

A New Reporting System in SAS®

Steven M. Epp, The Boeing Company

I. Introduction

This paper describes the process by which a large reporting system was created for the Reliability and Maintainability (R&M) engineering organization of the Boeing Commercial Airplane Group. Among other things, the R&M group is charged with managing the in service data on all of the Boeing planes currently flying. Almost all of this data comes from the various operators of the airplanes, and while the coverage is not perfect, there is a large amount of data. This data is stored in relational tables in NCR 5100M computer, a successor to the Teradata 1012 database computer (DBC). At the time the project began, the Teradata 1012 DBC was being used.

Now it is a cliché that acquiring and keeping data of and by itself is not very useful. There have always been reports from the data, mostly trend analysis to identify potential problems for further analysis. Over time, these reports had become hard to manage, written in several different languages and requiring specialized data processing skills to use. While in most cases, the reports themselves were acceptable, the process of producing them had become difficult, slow, inflexible and expensive. Clearly a change had to be made. Several meetings between the R&M group and the information services provider took place, and a set of fairly general requirements was given to the development team.

II Requirements

The requirements for the reporting system were very complex and demanding. Apart from the technical requirements of the individual reports, the system had to:

1. Support a geographically dispersed user community.
2. Allow "mod-hoc" capability. This was interpreted to give the user significant control over the content of the report by the data selection process and some control over the presentation while not requisitioning Information Services (IS) resources to run the reports.
3. Process a large amount of data
4. Produce high quality, complex graphical reports on a variety of output devices.
5. Many of the reports were already in use. The customer wanted to keep the physical appearance of the reports the same in the new system.
6. It was also important that the reports had a consistent look and feel. This sometimes resulted in some conflict with the previous requirement.
7. Provide the capability for batched overnight reporting and same day response.

An additional requirement was placed on the IS function to reduce the number of tools and platforms used by the then current reporting system.

Whatever the details every formal discipline of application development stresses the importance of requirements management. It is also one of if not the most difficult methods to implement and still have most, if not all of the participants in the project on speaking terms at the end of the exercise.

Ideally, the development team should be working to the same set of requirements the customer has expressed. In turn, these requirements should meet the business case the application is designed to satisfy. The result of the requirements management should be a process that ensures, among other things, that the customer requirements have been clearly communicated. The customer has to be satisfied the development team understands the requirements as given by the customer. The development team should make sure the requirements are feasible and that they remain feasible given any time or resource constraints that may exist.

While the general requirements were fairly clear, they did lack a certain instrumentality. What did exist were some reports and a general direction to make the reports written in SAS® to be the same. This was supplemented with informal discussions between some of the developers and some of the R&M engineering staff as to how the results on the reports had been derived from the information in the database.

At the same time, some of the developers felt a completed report was all of the specification they needed and they started to code. Not surprisingly, this did not work well, and for a while there were unhappy customers and a frustrated development team. The situation improved when the customer appointed a sub team to work with the development team to write down the requirements. Every one was much happier and the production rate increased.

III. Choosing A Tool

The choice of a delivery platform was relatively easy. When this system was being designed, there were still some members of the user community who did not have access to a PC, but did have access to a terminal into the Boeing owned mainframe computer system. An MVS-TSO system was used.

The software choice was to use SAS® for the reports. SAS® was chosen because it has a flexible and powerful programming language that could read and write information in almost any format. In addition the SAS/GRAPH® product has powerful graphics tools that will produce high quality custom graphics, and the list of supported devices is quite extensive.

NOMAD[®] was chosen as the tool for the user interface. NOMAD[®] was chosen because there was already an application written using NOMAD[®] that was for another reporting suite. Let's call this the AIS reports. While the new graphic reports in particular were not well suited to NOMAD[®], the AIS reports and their interface on the mainframe already had user acceptance. We could leverage off this acceptance and modify the user interface from the AIS report system. This interface would write out the user's information that would select and customize the desired report. The interface was completely text based, and therefore could be used even by users whose access was on native 3278 terminals, as required. NOMAD[®] also has an easy interface to the DBC, which meant that tables on the DBC could be used to directly validate the user's selections.

The NOMAD[®] interface passed the validated selection and custom choices from each user to a REXX job. The REXX program used this information to launch a batch job that actually ran the report, using Job Control Language custom made for the particular requirements of the job. For example, the user could request the desired level of service by setting the job's priority on the mainframe system.

The last tool was the one that allowed the data to pass from the Teradata database computer to the SAS[®] application running on the mainframe computer. The problem was, and is, that there is no direct interface between SAS[®] applications and the Teradata DBC. An attempt was made to use the SAS/ACCESS[®] interface for DB2[®] in combination with an interface on the DBC that would make the DBC look like a DB2[®] system to the SAS/ACCESS[®] interface program. This was tried on a VM mainframe system, and while it worked after a fashion, the results were not satisfactory from performance or stability criteria.

The only remaining option was to use SQL scripts to transfer data from the DBC to the mainframe. The output from this process was an ordinary dataset on the mainframe, not a SAS[®] data set. In order for the data to be processed by the SAS applications, the data had to be read from the DBC by SQL, written to the mainframe, then read back into a SAS[®] data set before any further processing could be done.

IV. Development

Along the way, there were some problems. Let's look at the way SAS[®] helped to solve or at least mitigate each situation as it came up.

Even given the large number of supported graphics devices supported by SAS[®], our data center used a XEROX[®] printer and software that was not on the list. SAS Institute people worked with our technical support people at our data center to put together another device driver that worked with the software used by the graphics device. This functionality allowed users who were at locations without a graphics device to get graphical output from the mainframe sent directly to them as part of the normal data center process. We found the ability to create our own device drivers based either on existing drivers in the libraries or on the metagraphics device driver to be most useful.

Each report has a long list of parameters passed to the program by the user interface. These parameters control things like sort order, data selection criteria, and some aspects of report appearance. The macro facility was most helpful in solving these problems. It allowed the SAS[®] routines to generate problem specific code for a particular circumstance of a report.

For example, a report might be generated for one model of airplane, say a 767, or many models. The information for these reports exists for a long period of time. A date range can be selected. Minimum and maximum ranges on the charts and number of axis divisions can be selected. Text for report headers and footers can be supplied by the user or allowed to default.

The other advantage of using the macro facility is of course, the ability to share and reuse SAS[®] code in a macro library. One common task was a work around resulting from the lamentable fact that the longest character variable in SAS[®] is 200 characters long, while character columns in the database can be up to 32K characters long. Many of the reports need to be able to take a very long character string kept in an array of SAS[®] variables, usually of length 200 and convert them into a set of variables of shorter length, usually of length less than 132. This was done to write out the text item, but not to break a word between the shorter variables. Functionally, this is the same as the wordwrap option found in many PC text viewers such as Notepad. I have put a copy of our version of "WordWrap" at the end of the paper.

As for the graphical reports, heavy use was made of the annotate facility. In simple terms, the Annotate facility of SAS/GRAPH[®] uses a SAS[®] data set to place graphic objects on the page with complete control of placement and orientation. In some cases, the entire chart is drawn using the annotate facility! I have included an example of one of the graphical reports.

V. Ahead of our Time?

As the development of the application progressed, it became apparent that most of the reports were requesting the same information from the DBC. The architecture of the system allowed the same report to be run multiple times, either by the same or different users. With increased usage, this process would take a lot of time, be expensive, and adversely impact other users of the DBC. On the other hand, the information in the DBC was constantly being updated in some areas, while updated periodically in others. There was an advantage of getting a fresh copy each time the report was run. Other reports also requested so much information from the DBC that the cost was prohibitive.

After some thought and planning, we decided to make a copy of the most commonly used information on the mainframe in SAS[®] data libraries. When we did this, we arranged the data in the libraries not as copies of the relational tables, but optimized to support our reporting. The refresh cycle was synchronized with the regular data inputs. The reports may not show all of the latest data, but the cost and time savings more than justified this minor loss.

Did we have a prototype of a data warehouse in 1992? In the words of the immortal Bugs Bunny, "Mmmm...Could be!"

VI. Conclusions

This reporting system has been in use for about four years now, and is still heavily used for the basic reporting needs of the R&M Engineering group. It has in general performed well.

Acknowledgments

My thanks to The Boeing Company for allowing the presentation of this material and for production support. I would also like to thank Karen Thurber for her unstinting support, both then and now.

SAS® and SAS/GRAPH® are registered trademarks of SAS Institute, Inc. in the USA and other countries ® indicates USA registration. Other brand and product names are registered trademarks of their respective countries.

The WordWrap Macro

```
%MACRO WORDRAP (INBASE=, OUTBASE=, INLEN=, OUTLEN=, INVAR=, OUTVAR=,
                CLEAROUT=NO, INARRAY=, OUTARRAY=);
/*****
*   WRITTEN: 1/92
*
*   MACRO %WORDRAP COPIES TEXT FROM ONE SET OF CHARACTER VARIABLES
*   TO ANOTHER SET OF CHARACTER VARIABLES USING DATA STEP PROCESSING.
*   WORD SPLITTING IS AVOIDED SO THAT THE TEXT IN THE OUTPUT
*   VARIABLES MAINTAINS READABILITY.
*
*   A LENGTH STATEMENT IS GENERATED FOR THE OUTBASE VARIABLES,
*   SO IT IS NOT NECESSARY TO PRE-DEFINE THE OUTPUT TEXT VARIABLES.
*
*   LIMITATIONS:
*       1) THE INPUT VARIABLE NAMES HAVE TO BE NAMED XXX1, XXX2,
*          XXX3, ..., XXX10, ETC., WHERE INBASE=XXX. NAMES
*          CONTAINING LEADING ZEROS ON SUFFIX, LIKE XXX01, XXX02,
*          WILL NOT WORK.
*       2) INVAR AND OUTVAR MUST BE 2 OR GREATER.
*       3) INLEN AND OUTLEN MUST BE 3 OR GREATER.
*
*   USE EXAMPLE:
*       DATA _NULL_;
*           SET X;          ( X CONTAINS INBASE TEXT VARIABLES )
*           %WORDRAP(INBASE=LONG, OUTBASE=SHORT,
*                   INLEN=200, OUTLEN=100,
*                   INVAR=25, OUTVAR=50);
*           LINESOUT = WORDRAP;
*           DO J = 1 TO LINESOUT;
*               PUT @43 LONG{J};
*           END;
*
*   REQUIREMENTS:
*       1) INPUT MUST BE IN A SET OF SAS VARIABLES, ALL WITH THE
*          SAME LENGTH, AND ALL IN THE VARIABLE NAMING SEQUENCE,
*          EG DESC1-DESC25.
*       2) OUTPUT MUST ALSO BE A SET OF SAS VARIABLES, ALL WITH THE
*          SAME LENGTH, AND ALL IN THE VARIABLE NAMING SEQUENCE,
*          EG OUT1-OUT50.
*       3) A MINIMUM OF TWO VARIABLES ARE REQUIRED FOR BOTH INPUT
*          AND OUTPUT.
*       4) IF YOUR DATA STEP CODE CONTAINS 'RETAIN;' THEN SET
*          CLEAROUT=YES TO INSURE OUTPUT VARIABLES ARE ALWAYS
*          INITIALIZED TO BLANKS EACH TIME THRU DATA LOOP.
*****/
```

```

* INPUT:
*   INBASE= THE 'BASE' PORTION OF THE NAMES OF THE SET OF
*           VARIABLES IN WHICH TEXT CAN BE FOUND THAT IS TO BE
*           TRANSFERRED. FOR EXAMPLE, IF DESC1-DESC25
*           IS THE TEXT SOURCE, THEN INBASE=DESC.
*   OUTBASE= THE 'BASE' PORTION OF THE NAMES OF THE SET OF
*           VARIABLES IN WHICH WORD-WRAPPED VERSIONS OF THE
*           TEXT IS TO BE PLACED. FOR EXAMPLE, IF
*           OUT1-OUT50 IS THE TEXT TARGET, OUTBASE=OUT.
*   INLEN= LENGTH OF SOURCE TEXT VARIABLES, $1 TO $200.
*   OUTLEN= LENGTH OF TARGET TEXT VARIABLES, $1 TO $200.
*   INVAR= NUMBER OF VARIABLES IN SET OF SOURCE VARIABLES.
*   OUTVAR= NUMBER OF VARIABLES IN SET OF TARGET VARIABLES.
*   INARRAY= NAME TO USE AS ARRAY NAME FOR ARRAY OF INPUT
*            VARIABLES. IF NOT SPECIFIED, &INBASE IS USED.
*   OUTARRAY= NAME TO USE AS ARRAY NAME FOR ARRAY OF OUTPUT
*            VARIABLES. IF NOT SPECIFIED, &OUTBASE IS USED.
*
* OUTPUT:
*   1) TEXT IN INBASE VARIABLES IS WORD-WRAPPED INTO
*      OUTBASE VARIABLES.
*   2) AS PART OF THE PROCESS, ARRAY NAMES ARE ASSIGNED TO
*      THE SET OF INPUT VARIABLES AND THE SET OF OUTPUT
*      VARIABLES (SEE INARRAY= AND OUTARRAY= ABOVE). THE
*      ARRAY NAMES CAN BE USED IN PROCESSING LATER IN THE
*      SAME DATA STEP. THE ARRAYS ARE EXPLICIT ARRAYS.
*   2) VARIABLE WORDSLT TELLS WHAT STOPPED PROCESS:
*      'INVAR EMPTY' INDICATES ALL TEXT SUCCESSFULLY TRANSFERRED
*                   THE OUTVAR VARIABLES
*      'OUTVAR FULL' INDICATES THERE WAS NOT ENOUGH ROOM IN THE
*                   OUTVAR VARIABLES TO HOLD ALL OF THE TEXT
*                   FOUND IN INVAR VARIABLES.
*   3) VARIABLE WORDRAOP TELLS HOW MANY OUTVARS HAD DATA
*      TRANSFERRED TO THEM. IF WORDRAOP = 0 THEN THE FIRST
*      INPUT TEXT VARIABLE WAS BLANK AND NO TEXT WAS TRANSFERRED.
*
* NOTE: VARIABLES WORDSLT AND WORDRAOP ARE DROPPED FROM
*       THE OUTPUT DATA SET. TO SAVE THIS INFO FOR LATER
*       PROCESSING, STORE IN ANOTHER VARIABLE, E.G.,
*       LINESOUT = WORDRAOP;
*
* CODE NOTES:
*   TO SIMPLIFY CODE, THE MACRO %PSEUDO WAS WRITTEN TO SIMULATE
*   A SUBSTRING FUNCTION THAT OPERATES ON A TEXT STRING OF ANY
*   LENGTH THAT CONSISTS OF THE CONCATINATED VALUES OF THE SET
*   OF INPUT TEXT STRINGS. THE TEXT VALUE RETURNED FROM %PSEUDO
*   CAN BE A MAXIMUM OF 200 BYTES.
*
*   FOR EFFICIENCY, WORD WRAP STOPS AT THE FIRST BLANK INBASE TEXT
*   VARIABLE. THIS IS DONE WITH A PRE-SCAN OF THE INBASE TEXT.
*   POINTER WORDMAXC IS SET TO THE LAST CHARACTER (CONCATINATED)
*   THAT IS TO BE WRAPPED. A FURTHER LIMITING OF THE AMOUNT OF
*   TEXT PROCESSED IS GAINED THRU USE OF LENGTH FUNCTION TO LIMIT
*   AMOUNT OF LAST TEXT LINE PROCESSED.
*****/

/* CHECK FOR MINIMUM INPUT */
%IF &INBASE= OR &OUTBASE= OR &INLEN= OR &OUTLEN= OR
&INVAR= OR &OUTVAR= %THEN %DO;
%PUT **** FOR MACRO WORDRAP YOU MUST SPECIFY ALL OF THE;
%PUT FOLLOWING: INBASE=, OUTBASE=, INLEN=, OUTLEN=;
%PUT INVAR=, OUTVAR=.;
%GOTO EXIT;
%END;

/* TO ALLOW MULTIPLE CALLS TO WORDRAP, LABELS MUST DIFFER */
%LET WORDSKIP = WDS&SYSINDEX;
%LET WORDEXIT = WDX&SYSINDEX;
%LET WRDAI = WAI&SYSINDEX; /* ALSO, ARRAY NAMES MUST DIFFER */
%LET WRDAO = WBS&SYSINDEX;
%IF &INARRAY= %THEN %LET INARRAY = &INBASE;
%IF &OUTARRAY= %THEN %LET OUTARRAY = &OUTBASE;

/*****
*   *** WHAT VARIABLES ARE FOR ***
*   PSEUDOFC FIRST CHARACTER OF A STRING
*   PSEUDOLC LAST CHARACTER OF A STRING
*   PSEUDO1 POINTER TO FIRST INPUT TEXT STRING
*           INVOLVED IN A PSEUDO-SUBSTRING
*   PSEUDOR1 POINTER TO FIRST INVOLVED CHARACTER OF
*           FIRST INVOLVED STRING IN PSEUDO-SUBSTRING
*   PSEUDO2 POINTER TO LAST INPUT TEXT STRING
*           INVOLVED IN A PSEUDO-SUBSTRING
*   PSEUDOR2 POINTER TO LAST INVOLVED CHARACTER OF
*           LAST INVOLVED STRING IN PSEUDO-SUBSTRING
*****/

```

```

*   PSEUDOE   POINTER TO LAST CHARACTER IN SUM TOTAL   *
*   STRING OF TEXT FOR PSEUDO-SUBSTRING               *
*   PSEUDOJ   POINTER FOR STEPPING BACKWARDS TO AVOID  *
*   SPLITTING A WORD.                                *
*   PSEUPTR   POINTER DURING CONSTRUCTION OF OUTPUT STR *
*   WORDMAXC  SUM TOTAL OF INPUT TEXT CHARS TO CHECK  *
*   WORDRAIP  POINTER TO INPUT TEXT VARIABLE LIST     *
*   WORDRAOP  POINTER TO OUTPUT TEXT VARIABLE LIST    *
*   WORDRPPS  POINTER TO SUM TOTAL STRING FOR START   *
*   OF STRING TO EXTRACT                             *
*   WORDRPPE  POINTER TO SUM TOTAL STRING FOR END     *
*   OF STRING TO EXTRACT                             *
*   WORDRSLT  TEXT STRING DESCRIBING MACRO RESULT     *
*****/

/*****
*
*   IMBEDDED MACRO:
*   %PSEUDO - PSEUDO-SUBSTRING FUNCTION.
*   ACTS LIKE SUBSTRING FUNCTION ON A LONG TEXT
*   STRING OF CONCATINATED VALUES IN &INBASE.
*   ONE VERSION IS FOR STRINGS OF LENGTH '1', AND
*   THE OTHER VERSION FOR ALL OTHER LENGTH SPECS.
*****/

%MACRO PSEUDO(PSEUDOS,PSEUDOL,OUTVAR);
  &OUTVAR = '';
  /* GENERATE POINTER AND OFFSET TO FIRST VARIABLE IN INBASE */
  PSEUDO1 = INT((&PSEUDOS-1)/&INLEN)+1;
  PSEUDOR1 = MOD(&PSEUDOS,&INLEN);
  IF PSEUDOR1 = 0 THEN PSEUDOR1 = &INLEN;
  %IF %QUOTE(&PSEUDOL)=1 %THEN %DO;
    WORDRAIP = PSEUDO1; /* VERSION FOR ONE BYTE LENGTHS */
    &OUTVAR = SUBSTR(&INARRAY{WORDRAIP},PSEUDOR1,1);
  %END;
  %ELSE %DO;
    /* VERSION FOR 1-200 BYTE LENGTHS */
    PSEUDOE = &PSEUDOS + &PSEUDOL - 1; /* END POINT */
    PSEUDO2 = INT((PSEUDOE-1)/&INLEN)+1; /* LAST VAR IN WRDAI */
    PSEUDOR2 = MOD(PSEUDOE,&INLEN);
    IF PSEUDOR2 = 0 THEN PSEUDOR2 = &INLEN;

    /* PSEUPTR TRACKS WHERE TO ADD NEXT PIECE OF TEXT */
    DO WORDRAIP = PSEUDO1 TO PSEUDO2;
      IF WORDRAIP=PSEUDO1 AND WORDRAIP=PSEUDO2 THEN DO;
        &OUTVAR = SUBSTR(&INARRAY{WORDRAIP},PSEUDOR1,&PSEUDOL);
      END;
      ELSE IF WORDRAIP=PSEUDO1 AND WORDRAIP<PSEUDO2 THEN DO;
        &OUTVAR = SUBSTR(&INARRAY{WORDRAIP},PSEUDOR1);
        PSEUPTR = &INLEN - PSEUDOR1 + 2;
      END;
      ELSE IF WORDRAIP>PSEUDO1 AND WORDRAIP<PSEUDO2 THEN DO;
        SUBSTR(&OUTVAR,PSEUPTR) = &INARRAY{WORDRAIP};
        PSEUPTR + &INLEN;
      END;
      ELSE DO; /* WORDRAIP>PSEUDO1 AND WORDRAIP=PSEUDO2 */
        SUBSTR(&OUTVAR,PSEUPTR) =
          SUBSTR(&INARRAY{WORDRAIP},1,PSEUDOR2);
      END;
    END;
  %END;
%MEND;

/*****
*   START OF SAS CODE.
*****/
PSEUDOLC = ' '; /* SET LENGTH WITH ASSIGNMENT SO NO WARNING */
PSEUDOF1 = ' '; /* ASSOC. WITH MULT. LENGTHS IF MULT. USE */
DROP WORDMAXC PSEUDOLC PSEUDOF1 WORDRPPS WORDRPPE PSEUPTR
WORDRAIP WORDRAOP PSEUDO1 PSEUDOR1 PSEUDO2 PSEUDOR2
PSEUDOE PSEUDOJ WORDRSLT;
ARRAY &INARRAY (*) &INBASE.1 - &INBASE.&INVAR;
ARRAY &OUTARRAY (*) $&OUTLEN &OUTBASE.1 - &OUTBASE.&OUTVAR;
/* CLEAR OUT OUTPUT VARS - IN CASE OF RETAIN */
%IF &CLEAROUT=YES %THEN %DO;
DO WORDRAOP = 1 TO DIM(&OUTARRAY);
  &OUTARRAY{WORDRAOP} = ' ';
END;
%END;
/* FOR EFFICIENCY SET UP STOP POINT: WORDMAXC */
WORDRAOP = 0;
/* FIND FIRST BLANK TEXT LINE */
DO WORDRAIP = 1 TO &INVAR UNTIL(&INARRAY{WORDRAIP}=' ');
END;
IF WORDRAIP = 1 THEN GOTO &WORDEXIT; /* FIRST INPUT IS BLANK */

```

```

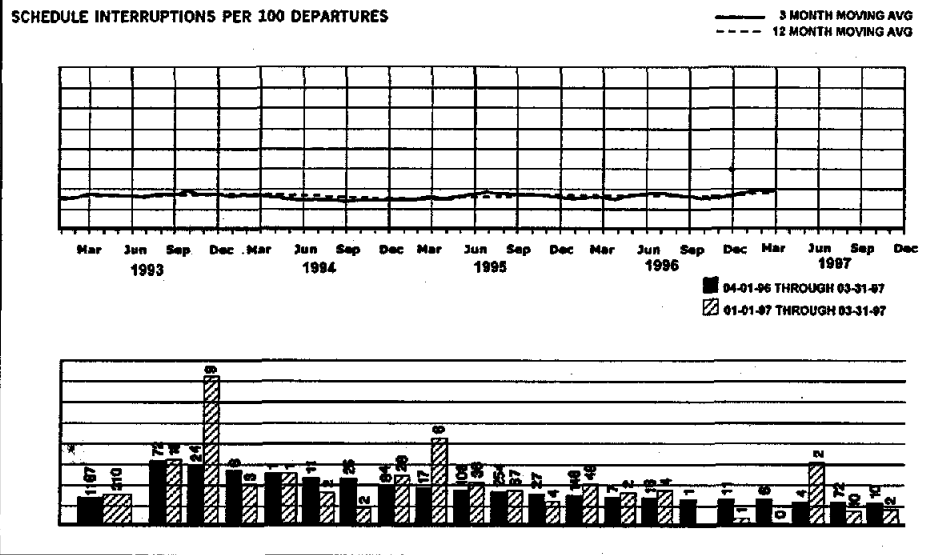
ELSE DO;
  WORDRAIP = WORDRAIP - 1; /* POINT AT LAST ENTRY */
  WORDMAXC = LENGTH(&INARRAY{WORDRAIP}) + (WORDRAIP-1) * &INLEN;
END;
WORDRAIP = 1; /* INITIALIZE POINTERS */
WORDRAOP = 1;
WORDRPPS = 1;
/* INFINITE LOOP - UNTIL NO MORE INPUT TEXT OR OUTPUT TEXT FULL */
DO UNTIL(0);
  /* CONTINUOUSLY TRACK WHERE IN 5000 BYTES YOU ARE. */
  /* USE %PSEUDO TO DO SUBSTRINGS ON 5000 BYTES */
  /* FIND ENDING POINT IN PSEUDO-STRING */
  /* ALREADY KNOW STARTING POINT. */
  WORDRPPE = WORDRPPS + &OUTLEN - 1;
  IF WORDRPPE > WORDMAXC THEN WORDRPPE = WORDMAXC;
  %PSEUDO(WORDRPPE,1,PSEUDOLC); /* GET LAST CHAR */
  IF WORDRPPE < WORDMAXC THEN DO;
    %PSEUDO(WORDRPPE+1,1,PSEUDOFC); /* GET FIRST CHAR */
  END;
  ELSE PSEUDOFC = ' ';
  /* IF LAST CHAR OF CURRENT STRING AND FIRST CHARACTER OF NEXT */
  /* STRING ARE BOTH NON-BLANK, THEN IN MIDDLE OF A WORD. */
  IF PSEUDOLC NE ' ' AND PSEUDOFC NE ' ' THEN DO;
    /* LOOK BACKWARDS FOR A BLANK AS MUCH AS 20 CHARACTERS */
    /* UNTIL A BLANK IS FOUND OR HAVE GONE BACK HALF A STRING */
    PSEUDOJ = 0;
    DO UNTIL(0);
      PSEUDOJ + 1;
      WORDRPPE + (-1);
      %PSEUDO(WORDRPPE,1,PSEUDOFC);
      IF PSEUDOFC = ' ' THEN GOTO &WORDSKIP;
      IF PSEUDOJ = 20 THEN GOTO &WORDSKIP;
      IF PSEUDOJ > &OUTLEN/2 THEN GOTO &WORDSKIP;
    END;
    &WORDSKIP:
    /* SET TO FULL LENGTH IF NO BLANK FOUND - WORD WILL BE SPLIT */
    IF PSEUDOFC NE ' ' THEN WORDRPPE = WORDRPPS + &OUTLEN - 1;
  END;
  /* NOW LOAD VALUE INTO OUTVAR */
  %PSEUDO(WORDRPPS,WORDRPPE-WORDRPPS+1,&OUTARRAY{WORDRAOP});
  /* SET VALUE OF NEXT START AT FIRST NON-BLANK CHARACTER */
  /* THIS INSURES OUTPUT TEXT ALWAYS LEFT JUSTIFIED */
  WORDRPPS = WORDRPPE;
  PSEUDOFC = ' ';
  DO WHILE(PSEUDOFC=' ');
    WORDRPPS + 1;
    IF WORDRPPS > WORDMAXC THEN DO;
      WORDRSLT = 'INVARs EMPTY';
      GOTO &WORDEXIT;
    END;
    %PSEUDO(WORDRPPS,1,PSEUDOFC);
  END;
  /* INCREMENT OUTVAR POINTER - STOP IF FULL */
  WORDRAOP = WORDRAOP + 1;
  IF WORDRAOP > &OUTVARS THEN DO;
    WORDRSLT = 'OUTVARS FULL';
    GOTO &WORDEXIT;
  END;
END;
&WORDEXIT:;
%EXIT: %MEND;

```

SCHEDULE INTERRUPTION SUMMARY

757

27 FLIGHT CONTROLS



BCAG Reliability & Maintainability Engineering
Report ID: RM21116

01-01-93 Through 03-31-97

Delays > 15 Min, Cancellations, Air Turnbacks, Diversions
Page 1

08 - 28 - 97