UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 30

(January 1985)

Title:

Virtual Video Package

Author: J.M. Murison
J.K. Yarwood
C.N. Duncan

Contact: Advisory Service

Software Support Category: D

Synopsis

The Virtual Video Package (VVP) is designed to enable EMAS programs to handle a variety of video terminals, by providing a terminal-independent interface through which a program can send output to the user's video and receive input from it. The main purpose of providing such a package is to free the programmer as far as possible from having to deal with the idiosyncrasies of the different videos through which EMAS is accessible.

Keywords

Video terminal, Virtual Video Package

Edinburgh Regional Computing Centre
James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ. Telephone 031-667 1081

Introduction

The Virtual Video Package (VVP) is designed to enable EMAS programs to handle a variety of video terminals, by providing a terminal-independent interface through which a program can send output to the user's video and receive input from it. The procedures also enable the program to determine characteristics of the terminal, and to remove from the screen the effects of the most recent input; this is found to be a useful facility.

Availability

The package uses facilities implemented in the Terminal Control Processors (TCPs) for full-screen operation of the main types of VDU. The package cannot operate with terminals connected via the new JNT PADs at the present time; however, it is possible that future versions of the PAD software will incorporate handling of the JNT-sponsored Simple Screen Management Protocol (SSMP), and when this is done VVP will be suitably enhanced. It is also possible that locally-written software for the BBC Micro and other 'intelligent terminals' will support SSMP even with existing PAD software, thus enabling VVP to be made available in advance of upgraded PAD software.

The Virtual Video Package is currently suitable for use with the following terminal types:

Perkin-Elmer 550	("Bantam")
Newbury 7001	
Lear-Siegler ADM-3A	
Visual 200	
Hazeltine Esprit	(Including Esprit II and Esprit III)
IBM 3101	
ICL KDS7362	
VT52	(Including BBC Micro with VT52 emulation)

The appropriate terminal-type must first have been indicated to the Subsystem to establish the relevant information for the session by means of the TERMINALTYPE command, or perhaps preferably by use of the TTYINIT command in conjunction with OPTION FSTARTFILE=file. Further details can be obtained by typing:

Command: HELP TERMINALTYPE
Command: HELP TTYINIT
Command: HELP FSTARTFILE

For some terminal-types it may also be necessary to ensure that switch-settings at the rear of the terminal or option-settings (using keyboard and screen) are suitably selected. Settings required are those which are also necessary for the screen-editor SCREED, described in a booklet from the Advisory Service. Up-to-date information on switch or option-settings should be found by typing:

Command: HELP SCREED

and looking at the subsection "5. Notes on each terminal".

To access VVP, it is necessary to include the library CONLIB.GENERAL in the library search-list, using:

Command: OPTION SEARCHDIR=CONLIB.GENERAL

Suitable IMP external routine specs are available in CONLIB.VVP_VVPSPECS, and bit values and standard character values in CONLIB.VVP_VVPFORMATS; each of these files is suitable for being %INCLUDEd into an IMP program.

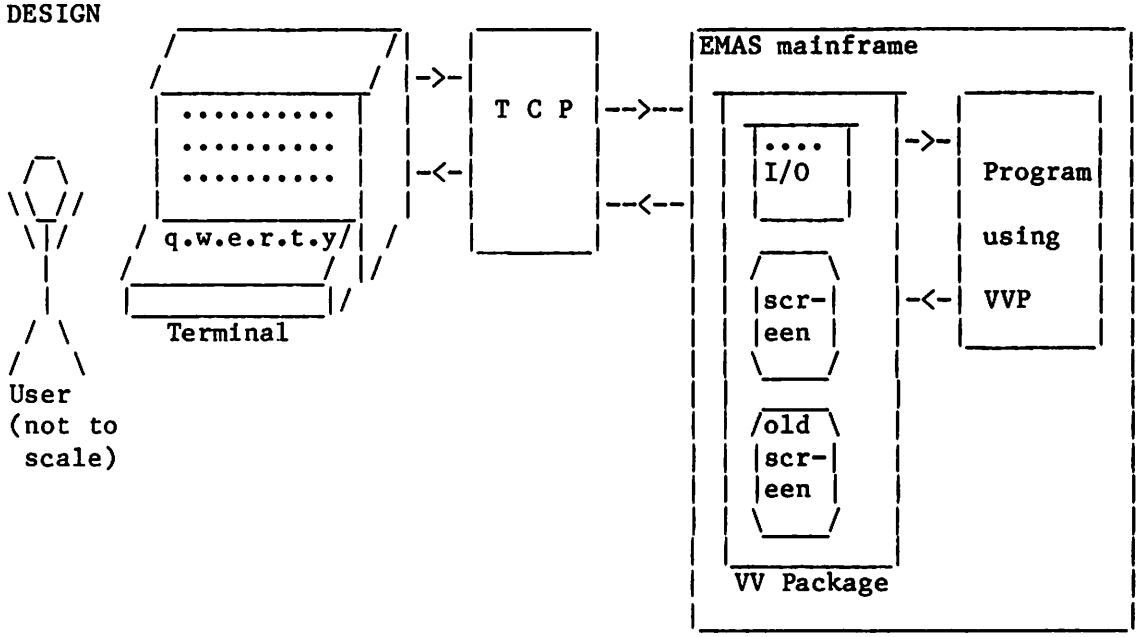
Technical Background

The main purpose of providing a package such as VVP is to free the programmer as far as possible from having to deal with the idiosyncrasies of the different videos through which EMAS is accessible. One difference between existing videos is that functions like "clear rest of line", "cursor down", "clear screen", etc. are effected by different character codes. This type of difference is easily handled by choosing a standard set of control codes for VVP's 'standard video' and mapping them onto the codes required by the actual video in use onto the standard set.

Unfortunately, other differences between video terminals cannot be handled so easily: scrolling characteristics, cursor movement at an edge of the screen, etc. Such things could be standardized for every terminal if the VV Package were able to control every character sent to the user's screen. However, when the user types at his video, what he types is echoed by the local Terminal Control Processor (TCP) to which the terminal is connected. At this stage EMAS is not involved, until the user types a forwarding ('trigger') character, when the codes accumulated by the TCP are sent down the line to EMAS. Normally the only trigger character is Return, but additional modes now provided in the TCPs enables one to specify any set of ASCII control characters (characters with code < 32 decimal) as triggers. In principle it would be possible to arrange for EVERY character typed by the user to be sent to the EMAS mainframe without being echoed by the TCP, so that the VV Package could control the echoing, and indeed every character sent to the user's screen. However, neither the communications network nor EMAS 2900 was designed for use in single-character mode, and the result would be unacceptably inefficient. Instead the programmer is able to specify which control characters typed by the user are to be triggers.

It follows that VVP only receives input retrospectively, and so cannot control the effect on the screen of user input. While VVP cannot stop the user changing the screen in some 'incorrect' way, it does give facilities to the program using it for 'repairing' the screen, as explained below.

DESIGN



VVP is either in INPUT MODE or in OUTPUT MODE. As a dialogue proceeds the modes change. When the program attempts to read data from the interactive terminal it makes use of one of VVP's input routines, described in detail below. Whenever an input routine is used after output has been generated by the program, the mode switches to input. VVP then reads the user's input, up to the trigger character which caused it to be sent from the TCP to the mainframe.

At this point VVP has an up-to-date version of the user's actual screen, plus the characters typed by the user since the last trigger character. VVP also has a record of the screen before this latest input was typed. The program can read the input characters from VVP if desired; if it switches to output mode, by using an output routine, then any unread input is no longer available for reading, but of course its effect has none the less been incorporated in the screen. If, having read the trigger character, the program issues another read request, then VVP stays in input mode.

From the user's point of view, he will not see any difference at this stage from the current arrangements: his input will be echoed by the TCP as at present. However, when he generates a trigger character (usually Return or CTRL+something), he will find that any further input typed by him will not be echoed until the program is ready to read it. One consequence of this is that the system can issue a prompt string whenever input is requested, in the sure knowledge that it will precede the relevant input on the user's screen.

Input Mode

When a switch from output mode to input mode is made, the following actions occur:

1) An implicit call of the routine VV UPDATE SCREEN occurs. This causes the user's screen to be modified by VVP as a result of the output generated by the program since the last call of VV UPDATE SCREEN.
2) VVP stores a copy of the screen thus updated.

Then, as input is received from the user terminal, VVP translates the input control sequence into standard control characters, passes these and the printable characters typed to the program (depending on what input routines are used), and notes how the screen has been changed by this input. However, VVP's stored copy of the screen (see (2) above) is not changed, and the program using VVP can restore the screen, or parts of it, to its state when input mode was entered. It does this by use of one of the 'repair' routines, described below; these simply refer to the appropriate part of the stored copy.

However, when the program causes a switch to output mode, by using an output routine, the stored copy of the screen is lost. Note that VVP takes account of ALL the user's input, whether the calling program chooses to read it or not.

There are two main situations for which the ability to repair all or part of the screen is useful:

1) Where the user types something which is not intended to become part of the screen display (whatever that might be), but merely information or instructions for the program. After the program has detected this information (by use of VVP's input routines) it can then restore the text which the user had to overprint to convey the information.
2) Where the program detects that the user has overtyped or otherwise destroyed text on the screen which was intended to be retained. For example, the user might have overtyped a prompt; in this case the program would probably repair the screen and ignore the input.

The routine VV READ CH enables the program to read the user's input as a stream of bytes, i.e. in the order in which it was typed. However, there are other routines which work in terms of the state of the SCREEN after (or before) the latest input. When one of the latter routines is used, effectively all the characters input up to the trigger character are read first, and so VV READ CH cannot subsequently be used to read them.

Output Mode

In output mode, the program can modify the screen by use of the various output routines provided. The screen changes thus caused are not actually sent to the user's terminal until the output routine VV UPDATE SCREEN is called, either explicitly or implicitly when a switch is made to input mode (through use of one of VVP's input routines).

VVP Procedures: summary

The VVP interface is procedural. There are three sets of procedures, described in detail in the following three sections and summarized below:

a) Initialisation procedures

VV INIT — A routine: must be called before any other VVP procedure.

VV DEFINE TRIGGERS
A routine: used to specify which ASCII control characters are to be triggers. A shorthand way of selecting commonly required sets of triggers is provided.

VV TERMINAL DETAILS
A routine: returns dimensions of the user's screen and the "Int:" character in use (ESC is often pre-empted, in the terminal design, by being used for cursor movement, control sequences, etc.)

VV CLOSE A routine: used to terminate VVP operation, and in particular to re-establish normal TCP mode operation.

VV RESTORE MODES
A command: used to restore standard TCP modes in the event that a program using VVP returns (e.g. through error) to command level without having called VV CLOSE.

b) Input procedures

VV READ CH A routine: returns the next input character, either printable or a VVP standard control character, or any other character which was generated at the keyboard but which had no effect on the screen. Various external integer variables are also set to enable the program to determine details of the effect of the character on the screen.

VV GIVE OLD LINE
A routine: returns the contents of a specified screen line, as it was at the start of the current input mode.

VV GIVE NEW LINE
A routine: returns the contents of a specified screen line, as it is after the current input has modified it.

VV REPAIR LINE
A routine: causes the specified line on the screen to be replaced by the corresponding line in the stored copy of the screen (the copy was made when input mode was entered).

VV REPAIR SCREEN
A routine: causes the whole screen to be replaced by the stored copy of the screen.

VV RSTRG A routine: provides a simple way of reading input confined to a single line, taking account of backspaces and deletions, etc. Various conditions attached to its use are detailed below.

c) Output procedures

VV PRINT CH A routine: prints a character on the screen.

VV PRINT STRING
A routine: prints a string on the screen, by repeated use of VV PRINT CH.

VV NEWLINE, VV NEWLINES, VV SPACE, VV SPACES
Routines affecting the screen.

VV CLEAR SCREEN
A routine: clears the screen.

VV GO TO A routine: moves the cursor on the screen.

VV UPDATE SCREEN
A routine: causes output accumulated thus far to be sent to the user's terminal. It is called implicitly at the end of each output mode, to 'flush out' all generated output.

VV WRITE A routine: equivalent to the IMP implicit routine WRITE (but can write %long %integers).

VV PRINT A routine: equivalent to the IMP implicit routine PRINT.

VV PRINTFL A routine: equivalent to the IMP implicit routine PRINTF.

d) Other high-level i/o procedures

The following routines are offered to assist in programming menu and page-edit applications. The VV EDIT PAGE and VV MENU routines are designed to work even if full-screen operations are not available (whether because of unsatisfactory terminal type, or because communications connection is via a PAD).

VV EDIT PAGE
A routine: allows a screen page of data to be displayed and edited, or an empty screen area to be presented for completion.

VV MENU A routine: allows a set of options to be presented, for a selection to be made by the terminal user.

Initialisation Procedures

%externalroutine VV INIT (%integername FLAG)

This routine must be called before any other call on VVP is made, except that the high-level procedures VV MENU and VV EDIT PAGE are able to perform this initialisation if it has not already been done. It is assumed that the users terminal type has been correctly specified (by use of the EMAS command TERMINALTYPE). Flag is set to zero if the terminal type (as set by the EMAS command TERMINALTYPE) is acceptable for use with VVP. Hard copy terminals and certain older video terminals cannot be used with VVP. The following terminals are currently acceptable:

Perkin-Elmer, Visual 200, Hazeltine Esprit and Esprit II & III
Newbury, Volker-Craig 404, ICL KDS7362, VT52

%externalintegerfn VV DEFINE TRIGGERS(%integer SET, TRIGGER BITS, ECHO BITS)

When parameter SET is positive, this function defines the control characters which are to be TCP trigger characters, and which of these

trigger characters are to be echoed back to the video terminal by the TCP. 'Tell' and Operator messages etc. are inhibited, the screen is cleared and the cursor is placed at (0,0) - the top left-hand corner. (Note that screen coordinates are given as (column, line) with column and line both starting at 0.)

A short explanation follows of the current TCP arrangements enabling transmission of screen control characters. A mainframe program may send a 32-bit 'mask' to the TCP. The mask determines which of the 32 ASCII control characters (decimal values 0-31) are to cause input to be dispatched to the mainframe. When the user generates one of the specified values, all preceding input up to and including the specified character (the 'trigger character') is dispatched.

The parameters TRIGGER BITS and ECHO BITS are not currently used. They will specify 32-bit masks which will determine which of the ASCII control characters are to be triggers and which are to be echoed back to the terminal by the TCP.

For the time being, only SET is used; the other two parameters should be set to zero. Further, at present trigger characters, and ONLY trigger characters, are not echoed.

The following values for SET are used to specify the required set of trigger characters:

SET	Meaning
1	makes every control character a trigger character (and hence a non-echoing character).
2	makes every control character a trigger except those involved in cursor movement.
3	makes every control character a trigger character except those involved in left-right cursor movement (thus allowing line-editing in the input line, without triggering input to the mainframe, by using left-arrow, backspace and right-arrow keys).

Thus 1 should be used if the cursor is not to be moved as a result of the user typing control characters at the keyboard; 2 should be used if the user is to be permitted to move the cursor using the keyboard's control keys; 3 should be used if the user is to be permitted to move the cursor on the current screen line only.

In order to terminate VVP operation and re-establish the normal TCP mode, VV DEFINE TRIGGERS should be called with one of the following values of SET:

SET	Meaning
0	Clear the screen and revert to normal TCP mode.
-1	as 0, but do not clear the screen. Cursor is left in its current position.
-2	as 0, but do not clear screen. Cursor placed in bottom LH corner of screen.

%external %routine VV CLOSE

This routine, or VV DEFINE TRIGGERS, q.v., should be called before the end of any program in which VVP has been used.

%external %routine VV RESTORE MODES(%string(255) s) {EMAS command}

In the event that the user has left a program using VVP with an "Int:" or if the program failed, so that VV CLOSE was not called, then VV RESTORE MODES can be called from command level to clear the screen and reset the modes to normal interactive-terminal mode.

%externalroutine VV TERMINAL DETAILS(%integername SCOLS, SLINES, INT CHAR)

This routine returns the number of columns and lines which the user's video has into the external integers SCOLS and SLINES. Note that in the descriptions below SCOLS1 is (SCOLS-1), i.e. the number of the rightmost column on the screen, and SLINES1 is (SLINES-1), the number of the bottom line on the screen. SCOLS, SLINES are set to zero if the terminal type cannot be determined.

The key which invokes the TCP "Int:" sequence - normally ESC - is commonly changed by VVP, because many terminals use the ESC character for the screen control functions. The control character which VVP has selected to replace the normal ESC key function is returned in the INT CHAR parameter. After an "Int:" escape from the calling program to Subsystem command level, the SETMODE command should normally be given, to re-establish the default, or required, mode settings.

Input Procedures

These procedures are used to read characters from the user's terminal, to determine the current state of the screen and to reset the screen if desired. Note that the program does NOT have to read each character explicitly: if output mode is selected, by use of an output routine, any input unread by the program is none the less taken into account by VVP, to bring it into line with the user's screen. When the program next selects input mode, however, previously unread input will NOT be available to it. On the other hand, if the program calls any input routine other than VV READ CH or VV GIVE OLD LINE, then all input characters up to the next trigger are first applied to VVP's copy of the screen, and cannot subsequently be read by use of VV READ CH.

%externalroutine VV READ CH(%integername CH)

CH is set with the next ASCII printable character, or with one of the standard control characters listed below.

In addition, the following external integer variables are set:

```
VV X
VV Y
VV BITS
VV CHARX
VV CHARY
```

The VV X, VV Y integers always give the cursor position AFTER the operation determined by the character returned by VV READ CH. The character may have been a normal printing character, or it may have been a VVP control character. Such a control character can be a translation of one or more characters received from the user's terminal, or it could have been generated as a result of some event occurring on the user's terminal. Exceptionally, (VV X, VV Y) might not lie within the screen area, reflecting a characteristic of the particular terminal; for example, one whose cursor moves to column SCOLS when a character is typed in column SCOLS1. Programs should not make use of any knowledge of this special situation, but rather should ignore (VV X, VV Y) when the 'strange cursor' bit is set in VV BITS (see below).

VV CHARX, VV CHARY give the position of the character on the screen when the 'printable' bit is set. In these cases, or when the character was a trigger character, bits are set in the VV BITS word as described below. The values describing the various situations are subject to change, and therefore should be taken from the file CONLIB.VVP_VVPFORMATS, using %include if IMP80 is being used. This file declares %constant %integers named as given in the "Name" column below. The value of each is a power of 2, so that, for example, to test whether PRINTABLE is set, the program could use the test:

```
%if VV BITS & PRINTABLE # 0 %then .....
```

NAME	MEANING
trigger	The character was a trigger character.
not echoed	The character was not echoed.
printable	The character was printed (ASCII values 32-126)
canon	The character is a standard ('canonical') control character from the following set: 9 Horizontal tab operation. The actual cursor movement is given only by the final (VV X,VV Y) 13 Carriage return 32 Cursor right 33 Cursor left (a backspace operation) 34 Cursor up 35 Cursor down 36 A "Home" cursor operation. Like the horizontal tab, the effect is defined only by the final (VV X,VV Y). 37 Clear screen 38 Clear rest of line %constant %integers for each of these values are included in the file mentioned.

no effect	The character had no effect on the screen, i.e. was neither printed, nor is it a standard control character. Situations such as {Cursor right at column SCOLS1 and the cursor remained at column SCOLS1} do not cause this bit to be set, but rather the 'strange cursor' bit, described below.
strange cursor	The cursor moved in a manner which was not deducible from the standard control character value, or moved in a singular manner following a printable character or backspace. The events associated with this bit are as follows: * Cursor movement to another line as side effect of typing beyond column SCOLS1 * Cursor movement to start of same line as a result of a cursor-right operation from column SCOLS1 * Cursor movement to end of same line as a result of a cursor-left operation from column 0 * Cursor movement to bottom of same column as a result of a cursor-up operation from line 0 * Cursor movement to top of same column as a result of a cursor-down operation from line SLINES1 * Cursor movement to another line as a result of a backspace operation from column 0 * Cursor movement to another line as a side effect of a horizontal tab operation * A "Home" operation which did not leave the cursor at (0,0) * No cursor movement for standard control characters 32-35 or for a character typed in column SCOLS1.
screen scrolled	The screen scrolled as a side effect of the operation.
screen rev scrolled	The screen reverse-scrolled as a side effect of the operation.

In addition to being able to get a stream of input characters by using VV READ CH, the calling program has access - by use of other input procedures - to the screen as it was before the current input changed it, AND to the screen after the current input changed it. Note that if any of these routines, other than VV GIVE OLD LINE, is used, then VV READ CH cannot be used subsequently to read the current set of input characters.

%externalroutine VV GIVE OLD LINE(%integer LINENO, %string(*)%name TEXT)

This routine returns line LINENO, as it was before the current set of input characters was typed. Trailing spaces are not removed, but if LENGTH(TEXT) is less than SCOLS1, that means that the rest of the line is blank.

%externalroutine VV GIVE NEW LINE(%integer LINENO, %string(*)%name TEXT)

This routine returns line LINENO as it is after the current set of input characters was typed. Trailing spaces are not removed, but if LENGTH(TEXT) is less than SCOLS1, that means that the rest of the line is blank. Once this routine has been used, VV READ CH cannot be used to read the current set of input characters.

%externalroutine VV REPAIR LINE(%integer LINENO)

This causes the new screen line to be abandoned, so that when the user's screen is next updated the displayed line will revert to its appearance before the user's latest input was typed.

%externalroutine VV REPAIR SCREEN

This causes the screen to revert to its state before it was changed by the most recent typed input. (It is equivalent to calls of VV REPAIR LINE for each line affected by input typed since the last updating of the screen.)

%externalroutine VV RSTRG(%string(*)%name S)

The routine prints the initial value of S, then delivers in S the string which the user has input, between the cursor position at the CALL of this routine and the cursor position when the RETURN key is pressed. A null string is returned in certain cases described below.

This routine is intended primarily for use following a VV DEFINE TRIGGERS(2 or 3,..) initialisation; it first prints the string S at the current cursor position, then places the cursor back at that same position and finally left-right cursor movement is enabled at the keyboard. This routine calls VV READ CH repeatedly, taking account of cursor movement and presenting a text string to the calling program. The line may be edited at the keyboard using backspace, left-arrow and right-arrow keys and overtyping previously typed characters. S should normally be set null before the routine is called, but if the program requires to present some text to the user for editing and re-input, then S should be set to the required initial text.

The input line to this routine is considered 'cancelled' in the following circumstances: (1) The input message is terminated by other than Return key. Thus, e.g., Control+X (CANcel) effectively cancels the input; (2) Characters are typed to the left of the position of the cursor at the start of input (for this message); (3) characters are typed on a line other than that on which the cursor lay before the routine was called.

The DEL (RUBOUT) character is ignored by this routine.

Note that since this routine simply calls VV READ CH repeatedly in order to reconstruct the input, calls of VV RSTRG and VV READ CH should not be intermixed.

Output Procedures

The output primitives are given below; the procedures described do not cause data to be sent to the terminal directly, but cause VVP to store the character codes implied. The user's screen will be updated when the output routine VV UPDATE SCREEN is called, or when any input routine is called.

%externalroutine VV GOTO (%integer X, Y)

The cursor is moved to position (X,Y). If the specified (X,Y) do not give a point within the screen, the cursor moves as far as it can towards that point.

%externalroutine VV PRINT CH(%integer CHAR)

The value of char must be in the range 32 to 126 and the cursor must lie within the screen (and not in the bottom right-hand corner), or it may be one of the standard control characters given below. The restriction in parentheses is to prevent scrolling from occurring on those terminals whose cursor moves to the next line when a character is typed at (SCOLS1, SLINES1). The restriction may be lifted for those terminals which do not show this trait. If the conditions described are not satisfied, the procedure call is null.

The following standard control characters have so far been defined.

LF	clear rest of line, move output pointer to start of next line. Scrolling does not take place if current line is SLINES1.
CLR SCREEN	Clear screen.
CLR ROL	Clear rest of line.

%externalroutine VV PRINT STRING(%string(255) S)

Calls VV PRINT CH, passing the characters of S.

The following are extra procedures to ease programming, but in fact cause effective VV PRINT STRING and/or VV GOTO procedures calls.

%externalroutine VV NEWLINE

Equivalent to calls of VV PRINT CH(CLR ROL) followed by VV GOTO(0, y+1).

%externalroutine VV NEWLINES(%integer I)
%externalroutine VV SPACE
%externalroutine VV SPACES(%integer I)

%externalroutine VV CLEAR SCREEN

%externalroutine VV UPDATE SCREEN

This routine causes all output generated by calls of any of the foregoing output procedures to be sent to the user's screen. It also has the effect of uninhibiting the echoing of input from the user. This can have unfortunate effects if the user has been typing ahead, and thus it is often advisable not to call VV UPDATE SCREEN explicitly, but to let VVP flush the output automatically, when input mode is next selected.

%externalroutine VV WRITE(%integer I, PL)

This routine is analogous to the IMP WRITE intrinsic output routine, and WRITEs the value I commencing at the current cursor position.

%externalroutine VV PRINT(%longreal X, %integer N, M)

This routine is analogous to the IMP PRINT intrinsic output routine; the output is placed at the current cursor position.

%externalroutines VV PRINTFL(%longreal X, %integer N)

This routine is analogous to the IMP PRINTFL intrinsic output routine; the output is placed at the current cursor position.

---

d) Other high-level i/o procedures

The following routines are offered to assist in programming menu and page-edit applications. The VV EDIT PAGE and VV MENU routines are designed to work even if full-screen operations are not available (whether because of unsatisfactory terminal type, or because communications connection is via a PAD).

%external %routine VV EDIT PAGE(%string(*)%array %name S, %integername FLAG)

This routine is used to edit, information in a string array. When the array has 20 elements each 80 characters in length it will fill the screen of most terminals. Cursor control keys (arrows) then allow the user to move freely over the information displayed, changing it at will.

The routine clears the screen, then displays in the first three lines the basic edit instructions in the form

```
Use cursor control keys to move and RETURN to make changes.    ESC is now CTRL-\
#A to abort editing, #U to finish editing, #R to restart editing
```

The string array, S, should be filled with text, one line per array entry, before the routine is called. (The bounds of the array are not important, but each array entry must be initialised, if only to a null string.) The contents of the array entries are placed on successive screen lines following the heading, and the rest of the screen (after the line holding the last non-null string in the array) is made blank.

The text on any one line is considered to be informative or descriptive text to the left, and the text to be edited to the right. The routine then permits the text to the right to be edited while protecting the text to the left. The value, N, of FLAG on entry gives the number of protected columns, i.e. the columns up to but not including N are assumed to contain informative text which is not to be edited. *If N=0 the entire screen is available for editing.

The columns beyond the first N may be left blank (length of array element less than or equal to N), or may contain text which the terminal user may edit. On return from this routine, the array S will contain the values of the 'unprotected' strings of the respective screen lines, corresponding to the initial array entries. Note that on entry to the routine S contained both the informative text and the text to be edited but on exit S contains only the text which was edited, the informative text has been removed. Trailing spaces will have been removed from the strings in S.

On exit, FLAG will be set as follows:

zero if any row of the array has been altered

1 if the screen is unchanged or if "#A" (abort) has been typed.

The 'protected' area is implemented by having the screen rewritten ab initio if the user types in that area, or if he presses cursor movement keys which cause the screen to scroll. The terminal user is expected to

move within the unprotected area, adding or altering text as he requires.

Note that the key which is to be pressed to get an Int: prompt is no no longer ESC. It is CTRL- with some other key which varies from terminal type to terminal type. The appropriate key is always shown when an array is being edited. However, #A should normally be used to exit from the routine in emergency.

%external %routine VV MENU(%string(*)%array %name S, %integer NHEADING, NLINES, %integer %name SELECTION)

This routine presents a menu on the screen and returns the line number which was selected by the user. It routine prints out NHEADING+NLINES strings from the array S. The first NHEADING lines of S (which should be declared with lower bound 1 and upper bound at least NHEADING plus NLINES) are regarded as non-menu or informative data.

A horizontal line is drawn to separate the NHEADING lines from the NLINES text.

The next NLINES are items of 'menu' data: these items are printed on successive lines of the screen, each preceded by a letter, A, B, C..., and the remaining screenlines are cleared.

The terminal user is invited to type one of these letters to select a menu item.

The value of SELECTION is set on return to be the number of the menu item selected (between one and NLINES), or minus one if an unsuitable terminal-type value is currently set.

It is most desirable to include in each menu an item (for example) "QUIT", which the user may select to exit from the program to command level.

Notes on VV EDIT PAGE and VV MENU

The VV EDIT PAGE and VV MENU routines suitably initialise the Virtual Video Package (VVP) if this has not already been done, by calls on VV INIT, VV DEFINE TRIGGERS. On exit from the routines, and assuming that the terminal type was suitable, VVP is left 'activated', that is, with a suitable communications-mode established and with a correct screen image 'remembered'. If repeated Page Edits or Menu selections are to be performed, VV EDIT PAGE and VV MENU can be repeatedly called with minimum rewriting of the screen contents. If, however, the calling program requires to issue ordinary terminal i/o requests after calling one of these 'VV' routines, and in any case before return to command level, the program should call VV CLOSE to terminate (if only temporarily) the VVP session.

A program demonstrating the use of VV EDIT PAGE and VV MENU is given in Appendix 2.

Appendix 1: Demonstration program No. 1

To call this demonstration program, first use the command OPTION SEARCHDIR=CONLIB.GENERAL. The program is called by command VVDEMO.

The source of the demonstration program is file CONLIB.VVP_VVDEMO.

The following notes are extracted from the comments in the VVDEMO program and describe the four modes in more detail than is available on the demonstration program screen. It should be clear that there are many possible variations on the way in which the screen interaction may be programmed, even within the general limits of the four modes suggested by the demonstration program.

Routine Model Routine

This is the simplest and probably most useful way of using routine VV RSTRG (one of the procedures provided by VVP). The cursor is placed at the input position, after the 'prompt', which is not really a prompt in the conventional EMAS sense, but merely a printstring. Then when VV RSTRG is called, the input is typed at the keyboard, with the facility of backspacing and overtyping. The input is terminated with Return. Input of any other control character causes the input to be ignored, and VV RSTRG returns a null string.

The user may be able to move the cursor off the original screen line, and to change the screen in an arbitrary way. If he does this, VVP is able to 'repair' the screen (i.e. change it back) as soon as it detects that this has happened.

After the Return key is pressed, the characters typed in are erased from the screen. The model routine then checks the input line for "Select mode" or a line commencing "n<", where n is an integer. The input is printed on the "Data received" line, and the action implied, if any, is taken. Possible special actions are: select new mode, send text to line n of Box A.

If alterations are made to the input line to the left of the initial cursor position (when VV RSTRG is called), the screen is 'repaired' to its former state and a null string is returned to the calling program.

Routine Mode2 Routine

All the comments for the model routine apply. In addition, if the input data comprise the characters "#n" where 1<=n<=8, then n identifies one of the eight numbered areas in "Box A". Using VV GIVE OLD LINE and the record array of starts and lengths of the areas, the demonstration program extracts the text from the Box and copies it to the normal input area. VV RSTRG is then called again after the cursor has been placed at the beginning of the text just printed. Now the user at the keyboard can edit the line using the same conventions described above, and the resulting message is passed back by VV RSTRG when the Return key is pressed. Again, VV RSTRG clears the input line.

Routine Mode3 Routine

This routine operates exactly as the mode2 routine, except that when the "#n" input line is received, data are not moved from the specified Box area, but instead the cursor is placed at the start of the specified text in the Box. The same editing functions are now available at the keyboard. The situation is in this case different from those described for modes 1 and 2, in that the intended input area does not reach to the end of the input line and data beyond the end of the input area is not intended to be altered. The VV RSTRG LIMIT procedure is supplied to specify the maximum length of string to be delivered by VV RSTRG, and hence to specify, with the initial cursor position when VV RSTRG is called, the limits of the intended input area. If any alteration is made outside this area, the VV RSTRG call returns a null string and the screen is 'repaired' to its former state.

In this case, where the VV RSTRG LIMIT routine has been used, VV RSTRG does not erase the input data from the screen line, but restores it to its former contents, before any editing took place.

Routine Mode4 Routine

This routine demonstrates the use of VVP as a menu input system. The VV DEFINE TRIGGERS routine is called to allow up/down cursor movements as well as left/right movements. When the routine is entered, the cursor is placed near the top and right of the Box area. (This allows backspace and line-feed keys to be used to get the desired positioning even at those terminals which do not have cursor-up/cursor-right keys.) Textual input is disallowed by this routine (input containing printable characters is ignored, and the screen is repaired to its former state). The required menu item is indicated by placing the cursor on the required item and pressing the Return key. The text from the indicated Box area is then taken as program input and written on the "Data received" line. The cursor goes back to the original position in the Box unless a mode change has been indicated.

Note that text can be placed (or replaced) in the Box areas by the use of an input line commencing "n<", where n specifies the Box area. (This comment applies equally to the other modes, described above.)

Using the facilities demonstrated by VVDEMO, it is possible to construct a menu-driven input system, or a system where any existing line on the screen can be selected, modified and made the next input line. Thus, for example, a long editor command line would not have to be retyped in its entirety because one character was mistyped.

The program also demonstrates how the screen can be split into areas (or 'windows'). This facility is useful in many applications where it is desirable to be able to see two files (or two separate parts of the same file) at once.

The VFILES program (also in CONLIB.GENERAL, and described in User Note 56) provides an example of a use of VVP, being a video-oriented version of the Subsystem FILES command.

Appendix 2: Demonstration program No. 2

The source for this program is in file CONLIB.VVP_VVDEMO2. The program demonstrates the use of VV MENU and VV EDIT PAGE, and will work both on hardcopy and video terminals.

```
%begin
! This demonstration program presents a menu and allows the user
! to create and edit records using a page editor

%external %routine %spec vv editpage(%string %array %name s,
   %integer %name flag)
%external %routine %spec vv close
%external %routine %spec vv menu(%string (*) %array %name s,
   %integer nheading, nlines, %integer %name selection)

%own %string (80) %array empty(1:10)=""(*)

%own %string (80) %array menu(1:4)= %c
   "This is the menu heading which contains information and is not selectable",
   "Edit records",
   "Edit an empty page",
   "Quit"

%own %string (80) %array s(1:20)=%c
   "Name.....",
   "Address, Street.....",
   "Address, Town.....EDINBURGH",
   "Address, County.....",
   "Address, Post Code.....",
   "Sex.....",
   "Nickname.....",
   "Job.....",
   "Age.....",
   "Favourite colour.....",""(*)

%const %integer no=0, yes=1
%integer i, array changed
%switch menuswitch(1:3)

   array changed=no
   %cycle
       vv menu(menu, 1, 3, i)
       ->menuswitch(i)

menuswitch(1):
       i=50 {"protected" columns"}
       vv editpage(s, i)
       %if i=0 %then array changed=yes
       %continue

menuswitch(2):
       i=0 {allow typing in all columns}
       vv editpage(empty, i)
       %continue
   %repeat
```

```
menuswitch(3):
   vv close
   %if array changed=yes %start
        printstring("Current contents of array S")
        newline
        %for i=1, 1, 20 %cycle
            printstring(s(i))
            newline
        %repeat
   %finish
%end %of %program
```

```
Command:run testvv
This is the menu heading which contains information and is not selectable
------------------------------------------------------------------------------
A Edit records
B Edit an empty page
C Quit
Which:A
End, Abort or Restart the edit by typing #U, #A or #R followed by RETURN.
------------------------------------------------------------------------------
Name.....
New data, #A/U/R or RETURN
:Walter Scott
Address, Street.....
New data, #A/U/R or RETURN
:College Wynd
Address, Town.....EDINBURGH
Replacement data, #A/U/R or RETURN
:
Address, County.....
New data, #A/U/R or RETURN
:Midlothian
Address, Post Code.....
New data, #A/U/R or RETURN
:
Sex.....
New data, #A/U/R or RETURN
:Male
Nickname.....
New data, #A/U/R or RETURN
:Not known
Job.....
New data, #A/U/R or RETURN
:Writer to the Signet
Age.....
New data, #A/U/R or RETURN
:Full
Favourite colour.....
New data, #A/U/R or RETURN
:Not known

New data, #A/U/R or RETURN
:#U
This is the menu heading which contains information and is not selectable
------------------------------------------------------------------------------
A Edit records
B Edit an empty page
C Quit
Which:C
Current contents of array S
Walter Scott
College Wynd
EDINBURGH
Midlothian

Male
Not known
Writer to the Signet
Full
Not known
```