IMPLEMENTING INTERACTIVE MULTIPURPOSE VIEW/3000 DISPLAYS

A.J. FEDANZO JR.
BECHTEL POWER CORPORATION
SAN FRANCISCO, CA

INTRODUCTION:

In many applications it is desirable to use a single formatted screen display for both data entry and data retrieval. With the VIEW/3000 subsystem, rapid development and reliable utilization of formatted screens by user programs is simple and straightforward. Unfortunately, VIEW does not provide an equally simple method of communications between the program and the user without going through a 'menu' type approach (which typically involves selecting a function on one screen display and executing it on another, or which restricts the user to a limited set of soft-key operations). At Bechtel Power Corporation, we have developed in SPL/3000 a simple, easy to use method of user-to-program interactive communications with an essentially unlimited function request capability. We've added a user 'request line' with each VIEW screen display to facilitate user-to-program communications in addition to the VIEW 'window line.' By means of a command interpreter within the user program, control of the VIEW screen being displayed, data entry, update, error correction and data retrieval functions are possible without ever changing screens, consulting a menu screen or employing soft-keys.

A simple design strategy is behind implementation of the request line. A procedure is called to open the 264X terminal as both a VIEW device (using VOPENTERM) and as an additional terminal device with separate input and output file numbers. This second file opening is directed to open $STDIN and $STDLIST, which correspond to the same physical session device as that opened by VOPENTERM. The program then has explicit access to one and the same terminal by two logical paths: VIEW/3000 intrinsics and two MPE file references.

Once the user's terminal is opened by the program with these two logical paths, the VIEW intrinsics are empoyed to handle all the screen I-O expected for data entry, update, retrieval, etc., while the second logical path is used as the interactive request line for user-to-program communications. In this way user commands to accept data, find data and the like, can be executed by alternating between request I-O mode and VIEW mode, but without the need to change screens. Commands can be as many and as complex as the program's command interpreter allows. Use of the MYCOMMAND intrinsic permits a very great variety of user defined commands.

It should be noted that this technique of single or few screen interactive processing is entirely independent of whether the user program interfaces with an IMAGE/3000 data base (as at Bechtel), a KSAM/3000 file set, or plain MPE files. A key feature is that with versatile commands capability, VIEW screens may be designed to serve multiple purposes, thus eliminating the need for many screen set-ups, simplicty and generality of screen design and programming ease. The technique adds little additional data stack or code overhead to user programs other than the array and miscellaneous variables needed to buffer request line data and the file control overhead associated with opening two additional logical devices per user process.

Since nothing is ever as easy as it sounds, the full SPL code listing of a simple program demonstrating the request line technique is given in the Appendix. An experienced programmer can enhance this code to interface it with data files and add a suitable command interpreter of greater sophistication. Since the code is in SPL, it may call or be called by COBOL/3000, FORTRAN/3000 or other language applications which may at the user's discretion do the actual files interface, command interpretation and so forth. The program given in the Appendix is a "plain vanilla", obvious illustration of the request line technique. It is NOT given as an example of SPL programming style. Actual applications may require optimization to meet specific environmental requirements and your own programming style preferences.

OVERVIEW OF THE SAMPLE PROGRAM

The program in the Appendix assumes VIEW line 24 is the 'window line' and 30 characters of line 22 (columns 10 - 41) are reserved for request mode communications. The request line has been defined arbitrarily at 30 characters long. It could obviously be of any useful length and occur anywhere on any of the screens rather than in the same location on all (which simplifies its use for users). If you change the request line location, the screen coordinate parameters for accessing it (row and column) must be changed each time a new screen is displayed.

How the VIEW/3000 subsystem can be interrupted with damage depends upon the intelligence of the MPE III operating system and its file handlers and terminal drivers. For applications programming all that you need assure is that the DL-DB data stack area is left alone for use by VIEW, and that its terminal control conditions are restored properly after each request mode activity.

Both VIEW and the request line handlers transmit in block mode. While out of VIEW mode, it is necessary to turn off format mode, then turn it on again when re-entering VIEW mode. The proper sequence of FREAD and FWRITE intrinsic calls therefore is critical. For example, the keyboard must be enabled after a VREADFIELDS call. Hence, an FWRITE(OUT,ENABLE'KEYBOARD,-2,%0); instruction is issued after each VREADFIELDS and before returning to request mode.

The basic state of operation of the program is request mode; that is, VIEW intrinsics are called to accomplish tasks initiated by user requests, then control is returned to the request mode procedures to get the next user request. Thus VIEW/3000 intrinsics are treated like IMAGE/3000 or other MPE intrinsics.

Going in and out of request mode necessitates numerous FWRITE's and FREAD's to the terminal and the therefore generates more I-O terminal interrupts than when simply using VIEW alone. However, since the relative frequency of requests is fairly low per on-line user in most cases, the added I-O interrupt overhead seems to be acceptable. Individual installations will of course need to make their own assessment of this additional I-O traffic.

It is worth mentioning that the same code that places the cursor at the request line can be used to place the cursor anywhere in any unprotected screen data field. This allows you to enter data starting at that field, then pressing ENTER, VIEW will still read the entire screen successfully. The strategy of entering and leaving VIEW mode is the same.

OPENING THE TERMINAL

After VOPENTERM has successfully opened the terminal, two FOPEN calls are issued to establish the request mode I-O capability. The input logical device is FOPEN'ed with FOPTIONS:=%45; that is, as $STDIN, ASCII, OLD PERMANENT FILE. Input AOPTIONS are %0; that is, READ only. The output logical device is opened with both FOPTIONS and AOPTIONS. The FOPTIONS for output are %415; that is, with CCTL, FIXED, $STDLIST, ASCII, OLD PERMANENT FILE. The output AOPTIONS are %1; that is, WRITE only.

With these two logical paths opened, an 'RS' (Record Separator) is written at the end of the request line in order to provide a machine recognizable transmission delimiter for non-VIEW data reads.

DISPLAYING THE REQUEST LINE CONTENTS

Once the terminal is open for both VIEW and request mode, the program may begin interactive processing. Typically this consists of reading and writing to the request line. Writing to the request line is done as follows.

Firstly, the request line buffer is cleared either to be written out to clear the screen, or to recieve some string. The format mode is turned off. The cursor is positioned to the start of the request line and the full 30 characters of the request line buffer are written. Since the next event will still be in request mode, format mode is left off.

READING THE REQUEST LINE

Obtaining data the user types on the request line requires a block mode data transfer. Because the cursor will be positioned after the last character typed, a dummy FREAD will be issued, the cursor repositioned at the start of the request line and the actual data transfer done. The GET'REQUEST procedure uses the MPE PAUSE to suspend execution while the terminal I-O is completed on 300 and 1200 baud lines. This is unnecessary with faster lines and only the last PAUSE may be required depending on your terminal data transmission rate and other site characteristics. Reading the request line proceeds as follows.

Firstly, the cursor is positioned to the start of the request line so that data may be entered. When the 'ENTER' key is pressed, an FREAD is begun to satisfy the I-O request which 'ENTER' creates. This is a dummy data transfer from the end of the character string just typed to the 'RS'.

Secondly, once a dummy FREAD is done, the cursor is repositioned to the start of the request line and a byte string transmitted to the terminal to force transmission of the full request line contents. This is the actual data transfer. Data thus read is passed to an optional subroutine to be left justified prior to being evaluated by the command interpreter.

INTERPRETING THE REQUEST COMMAND

Once reading and writing to the request line is possible, all that remains to act upon the contents the user entered. The sample program in the Appendix only uses a simplistic IF....THEN.... type of command interpretation. Our actual applications employ the MPE 'MYCOMMAND' intrinsic to allow commands with parameters. The allowed commands for the sample program are:

```
"X"           =  fill screen with X's to show display works
"ST" or "EX"  =  exit the program
"PHOTO"       =  list screen contents to line printer
"GET FORM1"   =  display the first form
"GET FORM2"   =  display the second form
"GET FORM3"   =  display the third form
"ADD"         =  add data; go to first field and read user input
                         after 'ENTER' is pressed (will use the
                         VIEW/3000 edits, if any)
```

Erroneous or undefined commands result in a message "COMMAND NOT DEFINED - TRY AGAIN" being displayed in the VIEW window area.

Increasing the sophistication of requests (commands) and program responses is something for each programmer to establish. Once it has been shown that data can be successfully read from and written to the screen in both VIEW and request modes, what to do with it is up to the needs of each application's design.

SUMMARY

The request mode technique enhances VIEW/3000 subsystem utility in cases where multipurpose screens are used and where complex and frequent user-to-program communications are desired. It is implemented by a readily modified SPL/3000 application program using standard HP3000 system intrinsics. The technique introduces little data and code segment overhead of its own and can interface with programs written in other languages.

```
$CONTROL USLINIT, SEGMENT=MAIN
$TITLE " *** REQUEST MODE DEMONSTRATION PROGRAM *** "
BEGIN
COMMENT:

   ********************************************************
   *                                                      *
   *   PROGRAM ENABLES THE USE OF "REQUEST MODE" WITH     *
   *   VIEW/3000 CRT FORMS (COMPILED FAST OR NOT).        *
   *   THIS IS ONLY A DEMONSTRATION OF THE TECHNIQUE.     *
   *                                                      *
   *   THE PROGRAM ASSUMES THREE VIEW FORMS ARE IN A FORM *
   *   FILE WHOSE NAME MUST BE COMPILED INTO THE PROGRAM. *
```

```
*   ALSO INCLUDE THE NAME OF ALL THREE FORMS IN THE      *
*   APPROPRIATE ARRAYS - THESE FORMS MAY BE ENTIRELY     *
*   TRIVIAL EXAMPLES OF VIEW FORMS.                      *
*                                                        *
*   BE SURE EACH VIEW FORM HAS A POSITION AVAILABLE FOR  *
*   THE "REQUEST LINE" ON SCREEN LINE 22.  IF YOU TYPE   *
*   THE STRING "REQUEST:" ON THAT LINE, START IN COL. 1. *
*                                                        *
*   YOU MUST SET THE "MAXBUF" PARAMETER BEFORE COMPILING *
*   AND SHOULD SET VALUES FOR "BUFLEN" FOR EACH SCREEN   *
*   IN THE GET'NEXT'FORM PROCEDURE. FOR SIMPLICITY,      *
*   YOU CAN SIMPLY LET BUFLEN:=MAXBUF/2.  HOWEVER, THIS  *
*   IS NOT OPTIMAL, SINCE IT WILL ALWAYS FORCE THE VIEW  *
*   INTRINSIC "VPUTBUFFER" TO TRANSFER THE SAME NUMBER   *
*   OF BYTES IRRESPECTIVE OF A PARTICULAR FORM'S DATA    *
*   AREA SIZE (IN BYTES).                                *
*                                                        *
*   THE PROGRAM'S DATA STACK IS FAIRLY SMALL, AS YOU'LL  *
*   FIND AT COMPILE TIME.  SINCE VIEW/3000 FORMS USE     *
*   A LARGE DL-DB AREA, PREP THIS PROGRAM WITH A MINIMUM *
*   MAXDATA=8192, JUST TO BE SAFE. INCREASE THE MAXDATA  *
*   PARAMETER IF NEEDED.                                 *
*                                                        *
**********************************************************
*                                                        *
*   PROGRAM WRITTEN BY -- A.J. FEDANZO JR.               *
*           12/10/79      DEVELOPMENT & PROGRAMMING      *
*                         BECHTEL POWER CORPORATION      *
*                         SAN FRANCISCO, CA              *
*                                                        *
**********************************************************

;

DEFINE VERS="   REQUEST MODE DEMONSTRATION  VERSION 01/23/80 "#;
DEFINE LA=LOGICAL ARRAY#, BA=BYTE ARRAY#;

DEFINE CLEAR'IN'BUF=BEGIN IN'BUF:=%20040; X'PTR:=0;
             MOVE IN'BUF(X'PTR+1):=IN'BUF(X'PTR),(MAXBUF); END;#;

DEFINE CLEAR'REQ'BUF=BEGIN REQ'BUF:=%20040; X'PTR:=0;
             MOVE REQ'BUF(X'PTR+1):=REQ'BUF(X'PTR),(14); END;#;

$PAGE
EQUATE MAXBUF=200; <<SET = TO #BYTES/2 IN LARGEST SCREEN DATA AREA>>
```

```
LA IN'BUF(0:MAXBUF);              << SCREEN I-O BUFFER AREA >>
BA IN'BY(*)=IN'BUF;
LA COMAREA(0:59);                 << VIEW/3000 COMMUNICATION AREA >>
LA ERRMSG(0:35);                  << MESSAGE BUFFER FOR VIEW 'WINDOW' >>
BA ERRMSG'BY(*)=ERRMSG;
LA RS'POS(0:5);                   << SCREEN ADDRESS OF 'RS' >>
BA RS'POS'BY(*)=RS'POS;
LA HANDLE'RS(0:1);                << ARRAY CONTAINING 'RS' FOR FWRITE >>
BA HANDLE'RS'BY(*)=HANDLE'RS;
LA REQUEST'POS(0:4);              << ADDRESS OF START OF REQUEST LINE >>
BA REQUEST'POS'BY(*)=REQUEST'POS;
LA REQ'BUF(0:14);                 << REQUEST LINE DATA BUFFER >>
BA REQ'BY(*)=REQ'BUF;
BA STDIN(0:7):="$STDIN ";         << REQUEST LINE INPUT LOGICAL DEV >>
BA STDLIST(0:8):="$STDLIST ";     << REQUEST LINE OUTPUT LOGICAL DEV >>
LA FORMAT'OFF(0:1):=%15530;       << ESCAPE 'X' - TURN OFF FORMAT MODE>>
LA FORMAT'ON(0:1):=%15527;        << ESCAPE 'W' TURN ON FORMAT MODE >>
LA KEYBOARD'ENABLE(0:1):=%15542;<< ESCAPE LOWER CASE 'B' >>
LA KEYBOARD'DISABLE(0:1):=%15543;  << KEYBOARD DISABLE >>
LA HOME'CURSOR(0:1):=%15510;      << ESCAPE 'H' - TO 1ST UNPROTECTED >>
LA FORCE'CRT'XMIT(0:1);           << FORCE TERMINAL TO TRANSMIT DATA >>
BA FORCE'CRT'XMIT'BY(*)=FORCE'CRT'XMIT;
LA TERM'NAME(0:1):="XYZ ";        << DUMMY TERMINAL FOR VIEW/3000 >>
BA TERM'BY(*)=TERM'NAME;
LA FORM'F(0:11):="SDRFASTS.SOURCE.PLIB  ";  << VIEW/3000 FORM FILE >>
BA FORM'FILE(*)=FORM'F;
LA FORM'N(0:4):=5("  ");          << WILL HOLD FORM NAME >>
BA FORM'NAME(*)=FORM'N;
LA FIRST'F(0:4):="SDRFORM1  "; << NAME OF FIRST FORM FILE FORM >>
BA FIRST'FORM(*)=FIRST'F;
LA SECOND'F(0:4):="SDRFORM2  ";<< NAME OF SECOND FORM FILE FORM >>
BA SECOND'FORM(*)=SECOND'F;
LA THIRD'F(0:4):="SDRFORM3  "; << NAME OF THIRD FORM FILE FORM >>
BA THIRD'FORM(*)=THIRD'F;

INTEGER X'PTR;                    << USED TO CLEAR BUFFERS >>
INTEGER I,J,L,                    << MISC. INTEGERS >>
        IN,                       << REQUEST LINE INPUT FILE NBR >>
        OUT,                      << REQUEST LINE OUTPUT FILE NBR >>
        BUFLEN,                   << #BYTES OF SCREEN DATA PER FORM>>
        TERMINAL;                 << REQUEST LINE DEVICE VARIABLE >>

LOGICAL  K, DUMMY,                << MISC. LOGICAL VARIABLES >>
         SCREEN'ONE,              << FIRST FORM FLAG >>
         SCREEN'TWO,              << SECOND FORM FLAG >>
         SCREEN'THREE;            << THIRD FORM FLAG >>
REAL PPAUSE;                      << #SECONDS TO PAUSE PROGRAM >>
```

```
INTRINSIC VOPENTERM,VCLOSETERM,VSHOWFORM,VGETNEXTFORM,
          VGETBUFFER,VPUTBUFFER,VERRMSG,VOPENFORMF,VCLOSEFORMF,
          VREADFIELDS,VGETNEXTFORM,VSETERROR,VPUTWINDOW,
          VGETFIELD,VPUTFIELD,VFIELDEDITS,VPRINTFORM,

          PRINT'FILE'INFO,PRINT,PAUSE,
          FOPEN,FCLOSE,FREAD,FWRITE,TERMINATE;

<< ************** END OF DECLARATIONS ***************** >>

$PAGE
$CONTROL SEGMENT=MISC

 PROCEDURE VIEW'ERR;
<<
   PROCEDURE HANDLES ALL VIEW/3000 ERRORS
   ABORTS PROGRAM AFTER DISPLAYING MESSAGE
>>
  BEGIN
   COMAREA(1):=3;  << LANG IS SPL >>
   COMAREA(2):=60; << COMAREA LENGTH IS 60 WORDS >>
   COMAREA(37):=0; << VIEW ERR MSG FILE NBR INITIALIZED >>

   I:=72;
   VERRMSG(COMAREA,ERRMSG'BY,I,J);  << VIEW ERR PROC >>
   PRINT(ERRMSG,-72,%60);
   VPUTWINDOW(COMAREA,ERRMSG'BY,J);

   COMAREA(0):=0;
   COMAREA(1):=3;
   COMAREA(2):=60;
   COMAREA(33):=%4;
   VSHOWFORM(COMAREA);
   VCLOSETERM(COMAREA);
   TERMINATE;
  END;
$PAGE
 PROCEDURE ERROR(FILE'NBR);
    VALUE FILE'NBR;  INTEGER FILE'NBR;
 <<
    PROCEDURE HANDLES ALL NON-VIEW/3000 FILE SYSTEM ERRORS
    ABORTS PROGRAM AFTER DISPLAYING MESSAGE
 >>
    BEGIN
       VCLOSETERM(COMAREA);
       PRINT'FILE'INFO(FILE'NBR);
       TERMINATE;          << END THE FIASCO >>
    END;
```

```
$PAGE
 PROCEDURE DISPLAY'WINDOW;
<<
   DISPLAYS THE DATA IN THE VIEW "WINDOW",
   CALLS VIEW'ERR IF IT FAILS
>>
  BEGIN
   K:=35;
   DO K:=K-1 UNTIL ERRMSG(K)<>%20040 OR K<=1;
   K:=K+K;    << GET LENGTH IN BYTES >>
   VPUTWINDOW(COMAREA,ERRMSG,K); IF COMAREA(0)<>0THEN VIEW'ERR;
   COMAREA(33):=%4;  << SHOW THE WINDOW >>
   VSHOWFORM(COMAREA);  IF COMAREA(0)<>0 THEN VIEW'ERR;
   K:=0; VPUTWINDOW(COMAREA,ERRMSG,K);IF COMAREA(0)<>0 THEN VIEW'ERR;
  END;  << OF DISPLAY'WINDOW >>

$PAGE
 PROCEDURE CLOSE'TERM;
<<
  CLOSE THE VIEW/3000 LOGICAL I-O DEVICES
>>
 BEGIN
  COMAREA(0):=0;
  VCLOSETERM(COMAREA);
     IF COMAREA(0) <> 0 THEN TERMINATE;
  VCLOSEFORMF(COMAREA);
     IF COMAREA(0) <> 0 THEN TERMINATE;
 END;
$PAGE
 PROCEDURE SETUP'REQUEST'POS;
<<
  INITIALIZE THE REQUEST LINE GLOBAL ARRAY VALUES FOR USE
  THROUGHOUT THE PROGRAM - THIS IS DONE ONCE ONLY.
>>
  BEGIN
    REQUEST'POS(0):=%15446;          << ESCAPE '&' >>
    REQUEST'POS'BY(2):=%141;         << LOWER CASE 'A' >>
    MOVE REQUEST'POS'BY(3):="21";    << ABSOLUTE ROW 22 >>
    REQUEST'POS'BY(5):=%162;         << LOWER CASE 'R' >>
    MOVE REQUEST'POS'BY(6):="10";    << ABSOLUTE COLUMN 10 >>
    REQUEST'POS'BY(8):=%103;         << CAPITAL 'C' >>

    RS'POS(0):=%15446;               << ESCAPE '&' >>
    RS'POS'BY(2):=%141;              << LOWER CASE 'A' >>
    MOVE RS'POS'BY(3):="21";         << ABSOLUTE ROW 22 >>
    RS'POS'BY(5):=%162;              << LOWER CASE 'R' >>
    MOVE RS'POS'BY(6):="41";         << ABSOLUTE COLUMN 41 >>
    RS'POS'BY(8):=%103;              << CAPITAL 'C' >>

    HANDLE'RS'BY(0):=%36;            << RECORD SEPARATOR >>
```

```
    FORCE'CRT'XMIT(0):=%15544;      << ESCAPE LOWER CASE 'D' >>
    FORCE'CRT'XMIT'BY(2):=17;       << DC1 - FORCE TRANSMISSION >>
  END;
$PAGE
 PROCEDURE GET'NEXT'FORM;
<<
  DO THE ACTUAL FORMS FILE FORM ACQUISTION HERE & DISPLAY FORM
>>
  BEGIN

   SCREEN'THREE:=SCREEN'ONE:=SCREEN'TWO:=FALSE;
   CLEAR'IN'BUF;
   MOVE COMAREA(18):=FORM'N,(5); << ARRAY WHICH EQUALS FORM'NAME >>
   COMAREA(26):=COMAREA(27):=0;
   COMAREA(1):=60;
   VGETNEXTFORM(COMAREA);
       IF COMAREA(0) <> 0 THEN VIEW'ERR;
   VPUTBUFFER(COMAREA,IN'BY,BUFLEN);
      IF COMAREA(0) <> 0 THEN VIEW'ERR;
   VSHOWFORM(COMAREA);
       IF COMAREA(0) <> 0 THEN VIEW'ERR;
<<
   NOW SET BYTE BUFFER LENGTH VALUES & WHICH SCREEN FLAGS
>>
   IF FORM'NAME=FIRST'FORM,(15) THEN
      BEGIN
       BUFLEN:=339; << NUMBER BYTES OF DATA ON SCREEN ONE >>
       SCREEN'ONE:=TRUE;
      END;
   IF FORM'NAME=SECOND'FORM,(15) THEN
      BEGIN
       BUFLEN:=400; << NUMBER BYTES DATA ON SCREEN TWO >>
       SCREEN'TWO:=TRUE;
      END;
   IF FORM'NAME=THIRD'FORM,(15) THEN
      BEGIN
       BUFLEN:=233; << NUMBER BYTES DATA ON SCREEN THREE >>
       SCREEN'THREE:=TRUE;
      END;

   FWRITE(OUT,FORMAT'OFF,-2,%320);
     IF <> THEN ERROR(OUT);
   FWRITE(OUT,RS'POS,-9,%320);<< GOTO 'RS' AT END REQ LINE >>
     IF <> THEN ERROR(OUT);
   FWRITE(OUT,HANDLE'RS,-1,%0); << WRITE THE 'RS' >>
     IF <> THEN ERROR(OUT);
  END;  << OF GET'NEXT'FORM >>
$PAGE
 LOGICAL PROCEDURE OPEN'TERM;
```

```
<<
  OPEN THE TERMINAL FOR BOTH VIEW/3000 & REQUEST LINE USE
>>
 BEGIN
  SUBROUTINE SECOND'OPEN;
    << OPEN TERMINAL AGAIN AS AN MPE I-O DEVICE >>
    BEGIN
     IN:=FOPEN(STDIN,%45);
       IF <> THEN ERROR(IN);
     OUT:=FOPEN(STDLIST,%415,%1);
       IF <> THEN ERROR(OUT);
    END;

 OPEN'TERM:=FALSE;
  I:=0; DO BEGIN COMAREA(I):=0; I:=I+1; END UNTIL I = 60;

  COMAREA(1):=3;     << LANGUAGE IS SPL >>
  COMAREA(2):=60;    << COMAREA LENGTH IS 60 WORDS >>
 COMAREA(33):=%4;     << DISPLAY SCREEN DATA >>
  VOPENTERM(COMAREA,TERM'BY);
     IF COMAREA(0) <> 0 THEN VIEW'ERR;
  SECOND'OPEN;  << NOW OPEN TERMINAL FOR REQUEST LINE I-O >>

  VOPENFORMF(COMAREA,FORM'FILE);
     IF COMAREA(0) <> 0 THEN VIEW'ERR;

 OPEN'TERM:=TRUE;
END;

$PAGE
$CONTROL SEGMENT=MAIN

  PROCEDURE DISPLAY'REQUEST;
 <<
   WRITE THE REQUEST BUFFER CONTENTS TO THE REQUEST LINE
 >>
    BEGIN
    FWRITE(OUT,FORMAT'OFF,-2,%0); << TURN OFF FORMAT MODE >>
      IF <> THEN ERROR(OUT);

    FWRITE(OUT,REQUEST'POS,-9,%320); << POSITION CURSOR AT START >>
        IF <> THEN ERROR(OUT);       << OF REQUEST LINE >>

    FWRITE(OUT,REQ'BUF,12,%0);   << WRITE TO REQUEST LINE >>
      IF <> THEN ERROR(OUT);

    END;
$PAGE
  LOGICAL PROCEDURE GET'REQUEST;
```

```
<<
  READ THE REQUEST LINE DATA ON THE SCREEN INTO REQUEST BUFFER

  PROCEDURE USES THREE 'PAUSE' INTRINSIC CALLS TO ASSURE SUCCESS
  IN DATA TRANSFER WITH 300 BAUD & 1200 BAUD DIAL-UP LINES

  FOR ALL 1200 BAUD TRANSMISSIONS, REMOVE THE FIRST TWO
>>
  BEGIN

    SUBROUTINE LEFT'JUSTIFY;
     << LEFT JUSTIFY REQUEST LINE DATA FOR COMMAND INTERPRETER >>
     BEGIN
      K:=0; WHILE REQ'BY(K)=" " AND K<30 DO K:=K+1;
      IF K=0 THEN GO GO'ROUND;
      << TEST IF K IS ODD OR EVEN >>
      IF K THEN
        BEGIN
         J:=30-K;
         MOVE REQ'BY(0):=REQ'BY(K),(J);  << DO LEFT SHIFT >>
         DO BEGIN REQ'BY(J):=" "; J:=J+1; END UNTIL J>=29;
        END
      ELSE
        BEGIN
         ASSEMBLE(LOAD K; LSR 1; STOR K;); << FAST DIVIDE BY TWO >>
         J:=15-K;
         MOVE REQ'BUF(0):=REQ'BUF(K),(J);
        END;
      GO'ROUND:
     END;

    GET'REQUEST:=FALSE;

    FWRITE(OUT,REQUEST'POS,-9,%320); << MOVE TO START REQ. LINE >>
       IF <> THEN ERROR(OUT);

    TERMINAL:=FREAD(IN,REQ'BUF,1); << FIRST READ, DUMMY TRANSFER >>
      IF <> THEN ERROR(IN);
     PAUSE(PPAUSE);        << DELAY TO LET TERM I/O CATCH UP >>
    FWRITE(OUT,REQUEST'POS,-9,%320);  << MOVE TO START REQ. LINE >>
       IF <> THEN ERROR(OUT);
     PAUSE(PPAUSE);         << DELAY TO LET TERM I/O CATCH UP >>
    FWRITE(OUT,FORCE'CRT'XMIT,-3,%320);<< FORCE DATA TRANSMISSION >>
      IF <> THEN ERROR(OUT);
     PAUSE(PPAUSE);         << DELAY TO LET TERM I/O CATCH UP >>
    TERMINAL:=FREAD(IN,REQ'BUF,15);
      IF <> THEN ERROR(IN);
```

```
      FWRITE(OUT,FORMAT'ON,-2,%0);
      LEFT'JUSTIFY;
      GET'REQUEST:=TRUE;
    END;
$PAGE
 PROCEDURE SHOW'SCREEN;
  BEGIN
 <<
   MOVE PROGRAM DATA BUFFER TO VIEW BUFFER AREA & DISPLAY
   NUMBER OF BYTES MOVED DEPENDS ON BUFLEN
 >>
   COMAREA(0):=0; COMAREA(1):=3; COMAREA(2):=60;
   VPUTBUFFER(COMAREA,IN'BY,BUFLEN);
     IF COMAREA(0) <> 0 THEN VIEW'ERR;
   COMAREA(33):=%2;  << FORCE DATA & WINDOW DISPLAY >>
   VSHOWFORM(COMAREA);
     IF COMAREA(0) <> 0 THEN VIEW'ERR;
  END;
$PAGE
 PROCEDURE PERFORM'VIEW'EDITS;
  BEGIN
<<
   USE VIEW/3000 EDITS HERE
>>
   COMAREA(1):=3;              << LANGUAGE IS SPL >>
   COMAREA(2):=60;             << VIEW BUFFER AREA = 60 WORDS >>

 DO                            << DO EDITS UNTIL NO ERRS >>
  BEGIN

   VFIELDEDITS(COMAREA); IF COMAREA(0)<>0 THEN VIEW'ERR;
   I:=0; DO BEGIN ERRMSG(I):=%20040; I:=I+1; END UNTIL I >= 35;


   I:=72;      << TELL VIEW ERRMSG BUFFER IS 72 BYTES LONG >>
   COMAREA(37):=0;                 << INIT ERRFILENUM >>
   VERRMSG(COMAREA,ERRMSG'BY,I,J); << HAVE VIEW GET MSG >>
    IF COMAREA <> 0 THEN VIEW'ERR;
   VPUTWINDOW(COMAREA,ERRMSG'BY,J);<< MOVE MSG TO WINDOW >>
    IF COMAREA <> 0 THEN VIEW'ERR;
   COMAREA(33):=%4;                << FORCE WINDOW TO DISPLAY >>
   VSHOWFORM(COMAREA); IF COMAREA(0)<> 0 THEN VIEW'ERR;

   VREADFIELDS(COMAREA); IF COMAREA(0)<>0 THEN VIEW'ERR;

   VGETBUFFER(COMAREA,IN'BY,BUFLEN); IF COMAREA<>0 THEN VIEW'ERR;

  END  UNTIL COMAREA(6) = 0;   << ERR FLAG NOT SET >>
 END;  << OF PERFORM'VIEW'EDITS >>
```

```
$PAGE
 PROCEDURE GET'SCREEN'DATA;
 <<
   GET DATA FROM SCREEN BY LEAVING REQUEST MODE,
   TURN ON FORMAT MODE FOR VIEW,
   HOME CURSOR TO FIRST UNPROTECTED FIELD ON TOP OF SCREEN,
   HAVE VIEW READ THE DATA,
   PERFORM ANY VIEW/3000 EDITS,
   MOVE DATA FROM VIEW BUFFER TO PROGRAM DATA AREA,
   ENABLE THE KEYBOARD THAT VIEW DISABLED.
 >>
  BEGIN

   FWRITE(OUT,FORMAT'ON,-2,%0);
     IF <> THEN ERROR(OUT);
   FWRITE(OUT,HOME'CURSOR,-2,%320);
     IF <> THEN ERROR(OUT);
   VREADFIELDS(COMAREA);
      IF COMAREA(0) <> 0 THEN VIEW'ERR;
   PERFORM'VIEW'EDITS;  << RETURNS IF NONE SPECIFIED >>
   COMAREA(1):=3; COMAREA(2):=60;
   VGETBUFFER(COMAREA,IN'BY,BUFLEN);
     IF COMAREA(0) <> 0 THEN VIEW'ERR;
   FWRITE(OUT,KEYBOARD'ENABLE,-2,%0);
     IF <> THEN ERROR(OUT);
  END; << OF GET'SCREEN'DATA >>
$PAGE
$CONTROL SEGMENT=PHOTO'CRT
 PROCEDURE  PHOTO;
   BEGIN

 COMMENT ****************** PHOTO ************************

         COPYS CRT VIEW DISPLAY TO LINE PRINTER

 CALL:    PHOTO

 FORMAL PARAMETERS: NONE

 DESCRIPTION OF FUNCTION:  COPYS CRT SCREEN FORMAT AND DATA
                           TO LP FILE. USES THE VIEW
                           VPRINTFORM INTRINSIC.  FILE PRINTS
                           WHEN SESSION ENDS.

 RETURN VALUE: NONE

 ERROR MESSAGES GENERATED:  NONE

 ********************************************************** ;
```

```
 I:=0;          << PRINT CONTROL - NO UNDERLINING >>
 J:=%61;        <15>

 VPRINTFORM(COMAREA,I,J);    << PRINT FORM ON LP >>
  IF COMAREA <> 0 THEN VIEW'ERR;

 MOVE ERRMSG:=30(%20040);
 MOVE ERRMSG:="SCREEN LISTED ON LINE PRINTER";
 DISPLAY'WINDOW;

END;    << OF PHOTO >>
$PAGE "            M A I N"
$CONTROL SEGMENT=MAIN
<< ************** M A I N  P R O G R A M *************** >>

 BEGIN
     MOVE IN'BUF:=VERS;
     PRINT(IN'BUF,-56,%60);
        IF <> THEN TERMINATE;

     PPAUSE:=.2;          << REQUEST LINE PAUSE DELAY SET >>
     SETUP'REQUEST'POS;  << INITIALIZE REQUEST CURSOR POSITION >>

    IF OPEN'TERM THEN
         BEGIN
          MOVE FORM'NAME:=THIRD'FORM,(15);
          GET'NEXT'FORM;
         MOVE ERRMSG:="REQUEST MODE DEMO PROGRAM READY";
         DISPLAY'WINDOW;
         END
    ELSE
         GO END'PROG;

 LOOP:
    CLEAR'REQ'BUF;
    DISPLAY'REQUEST;
    GET'REQUEST;
    IF REQ'BY = "ST" OR REQ'BY = "EX" THEN GO END'PROG;

    IF REQ'BY = "GET FORM1" THEN << DEMONSTRATE FORM CHANGE >>
       BEGIN
         MOVE FORM'NAME:=FIRST'FORM,(15);
        GET'NEXT'FORM;
        GO LOOP;
       END;
    IF REQ'BY = "GET FORM2" THEN << DEMONSTRATE FORM CHANGE >>
       BEGIN
        MOVE FORM'NAME:=SECOND'FORM,(15);
        GET'NEXT'FORM;
        GO LOOP;
       END;
```

```
    IF REQ'BY = "GET FORM3" THEN << DEMONSTRATE FORM CHANGE >>
       BEGIN
        MOVE FORM'NAME:=THIRD'FORM,(15);
        GET'NEXT'FORM;
        GO LOOP;
       END;

    IF REQ'BY = "X" THEN << DEMONSTRATE COMMAND TO WRITE TO SCREEN >>
       BEGIN
        K:=0;DO BEGIN IN'BUF(K):="XX";K:=K+1; END UNTIL K>=MAXBUF;
        SHOW'SCREEN;
        GO LOOP;
       END;

    IF REQ'BY = "AD" THEN << DEMONSTRATE GETTING DATA FROM SCREEN >>
       BEGIN
        GET'SCREEN'DATA;
        MOVE REQ'BUF:=IN'BUF,(15);
        GO LOOP;
       END;

    IF REQ'BY = "PH" THEN  << RUN PHOTO PROCEDURE >>
       BEGIN
        PHOTO;
        GO LOOP;
       END;

<< HANDLE UNDEFINED COMMANDS HERE >>

    MOVE ERRMSG:=" COMMAND NOT DEFINED - TRY AGAIN!";
    DISPLAY'WINDOW;
    GO LOOP;


  END'PROG:

  CLOSE'TERM;                  << CLOSE VIEW/3000 LOGICAL DEVICES >>
  FCLOSE(IN,0,0);              << CLOSE REQUEST MODE INPUT DEVICE >>
  FCLOSE(OUT,0,0);             << CLOSE REQUEST MODE OUTPUT DEVICE >>
  END;
END.
```