

A CORPORATE REVENUE PLANNING APPLICATION USING SAS/AF*

Joseph S. Bartling
Hallmark Cards, Inc.

William F. Vander Lippe
Hallmark Cards, Inc.

SAS/AF is the vehicle for an online, interactive revenue planning application at Hallmark Cards, Inc. It was developed in a rapid prototyping environment, going from initial concept to functional system in roughly 4 months. Programmer involvement began just 2 months prior to turning over a functional application to the users. SUBMIT block processing forms much of the foundation of the application and enabled us to use SAS Release 6.06 rather than the less familiar Screen Control Language(SCL) for the bulk of the data manipulation tasks. It also allowed us to centralize critical code and use multi-dimensional arrays. The result is an online, interactive system which meets users' needs and remains flexible, manageable, and maintainable.

THE PROJECT

THE PREVIOUS SYSTEM. The impetus for developing a new revenue planning system lay in the old system which was inadequate to the demands of the world's leading creator and manufacturer of social expression products. Revenue plans were developed at the product level using logic applicable to that product and, as often as not, inapplicable to any other product. Planning language was inconsistent and planning assumptions were undocumented. "Transactions", for example, might mean any of nearly half a dozen things and the numbers plugged into it might be based on myriad assumptions about the economy in general, the industry as a whole, the corporation's anticipated performance, or product-specific expectations. Such assumptions went unwritten, though.

In the old system, strategic evaluation was extremely difficult. Once combined into aggregates, plans could not be broken down into coherent components. Hence, it was nearly impossible to focus on specific plan items which warranted attention. In addition, the paper-driven nature of the system virtually assured the current plan information would be delivered to different people at different times, oftentimes in chunks and pieces spread across several documents. Lastly, "what if" scenarios were painfully difficult to execute and evaluate.

USER WISH LIST. The new system, as users envisioned it, would be an on-line, interactive spreadsheet which would apply the same planning logic to all products. It would put all plan data in one location but could be accessed by only those individuals with a real need to use the data. It would be a functional prototype capable of being modified, drastically changed, or maybe scrapped entirely. It would have simulation and report capabilities. In addition, users would be able to combine 2 or more products to create higher level aggregates known as "roll-ups." A few roll-ups would be pre-defined and users would be able to build additional roll-ups on the fly. No one knew how many users, products, or roll-ups there might eventually be, but it was certain that user/function/product/roll-up relationships would be dynamic. Most, if not all, users would be thoroughly unfamiliar with digital processing in general and none would have the slightest acquaintance with SAS. The system would be subject to heavy usage of short duration. TSO was the sole common platform available to all end-users of the system although there was speculation that the proposed system might eventually move to workstations.

DEVELOPMENT STRATEGIES. Several strategies for addressing user needs were developed early in the project while others took shape as we gained more experience with SAS/AF. A pseudo-database structure was selected with 10 tables driving and defining the user/function/product/roll-up relationships. In the absence of SAS/SHARE, files would be allocated for no longer than absolutely necessary, and users would do the bulk of their processing against working copies of the permanent data. Editing and updating of data would be allowed at only the product level with one user authorized to edit each product. A browse function would be set up to allow users to view plan data for products which they were not authorized to edit but with which they needed some familiarity. All roll-ups would be pre-defined and could be browsed but not edited. Users would be able to enter some values at the roll-up level to provide a "what if" capability. Pre-defined PF keys would enable the user to navigate the system with little more than single keystrokes. As the application evolved, SCL became more and more a "screen tool" while SAS, via SUBMIT blocks, became the "data tool." In addition, dynamic extended tables replaced the hard-coded selection lists initially used to drive user input to the system.

THE APPLICATION

OVERVIEW. The user enters the application by typing in a user ID and password on a log-on screen. Based on the user ID, tables are built which define the functions (Edit, Browse, and/or Print) the user may perform and the products/roll-ups he may work with. Next, the user chooses the desired function from an extended table selection list. This is followed by a second extended table selection list from which the user chooses the products/roll-ups he wishes to print, browse, or edit. Both extended tables are dynamic

and are tailored to the individual user, allowing them to choose only those functions, products, and roll-ups for which they are authorized.

When edit or browse functions are selected, the permanent data file(s) is(are) read and a working copy of the data is created. This working copy is always a single observation. In edit mode, the user may make changes to the data and save them. Since he is manipulating only a temporary working copy of the permanent data, he is free to type in whatever figures he wishes and observe their impact elsewhere in the plan. This provides the simulation capability. Changes are kept when the user enters a "Save" command which merges the working copy into the permanent SASLIB.

When the print function is selected, a batch job is created and submitted. This job uses the application displays as templates to write hardcopy reports which essentially duplicate the on-line system.

WHY USE SUBMIT BLOCKS? SUBMIT blocks facilitated the development of centralized code. Instead of repeating similar or identical code in multiple SAS/AF program entries, we were able to write SAS routines which could be executed from any entry. As a result, fixes and updates to calculations and processing routines are made by changing one set of code. The system is consistent throughout and easy to maintain. In addition, the routines can be executed from batch jobs, thereby adding flexibility and responsiveness in handling ad hoc requests.

SUBMIT blocks also enabled us to use multi-dimensional arrays which made table-handling easier in some situations.

As much as possible, DATA steps were compiled, stored, and executed using the Stored Program Facility. This eliminated the need to compile the same DATA step(s) repeatedly during execution and made the application run more quickly and efficiently.

By using SUBMIT blocks for data manipulation and calculations, we were able to work with familiar tools in SAS rather than having to learn to do the same things in SCL. This contributed to a reduced learning curve for developing the application and enabled us to deliver the system sooner than we might otherwise have done.

WHAT IS A SUBMIT BLOCK? A SUBMIT block begins with a SUBMIT statement and ends with an ENDSUBMIT statement:

```
SUBMIT < when < where > < host >> < 'STATUS' >;  
.  
.  
    SAS code (or SQL statements)  
.  
.  
ENDSUBMIT;
```

When SCL encounters a SUBMIT statement, it collects all the statements between the SUBMIT and the ENDSUBMIT statements and sends them to the SAS system for execution. Control returns to SCL.

The 'when' argument in the SUBMIT statement instructs SCL when to submit the generated statements for execution and what action the procedure must take. Our application uses 'CONTINUE' as the 'when' argument so at the end of the SUBMIT block, the procedure submits all statements stored in the PREVIEW window and returns control to the SCL program. In addition, the ENDSUBMIT statement is preceded by a RUN statement to force execution of all statements in the block. Execution of the program continues with the statement following the ENDSUBMIT statement. In effect, SAS processing replaces SCL.

PASSING VALUES FROM SCL TO SAS. Values can be passed from SCL to SAS (and vice-versa) quite readily and in several ways. Our application predominantly relies on the method in which SCL scans the text in a SUBMIT block for possible substitution. When it encounters a name prefixed by an ampersand(&), it checks to see if the name following it is an SCL variable (window or nonwindow). If so, the value of the variable is substituted. No substitution is performed if the name following an ampersand is not an SCL variable. The following illustrates this technique:

```

102      /*****
103      /* 114-128: SUBMIT BLOCK                               */
104      /* 115-117: DATA STEP SETS DATASET PRODDATA &        */
105      /*          VALUES CALCULATION SWITCH                 */
106      /* 118-123: ASSIGN VALUES OF SCL VARIABLES ON        */
107      /*          RIGHT SIDE OF EQUAL SIGN TO SAS            */
108      /*          VARIABLES ON LEFT SIDE                     */
109      /* 125-125: MACRO ROLLMAC3 EXECUTES A SERIES OF        */
110      /*          PROCS AND DATA STEPS TO CREATE THE        */
111      /*          DATASET WHICH IS BROWSED IN THE            */
112      /*          NEXT SAS/AF SCREEN.                         */
113      /*****
114      SUBMIT CONTINUE;
115          DATA PRODDATA;
116              SET PRODDATA;
117              CALCSW = 'ALL';
118              RSIP1=INPUT('&RSIP1',8.2) / 100;
119              RSIP2=INPUT('&RSIP2',8.2) / 100;
120              RSIP3=INPUT('&RSIP3',8.2) / 100;
121              RSIP4=INPUT('&RSIP4',8.2) / 100;
122              RSIP5=INPUT('&RSIP5',8.2) / 100;
123              RSIP6=INPUT('&RSIP6',8.2) / 100;
124
125          %ROLLMAC3
126
127          RUN;
128      ENDSUBMIT;

```

Macro variables are passed without modification to the SAS system where they are resolved. Therefore, it is important NOT to use the same names for macro variables and SCL variables.

The REPLACE statement and the REPLACE attribute in the ATTR window can also be used to pass values from SCL into SAS. However, neither was used in our application. REPLACE statements, which substitute text in a SUBMIT block for SCL variables which are not missing values, can be useful in conditional processing situations. This was not the case in our application. In addition, the REPLACE attribute in the ATTR window is not readily apparent to anyone looking at the source program.

PASSING VALUES FROM SAS TO SCL. The SYMPUT routine can be used in a SUBMIT block to assign a value to a macro variable (or create the variable if it doesn't exist). The SYMGET function can then be used in the SCL code to return the value of the same macro variable.

In another method, the SET routine in SCL is used to associate SAS data set variables with SCL variables of the same name and type. Then when the SAS data set is read, the SCL variables are valued from the corresponding SAS data set variables. Values are padded or truncated if variable lengths don't match. The UPDATE or APPEND functions can then be used to write the values of the SCL variables to the SAS data set. This technique significantly reduces the amount of code necessary to transfer values:

```

024      /*****
025      /* PASS VALUES FROM SAS DATASET TEMPDATA TO SCL FOR */
026      /* VARIABLES WITH SAME NAME AND TYPE (NUM/CHAR).    */
027      /* 035-035: OPEN SAS DATASET TEMPDATA FOR INPUT      */
028      /* 036-042: IF OPEN SUCCESSFUL, EXECUTE DO GROUP     */
029      /* 039-039: LINK SAS VARIABLES WITH SCL VARIABLES    */
030      /*              WITH SAME NAME AND TYPE               */
031      /* 040-040: FETCH OBS 1 FROM TEMPDATA AND VALUE SCL  */
032      /*              VARIABLES                             */
033      /* 041-041: CLOSE SAS DATASET TEMPDATA                */
034      /*****
035      DSID1 = OPEN('TEMPDATA','I');
036      IF DSID1 = 0 THEN MSG = SYSMSG();
037      ELSE
038          DO;
039          CALL SET(DSID1);
040          RC = FETCHOBS(DSID1,1);
041          RC = CLOSE(DSID1);
042      END;
```

CONCLUSION

Our goal, above all else, was to develop an on-line, interactive, prototype application in as short a time as possible. SAS/AF and

SUBMIT block processing were crucial to the successful attainment of that goal. SUBMIT block processing allowed us to shift tasks from SCL to SAS where we could use the code and techniques we knew best. This, of course, reduced the learning curve and shortened the initial development phase. It also enabled us to centralize the guts of the application, the DATA steps and PROCs which perform the primary calculations, and make them available for batch jobs and ad hoc requests. The application as a whole is more flexible, manageable, and maintainable as a result. It is a dynamic application in a dynamic environment and will remain so.

REFERENCES

SAS Institute Inc., SAS Screen Control Language: Reference, Version 6, First Edition, Cary, NC: SAS Institute, Inc., 1990.

SAS Institute Inc., SAS/AF Software Applications Using Screen Control Language (Release 6.06) Course Notes, Cary, NC: SAS Institute, Inc., 1991.

SAS Institute Inc., Advanced SCL Applications Course Notes, Cary, NC: SAS Institute, Inc., 1991.

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

TRADEMARK NOTICE

*SAS, SAS/AF, and SAS/SHARE are registered trademarks of SAS Institute, Cary, NC.

AUTHORS

Joseph S. Bartling, Associate Senior Management Science Analyst
Hallmark Cards, Inc. MD #203
2501 McGee Box 419580
Kansas City MO 64141-6580
(816)274-3007

William F. Vander Lippe, Research Analyst/Programmer
Hallmark Cards, Inc. MD #203
2501 McGee Box 419580
Kansas City MO 64141-6580
(816)545-6764