>FC175</td>
<td>450</td>
</tr>
<tr>
<td>01JAN95:01:00</td>
<td>FC175</td>
<td>444</td>
</tr>
<tr>
<td>01JAN95:02:00</td>
<td>FC175</td>
<td>438</td>
</tr>
<tr>
<td>01JAN95:03:00</td>
<td>FC175</td>
<td>444</td>
</tr>
<tr>
<td>01JAN95:04:00</td>
<td>FC175</td>
<td>438</td>
</tr>
</tbody>
</table>

This makes it very easy to integrate with laboratory data. But after the table grew past 4,000,000 rows performance dropped rapidly and could not be improved. So we tried a trick with PROC TRANSPOSE to store 24 hourly averages per row by creating the following table:

<table>
<thead>
<tr>
<th>Datetime</th>
<th>Result10</th>
<th>Result1</th>
<th>Result2</th>
<th>Result23</th>
</tr>
</thead>
<tbody>
<tr>
<td>01JAN95:00</td>
<td>FC175</td>
<td>450</td>
<td>465</td>
<td>425</td>
</tr>
<tr>
<td>02JAN95:00</td>
<td>FC175</td>
<td>444</td>
<td>445</td>
<td>450</td>
</tr>
<tr>
<td>03JAN95:00</td>
<td>FC175</td>
<td>458</td>
<td>465</td>
<td>438</td>
</tr>
<tr>
<td>04JAN95:00</td>
<td>FC175</td>
<td>444</td>
<td>465</td>
<td>458</td>
</tr>
<tr>
<td>05JAN95:00</td>
<td>FC175</td>
<td>458</td>
<td>485</td>
<td>499</td>
</tr>
</tbody>
</table>

While this makes it more difficult to merge with laboratory data and takes additional processing to "unwrap" the data, the time to extract data improved by a factor of fifteen. So total time dropped significantly. So what we learned is that normalized data is a noble
goal which we can often achieve since our port to DEC's new ALPHA platform. However, one must still take into account machine performance. For some programs on our system we had to resort to monthly or quarterly data summaries. Even with today's fastest computers a global join of millions of records still takes longer than most people want to wait on an interactive command. With this performance problem solved we added additional process, customer, inventory and production data. The database now covers seven gigabytes of space over five drives.

To take greater advantage of the database we created user accounts on the production system and gave endusers access to the data. For ease of use we created several data views and sample programs to be used as templates. We use PROC CONTENTS of the tables and views to show the user how and where the data is stored. This enables the novice to quickly get the information they need. We now have many non-IS people meeting their information needs quickly and directly.

OOPS

I do not profess to be knowledgeable in Object Oriented Programming System (OOPS) technology. But a primary benefit is to have small reusable bits of code that can be linked together to rapidly build programs. Base SAS enables this through macro code even in SCL applications. Here are a few examples of how we put this into practice. Below is a batch file that shares a block of macro code with an SCL application. Since both programs share the same code, maintenance is reduced as there is only one place to make changes. At the end of the example I call a common routine to send the file to anyone who wants it. The sending code is only about ten lines but since it is used by over twenty programs, the macro reduces ongoing maintenance and insures consistency between the programs.

BATCH PROCESSING (Macro Use)

* ACETYL.SAS;
* This program is run by BATCH:
  DB_INVENTORY each morning. It
  creates the acetyl report and sends
  a copy to users that have subscribed
  to it under the SUBS message.;

* delete the old file first;
X 'DELETE SASLIS:ACETYL.LIS;';

* Naming the output file to release
  the file before I send a copy of
  the listing. SAS can not release
  the file unless defined first;
PROC PRINTTO PRINT='acetyl.lis';
run;

* Getting the dates for the program
  and creating macro variables for
  use later;
DATA NULL ;
  CALL SYMPUT('sd',dhms(today()-2,  
            0,0,0));
  CALL SYMPUT('edt',dhms(today()),  
            23,59,0));
RUN;

* Pulling in the code for the program. A "subroutine" is
  used because it is the same code
  used by the ACET message.;
INCLUDE 'SAS COMMON ACETYL.BAT';

* Defining the report (output file)
  name;
%LET REPORT = ACETYL;

* This code sends the report to
  those requesting it via the SUBS
  message;
INCLUDE 'SAS COMMON SYMTRACACETYL.SAS';

SCL PROCESSING (Macro Code)

In many places we need to validate the product number. While not difficult, it
became time consuming to maintain the same thirty lines of code in more than twenty-five separate programs. Again this insures consistency for both the user and programmer for before we started to use the macro, we found the validation performed differently in different programs. SCL does not allow one to use a subroutine since we are validating a screen variable, so we use a macro to pull in the necessary lines of code. To do this one must first set the SASAUTO’s option in SAS to point to the directory where common code is kept. Then the BUILD PROCEDURE can locate, insert the code directly into the SCL program and finally compile the total program.

Proc build code:

LIBNAME DEFAULT '['];
OPTIONS LS=80 PS=59 FULLSTIMER
SASAUTO = 'SAS COMMON';
PROC BUILD CATALOG = DEFAULT.QSUM;
RUN;

SCL code:

IF MODIFIED(PM_NO) THEN DO;
* Macro code is stored in COMMON:
  IPM_NO;
END;

Actual code inserted upon compile:

```
/* This code verifies PM NOs in SCL 
programs. */
MACRO PM NO;
  MSG = "Verifying the PM number:";
  REFRESH;
  * LOCATE and DATALISTC work much faster on sasdata sets than on
  view descriptors. So I copy;
  IF EXIST('WORK.PM MAST') = 0 THEN
    RC = COPY('DATABASE PM MAST(KEEP=
      PM NO S NAME NAM RESP NAM REAC)"
    , 'WORK.PM MAST', 'DATA');
    IF PM NO = '? THEN PM CK = OPEN
      ('WORK.PM MAST', 'I');
    IF PM NO = '? OR SUBSTR(PM NO,1,1) = '? THEN DO;
    PM NO = DATALISTC(PM CK,'PM NO
      S NAME');
    MSG = ' ?';
    S NAME = GETVARC(PM CK,VARNUM,
      PM CK,'S_NAME');
    END;
  ELSE DO;
  PM NO = WHERE([PM CK,'PM NO=
      "\"[PM NO]"\"');
  RC = FETCH(PM CK,'NOSET');
  MSG = ' ?';
  IF RC = 0 THEN DO;
    MSG = 'PM # not found use ? for list?'
      ALARM;
    CURSOR PM NO;
  END; /*rc1=0 record not found */
  ELSE S NAME = GETVARC(PM CK,VARNUM,
      PM CK,'S_NAME');
  END;
  RC = CLOSE(PM CK);
  PM CK = .;
%MEND PM NO;
```

Another method to standardize SCL code is via the call display as in the following example. Since we are not validating a screen variable we can use common SCL code.

```
CALL DISPLAY("DEFAULT.COMMON.PRINT.
  PROGRAM",FILE,MSG);
```

This calls an entry from a catalog that contains common groups of code. The above routine checks to make sure a file exists and if not gives a warning note. If the file does exist it shows the contents to the
user and allows them to print or save it in various ways, then the file is deleted. All
text based output passes through this one
routine. Again this is not OOPS in terms of
classes and objects but it gives us significant
standardization, reusability and improved
productivity which are primary objectives of
OOPS.

CLIENT SERVER (basic concepts)

The main goal in client server is to leverage
the storage and processing capability of
midrange computers with the convenience
and power of desktops. The trick in our
environment is to figure out how to
implement the benefits without the cost of a
networking infrastructure. With our users
spread out over several miles it is not trivial
to run the necessary networking fiber.
However, with standard options available on
windows based terminal emulators and
spreadsheets there is an inexpensive option.

On the midrange we have an application that
allows the users to extract and match lab
and/or process data. We format a simple
tabular report and allow the user to scroll the
data past their screen. Then with a simple
click to copy a table of data from the
terminal emulator, then paste into a
spreadsheet, the data is moved from the
midrange to the desktop for further
manipulation, analysis or plotting. Again
while not client server in the true sense it
does enable up to reach most of the goals of
client-server at only a fraction of the costs.

PROBLEMS

Unfortunately this system did have a few
growing pains. One that was difficult to
anticipate was database tuning. We found
databases do not follow linear analysis. We
found several "cliffs" in performance and had
to tune over the years. While this was not
difficult to correct once detected, it was
difficult to anticipate, as an unchanged

![Time to Extract Data from Different Sized Tables](Image)

program that worked fine yesterday suddenly
took ten to fifteen times the time to process.
This was frustrating to the users and very
frustrating for the programmers.

A second problem is the user interface with
SAS and the existing system is slightly
different. For example in the VAX world
CNTRL/Z usually means finished but in SAS
that is one of the TWO keys that one cannot
remap. Fortunately this appears to be a
small problem from the users' perspective,
but it does bother the programmers.
The final problem we encounter is when we try to leverage both SAS and fortran to solve a problem. Because the two languages have very different approaches we often spend as much time on the handshaking as we save in programming time using SAS. This I believe, is a solvable problem with more training and experience.

RESULTS

We have been very pleased with the results of our hybrid system. The extra storage and problems required for duplicating most laboratory data has been more than offset by:

Project completions increasing from 7 to 75/yr. while staff only increased from 2 to 3.
Data volume has grown from 2,000,000 to 37,000,000 values/yr
Computer performance sometimes improved with SAS and RDB.

We had a fortran program that calculated variation in all of our test properties. The program ran each month to inform us on a global scale, if we are reducing variation and thus improving quality. We found, after several years, some numbers were in error. Our new analyst was faced with the daunting task of finding the error(s) in the 2,631 lines of direct code. To make matters worse, numerous subroutines are called for a total of 25,025 lines of code. Due to the anticipated time to debug we decided to redo the entire system in SAS.

The comparison is enlightening:

<table>
<thead>
<tr>
<th></th>
<th>SAS</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time to write program</td>
<td>2 Wks</td>
<td>20wks</td>
</tr>
<tr>
<td>Charged project time</td>
<td>3 dys</td>
<td>17dys</td>
</tr>
<tr>
<td>Lines of code (base program)</td>
<td>196</td>
<td>2,631</td>
</tr>
<tr>
<td>Time to run program</td>
<td>58 min</td>
<td>330 min</td>
</tr>
</tbody>
</table>

This help debug the myth that SAS always consumes much more computer resources than a 3rd generation language like Fortran.

CONCLUSION

Overall we greatly increased the life and functionality of our legacy manufacturing information system, while improving productivity of both the programmers and users. This was accomplished with no disruptions to the production system, or significant retraining of users.

The costs to implement our system was very low. Payback was well under a year or less than 20% of any other system that would give us the functionality we sought. We accomplished this feat by careful integration of SAS Software and Data warehousing, and techniques borrowed from OOPS and Client-Server technology.

ACKNOWLEDGEMENTS

SAS is a registered trademark of the SAS Institute, Inc. in the USA and other countries.