

Simplifying Screen Specifications – the ‘Full Screen Manager’ Interface and ‘Screen Form’ Generating Routines

F. J. DIXON*

Scottish Office Computer Service, Broomhouse Drive, Edinburgh**

This paper describes some routines developed to provide ways of defining and accessing Visual Display Unit (VDU) screen fields which are easier to use than those provided by the manufacturer's software. Although the details may be of interest only to those running interactive FORTRAN programs under IBM's VSPC (Virtual Storage Personal Computing), the general principles may be usefully applied elsewhere: experience suggests that the long-term benefits of having good 'service' routines available greatly outweigh the initial costs of setting them up. Section 1 provides a little background, section 2 outlines the screen-control facilities provided by IBM for use by VSPC FORTRAN programs, section 3 describes the 'Interface' routines developed to facilitate use of IBM's facilities and section 4 covers some routines which generate 'screen forms' from specifications held in 'card-image' files. Finally, section 5 gives examples of the kinds of application for which the routines are used and how they have evolved over the years.

1. BACKGROUND

VSPC provides users of terminals connected to a mainframe with facilities for 'personal computing' (they can program in, say, BASIC or APL or FORTRAN and maintain their own files of data), Remote Job Entry (on-line set-up, submission and, in due course, examination of the results of batch jobs) and the interactive running of some packages. The Scottish Office Computer Service (SOCS) has also used VSPC to develop and install interactive systems for specific purposes: economic and statistical models and some small, generally single-user 'administrative' systems for use by clerical staff (other IBM software being more appropriate for larger, multi-user applications). Good 'screen form' data entry and amendment facilities are required for 'record-keeping' systems and it should be possible to provide them without excessive programming effort.

VSPC FORTRAN is a dialect of FORTRAN IV and includes many non-standard features found in other IBM implementations of FORTRAN IV: it is very similar to IBM FORTRAN IV G1. In addition, VSPC 'operating system' facilities may be invoked from within a VSPC FORTRAN program using a special routine called OPSYS (which is provided as part of the language along with the usual mathematical functions etc.): by CALLing OPSYS (with the appropriate arguments) various VSPC commands can be issued while a program is running (e.g. list all files in the library, create new file called X, allocate file named Y to unit Z, terminate the session etc.). One of OPSYS' other purposes is to allow the programmer access to the screen-control facilities required to set up 'screen forms' – ordinary READ/WRITE statements alone are insufficient as their input and output march down the screen, line after line, as if a teletype were being used.

2. IBM FACILITIES

IBM's Full Screen Manager (FSM) can be used to specify the characteristics of fields on an IBM 3270 (or equivalent) VDU screen:

* Any views expressed by the author are not necessarily shared by his employer.

** Present address: Scottish Education Department, 43 Jeffrey Street, Edinburgh.

- (a) position – row and column of start of field,
- (b) size – length of field in characters,
- (c) type of data that may be entered (if any) – fields may be:
 - (i) 'protected': user cannot overwrite their contents (e.g. headings and messages),
 - (ii) 'numeric': only digits, decimal points and minus signs may be entered,
 - (iii) 'character': any available character may be entered.
- (d) type of display – normal intensity or highlighted or display suppressed (e.g. for entry of passwords).

and transfer data between the screen fields and program's internal storage.

After entering any data in the 'unprotected' fields the user presses a control key (such as 'ENTER' or one of the 'programmable function keys'). FSM can be used to determine which control key was pressed and the position of the cursor on the screen at that time, allowing the user to initiate further processing (e.g. print current screen, select from 'menu', etc.) 'at the touch of a button'.

FSM can also be used to specify the position of the cursor on the screen and offers other facilities which are not described here.

FSM is used from VSPC FORTRAN programs by calling IBM's OPSYS routine. Unfortunately, complicated 'control variables' (containing codes for the field attributes, etc.) must be set up and the resulting code is not particularly comprehensible. For example, to display the message 'FULL SCREEN TEST' on the 9th line of the screen and read a single character field from the 23rd line, IBM suggest the 13 statements shown in Fig. 1 (about which the less said the better). Clearly, setting up a more complicated 'screen form' with, say, a dozen different fields, side-headings and messages to the user would be a major task using the OPSYS routine and the resulting code would be difficult to maintain.

3. FSM INTERFACE ROUTINES

The facilities offered by FSM are potentially very useful but the effort required to use them through the IBM OPSYS routine (and the difficulty of later modifying programs) prevented them from being exploited at SOCS until the FSM Interface routines were developed at

```

10 dimension ictl(18), ictlr(18), idat(8), idatr(8)
20 data ictl/0, 23, 1, 16, 0, 0, 269, 9, 1, 16, 1, 16, 12, 23, 1, 1, 0, 1/
30 data idat(1)/'full'/, idat(2)/'scr'/, idat(3)/'een' /, -
40 idat(4)/'test' /
50 data ictlr/0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0/
60 ictlsz = 18
70 idatln = 16
80 ireq = 64
90 call opsys ('fsm', ireq, irtn, irsn, ictl, ictlsz, idat, -
100 idatln)
110 ireq = 32
120 ictlsr = 12
130 idatlr = 32
140 call opsys ('fsm', ireq, irtn, irsn, ictlr, ictlsr, idatr, -
150 idatlr)
160 end

```

Figure 1. IBM example of the use of the 'OPSYS' routine. The example uses the FSM function of OPSYS, together with the assignment statements needed to assign values to its sub-parameters, to display the message **FULL SCREEN TEST** on the 9th row of the screen, and to read a one-position input field from the 23rd row of the screen. (Note: the above program is in IBM free-format VSPC FORTRAN – a dialect of FORTRAN IV).

SOCS. These routines are written in VSPC FORTRAN and allow programmers to use FSM's facilities quickly and easily. Detailed knowledge of the OPSYS 'control variables' is not required – they are hidden from the programmer by the Interface routines.

3.1. Clearing the screen and displaying 'menus', etc.

The screen is cleared using routines which can set it up in such a way that either the cursor will automatically skip over unused or protected areas of the screen (if the user types in past the end of one field the cursor jumps to the start of the next unprotected one) or cursor control keys must be used to move the cursor from one field to the next (making accidental entry of data in the wrong field less likely): the choice is up to the programmer. If required, the initial display can be taken from a text file, simplifying the task of setting up a 'menu' or block of instructions: the routine just displays the text as it stands. The routines are listed in Fig. 2(a).

3.2. Specifying individual screen fields

The programmer must specify for each field its 'attributes' (type of data – if any – that may be entered and type of display), the position on the screen at which the field starts (row and column), its length and the data to be displayed in the field. This is done by calling one of the Interface routines, e.g.

CALL CHARHI (IROW, ICOL, LENGTH, DATA)

```

Routine: CLEAR
Purpose: Clears screen; no 'automatic skip'
Routine: CLSKIP
Purpose: Clears screen; unused and protected areas of the screen will be skipped automatically when the user types in data
Routine: CLFILE (FILE, IUNIT, ISKIP, IROW, ICOL)
Purpose: Clears screen and displays introductory messages/menu/whatever taken from a text file
FILE   : name of VSPC file containing text to be displayed (e.g. 'MENU')
IUNIT  : unit/channel to be used to read the text from the file
ISKIP  : 1 if 'automatic skip' required; 2 if not
IROW   : row on which cursor should be placed when screen displayed
ICOL   : column in which cursor should be placed when screen displayed

```

Figure 2(a). Interface routines for clearing the screen and displaying menus, etc.

where 'CHARHI' is the routine that defines a field as 'character data, not protected, highlighted', 'IROW' is the screen row of the start of the field, 'ICOL' is the screen column of the start of the field, 'LENGTH' is the length of the field in characters, 'DATA' is the initial content of the field. The routines currently available are listed at Fig. 2(b).

Arguments

'IROW', 'ICOL' and 'LENGTH' are integer constants, variables or expressions (the Interface routines check that they are credible – the field must be completely contained within the 24 row × 80 column screen). 'LENGTH' is not required for date fields as all dates are displayed in '99 XXX 9999' form.

'DATA' holds the initial content of the field and may be a variable, array, array element, constant, etc. – see Fig. 2(b). If 'DATA' contains characters they can be displayed as they stand; if not, the character string corresponding to the internal numeric value of 'DATA' must be set up for display. (FSM assumes all data to be displayed are characters rather than internal numeric values – it cannot detect and convert the latter so the Interface routines do the work, without altering the content of DATA, relieving the programmer of the chore of arranging for this to be done in the calling program).

When internal floating-point values (REAL, DOUBLE PRECISION) are to be displayed an extra argument is required to give the number of decimal places to be shown.

'NDECPL' is set to

- 0 if the value is to be rounded to the nearest whole number
- 1 if rounding to one decimal place is required etc.
- 1, –2, etc., could be used to specify *rounding* to the nearest ten, hundred, etc.

(FSM does not offer any facilities for displaying N decimal places, so the work is done by the Interface routines when setting up the character string to be displayed.)

3.3 Displaying the 'screen form'

When all the screen fields have been defined, the 'write to screen' Interface routine is used to display them, automatically positioning the cursor at the start of the first unprotected field (if any) unless the programmer has specified a different initial cursor position.

| Routine | Type of data provided by calling program* | Type of data user may enter | Type of display |
|---------|---|-----------------------------|-----------------|
| CHARAC | 'Character' | Any characters | Normal |
| CHARHI | 'Character' | Any characters | Highlighted |
| SECRET | 'Character' | Any characters | Suppressed |
| PROTEC | 'Character' | None: protected | Normal |
| PROTHI | 'Character' | None: protected | Highlighted |
| INTEG4 | INTEGER (*4) | Whole number | Normal |
| INT4HI | INTEGER (*4) | Whole number | Highlighted |
| INTEG2 | INTEGER *2 | Whole number | Normal |
| INT2HI | INTEGER *2 | Whole number | Highlighted |
| SINGLE | REAL (*4) | Any number | Normal |
| SINGHI | REAL (*4) | Any number | Highlighted |
| DOUBLE | DOUBLE PRECISION (REAL *8) | Any number | Normal |
| DOUBHI | DOUBLE PRECISION (REAL *8) | Any number | Highlighted |
| DAY4 | INTEGER containing a date | Date | Normal |
| DAY4HI | INTEGER containing a date | Date | Highlighted |

The above routines are used as follows:

CALL XXXXXX (IROW, ICOL, LENGTH, DATA, NDECPL)

Where XXXXXX is the name of the routine appropriate to the type of field the programmer wishes to define, e.g. CHARAC to define an unprotected, normal intensity character field

IROW, ICOL give the starting position of the field on the screen

LENGTH indicates the size of the field on the screen (not required for dates)

DATA contains the data to be converted (if required) and displayed

NDECPL specifies the number of decimal places required (only for REALs)

* The types of data provided by the calling program are as follows: 'Character' – any variable, array, array element containing characters or a character string. INTEGER – full-word (4 byte) internal INTEGER value**. INTEGER *2 – half-word (2 byte) internal INTEGER value**. REAL – full-word (4 byte) internal REAL value**. REAL*8/DP – double word (8 byte) internal REAL*8/DOUBLE PRECISION value**. INTEGER-date – full word INTEGER giving the date as the number of days since the start of the current calendar (15 October 1582); displayed on the screen in '99 XXX 9999' form (e.g. 13 May 1982).

** Internal numeric values may be provided as constants, expressions, variables or array elements.

Figure 2(b). Interface routines for defining screen fields.

Routine: FREEZE (IROW, ICOL)
 Purpose: Protects the contents of the field starting at IROW, ICOL

Routine: PUTCUR (IROW, ICOL)
 Purpose: Specifies the position the cursor should be put at when the 'write to screen' routine FSWRIT is next used. If the cursor position is already set or the 'pick up' routines reset it after finding a gross error in one of the data fields, the position specified by PUTCUR is ignored.

Routine: FSWRIT (IROW, ICOL)
 Purpose: Write to the screen all the fields that have been specified since the last time (if any) FSWRIT was used, where they join all the fields already being displayed, overwriting the older fields if their positions overlap. The cursor will be positioned at IROW, ICOL unless its position has already been fixed (e.g. by an earlier CALL of PUTCUR) or IROW and ICOL are 0 when the cursor is placed at the start of the first unprotected field (if any).

Routine: FSREAD (KEYHIT)
 Purpose: Waits until the user has made any changes to the data on the screen and pressed a 'control' key (such as 'ENTER' or one of the 'function keys') and then reads the data from the screen into an internal storage area which the program can access later via the 'pick up' routines. Only the contents of fields which have been changed are transmitted; the 'current screen work area' holds the latest version of each screen field. KEYHIT returns a code identifying the 'control' key that was pressed.

Routine: CURSOR (IROW, ICOL)
 Purpose: Returns the position of the Cursor at the time the last FSREAD was executed.

Routine: PRINT
 Purpose: Prints the screen (as it was at the time the last FSREAD was executed) on the printer allocated to the terminal.

Figure 2(c). Interface routines accessing the screen, etc.

3.4. Picking up data entered by the user

The user then enters (or amends) the data on the screen and presses a control key to transmit the changed fields to the computer. The 'read from screen' routine determines which control key was used and various Interface routines are used to pick up the content of the unprotected fields, with conversion from character to internal numeric if appropriate. There is a separate 'pick up' routine for each type of data (character, integer,

etc.) – see Fig. 2(d) – returning flags and messages indicating the results of any character to internal conversions. The 'pick up' routines detect gross errors in the data entered by the user (e.g. '49 GUG – 123' in a date field, '1.2.3–' in a numeric field, etc.) and automatically reset the cursor position so that it appears at the start of the first erroneous field when the screen is next displayed. It is up to the calling program to test the error flags and, if appropriate, display the error message using the 'define character field' Interface routines. The

| Routine | Type of data handled |
|---------|--------------------------------|
| PICKUP | Character |
| PUINT4 | INTEGER (*4) |
| PUINT2 | INTEGER (*2) |
| PUSING | REAL (*4) |
| PUDOUB | DOUBLE PRECISION (REAL *8) |
| PUDAY4 | INTEGER (*4) containing a date |

The above routines are used as follows:

CALL XXX (IROW, ICOL, LENGTH, DATA, ISITOK, MESS)
 Where XXX is the name of the routine appropriate to the type of field programmer wishes to access
 IROW, ICOL give the starting position of the field on the screen
 LENGTH Specifies the size of the field on the screen (not required for dates)
 DATA returns the (converted) data taken from the screen
 ISITOK Indicates whether the data entered by the user contain 'gross' errors (not required for character data)
 +1: no error
 0: blank assumed zero or date not known (99 XXX 9999)
 -1: 'gross' error (e.g. 1.2.3.4 or -1XYZ 9876 as a date)
 MESS 40 character error message (not required for character data)
 blank if ISITOK = +1
 else warns (e.g. 'BLANK ASSUMED ZERO') or describes error (e.g. 'TOO MANY DECIMAL POINTS' or 'INVALID DAY, MONTH AND YEAR')

Figure 2(d). Interface routines 'picking up' data from the screen.

calling program must also check the values are 'in range' and put up error messages if they are not.

Additional Interface routines are used to determine the position of the cursor when the user pressed the control key and to print the content of the current screen if required.

3.5. Advantages of the Interface routines

With the Interface routines, the task which required 16 (rather obscure) lines of code using IBM's OPSYS routine (Fig. 1) takes only 6 lines:

| Code | Notes |
|--|--|
| CALL CLEAR | Clear screen |
| CALL PROTEC (9, 1, 16, 'FULL SCREEN TEST') | Message is protected |
| CALL CHARAC (23, 1, 1, ' ') | Initial content a space |
| CALL FSWRIT (0, 0) | Write to screen, Cursor positioned automatically at start of first unprotected field |
| CALL FSREAD (KEYHIT) | Read amended screen |
| CALL PICKUP (23, 1, 1, LOCAL) | Store data entered by user in program variable 'LOCAL' |

moreover, the process of defining a more complicated 'screen form' with several fields (each with a side-heading), instructions to 'press control key X to obtain a print', error message areas and so forth is reasonably straightforward (and changes to the screen layout much easier) because, given a little familiarity with the Interface routines, it is clear what is going where.

An example of such a screen form is given at Fig. 3 and the code required to produce it at Fig. 4.

The Interface routines also allow 'screen-editing' problems encountered using IBM's OPSYS routine to be overcome very easily. Users of 3270 (or equivalent) terminals can amend data in a screen field by

- (a) Deleting characters (using the 'DELETE' and 'ERASE to End Of Field' keys)

- (b) Inserting characters (using an 'INSERT MODE' key)

- (c) Keying over its initial content

or a combination of the above operations.

For example, if a 'name' field contains

'FRANK DIXON'

the user might wish to change it to

'F DIXON' (by deleting 'RANK')

or

'FRANK MACDIXON' (by inserting 'MAC')

Spaces and 'nulls' (indicating no characters present at all) both appear as blanks on the screen but have *different* internal codes. Deleting characters from a field causes it to be 'padded' with trailing nulls (which can cause problems when the amended information is processed later) and the insertion of characters into a field is only possible if there are nulls at the end (insertions into a field with trailing spaces fail because the spaces cannot be 'pushed' past the end of the field). The Interface routines convert trailing spaces to nulls before displaying the fields

| | |
|------------------|----------------------------------|
| STAFF REF NO | <u>1234</u> |
| SURNAME | <u>dixon</u> |
| INITIALS | <u>f j</u> |
| GRADE | <u>neo</u> |
| ROOM | <u>sl/255</u> |
| EXTENSION | <u>3206</u> |
| DATE OF BIRTH | <u>15</u> <u>may</u> <u>1952</u> |
| DATE JOINED SOCS | <u>11</u> <u>feb</u> <u>1978</u> |

PRESS "ENTER" WHEN ALL CHANGES MADE
OR "SF6" FOR PRINT OF SCREEN

— indicates area in which user can enter data.

Figure 3. Screen produced using F.S.M. interface routines.

```

10 C      FIGURE 4      PRODUCES FIGURE 3  USING THE INTERFACE ROUTINES
20 C      -----
30 C
40 C      DIMENSION  SURNAM(5), ROOM(2),
50 C      -          MESS1(10), MESS2(10), MESS3(10), MESS4(10)
60 C      DATA  IREFNO, SURNAM, INITS, GRADE, ROOM, IEXTNO, IBORN, JOINED
70 C      -      /      0, 5*' ', ' ', ' ', 2*' ',      0,      0,      0 /
80 C
90 C      CLEAR SCREEN, WITH 'AUTO SKIP'
100 C
110 C      CALL CLSKIP
120 C
130 C      SIDE-HEADINGS
140 C
150 C      CALL PROTEC ( 3, 1, 19, 'STAFF REF NO      ' )
160 C      CALL PROTEC ( 5, 1, 19, 'SURNAME      ' )
170 C      CALL PROTEC ( 7, 1, 19, 'INITIALS      ' )
180 C      CALL PROTEC ( 9, 1, 19, 'GRADE      ' )
190 C      CALL PROTEC ( 11, 1, 19, 'ROOM      ' )
200 C      CALL PROTEC ( 13, 1, 19, 'EXTENSION      ' )
210 C      CALL PROTEC ( 15, 1, 19, 'DATE OF BIRTH      ' )
220 C      CALL PROTEC ( 17, 1, 19, 'DATE JOINED SOCS      ' )
230 C      CALL PROTEC ( 21, 1, 35, 'PRESS "ENTER" WHEN ALL CHANGES MADE' )
240 C      CALL PROTEC ( 22, 1, 35, ' OR "SF6" FOR PRINT OF SCREEN ' )
250 C
260 C      DATA FIELDS
270 C
280 C      CALL INT4HI ( 3, 26, 4, IREFNO )
290 C      CALL CHARHI ( 5, 26, 20, SURNAM )
300 C      CALL CHARHI ( 7, 26, 4, INITS )
310 C      CALL CHARHI ( 9, 26, 4, GRADE )
320 C      CALL CHARHI ( 11, 26, 8, ROOM )
330 C      CALL INT4HI ( 13, 26, 4, IEXTNO )
340 C      CALL DAY4HI ( 15, 26, IBORN )
350 C      CALL DAY4HI ( 17, 26, JOINED )
360 C
370 C      DISPLAY FIELDS ON SCREEN
380 C
390 C      50 CALL FSWRIT ( 0, 0 )
400 C
410 C      READ/COPY ALL FIELDS FROM SCREEN INTO 'CURRENT SCREEN WORK
420 C      AREA' AFTER USER PRESSES 'ENTER' / 'SF6' / OTHER KEY
430 C
440 C      CALL FSREAD ( KEYHIT )
450 C
460 C      PRODUCE PRINT OF CURRENT SCREEN IF 'SF6' PRESSED
470 C
480 C      IF ( KEYHIT .EQ. 6 ) CALL PRINT
490 C
500 C      IF SOME OTHER KEY WAS PRESSED ( EG 'CLEAR' ) MAY HAVE PROBLEMS
510 C
520 C      IF ( KEYHIT .NE. 0 .AND. KEYHIT .NE. 6 )
530 C      * * * TAKE APPROPRIATE ACTION * * *
540 C
550 C      PICK UP CONTENTS OF FIELDS USER MAY HAVE CHANGED
560 C
570 C      CALL PUINT4 ( 3, 26, 4, IREFNO, IOK1, MESS1 )
580 C      CALL PICKUP ( 5, 26, 20, SURNAM )
590 C      CALL PICKUP ( 7, 26, 4, INITS )
600 C      CALL PICKUP ( 9, 26, 4, GRADE )
610 C      CALL PICKUP ( 11, 26, 8, ROOM )
620 C      CALL PUINT4 ( 13, 26, 4, IEXTNO, IOK2, MESS2 )
630 C      CALL PUDAY4 ( 15, 26, IBORN, IOK3, MESS3 )
640 C      CALL PUDAY4 ( 17, 26, JOINED, IOK4, MESS4 )
650 C
660 C      FOR CERTAIN FIELDS, IF 'OK' FLAGS INDICATE 'GROSS' ERROR
670 C      PUT UP ERROR MESSAGE VIA 'PROTEC' / 'PROTHI'
680 C
690 C      DO 'RANGE CHECKS' IF APPROPRIATE & PUT UP ANY ERROR MESSAGES
700 C      VIA 'PROTEC' / 'PROTHI'
710 C
720 C      GO BACK IF ANY ERRORS FOUND - USER MUST CORRECT DATA
730 C
740 C      ( APPROPRIATE CODE OMITTED )
750 C
760 C      STOP
770 C      END

```

ASU091 RETURN CODE = 00.

Figure 4. Program using the interface routines to produce screen shown in figure 3.

(and trailing nulls to spaces when picking up the amended data) thus allowing the 'DELETE', 'INSERT' and 'ERASE EOF' keys to be used without problems.

The Interface routines also 'left align' the data (removing leading spaces) to simplify subsequent processing (e.g. checking an abbreviation entered by the user against a list of valid abbreviations).

3.6. Other points

The Interface routines use IBM's OPSYS routine – essentially each CALL of a 'define screen field' routine stores details of the field being defined in an area within the Interface routines that is used as the 'control variable' when the next CALL of the 'write to screen' routine CALLs IBM's OPSYS routine. The other Interface routines use OPSYS more directly – the parameters supplied by the calling program are used in an immediate CALL of OPSYS. The Interface routines therefore require more computer time and storage than 'direct' use of OPSYS but this must be set against the savings in programmer time that they provide.

As the use of the Interface routines has increased they have evolved to meet new needs and, no doubt, will continue to do so. The routines are not perfect – for example, it can be argued that the 'character' routines should *not* automatically 'left align' the data entered by the user and the calling sequences might be improved – and so minor enhancements (or corrections!) are made from time to time. Other weaknesses are unavoidable – for example, the standard FORTRAN limit of 6 characters accounts for some of the more cryptic routine names and prevented the adoption of a convention that all 'service' routines have an identifying prefix as that would have resulted in even less comprehensible names (e.g. 'FSMI4H' instead of 'INT4HI' and so on): if longer names were permitted then they could have been given more meaningful names (e.g. FSMCLEAR, FSM-CLEARSKIP, etc.).

4. THE 'SCREEN FORM' GENERATING ROUTINES

Although the FSM Interface routines were a considerable improvement on the facilities provided by IBM (and sufficiently easy to use that several systems benefited from them) quite a lot of repetitive coding was required to set up each 'screen form'. Clearly, an application requiring several 'screen forms' would need several blocks of code using the Interface routines and although the effort involved could be reduced by copying and amending chunks of code such a solution would not be totally satisfactory.

4.1. Specification of individual fields

The 'Screen form' generating routines were therefore developed to allow a program to set up several 'screen forms' from specifications provided in a 'card-image' lookup table file. The programmer gives for each field to be displayed on the screen:

- (a) The description (or side-heading) – normally displayed beside the field
- (b) the number of the screen form which the field is to form part of

- (c) the type of data to be displayed
 - at present the main options are
 - (i) text
 - (ii) integer (whole number)
 - (iii) floating-point number (with N decimal places)
 - (iv) date (displayed in '99 XXX 9999' form – e.g. 13 MAY 1982 – and held internally as the number of days since the start of the current calendar)
 - (v) 'coded' (the user enters an abbreviation or code-number to identify, say, local government areas which may be coded thus
EDIN Edinburgh
GLAS Glasgow
BADE Badenoch & Strathspey
and so forth. For each entry in the code-list, up to 8 characters are allowed for the abbreviation and up to 24 characters for a more detailed description: the description corresponding to a code being displayed when the user has entered the abbreviation. All abbreviations and descriptions are listed if 'HELP' is invoked. The *full* abbreviation need not be entered – abbreviations of abbreviations are allowed provided the entry is unambiguous. The code-lists would normally be held as a 'card-image' file read by the program into its local storage).
 - (vi) cash (££££.pp – the number of digits before the point can vary)
 - (vii) none – only the 'description' is to be displayed (this allows the programmer to specify extra headings, etc., which may not be associated with any particular field)
 - (d) the position of field on the screen (row *x*, column *y*) and its length (*z* characters)
 - (e) the location of the data within the calling program (by pointers to the elements of the array in which all the data for display on a particular screen are held)
 - (f) range/validity checks
 - the details depend upon the type of data.
 - (i) numbers – the user specifies the minimum and maximum values,
 - (ii) dates – the user specifies the earliest and latest dates as either 'fixed' dates – e.g. 15051952 for 15 May 1952 or *N* days relative to the current date (e.g. – 14 if 2 weeks ago, +7 if a week hence),
 - (iii) 'coded' fields – the user specifies the code-list for the field by pointers to the first and last entries for the field in an array of valid abbreviations and descriptions.
- As the look-up table is read into the program's local storage the check values can be altered by the program during a run (if required) given appropriate coding: for example, a value entered on one screen form may be used as the maximum value for a field on another screen form.
- (g) the position and length of the description of the field and the error-message area for the field,
 - (h) the number of decimal places to be displayed (if appropriate),
 - (i) whether 'not known' is acceptable – a 'zero' value may be valid even if failing the range checks,
 - (j) whether or not the data in the field can be amended. Generally the content of every *data* field can be changed by the user (descriptions and error messages)

cannot); in certain circumstances, however, particular fields may only be entered the *first* time a screen is displayed or may be displayed on one screen *for information only* (any alterations being made via a *different* screen); for example, in a 'personnel' system an individual's name might be entered via an 'identifying details' screen and be displayed *for information only* on other screens.

There are various 'default' specifications so the programmer need not enter *all* of the above for *every* field.

4.2. Limitations of the screen form generating routines

The form generating routines do not provide the *full* flexibility of the Interface routines – they just cope with the bulk of the normal requirements for defining 'screen forms'. In particular

- (a) the programmer cannot specify 'highlighting' – the routines automatically display at 'high intensity' data areas, any error messages and certain 'standard' instructions to the user and display everything else at normal intensity. At present they do not cater for 'display suppressed' (e.g. for passwords).
- (b) The programmer has no control over the form of the error messages. These are, however, reasonably self-explanatory – e.g. 'MUST BE IN RANGE (EARLIEST) TO (LATEST)' where '(EARLIEST)' and '(LATEST)' are dates derived from the programmer's specifications and displayed in 99 XXX 9999 form.
- (c) The only control keys available are those used to:

| | | |
|----------------------------|-----------|-----------|
| record changes made to the | | |
| data on the screen form | | ('ENTER') |
| record changes and print | | |
| current screen |) | |
| undo most recent changes |)function | |
| page forward |)keys | |
| page backwards |) | |

The last two can be used only if the programmer specifies (via an argument) that 'paging' is allowed. If any other 'programmable function key' is pressed a 'PLEASE PRESS CORRECT KEY' message is sent to the user.
- (d) The cursor positioning is totally under the control of the 'screen form' routines: they put it at the start of the first field containing invalid data (e.g. an out-of-range value) or, if no errors are found, at the start of the first 'unprotected' field. The programmer *cannot* specify the initial cursor position nor do the routines indicate where it was when the user pressed a control key.

4.3. Advantages of the screen form generating routines

Given a suitable look-up table (which can be laid out in 'any' form the programmer wishes as the format it is read in is specified in the calling program) setting up a 'screen form' reduces to a few statements to read in the table, etc., and a *single line* CALL of the form generating routine (with appropriate arguments – the number of the screen form to be displayed and so forth) compared to *at least* one line *per field* with the Interface routines. The savings are greatest for non-text fields as, using the Interface routines, quite a lot of coding is required to check the

user's entry is 'in range' and, if not, to put up an appropriate error message and position the cursor at that field. Fig. 5(a) and (b) show 'screen forms' generated by the program given in Fig. 6(a), the look-up table from Fig. 6(b) and the code-list at Fig. 6(c); Fig. 7 explains the use of the routines.

Setting up the look-up table to define the 'screen forms' is a fairly straight forward task given some familiarity with the various options available. It could be simplified by using a program to obtain (via 'screen forms' of course) the details of each field or heading to be displayed – no doubt this will follow once time permits.

The power of the 'screen form' generation routines is such that one can sit down and in just a few hours put together a quite complicated series of screens. Essentially all the work is done by the routines and the look-up table and the only programming required is a relatively short 'main' program to read in the look-up table, specify the order in which screens are displayed, access the system's files and so forth. In practice, developing a production system takes far longer because extra coding is required to handle special cases, perform cross-checks and so forth. However, a quickly put together 'demonstration system' gives the potential user an idea of what the final version would be like and having such a prototype often helps identify possible problems and areas where extra facilities are required. Changes to the layout of a screen form are quite easily made by amending the relevant look-up table entries.

4.4. Other points

The 'screen form' generating routines are written in VSPC FORTRAN and use the Interface routines – essentially they contain a series of CALLs of the Interface routines for each of the permitted types of data plus code to check the user's entries, handle cursor positioning, etc. They 'interpret' the specifications provided in the look-up tables and thus require more computer time and memory than 'teletype' dialogue or code using IBM's OPSYS routine directly. For small applications this does not matter greatly – only a few seconds of computer time per day are 'wasted' by 'inefficient' code – but the implications of the routines' heavier CPU usage would have to be assessed carefully before they were used for large applications. In theory the routines could be speeded up (e.g. by replacing certain parts with, say, Assembler) for use in larger applications, but as yet this has not been tried.

5. APPLICATIONS AND EVOLUTION

5.1. Applications

The routines described in this paper have been used for two types of application:

(a) 'Record-keeping' systems

An example of these is a Periodicals Circulation system which holds details of periodicals taken by the Scottish Office Library, their suppliers and recipients (individuals in the Scottish Office to whom the periodicals are circulated). A series of 'screen forms' are used to enter and amend periodical details (e.g. title, frequency of publication, etc.), supplier details (e.g. name and address, bank account(s), etc.) and

(a) *Error messages produced automatically*

| | | |
|---------------|--------------------|----------------------------------|
| STAFF REF NO | <u>1.23</u> | NOT A WHOLE NUMBER |
| SURNAME | <u>dixon</u> | |
| INITIALS | <u>f j</u> | |
| GRADE | <u>xxx</u> | (OR HELP) INVALID ABBREVIATION |
| ROOM | <u>a1/235</u> | |
| EXTENSION | <u>-123</u> | MUST BE 3000 TO 3999 |
| DATE OF BIRTH | <u>15 may 1983</u> | MUST BE 5 APR 1918 TO 8 APR 1967 |
| DATE JOINED | <u>11 feb 1983</u> | |

MAKE CHANGES THEN PRESS "ENTER"
 "SF6" IF PRINT REQUIRED
 "SF10" TO CANCEL RECENT CHANGES * * * PLEASE CORRECT ERRORS

(b) *Credible data entered*

| | |
|---------------|--|
| STAFF REF NO | 1234 |
| SURNAME | dixon |
| INITIALS | f j |
| GRADE | neo (OR HELP) HIGHER EXECUTIVE OFFICER |
| ROOM | a1/235 |
| EXTENSION | 3206 |
| DATE OF BIRTH | 15 may 1952 |
| DATE JOINED | 11 feb 1978 |

MAKE CHANGES THEN PRESS "ENTER"
 "SF6" IF PRINT REQUIRED
 "SF10" TO CANCEL RECENT CHANGES * * * NO ERRORS FOUND

Figure 5. Screen forms produced using screen form generating routines.

recipient details (e.g. name, room, building, etc.). The facilities provided by the 'screen form' generation routines for checking the user's entries and offering HELP are particularly important here as there are several long code-lists for subjects covered by the periodical, recipients' locations, etc.

on the screen and all other options (e.g. destination of output, number of copies of results, etc.) re-used.

5.2. Evolution

The Interface routines were originally developed to handle only 'character' data – principally to facilitate minor corrections to, say, names and addresses in 'record-keeping' systems (data for other types of field being entered and amended via a 'teletype' question and answer sequence). This arbitrary split was unsatisfactory and, when time permitted, facilities for converting and displaying internal numeric values and dates were added so that the 'screen forms' could display the various fields in a logical order for the users.

Menus were initially specified by a series of CALLs of PROTEC, which required quite a lot of coding and sometimes made minor changes to their layouts or wording rather awkward. The CLFILE routine was therefore added to simplify this process.

The cursor handling has become more sophisticated. At first the cursor would be placed at the top left of the

(b) *Remote Job Entry 'pre-processor' systems*

These are programs which simplify the submission of batch jobs by asking the user for details of the processing required and modifying the Job Control Language accordingly (e.g. changing the name of the dataset to be accessed). Various checks reduce the possibility of the job failing – for example, if the user enters an invalid dataset name the program refuses to continue until a correction is made. In some cases the options selected from a previous batch run are displayed on the 'screen form' for amendment if required – e.g. if the user wishes to repeat a statistical analysis done for 1979 data on the 1980 file, the part of the dataset name giving the year can be changed

(a) Program to produce screen shown in Fig. 5

```

10 C      PROGRAM TO SET UP FIGURE 5 SCREEN FORM - F J D, APR 83
20 C
30      DIMENSION      IPERSN(99), LFIELD(30,40), LVALID(8,50)
40 C
50 C      READ & CHECK FIELDS LOOK-UP TABLE & SET DEFAULT VALUES
60 C
70      DO 40 N = 1,13
80          READ ( 1, 50 ) JUNK
90      40 CONTINUE
100 C
110      DO 60 N = 1,40
120          READ ( 1, 50 ) ( LFIELD(K,N), K=1,30 )
130      50      FORMAT ( A1, A1, T1, 6A4, 7I4, 2I7, 13I1 )
140      60 CONTINUE
150      CALL FSCHKL ( LFIELD, 30, 40 )
160 C
170 C      READ VALID ABBREVIATIONS LOOK-UP TABLE
180 C
190      READ ( 2, 200 ) LVALID
200      200 FORMAT ( 8A4 )
210 C
220 C      CLEAR FIELDS & DISPLAY SCREEN
230 C
240      CALL CLRREC ( 1, IPERSN, LFIELD, 30, 40 )
250      CALL FSFORM ( 1, IPERSN, LFIELD, 30, 40, LVALID, 8, 2, 1, IRET )
260 C
270      STOP
280      END

```

(b) Look-up table defining screen form

```

10
20 FIGURE 6
30 -----
40
50 A      DESCRIPTION OF FIELD FOR SIDE-HEADING
60 B      BLANK / ZERO IF SPARE LINE IN LOOK-UP TABLE
70 C      SCREEN NUMBER ( SEVERAL SCREENS MAY BE DEFINED IN ONE TABLE )
80 D & E  START OF FIELD AND LENGTH IN WORDS IN ARRAY IN CALLING PROGRAM
90 F      TYPE OF DATA - 1: TEXT, 2: CODED, 3: INTEGER, 4: DATE ETC
100 G & H  START OF FIELD ON SCREEN - ROW AND COLUMN
110 I & J  BASIC RANGE / VALIDITY CHECKS - MINIMUM & MAXIMUM VALUES ETC
120
130 A.      B.      C.      D.      E.      F.      G.      H.      I.      J.
140 STAFF REF NO      1      1      1      1      3      3      26      1000      9999
150 SURNAME           1      1      2      5      1      5      26
160 INITIALS          1      1      7      1      1      7      26
170 GRADE              1      1      8      1      2      9      26      1      10
180 ROOM               1      1      9      2      1      11     26
190 EXTENSION          1      1      11     1      3      13     26      3000      3999
200 DATE OF BIRTH      1      1      12     1      4      15     26     -23750     -5850
210 DATE JOINED        1      1      13     1      4      17     26     -14625      0
220
230
240      (Spare lines omitted)
250

```

(c) Look-up table giving valid abbreviations and descriptions for 'coded' fields

```

10 EO      EXECUTIVE OFFICER
20 HEO     HIGHER EXECUTIVE OFFICER
30 SEO     SENIOR EXECUTIVE OFFICER
40 PRIN    PRINCIPAL
50 AS      ASSISTANT SECRETARY
60 SPARE
70 SPARE
80 SPARE
90 SPARE
100 SPARE

```

(Subsequent entries omitted)

Figure 6. Use of screen form generating routines.

In the program listed in Fig. 6(a)

- IPERSN** holds the data for the current individual according to the specifications given in the look-up table shown in Fig. 6(b)
 i.e. **IPERSN(1)** holds the Staff Ref No. as an **INTEGER**
 and **IPERSN(2)** to (6) holds the Surname (text)
 etc. and **IPERSN(13)** holds the Date Joined ('date integer')
 In a proper system **IPERSN** would be read from and written to a direct-access file using the Ref No to determine the appropriate record.
- LFIELD** holds the screen form generating look-up table read from the card-image file shown in Fig. 6(b) (the first 13 lines of which are skipped). In this case, up to 40 screen fields can be defined, each being specified by one line of the look-up table, and a lot of defaults have been used. The description of field **J** is read into **LFIELD(3,J)...**(8,J); the 'spare' indicator into **LFIELD(9,J)**; the screen number into **LFIELD(10,J)**; the start of the field in the data array (**IPERSN**) into **LFIELD(11,J)**; its length in the data array into **LFIELD(12,J)**; its type into **LFIELD(13,J)** and so on.
- LVALID** holds the code-lists read from the card-image file shown in Fig. 6(c). In this case up to 50 entries are allowed – for entry **J**, **LVALID(1,J)** and (2,J) hold the 8-character abbreviation and **LVALID(3,J)...**(8,J) hold the 24 character description.

The routines used are:

- FSCHKL** Checks the screen form generating look-up table and sets default values where certain entries have been read as zero (blank on the card-image file). The checks include valid type of data code? row and column in range? etc.
- CLRREC** (Screen form number, data array, look-up table, words, entries).
 Initialises all fields in 'data array' which are to be displayed on screen form 'screen form number' to appropriate default values (spaces if 'text' field, '999 XXX 9999' if date field, etc.). The look-up table provides details of which fields belong to that screen form, where they are in the data array, what their types are.
- FSFORM** (Screen form number, data array, look-up table, words, entries, code-list, words per entry in code-list, paging allowed indicator, new record/entry allowed indicator, return code).
 This routine displays the 'screen form', checks the data entered by the user for 'internal' validity, puts up error messages on the screen and so on.
- | | |
|----------------------|--|
| 'screen form number' | identifies the 'screen form' required – e.g. 'screen form 1' might be basic personal details, 'screen form 2' job details, etc. |
| 'data array' | holds the data to be displayed, amended and returned |
| 'look-up table' | contains the screen form generating look-up table for this particular application |
| 'words', 'entries' | specify the size of the look-up table |
| 'code-list' | gives the valid abbreviations, etc., for coded fields |
| 'words per entry...' | indicates the size of each entry in the code-list |
| 'paging allowed?' | set to 1 if the user is to be allowed to use certain function keys to move between screen forms – e.g. if there are up to 10 screens showing job details for an individual the user may wish to move from, say, the 'current job' screen to the 'previous job' screen |
| 'new...allowed?' | set to 1 if the user is to be allowed to enter data in 'key' fields (e.g. when creating a new individual's record a staff reference number must be entered; once the record has been created the permanent key cannot be changed via the normal data amendment sequence) |
| 'return code' | – 1 if the user wishes to page back 0 if the user has finished with this 'screen form' (or set of 'screen forms') + 1 if the user wishes to page forward |

As an example of the coding required to handle 'paging' and 'new records' assume that **JOBS(20,10)** holds details of up to 10 jobs held by an individual: **JOBS(1...20,J)** being the details of the **J**th job. Let **NJOBS** be the number of jobs for the current individual. We have (in pseudo-Fortran):

```

      NEXT = 1                                (pointer to next job entry to be displayed)
100  IF NEXT > NJOBS                          (new job entry to be created for current individual)
      THEN NEWOK = 1
          CALL CLRREC (2,JOBS(1,NEXT),...) (jobs on screen form 2)
          NJOBS = NJOBS + 1
      ELSE NEWOK = 2                          (not creating new job entry)
      END IF
      CALL FSFORM (2,JOBS(1,NEXT),...,1,NEWOK,IRET)
      IF IRET = 0 'EXIT'                      (user finished with 'jobs' screens)
      IF IRET = +1                            ('page forward' requested)
          THEN NEXT = NEXT + 1                (provided this does not go above 10)
          ELSE NEXT = NEXT - 1                (provided this does not go below 1)
      END IF
      GO TO 100                              (to display details of NEXT entry)

```

This would have to be refined a little (if only to prevent the creation of new job entries by accidentally paging too far forward) but the general idea should be clear.

Figure 7. Use of the 'screen form' generating routines.

'screen form' when the form was first displayed; subsequently the routines were modified to put it automatically at the start of the first amendable field (if any) and, later, at the start of the first field (if any)

containing 'gross' errors. The **PUTCUR** and **CURSOR** routines were added to give the programmer more control over the cursor and detect its position for selection from 'menus'.

Various other changes have been made over the years and the routines are now considerably more powerful than was originally envisaged. Fortunately it has been possible to introduce the enhancements without requiring major changes to existing applications: the routines have proved 'upward compatible'.

The 'screen form' generating routines were developed, after several applications using the Interface routines had been set up, to reduce the work involved in generating 'screen forms' with 'standard' facilities (i.e. those required by most existing and proposed systems). The coding was based on an existing application using the Interface routines, generalised to a 'table-driven' series of CALLs of the Interface routines.

6. CONCLUSIONS

These routines are of interest as an example of how the basic facilities offered by the manufacturer's software may be improved and made easier to use; it is not claimed that they represent a 'great leap forward' for Computer Science (no doubt similar routines are in use elsewhere –

indeed some might argue that such routines are, or soon will be, unnecessary as 'easy screen design' facilities are already offered by some database management systems and this seems to be an area in which development is rapid). The time spent developing these (and other) routines had been more than justified by the time saved using them, and the net benefits increase with every new application that uses them. The routines have also allowed facilities to be used that would *not* otherwise have been used because of the difficulty of using the versions supplied by the manufacturer.

The use of such routines should also make programs more 'portable' as the non-standard installation-dependent features are restricted to the 'service' routines rather than appearing throughout each program.

Acknowledgements

Colleagues, particularly Duncan Leuchars and Malcolm McNeil, for assistance and suggestions; SOCS for computing time; IBM for permission to reproduce Fig. 1 and the referee for suggesting some improvements.