L1 MOS

T-FORM / VPL
Form Preparation Tools
Reference Manual

olivetti

PREFACE

This manual describes the T-FORM/VPL program, used for preparing forms which can be executed on L1 MOS.

A knowledge of the VISA environment on L1 MOS is assumed.

SUMMARY

The manual is divided into two parts:

- the first contains a description of T-FORM used for creating and/or modifying forms which can be interpreted by VISA.
- the second part gives information and rules on writing a Validation Program.

References:

Read first ...

MOS-VISA Form Management Package Programmer Guide

Code 4004390 B (vol. 6g)

For further information, read ..

MOS-SHELL COMMANDS Reference Manual

Code 4002770 Q (vol. 3)

COBOL Reference Manual

Code 4004290 H (vol. 6d)

First Edition:
July, 1984 - Release 3.0

Second Edition:
October, 1984 - Release 4.0

Update:
February, 1985 - Release 4.2

PUBLICATION ISSUED BY:

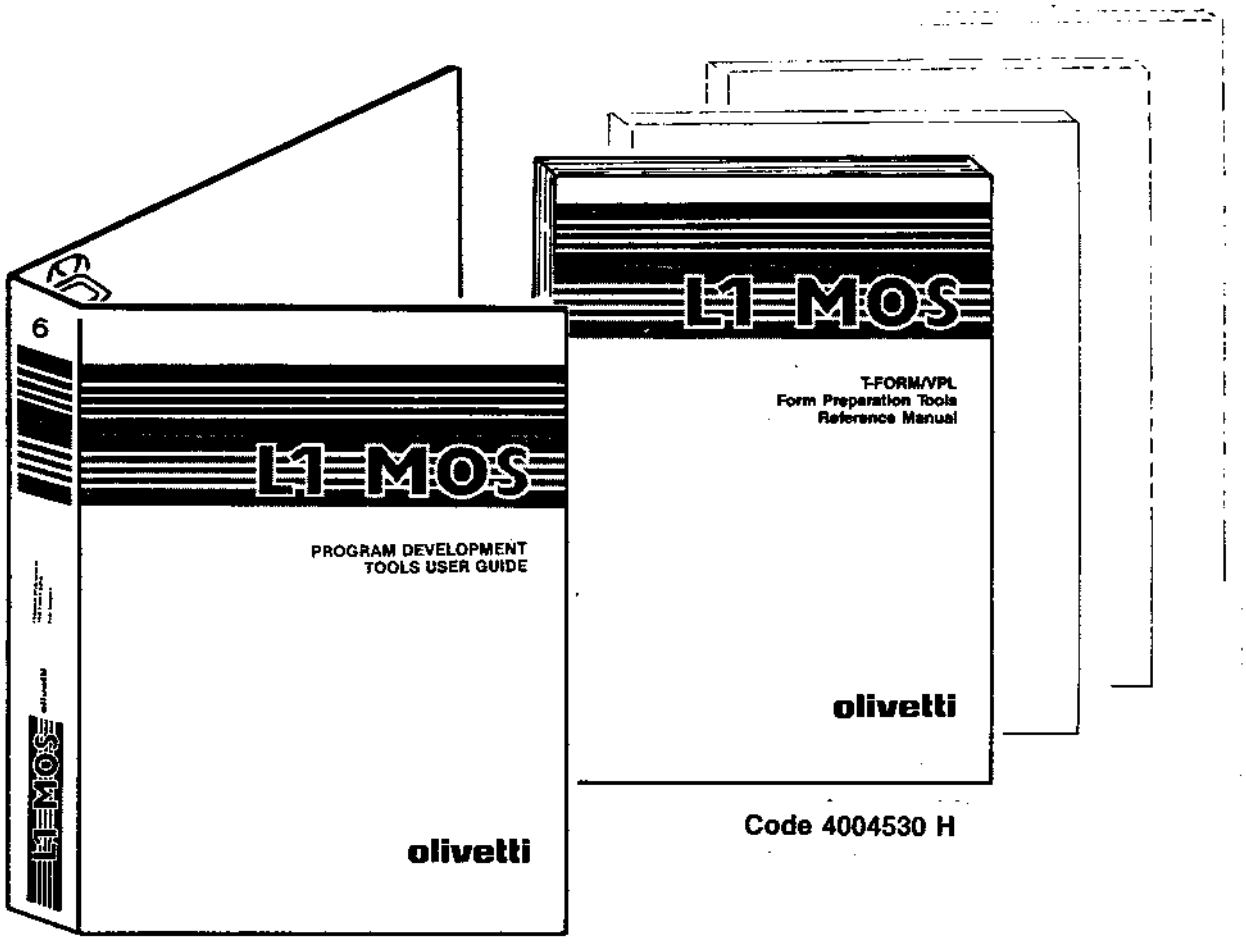
Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione,
77, Via Jervis - 100151 IVREA (ITALY)

GENERAL INTRODUCTION

The second part of this manual contains information and rules to be followed when writing a validation program, divided as follows:

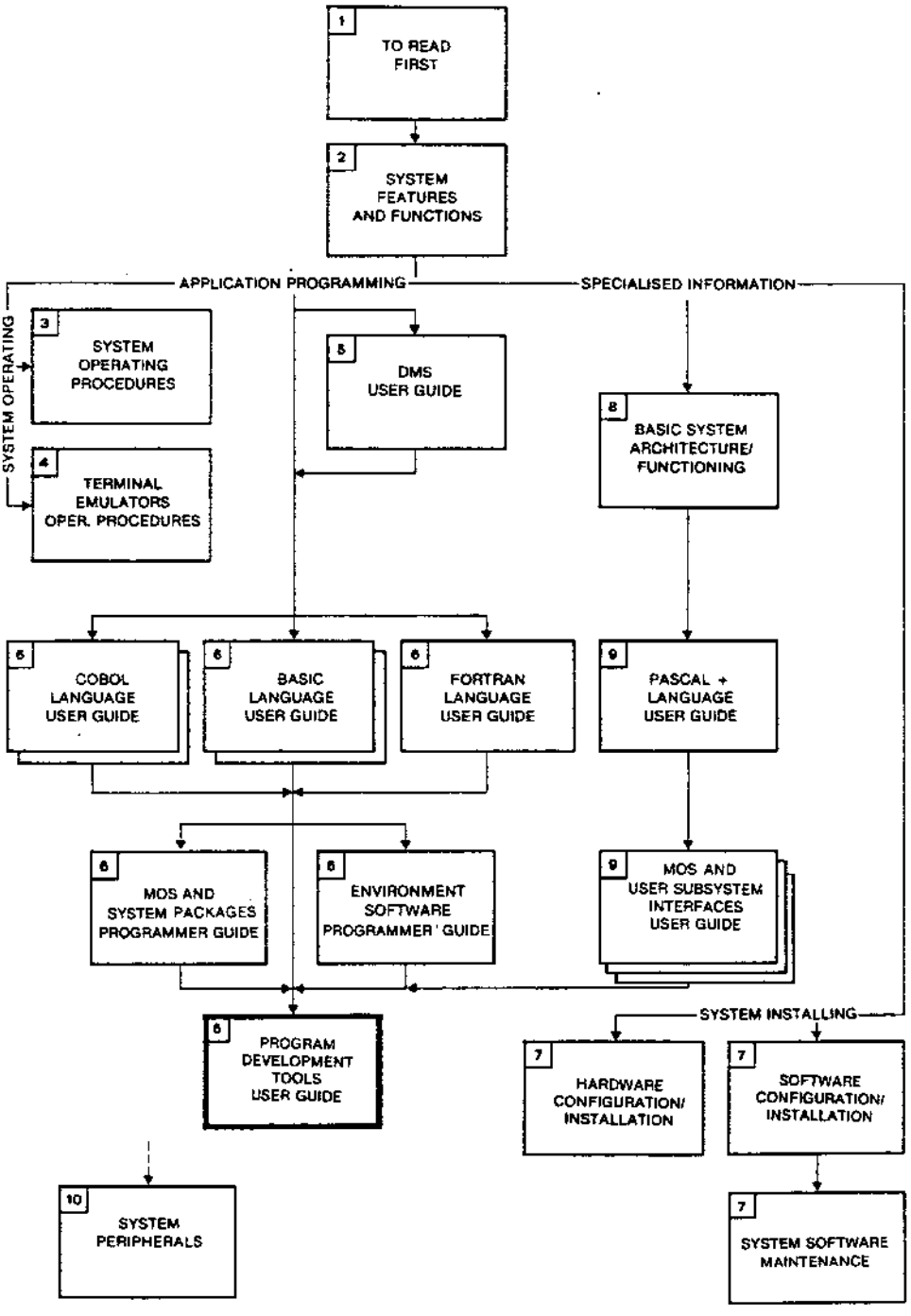
- the validation program

- the validation language.



Code 4004530 H

Code 4004850 S


1
TO READ FIRST
2
SYSTEM FEATURES AND FUNCTIONS
APPLICATION PROGRAMMING
SPECIALISED INFORMATION
SYSTEM OPERATING
3
SYSTEM OPERATING PROCEDURES
4
TERMINAL EMULATORS OPER. PROCEDURES
5
DMS USER GUIDE
8
BASIC SYSTEM ARCHITECTURE/ FUNCTIONING
6
COBOL LANGUAGE USER GUIDE
6
BASIC LANGUAGE USER GUIDE
6
FORTRAN LANGUAGE USER GUIDE
9
PASCAL + LANGUAGE USER GUIDE
6
MOS AND SYSTEM PACKAGES PROGRAMMER GUIDE
6
ENVIRONMENT SOFTWARE PROGRAMMER GUIDE
9
MOS AND USER SUBSYSTEM INTERFACES USER GUIDE
6
PROGRAM DEVELOPMENT TOOLS USER GUIDE
SYSTEM INSTALLING
7
HARDWARE CONFIGURATION/ INSTALLATION
7
SOFTWARE CONFIGURATION/ INSTALLATION
10
SYSTEM PERIPHERALS
7
SYSTEM SOFTWARE MAINTENANCE

PART 1- FORMS TRANSLATION PROGRAM: TFORM

INTRODUCTION TO PART 1

PART 2- VPL

PART 1

- FORMS TRANSLATION PROGRAM: TFORM

INTRODUCTION TO PART 1

This part describes the TFORM utility program used to create (or modify) forms interpretable under L1 MOS.

1. INTRODUCTION

TFORM is a utility program that allows you to create and handle forms interactively.

The Form

A form is a set of fields identified by a name (the name of the form).
Each field of the form is identified by a name and described by a set of attributes that define its characteristics (for example, numeric field, with or without sign, number of digits in the fractional part etc.).

When the form is executed (i.e. interpreted by VISA), the field attributes are used to handle input-output operations on the fields of the form.

When creating a form, you use the screen area to physically position the various elements of your form.

TFORM is fast and easy to use as it differs from the conventional description of screen space using coordinates.

When you have finished creating your form, TFORM stores the form as object code in a library, which is usually allocated on a disk unit.

The "user type" of the byte-stream file containing the forms is handled by TFORM according to the distribution code conventions.

Once the form has been catalogued in a library, it can be used under L1 MOS either for execution by VISA or you may call it from the library and modify, display or print it.

A form may be executed by any type of programming system not provided with screen management and by the COBOL language, which interfaces VISA with calls to external procedures.

TFORM Functions — TFORM provides the following six functions:

Form Editing — To create a new form or display and modify a form already catalogued in the library.

creation — A form is created in a number of different phases:

- layout definition: during which you draw the frame of the form on the screen and position the fields and labels.
  Definition of a simple layout is illustrated below (fields are represented on the screen by as many "*" characters as are necessary to define its length):

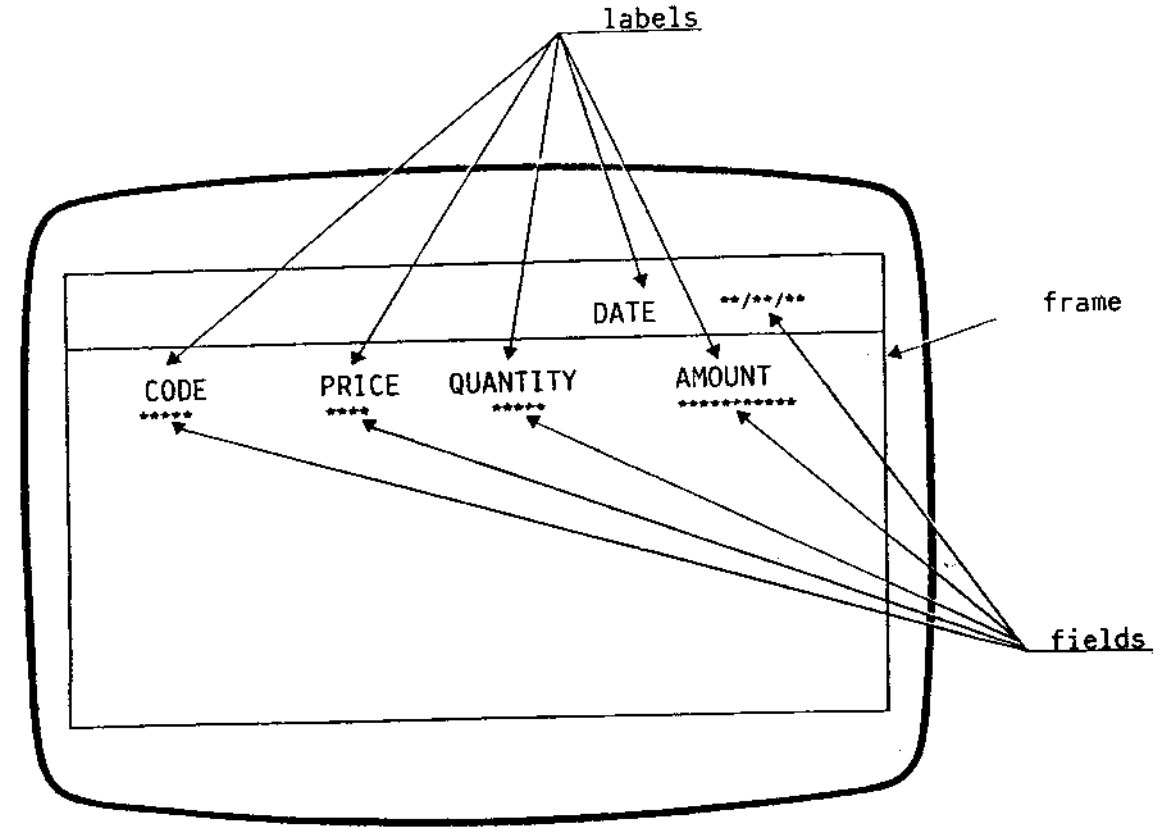


Fig. 1. 1 - Layout Definition

- field attribute definition: during which you define the characteristics of each field of your form.

- definition of subform structure: once you have completed the first two operations, you can now define the actual internal structure of the form itself. In this way, you can establish the order in which VISA is to execute the various fields of the form. A subform may be at least one field or at the most the entire form.

- device parameter definition: during which you define the parameters of each device defined for the form in reply to the editing menu.

- function keys definition: during which you can assign a different function to these keys other than their standard function, according to your specific requirements.

- validation program definition: if you have asked for at least one field to be validated when defining the field attributes, TFORM starts the editing phase in order to create the validation program (see the second part of this manual).

At any time during form creation, you can correct parts already defined using the same TFORM function as used to modify a form already present in the library (see below).

modification — The form is displayed at the start of this function, so you can modify any of the parts already defined also using special editing features that make it easier to move the characters on the screen.

cataloguing — Once you have created or modified your form, you catalogue it in a library.

Form Printing — You can print a form on the system printer (or the work station printer if the system printer is not available) or append the listing to a text file.

Directory Display — This function displays a list of all the forms catalogued under the current directory.

Form Testing — This function allows you to execute a sequence of VISA calls to simulate an application program using the forms catalogued under the current library.

Form Linking	This function allows you to create a library containing a specific set of forms and their corresponding validation programs.
Linked Forms Display	This function displays a list of all forms linked under a specified library.
The Work Station	This includes: - a 9" or 15" screen with a capacity of 2000 characters; smaller screens cannot be used - the keyboard (DP version) - an external storage device - a printer where printing is required.

2. TFORM EXECUTION

This chapter starts with a description of TFORM functions which also shows their logical and operating connections. This is followed by a description of:

- the files used by TFORM
- the commands used to execute TFORM
- the initial menu displayed by TFORM.

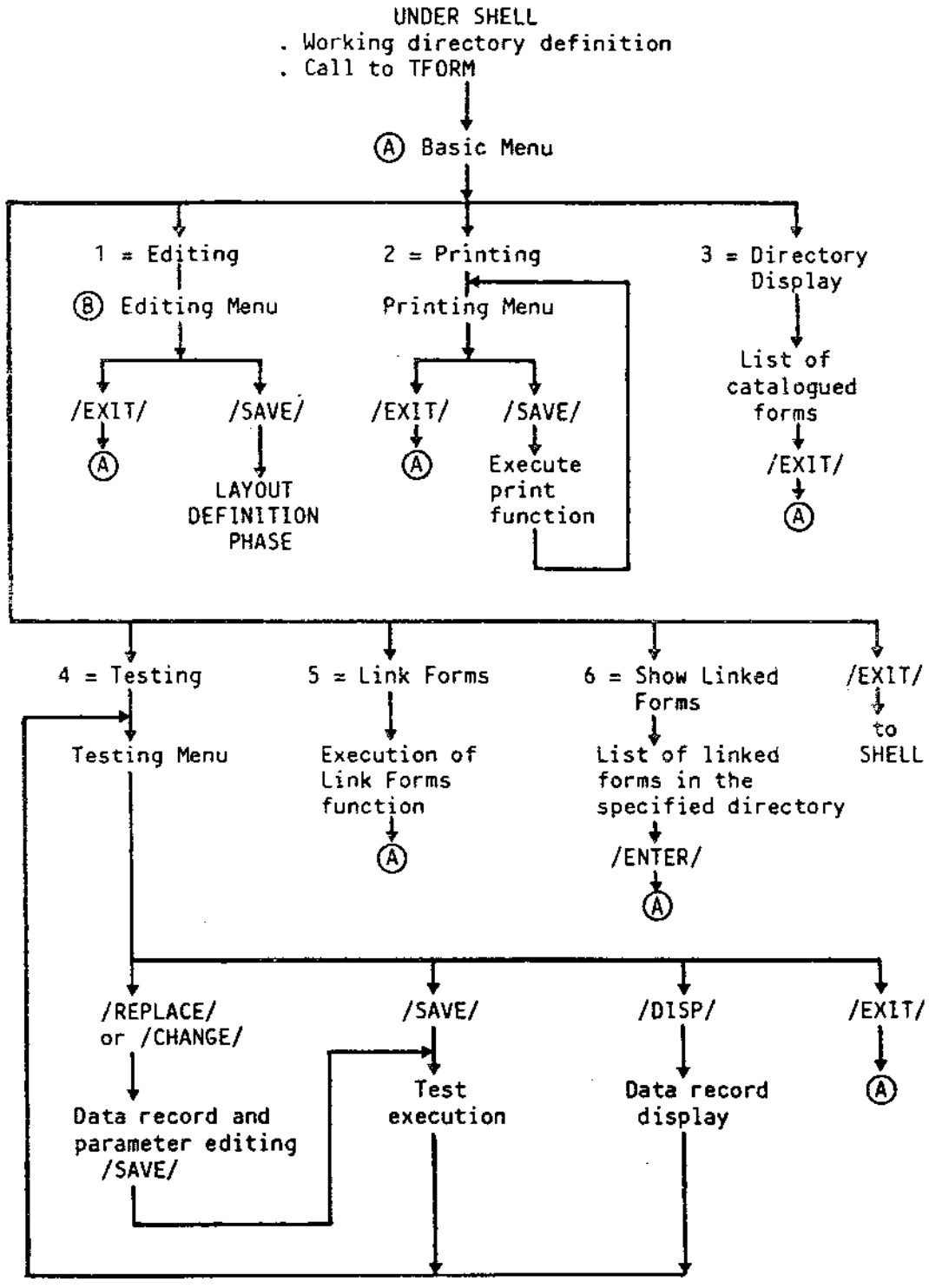


Fig. 2. 1 - TFORM Functions Logical Scheme (I)

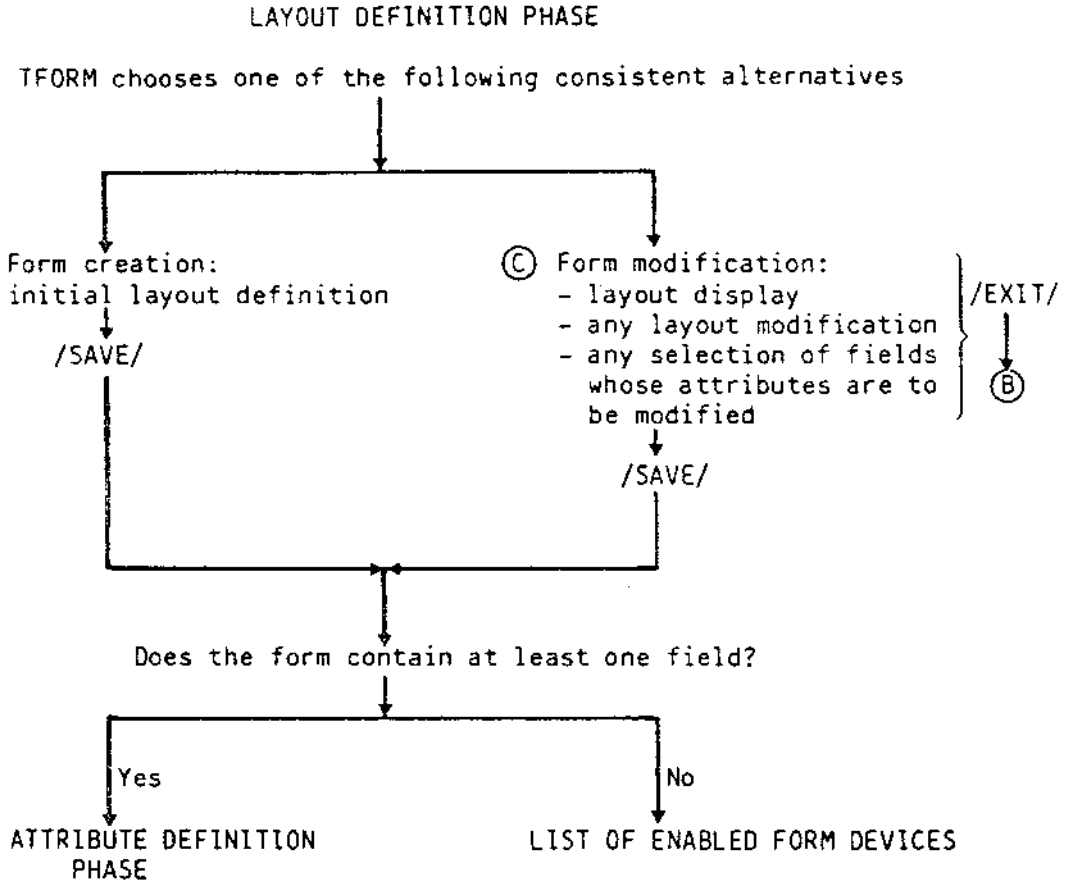


Fig. 2. 2 - TFORM Functions Logical Scheme (II)

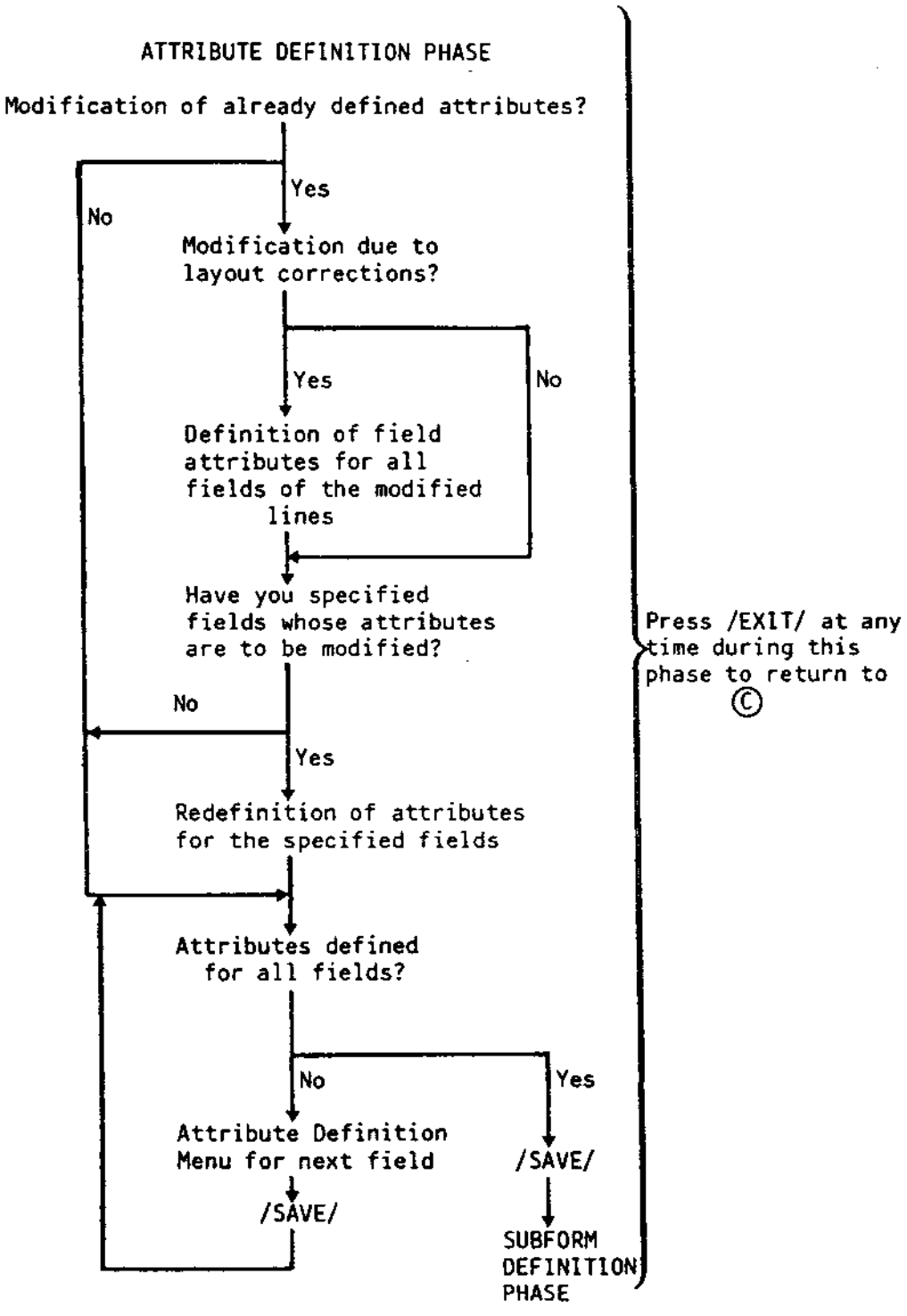


Fig. 2. 3 - TFORM Functions Logical Scheme (III)

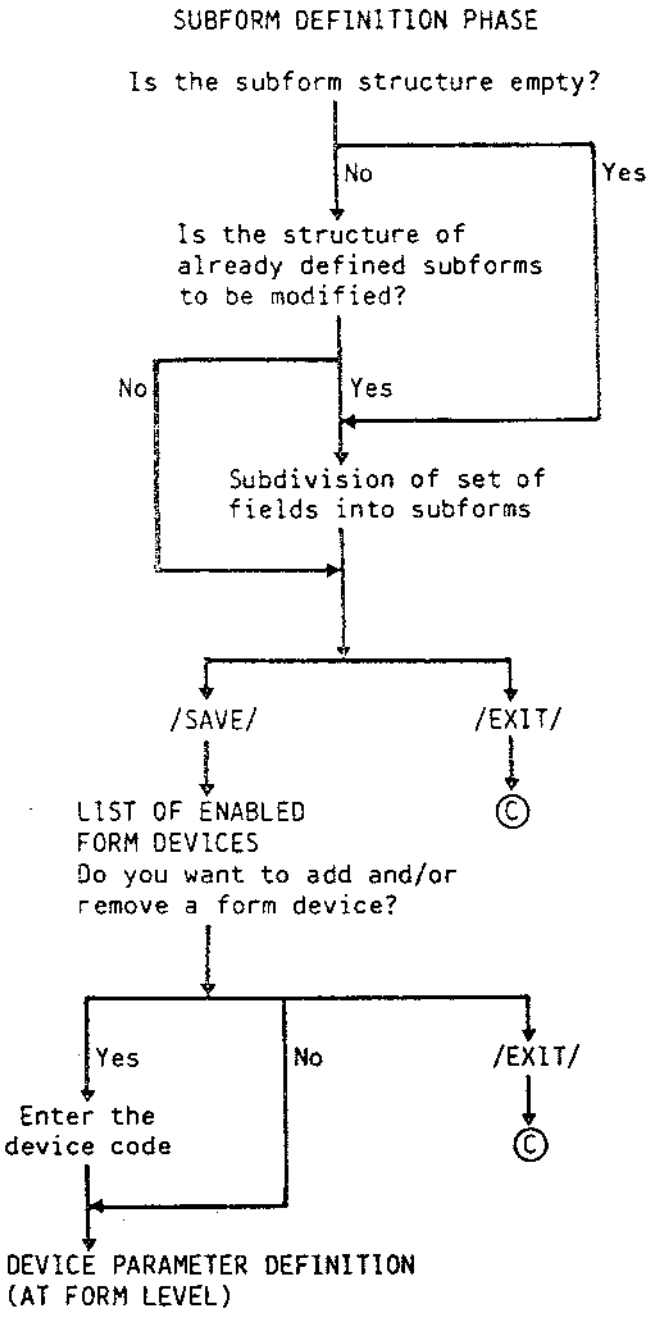


Fig. 2. 4 - TFORM Functions Logical Scheme (IV)

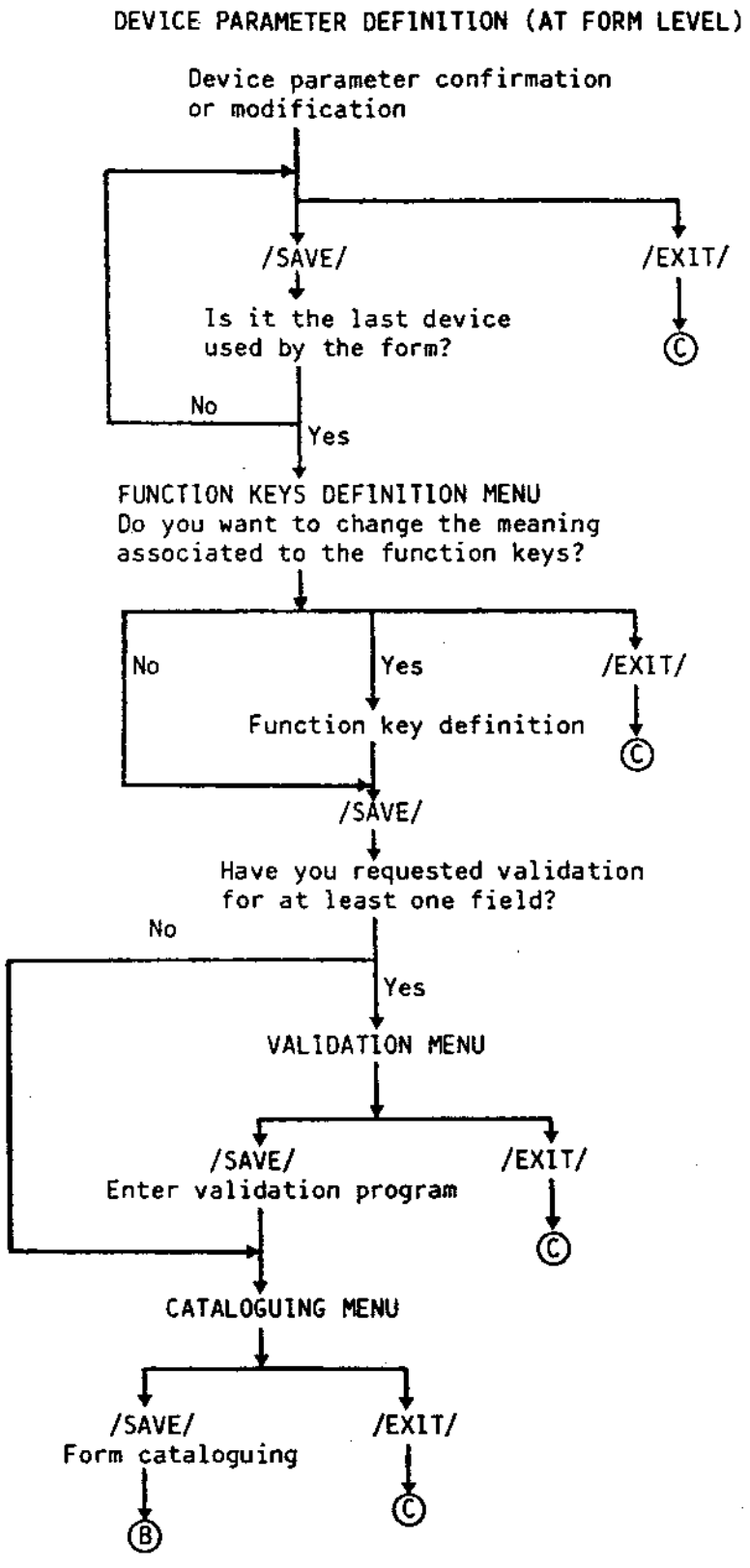


Fig. 2. 5 - TFORM Functions Logical Scheme (V)

Files Used by TFORM

These are as follows:

- an executable program directory called "TFORM"
- the form library in which TFORM catalogues the new or modified forms and from which it reads the forms to be modified or printed.

Commands Used to Execute TFORM

If under SHELL, you want to specify a work directory other than the default directory, you execute the following command before calling TFORM:

```
SETWDIR    <directory pathname>
```

TFORM is then executed by entering:

```
TFORM
```

Obviously, if you are already positioned in the required directory, the SETWDIR command is not necessary.

For further information, refer to the following manual: "MOS - SHELL COMMANDS-REFERENCE MANUAL", code 4002770 Q.

To execute TFORM, you must complete the menus displayed by TFORM. For each parameter of the menu, you can:

- enter the value required or confirm any existing value, then press /ENTER/ to position on the next parameter
- press /HOME/ to return the cursor to the first parameter of the menu
- press / |← / to move the cursor to the previous parameter.

Initial TFORM Menu

At the start of the work session, TFORM displays the following menu:

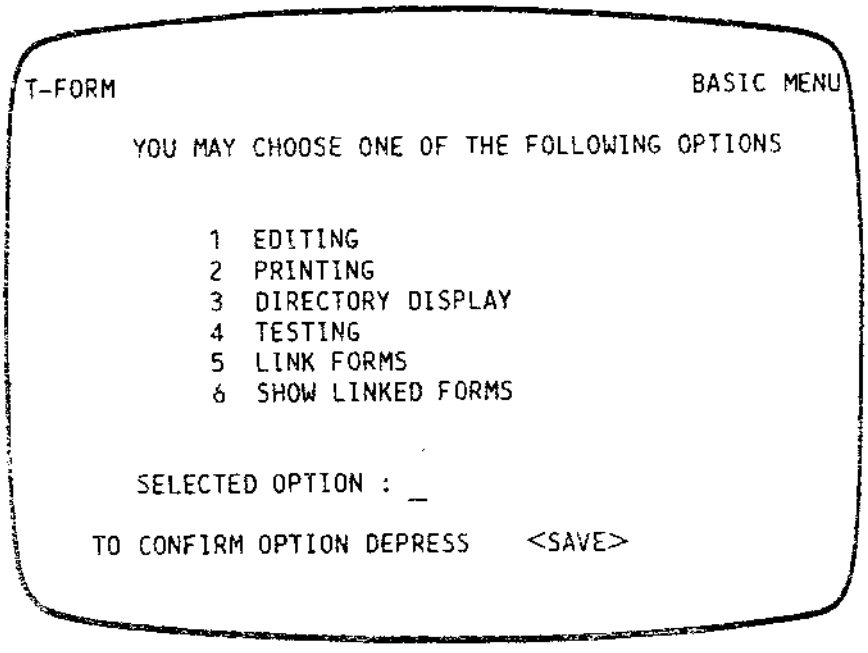


Fig. 2. 6 - Basic Menu

You must enter the number of the TFORM option required, as shown in the menu, and confirm your choice by pressing /SAVE/.
TFORM provides the following default values:

- at the start of the session: 'blank'
- during the session: the last value you have entered.

If you press /EXIT/ on this menu, execution of TFORM is interrupted and control is returned to SHELL.

The TFORM functions are described in the following chapters. Form creation and form modification, which are sub-functions of the editing function, are described separately.

3. FORM CREATION

This chapter describes how to create a form; refer to the next chapter "FORM MODIFICATION" for any modifications that may be necessary during form creation.

As the operations are the same both for modification of a form being created and a form already in the library, they have been described once only for the sake of clarity.

EDITING MENU

If you have selected option 1 (EDITING) of the Basic Menu, TFORM displays the following menu:

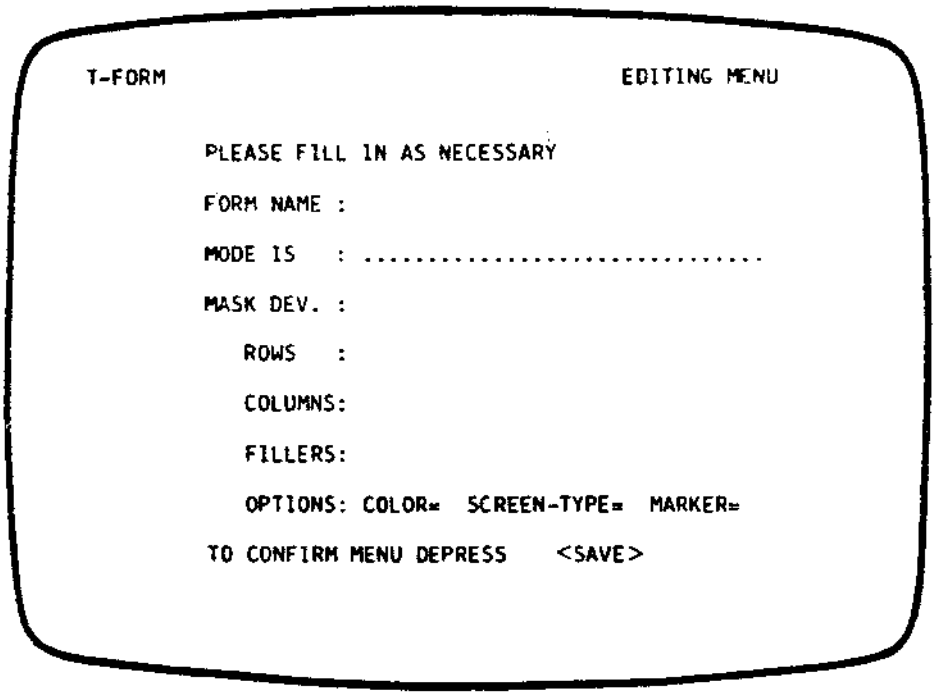


Fig. 3. 1 - Editing Menu

where:

FORM NAME — This is the name of the form, consisting of 1 to 12 alphanumeric characters, the first of which must be alphabetic.

MODE IS — This is a display field in which TFORM displays one of the following:

- CREATION OF A NEW FORM
  if the form name you have entered does not already exist in the form library.

  or

- MODIFICATION OF AN EXISTING FORM
  if the form name you have entered exists in the form library. In this case, the parameters you defined for the form are re-displayed.

In both cases, you can decide to enter a different form name by pressing /EXIT/.

MASK DEV. — Here you must define one of the following form output devices:

- SK = screen (default value)
- SP = sprocket
- PL = whole platen
- LP = left platen (journal)
- RP = right platen (tally)
- AF = automatic front feed
- MF = manual front feed

ROWS — Here you must define the number of rows of the form (between 1 and 70). During form creation, the default value is = 1.

COLUMNS — Here you must specify the number of columns of the form (between 1 and 255). During form creation, the default value is = 1.

FILLERS — Here you must specify two characters which during form execution are used as filling characters after entry of a value in the fields defined as edited numeric (see Field Attribute Definition Menu). You define which of these two characters is to be used for each field during attribute definition.

The default values are '*' and 'blank'. The allowed values are any character except for:

- the digits 1 to 9
- the "." and "," characters used to edit numeric values.

COLOR This parameter defines which visual attributes will be enabled for the terminal screen. The value entered for this parameter must be compatible with the screen configuration; for colour screens you can enter "y" or "n", but for black and white screens only "n" is allowed. The default value is "n".

If you enter "y" (only for colour screens), the following visual attributes are enabled:

green

red + blinking

white

white + blinking

yellow + reverse

yellow + blinking + reverse

magenta + reverse

yellow

red

cyan

green + blinking

green + reverse

red + reverse

blue

magenta

If you specify "y" for a black and white screen, the requested visual attributes are not obtained and you will not be alerted of this by an error message.

If you enter "n", the following visual attributes are enabled:

COLOUR SCREEN	B/W SCREEN
green	normal
red + blinking	blinking
white	highlight
white + blinking	blinking + highlight
yellow + reverse	reverse
yellow + blinking + reverse	blinking + reverse

The choice of the required visual attribute is made during the Layout Definition phase.

SCREEN-TYPE This parameter specifies that the form can be associated to different sized screens, that is to say, the size of the screen characters can be modified to meet your specific requirements.

The allowed values are as follows:

0 : to indicate no change of the character size on the screen

1 : to indicate 520 characters (13 rows x 40 columns)

2 : to indicate 1000 characters (25 rows x 40 columns)

3 : to indicate 2000 characters (25 rows x 80 columns).

If this parameter is not to be enabled, enter 0 (the default value).

MARKER This optional parameter allows you to specify a filler for the input fields; the specified filler will be displayed in all the input fields at run time prior to user input. The allowed values are any character except for:

- the digits 1 to 9
- the characters a to z and A to Z.

During execution of the Editing Menu:

- press /EXIT/ to return to the "FORM NAME" parameter. If you press /EXIT/ on the "FORM NAME" parameter, TFORM returns to the Basic Menu

- press /SAVE/ to go to the TFORM layout definition phase.

LAYOUT DEFINITION

During this phase, you use the screen to define the physical characteristics of your form, that is to say:

- you draw the frame
- you position the labels and fields
- you define the visual attributes.

Remember that you can omit any form element, depending on your specific requirements.

You can move the cursor only inside the area of the form you have defined.

Use the following keys to move the cursor:

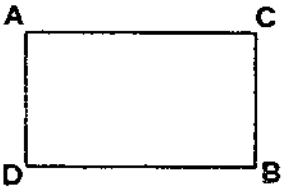
/ ↑ /	to move the cursor one position up
/ ↓ /	to move the cursor one position down
/ ← /	to move the cursor one position left
/ → /	to move the cursor one position right
/ENTER/	to move the cursor to the first position of the next line. If the current line is the last line of the form, the cursor returns to the first position of the line itself.
/ ⇤ /	to move the cursor to the first position of the previous line. If the current line is the first of the form, the cursor returns to the first position of the same line.
/ →| /, / ↵ /	same effect as /ENTER/

Form Frame

You can trace rectangles and/or lines on the screen following the rules given below:

- to define a rectangle you must specify the two extremities of a diagonal.

For example, in the following rectangle:



the extremities are A, B or C, D.
To indicate the two extremities of the diagonal, position the cursor at the point required and press /PICK/, repeat for the other extremity.

Once you have defined the two extremities of the diagonal, TFORM will trace the rectangle on the screen when you press /WIND/.

Therefore, if you wanted to draw the above rectangle, you would proceed as follows:

. cursor to A - /PICK/

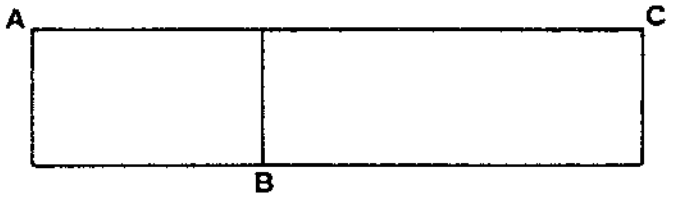
. cursor to B - /PICK/

. /WIND/

TFORM draws the rectangle.

You could draw the same rectangle by specifying the extremities C and D.

- TFORM assumes that the vertical side of an already defined rectangle may also be the side of a contiguous rectangle.
For example, the following rectangles are drawn as follows:



. cursor to A - /PICK/

. cursor to B - /PICK/

. /WIND/

TFORM draws:

. cursor to C - /PICK/

. /WIND/

the figure becomes:

- to draw a horizontal or vertical line, all you have to do is define two points on the same row or column.
  For example, the following horizontal line:

  A _____ B

  is drawn as follows:

  . cursor to A - /PICK/

  . cursor to B - /PICK/

  . /WIND/

- the same operation is used to delete an already existing rectangle or line; that is to say, you specify the two points using the cursor and the /PICK/ key and then press /WIND/.

  When you press /WIND/, TFORM proceeds as follows:

  . if a figure does not already exist, it is created

  . if a figure already exists, it is deleted.

Remember that:

- the lines and geometric figures drawn by TFORM do not occupy screen positions

- the /WIND/ key always refers to the last screen sector you have defined, regardless of where the cursor is pointing when /WIND/ is pressed

- during layout definition, only the last two points defined with the /PICK/ key are stored at each step.

As the /WIND/ key can be used both to create or delete a frame, its specific functions are explained in more detail below:

- in a sequence of the following type, /PICK/ /PICK/ /WIND/, the positions at which the /PICK/ key was pressed identify the opposite corners of the figure that has been created; if the sequence is /WIND/ /PICK/ /WIND/, the /WIND/ command considers the positions at which the /PICK/ key was pressed last and the time before as opposite corners of the figure

- if the points specified have the same row number, a continuous horizontal line is drawn (or deleted) between and including the two positions

- if the points specified have the same column number, a vertical line is drawn (or deleted) to the right of the positions indicated by the cursor

- if neither of the two cases above apply (same row number, same column number), a rectangular frame is drawn (or deleted), which "surrounds" the positions indicated by the cursor when the /PICK/ key is pressed

- before drawing or deleting a frame, TFORM analyses only the upper lefthand corner; if there is a section of frame in this position, this is deleted, otherwise it is created

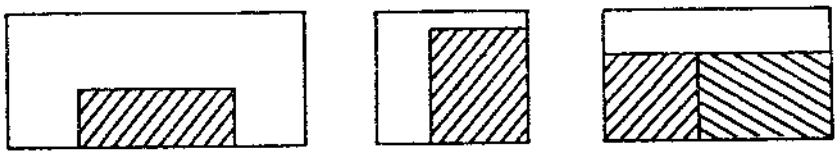
- each time a frame is created or deleted, TFORM automatically draws or deletes the section of frame for each position indicated by the cursor (i.e. the upper, lower, right, left line or no line).

Initially, the section of frame required is drawn in the position indicated by the cursor, thus deleting any previously defined section.

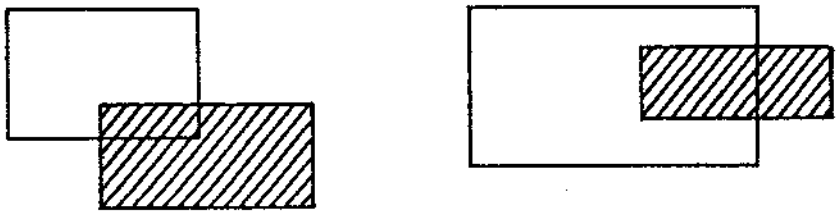
If certain sections of two frames intersect, the sections of the second frame remain at the points of intersection while the previous section is deleted. This does not happen when the two frames have one side or part of this in common, or even better, if they do not intersect each other.

Given below are a few examples of how figures can be composed and where this is not possible.

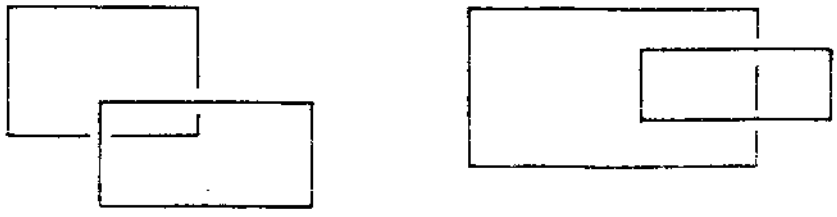
possible compositions



impossible compositions



In the first case (possible compositions), the second figure (shaded) is completely contained inside the first; in the second case (impossible compositions), the second figure (shaded) is only partially contained inside the first, so the sectors of the first figure are deleted where the two figures intersect each other, as shown below:



form larger than screen

When the your form is bigger than the screen (24 rows, 80 columns), you can display the excess area as follows:

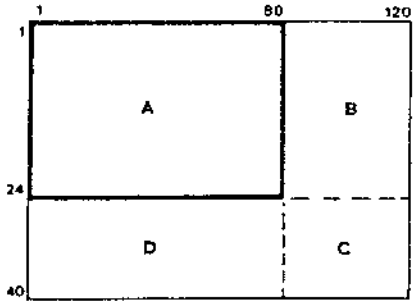
- imagine that the screen is a fixed frame in which you can scroll your form: if you press /+PAGE/, the form is scrolled so that the position actually indicated by the cursor moves to ROW 1 - COLUMN 1 of the screen
- if you press /HOME/, ROW 1 - COLUMN 1 of your form is returned to ROW 1 - COLUMN 1 of the screen.

Example

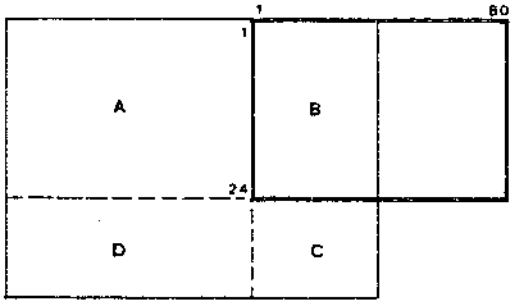
Assuming that you want to define a form measuring 40 rows and 120 columns (see figure below).
In the figure, the letters A, B, C, D indicate the four areas into which the form is ideally divided; the heavily scored line indicates the screen.

The figure shows the initial situation of the screen: to display areas B, C and D as shown in the figure you proceed as follows:

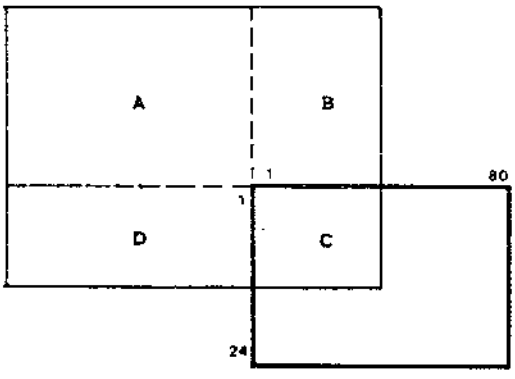
- to display area B, move the cursor to ROW 1 - COLUMN 80 and press /+PAGE/
- to display area C, move the cursor to ROW 24 - COLUMN 1 and press /+PAGE/
- to display area D, you return first of all to area A by pressing /HOME/ and then move the cursor to ROW 24 - COLUMN 1 and press /+PAGE/.



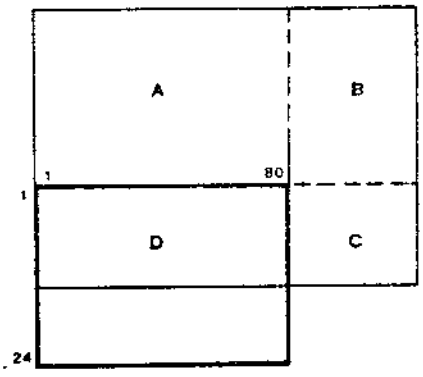
Initial situation



Display of area B



Display of area C



Display of area D

Fig. 3. 2 - How to Display Different Areas of the Same Form

Labels

Labels are usually used to identify the meaning of a field or the contents of a certain area of your form.

Fields

You indicate the length of a field with asterisks, for example, a 5-character field is indicated with 5 asterisks.

A field may occupy an entire line of the form.

A field cannot be continued onto the next line.

Contiguous fields must start with the @ or § character depending on the keyboard used.

You can, in theory, define up to 96 fields in a form; if you define more than 96, the "TOO MANY FIELDS" error message is displayed and control is returned to the layout phase.
For further details, see the section on "subforms".

Visual Attributes

These are used to define how the form is to be displayed (for example, highlight, blinking, reverse, colour etc.). The available attributes depend on what you have specified for the COLOR option on the Editing Menu.

You define the screen sector you want to display with certain visual attributes in the same way as when defining form layout, i.e. by defining two points of the sector.

The sector may therefore be a vertical or horizontal line or the area of a rectangle.

You can also define how you want a field, a label, a character or a set consisting of a field and a label to be displayed.

Once you have defined the sector, you press the /DISP/ key one or more times until the contents of the sector are displayed as required.

Each time you press the /DISP/ key, TFORM changes the visual attribute, displaying the sector in the various ways possible and then returning to the beginning.

Example DATE ******

To define the visual attributes of the DATE label and the following field, proceed as follows:

- move the cursor under the D of DATE and press /PICK/
- move the cursor under the E of DATE and press /PICK/
- press /DISP/ until the DATE label is displayed in the way you want
- move the cursor under the last * of the field and press /PICK/
- press /DISP/ until the field is displayed in the way you want.

Note that when you press /DISP/, the label or field will be displayed in sequence with all the possible visual attributes; when you release the key, you automatically select the way in which the field, label etc. indicated by the cursor is displayed at that moment.

If you have gone past the particular way in which you want to display the field, you just continue to press /DISP/ until the field, label etc. is displayed as you require and then release /DISP/. The notes applicable to frame design also apply to visual attributes:

- the visual attributes do not occupy screen positions
- pressing of the /DISP/ key always refers to the last screen sector you have defined with /PICK/, regardless of the position of the cursor when /DISP/ is pressed.

You can return to the Editing Menu at any time during the layout phase by pressing /EXIT/.

FIELD ATTRIBUTE DEFINITION

Now that you have defined the layout of your form, you can start to define the attributes of the various fields. You start by pressing the /SAVE/ key.

TFORM will display a menu for each field of the form you have defined. Part of this menu is already compiled; you must enter the values of certain parameters or confirm the default value.

The first two rows of function keys have special significance during field attribute definition. For most parameters a set of possible replys is shown, where each possible reply is represented by a function key; you simply press the appropriate function key to make your selection. You confirm your selection by pressing one of the following keys:

- /ENTER/ or /RETURN/ to go on to the next parameter
- /←/ to go back to the previous parameter
- /HOME/ to go back to the first parameter
- /EXIT/ to suspend the input phase, returning to the layout definition phase
- /SAVE/ to terminate the current input phase.

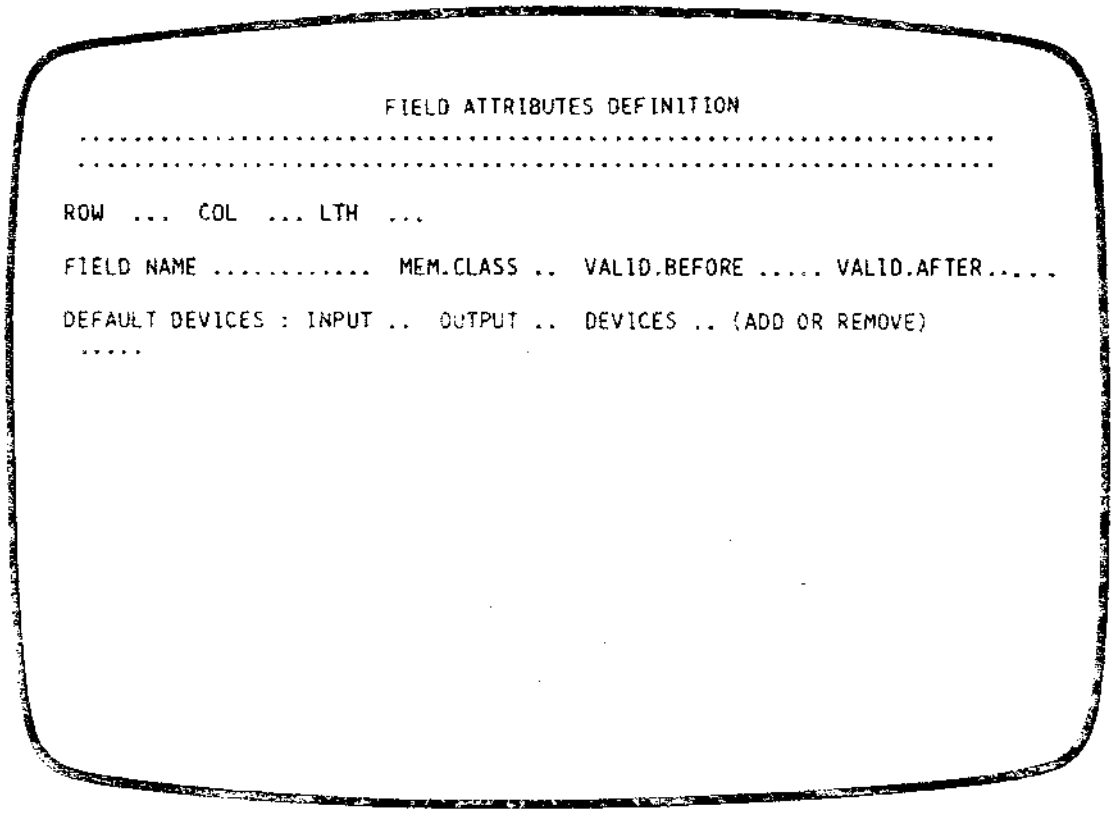


Fig. 3. 3 - Field Attribute Definition Menu

Each field attribute definition menu refers to a single field, even though the second line of the menu shows the entire line of the form (or the first 80 columns if your form is larger than the actual screen size) containing the field whose attributes are to be defined. In the example given in Fig. 3.3, this line consists of two fields that start respectively at positions (10,6) and (10,18) with a length of 4 and 3 characters.

Note that:

- TFORM indicates the current field by positioning a pointer (or logical cursor) under the first asterisk of the field (you cannot move this cursor).

- if your line is more than 80 columns wide, TFORM will display the fields in columns 81 onwards, one at a time, after you have defined the

attributes of the last field in the first part of the line (columns 1 through 80).

For the field in the menu shown above, TFORM first of all establishes the position of the field (ROW and COLUMN number) and its length (LENGTH) and then sets the cursor to the first parameter you must define.

You must specify the following field attributes:

FIELD NAME

This consists of 1 to 12 alphanumeric characters, the first of which must be alphabetic.
The default value is "blank".

If the name is less than 12 characters long, you must align these to the left.
You can use any printable character belonging to the ISO code between hexadecimal 21 and hexadecimal 7F (inclusive).

If the name of the field starts with an asterisk (*), it is interpreted as a "field for internal use only". This means that the Application Program cannot refer to it explicitly using VISA but only through a subform that contains it.
(You will find more information about subforms later in this manual).

MEM.CLASS

This indicates how the field is represented in the memory in the Application Program record.
The function keys have the following meanings:

LEADING INTERNAL	LEADING EXTERNAL	TRAILING INTERNAL	TRAILING EXTERNAL	ZONED UNSIGNED	STRING
PACKED	BINARY WORD	BINARY BYTE			

where:

- STRING (st) = string of ASCII characters (default value)
- ZONED UNSIGNED (un) = zoned type value without sign
- TRAILING EXTERNAL (te) = zoned type value with the sign separate, placed after the last digit
- TRAILING INTERNAL (ti) = zoned type value with the sign included in the last digit

- LEADING EXTERNAL (le) = zoned type value with the sign separate and placed before the first digit

- LEADING INTERNAL (li) = zoned type value with the sign included in the first digit

- PACKED (pk) = packed type value, i.e. with the sign always in the last digit

- BINARY WORD (bi) = binary value with sign, represented in words; : its length in the record is calculated as 2, 4 or 8 bytes according to the number of digits defined on the screen

- BINARY BYTE (bb) = binary value with sign, expressed in bytes; its length in the record is calculated as from 1 to 8 bytes according to the number of digits defined on the screen.

Press the corresponding function key and the abbreviated choice is displayed on the menu.

You should refer to the COBOL standard in the "COBOL Language Reference Manual" (code 4004290 H) for further details of the various meanings of the MEM.CLASS parameter.

VALIDATION BEFORE

This is a parameter which requests a line number. In the Validation Program this will be the starting line number of the routine associated to the field in question. (Refer to the second part of this manual for more details about the Validation Program).
VALIDATION BEFORE indicates that, at VISA time, when this field is reached, the validation routine associated to that field will be executed before any value is accepted.

This is useful, for example, only for deciding whether or not to go to that field, on the basis of previous input.
If no validation is required, enter 0 (the default value).
The maximum line number you can specify is 65529.

VALIDATION AFTER

Here you must enter a line number with the same meaning as that for VALIDATION BEFORE.

In this case, however, the routine located at the line number indicated will validate the field after entry of the field value.
Enter 0 (the default value), to indicate no validation for that field.
The maximum line number you can specify is 65529.

DEFAULT DEVICES: INPUT

Here you must indicate the type of peripheral unit from which the value of the field is to be entered when the form is executed.

The function keys have the following meanings:

KEYBOARD KEYS	CHEQUE READER	BADGE READER	MAGNETIC STRIPE	KEYBOARD	NULL DEVICE
PIN PAD					

Press the appropriate key and the corresponding abbreviation will be displayed on the menu as follows:

- sk for keyboard (default value)
- nd for null device
- bg for badge reader
- pp for pin pad
- ms for magnetic stripe reader
- ch for cheque reader
- kk for keyboard switch keys.

DEFAULT DEVICES: OUTPUT

Here you must indicate the peripheral unit on which the value of the field is output after form execution.

The function keys have the following meanings:

SPROCKET	RIGHT PLATEN	LEFT PLATEN	WHOLE PLATEN	SCREEN	NULL DEVICE
AUTOMATIC FRONT FEED	MANUAL FRONT FEED	MAGNETIC STRIPE	BADGE WRITER	- more -	

You can select one of these devices or press the '- more -' key to see the other options, which are:

CHEQUE WRITER	CASH ADAPTER				
				- more -	

Press the appropriate key (the '- more -' key allows you to see the previous options again) and the corresponding abbreviation will be displayed on the menu as follows:

- sk for screen (default value)
- nd for null device
- bg for badge writer
- ms for magnetic stripe writer
- sp for sprocket
- pl for whole platen
- lp for left platen (journal)
- rp for right platen (tally)
- af for automatic front feed
- mf for manual front feed
- ch for cheque writer
- ca for cash adapter.

ADDITIONAL DEVICES

This parameter allows you to specify other input and/or output devices to allow the Application Program, using VISA functions, to redefine the Input and Output device at run-time.

To do this, the Application Program must be able to refer to other additional devices, besides the default devices, as the device redefined by the Application Program must be declared for all the fields of the subform concerned.
(For further details, refer to the MOS-VISA Form Management Package, Programmer Guide, code 4004390 B).

The function keys have the following meanings:

MAGNETIC STRIPE	BADGE	CHEQUE	KEYBOARD KEYS	PIN PAD	CASH ADAPTER
SK/PR					

You can select any device which has not already been defined as a default device. You can also delete a previously defined additional device by pressing the appropriate key.

You can define up to 3 devices (input, output and additional), bearing in mind however the weight assigned to each device, that is to say, the number of devices defined must be less than or equal to a total weight of 3.

The weights of the various devices are given in the table below:

DEVICE	WEIGHT
sk/pr	1
nd	0
bg	1
pp	1
ms	1
ch	1
kk	1
ca	1

Note that sk/pr, where pr indicates any of sp, pl, lp, rp, af or mf, must be seen as a single peripheral unit.

This means that even if neither the form nor any of the fields contains the sk unit, when any of the pr devices is selected for the form, the connection with the sk is also enabled.

Given below are a few examples of additibnal device selection:

1) INPUT = sk
   OUTPUT = sk
   ADDITIONAL DEVICES = bg, ms

   in this case, you can define two additional devices as the weight of the Input and Output devices is equal to 1;

2) INPUT = sk
   OUTPUT = pl
   ADDITIONAL DEVICES = kk, ch

   also in this case, you can select two additional devices as the sum of the weights of the previous devices is equal to 1;

3) INPUT = bg
   OUTPUT = ms
   ADDITIONAL DEVICES = ch

   in this case, after removing sk from the default devices, you can select another additional device as the previous ones have a total weight of 2;

4) INPUT = nd
   OUTPUT = nd
   ADDITIONAL DEVICES = bg, ms, ch

   in this case, after removing sk from the additional devices, you have three possibilities of choice, as nd devices have weight = 0.

After defining all the devices (input, output, additional), you must specify the relative parameters for each.

In the lower part of the screen, the Field Attribute Definition Menu displays each device selected, asking you to define the parameters.

device=SK/PR The following menu is that displayed for the sk/pr device:

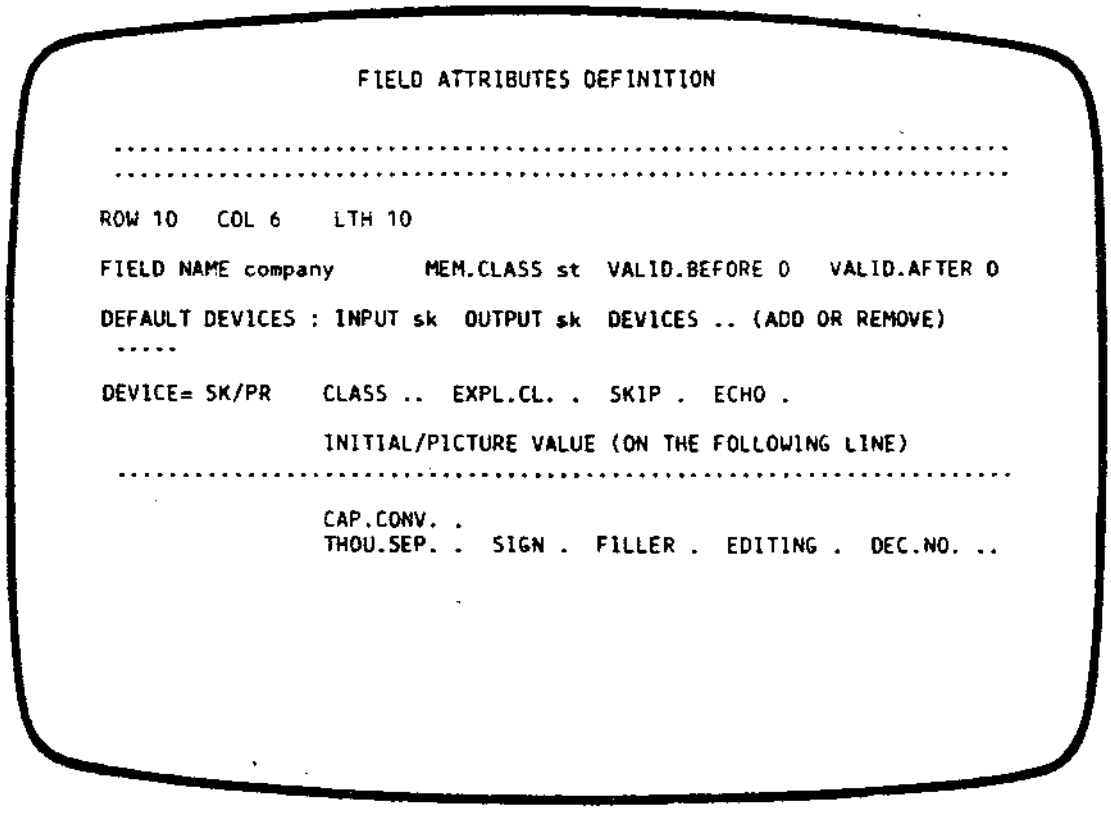


Fig. 3. 4 - Field Attribute Definition Menu with Device = SK/PR

where:

CLASS Defines how the field is represented on the device. The function keys have the following meanings:

NUMERIC CHARACTER	PICTURE FIELD	CODE	ALPHA NUMERIC	ALPHABETIC	GRAPHIC
NUMERIC EDITED					

where:

- ALPHABETIC (al) includes the characters A to Z, a to z and "blank"
- NUMERIC CHARACTER (nc) includes the characters 0 to 9 without editing characters
- ALPHANUMERIC (an) includes the characters A to Z, a to z, 0 to 9 and "blank"
- GRAPHIC (gr) includes any character that can be entered from keyboard (the default value)
- NUMERIC EDITED (ne) includes the characters 0 to 9 and the '+', '-', '.' and ',' editing characters
- PICTURE FIELD (pi) means that a valid PICTURE value must be entered for the field (see below)
- CODE (cd) means that the length of the data in the record is always set equal to 1.

EXPL. CL.

Indicates whether the field is explicitly closed. Only two of the function keys are enabled for this parameter, giving the options YES and NO.

Select YES (the default value) if the field is to be closed with a function key such as /ENTER/, /SKIP/, etc.

Select NO if the field is to be closed when the number of characters entered completes the length of the field.

SKIP

Indicates whether entry of the field value can be skipped when the form is executed. Again, only two of the function keys are enabled, giving the options YES and NO.

ECHO

Indicates whether the contents of the field are to be displayed. Again, only two of the function keys are enabled, giving the options YES and NO.

INITIAL VALUE

This is the value that you can assign to the field when creating or modifying your form.
If you wish to specify this value, you enter it on the next line of the screen. The value you enter must be consistent with the class specified for that field, with a maximum length of 80 characters.

PICTURE VALUE

This parameter is only requested if you have selected PICTURE for MEM.CLASS. You must enter a valid PICTURE value; refer to the "COBOL Language Reference Manual" for details.

CAP. CONV.

Indicates whether any alphabetic characters entered in lower case (a to z) are to be automatically converted to upper case (A to Z). Two of the function keys are enabled, giving the options YES and NO (the default).

In the case of a class ne field (numeric edited value), the following parameters to be defined will be displayed on the line below INITIAL VALUE.

THOU. SEP.

Indicates whether the field is to be edited by adding the thousands separator.
Two function keys are enabled, giving the options YES and NO (the default).

SIGN

Indicates whether the field contains a value with or without the sign.
Two function keys are enabled, giving the options YES and NO (the default). If you select YES, the sign will be displayed to the right of the number.

FILLER

Indicates whether the first or second of the two filling characters defined in the Editing Menu is to be used.
Two function keys are enabled, giving the two options FILLER1 and FILLER2.

DEC. NO.

Indicates the number of decimal digits in the field. Enter 0 (default value), for no decimal digits.

EDITING

Indicates the type of keyboard to be used to enter the editing characters.
Three function keys are enabled, giving the following options:

EUROPEAN
(default value) to indicate the European keyboard where the period (.) is used to separate the thousands and the comma (,) as decimal point

ENGLISH
to indicate the English keyboard where the comma is used to separate the thousands and the period as decimal point

AMERICAN
to indicate the American keyboard which is the same as the English keyboard except that decimal numbers with no integer part, will not have a leading zero.

device=BG	If you have defined the Badge device, the following parameters will be displayed for definition.
SKIP	See the SK/PR device above; the default value is instead NO.
LENGTH ON DEVICE	Represents the length in bytes of the field as it is stored on the external BG device. If you have associated the SK/PR device with the field (amongst either the default or additional devices), this length must be the same as that of the field, otherwise the length must be edited.
TRACK	Specifies which of the two tracks is to be read/written. You can define only track 2 or track 3, depending on the Badge Reader used.
device=MS	If you have specified the Magnetic Stripe device, you must define the following parameters:
SKIP	See the BG device.
LENGTH ON DEVICE	Represents the length of the field in bytes as stored on the MS device. If the SK/PR device has also been associated to the field, this length is computed from the layout phase; if SK/PR has not been specified, the length must be entered.
device=KK	If you have selected the Keyboard Keys, the following parameters must be defined:
SKIP	See the BG device
LENGTH ON DEVICE	Represents the length in bytes of the field as stored on the KK external device. In this case however this length is decided automatically and set equal to 3; you cannot therefore change it.
device=CH	If you have selected the Cheque Reader or Cheque Writer device, you must define the following parameters:
SKIP	See the BG device

LENGTH ON DEVICE — Represents the length in bytes of the field as stored on the CH external device

MODE — Specifies which operating mode is to be used. The function keys have the following meanings:

VAR_STAMP	MARK	BACK_VAR_FIX	BACK_VAR	BACK_MARK	AL_MARK

where:

- VAR_STAMP writes on the back of a cheque
- MARK writes on the front of a cheque
- BACK_VAR_FIX puts the cheque in the writing position, writes on the back of it and stamps the front
- BACK_VAR puts the cheque in the writing position and writes on the back of it
- BACK_MARK puts the cheque in the writing position and writes on the front of it
- AL_MARK feeeds the next cheque and writes on the front of it.

POSITION — Specifies the position from which to start printing; enter a value in the range 0 to 127.

device=CA — If you have selected the Cash Adapter device, the following parameters must be defined:

LENGTH ON DEVICE — Represents the length in bytes of the field as stored on the CA device.
If the SK/PR device is also associated with the field, the length is computed from the layout phase; if SK/PR has not been specified, the length must be entered

DISPENSE — Indicates whether the CA device has to dispense the notes directly into the delivery throat, or to keep the notes in the stacker until a reject or deliver command is issued. Two function keys are enabled, giving the following options:

- YES to dispense directly

- NO to stack.

device=PP — If you have selected the Pin Pad device, you must specify the following parameters:

SKIP — See the SK/PR device

LENGTH ON DEVICE — Represents the length in bytes of the field as read from the PP external device. If the SK/PR device is also associated with the field, the length is computed from the layout phase; if SK/PR has not been specified, the length must be entered

LIGHTS — Indicates which leds are to be activated during the input phase. Four function keys are enabled, giving the following options:

- no leds

- the green led

- the red led

- both the green and red leds

MATCH — Indicates the type of pin-pad input termination. Two function keys are enabled, giving the following options:

- YES : the green key terminates the input and the red key clears it; the maximum input length is as defined in LENGTH ON DEVICE

- NO : exactly as many characters as defined in LENGTH ON DEVICE are accepted.

As already mentioned above, the device defined must be consistent with the MEM.CLASS specified.
If SK/PR has been specified, the type of MEM.CLASS and the value of the CLASS parameter are checked for compatibility.

The possible combinations are shown below:

MEM.CLASS : CLASS

ST	CD
ST	AL
ST	AN
ST	GR
UN	NE
UN	NC
TE	NE
TE	NC
TI	NE
TI	NC
LE	NE
LE	NC
LI	NE
LI	NC
PK	NE
PK	NC
BB	NE
BB	NC
BI	NE
BI	NC

Once you have filled in the entire menu, press /SAVE/. TFORM will check whether you have defined the field attributes correctly:

- if the check is negative, TFORM will display an appropriate message and re-propose the field concerned for correct attribute definition
- if the check is positive, two different cases may arise:
  - if the current field is not the last of the form, TFORM displays the attribute definition menu for the next field
  - if the current field is the last of the form, TFORM will display the subform structure menu.

SUBFORM STRUCTURE

When you have defined all the field attributes, you can start to define the structure of the subforms.

Subform Definition

A subform consists of a set of one or more fields of the form. You define which fields form part of a subform according to the order in which you want to execute them.

Each subform consists of at least one field of the form; the subforms are organized in a hierarchy, that is to say, they are nested.

In the subform structure, each subform is displayed with its level number, which determines its position within this hierarchy.

If the form contains labels, frames, visual attributes but no fields, note that the subform structure will be empty, therefore, this phase is omitted.

A subform that contains only one field is an "elementary" subform and is indicated only by the name of the field. A subform that contains another subform and/or several fields is called a "composite" subform and is identified by the name you assign during subform structure editing.

Below, reference is made to subforms in general, meaning both elementary and composite subforms.

You can decide to have V15A execute the entire form or the individual field, as long as these are identified by a subform.

A new subform can be created by joining one or more already existing and/or recently defined subforms, provided that these are physically adjacent in the subform structure.

Subform Definition Menu

The following menu is displayed:

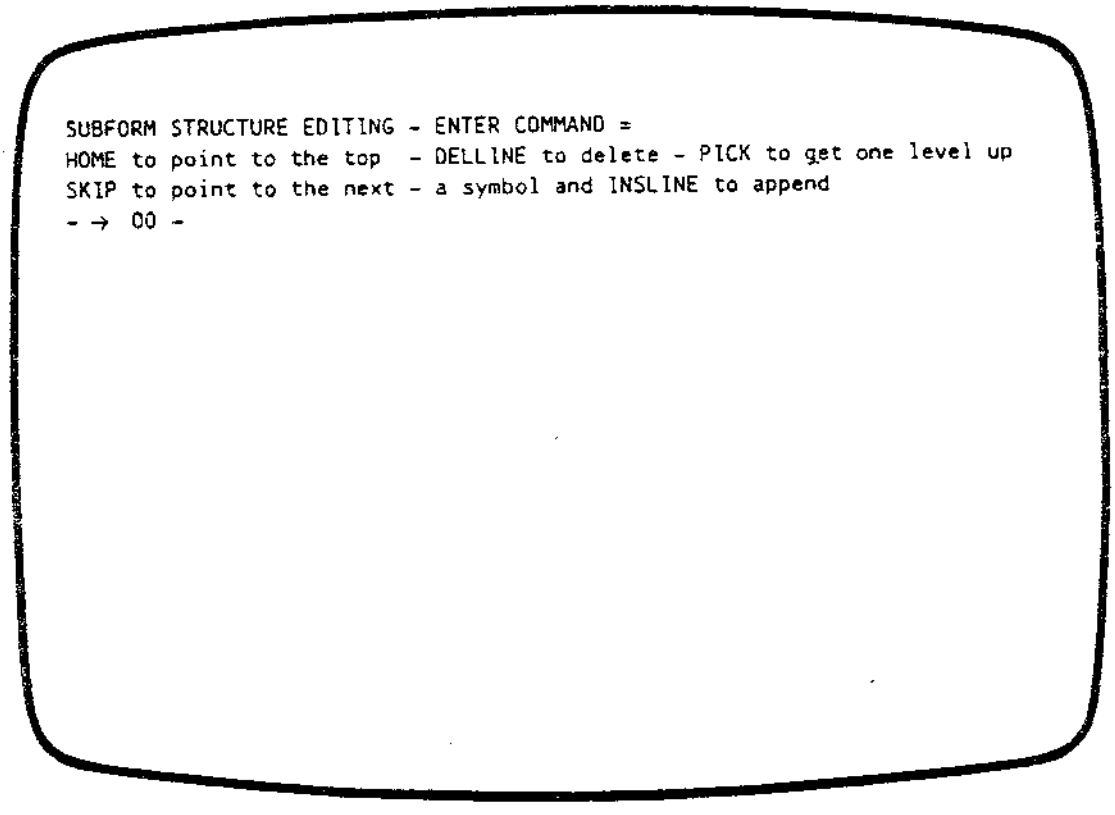


Fig. 3. 5 - Subform Definition Menu

When defining subforms, you work from the top down, scanning all the elementary and composite subforms one after the other.

You will be guided during compilation of the above menu, as follows:

- the cursor will be positioned on the first line of the menu as you must enter the name of the subform after ENTER COMMAND;

- the name of the subform must be at the most 12 alphanumeric characters, of which the first alphabetic or * followed by the /INSERT LINE/ key; also

  . if the name you enter refers to a subform that has already been identified in the structure, an error message will be displayed and nothing will happen;

  . if the name you enter starts with one or more blanks, it will be ignored and no error message will be displayed;

  . if the name you enter is that of a field, it is inserted in the subform structure together with the 'F' character in the first column before the subform level number, to indicate that that particular subform is of the elementary type;

  . if the name does not match any of the fields of the form, it is interpreted as a composite subform.

- you enter the name of the subform on the line below that indicated by the arrow, that is to say, after level number 00.
  Level 00 is only to indicate the start of the subform structure and not the subform which matches the form itself.
  Once you have entered the name of the subform, the arrow will move down one line to indicate the position in which you can enter another subform;

- as you enter each subform, this is nested inside the preceding subforms. All the subforms having the same level number are displayed starting from the same column.
  If you want to modify a level, press the /PICK/ key and the subform will be positioned a number of levels higher than that assigned automatically by TFORM;

- to delete the subform indicated by the current position, press the /DELETE LINE/ key; any elementary subform you have deleted can then be re-entered;

- use the /SKIP/ key to scan the subforms entered. The arrow will move down until it reaches the last subform displayed; if you press the /SKIP/ key at this point, the arrow will return to the

starting position, i.e. to the left of level number 00, to allow further editing;

- to move the arrow directly to the first level number of the structure, just press /HOME/.

Up to 20 subforms can be displayed on each screen page. If you wish to define more than 20 subforms, the rest will be displayed on the next screen page; press the /HOME/ key to return to the first 20 subforms defined.

As explained above, you can in theory define up to 96 fields in a form: as the field is an elementary subform, the same applies for the subforms to be entered, remembering however that the following algorithm is valid for subform structure:

n. elementary subforms + n. composite subforms<= 96

This is a theoretical limit as it is reached only when elementary subforms are defined.

If you define a 97th subform, a "BUFFER OVERFLOW" message will be displayed and TFORM will return to the layout phase.

Once you have finished editing your subforms, a number of checks are made to see whether you have forgotten an elementary subform (all the fields must be inserted in the structure).

To confirm the structure you have defined, press /SAVE/; if you press /EXIT/, the layout will be displayed.

Example

Given below is an example of how to define the subform structure of a given form.

Let us assume that a form of the following type has been created:

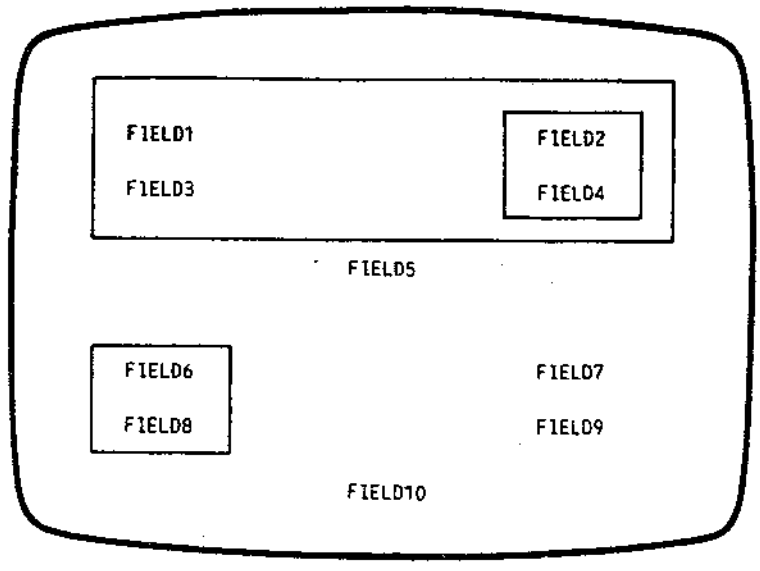


Fig. 3. 6 - Example of a Form

To make it easier to understand the example and how the form is represented, the name of the field is the same as the label that appears in the form and the subforms that this contains are rectangular frames, where the asterisks indicating the length of each field have been left out.
The subforms could have the following structure:

```
SUBFORM STRUCTURE EDITING - ENTER COMMAND =
HOME to point to the top - DELLINE to delete - PICK
to get one level up
SKIP to point to the next - a symbol and INSLINE to
append

      00 -
       .01 SUBFORM1
     F..02 FIELD1
       ..02 SUBFORM2
     F...03 FIELD2
     F...03 FIELD4
- - ->F...03 FIELD3
```

TFORM positions FIELD3 at level number 03 but as it is not to form part of SUBFORM2, you must move it up to a higher level using the /PICK/ key. The arrow indicates the current position on which you operate. The complete example is given below where the structure includes all the fields of the form.

```
SUBFORM STRUCTURE EDITING - ENTER COMMAND=
HOME to point to the top - DELLINE to delete - PICK
to get one level up
SKIP to point to the next - a symbol and INSLINE to
append

      00-
       .01 SUBFORM1
      F..02 FIELD1
       ..02 SUBFORM2
      F...03 FIELD2
      F...03 FIELD4
      F..02 FIELD3
      F.01 FIELD5
       .01 SUBFORM3
      F..02 FIELD6
      F..02 FIELD8
      F.01 FIELD7
      F.01 FIELD9
- -> F.01 FIELD10
```

List of Enabled Devices

Once you have confirmed the structure of the subforms, the menu of devices enabled for the form will be displayed.

This menu shows all the devices defined for the form and for all the fields.

Given below is an example of a menu which displays all the devices that have been defined: SK for the form, MS and BG for the fields.

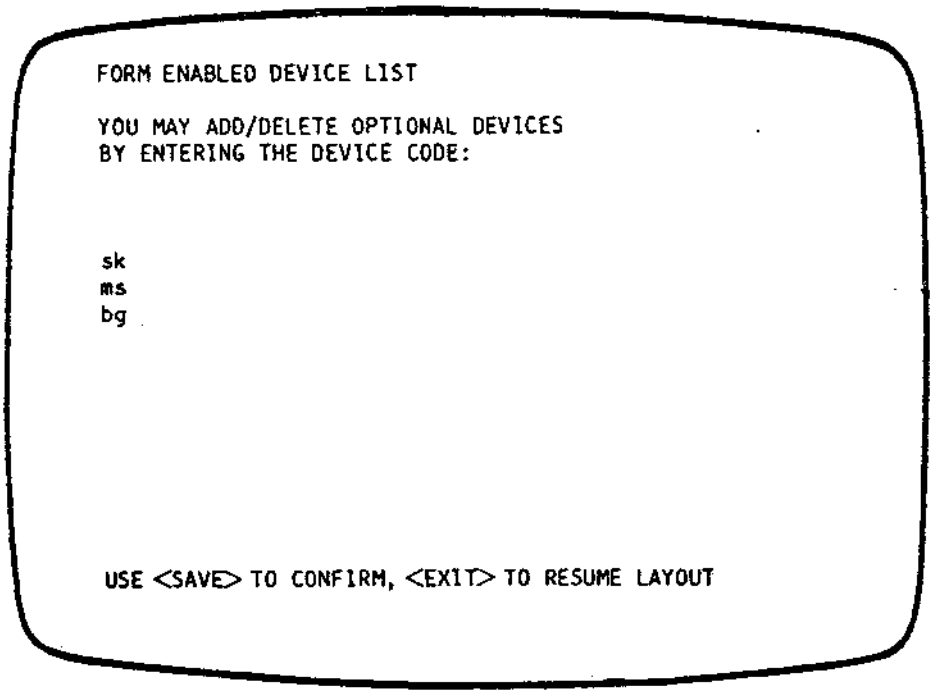


Fig. 3. 7 - Example of List of Enabled Devices for the Form

You can add and/or delete devices defined for the form by entering the code of the device and pressing /↵/.
Furthermore, you may specify DS which allows you to specify parameters (see below) for opening and closing work sessions on the printer and/or cash adapter.
You cannot change the devices specified for the fields.
To confirm the list of enabled devices, press /SAVE/; to return to the layout phase, press /EXIT/.

Device Parameters

For each of the devices defined for the form, you must fill in the matching menu, either confirming or modifying the default values of each parameter.
You confirm each set of values entered for a device by pressing /SAVE/.

screen-keyboard

The following menu is displayed for the SK device:

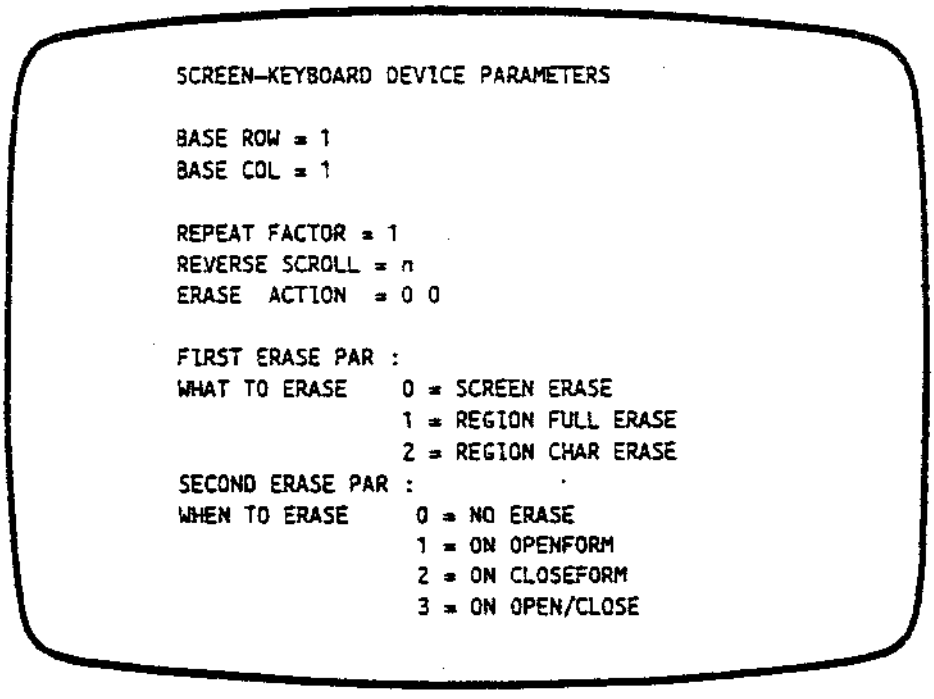


Fig. 3. 8 - SK Device Parameter Definition Menu

where:

BASE ROW, BASE COL

Indicate respectively the row and column number starting from which the form is displayed by VISA. In other words, row 1 and column 1 of the form are the same as the screen co-ordinates (BASE ROW, BASE COL) which have just been defined.
The default values are (1, 1).

REPEAT FACTOR

When the form is interpreted by VISA (see the VISA manual) this indicates how many forms there can be vertically on the screen after subsequent executions of the WRITEFM routine.
The default value is 1.
However, when defining this parameter, you must respect the following restriction:

BASE ROW + (REPEAT FACTOR x n. form lines) <= 256.

If this value is exceeded, an error message will be displayed and you must re-enter a different value for REPEAT FACTOR.

REVERSE SCROLL

Where the REPEAT FACTOR has been defined greater than 1, the contents of the screen will be scrolled down by the number of lines of the screen.
You can enter "y" or "n".
The default value is "n".

If REVERSE SCROLL = n the same restriction valid for the REPEAT FACTOR parameter, must be respected.

If REVERSE SCROLL = y then you must respect the following restriction:
BASE ROW - [(REPEAT FACTOR -1) x n. form lines] > 0
to ensure that the first line of the last form instance executed corresponds to the first line of the screen.

ERASE ACTION

This parameter determines which part of the screen will be erased and when it will be erased.
You must define two values:

- the first value indicates which part is to be erased: you may enter one of the following values:

  0 = to erase the entire screen (default value)

  1 = to erase the entire screen area defined by the size of the form multiplied by the REPEAT FACTOR, always starting from the base

  2 = to erase the labels and the value of the fields inside a screen area.

- the second value indicates when to erase. You may enter one of the following values:

  0 = to never erase (default value)

  1 = to erase when VISA executes the OPENFORM command

  2 = to erase when VISA executes the CLOSEFORM command

  3 = to erase when VISA executes either the OPEN or CLOSE command.

There are 12 possible combinations of erasing using the ERASE ACTION parameter of which 9 are effective, because if you assign 0 to the second parameter value no erase is made.

sprocket, automatic front feed, manual front feed

Given below is the menu displayed for the SP (sprocket) device; the same menu is used for the automatic front feed (AF) and manual front feed (MF) devices:

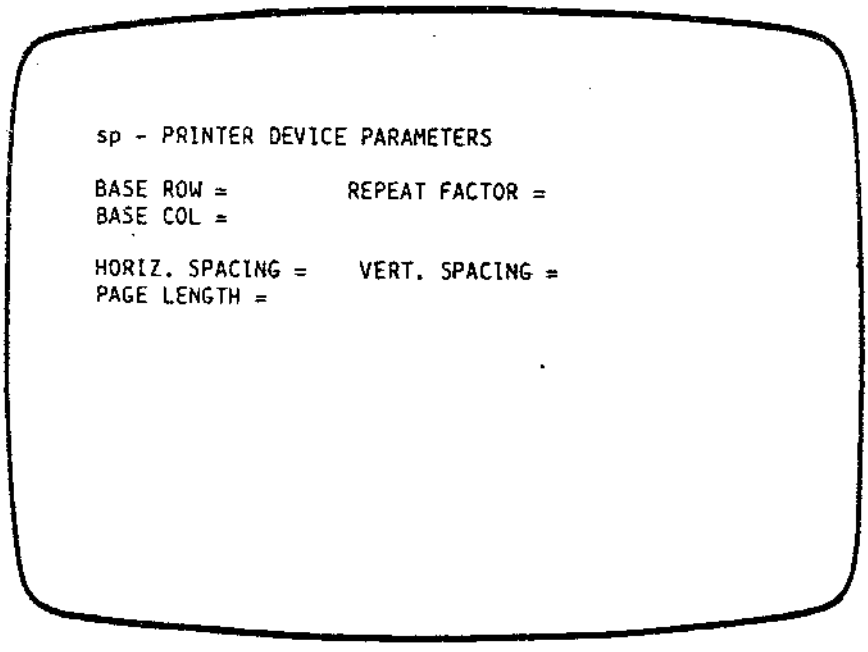


Fig. 3. 9 - SP Device Parameter Definition Menu

where:

BASE ROW, BASE COL, REPEAT FACTOR	Have the same meaning as seen above for the SK device (see above).
HORIZ. SPACING	This parameter can be used to modify the horizontal spacing on the printer. The possible values are 0 to 4. The default value is 0.
VERT. SPACING	This allows you to modify the vertical spacing i.e. the line feed on the printer. The possible values are 0 to 6. The default value is 0.
PAGE LENGTH	This indicates the length of the physical page i.e. the number of lines it contains. The default value is 70 lines.
whole platen, left platen, right platen	Given below is the menu displayed for the PL (whole platen) device; the same menu is used for the LP (left platen) and RP (right platen) devices:

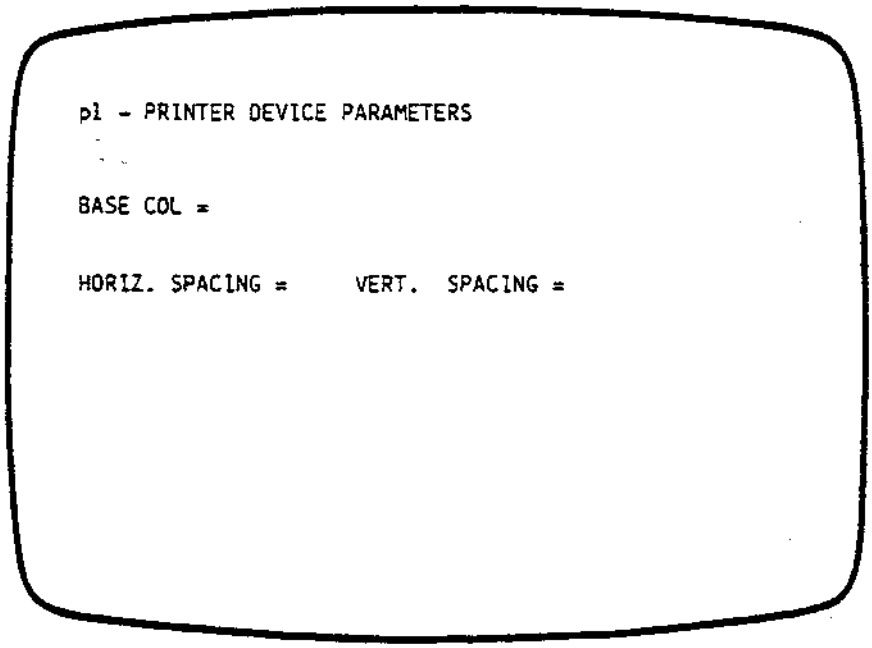


Fig. 3. 10 - PL Device Parameter Definition Menu

where:

BASE COL, HORIZ. SPACING, VERT. SPACING

These have the same meaning as explained for the PL device (see above).

device session parameters

Given below is the menu displayed for the device session parameters. This menu is only displayed when DS has been selected on the Enabled Devices Menu.

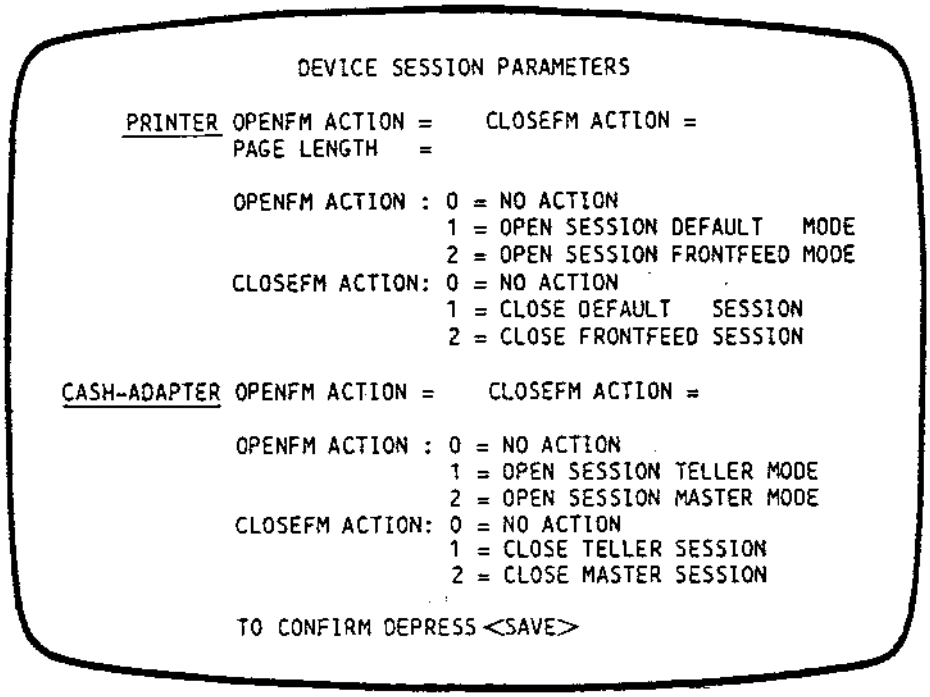


Fig. 3. 11 - Device Session Parameters Menu

The significance of the parameters is self-explanatory but refer to the manual "MOS-VISA Form Management Package Programmer Guide" for details. Note that PAGE LENGTH is only requested for a printer session if you have selected option 2 for OPENFM ACTION.

other devices

No parameters are requested.

FUNCTION KEYS DEFINITION

TFORM allows you to modify the meaning of the keys used for form execution at VISA time so that you can define their functions according to your needs.

If you have defined the SK device for the form, TFORM displays the present keyboard layout with the default values of the keys or with those defined during the last work session.
You can then confirm or modify these values.

Function Keys Definition Menu

Given below is the function keys definition menu where a "D" has been indicated in the place of the keys to show that they have been assigned the default value, that is to say, the standard function of the keyboard.

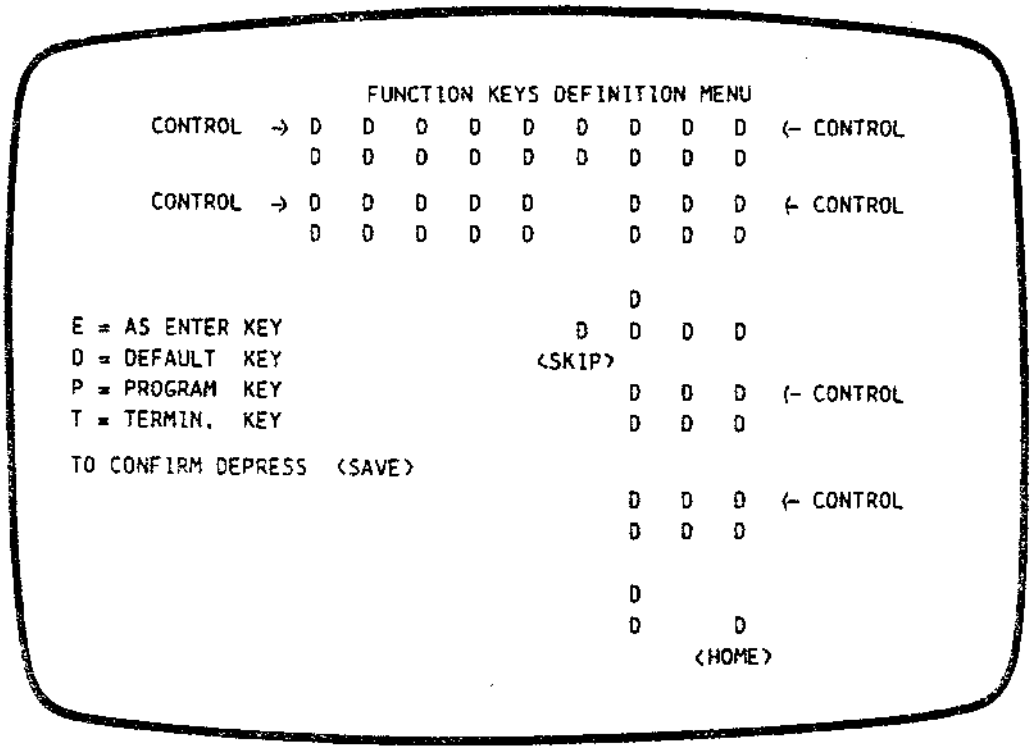


Fig. 3. 12 - Function Keys Definition Menu

The menu shows all the function keys, some of which are divided into two as their functions change when they are pressed together with the /CONTROL/ key. This menu is self-explanatory as regards changing the meaning of the keys; you position the cursor under the D symbol of a key and then enter one of the following characters, to assign that particular function to the key you are positioned on:

- E: as /ENTER/ key
- D: to keep the standard function
- P: acts as an interrupt during input at run-time
- T: used to explicitly close the input phase.

Refer to the VISA manual for further details of the functions of the keys.

You can confirm the menu by pressing /SAVE/ or you can press /EXIT/ to return to the layout definition phase.

PROGRAM VALIDATION DEFINITION

If you have requested any validation (BEFORE and/or AFTER) of at least one field of the form when defining field attributes, TFORM will now display the following menu:

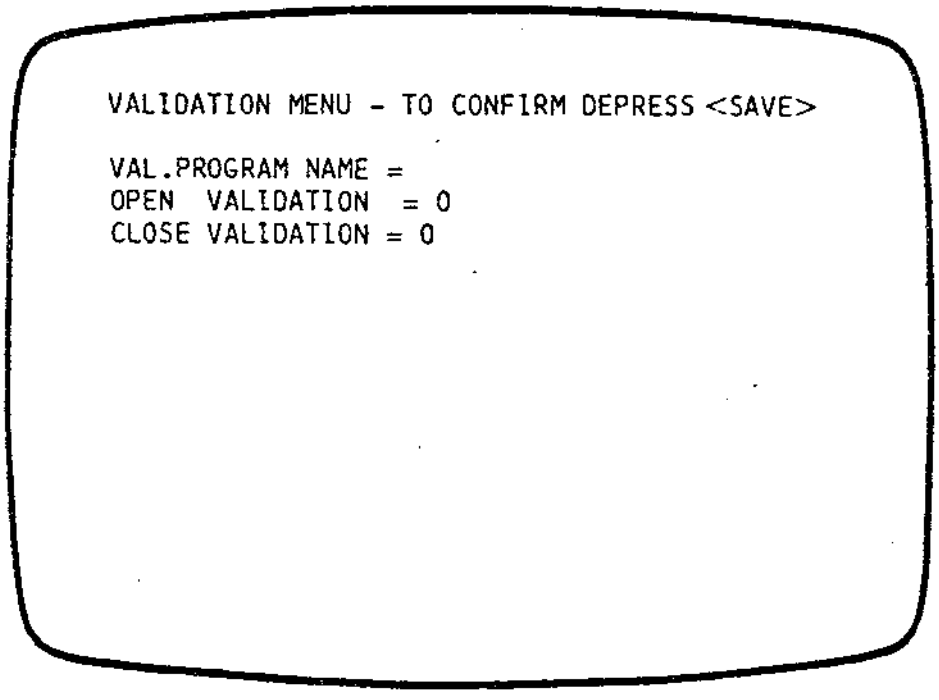


Fig. 3. 13 - Validation Definition Menu

where:

OPEN VALIDATION — Is the line number where the open validation routine starts.
VISA will execute this routine when the OPENFORM command is executed.

If you confirm the default value, 0, no open validation process will be executed.

CLOSE VALIDATION — Is the line number where the close validation routine starts.
VISA will execute this routine when the CLOSEFORM command is executed.
If you confirm the default value, 0, no close validation process will be executed.

VAL. PROGRAM NAME — Is the name of the Validation Program which will be used to validate the entire form. You must enter a name of up to 12 alphanumeric characters of which the first must be alphabetic.

When you have finished filling in the menu, you confirm the values entered with the /SAVE/ key and control is passed to the VPL environment (described in the second part of the manual).
You must write the specified Validation Program using the Validation Language statements.

Once you have written the Validation Program, control is returned to TFORM which displays the following menu:

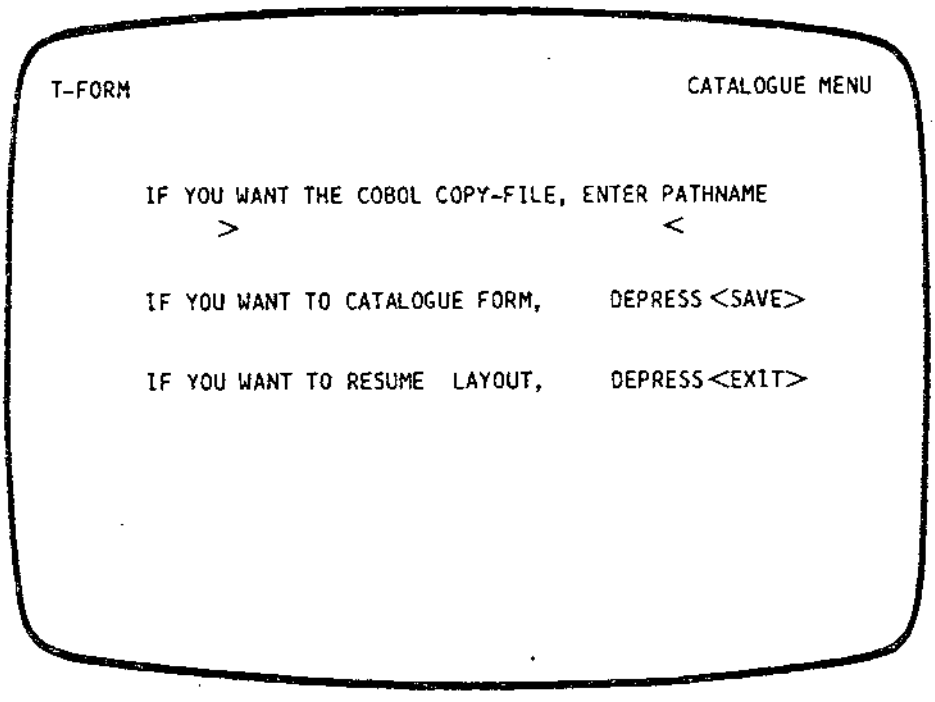


Fig. 3. 14 - Cataloguing Menu

You may specify the pathname of a text-file which will be created (or overwritten if it exists) containing a "COBOL DESCRIPTOR" for the form data record. This descriptor may be COPY'ed within each COBOL source program which accesses the form using VISA calls and the COBOL programmer need not be concerned about correct definition of the form data record.

If you press /SAVE/, TFORM will catalogue the form in the current library (and create the COBOL DESCRIPTOR if a pathname has been specified), and then display the Editing Menu again.

If you press /EXIT/, TFORM will return to the layout definition phase and you can modify the form as described in the next chapter.

4. FORM MODIFICATION

A form can be modified during any of phases described above. To make this operation perfectly clear, it is described in two separate paragraphs:

- modification during form creation

- modification of a catalogued form.

To enter the form modification phase, you must:

- press /EXIT/ to return to the layout phase

- press /SAVE/ the number of times required to reach the modification phase.

Note that in the case of a form already stored in a library, when you catalogue your new form, it will take the place of the original form.

Therefore, if you don't want to lose the original form, you must first of all make a copy of this using the appropriate utility program, run under SHELL.

In the pages that follow, implicit reference is made to figures 2.1 through 2.5: "TFORM Functions Logical Scheme".

MODIFICATION DURING FORM CREATION

Layout Modification

You can modify the layout of your form either during the actual layout definition phase or during any of the subsequent phases.

during the layout phase

Using the following keys:

/ ← / / → / / ↑ / / ↓ / /ENTER/ / ↵ /

you move the cursor around the screen to modify the frame, labels, fields and visual attributes in the same way as during initial definition of these elements (see LAYOUT DEFINITION in the chapter on "FORM CREATION").

For form layout modification, TFORM provides a number of editing features based on the use of the following function keys:

/1C/ this is used to put TFORM into "character insertion status", during which the following applies:

- for each character entered from keyboard: the cursor, the character indicated by the cursor and all the following characters are moved 1 position to the right; the character you have entered will occupy the position immediately preceding the cursor.

- if the characters you have inserted move the last character of the line to the last screen column (or the last column of your form, where this is less than 80 columns wide), TFORM will emit a bell signal to indicate that you cannot continue insertion.

  Therefore, if your form is wider than the actual width of the screen, you cannot insert characters in the part of the line that is not displayed.

- press /1C/ once again to exit from character insertion status.

/DC/ this is used to delete the character indicated by the cursor; the cursor will not move, the characters to the right of the cursor are moved 1 position to the left and a "blank" will be inserted in the position that was occupied by the last character of the line.
The /DC/ key, in the same way as the /IC/ acts only on the part of the line shown on the screen; therefore, if your form is more than 80 columns wide, the characters after column 80 will not be moved.

/IL/ this key inserts a new line in the form.
When you press /IL/:

- the contents of the lines starting from and including the line indicated by the cursor will be moved down by one line

- the contents of the last line of the form will be lost

- the cursor will not move and remains therefore pointing to the new line you have inserted and which contains "blanks".

Any frame characters and the visual attributes are not moved.

It is important to note that the /IL/ key differs from the /IC/ and /DC/ keys in that it acts on the entire form and not just the part displayed. So if your form measures 40 lines by 100 columns and you have positioned the cursor under line 10 and pressed /IL/:

- the contents of all the lines of the form starting from and including line 10 are moved on line down

- the contents of line 40 are lost

- line 10 contains "blanks" from column 1 through 100

- the previous contents of line 24 are no longer displayed (this is now line 25).

/DL/ this key deletes a line of the form.
If you press /DL/:

- the contents of the line under which you have positioned the cursor are deleted

- the contents of the lines following the deleted line are moved 1 line up

- the last line of the form contains "blanks".

Any frame characters and visual attributes are not moved.

The /DL/ key is similar to the /IL/ key in that it acts upon the entire form and not just the part displayed.

/PUT/ this key is used to copy the contents of one screen sector into another screen sector as follows:

- you define the screen sector to be copied by defining two points, as seen when drawing the frame of the form

- you move the cursor to the first point at which you want to copy the sector. There is no need to specify the second point as TFORM assumes that the two sectors are of the same size

- the original sector may partially overlay the destination sector

- the following are also transferred during this operation:

  . labels

  . the '*' characters defining the fields.

  The following are not transferred:

  . the field attributes, if these have already been defined for the fields being copied; this means that you will have to redefine the attributes of the fields copied.
  This situation may arise when you have interrupted the field attribute definition phase during form creation and when modifying the layout of an already catalogued form.
  (Both these situations are described later in this chapter).

  . any visual attributes

  . any frame sections.

- the contents of the original sector remain unchanged.
  Any contents of the destination sector are lost.

- if you try to transfer a sector that will not fit into the other section, the "PUT OR MOVE OUT OF FORM SIZE" message will inform you that the copy cannot be made.

Have a look at the following figure:

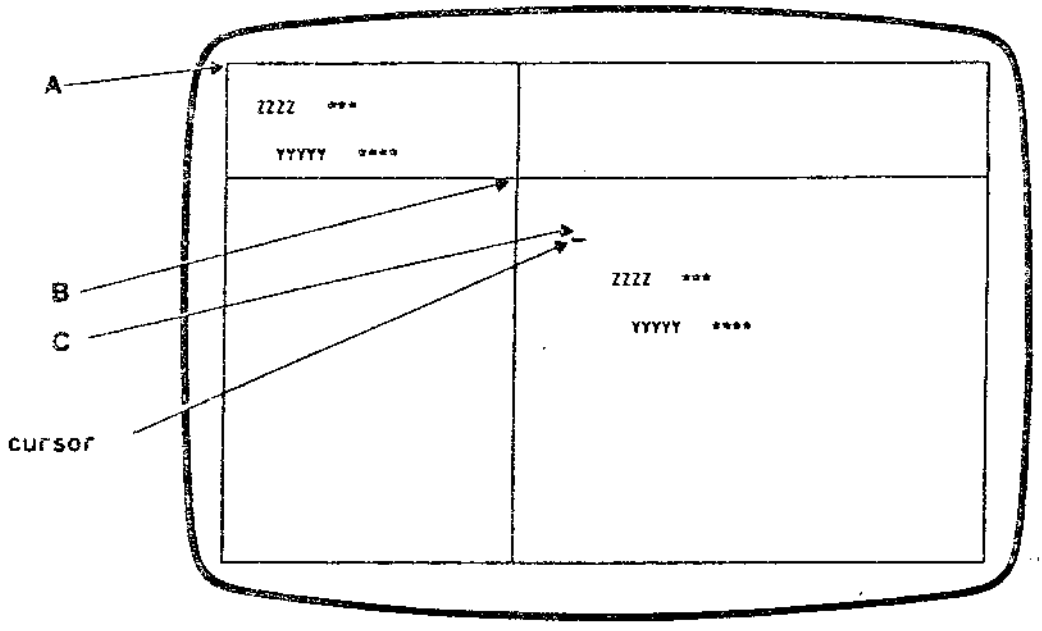


Fig. 4. 1 - Copy Operation

In this example, the sector identified by the two points A and B has been copied starting from the point identified by C.

Points A and B were defined by positioning the cursor and pressing /PICK/, while C was defined by moving the cursor and then pressing the /PUT/ key.

/MOVE/ this key is used to move one sector of the screen into another sector.
It has the same effect as when making a copy except that at the end of the operation, the original sector contains "blanks". Except for this, the rules to be followed are the same.

Once you have finished modifying that part of the layout already defined, you return the cursor to the original position and continue definition of the layout itself.

At any time during layout definition (or modification), you can recover the situation that existed before the last time you pressed one of the following keys:

- /IL/
- /DL/
- /PUT/
- /MOVE/

by pressing the /REPLACE/ key.
If you have not used any of these keys, the situation existing at the start of the layout phase will be recovered.

during the attribute definition phase

To modify form layout when TFORM is in the attribute definition phase, you must:

- press /EXIT/ to interrupt the attribute definition phase and return TFORM to the layout definition phase
- make the necessary modifications, using the editing features described above, if required
- press /SAVE/ to re-enter the attribute definition phase.

At this point, TFORM checks the modifications you have made to the layout and:

- if the modifications made do not require redisplaying of the field attributes of the fields on the lines modified, for modification (or confirmation), such as:
  - modifications only to the frame of the form and/or of the visual attributes
  - moving of the lines of a form with the /IL/ and /DL/ keys.

  TFORM continues form creation from the point at which it was interrupted

- if the modifications are not of the above type (for example, fields have been added or deleted or their length or position in the form has been modified), TFORM asks you to redefine (or confirm) the attributes for all the fields of the line modified (see below 'redefinition'). Then, TFORM continues form creation from the point at which it was interrupted.

Field Attribute Modification

Field attributes are modified in two distinct logical moments:

- you select the fields whose attributes are to be modified during the layout definition phase
- you actually modify the attributes of the previously selected fields during the attribute definition phase.

To modify the attributes of already defined fields, proceed as follows:

- if TFORM is in the attribute definition phase, press /EXIT/ to enter the layout definition phase
- TFORM displays the form layouts and you select the fields whose attributes are to be modified by positioning the cursor under the first position of the field and pressing /CHANGE/.
  You then press /SAVE/ to go to the attribute definition phase
- TFORM will display the previously selected fields in order.
  For each field, TFORM displays the attribute definition menu with the already defined values and you can make the required modifications, pressing /SAVE/ to go on to the next field
- once you have modified the field attributes, TFORM continues form creation from where it was interrupted.

redefinition

Redefinition of field attributes because of layout modifications is described below.

For each line modified, TFORM handles:

- the attributes already defined for the fields of the line before the modification (old line). In the description below, A1 indicates the attributes of the first field, A2 the attributes of the second field.... and Am those of the last field.

- the fields of the modified line (new line) indicated, in order, by C1, C2,..... Cn (note that $m \leqslant n$).

TFORM displays in order the fields of the new line, proposing an initial combination with the attributes defined for the fields of the old line.

This process follows the rules given below:

- for field C1, TFORM displays the attribute definition menu containing the values A1; you can choose one of the following alternatives:

  1. press /SAVE/, to confirm the combination without modifying any value of A1. The A1 attributes are removed from the list of available attributes and they will not be used for any successive combinations.

  2. modify the value of one, or more, of the A1 attributes (you can modify them all, if required) and press /SAVE/. Also in this case, the combination is considered as confirmed if the A1 attributes are removed from the list of available attributes.

  3. enter the name of a field of the old line and press /CHANGE/. TFORM matches C1 with the attributes of the field indicated and you can proceed as indicated in points 1, 2 or 3.

  4. press /CHANGE/. TFORM matches C1 with A2 which is the next set of available attributes and you can proceed as indicated in points 1, 2, 3 or 4. Note that if you press /CHANGE/ again, TFORM matches C1 and A3 and so on until all the

available attributes have been used.
In this case, therefore, the combination C1 A4 is proposed, and when this is rejected (/CHANGE/ key), TFORM displays a blank attribute definition menu for C' (with only the default values) and you must define the relative values.

- the process described above is repeated for the following fields: C2, C3 ....Cn using only those attributes that have not yet been combined. With particular reference to point 3, note that you cannot indicate the name of a field whose attributes have already been combined, as these are no longer available.

- for each line modified, any sets of attributes that are not combined are ignored and this information is lost when the form is catalogued.

Example Given below is a form line containing four fields:

```
A (A1)    B(A2)    C(A3)    D(A4)
 ****      **       ***      *****
```

where:

A, B, C, D identify the fields
A1, A2, A3, A4 indicate their respective attributes.

This line is modified as follows:

```
  A      E       D       C       F
****    ***    *****    ***    ****
```

where:

'A' keeps the previous attributes (A1)
'E' is a new field to be defined
'D' keeps the previous attributes (A4)
'C' keeps part of the former attributes (A3)
'F' is a new field which is defined with the same attributes as 'A'.

TFORM starts the following process:

- 'A' is matched with A1;
  press /SAVE/ to confirm this and the A1 attributes are no longer available for subsequent combinations.

- 'E' is matched with A2: press /CHANGE/;
  'E' is matched with A3: press /CHANGE/;
  'E' is matched with A4: press /CHANGE/:

  When all the available attributes have been used, TFORM displays a blank attribute definition menu; you must define the values and press /SAVE/.

- 'D' is matched with A2; to perform the operations faster, you can enter 'D' and then press /CHANGE/.
  'D' is matched with A4 and you press /SAVE/ to confirm this (the A4 attributes are no longer available).

- 'C' is matched with A2: press /CHANGE/;
  'C' is matched with A3:
  modify the A3 values accordingly and press /SAVE/ for confirmation (the A3 attributes are no longer available).

- 'F' is matched with A2: press /CHANGE/;
  TFORM displays a blank attribute definition menu; define the appropriate values and press /SAVE/.
  Note that, even if you want to assign A1 attributes to field F, you cannot press 'A' and then /CHANGE/ as the A1 attributes have already been matched.

- the A2 attributes, which have not been used, will be lost when the form is catalogued.

Subform Structure Modification

During the actual subform structure definition phase, you can use the following keys:

/HOME/ : to return directly to the top of the subform structure

/DL/ : to delete a subform

/PICK/ : to increase the level number of a subform

/IL/ : to introduce a new subform in the structure.

Device Parameter Modification

You can modify these by entering the code of the device to be added and/or removed in the menu showing the list of enabled devices for the form. You can also redefine the values of each parameter of each device.

Function Keys Modification	The functions assigned to each key can be modified, even if they have just been defined, using the symbols described in the menu itself.
Modification of the Validation Menu	You can modify the line numbers of the OPEN VALIDATION and CLOSE VALIDATION parameters in relation to the indicated Validation Program.

MODIFICATION OF A CATALOGUED FORM

To modify an existing form, i.e. a form that has been catalogued, TFORM reads the form from the library, loads it in the memory and displays it.

Editing Menu Modification

You can modify any of the parameters of the Editing Menu of an already existing form, except that of the form name.

The only exception to this is the COLOR parameter, which cannot always be modified as required.

If you have defined COLOR = "blank", you can redefine this with "y" or "n".

If you have defined COLOR = n (i.e. 6 attributes enabled), this can be changed to "y", to extend the range of attributes from 6 to 15, obviously in the case of a colour screen.

If you have defined COLOR = y, this can no longer be modified, i.e. you cannot change it to "n", as this would reduce the range of enabled attributes (from 15 to 6), when all of these have been enabled.
In this case, if the number of colours is to be modified, you must act directly on the layout and not on the COLOR parameter.

Layout Modification

Given below are the operations performed to modify the layout of a form catalogued in a library:

- select option 1 (EDITING) of the Basic Menu and then enter the name of the form in reply to the FORM NAME parameter of the Editing Menu
- TFORM will display the following message:
  MODE IS MODIFICATION OF AN EXISTING FORM
  followed by the subsequent parameters of the Editing Menu with the existing values
- you can modify the values of the parameters required and then press /SAVE/
- TFORM enters the layout definition phase and displays the layout of the form required

- you can now make the required modifications using, where necessary, the editing facilities described above and then press /SAVE/

- TFORM enters the attribute definition phase and checks the corrections made to the layout:

  . if these corrections do not require redefinition of the field attributes of the modified lines, TFORM displays the subform structure.
  As already explained above, modifications of this type include changes to only the frame of the form and/or visual attributes or moving of form lines using the /IL/, /DL/ keys.

  . if the corrections are not of the type mentioned above, TFORM asks you to redefine (or confirm) the attributes for all the fields of the lines modified (see 'redefinition' in this chapter).
  The form is then displayed for definition of the new subform structure.

Field Attribute Modification

Given below are the operations to be performed to modify the field attributes of a catalogued form:

- select option 1 (EDITING) of the Basic Menu, then enter the name of the form in reply to the FORM NAME parameter of the Editing Menu

- TFORM displays the message:
  MODE IS MODIFICATION OF AN EXISTING FORM
  followed by the subsequent parameters of the Editing Menu with the existing values

- you can now modify, where required, the values of these parameters and then press /SAVE/

- TFORM enters the layout definition phase and displays the layout of the form required

- select the fields whose attributes are to be modified by positioning the cursor under any position of the field and press /CHANGE/.
  Press /SAVE/ to go on to the attribute definition phase

- TFORM displays the previously selected fields in order.
  For each field, TFORM displays the Attribute Definition Menu with the already defined values; you can now make the required modifications and press /SAVE/ to go on to the next field

- when the field attributes have been modified, TFORM displays the subform structure menu.

Subform Structure Modification	The structure of the subforms must be modified when one or more fields have been added to the form or modified. When the field attributes have been defined the indented list of the previously defined subforms is displayed. Therefore, you must position the new fields as described in the creation phase.
Device Parameter Modification	Refer to the creation phase for a description of how to modify the list of enabled devices for the form and the relative parameters.
Function Keys Modification	These are modified in the same way as for a form being created.
Validation Menu Modification	Validation of a catalogued form can be modified assigning new values to the parameters of the validation menu and assigning a new name to the Validation Program which will validate the modified form.

5. FORM TESTING

This chapter describes the TFORM testing function which allows you to test forms catalogued under the current directory.

TFORM tests the forms by executing a user-supplied sequence of VISA calls which simulates a program using the forms.

As well as specifying the sequence of VISA calls, you must specify the parameters (form name, timeout, data size, etc.) for each VISA routine used in the sequence.

The TFORM testing function also allows you to display and edit FORM DATA RECORDs which are automatically created in user-defined buffers. These records show, in hexadecimal format, the data used during execution of the sequence of VISA calls.

TESTING MENU

When you select option 4 (TESTING) of the Basic Menu, TFORM displays the following menu:

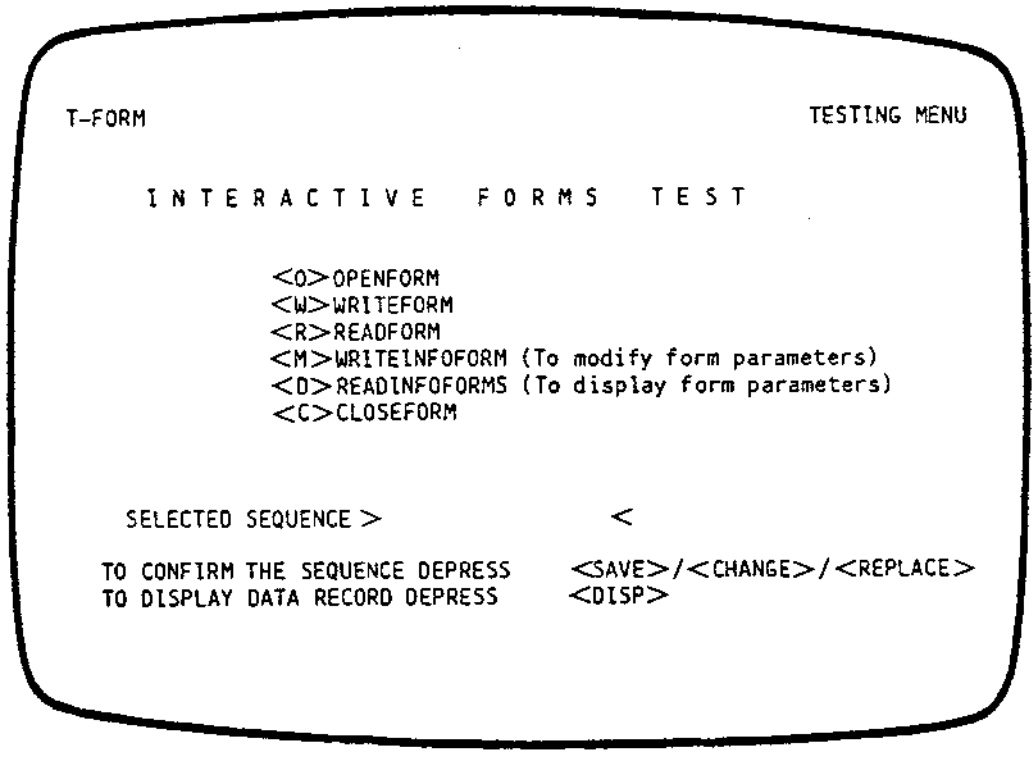


Fig. 5. 1 - Testing Menu

Your action now depends on whether or not the Testing function is being activated for the first time in the current TFORM work session.

If it is the first activation, you must enter the required sequence of VISA calls by typing the appropriate identifiers without any embedded spaces. You must then press one of the following keys:

- /REPLACE/, /CHANGE/ or /SAVE/; in each of these cases TFORM then requests the parameters for each VISA call (see below) and then executes the sequence.

- /EXIT/; in this case TFORM returns to the Basic Menu.

If it is not the first activation, TFORM displays the previously selected sequence. You then have the following options available:

- pressing /SAVE/, in which case TFORM executes the previous sequence using the previously defined set of parameters.

- pressing /REPLACE/, in which case TFORM asks for new parameters for each VISA call, ignoring the old ones, and then executes the previous.

- pressing /CHANGE/, in which case TFORM asks for new parameters for each VISA call, proposing those defined for the previous sequence as default, and then executes the previous sequence.

- entering a new sequence of calls and pressing /CHANGE/ or /REPLACE/. TFORM then asks for the new parameters and executes the sequence. If you press /SAVE/ after entering the sequence, TFORM displays the following message and waits for you to press another key:

  /SAVE/ INCONSISTENT WITH A MODIFIED SEQUENCE

- pressing /DISP/ to display a FORM DATA RECORD created during execution of a previous sequence. TFORM then asks for the name of the desired record (i.e. the user-defined buffer name) and displays it. You can modify the record using the HEXED editor; the commands are described later in this chapter.

- pressing /EXIT/ in which case TFORM returns to the Basic Menu.

The only check made by TFORM on the validity of your selected sequence is whether the number of CLOSEFORM calls is less than the number of OPENFORM calls. If so, the following message is displayed:

INCORRECT SEQUENCE - CLOSEFORM CALLS MISSING

and TFORM returns to the Testing Menu.

Any other sequence control is left to VISA. Note that the CONNECT and DISCONNECT calls are automatically activated by TFORM.

If any VISA call returns a reply other than FORMCORRECT, the reply is displayed on the last line of the screen. If this reply is FORMENDINPUT or FORMOPERINT, execution of the sequence continues. For any other reply, execution of the sequence will be aborted.

On completion of the sequence, TFORM displays the Testing Menu.

Parameter Specification

After you have selected the sequence, TFORM asks for parameters for each VISA call according to the COBOL-VISA interface, and definition of the buffer(s) for the FORM DATA RECORD(S). You can press /EXIT/ during this stage to return to the Testing Menu.

Definition of the data buffers is requested first. You must specify the number of buffers required and, for each buffer, its name and length in bytes and whether or not you want to initialise it. If you answer Y to the EDITBUF prompt, TFORM will display the buffer, allowing you to initialise it using the HEXED editor.

After defining the first buffer (when more than one has been specified), TFORM displays the previously defined characteristics for subsequent buffers. Buffer names must be unique so you must enter a new name, although the length and EDITBUF option may be left as they are. If you do not change the buffer name, TFORM displays an error message requesting new input.
TFORM creates the buffers in the current working directory.

The parameters for each VISA call are then requested. In this manual, only the names of the required parameters are given; refer to the 'MOS-VISA Form Management Package Programmer Guide' for details about the parameters. The requested parameters are:

```
OPENFORM  : FORMNAME
            TIMEOUT

CLOSEFORM : FORMNAME
```

WRITEFORM : FORMNAME
SUBFORMNAME
FORMDATA (this must be the name of a previously defined buffer)
DATASIZE
WRITEMODE
WRITEMASK
ACTION

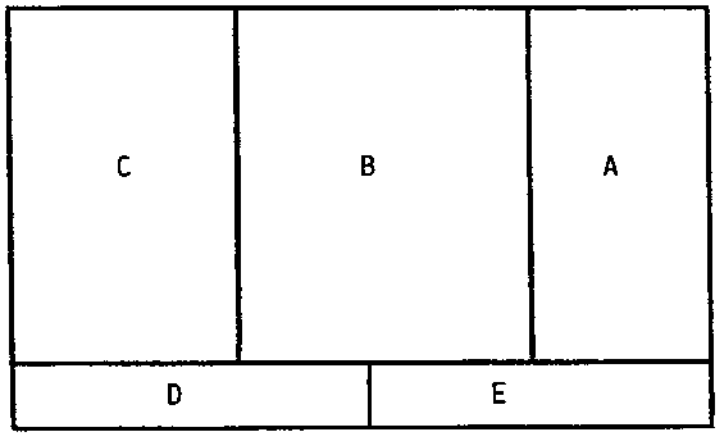
READFORM : FORMNAME
SUBFORMNAME
FORMDATA (this must be the name of a previously defined buffer)
DATASIZE

WRITEINFOFORM : FORMNAME
SUBFORMNAME
INFOREQUEST
DEVICE
PARAM1
PARAM2
PARAM3
PARAM4

READINFOFORM : not yet implemented

The HEXED Editor

The HEXED editor allows you to display and modify the FORM DATA RECORDS created during execution of a sequence of VISA calls.
Before describing the available commands, the screen layout is explained. The screen is divided into five windows as shown below.



where:

each line of A contains 16 bytes of information in ASCII representation

each line of B contains the information of the corresponding line in A in hexadecimal format

each line of C contains the hexadecimal address of the first byte of the corresponding line in A

D contains the name and size of the working FORM DATA RECORD

E contains HEXED messages to the user.

In total, 256 bytes are represented on the screen.

Commands

The HEXED commands comprise single keys from the alphabetic keyboard and occasionally parameters. There are four categories of commands:

General commands

/?/ HELP; it displays the list of available commands

/q/ EXIT; it returns you to the Testing function

Cursor commands

/h/ MOVE LEFT; it moves the cursor one position to the left

/j/ MOVE DOWN; it moves the cursor one position down

/k/ MOVE UP; it moves the cursor one position up

/l/ MOVE RIGHT; it moves the cursor one position to the right

These commands are only available within the current page.

Paging commands

/b/ BACKWARD; goes back 256 bytes, i.e. to the previous page. If this command is issued on the first page, no action is performed

/f/ FORWARD; goes forward 256 bytes, i.e. to the next page. If this command is issued on the last page, the first page is displayed

/G/ JUMP; jumps to the end of the file, displaying the last page

/g/ JUMP ADDRESS; jumps back or ahead to the specified address. The syntax is:

/g/ <hex address> /CR/

If no address is specified or the address is out of bounds, address 0 is assumed. If the input characters are not hexadecimal, an error message is displayed in the bottom right hand window

Replace commands

/r/ REPLACE; replaces the byte at the current cursor position with the specified value. The syntax is:

/r/ <hex value>

The new ASCII value of the replaced byte is also displayed in window A. If the specified value is not within the hexadecimal range, no action is performed and a bell signal is sent.

/R/ CONTINUOUS REPLACE; replaces as many bytes as there are values specified after the command, starting with the byte at which the cursor is positioned. The replacement continues onto the next line or page if necessary. Note that if the end of the file is reached before all the specified values have been replaced, a blank page is displayed and the remaining values are ignored. The syntax is:

/R/ <hex value> .....<hex value> /CR/

The new ASCII value of each replaced byte is also displayed in window A. If any value is not within hexadecimal range, the cursor position is not changed and a bell signal is sent.

6. LINKING FORMS

This chapter describes how to create 'Environment Libraries' to be used by VISA by means of the LINKFORMS option of TFORM.
Before describing the operational aspects of linking forms, the concept of the environment libraries is introduced.

Environment Libraries

The 'Environment Libraries' are the form and validation program libraries that can be created using a source library (editing library) which has been generated with TFORM.
Each environment library is created by executing the LINKFORMS option of TFORM and contains the same forms and validation programs as the source library. The forms are, however, divided into two user-defined groups:

- forms which are loaded into VISA memory when VISA is connected to the library, independently of requests for execution made by the application program.
  These forms make up a single load module with the fixed name $RESIDENT and remain in memory during use of the library.

- forms which are loaded into VISA memory when they are executed and which remain in memory only during this time.
  Each of these forms is a single load module, like any other source library form.

Clearly, the forms of the $RESIDENT module are not read from the library when they are executed, but there is an increase in the permanently occupied memory area.

The validation programs which exist in the source library are simply copied into the environment library.
Note that validation programs used by resident forms are not permanently resident but are loaded into VISA memory at form run-time.

VISA can connect to any of these libraries (source or environment) as shown in the following figure.

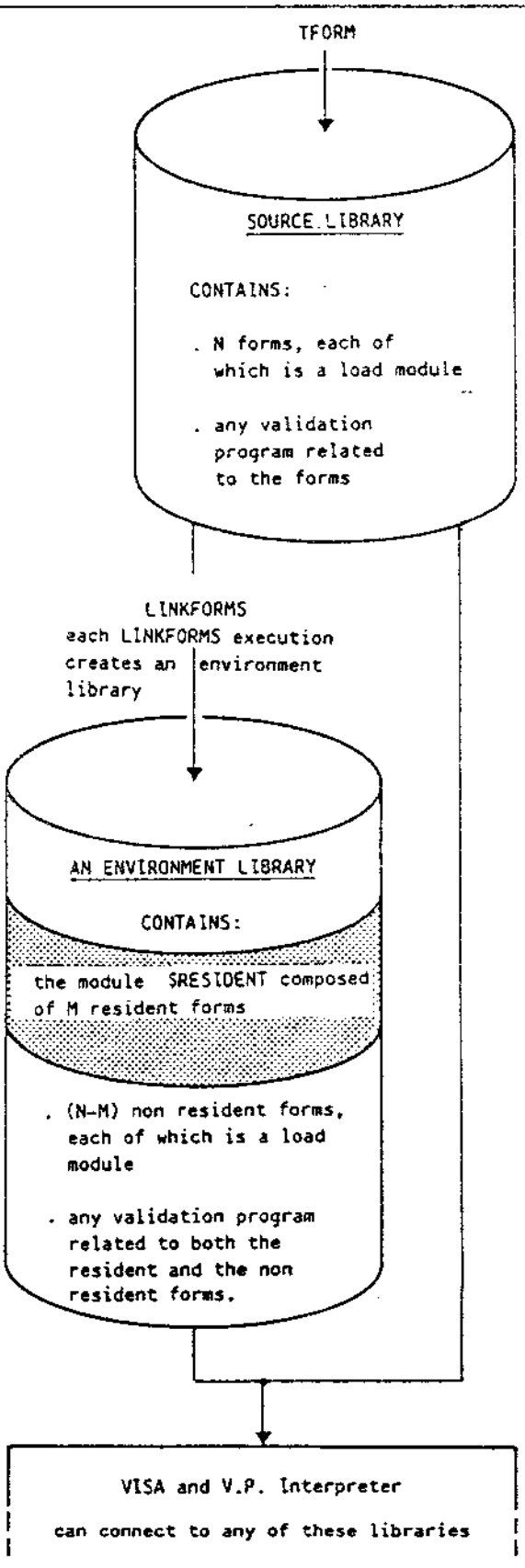


Fig. 6. 1 - Form Libraries Accessed by VISA

LINKFORMS EXECUTION

The LINKFORMS option of TFORM reads a source library and produces an environment library containing:

- the $RESIDENT module, consisting of all the source library forms to be held permanently in VISA memory when VISA connects to the library; the user must supply the names of these forms
- the remaining (non-resident) forms of the source library, each as a single load module
- any validation programs used by the forms (both resident and non-resident) of the library.

The LINKFORMS phase has two operating modes:

- creation of a new environment library
- appending of new forms to the $RESIDENT module of an existing environment library.

The LINKFORMS phase is invoked by selecting option 5 from the basic menu. You are then asked to enter:

- the name of the destination directory
- an option, which can be one of the following:

  Ø : creation of a new environment library. In this case you must supply the names of all forms which are to be included in the $RESIDENT module, i.e. made resident

  W : creation of a new environment library in which all the source forms are automatically included in the $RESIDENT module

  A : addition of forms to an existing environment library. In this case you must specify the names of the forms to be added to the $RESIDENT module. Note that forms can be repeated, VISA will always use the first occurrence of the form.

The destination library cannot be set to the current

working directory and, if creating a library (options Ø and W), this directory must initially be empty. If it is not, the following message is displayed:

DIRECTORY NOT EMPTY : WANT TO CLEAR (Y/N)?

If you enter 'N', execution of Link Forms is aborted and the TFORM basic menu is displayed.

If you enter 'Y', the directory is cleared and execution of Link Forms continues.

The following message is then displayed:

ENTER NUMBER OF CONTEMPORANEOUSLY OPEN FORMS:

You must enter the maximum number of resident forms that can be opened concurrently by an application program. This value is interpreted at library level; since several application programs can use the same library concurrently, you must enter the highest value used.

If you have specified option 'W', the LINKFORMS phase is now complete and the TFORM basic menu is displayed.
Otherwise, the following message is displayed:

ENTER FORM NAMES:

You must specify the name of each form to be made resident, pressing the /CR/ key after entry of each name. To terminate entry, you must press the /CR/ key twice, as shown below:

<name> /CR/ <name> /CR/.....<name> /CR//CR/

Note that when you specify option 'A', no further validation programs are transferred into the environment library. Therefore, if the appended forms use validation programs which are not already present in the library, you must include the names of the validation programs in the list of form names.

When constructing the $RESIDENT module, LINKFORMS identifies the resident form with the longest data section and takes this value as the page size for the library being created.

Error Messages

LINKFORMS may display an error message that signals abnormal ending of execution. As well as the standard error messages, it may emit the following messages:

```
LINKFORMS : NOT A DIRECTORY
```

The specified object is not a directory.

```
LINKFORMS : OUT OF LIMIT
```

The page size multiplied by the maximum number of resident forms that an application program can execute concurrently exceeds one segment.

```
LINKFORMS : INVALID PARAMETERS
```

You have specified an invalid parameter.

```
LINKFORMS : SPACE LIMIT REACHED
            m FORMS OF n LINKED
            WANT TO ABORT (Y/N)?
```

The $RESIDENT module has reached the maximum allowed size of about 320 Kbytes.

If you enter Y, the work performed by LINKFORMS is lost.
If you enter N, the environment library is not lost but it only contains the $RESIDENT consisting of the 'm' processed forms. The remaining resident forms, the non-resident forms and any validation programs are not included.

```
WARNING : <name> IS NOT A FORM NOR A VALIDATION
                                              PROGRAM
```

LINKFORMS will not take into consideration the element specified by <name> . This element will therefore not be present in the environment library. LINKFORMS continues execution.

7. HARD/SOFT COPY OPTIONS

This chapter describes the TFORM print and display functions. The following functions are provided:

- printing of a form
- display of a form
- display of the list of catalogued forms
- display of the list of linked forms.

PRINTING A FORM

When you select option 2 (PRINTING) of the Basic Menu, TFORM displays the following menu:

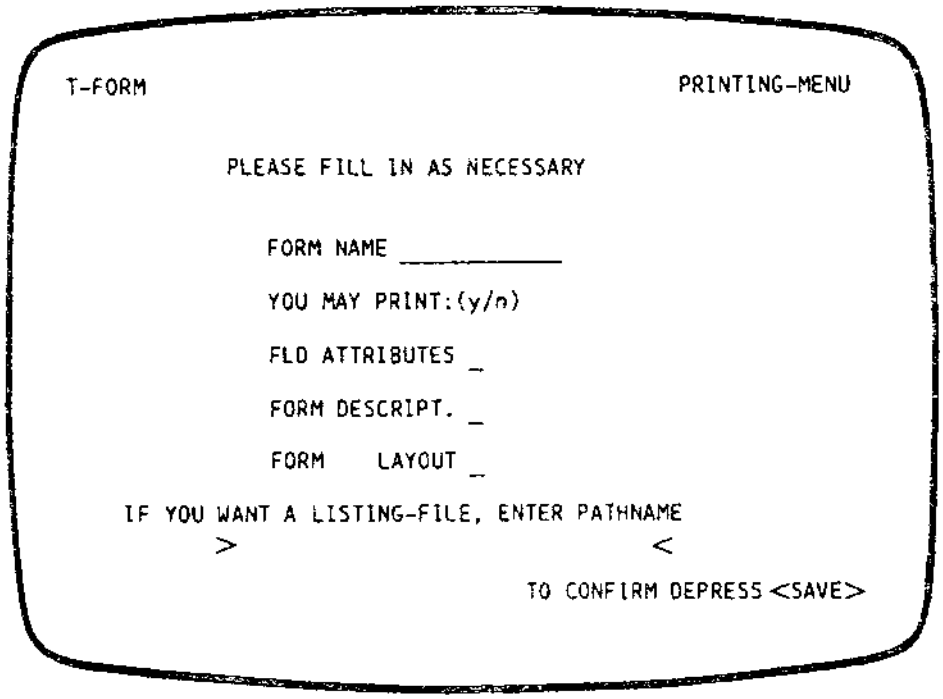


Fig. 7. 1 - Printing Menu

You must enter the name of the form that you want to print and then select the required print options. By default, all the options are printed but you may suppress any of the options by entering 'n' against it. The options provide the following information:

- FLD ATTRIBUTES
  the attributes of each field and the subform structure

DISPLAYING THE LIST OF CATALOGUED FORMS

Select option 3 of the Basic Menu.
TFORM will display the list of all forms catalogued in the current library in the following format:

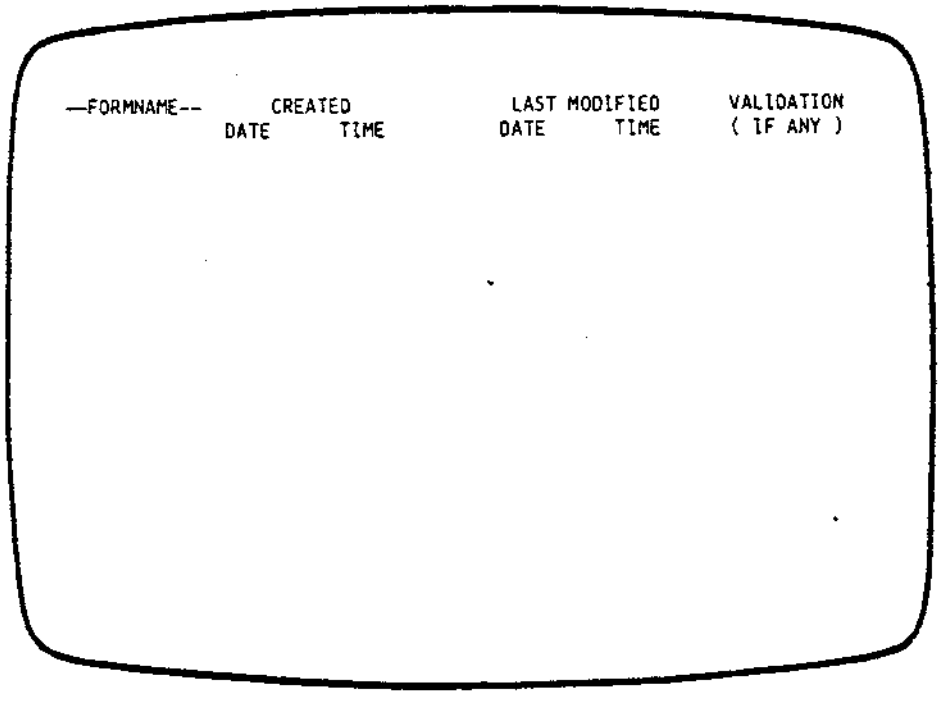


Fig. 7. 2 - Form List Display

For each form, TFORM displays its name, the date and time of creation and last modification, and the name of the related validation program (if defined). An asterisk (*) is displayed next to any validation program name if that validation program is not compatible with the TFORM release being used; in such cases you must follow the 'conversion

procedure' (as explained in the "Guide to the Release") to make the programs compatible.

Each screen page gives the column headings and a maximum of 20 lines of information.

Press:

- /+PAGE/ to display the next screen page

- /HOME/ to display the first screen page

- /EXIT/ to return to the Basic Menu.

DISPLAYING THE LIST OF LINKED FORMS

Select option 6 of the Basic Menu.
TFORM requests the pathname of a directory used as the destination directory in a previous LINKFORMS operation. Enter the pathname of the required directory; TFORM then displays the following information:

PRELOADED FORMS NUMBER = ..

MAX. FORMS SIMULTANEOUSLY OPEN NUMBER = ..

and a list of all the linked form names.

Press the /ENTER/ key to continue, that is to return to the Basic Menu.

8. ERROR MESSAGES

During execution, TFORM may display error messages to indicate an error situation, which can however usually be eliminated immediately and in a simple manner.

Each message is accompanied by an error code.

Note that you must always press one of the following keys: /EXIT/, /↵/, /ENTER/, etc. after display of an error message before any correction can be attempted.

01 PRINTER ERROR - RETRY COMMAND
A printer error (out of paper, in local mode, etc.) has occurred; TFORM automatically retries the command a maximum of three times, and aborts if the error still exists.

02 DEVICE LIST IS FULL
The allowed number of devices has been selected: no other device may be added to the list.

03 DIRECTORY NOT FOUND
You have specified a non-existing directory name.

04 ILLEGAL DEVICE CODE
The value introduced does not conform to device code.

05 FILE ALREADY EXISTS - REWRITE?
You have specified a filename for writing which already exists. Press /SAVE/ to overwrite it or any other key to abort.

07 FIRST CHAR. MUST BE ALPHABETIC
You have not followed the rules for name selection: enter a different value for the parameter.

08 EMBEDDED BLANK NOT ALLOWED
You have not followed the rules for name selection: enter a different value for the parameter.

09 DUPLICATE SUBFORM NAME
The name specified for the subform has already been used: enter a different subform name.

10 INVALID DEVICE REQUEST
Illegal value: enter a different value for the parameter.

11 ILLEGAL CLASS
Illegal value: enter a different value for the parameter.

PART 2

- VPL

INTRODUCTION TO PART 2

GENERAL INTRODUCTION

The second part of this manual contains information and rules to be followed when writing a validation program, divided as follows:

- the validation program

- the validation language.

9. THE VALIDATION PROGRAM

An application program that handles data using forms may require various validation operations on the fields of these. These operations are not performed directly inside the application program but by an appropriate program associated to the form and executed at VISA time.

This program is called the validation program and it is written in the Validation Processing Language (VPL).

Definition

A validation program is a program written in VPL consisting of one or more routines, which carry out validation operations on the form's fields.

When at least one field has been defined 'to be validated' during FIELD ATTRIBUTE DEFINITION (a number, indicating the starting line of the validation routine associated to that field, has been given in at least one of the VALIDATION BEFORE or VALIDATION AFTER parameters), TFORM will activate the form validation phase (Fig. 1.1).

After completion of the KEYS DEFINITION MENU phase, TFORM enters VPL environment, which can be seen as a subenvironment of TFORM.
The VPL environment can be exited from by entering one of the following:

- The EXIT command or the /SAVE/ key.

- The ABORT command or the /EXIT/ key.

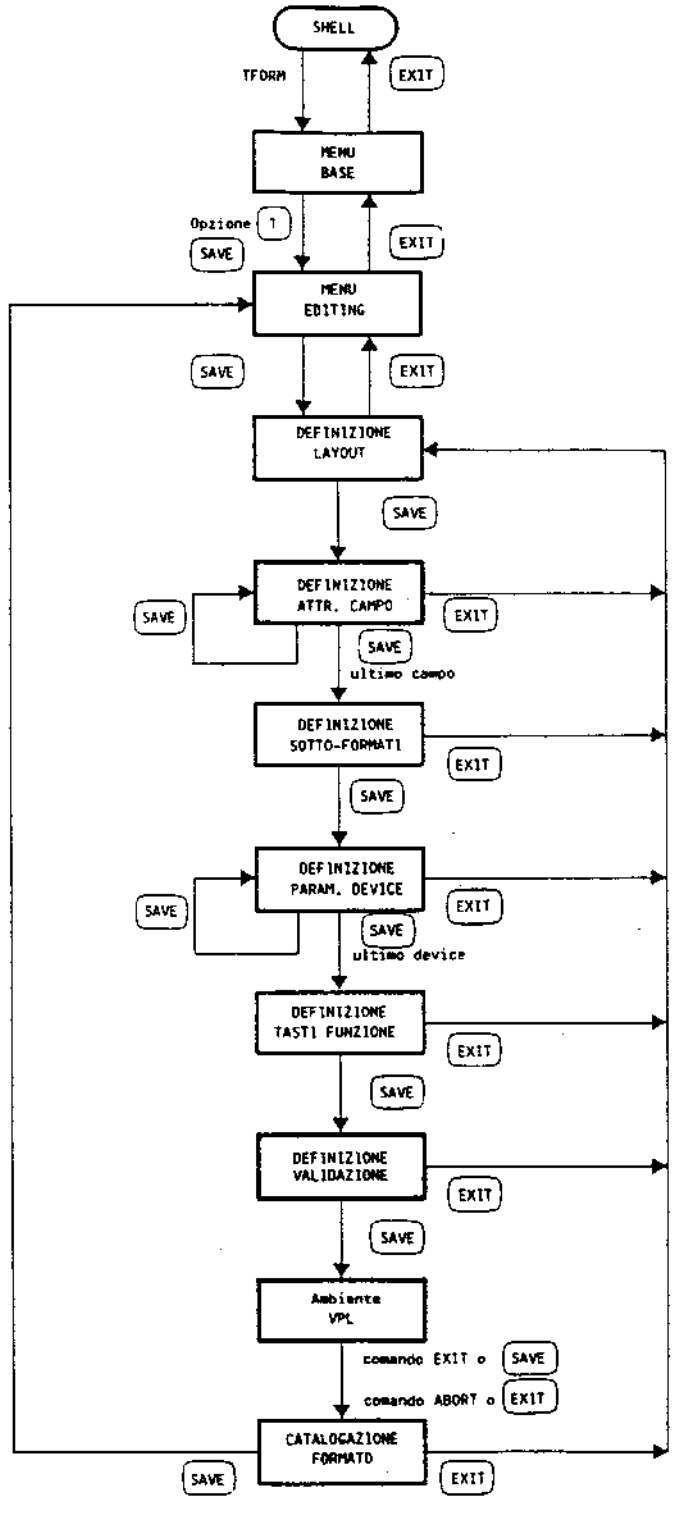


Fig. 9. 1 - Form Creation and Modification Cycle with Validation

TFORM time and VISA time

A validation program is written at TFORM time. The statements entered by the user constitute the source program; the source program is semi-compiled at TFROM time producing an intermediate code which will be executed at VISA time.

A validation program can be used to perform the following operations on the fields of a form:

- Input
- Check
- Computation
- Positioning
- Display
- Immediate return to the application
- Device change
- Enabling of particular attributes
- Read internal data.

debugging

As well as the normal run time execution at VISA time (associated to a form) a validation program can be executed in VPL environment at TFORM time for debugging purposes.

VPL Program

A VPL program is written in the VPL environment. The routines are executed under the control of VISA:

- at OPENFM (open form) time, to perform the initialization routine
- during INPUTFM (input to the fields by operator) whenever has a 'before' or/and 'after' associated routine
- at CLOSEFM (close form) time, to perform the termination routine.

Note that the initialization routine and/or the termination routine may be omitted.

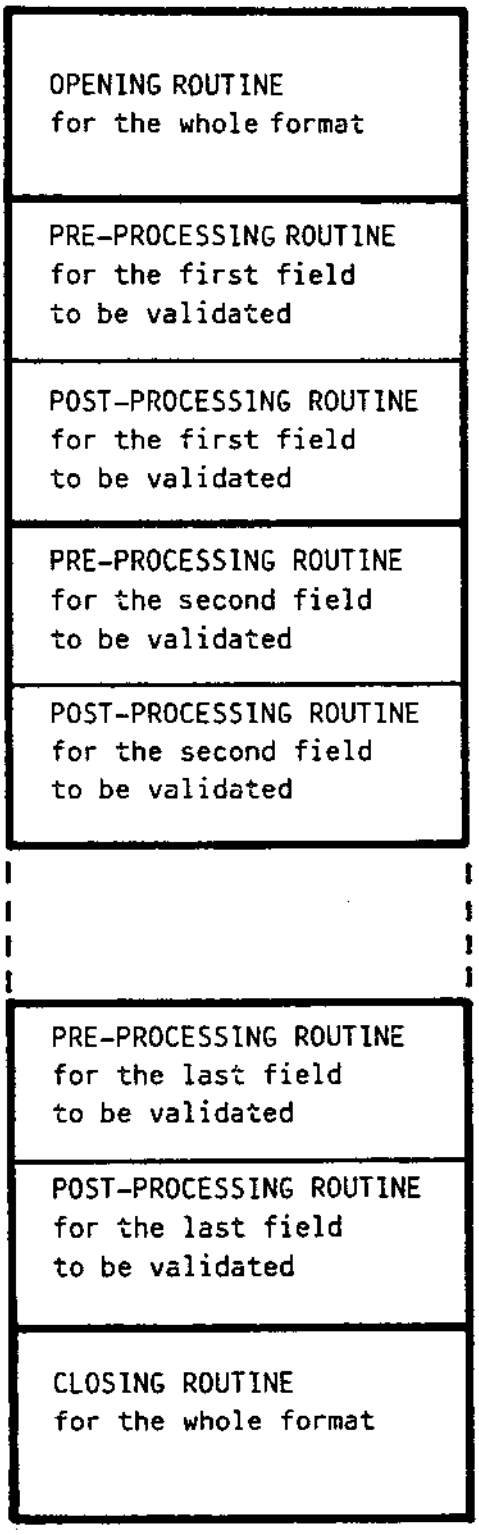


Fig. 9. 2 - Validation Program Routines

The following menu is displayed during the VALIDATION DEFINITION phase:

OPEN VALIDATION :

CLOSE VALIDATION :

VALIDATION PROGRAM NAME :

To validate an entire form with an open and/or close routine, you must enter the line number at which these routines start.

You must also enter the name that identifies the validation program; the program will be catalogued with this name in the directory which will also contain the matching form.

Once you have defined the above, press the /SAVE/ function key to enter the VPL environment or the /EXIT/ function key to return to the previous phases.

TFORM time

When the VPL environment is entered, the VPL prompt is displayed and the environment is in Commands Status.

In VPL environment (TFORM time), you can operate in two different ways:

- Direct Mode
- Indirect Mode

Direct Mode is when you enter an immediate line, Indirect Mode is when you enter a program line.

Direct Mode

In this case, you write statements, commands or functions and these are executed immediately when you press the /ENTER/ key.

You can write several commands on the same line (a line has a maximum length of 255 characters) separating them with the : (colon) character; the commands are executed in sequence when you press the /ENTER/ Key.

Direct mode is used for arithmetic and logical computations with immediate display of the result or storing of this for subsequent use (while the command or sequence of commands is lost).

Direct mode is also used for program debugging.

Indirect.Mode	In this case, statements, commands and functions are written preceded by the line number; this line is transferred to memory and will be executed only when the RUN command is invoked. These numbered lines will make up the lines of a program.
TFORM time command use	Certain commands used for program input, editing, execution in VPL environment and its debugging are valid only at TFORM time. Program execution can be simulated in the same VPL environment. The main commands used for validation program input and debugging are as follows:
AUTO	for automatic numbering of program lines
EDIT	to enter Edit Mode at the specified line
LIST	to list the proram on the video
LLIST	to print the program or append it to a file
LOAD	to read a VPL program from an external file
DELETE	to delete program lines
NEW	to delete the entire program
RENUM	to renumber the lines of the program
BREAKON	to interrupt program execution at the specified line number or keyword
BREAKOFF	to disable use of the BREAKON command
STOP	to temporarily interrupt program execution and return to command level
STON	to enable step by step execution of a program
STOFF	to disable the STON command
CONT	to resume the execution of a program interrupted by a /CONTROL/ /C/ or by the BREAKON, STOP, STON commands
RUN	to start program execution.

Exit from VPL — Can be made by entering one of the two following commands:

EXIT — exits VPL environment and catalogues the program.

ABORT — exits VPL environment without cataloguing the program.

Routine — The starting line of each routine of a VPL program must be consistent with what has been defined in previous phases.

Each routine must terminate with the END command.

Variables declared in one routine can also be accessed by other routines (activated after the first routine) as the set of routines making up a program is seen as a single program.

FIELD command — The FIELD command is particularly important in the Validation Language.

It provides a view of the form data record (allocated in the memory by VISA when the form is processed), which contains the set of fields of a form.

This view or representation refers to the "logical" structure of the form as described during SUBFORM TREE EDITING.

The fields are identified according to their length in the form data record and are associated to internal VPL variables which will identify them in the program.

Syntactically, field length is expressed in number of bytes and the AS keyword is used to associate an internal VPL variable to this number; the variable is indicated by one or more characters always followed by the $ character (string type variable).

Example 1 — The representation given below is an example referred to a simple form.

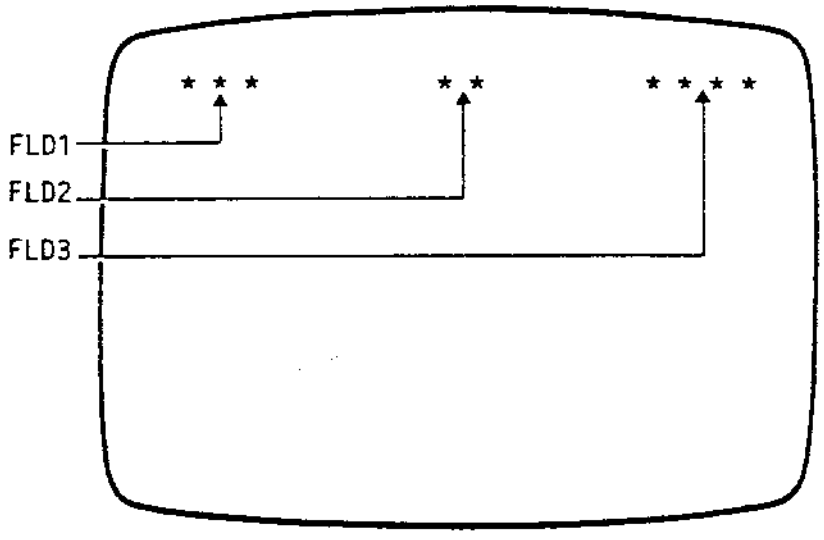


Fig. 9. 3 - Example No.1

It is assumed that USER CLASS = ST has been assigned to all three fields, i.e. a string type internal representation, and at least one field is to be validated, for example FLD3.

With a subform description as follows (which in this case represents the entire form):

```
01 TOT
 02 FLD1
 02 FLD2
 02 FLD3
```

to allow the validation program to refer to this form you must write:

```
FIELD 3 AS A$, 2 AS B$, 4 AS C$
```

therefore fields FLD1, FLD2, FLD3 are identified in the form data record with length 3, 2, 4 and the program will recognise these fields as A$, B$, C$.

Example 2

In its representation, the FIELD command takes into account the space occupied in the memory by the individual fields.

Therefore, with an application program that defines the following data structure (in COBOL):

```
01  AP_REC.
 02 F1 PIC 9(3) USAGE COMP-3.
 02 F2 PIC 9(2) USAGE COMP-3.
 02 F3 PIC 9(4) USAGE COMP-3.
```

that occupies the following memory area (See COBOL Language Referance Manual):

```
field    F1   2 bytes
field    F2   2 bytes
field    F3   3 bytes
```

the following will be obtained at TFORM level where the relative data are to be expressed using a form:

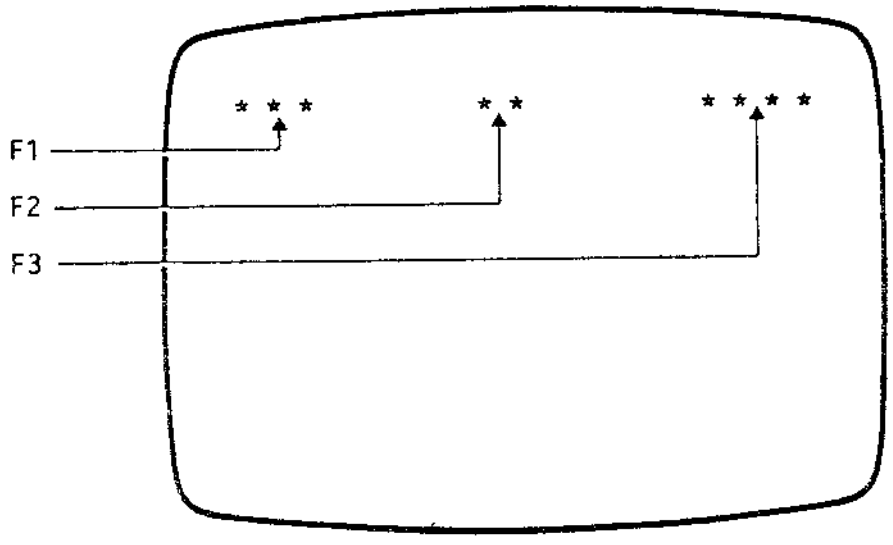


Fig. 9. 4 - Example No.2

where, to ensure consistency, fields F1, F2, F3 must have a USER CLASS : PK.

With the following subform:

```
01  TTL
 02 F1
 02 F2
 02 F3
```

and assuming that at least one field is to be validated, for example F3, the view provided by the FIELD command, in the validation program, may be:

```
FIELD 2 AS A1$, 2 AS A2$, 3 AS A3$
```

Example 3

The FIELD command can be used to view the entire data record, i.e. all the fields of the form or to obtain a partial view. It must however always be possible to identify the position in memory, i.e. in the form data record, of the fields to be validated. Thus, with the following form:

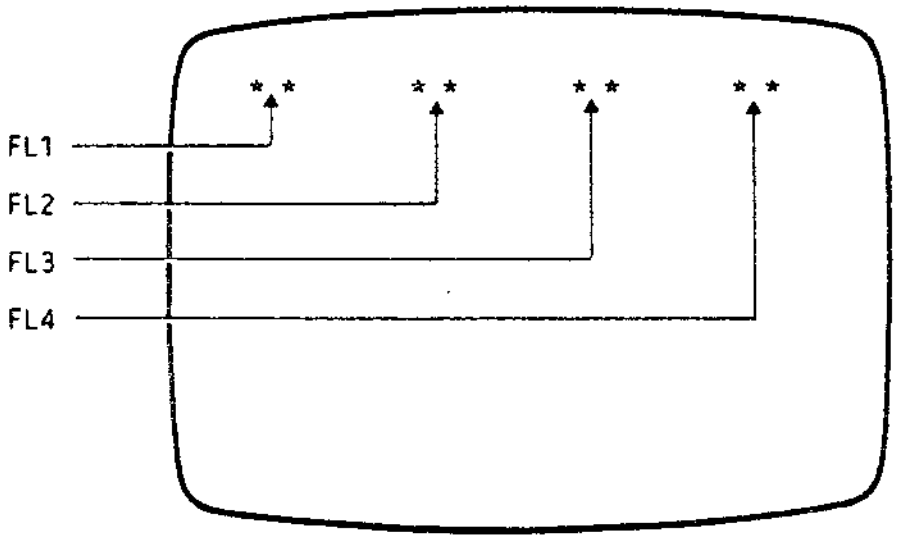


Fig. 9. 5 - Example No.3

with all ST (string) type fields and with the following subform description:

```
01  TOT
 02 FL1
 02 FL2
 02 FL3
 02 FL4
```

we can write, for example, in the validation program:

```
FIELD 2 AS Y1$, 2 AS Y2$, 2 AS Y3$, 2 AS Y4$
```

to view the entire data area, when all the fields

have been validated.

If, for example, FL3 is the only field to be validated the following can be written:

FIELD 2 AS Y1$, 2 AS Y2$, 2 AS Y3$

If, however, a pre-processing routine starts on FL3 and concerns only FL1, the following is written:

FIELD 2 AS Y1$

Example 4 In the case of a form split into several subforms:

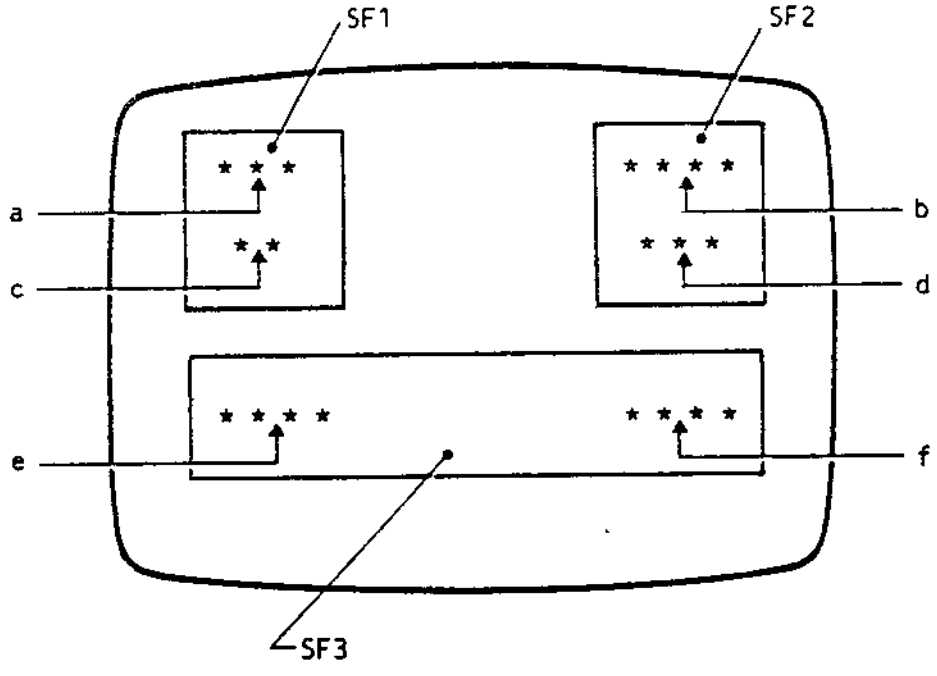


Fig. 9. 6 - Example No. 4

with USER CLASS : ST fields and where field 'd' is the only field to be validated.

The subform description may be as follows:

```
01  TOTAL
 02 SF1
  03 a
  03 c
 02 SF2
  03 b
  03 d
 02 SF3
  03 e
  03 f
```

The FIELD command, with specific reference to the field to be validated, can be written as follows:

```
FIELD 3 AS A$, 2 AS B$, 4 AS C$, 3 AS D$
```

Moreover, in a FIELD command, the fields may be grouped together, representing their total length with a single variable. Or they may be split, representing a single field in several parts, with several variables.
Referring to the above example, the command could be:

```
FIELD 5 AS S1$, 4 AS C$, 3 AS D$
```

or even:

```
FIELD 9 AS N$, 3 AS D$
```

and if field 'b' is split:

```
FIELD 5 AS S1$,2 AS C1$,2 AS C2$,3 AS D$
```

The PRINT, END, INDEV, OUTDEV and ATR statements may refer to only single fields, and not to groups or subdivisions of fields.

A validation program may provide several views of the form data record, i.e. the FIELD command may be used more than once, to meet for example the various requirements of different routines.

The FIELD command acts as a descriptor of the form data record, for use by the VPL environment.

Using this command, the fields are addressed by displacement and are assigned an identifier name for use by VPL.

During input/output operations between the VPL environment and the form data record, data may be handled in two different ways according to whether the values in the data record are of the string or numeric type (in their various representations).

- string fields.
  The VPL program can handle these fields referring to them directly with the name indicated in the FIELD command

- numeric fields.
  The VPL program can handle these fields only using the CVS and MKS$ functions.

CVS function

This is used to transfer numeric data from the form data record to the validation program.

If we have, for example:

```
FIELD 5 AS N$
```

and if the N$ variable identifies a numeric field, the CVS function must be used as follows:

```
A = CVS (N$)
```

so that the numeric value to be accessed for validation is in variable A.

MKS$ function

Used to transfer numeric data from the validation program to the form data record.

If we have, for example:

```
FIELD 2 AS D$
```

and if the D$ variable identifies a numeric field, the MKS$ function must be used as follows:

```
LET X = 150
D$ = MKS$(X)
```

Execution

When an application program contains calls to VISA to process a form with an associated validation program, there is an interchange between VISA and the validation program.
That is to say:

- a OPENFM type VISA call may correspond to the starting of the open validation routine

- a INPUTFM type VISA call may correspond to one or more pre-processing and/or post-processing type routines according to the number of fields with associated validation rountines.

- a CLOSEFM type VISA call may correspond to the starting of the close validation routine.

VISA and the validation program communicate through the form data record which contains the set of processed fields. The validation program refers to this data record using the FIELD command.

The validation program can therefore make checks and computations on all the fields of the form, write in the fields and send messages to the operator.
Given below is an outline of the operations made possible by a validation program:

- to know the functions key that has been used by the VISA operator to close input on a field

- to know the contents of any field of the form

- to modify the contents of any field of the form in the data record

- to output the value of any field on the device associated to that field.

- to redefine the order in which the fields are processed during input from the validation program, directing the operator to a certain field

- to change the visual attributes of any field of the form

- to redefine the input and output devices for any field of the form

- to read internal data assigning them to variables.

VISA time — During the VISA - validation program dialogue, the following commands and functions are particularly important, as well as the above mentioned FIELD command and CVS and MKS$ functions:

LTERM — With this function, it is possible to know which functions key has been used to close the field before starting validation operations

LET — with this command, it is possible to assign a value to a variable

PRINT — with this command, it is possible to display the contents of a field of the form on the device associated with that field. It can also be used to send messages to the operator (reserving space for this is the form).

ATR — With this command, the visual attributes of an output field can be activated or removed.

INDEV — With this command, the output device, assigned to a field, can be redefined.

OUTDEV — With this command, the output device, assigned to a field, can be redefined.

ERROR — With this command, an error code defined in the validation program can be sent to the application. Control is passed to the application and it is given the current data area portions.

END — This command must be entered to close each routine of the validation program.
By associating a variable identifying a field to this command, the normal field processing order can be interrupted and input addressed to the operator on the desired field.

DEFKYB — Redefines the correspondence between the function keys and their codes.

Programming guidelines

This section provides some notes about how to program efficiently in VPL.

1. It is useful to write a single validation program (VP) which handles all the forms used by an application program. This VP will be loaded into memory at the start of the application and will remain there throughout execution of the application program. This will reduce run-time overheads since separate VPs need not be loaded and unloaded.

2. Each routine in a VP should be utilised by as many different forms and fields as possible, thus preventing the duplication of code.
   A single VP routine can handle similar forms and subforms if it is structured such that certain fields can be skipped when they are not relevant. For example, taking two forms which only differ in one field, it is possible to use the INDEV command to skip the field in question when handling the form that does not use it.

3. VPL is a language which is semi-compiled at TFORM time; the intermediate code is then executed at VISA time. Therefore, in order to obtain low memory occupation and acceptable interpretation speed, the following points should be remembered when writing a VP:

   - alwas use integer not single precision variables as loop controllers
   - variables should not be initialised unnecessarily
   - in the FIELD command only the variables which are used must be declared individually, other variables should be grouped as filler.

4. A source VP program must not exceed 192K bytes, and a semi-compiled program must not exceed 64K bytes. Should a VP program exceed this limit, a second VP has to be used. In this case the application has to be structured with 'overlays'. Two or more application phases must be identified, each with its own VP, such that transitions between the phases are kept to a minimum and VISA only has to load a new VP when there is a transition from one phase to another.

10. THE VALIDATION LANGUAGE

This chapter, and the following chapters, describe the elements and rules of the validation language.

The VPL language character set consists of the following:

- alphabetic characters
- numeric characters
- special characters
- control characters
- function keys

Alphabetic Characters

The alphabetic characters of the VPL language consist of the upper and lower case letters of the English alphabet.

Numeric Characters

The numeric characters are the digits from 0 to 9.

Special Characters

The special characters available are the following:

␣ ; = + - * \ ^ () % # $![ ], .':& ? < > / @ _ "

Control Characters

The control characters are entered by pressing a specific key and the CONTROL key simultaneously. The following character is available :

/CONTROL/ /C/: suspends program execution and causes the system to return to Command Status.

Function Keys

The function keys which are used are classified as follows:

- special function keys (→, ←, |←, →|, RESET, IC, DC, ↖, ↑, ↓)
- end of input keys.

See the LTERM command for a complete list of the function keys and relative codes.

The /BS/ or / ⟵ / key is also available. This key deletes the last character that has been input.

National Keyboard

This national equivalents of the ASCII characters are given in the table below.

ASCII VALUE		NATIONAL EQUIVALENT														
DECIMAL	HEXADECIMAL	USA	ITALY	FRANCE	GREAT BRITAIN	GERMANY (ORIGINAL)	GERMANY (WEST)	SPAIN	PORTUGAL	DENMARK	SWEDEN FINLAND	NORWAY	SWITZERLAND FRENCH	SWITZERLAND GERMAN	GREECE	YUGOSLAVIA
35	23	#	£	£	£	#	#	£	#	£	#	£	£	£	£	#
36	24	$	$	$	$	$	$	$	$	$	¤	$	$	$	$	¤
64	40	@	§	à	@	§	§	§	§	´	@	´	§	§	@	Š
91	5B	[	°	°	[	Ä	Ä	¡	Ã	Æ	Ä	Æ	à	ä	[	Đ
92	5C	\	ç	ç	\	Ö	Ö	Ñ	Ç	Ø	Ö	Ø	ç	ç	\	Ć
93	5D	]	é	§	]	Ü	Ü	¿	Õ	Å	Å	Å	è	è	]	Ž
96	60	`	ù	`	`	`	`	`	`	`	`	`	`	`	`	š
123	7B	{	à	é	{	ä	ä	°	ã	æ	ä	æ	ä	ä	{	đ
124	7C	|	ò	ù	|	ö	ö	ñ	ç	ø	ö	ø	ö	ö	|	ć
125	7D	}	è	ê	}	ü	ü	ç	õ	å	å	å	ü	ü	}	ž
126	7E	~	ì	¨	~	ß	ß	~	°	¨	—	¨	é	é	~	č

Fig. 10. 1 - National Keyboard Equivalents.

Scientific Keyboard

The scientific keyboard has an alphabetic part with associated VPL statements.
They are entered by simultaneously pressing the corresponding key and SHIFT.

11. NOTES ON THE SYNTAX AND LEXICAL STRUCTURE

This chapter describes the syntax adopted by VPL and its lexical rules.

Program Elements

A VPL program consists of a set of numbered lines containing statements.

You can enter lines with one or more statements. In the latter case each statement must be separated by a colon (:).
In a program each line starts with a line number: an integer greater than or equal to 0 or less than or equal to 65529 and ends when you press the carriage return/line feed key. You can enter up to 255 characters per (logical) line. A logical line can include several physical lines.

For example:

```
. . . . .
20 FIELD 5 AS A$, 10 AS B$, 15 AS C$, 4 AS D$, 6 AS
E$, 5 AS F$
. . . . .
```

is one logical line divided into two physical lines.

Each VPL statement contains:

a line number - a keyword - the statement body.

For example:

```
1010 PRINT A+B
```

1010 is the line number
PRINT is the keyword
A + B is the statement body

Some statements contain more than one keyword or statement body.

For example:

```
320 IF X THEN A = 200
```

IF and THEN are keywords, while X and A = 200 are

the statement bodies

The various element types that make up a statement must be separated by at least one blank.
For example :

```
100 PRINT A
```

Line Number

The line number must be a positive integer (from 0 to 65529). You must enter the line number as it distinguishes one program line from another. Program line numbers are ordered in memory in ascending line number sequence, irrespective of the order in which they are entered.

It is conventional to use an interval of 10 between each line number. This allows you to insert new program lines between the existing ones.

You can ask the system to number your lines for you through the use of the AUTO command. You can also change the order of the line numbers by altering the first line number and selecting a different increment between two numbers. You do this with the RENUM command.

The current program line can be referred to at TFORM time (using the EDIT, LIST, AUTO and DELETE commands) by indicating the "." character (full stop) in the body of the command.

Keywords

Each statement begins with a keyword (or reserved word). The keyword is a mnemonic of an English word. For syntax reasons it must be preceded or followed by at least one blank.
The keyword defines the type of statement to be carried out. One or more operands (constants or variables) or expressions can be entered after the keyword.
Some statements have more than one keyword, e.g. IF...THEN.
You can enter keywords in lower-case or upper case letters. They are converted into upper-case letters when listing the program.

Statement Body

The statement body contains operators and operands. These are preceded by the keyword of the statement.

You can write some operators and/or operands before the keyword.

For example:

```
100 IF A > B THEN 200
```

Operators

The operators consist of one or more special symbols.

- the same operator can have a different meaning depending on the statement in which it is used. For example, the equals sign (=) can be used to assign a value (in the LET statement, for example) or it can be a relational operator (in a logical expression).

- some operators are not used with any particular statement and have no pre-established position inside the statement itself.

  For example:

  ```
  10 A = (B+C)/D
  20 IF A>=A1 THEN 100
  ```

  "(" "+" ")" "/" are examples of these operators

- some operators are used with a particular statement and have a pre-defined position inside the statement

  For example:

  ```
  1000 A = A+B/C
  ```

  "=" is an example of this type of operator.

Operands

You can use the following operands in a VPL statement:

- simple variables, array elements, arrays
- control items
- numeric constants (i.e. an integer, a fixed decimal point number, a floating decimal point number)
- a string constant
- a hexadecimal constant
- any string of characters in the REM statement .

Blanks

You can insert blanks in any position to make your program easier to read. The only restrictions are:

- a keyword must be preceded and followed by at least one blank.
- blanks are significant within string constants
- blanks are forbidden within numeric constants (including line numbers), keywords, variable and function names.

12. INTRODUCTION TO VPL STATEMENTS

This chapter gives information on the VPL statements.
The description has arranged to aid the user operating at TFORM time.

You can insert a comment at any point in a program, in one of two ways:

- using the REM statement
- using comment fields (character strings preceded by an apostrohpe (')) and followed by a carriage return.

Any string of characters can be inserted after the REM keyword.

For example:

```
100 REM RECTANGLE
.
.
.
350 'CALCULATION OF THE AREA OF A RECTANGLE
```

There are simple data and subscripted data.

The types of simple data are:

- integer
- single precision
- double precision
- string

The types of subscripted data are:

- array

The statements DEFINT/DEFSNG/DEFDBL/DEFSTR can be used to explicitly declare the variables of a program, that is, to associate a type to each program variable.

These can be:

- numeric variables
- string variables
- numeric arrays
- string arrays.

An array is a series of simple variables (indexed variables) which are all of the same type.

A type is associated to a constant when this appears in a VPL statement according to the rules stated in chapter "DATA".

A type is associated to an expression when this is evaluated according to the rules in chapter "EXPRESSIONS".

A type is associated to a function value when this function is evaluated.
This type is also called the "function type".

It depends on the algorithm used to calculate the function in the case of a user-defined function. Built-in numeric functions are of a pre-defined type.

A type is associated to a simple variable when the variable is implicitly or explicitly declared (see chapter "DATA").

A common type is associated to each element of an array when the array is implicitly or explicitly declared (see chapter "DATA").

The common type is also called "array type".

Conversion is automatic during assignments and argument-parameter transfer (see chapter "DATA").

See chapter "DATA" for rules on compatibility.

Simple numeric variables and elements of numeric arrays which have not been explicitly declared in a VPL program, will be implicitly declared in single precision.

Array Dimensions

The dimensions of a numeric or string array and the upper and lower subscript bounds, can be defined explicitly with the DIM statement otherwise they may be assumed implicitly.

ASSIGNMENT STATEMENTS

There are three assignment statements in VPL:

- the CLEAR statement which sets all numeric variables to zero and initializes all string variables to null.
- the LET statement which assigns the value of an expression to a variable. The variables and the expression must both be of the same type (either numeric or string).
- the SWAP statement which exchanges the values of variables provided they are of the same type (integer, single-precision, double-precision, string).

Numeric Assignments

If the value of the numeric expression is not the same as that of the receiving variable, VPL converts the type of the expression value to the type of the receiving variable according to the rules described in chapter "EXPRESSIONS".
Rounding or overflow may occur, if the receiving variable is not able to contain the computed value.

String Assignments

String assignment is performed by moving the string expression value into the receiving variable, character by character, from left to right.

CONTROL STRUCTURES

Control statements alter the normal flow of program execution, by branching to another part of the program.
Branches may be conditional or unconditional.

Unconditional Branches

The GOTO statement causes an unconditional transfer of control. You simply indicate the number of the line to which control is to be transferred.

Conditional Branches

You may use the following statements to branch to a specific statement:

- IF...THEN...ELSE
- IF...GOTO...ELSE
- ON...GOTO

Loops

You can create loops using:

- the FOR and NEXT statements, that are used to enclose a series of statements enabling you to repeat those statements a specified number of times
- WHILE and WEND statements that can be used to enclose a series of statements, enabling you to repeat these statements as long as a given condition occurs.

FUNCTIONS AND SUBROUTINES

The main difference between a function and a subroutine is the way in which they are used. Both can use and modify the values of program variables, but a function is used mainly to compute a single value and return this to the statement that requested it.

Functions

A function is a series of pre-defined operations. VPL will interpret the function when it finds its name in an expression.
A function computes a single numeric or string value depending on the type of expression used to define the function.
One, more than one or no argument may be transferred to the function. Arguments are separated by commas. They may be constants, variables or expressions.
Parameters are also separated by commas, but a parameter may only be a variable.
The number of arguments must be the same as the number of parameters and their types (numeric or string) must match. The association between arguments and parameters is positional.

Each function can be called simply by stating its name followed in parentheses by one or more "arguments" representing the values to be passed to the parameters. We can classify VPL functions into two main categories:

- Built-in or system functions
- User-defined functions

Built-in or System Functions

Built-in functions are an instrinsic part of VPL. They provide a set of commonly used numeric and string operations. They can be used in any program without being explicitely defined. A complete list and a detailed description of system functions is provided in chapter "VPL STATEMENTS, COMMANDS AND FUNCTIONS".

Built-in Numeric Functions	VPL provides a number of pre-written routines that permit certain mathematical functions to be calculated such as square root, logarithms, sinus, cosinus, tangent, arctangent etc.
Built-in String Functions	These are instrinsic functions that calculate a string or a numeric value and permit one or more than one numeric and/or string argument(s) to be used. They simplify string operations such as the extraction of a group of characters i.e. a substring, from a larger string.
User-Defined Functions	The user can define an arbitrary number of functions within a VPL program using the DEF FN or the DEF FN/FNEND statements. User-defined functions are called in exactly the same way as built-in functions. The only limitation is that the definition is program-dependent and must therefore be redefined in each program that needs to use it. DEF FN on DEF FN/FNEND define a numeric or string function.
Subroutines	A subroutine consists of any sequence of VPL statements that can be called repeatedly and it forms an integral part of the program. A subroutine can be called by a GOSUB or an ON...GOSUB statement. At the end of execution of a subroutine, control is returned to the first statement following the most recent GOSUB (or ON...GOSUB) that has been executed. A subroutine ends with the RETURN statement. If a program refers to the same subroutine more than once, control is always returned from the subroutine to the statement following the most recent GOSUB (or ON...GOSUB) executed. A subroutine may be called by another subroutine. In this case, the called subroutine is "nested" within the calling one. The number of "nested" subroutines that are active at this same time, is only limited by the amount of memory available.
Storage	A source VP cannot exceed 192K bytes. A semi-compiled VP cannot exceed 64K bytes.

SUSPENSION OF PROGRAM EXECUTION AT TFORM TIME

Program execution is suspended when:

- /CONTROL/ /C/ is entered

- a STOP statement is executed

- an error message is issued.

The System will enter Command Mode in the above cases (in the case of a syntax error, however, Editor Mode is entered).
In Command Mode you can display the program variables (using the immediate statements PRINT) or modify the value (using the LET or CLEAR statements).
It is possible to resume execution by entering the CONT command. Program execution cannot be resumed if it has been modified or if an error has been detected.

DEBUGGING AND ERROR RECOVERY AT TFORM TIME

Leaving aside errors made when entering a line, there are two types of error that can be made at TFORM time:

- run-time errors which halt execution and which cause an error message to be displayed

- logic errors which permit complete program execution but cause incorrect or unexpected results.

Run-time errors may be syntax errors or other types of run-time errors (NEXT without FOR, RETURN without GOSUB, etc.)
You can also simulate a VPL error or generate a user-defined error. See ERROR statement.

Tracing Program Execution

A convenient method of debugging logic errors is to trace the order of statement execution in all or part of a program.
VPL provides the following two tracing commands (TRON/TROFF, STON/STOFF) that may also be used as program statements and wich cause the line number of each statement executed to be listed and stop the line number listing.

Error Testing and Recovery

Normally, when a run-time error is encountered, VPL handles the error by halting execution, displaying an appropriate message and returning to Command Mode.
In same cases you may want to handle the error in a different way. This can be done by writing an error-handling routine.

13. DATA

Each data-item may appear in a VPL program as either a constant or a variable.

Constants may be:

- numeric
- string.

Variables may be:

- numeric
- string.

Variables may also be simple or subscripted (arrays).

CONSTANTS AND VARIABLES

Constants

Constants are specific numbers such as 15, -2, 3.41 etc. or specific strings such as "AAA.b1", "Cursor***". This means that their values remain the same throughout program execution.

Variables

A variable is a named data-item whose value may change during program execution.

Variable Names

The identifier (or name) of a variable may be of any length, but only the first 40 characters are significant. The characters allowed may be letters or numbers. The period (.) is also allowed. The first character must be a letter. The last character may be a letter, a number, a period, or a type declaration tag (%, !, #, $). The meaning of a type declaration tag is explained later in this chapter. Lower case letters in a variable identifier are considered equivalent to their corresponding upper case letters and are converted to their corresponding upper case letters when the program is listed. A reserved word (a keyword, a command or function name) cannot be used as a variable identifier, but VPL allows reserved words to be embedded within, before or after a variable identifier.

For example:

```
10 PERFORMANCE = 105.3
20 SINGLE = 1371.2
```

are valid program lines even though PERFORMANCE contains the keyword FOR and SINGLE begins with the name of the built-in function SIN.

HOW VPL CLASSIFIES CONSTANTS

The way VPL stores a data-item determines:

- the amount of memory (in bytes) that it will occupy
- the speed at which VPL can process it.

Numeric Data

VPL can store all numbers in a program as follows:

- integers (fastest processing speed, limited range)
- single precision numbers (general purpose), or
- double precision numbers (maximum precision, slower processing speed).

	INTEGER NUMBERS	SINGLE PRECISION NUMBERS	DOUBLE PRECISION NUMBERS
Memory space (in bytes)	2	4	8
Range of values	From -32768 to 32767	From $\pm 10^{-38}$ to $\pm 10^{38}$	From $\pm 10^{-308}$ to $\pm 10^{308}$
Significant Digits	Up to 5	Up to 7	Up to 16
Displayed Digits	Up to 5	Up to 6 (with rounding)	Up to 15 (with rounding)

Tab. 13. 1 - Numeric Data

String Data

Strings (sequences of ASCII characters) are useful for storing non numeric information.
For example, the constant:

"FORD, RENAULT"

is a quoted string of 13 characters. Each character in the string (including the blank) is stored in one byte and corresponds to an ASCII code.

A string can be up to 255 characters long. A string with length zero is called a "null" string and is represented by a pair of double quotes (""). VPL allocates strings dynamically i.e. the memory space reserved for a string may vary during program execution from 0 to 255 bytes.

NORMAL CRITERIA FOR THE CLASSIFICATION OF CONSTANTS

The following are the rules by which you may distinguish a constant:

IF....	THEN....	EXAMPLES
the value is enclosed in double quotes	it is a string	"NO" "YES"
the value is not in quotes	it is a number	521# - 15 3.7345E-2
a number is whole and in the range -32768 to 32767	it is an integer constant	1024 721 -32768
the value has the prefix &H, and is enclosed between the digits 0-9 and the letters A-F (in the range 0 to FFFF)	it is a hexadecimal constant.	&H20F0 &HF1 &H35 &HFE98 &HFFFF &H0
the value has the prefix &0 or & and is enclosed between the digits 0-7 (in the range 0 to 177777)	it is an octal constant	&070 &044 &71175
a number is not an integer and does not contain more than 7 digits	it is a single precision constant	-2.3 32768 45.314 -65000
a number contains more than 7 digits	it is a double precision constant	52174593 -54.397124

TYPE DECLARATION TAGS FOR NUMERICAL CONSTANTS

You can override VPL's normal criteria by adding the following "tags" to the end of a numeric constant:

TAG	MEANING	EXAMPLES
!	single precision	5.72110333! the constant is in single precision and only the first 7 digits are memorized (i.e. 5.721103)
E	single precision floating point The E indicates that the constant is to be multiplied by the power of 10, specified after E	7.31E4 means 7.31×10^4 i.e. 73100
#	double precision	4# 5.21#
D	double precision floating point. The D indicates that the constant is to be multiplied by a power of 10 specified after D.	7.2D-3 means 7.2×10^{-3} i.e. 0.0072

HOW VPL CLASSIFIES VARIABLES

Each variable identifier that appears in a VPL program, can be classified as either a string, or integer or as a single or double precision number.

Unless otherwise indicated, VPL classifies all numeric variables in single precision.

However, you may assign different type attributes to variables using either definition statements or a type declaration tag at the end of the variable identifier.

VPL provides four type definition statements (DEFINT/DEFSNG/DEFDBL/DEFSTR).
A type definition statement declares the type of all the variables whose names begin with a specified letter.

The length of a variable identifier is 40 characters.

Type Declaration Tags

As with constants, you can override the type of a variable by adding a type definition tag at the end of the name. There are four type declaration tags for variables.

DECLARATION TAG	MEANING	EXAMPLES
%	integer variable	A% STEP% INCREMENT% are all integer variables, regardless of the type definition statements for the variables beginning with the letters A, S and I.
!	single precision variable	SPEED! SPACE! TIME! are all single precision variables regardless of the type definition statements for the variables beginning with the letters S and T
#	double precision variables	TOTAL# SUBTOTAL# X1# are all double precision variables regardless of type definition statements for the variables beginning with the letters T,S and X
$	string variable	A$ B1$ NAME$ are all string variables regardless of the type definition statement for the variables beginning with the letters A, B and N.

NUMERIC CONVERSIONS

Often a program or immediate line may ask VPL to assign one type of constant to a different type of variable. At this point the conversion procedures described below are used.

Single or Double Precision to Integer

VPL converts the original value in single or double precision to an integer by rounding the fractional part.
The fractional part must be greater than or equal to -32768 and less than 32767 otherwise an Overflow occurs.

For example:

```
C%=-15.1
PRINT C%
-15
```
the value -15 is assigned to the variable C%.

For example:

```
C% = 4,1E2
?C%
 410
```
the value 410 is assigned to the variable C%

For example:

```
C% = 47.8
?C%
 48
```
the value 48 is assigned to the variable C%

Integer to Single or Double Precision

In this case the conversion does not cause an error. The converted value corresponds to the original value with zeros to the right of the decimal point.

For example:

```
S! = 326
?S!
 326
```
326 is stored in the variable S! as 326.000 but it is displayed as 326

For example:

```
D# = 326
?D#
 326
```

326 is stored in D# as 326.0000000000000 but displayed as 326.

Single to Double Precision

VPL adds trailing zeros to the single precision number.
If the original value:

- has a binary representation, no error occurs in the conversion
- does not have an exact binary representation, an arithmetic error is introduced when converting the value.

For example:

```
B# = 1.5
?B#
 1.5
```

When entering B# = 1.5 the value 1.500000000000000 is assigned to variable B# but only 1.5 is displayed.

1.5 has an exact binary representation.

For example:

```
C#=1.3
?C#
 1.29999995231628
```

When entering C# = 1.3 the value 1.299999952316280 is assigned to variable C# but it is only displayed as 1.29999995231628.

Double to Single Precision

This involves converting a number with up to 16 significant digits into a number that has no more than 7.
Only the first seven digits of the original value are valid and the last digit is rounded. Before displaying or printing such a number, VPL rounds it down to six digits.
If the double precision value is outside the range of the single precision values, an Overflow occurs.

For example:

```
P! = 2.03999996
?P!
 2.04
```

the value 2.040000 is assigned to the variable P!

but it is only displayed as 2.04

Illegal Conversions

You cannot convert numeric values to string values and vice versa by means of an assignment statement.

For example:

```
C$=321.7
```

is illegal. (In these cases, conversion takes place with the functions STR$ and VAL).

SUBSCRIPTED VARIABLES AND ARRAYS

An array is a collection of variables of the same type under the same name, which can be distinguished by the value of one or more subscripts.
An array can have a maximum dimension number of 10.
To define an array, you must:

- give it a name (the rules given for naming simple variables apply)

- establish the upper and lower subscript bounds.

To do this, you must write a DIM statement bounds.

It is also possible to redefine an array by means of a new DIM statement that is preceded by an ERASE statement.

14. EXPRESSIONS

VPL can process three types of expressions:

- numeric expressions
  For example:

  A+B*C

- string expressions
  For example:

  A$+B$

- logical expressions
  For example:

  A+B>C

The value of a numeric expression is a numeric value (for example 350.71).
The value of a string expression is a string value (for example "ABC").
The value of a logical expression is a boolean value (true or false).

NUMERIC EXPRESSIONS

A numeric expression contains one or more operands (constants, simple variables, array elements, or functions) which are connected by means of numeric operators (+,-,\,*,MOD,-,/,^).

Numeric Operators

VPL uses operators to define numeric operations. There are eight numeric operations, each identified by its own symbol.

SYMBOL	OPERATION	EXAMPLES
+	addition	X = 3.2 ?X+1.1 4.3
-	subtraction	?X-1.3 1.9
\	integer division. The operands are rounded to the nearest integers (which must be in the range - 32768 to 32767) before the division is performed and the quotient is truncated to an integer.	?10\4 2 ? 25.68\6.99 3
MOD	modulus arithmetic. Provides the integer value which is the remainder of an integer division.	?10.4 MOD 4 2 (10/4 = 2 with remainder 2)
*	multiplication	?X*3.92 12.544
/	division	?3/6.05 0.495868
-	negation	?-X -3.2
^	exponentiation	?X^3 32.760

Tab. 14. 1 - Numeric Operators

Numeric Operator Priority

VPL has priority levels for performing different numeric operations, when these are present in the same expression.
For operators with the same priority (e.g. / and *) operations are carried out from left to right.

Numeric operators and priority levels are described below (in order of descending priority).

Order of priority

1. exponentiation
2. negation
3. multiplication and division
4. integer division
5. modulus arithmetic
6. addition and subtraction.

Use of Parenthesis

There are cases when the normal priority of operations must be changed. To do this you use pairs of parentheses exactly as you would in algebra. When parentheses are used, the operations within the innermost pair of parentheses are performed first, followed by operations within the second innermost pair and so forth. Within a given pair of parentheses, normal priority of operations applies.

NUMERIC EXPRESSION	VPL EQUIVALENT	INTERPRETATION
x+y+z/2	(X+Y+Z)/2	1. Add X,Y and Z 2. Divide the sum by 2
$x+\frac{y+z}{2}$	X+(Y+Z)/2	1. Add Y to Z 2. Divide the sum by 2 3. Add X to the result
2x + 5	2*X+5	1. Multiply X by 2 2. Add 5 to the result
2(x+4)	2*(X+4)	1. Add 4 to X 2. Multiply the sum by 2
$x^2 + 3$	X^2+3	1. Square X 2. Add 3 to the result
$(x + 3)^2$	(X + 3)^2	1. Add 3 to X 2. Square the result
$\frac{(x + 3)^2}{4}$	(X+3)^2/4	1. Add 3 to X 2. Square the sum 3. Divide the result by 4
$\frac{x^2}{6} \cdot \frac{x + y}{2}$	(X^2/6)*(X+Y)/2))	1. Square X and divide the result by 6 2. Add X to Y. and divide by 2 3. Multiply the two results by each other.

Tab. 14. 2 - Examples of Numeric Expressions

Type of Expression

The type of numeric expression depends on the type of its operands.
If the expression involves more than 2 operands, it can be considered as a series of calculations involving two operands.

There are four possible situations depending on the type of the two operands involved:

- if both operands are of the same numeric type (integer, single precision or double precision) the result is also of that type.

- if one operand is integer and the other is single precision: the result is single precision

- if one operand is integer and the other is double precision: the result is double precision

- if one operand is single precision and the other is double precision: the result is double precision.

Rounding, Overflow and Underflow

Floating point types are forms of approximation to the real numbers of mathematics.

- if one or more operands in a numeric expression are floating point, calculations are approximate and accuracy can be lost.
If this occurs the less significant digits are not regarded and the last digit maintained is rounded.

- if the value of the expression exceeds the maximum value allowed for that data-type, an "OVERFLOW" message is displayed; machine infinity with the algebraically correct sign is supplied as the result and execution continues.

- if a division by zero is encountered out, the "DIVISION BY ZERO" error message is displayed; machine infinity with the sign of the numerator is supplied as the result of the division and execution continues.

- if the evaluation of an exponentiation results in zero being raised to a negative power, the "DIVISION BY ZERO" error message is displayed; positive machine infinity is supplied as the result of the exponentiation and execution continues.

- if the value of the expression is less than the

smallest representable value; the value becomes zero (Underflow) and execution continues.

- if in a numeric assignment the execution type is different from the type of the receiving variable, the expression type is automatically converted to the type of receiving variable.

Note Machine infinity is displayed as 1.79769313486231 D +308

Undefined Values

If a numeric variable in a numeric expression has not yet been initialized, it is set to the value zero.

Undetermined Forms

The evaluation of a numeric expression may result in an undetermined form, such as:

0/0: the message "DIVISION BY ZERO" is displayed and the value 1.79769313486231 D +308 (machine infinity) is supplied

0 ^ 0 : the value equal to 1 is assumed

STRING EXPRESSSIONS

A string expression can be either a string constant, a single string variable, a string array element, a string function or a chaining of all these elements through the use of the plus sign (+).

Concatentation of Strings

Strings can be joined i.e. "concatenated". The result of this operation is the string obtained by joining the start of the second string to the end of the first.
When two or more strings are concatenated, the length of the resulting string is equal to the sum of the individual string lengths.

Null Strings

The operand in a string expression may also be a null string (""). The null string may also be the default value of a non-initialized string variable.

note

Care must be taken not to assign more than 255 characters to a string variable, otherwise the system issues the message: STRING TOO LONG.

When a value is assigned to a string variable, a new area is used to store the value if the length of the new value exceed the length of the previous one.

RELATIONAL EXPRESSIONS

Relational expressions compare either two numeric or two string expressions. The type of comparison being defined by a relational operator.

Relational operators

The relational operators are:

= equals

> greater than

< less than

>= or => greater than or equal to

<= or =< less than or equal to

<> or >< not equal to

It is illegal to compare a numeric expression with a string expression and vice versa. Comparison of numbers has an obvious meaning. Character strings may also be compared depending on the numeric value of the character's representation (this is established on the basis of the ASCII decimal value).
String scanning is performed from left to right. Numeric or string expressions are performed first, then relational operators are applied to the result of such expressions.

For example, to write
A>B+C

is equivalent to
A>(B+C)

The result of the relational expression is numeric. It is displayed as either +1 (if the relation is true) or 0 (if it is false).

Using Relational Expressions

The result of a relational expression may be used to make a decision regarding program flow. It is possible to use relational expressions in the following control statements:

- IF... GOTO... ELSE
- IF... THEN... ELSE
- WHILE

where a condition is tested to determine subsequent operations in the program.

For example, the following statement:

```
100 IF A$>B$ THEN 50
```

will transfer control of execution to statement 50 if the condition (A$>B$) is true. If the condition is false, (i.e. A$ is not greater than B$) the next statement will be executed.

LOGICAL EXPRESSIONS

A logical expression consists of one operand preceded by the logical operator NOT, or two operands separated by another logical operator (AND, OR, XOR, EQV and IMP) or two operands separated by a logical operator and NOT.

The operands in a logical expression may be numeric or relational expressions. Both provide a numeric result. The result of a logical expression is also numeric. Examples (of logical expressions) are:

```
NOT X
X AND Y
A>B OR C>D
1% AND A$<B$
A$ XOR B$ not valid (as the operands are strings).
```

Logical Operators

Logical operators work by converting their operands to sixteen bit, signed two's complement integers in the range -32768 to +32767. If the operands are not in this range, an error occurs. If both operands are 0 or -1 the logical operators will provide the result 0 or -1.
The given operation is performed on these integers, where each bit of the result is determined by the corresponding bits in the two operands.
The logical operators are listed below in a 'truth table'.
This describes graphically the results of the logical operations on a bit-by-bit basis.

OPERATOR PRIORITY

The table below lists all the operators (numeric, string, relational, and logical) in the order in which VPL evaluates them.

1. ^ (exponentiation)
2. - (negation)
3. */ (multiplication and division)
4. \ (integer division)
5. MOD (modulus arithmetic)
6. + - (addition and subtraction)
7. + (chaining of strings)
8. All relational operators
9. NOT
10. AND
11. OR
12. XOR
13. IMP
14. EQV

Operators shown on the same line have the same priority.

All relational operators have equal precedence.

Evaluation order of expressions can be overridden by use of parentheses.

For example, the evaluation order of:

NOT A>B AND C>D OR E>F

is different from the evaluation order of:

```
NOT (A>B AND (C>D OR E>F))
```

15. VPL STATEMENTS, COMMANDS AND FUNCTIONS

Analytical Index

The chapter groups the statements, commands and functions into functional groups and gives a brief description of each one.

comment statements

- insertion of any string of characters as a comment in a program: REM

data definition statements

- definition of variables in double precision: DEFDBL
- definition of variables as integers: DEFINT
- definition of variables in single precision: DEFSNG
- definition of variables as string: DEFSTR
- definition of the name of one or more arrays, of the dimensions of each array and of the maximum limit for each dimension: DIM
- erasing the space for one or more arrays and making the corrisponding names available: ERASE
- storing constants to be accessed in VP using the READ statement: DATA

assignment statements

- resetting of numeric variables, initialisation of the string variables to the null string: CLEAR
- assignment of an expression value to a variable: LET

- swapping of two variable values: SWAP
- reading the constants defined with a DATA statement: READ
- specifying the starting point from which to read the constants specified in the DATA statement: RESTORE

control structures

- looping a set of statements a set number of times: FOR/NEXT
- transfer of control to a specified line in the program: GOTO
- transfer of control, based upon a condition: IF...GOTO...ELSE, IF...THEN...ELSE
- transfer of control to a line chosen from a specified set of lines: ON...GOTO
- looping a set of statements until a particular condition is true: WHILE/WEND

definition of a user function

- user definition of numeric and string functions; the definition can only occupy one program line: DEF FN
- user definition of numeric and string functions; the definition can occupy more than one line: DEF FN/FNED

built in numeric functions

- returning the absolute value of a numeric expression: ABS
- returning the arctangent value of a given value: ATN
- conversion of any numeric format into an argument in double precision: CDBL
- conversion of any numeric argument into an

integer: CINT

- returning the argument cosine: COS
- conversion of a numeric argument into a single precision number: CSNG
- raising the power of "e": EXP
- returning the argument integer: FIX
- returning the integer that is higher, lower or equal to the argument: INT
- calculation of the logarithm: LOG
- returning the octal value of a decimal argument: OCT$
- returning value 1 if the argument is positive, of value 0 if it is null or value -1 if it is negative: SGN
- calculation of the argument sine: SIN
- calculation of the square root of the argument: SQR
- calculation of the tangent of the argument: TAN

built in string functions

- returning an ASCII decimal code for the first character of the assigned string: ASC
- returning a character whose decimal code is the argument value: CHRS
- conversion of the decimal argument to the corresponding hexadecimal value: HEX$
- extracting a substring from an assigned string: LEFT$
- calculating the length of an assigned string: LEN
- left flush of a string value in a string variable: LSET
- extracting a substring from an assigned string at a given position: MID$

- totally or partially replacing one string with another: MID$
- searching for the first time a sub-string appears in a string returning the position: INSTR
- returning a sub-string by extracting a specified number of characters from an assigned string, starting from the right: RIGHT$
- right flush of a string value in a string variable: RSET
- returning a string of blanks: SPACE$
- returning a string of a specifed length and with characters which are all the same with respect to the ASCII code or with respect to the first character of a specifed string: STRING$
- conversion of an expression from numeric to string: STR$
- conversion of an expression from string to numeric: VAL

special functions

- "logical" conversion of the value of a variable defined in a FIELD command with a numeric value: CVS
- exit from VPL environment and return to TFORM with catalogation of the eventual program: EXIT
- exit from VPL environment loosing the work done in VPL environment: ABORT
- allocating space for variables in the data area: FIELD
- indicating the last input Key entered during the last input operation at VISA time: LTERM
- assigning a numeric value to a variable defined in the FIELD command: MKS$
- redefines phisically the function key use: DEFKYB
- reading a VPL program from an external file: LOAD

use of sub-routine

- calling a sub-program and returning control to the main program when execution terminates: GOSUB/RETURN
- calling a sub-program selected from n sub-programs specified and returning control to the main program when execution terminates: ON...GOSUB/RETURN

data output

- changing the visual attributes of a field for the output devices associated to that field: ATR
- changing the input devices, associated to a field, at run time: INDEV
- displaying all or part of the program resident in memory: LIST
- printing all or part of the program resident in memory: LLIST
- changing the output devices, associated to a field, at run time: OUTDEV
- sending a value to VISA with the characteristics specified for the format: PRINT
- displaying expressions in VPL environment (TFORM time): PRINT

program entering, modifying and executing a TFORM time

- automatic numbering of program lines: AUTO
- deleting program lines: DELETE
- activating the system in Editor Mode as from the specified line: EDIT
- removing the stored program and the variables: NEW
- modifyng the line numbers of the stored program: RENUM
- activating execution of the stored program: RUN

debugging

- interrupting program execution at the specified line or Keyword: BREAKOFF
- cancelling the BREAKOFF command: BREAKON
- resuming execution of a program: CONT
- terminating program execution: END
- simulating a VPL error or generating an error defined by the user: ERROR
- enabling execution of a step-by-step program: STON
- terminating execution of a step-by-step program: STOFF
- temporarily interrupting program execution and returning to the Command Status: STOP
- interrupting listing activated with the TRON command: TROFF
- generating a list of the line numbers of all the statements that have been executed: TRON

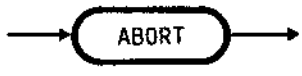
Description of the Statements, Commands, and Functions

The statements, commands and functions are listed in alphabetical order for easy reference.
Each description is organised in the following way:

- Function of the statement, command and function, showing when it can be used at TFORM time or VISA time.
- Syntax.
  The graphic symbology described in the "Guide to the Literature", is used.
- Operating modes of the statement, command or function.

Exits from VPL environment and returns to TFORM (Cataloguing Menu)

Only at TFORM time.



Characteristics

All the work done in VPL session is lost.

No program is written on disk.

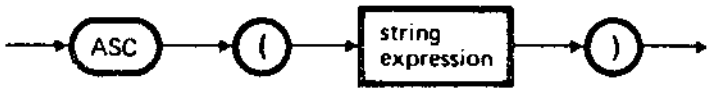
The TFORM /EXIT/ key has the same effect as ABORT.

This function returns the absolute value of a numeric expression.

Example

```
PRINT ABS(7*(-5))

 35
```

This function returns a numeric value, which is the first character of the given string, in ASCII decimal code.



Characteristics

To convert an ASCII decimal code into its corresponding character see the CHR$ function.

See Appendix for ASCII codes.

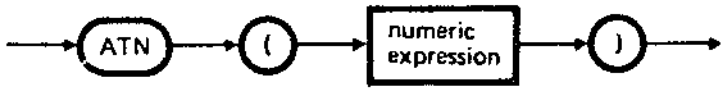
If string expression is null,an "ILLEGAL FUNCTION CALL" error is returned.

Example

```
10 X$ = "TEST"
20 PRINT ASC(X$)

RUN

 84
```

ATN

This function returns the arctangent of the argument.



Characteristics

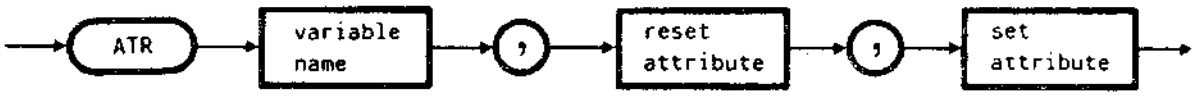
The result is given in radians within the range $-\pi/2$ and $+\pi/2$. Calculation for ATN is carried out in double precision.

The numeric expression can be of any type; the result is always returned in double precision. It is advisable to use a double precision numeric expression as input.

Example

```
PRINT ATN(3)

 1.24904577239825
```

This command allows the visual attributes of a field on the output device indicated for that field to be enabled or disabled.



where:

variable name — specifies the name of a field defined with the FIELD command.

eset attibute — Indicates the code of the attribute to be reset

set attribute — Indicates the code of the attribute to be set.

Characteristics — The visual attibutes that can be reset/set are indicated with their respective codes in the tables below.

MONOCHROME SCREEN

No-effect	0
Overscore	1
Underscore	2
Left margin	4
Right margin	8
Blinking	16
Highlight	32
Reverse	64
No-echo	96

These codes can be added together to obtain a multiple effect (32 cannot be added to 64).

When the printer is specified as output device, the attribute codes are as follows:

Underscore	2
Double size set	32

COLOUR SCREEN

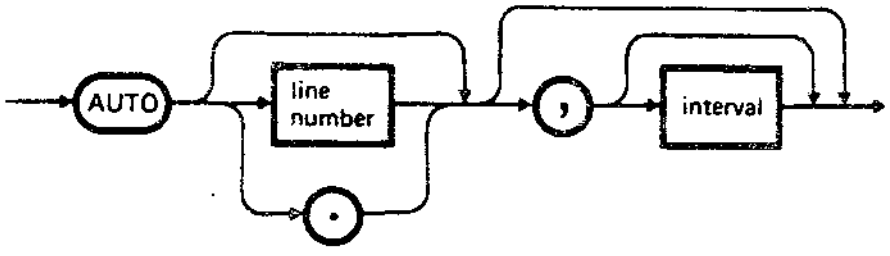
No-effect	0
Overscore	1
Underscore	2
Left margin	4
Right margin	8
Red + blinking	16
White	32
White + blinking	48
Yellow + reverse	64
Yellow + reverse + blinking	80
No-echo	96
Magenta + reverse	112
Yellow	128
Red	144
Cyan	160
Green + blinking	176
Green + reverse	192
Red + reverse	208
Blue	224
Magenta	240

Visual attributes defined for the screen are immediately effective while those for the printer become effective at the next output command that refers to the printer.

AUTO

Automatically numbers the program lines.

Only at TFORM time.



where:

line number — is the first line number generated. The default value is 10. If there is a comma in the command, the default value is 0.

. — is the first line number generated: it is the number of the current line (the last numbered line entered in memory, the last updated or that which contained an error).

interval — is the interval between line numbers. The default value is 10. If the comma is inserted but the interval omitted, the interval value is the last increase specified.

Characteristics — If the AUTO command generates a line number which already exists, an asterisk is displayed after the line number, warning the user that a previously stored line is about to be replaced. By pressing the carriage return/line feed key, the existing line will remain unchanged and another line number will

be generated. To terminate automatic numbering, the user must enter the control character /CONTROL/ /C/; in this way, the line currently being operated on is deleted and the system returns to the Command State.

Example 1

```
AUTO
```

Line numbering starts with 10 (default value) and increases by 10 for each line (default value).

Example 2

```
AUTO .,30
```

Line numbering starts with the current line and increases by 30.

Example 3

```
AUTO 100
```

Line numbering starts with 100 and increases by 10.

Example 4

```
AUTO 150,
```

Line numbering starts with line 150. The 'interval' is the last one which has been specified during the Basic session. If 'interval' has not been specified the default value 10 is assumed.

Example 5

```
AUTO ,3
```

Line numbering starts with line 0 and increases by 3.

Example 6

```
AUTO 100,50
```

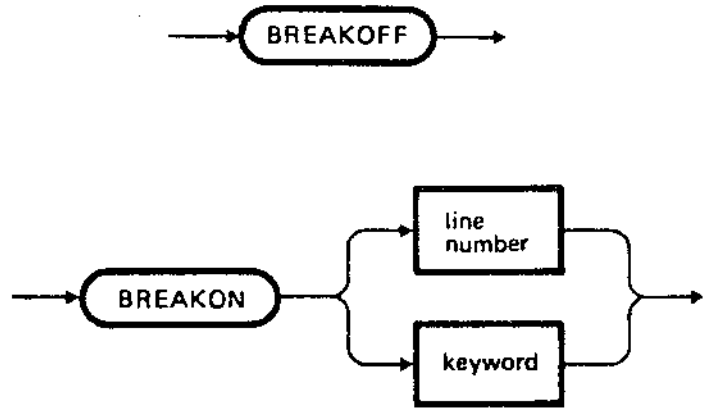
Line numbering starts with line 100 and increases by 50.

BREAKOFF/BREAKON

BREAKON interrupts program execution at a specified line number or keyword.

BREAKOFF disables use of the BREAKON command.

Only at TFORM time.

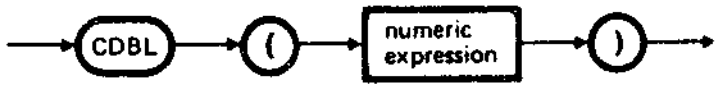


Characteristics

The programmer can use both form of BREAKON in the same program.
The last BREAKON defined replaces the old.
Break stops the program execution before the execution of the of statement required.
After a breakpoint, program execution is resumed by entering CONT.

CDBL

This function converts any numeric format to an argument in double precision.

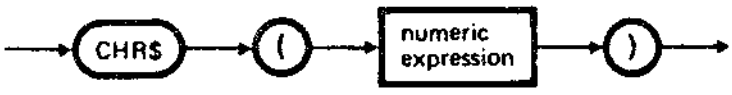


Example

```
10 A = 454.67
20 PRINT A; CDBL(A)
RUN
 454.67    4 454.670013427734
```

CHR$

This function returns the character whose ASCII decimal code is the value of the argument.



where:

numeric expression — is a real numeric expression that is rounded off to the nearest integer. It must be within the range 0 to 255 and it is interpreted as an ASCII decimal code.

Characteristics — The CHR$ function is frequently used to send a special character to the terminal or to the printer. For example, the character BEL(CHR$(7)) can be given at the prefix of an error message.

See Appendix C for ASCII decimal codes.

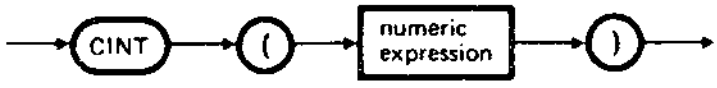
To convert an ASCII character into its corresponding numeric code see the ASC function.

Example

```
PRINT CHR$(66)

B
```

CINT

This function converts any numeric argument into an integer, by rounding its fraction to the nearest integer.



Characteristics

The argument of the CINT function must be in the -32768 and 32767 range.

If the fraction is >.5 it is rounded up, otherwise it is rounded down.

See the CDBL and CSNG functions for converting numbers into single and double precision formats, and the FIX and INT functions for convertion to integer format.

Example

```
PRINT CINT(45.67)
 46


PRINT CINT(45.45)
 45
```

CLEAR

Sets the numeric variables to zero and the string variables to null.

Characteristics

This command has no options except for entry of the command itself.

Resumes the execution of a program interrupted by a /CONTROL/ /C/, or a BREAKON command, or by a STOP or STON statement.

Only at TFORM time.

Characteristics

Program execution is resumed from the point where the interruption occurred.
Apart from the CONT command, execution can be resumed by using the GOTO statement in Direct Mode; this passes execution control to a specified line number.
CONT may be used to continue execution after an error.

The CONT command is normally used with a STOP, or STON/STOFF statement or with a BREAKON/BREAKOFF command for program debugging.
When execution is stopped, intermediate values may be examined and changed using Direct Mode statements.

CONT is invalid if the program has been edited during the break.

COS

This function returns the cosine of the argument.

Characteristics

The function argument is the size of an angle in radians. The cosine is calculated in double precision.

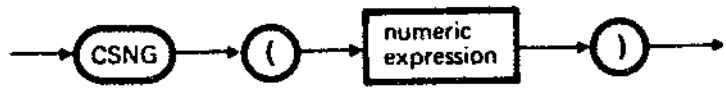
The numeric expression can be of any type. The result returned is in double precision. It is advisable to use a double precision numeric expression as input.

Example

```
PRINT COS(.4)

 .921060991671771
```

CSNG

This function converts a numeric type argument to a number in single precision.



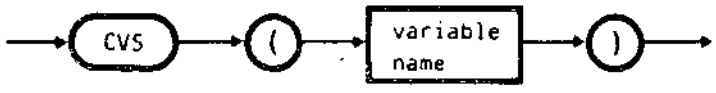
Characteristics

To convert numbers to integer or double precision formats, see the CINT and CDBL functions.

Example

```
10 A# = 975.3421
20 PRINT A# ; CSNG(A#)

RUN

 975.3421     975.342
```

The numeric values, that have been defined with the FIELD command using string variables, must be converted, using this function, before being transferred into internal VPL variables.



where:

variable name — name of the variable which identifies the numeric field to be converted.

Characteristics — Used at Visa time, this function can contain only variable names defined in the FIELD command.

The converted value may be assigned to integer type numeric variables, with single or double precision.

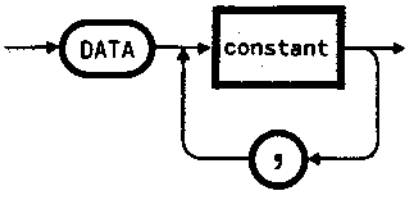
Used at TFORM time, this function is the same as the VAL function.

Examples

Assuming that the FIELD command refers to two numeric fields in the data record, we must specify:

```
....

50 FIELD 5 AS N1$, 10 AS N2$
60 LET X = CVS(N1$)
70 LET Y = CVS(N2$)

....
```

Stores the numeric and or string constants to be accessed by the READ statement(s).



where:

constant — is a numeric or string constant.
Numeric constants can be in any format, i.e. integer, fixed point, floating point....
String constants must be inclosed in double quotation marks only if they contain commas, colons, significant leading or trailing spaces.
The type of the constant must agree with the type of the corresponding variable in the READ statement, otherwise a syntax error will arise.

Characteristics

DATA statements are non executable, and can be placed anywhere in the program.
A DATA statement may contain as many constants (separated by commas) as will fit on a line.
Any number of DATA statements may be used in a program: the READ statement will access the DATA statements in order of line number and the data contained therein may be thought of as one contiguous list of items.

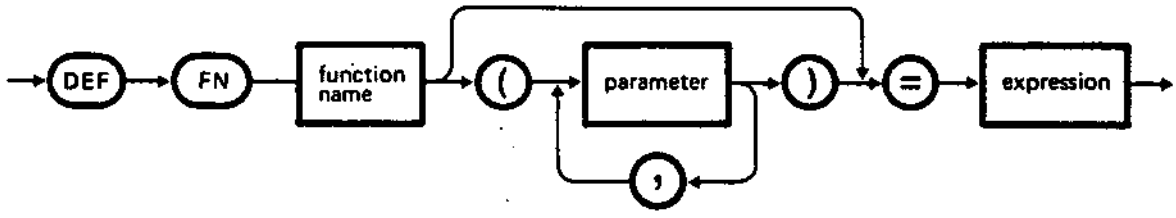
Example

```
110 DATA 3.08, 5.19, 12.04
120 DATA "DENVER," , COLORADO
```

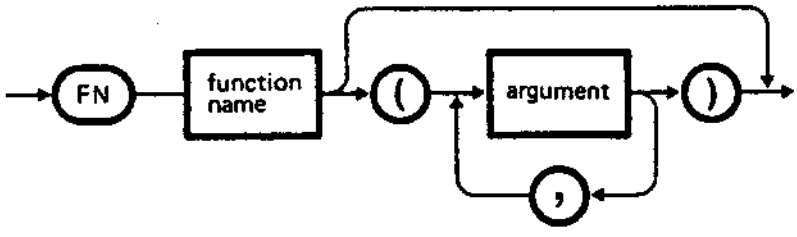
DEF FN

Defines a numeric or string type user-written function. It is limited to one line.

Function Definition



Calling the Function



where:

function name	is a valid variable name beginning with FN (numeric or string type names may be specified). No blanks may be inserted between FN and the name of the function. The first character of the name must be a letter. The computed value of the function will be of the same type as declared by the function name's structure (numeric or string).
parameter	is one or more variables that must be replaced by the value(s) of the corresponding arguments when the function is called. The association between arguments/parameters is of a positional type (i.e. the first parameter corresponds to the first argument and so on).
expression	is an expression which defines the operation to be executed by the function. The parameter names in the expression only define the function and do not affect program variables with the same name in any way. To render the program easy to read, the user should avoid giving the same name to parameters and variables. A variable name used in the expression which defines the function, may or may not appear in the list of parameters. If it does appear, it will be replaced by its corresponding argument. If it does not appear (global variable) the current value of the variable is assumed.
argument	is the effective value attributed to the corresponding parameter. Each argument can be a constant, a variable, an expression or an array element. The type of the argument(s) must agree with the type implicitely declared by the function name; if this does not occur a 'type mismatch' error is given. Note that if the computed value of a function is assigned to a variable, a numeric conversion is made if necessary (see NUMERIC CONVERSION in the chapter DATA).

Characteristics

If a user-defined function is invoked by another user-defined function, the function being invoked must be defined within the program itself in a previous position.

Example:
```
10 DEF FNA(X)=(SIN(X/5)*3.1)/180
20 DEF FNB(X)=(FNA(X)+SIN(X))*.5
```

A DEF FN statement must be executed before the function it defines may be called.
If a function is called before it has been defined, an "UNDEFINED USER FUNCTION" error occurs.

DEF FN is illegal in Direct Mode.

Example

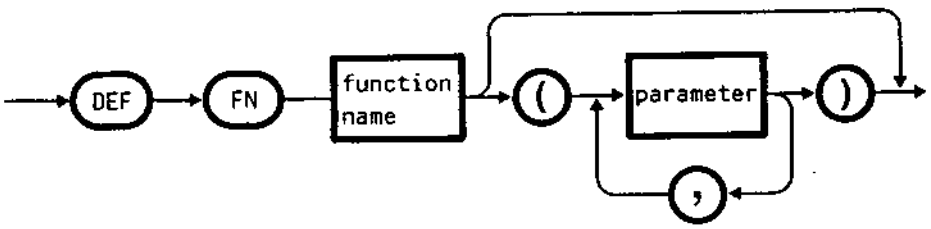
```
410 DEF FNAB(X,Y)=X 3/Y 2
420 T=FNAB(I,J)
```

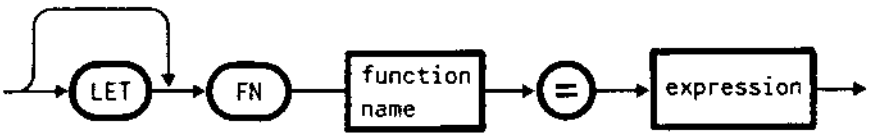
Program line 410 defines the FNAB function.
Line 420 invokes the function.

DEF FN/FNEND

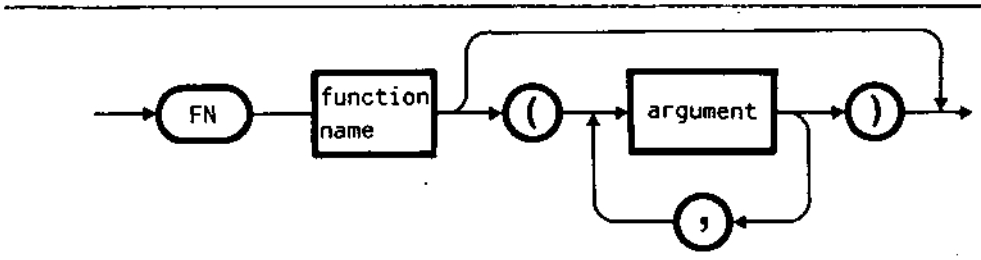
Defines a numeric or string type user-written function.
The function definition can occupy more than one line.

Function Definition.





Calling the Function



where:

function name	is a valid variable name beginning with FN (numeric or string type names may be specified). No blanks may be inserted between FN and the name of the function. The first character of the name must be a letter. The computed value of the function will be of the same type as declared by the function name's structure.
parameter	is one or more variables that must be replaced by the value(s) of the corresponding arguments when the function is called. The association between arguments/parameters is of a positional type (i.e. the first parameter corresponds to the first argument and so on).
program line	denotes one or more VPL statement(s) which can be used to define a subprogram that is executed whenever a reference to the function is made.
expression	defines the operation to be executed to assign a value to the function.
argument	is the effective value attributed to the corresponding parameter. Each argument can be a constant, a variable, an expression or an array element. The type of the argument(s) must agree with the type implicitely declared by the function name; if this does not occur a 'type mismatch' error is given. Note that if the computed value of a function is assigned to a variable, a numeric conversion is made if necessary (see NUMERIC CONVERSION in the chapter DATA).

Characteristics

The following should be usted:

- The parameter names only define the function and do not affect program variables with the same name in any may.
- A variable name used in the function defition may or may not appear in the list of the parameters. If it does appear, it will be replaced by its corresponding argument; if it does not appear (global variable) the current value of the variable is assumed.
- Within the function definition, there can be one or more LET statement to assign a value to the function.
- A function must be defined before being called, otherwise an "undefined user function" error occurs.
- References to user function(s) (single or multi-line) can appear in all the staments of a multi-line function definition.
- A multi-line function definition may not be defined within a multi-line function definition.
- The FOR/NEXT and WHILE/WEND cycle can be defined only within a multi-line function definition.
- The ERASE statement is not allowed within a multi-line function definition.

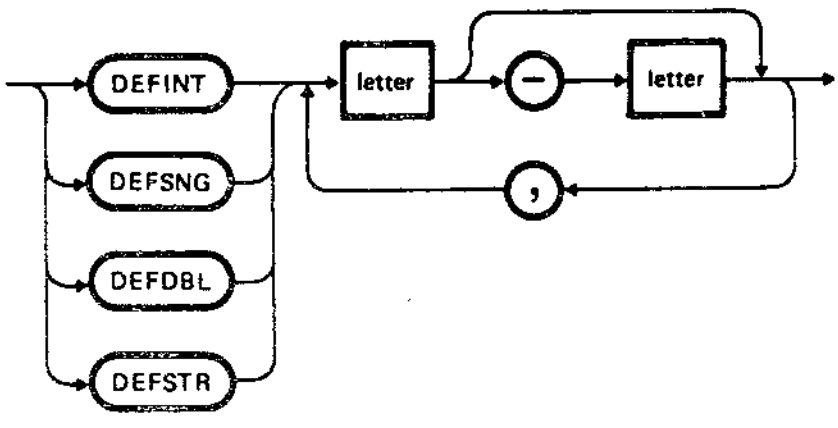
Example

```
400 DEF FNAB(X,Y)
405 X = X/Y
410 FNAB = X
415 FNEND
420 T = FNAB(I,J)
```

Program lines 400-415 define the FNAB function.
Line 420 invokes the function.

DEFINT/SNG/DBL/STR

Declare the type of all the variables whose names begin with a specified letter.



Characteristics

DEFINT declares the variables as integers.
DEFSNG declares the variables in single precision.
DEFDBL declares the variables in double precision.
DEFSTR declares the variables as strings.

Variable types must be declared before they may be used and are normally inserted at the start of the program.
If the user over-rides the type of a variable by adding a type-definition tag at the end of the name, this operation has precedence over a DEF type statement. If no type definition statement is carried out, VPL will consider all variables without type-definition tags in single precision.
If a variable is defined twice, the last definition is valid.

Example 1

```
10 DEFINT A-Z
```

All the program variables are integers.

Example 2

```
10 DEFDBL D
```

All the program variables beginning with the letter D are in double precision.

Example 3

```
10 DEFSTR S,U-W
```

All the program variables beginning with the letters S, U, V and W are string variables.

Example 4

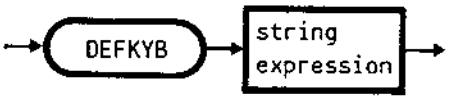
```
10 DEFINT A-Z
20 DEFDBL D
```

All program variables are integers except for variables beginning with the letter D which are in double precision.

DEFKYB

Allows to redefine physically the function keys use, by changing the default association between the function key and their codes.

Function Definition



where:

string expression — is a string of exactly 60 characters.
Each character must be an ISO character in the range 21...5B (kex), and it corresponds to a function key symbol.
The description which follows refers to the table below, which provides the default association between each position in the string and its ISO character.

BYTE POSITION IN THE STRING	FUNCTION KEY SYMBOL	ISO CHARACTER	HEXADECIMAL CODE
1	* P5	!	21
2	* P4	"	22
3	* P3	#	23
4	* P2	$	24
5	* P1	%	25
6	P5	&	26
7	P4	'	27
8	P3	(	28
9	P2	)	29
10	P1	*	2A
11	* S5	+	2B

12	* S4	'	2C
13	* S3	-	2D
14	* S2	.	2E
15	* EXIT	/	2F
16	* ↑	0	30
17	* ↓	1	31
18	* →	2	32
19	* ←	3	33
20	F16	4	34
21	F15	5	35
22	F14	6	36
23	F13	7	37
24	F12	8	38
25	F11	9	39
26	F10	:	3A
27	F9	;	3B
28	F8	<	3C
29	F7	=	3D
30	F6	>	3E
31	F5	?	3F
32	F4	@	40
33	F3	A	41
34	F2	B	42
35	F1	C	43
36	S5	D	44
37	S4	E	45
38	S3	F	46
39	S2	G	47
40	EXIT	H	48
41	DL	I	49
42	IL	J	4A
43		K	4B
44		L	4C
45	SEND	M	4D
46	SKIP	N	4E
47	DC	O	4F
48	IC	P	50
49	*CL.ERR	Q	51
50	↑	R	52
51	↓	S	53
52	→	T	54
53	←	U	55
54	→|	V	56
55	|←	W	57
56		X	58
57	CLEAR	Y	59
58	HOME	Z	5A
59	ERASE	[	5B
60		\	5C

In the table:

The key symbols marked with '*' represent the second keyboard level (that is, they must be entered at the same time as the /CONTROL/ key).

The positions left blank in 'FUNCTION KEY SYMBOL' correspond to function key not available to VISA.

The DEFKYB workes as follows:
the correspondence between each position in the string and its function key symbol is fixed (i.e, position 35 will always correspond to the /F1/ key).
The user can change the default association by changing the ISO character in the relevant position in the string.
For example, if the user wants the function key /F1/ to have the same functionality as the function key /HOME/, he has to put the character 'Z' in the 35 string position.

Characteristics

The user must specify a selected ISO character for all the positions in the string.

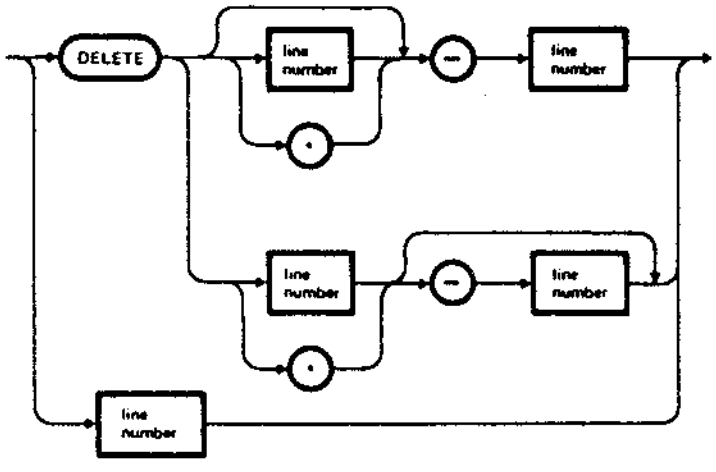
The 'blank' character must be used to disable the use of a function key.

The characters put in the string positions which do not have a corresponding function key symbol (that is position 43, 44, 56, 60) are not significant.

DELETE

Deletes program lines.

Only at TFORM time



Characteristics

VPL always returns to command level after a DELETE is executed.

If < line number > does not exist, an "ILLEGAL FUNCTION CALL" error occurs.

Example 1

DELETE .

The current line is deleted.

Example 2

500

or

DELETE 500

Line 500 is deleted.

Example 3 DELETE 100-200

All the lines from 100 to 200 inclusive are deleted.

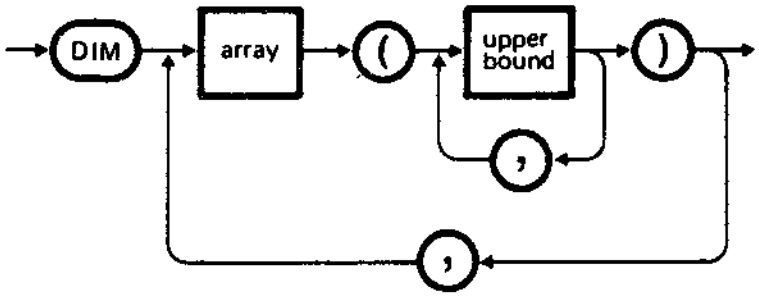
Example 4 DELETE -400

All the lines from the beginning of the program to line 400 inclusive are deleted.

Example 5 DELETE .-500

All the lines starting from the current line to line 500 inclusive are deleted.

Specifies the name of one or more arrays, the dimension of each array and the maximum value for each subscript.
It also sets all the elements of the specified arrays to zero and allocates the space in memory for the array.



where:

array — is an array name (any variable name is allowed).

upper bound — is any positive numeric constant or numeric variable with a positive value. If this value is not an integer it is rounded off to the nearest one. It establishes the maximum value for each subscript. The number of subscripts is determined by the number of upper bounds provided.

Characteristics — In VPL arrays can be defined with a maximum number of 10 subscripts.
The minimum index value for each subscript is 0.
If the user does not insert a DIM statement in the program, the array is created when VPL encounters one of its elements for the first time.
The number of subscripts is determined on the basis of the number found and the maximum value of each is

10.
For example, if the following appears in a statement:

```
AR1 (3,5,10)
```

the array AR1 is created with three subscripts and with a maximum value for each of 10.

This command is only used in Indirect Mode.

Example

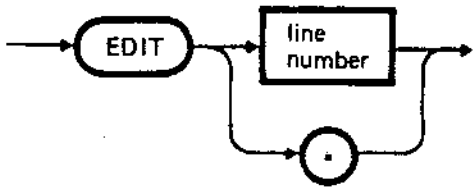
```
. . . . . .
50 DIM A(5), B$(20,30)
. . . . . .
```

Defines array A as a one-dimensional numeric array with an index between 0 and 5, (both numbers inclusive), and array B$ as a two-dimensional string array with its row index between 0 and 20 (both numbers inclusive) and its column index between 0 and 30 (both numbers inclusive).

Before redimensioning an array the definition must be cancelled using an ERASE command or the operation will result in the message "DUPLICATE DEFINITION".

Enters the Editor Mode at the specified line.

Only at TFORM time.



Characteristics

When the Editor Mode is activated, the line number and the whole line to be edited are displayed. The cursor is positioned at the beginning of the line. Parts of a line can be modified. The following keys can be used to modify the line of a program:

→ moves the cursor to the right of its current position.
If the cursor is already at the end of the line, the command is ineffective.

← moves the cursor to the left of its current position.
If the cursor is already at the beginning of the line, the command is ineffective.

→| moves the cursor to the end of the line.

|← moves the cursor to the beginning of the line.

|←| the cursor shifts one position to the left and the character on which it is positioned is deleted.
The Keyboard may have the /BS/ key instead, depending on the Keyboard type.

DC — the character, on which the cursor is positioned, is deleted; all the characters on the right shift one space to the left. The cursor position remains the same.

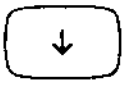
IC — enters Insert Mode. Any character that is keyed in is inserted at the current cursor position and will cause all the characters to shift one space to the right of the cursor.
Even the current cursor position will be shifted one space to the right.
During the Insert Mode all the keys described above may be used.
It is possible to exit from the Insert Mode by entering IC.

RESET — resets all characters to zero starting from the current cursor position.

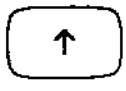
any character — any character is displayed at the current cursor position, and replaces the previous character ("tape over")

end of input key or carriage return key — saves all the changes made to the line and allows the user to exit from the Editor Mode.

/CONTROL/ /C/ — the system returns to Command Mode.
Any changes made whilst in the Editor Mode are lost.
The same will happen if an end of input key or carriage return is entered on a line with no characters.



changes made to the line are saved. The Editor Mode will operate on the next line.



changes made to the line are saved. The Editor Mode will operate on the previous line.

Characteristics — If a Syntax Error occurs during program execution, the Editor Mode automatically operates on the line that caused the error.

Example:

```
10 K=2(4)
RUN
? SYNTAX ERROR IN
10.........
```

When changes have been made and the end of input key has been entered, the line is reinserted, and the program execution continues.
Variable values are preserved for examination.

END

This command is used to terminate execution of the validation routine and to return control to VISA. It can also be used to force continuation of VISA operations from a specific field.

where:

variable name — Specifies the name of a field (defined with the FIELD command); VISA operations will be continued from this field.

Characteristics — The END command at the end of the validation program is optional.

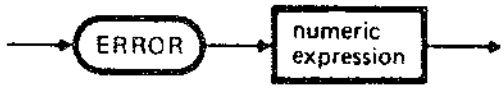
When no 'variable name' has been specified or if the program terminates without the END command, VISA restarts execution following the order of the subform being executed.

The validation routine can be repeated on the current field by specifying the SAME keyword instead of the field name.

At TFORM time, execution of an END statement always causes a return to the Command Status and any 'variable name' is ignored.

ERROR

This command is mainly used at VISA time, and allows an application program to return immediately to VISA, with error code generation.
The error may be one of those given in the VPL error list, or may be a user-defined code.
When control is returned to the application, the data area relating to the current subform is passed to the application program.



---

where:

numeric expression — specifies the error code value which is assigned to ERROR.
It must be between 0 and 99 for COBOL applications, and 0 and 255 for PASCAL applications.

Characteristics — The VPL error are listed in the Appendix, with their meaning.
Each application program can define its own error list, in addition to the general VPL list.
In this case, the user must use higher code numbers than those in VPL environment. (It is advisable to start with the highest values allowed in order to maintain compatibility when implementing the VPL error list).

At TFORM time if the value of "numeric expression" matches a VPL error code, ERROR simulates the occurence of this error and the corresponding error message is displayed.
If ERROR specifies a code number not included in the list of VPL errors the message "UNPRINTABLE ERROR" is displayed.

This command is used to execute the semi-compilation of the validation program entered, to exit from VPL and return to TFORM.
If the compilation gives a positive result the validation program is catalogued on disk.

Only at TFORM time.

Characteristics

When the compilation is executed the following applies:

- If no errors are found both the source and the compiled program are written on disk.
  The previous version of the program, if any, is then updated.
  The message:
  VPL PROGRAM EXTENT..... is displayed, providing the length of the program.

- If invalid entry points are found they are displayed, the compilation is completed and both the source and the compiled program are written on disk.
  The previous version of the program, if any, is then updated.

- During compilation, reference to line numbers can be made (GOTO, GOSUB,....statements).
  If undefined lines are found they are displayed and the compilation is aborted.
  The source program entered is written on disk

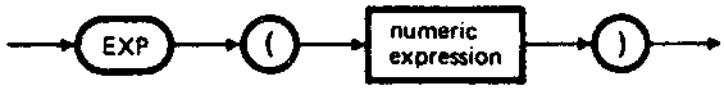
(the source previous version, if any, is then updated);
the compiled program is nct written as the compilation has aborted: the previons version, if any, is maintained.

- If the compiled program exceeds 64K bytes what said in the previons point applies.

- If a disk error arises when writing on disk, the program is not written and the previous version, if any, is maintained.

The TFORM key /SAVE/ produces the same effect as the EXIT statement.

This function raises the power of the constant 'e'. The exponent of the power is the value of the argument.



Characteristics

Computation of EXP is carried out in double precision.

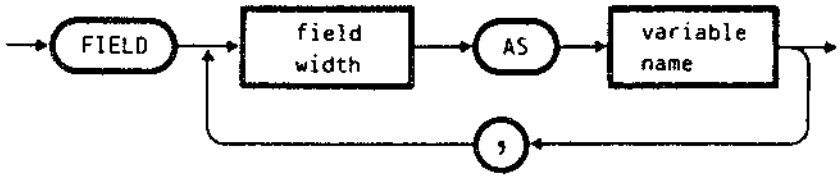
The numeric expression can be of any type. The result provided is in double precision. It is advisable to use a double precision numeric expression as input.

Example

```
10 X=5
20 PRINT EXP(X-1)

RUN

 54.5981500331438
```

FIELD

This command is used to identify the fields of a form in the form data record, by indicating their length and also assigning a name.



where:

field width — specifies the space occupied by a field in the form data record.

AS — keyword which logically links the space occupied by a field and its identifying name in VPL.

variable name — name, which always consists of a string variable, that identifies a field.

Characteristics

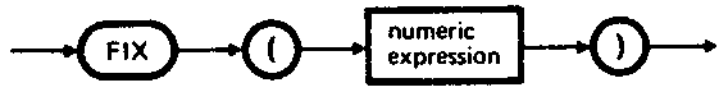
The FIELD command must always precede any other command which performs input or output operations on the data record.
Only FIELD variables can be used for data interchange between VPL and VISA.

The FIELD command may used several times in the same validation program, providing different views of the same data record; all the variables used to identify the individual fields (or their grouping together or division) are always recognised inside the program.

The total number of bytes indicated with a FIELD command must not exceed the total length of the fields making up the form, otherwise a "FIELD OVERFLOW" messages will be displayed at VISA time.

See the chapter on 'THE VALIDATION PROGRAM'.

This function returns the integer part of the argument.



Characteristics

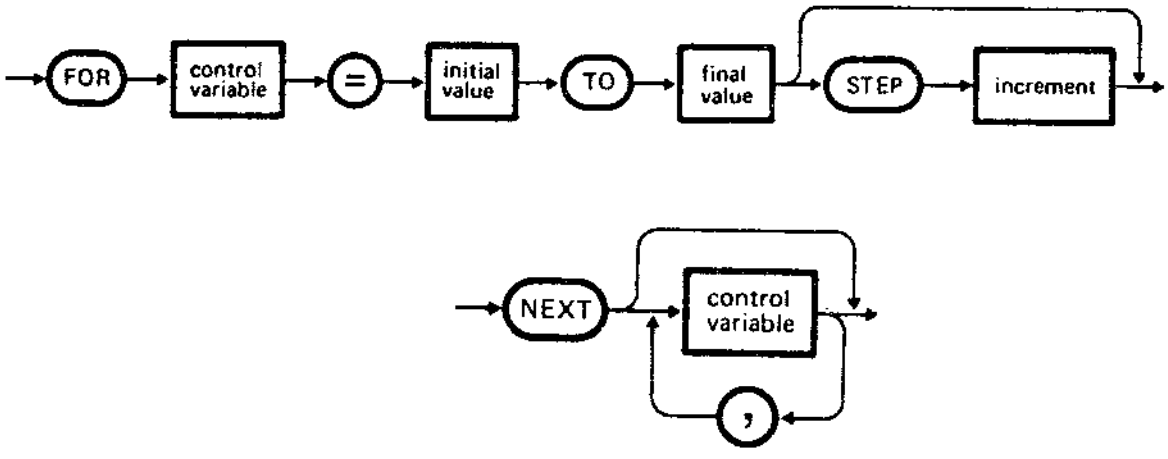
FIX(X) is equivalent to SGN(X)* INT(ABS(X)).
By comparison to INT, FIX does not provide the immediately lower value, in the case of negative arguments.

Examples

```
PRINT FIX(58.75)

 58


PRINT FIX(-58.75)

-58
```

FOR/NEXT

Allows a series of statements to be executed in a loop for a specified number of times.



where:

control variable — is a numeric variable (integer or in a single precision) that is used as a counter. The control variable names specified in the FOR and NEXT statements must be the same. A list of control variables can follow the word NEXT; however a NEXT statement can be on its own. If no control variable follows the word NEXT, the NEXT statement will be automatically associated with the last FOR statement executed.

Characteristics

Positive Increment.
If the value of the increment is positive, the FOR/NEXT loop is executed until the value of the control variable is not greater than the final value.

Example:

```
10 K=10
20 FOR I=1 TO K STEP 2
30 PRINT I;
40 K=K+10
50 PRINT K
60 NEXT

RUN

 1   20
 3   30
 5   40
 7   50
 9   60
```

In this case, the loop is repeated five times.
If the value of the increment is positive and if the initial value is higher than the final one, the loop is not executed.

Example:

```
10 J=0
20 FOR I=1 TO J
30 PRINT I
40 NEXT I
50 PRINT "Exit from loop"

RUN

Exit from loop
```

Negative Increment.
If the value of the increment is negative, the loop is executed until the value of the control variable is not less than the final value.

Example:

```
10 FOR I%=1 TO -10 STEP -3
20 PRINT I%;
30 NEXT I%
40 PRINT "Exit";
50 PRINT " CONTROL VARIABLE=";I%

RUN

1 -2 -5 -8 Exit
CONTROL VARIABLE=-11
```

In this case, the loop is executed four times.
When it is terminated, the control variable keeps the last value that has been assumed (-11) which is displayed with statement 40.
If the value of the increment is negative and if the initial value is less than the final one, looping is not carried out.

Example:

```
10 FOR K%=1 TO 10 STEP -2
20 PRINT K%
30 NEXT K%
40 PRINT "Exit"
50 PRINT " CONTROL VARIABLE=";K%

RUN

Exit
CONTROL VARIABLE=1
```

Zero Increment.
If the value of the increment is equal to zero, the loop is repeated continuously (unless the initial and the final values coincide; in this case, the loop is not executed).

Example:

```
100 FOR A%=1 TO 30 STEP 0
110 PRINT A%
120 NEXT A%

RUN

1
1
.
.
.
```

The user must enter /CONTROL/ /C/ to interrupt execution.

Nested Loops.
Two or more FOR/NEXT loops can be nested on condition that the internal FOR/NEXT loop is completely included in the outside one. The following loops, for example, are correct:

```
 50 FOR I = 1 TO 10
100 FOR J = 2 TO 20
200 NEXT J
300 NEXT I
```

whereas the following are not:

```
 50 FOR I = 1 TO 10
100 FOR J = 2 TO 20
150 NEXT I
200 NEXT J
```

Two or more FOR/NEXT nested loops must not have the same control variable.
For every FOR statement, there must be a corresponding NEXT statement.
If all nested loops have the same final period, a single NEXT statement, together with a list of all the control variables separated by a comma may be used to group all loops.
In a series of nested FOR statements, if one or more NEXT statements are missing, the error 'FOR WITHOUT NEXT' is always signalled on the first FOR even if it is correctly closed with the relative NEXT.

GOSUB/RETURN

GOSUB calls a subroutine by passing control to the first line number of the subroutine.
RETURN transfers control to the first statement following the last GOSUB executed.

where:

ine number — is the first line number of a subroutine.

Characteristics

A subroutine can begin with any statement (excluding NEXT). A subroutine should always begin with REM (or with a statement that terminates with a comments field).
A subroutine can end with a RETURN statement, which must be executed last, since it is the only statement which allows control to be passed back to the main program.

A subroutine can have more than one RETURN statement when it is structured so as to have several branches, each of which request that control be passed back to the main program.

A subroutine can be called in any point of the program, any amount of times; if a program calls one of its own subroutines more than once when the subroutine has been executed, control is passed on to the statement following the GOSUB executed last.

A subroutine can be inserted at any point in the program, but it is recommended to write subroutines one after another, at the end of the program and close the program (before the start of the first subroutine) with either an END or GOTO or STOP statement.

One subroutine can call another subroutine. The number of "nested" subroutines that may be active at the same time is limited by available memory.

A subroutine can access all program variable. Infact, all the variables that are defined in the "main" program are also known to the subroutine. Thus, these can operate without restriction on each variable (and also change their value).

GOTO

Passes control to the specified program line.

Characteristics

If the statement specified by GOTO is not executable (e.g. it is a REM statement), control is passed to the first executable statement following the specified line number. Use of the GOTO statement in Command State ensures that program execution begins at the specified line number. This allows values to be assigned to program variables in the Command State. This technique can be used in the program debugging phase.

HEX$

This function converts the decimal argument into its corresponding hexadecimal value according to the ASCII table.

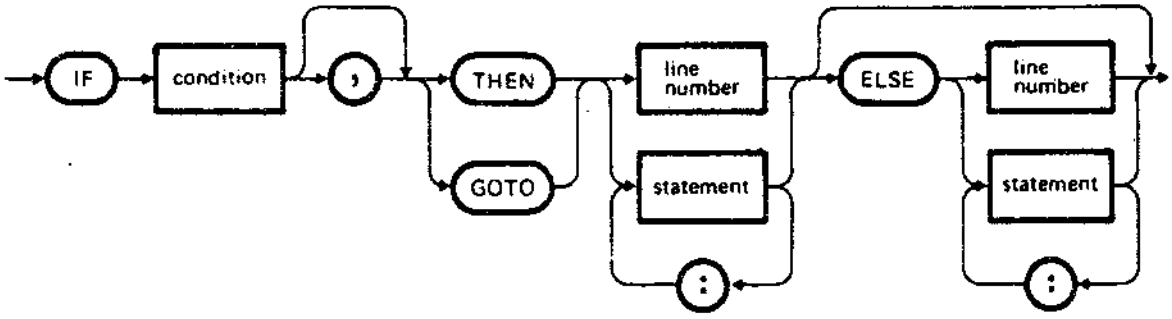
where:

numeric expression — is a numeric expression that is rounded to the nearest integer.

Characteristics — See the function OCT$ for octal conversion.

IF...GOTO...ELSE
IF...THEN...ELSE

Both these statements pass control, conditionally, to a specified statement.



where:

condition — can be one of the following expressions:

- numeric
- relational
- logical

Characteristics — VPL determines whether the condition is either true or false, by checking if the result (numeric) is different from zero (true condition) or equal to zero (false condition). For this reason, it is possible to check whether the value of a variable is equal to, or different from zero, by specifying the variable name as a "condition".
If the condition is true, control is passed to the statement whose line number is specified after GOTO (or THEN), or to the first statement after THEN.
If the condition is false and the ELSE clause is

omitted, control is passed to the first executable statement following the statement IF... GOTO or IF...THEN.
If the condition is false and the ELSE clause is included, control is passed to the statement whose line number is specified after ELSE or to the first statement that follows ELSE.
After executing the statement/s that follows ELSE, control is passed to the first executable statement.

The IF...GOTO...ELSE or IF...THEN...ELSE statements can be nested.
This process is limited to a VPL line (255 characters).

Example:

```
IF X>Y THEN PRINT "GREATER" ELSE IF Y>X THEN PRINT
"LESS THAN" ELSE PRINT "EQUAL"
```

is a valid statement

If the statement does not contain the same THEN and ELSE clause number, then the most internal THEN clause corresponds to each ELSE.

Example:

```
IF A=B THEN
IF B=C THEN
PRINT "A=C"
ELSE PRINT "A<>C"
```

"A<>C" is not printed when A is different from B.

If an IF...GOTO...ELSE or an IF...THEN...ELSE statement is used to check the result of a floating point computation, as the internal representation of the value may not be exact, checking is carried out in the range of the accuracy defined, i.e. to test whether variable A is equal to 1.0 the following must be used:

```
IF ABS(A-1.0)<1.0E-6 GOTO...
```

or

```
IF ABS(A-1.0)<1.0E-6 THEN...
```

This test returns true if the value of A is equal to 1.0 with a relative error of less than 1.0E-6.

Example 1

```
50 IF I THEN A=1000
```

1000 is assigned to variable A if I is not equal to 0.

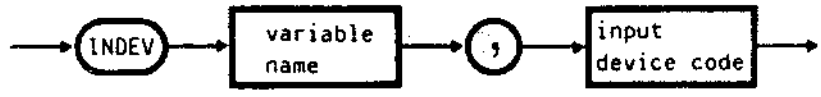
Example 2

```
70 IF (I<30) AND (I>5) THEN
   A=B+C:GOTO 350
80 PRINT "OUT OF RANGE"
       .
       .
       .
```

A test determines whether I is greater than 5 and less than 30. If I is in this range the value of A is calculated and execution passes to line 350. If I is not in this range, execution continues from line 80.

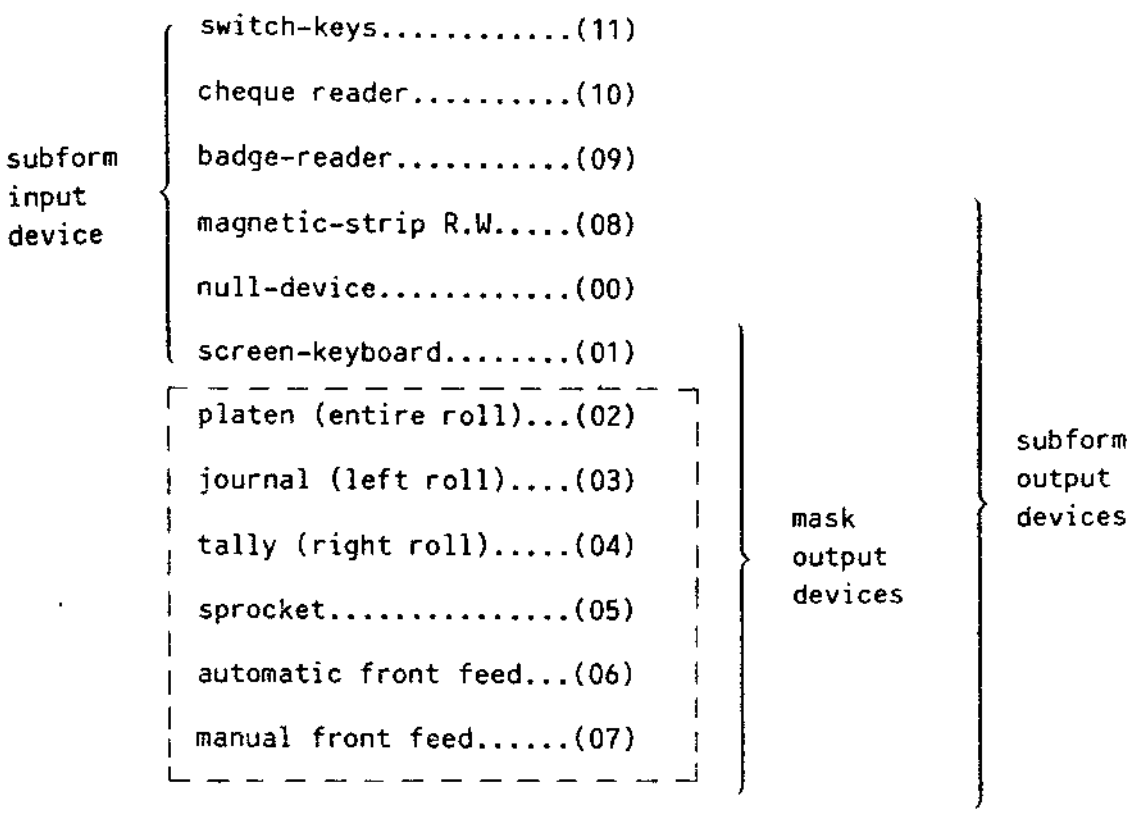
INDEV

This command allows the input device for the field indicated to be redefined.



where:

variable name	indicates one of the fields defined by the command FIELD
input dev. code	code of the device to be redefined for the above field.
Characteristics	A list of the device code is given on the following page, organised logically.

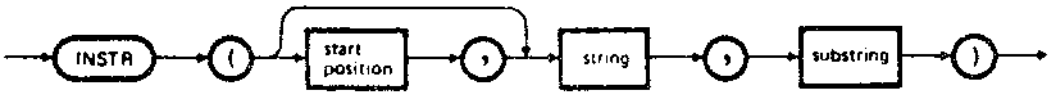


The device codes inside the dotted area refer to the printer.

One of the devices defined for the field at TFORM time can be specified.

INSTR

Searches for the first substring in a string and returns the position in which it has been found.



where:

start position — is a numeric expression rounded off to the nearest integer which specifies where the search must start. Its value must be between 1 and 255. If omitted, 1 is assumed by default.

string — is a string expression or string variable or string constant or array element, whose value matches the string to be found.

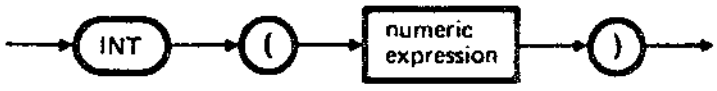
substring — is a string constant or string variable or string expression or array element, which is being searched for.

Characteristics — If start position > LEN (string), the value returned by this function is 0.
In the case of a null string (''''), the value returned is 0.
If substring is not found, the function value is 0.
If substring is a null string and start position is specified, the function value is equal to that of start position.
If substring is a null string and start position has not been specified, the function value is 1.

Example

```
10 X$ = "ABCDEB"
20 Y$ = "B"
30 PRINT INSTR(X$,Y$);INSTR(4,X$,Y$)

RUN

 2  6
```

This function returns the integer part of the argument. If the argument is negative, it returns the integer which is just less than the argument.



Characteristics

Note the difference between INT and FIX. With negative values, the returned value for INT is always less than or equal to the argument, whilst for FIX it is always greater than or equal to the argument.
Also compare with CINT.

Examples

```
PRINT INT(99.89)
  99
PRINT INT(-12.11)
-13
```

LEFT$

This function returns a substring by extracting the amount of characters equal to the assigned length from an assigned string; this is done starting from the left.

where

string — is a string expression whose content is the string from which the substring is to be extracted.

length — is a numeric expression rounded off to the nearest integer (from 0 to 255) whose value represents the length of the string required.

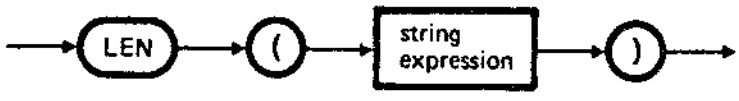
Characteristics — If the value of the parameter length is 0, the null string is returned.
If length >=LEN (string), the entire string is returned.

Example

```
10 A$ = "BASIC LANGUAGE"
20 B$ = LEFT$(A$,5)
30 PRINT B$

RUN

BASIC
```

LEN

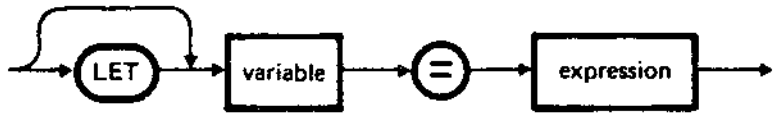
This function calculates the length of a given string by counting all the characters (printable or otherwise) and the blanks.



Example

```
10 X$ = "PORTLAND, OREGON"
20 PRINT LEN(X$)

RUN

 16
```

Assigns the value of an expression to a variable.



Characteristics

The LET keyword is optional. To assign an expression to a variable name, the equal sign is used. Simultaneous assignments are not allowed.

Example 1

```
LET K = 1.5
```

The value 1.5 is assigned to numeric variable K.

Example 2

```
LET X = K + 2
```

The value of the numeric expression K + 2 is assigned to the numeric variable X

Example 3

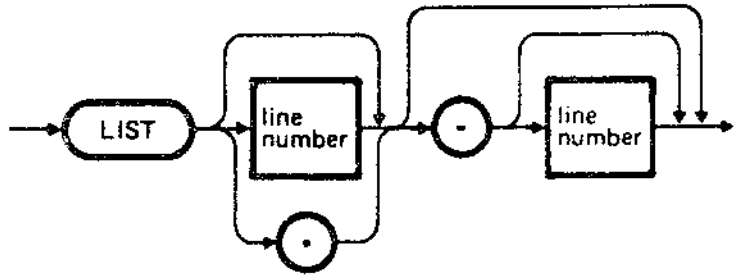
```
A$ = "ABC"
```

The value of the string constant "ABC" is assigned to the string variable A$.

LIST

Displays a list of all or part of the program stored in the memory.

Only at TFORM time.



where:

line number — specifies the number of the list's beginning or end.

Characteristics — The LIST command's applications are shown in the examples below.
A list can be interrupted by /CONTROL/ /C/.

Example 1 — LIST

The program held in the memory is listed.

Example 2 — LIST 500

Line 500 of the program in the memory is listed.

Example 3 — LIST 150-

Program lines from 150 to the end of the program are listed.

Example 4

```
LIST -1000
```

The program is listed from the beginning to line 1000.

Example 5

```
LIST 150 - 1000
```

All the lines between 150 and 1000 inclusive are listed.

Example 6

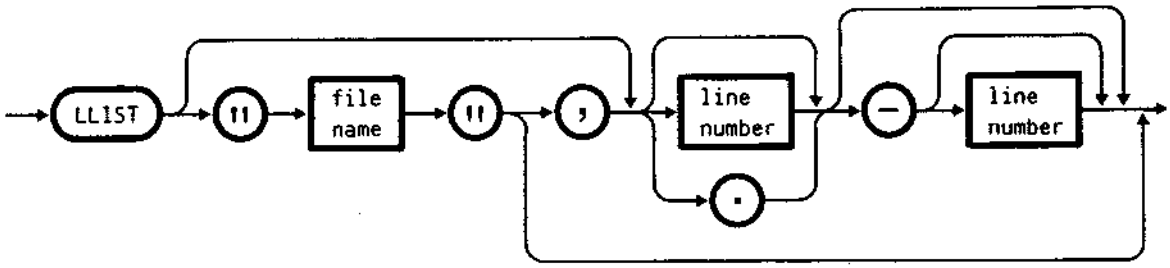
```
LIST.-300
```

All the lines from the current line to line 300 inclusive are listed.

LLIST

Lists all or part of the program currently stored in memory at the line printer or appends it to a file.

Only at TFORM time.



where:

file name	is the name of the file to which all or the specified part of the program is to be appended. If this file does not exist it is automatically created under the current directory.
,	must follow the file name when a line number is specified.
line number	indicates the first and/or last line in the program to be listed.
.	indicates that the listing is to start at the current line.
-	is a sign separating the first and last line numbers of the part of the program to be listed.

Characteristics	When the requested list has been printed/appended, the system returns to Command Status.
Examples	See the examples of the LIST command.

Allows to read a VPL program from an external file.

Only at TFROM time.

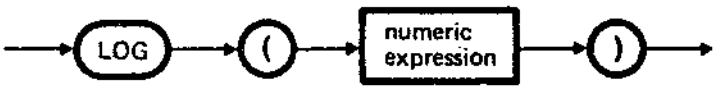
where:

listing file — specifies the external file which contanis the VPL program to be read.
A line statement is read from the file and it is either stored (if the line has a line number) or executed (if the line has not a line number) as if the stament was input from the keyboard..
Note that if a VP program is already in memory a merge is executed.
The file is read until the end of file, after which the VPL allows to input again from the keyboard.

Characteristics — By using the LOAD command, the user can prepare a VPL program with the standard system EDITOR and then load it.

LOG

This function calculates the natural logarithm. Computation is carried out in double precision.



where:

numeric expression — is a numeric expression. Its value must be positive.

Characteristics — On the basis of the equation $\log_a x = \log_e x / \log_e a$ the logarithm can be calculated in base 10 (or in any other base).

The numeric expression can be of any type; but the result is always in double precision. It is advisable to use a double precision numeric expression as input.

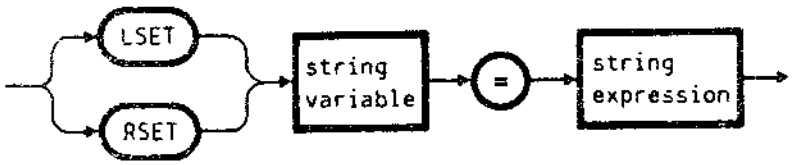
Example

```
PRINT LOG(45/7)

 1.86075230892586
```

LSET/RSET

LSET aligns a string value on the left in a string variable.

RSET aligns a string value on the right in a string variable.



---

where:

tring variable — is the name of a string variable.

string expression — is the string to be left or right justified in a given field. The string expression parameter cannot contain concatenation operations.

Characteristics — If the contens of the variable string do not occupy all the available space, the remainer is filled with blanks.
If the contens exceed the available space they are clipped.

Characteristics If 0 is assigned, the following values will be returned:

Code	Class/VISA action	Description
0	No action	The application program has already positioned on the first field of the form
1	Class = LIKE ENTER	The current field has been closed with an ENTER class functions key
3	Class = TERMINATION	Input operations have been ended with a TERMINATION class functions key
4	Action= HOME	The HOME key has been used to position on the field to be validated
5	Action= BACK-FIELD	The BACK-FIELD key has been used to position on the field to be validated
6	Action= RETURN	The RETURN key has been used to position on field to be validated
7	Action= SKIP	The SKIP key has been used to position on the field to be validated