NOTED
JUN 15 1987
G. L. M.

North American Response Centers

# HP 3000 APPLICATION NOTE #29

# A Programmer's Guide to VPLUS/3000

HEWLETT PACKARD

JUNE 01, 1987
Document P/N 5958-5824R2724

**HP 3000 APPLICATION NOTES** *are published by the North American Response Centers twice a month and distributed with the Software Status Bulletin. These notes address topics where the volume of calls received at the Centers indicates a need for addition to or consolidation of information available through HP support services. You may obtain previous notes (single copies only, please) by returning the attached Reader Comment Sheet listing their numbers.*

| Note # | Published | Topic |
|---|---|---|
| 3 | 4/01/86 | *HP 3000 Plotter Configuration Guide* |
| 4 | 4/15/86 | *HP 3000 Printer Configuration Guide - Revised* |
| 5 | 5/01/86 | *MPE System Logfile Record Formats* |
| 6 | 5/15/86 | *HP 3000 Stack Operation* |
| 7 | 6/01/86 | *COBOL II/3000 Programs: Tracing Illegal Data* |
| 8 | 6/15/86 | *KSAM Topics: COBOL's Index I/O; File Data Integrity* |
| 9 | 7/01/86 | *Port Failures, Terminal Hangs, TERMDSM* |
| 10 | 7/15/86 | *Serial Printers - Configuration, Cabling, Muxes* |
| 11 | 8/01/86 | *System Configuration or System Table Related Errors* |
| 12 | 8/15/86 | *Pascal/3000 - Using Dynamic Variables* |
| 13 | 9/01/86 | *Terminal Types for HP 3000 HPIB Computers - Revised* |
| 14 | 9/15/86 | *Laser Printers - A Software and Hardware Overview* |
| 15 | 10/01/86 | *FORTRAN Language Considerations - A Guide to Common Problems* |
| 16 | 10/15/86 | *IMAGE: Updating to TurboIMAGE & Improving Data Base Loads* |
| 17 | 11/01/86 | *Optimizing VPLUS Utilization* |
| 18 | 11/15/86 | *The Case of the Suspect Track for 792X Disc Drives* |
| 19 | 12/01/86 | *Stack Overflows: Causes & Cures for COBOL II Programs* |
| 20 | 1/01/87 | *Output Spooling* |
| 21 | 1/15/87 | *COBOLII and MPE Intrinsics* |
| 22 | 2/15/87 | *Asynchronous Modems* |
| 23 | 3/01/87 | *VFC Files* |
| 24 | 3/15/87 | *Private Volumes* |
| 25 | 4/01/87 | *TurboIMAGE: Transaction Logging* |
| 26 | 4/15/87 | *HP 2680A, 2688A Error Trailers* |
| 27 | 5/01/87 | *HPTrend: An Installation and Problem Solving Guide* |
| 28 | 5/15/87 | *The Startup State Configurator* |


NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This document contains proprietary information which is protected copyright. All rights are reserved. Permission to copy all or part of this document is granted provided that the copies are not made or distributed for direct commercial advantage; that this copyright notice, and the title of the publication and its date appear; and that notice is given that copying is by permission of Hewlett-Packard Company. To copy otherwise, or to republish, requires prior written consent of Hewlett-Packard Company.

Copyright © 1987 by HEWLETT-PACKARD COMPANY

# A Programmer's Guide to VPLUS/3000

## I. INTRODUCTION

VPLUS/3000 is an HP tool used by programmers to design screens and implement an online application to collect and display information. With VPLUS, you are able to design screens which prompt the user for input in familiar business terms. VPLUS/3000 intrinsics are available to help you develop a customized application.

This note is intended for programmers who want to understand the purpose of each intrinsic and the sequence of calls. For each intrinsic, a discussion of the passed parameters, some programming tips, and some details of what the intrinsic does is presented. Also included is a summary of VPLUS/3000 enhancements which can make an existing or new application easier to use. A brief synopsis is presented on each enhancement. For more details on their use, please refer to the *HP Data Entry and Forms Management System (VPLUS/V) Reference Manual* (Part No. 32209-90001). Note that in the remainder of this paper this manual will be referred to as the *VPLUS/V Reference Manual.*

This Application Note covers two topics to help you code your application.

1. VPLUS/3000 Intrinsic Calls
2. VPLUS/3000 Enhancements

This note assumes that the reader has some basic knowledge of the HP Data Entry and Forms Management System (VPLUS).

## II. VPLUS/3000 INTRINSIC CALLS

ENTRY.PUB.SYS is an example of an application which uses VPLUS/3000 intrinsic calls. This program accepts data from a user and writes the information to an MPE file. The MPE file contains information in the user label for use by ENTRY.PUB.SYS.

Since ENTRY.PUB.SYS does not meet the needs of some business environments, many programmers develop customized programs to write the data to a data base, an MPE file or a KSAM file; or to perform edits on the data. An understanding of the purpose of each intrinsic and the sequence in which the intrinsics are called is crucial to a successful program. The intrinsics can be logically grouped into one of six functions. An application program must use these functions in this sequence:

1. Opening Files
2. Preparing and Showing the Screen
3. Reading Data from the Screen

4. Editing

5. Returning Data to the Program

6. Closing Files

*Illustration 1* in the appendix of this note is a flowchart of the VPLUS/3000 intrinsic calls. The flowchart illustrates the sequence of calls that an application program may follow. It groups the intrinsics into one of the six functions listed above. Note that it does not include examples of local form storage, softkey labeling, or interfacing with VPLUS/3000 batch files.

## 1. Opening Files

Before a VPLUS/3000 application can run, it must open the necessary files. VOPENFORMF is called to open a formsfile. VOPENTERM is called to open a terminal. Note that VPLUS/3000 requires an HP block mode terminal. For a list of the supported terminals, refer to Appendix G of the *VPLUS/V Reference Manual.*

### a. VOPENFORMF

*Purpose: to open the formsfile to be used by the application.*

VOPENFORMF should be the first VPLUS/3000 intrinsic call executed in an application program. This intrinsic opens the formsfile specified in the call. Parameters passed to VOPENFORMF are the comarea array and the formsfile name. The comarea array is a 60 word communication area between the program and VPLUS/3000. The comarea array contains assorted fields to be set before an intrinsic call. One important field is CSTATUS or the status word. This is set on the completion of each VPLUS/3000 intrinsic call. The programmer should check this status after each intrinsic call to determine the success or failure of the call. If the status word is not zero, subsequent VPLUS/3000 intrinsic calls are not executed. If the program uses local forms storage or labeled softkeys, the FORM'STORE'SIZE and LABEL'OPTION words of the user's COMAREA must be set prior to calling VOPENFORMF. VOPENFORMF allocates additional space on the user's stack based on the values of these fields. Other words in the comarea array that need to be set prior to VOPENFORMF's call are (1) LANGUAGE which contains the language code of the calling program and (2) COMAREALEN which contains the length in words of the comarea array. In addition, for BASIC/3000 programs, USRBUFLEN should specify the number of words needed for the comarea extension on the stack.

When VOPENFORMF is executed, it opens the formsfile and allocates space on the stack for the comarea extension. The comarea extension is used by VPLUS/3000 to protect data between calls, to provide a buffering mechanism so that terminal reads are transparent to the user, and to aid in directory searches for formsfile information. *Illustration 2* in the appendix of this note contains a layout of the tables which are in the comarea extension. For most programming languages, the comarea extension resides in the DL-DB area of the stack. For BASIC/3000, the comarea extension is declared as a variable of the program and resides with the global variables.

## b. VOPENTERM

*Purpose: to open the terminal as a file and set the terminal for block mode.*

Another intrinsic which performs an open function within VPLUS/3000 is VOPENTERM. This intrinsic is called after VOPENFORMF. VOPENTERM does exactly what its name suggests. It opens the terminal as a file. In addition it sets the terminal in block mode and initializes the user function keys to contain the escape sequences which VPLUS/3000 expects.

Two parameters are passed to VOPENTERM: COMAREA and TERMFILE. COMAREA is the comarea array which VPLUS/3000 uses to communicate with the program. The words within the comarea array that VOPENTERM requires before the call are: CSTATUS, LANGUAGE, COMAREALEN and FORM'STORE'SIZE. These words were set at the time VOPENFORMF was called and therefore do not need to be reinitialized. Upon the completion of the call, VOPENTERM returns additional information (such as CSTATUS) in the comarea array which indicates the success or failure of a call. It also returns FILERRNUM which is the file system error if it could not open the terminal; FILEN which is the MPE file number of the terminal file; IDENTIFIER which contains a unique terminal ID assigned to a specific terminal; and ENVIRON which contains the logical device number of the terminal.

TERMFILE specifies a file name which is opened. This file defaults to the logical device from which the program is being run.

When debugging your VPLUS/3000 application, the messages displayed from your programs are only displayed within the unprotected fields on a form. For debugging purposes, it is possible to redirect the screen to another logical device so that an error message sent to $STDLIST is displayed in its entirety.

To redirect the VPLUS/3000 screen to another LDEV, you should:

1. Ensure that the other terminal's LDEV number is AVAIL by issuing a :SHOWDEV command. The terminal must be logged off. Do not use the keyboard or the terminal becomes unavailable at that time. The terminal must be configured at the same baud rate as the port or garbage will print when VPLUS/3000 displays the screen. This is because no speed sensing is done when the screen is routed to the alternate terminal.

2. Issue a file equation as follows:

   :FILE {termfilename};DEV=xx

   {termfilename} is the name used in your call to VOPENTERM; 'xx' is the LDEV of the terminal to which you are redirecting the screens.

Another method that can be used in your application is to redirect the display of the error message from the program to another device. This allows the screen to be displayed on the terminal from which you are running the program, but the error messages will be directed elsewhere. This method of redirecting $STDLIST is the easiest to use.

1. Issue a file equation to the device where you would like the displays directed, for example:

```
: FILE TEST;DEV=LP
    or
:BUILD TEST;REC=-132,1,F,ASCII;DEV=DISC
:FILE TEST,OLD;DEV=DISC
    or
:FILE TEST;DEV={LDEV # of terminal, which must be in AVAIL state}
```

2. After setting the file equation in step 1, run the program as follows:

:RUN {vprog};STDLIST=*TEST

## 2. Preparing and Showing the Screen

The next stage in a VPLUS/3000 application program is to prepare and show the screen. Preparing the screen to be displayed involves retrieving the form definition and initializing fields with the desired values. Optionally it may include initializing the terminal window with a message and labeling the function keys. The intrinsics which prepare the screen to be shown are VGETNEXTFORM which retrieves the specified next form's design; VINITFORM and/or VPUTBUFFER which places values into fields on the screen; VPUTWINDOW which places a message to be shown on the terminal window; and VSETKEYLABEL(S) which can label function keys. After all the information to be displayed is set up, VSHOWFORM is called to display the form.

This section discusses the VGETNEXTFORM, VINITFORM, VPUTBUFFER and VSHOWFORM intrinsics. When used in a program, the intrinsics are called in the order listed.

### a. VGETNEXTFORM

*Purpose: to retrieve the next form's definition.*

VGETNEXTFORM is the next intrinsic called after VOPENTERM. It is the first intrinsic to prepare and show the screen. It may also be called when a new form is to be displayed on the terminal.

The only parameter passed to VGETNEXTFORM is COMAREA. Within the comarea array, there are several variables pertinent to VGETNEXTFORM such as NFNAME, REPEATAPP, FREEZAPP and CFNAME. NFNAME is the variable containing the next form's name. CFNAME is the variable containing the current form's name. After the call to VGETNEXTFORM, the CFNAME variable in the comarea array is updated to contain the name of the form just read from the formsfile.

REPEATAPP is the repeat append option and will take on a numeric value from 0 - 2. These values are defined as follows:

0 - Do not repeat the current form
1 - Current form is to be repeated
2 - Current form is to be repeated and appended

FREEZAPP is the freeze append option. This will also take on a numeric value from 0-2. The values for FREEZAPP are defined as follows:

0 - Clear the current form before displaying the next form
1 - Next form is to be appended to the current form
2 - Current form is to be frozen before the next form is appended

VGETNEXTFORM retrieves the next form based on the value of two words in the COMAREA: REPEATAPP and NFNAME. If the repeat append word indicates the current form is to be repeated (REPEATAPP=1) or the current form is to be repeated and appended to itself (REPEATAPP=2), VGETNEXTFORM does not get the next form. REPEATAPP must be set to 0 before VGETNEXTFORM will retrieve the next form. The next form is determined by NFNAME. If NFNAME is blank, it gets the $HEAD form.

Other options for NFNAME, other than an actual form name are:

$END
$HEAD
$REFRESH
$RETURN

Upon the completion of the call to VGETNEXTFORM with $END as the NFNAME, the CFNAME becomes $END. The application must check the CFNAME to determine to shut down the application.

With $HEAD, the head form which is specified in FORMSPEC, is retrieved. Within FORMSPEC, a specific form may be designated as the head form. If this field is left blank, the head form becomes the first form in the formsfile ( alphabetic order of forms).

When VGETNEXTFORM is called with $REFRESH, the terminal is reinitialized. If there are forms in local forms storage, the forms are cleared. Only the current form is displayed with $REFRESH, therefore prior appended screens are lost. Data, screen design, windows and softkeys are displayed again. An alternative to using $REFRESH which takes less overhead is to simply display the data again. In many cases, where the data is overlaid but the screen design is intact, this method is sufficient. An example of when this type of refresh can be used is when another user has sent a WARN message through the user's form. This refresh method would simply force data and field enhancements to be rewritten to the screen. An example of the alternate method is:

```
VINITFORM (COMAREA)
SHOWCONTROL (14:1)=1
  [COBOL programmers must add 2 to SHOWCONTROL]
VSHOWFORM (COMAREA)
SHOWCONTROL (14:1)=0
  [COBOL programmers must subtract 2 from SHOWCONTROL]
```

Lastly, VGETNEXTFORM may be called with $RETURN. Before performing VGETNEXTFORM, REPEATAPP must be 0 as in all the above cases. VPLUS/3000 keeps track of the forms most recently used by your program. $RETURN returns the last form called. If the current form is $HEAD, it returns $HEAD again (no formsfile access is actually done).

The form file definition which is retrieved by VGETNEXTFORM is stored in an area of the user's stack called the comarea extension.

### b. VINITFORM

*Purpose: to initialize fields on the current form.*

VINITFORM is called after VGETNEXTFORM in order to initialize any fields on a form. For example, on the field menu, the designer may specify an initial value. Or within the field processing section of the field menu, a designer may specify INIT phase specifications such as setting the field to a value of another field on the form.

VINITFORM uses any initial value specified in the FIELD MENU and performs any processing specifications defined in the INIT phase.

Any field that does not have a specified initial value is by default set to $EMPTY (all blanks). If this is a child form, the values for any fields that do not have an initial value are retrieved from the parent form. The only parameter passed to VINITFORM is COMAREA. The fields within the comarea array which VINITFORM may set are listed in the *VPLUS/V Reference Manual.*

Any initialized values are placed in the comarea extension's data buffer.

### c. VPUTBUFFER

*Purpose: to place data from a program buffer into the VPLUS data area.*

Before the form is displayed with the initialized fields, additional information may be retrieved from a data base, KSAM file, MPE file, or from within the program. This is application dependent. Once the data is retrieved with a DBGET or FREAD call, VPUTBUFFER transfers the data from the program's form buffer to the VPLUS/3000's data buffer. VPUTBUFFER, however, does *not* format the data being sent to the data buffer (e.g., strip leading zeros or format signs on numeric fields); this must be handled by the application before VPUTBUFFER is called. The comarea extension contains its own copy of the data buffer which aids VPLUS/3000 to interface with the terminal.

VPUTBUFFER is an optional step for preparing and showing the screen function. It is used if information from other sources needs to be displayed on the screen. An example is: showing an employee's name which is retrieved from a personnel data base after the user has entered the employee's number.

The call to VPUTBUFFER in pseudocode looks like:

VPUTBUFFER (COMAREA, BUFFER, BUFLEN)

There are three parameters passed to VPUTBUFFER: COMAREA, buffer, and buffer length. The COMAREA words have been set from preceding VPLUS/3000 intrinsic calls. If CSTATUS word of the user's COMAREA is not zero, the call to VPUTBUFFER does not execute. The buffer is the name of the program's form buffer whose contents is to be put to the form. The third parameter is the length of the buffer in bytes.

### d. VSHOWFORM

*Purpose: to display screen definition, data, window function keys, enhancements or terminal screen.*

This intrinsic takes the current form as defined in the comarea extension form definition and displays it on the screen. A prior call to VGETNEXTFORM places the form definition in the comarea extension. VSHOWFORM displays the field values from the data buffer of the comarea extension. The content of the data buffer is affected by calls to VINITFORM, VPUTBUFFER or VREADFIELDS. VSHOWFORM also uses the global/form function keys and window buffer from the comarea extension to display the softkey labels and the window message.

The only parameter which is passed to VSHOWFORM is COMAREA. The SHOWCONTROL word in the COMAREA allows the programmer control of several VSHOWFORM options. The following paragraphs discuss some of the options and how they are used.

Bit 15=1 - Forces the form to be displayed again.

> This overrides the VPLUS/3000 default. This may be useful when using process handling and returning to the father process' screen. In some cases, VPLUS/3000 does not show the father process screen when control is returned to the father process because it believes the father screen is the current form. Bit 15 must be set to 1 in this case.

Bit 14=1 - Forces the window to be displayed again.

The other intrinsics which force the window to be displayed again are VPUTWINDOW, VSETERROR, VGETNEXTFORM with NFNAME=$REFRESH.

Bit 13=1 - Forces the data to be displayed again.

VPLUS normally does not re-display values that are already on the screen. Setting bit 13 to 1 overrides the default, and may be used as another alternative to $REFRESH. In most cases, forcing the data to be displayed again is sufficient rather than using $REFRESH which will re-display data, screen design, window and softkeys. (Refer to the example of how this is performed in the preceding VGETNEXTFORM discussion).

Bit 10=0 - Enables the keyboard (default).

Bit 10=1 - Disables the keyboard.

VREADFIELDS locks the keyboard so that no user input can take place until the screen is displayed. VSHOWFORM unlocks the keyboard so that a user may enter data after the screen is displayed. If more than one VSHOWFORM is called in a row without an intervening call to VREADFIELDS, bit 10 should be set to 1. After the last VSHOWFORM is called, bit 10 should be set back to 0 to allow user input of data.

Bit 9=0 - Do not preload current form into local form storage (default).

Bit 9=1 - Preload form into local form storage.

For terminals that have local form storage, the current form may optionally be loaded into local form storage if it is not already present.

Bit 8=0 - Do not enable [AIDS]/[MODES]/[USERKEYS] (default).

Bit 8=1 - Enables [AIDS]/[MODES]/[USERKEYS].

If bit 8 is set to 1, users may access the [AIDS]/ [MODES]/[USERKEYS]. Currently on VPLUS/3000 version B.04.20, a VENVCNTL.PUB.SYS file can also control this option; the format of this file is discussed in the *Communicator*, Volume 2, Issue 13 (UB-DELTA-1).

Bit 0=0 - Do not enable the touch screen feature (default).

Bit 0=1 - Enable the touch screen feature.

As of VPLUS/3000 version B.04.10, VREADFIELDS can be used to read a field number that has been "touched". Terminals that support the touch screen feature are listed in the *VPLUS/V Reference Manual* Appendix G.

## 3. Reading Data from the Screen

After displaying the screen along with any initial values, the user is ready to input data. VREADFIELDS triggers a read from the user. The read is satisfied when the user presses [ENTER] or one of the function keys. Also if the HPTOUCH feature is implemented, the read is satisfied when a user selects a field on the form. For more information on the HPTOUCH feature, refer to the *VPLUS/V Reference Manual.* The VPLUS/3000 Enhancement section briefly discusses this feature.

### a. VREADFIELDS

*Purpose: to trigger a read from the user.*

VREADFIELDS allows users to enter data. The read is satisfied if the user presses [ENTER] or any other function key. If [ENTER] is pressed, data is transferred from the screen to the data buffer in the comarea extension. Data is read in screen order, from top to bottom, and left to right.

If a function key is pressed, control is returned to the program without any data being transferred to the VPLUS/3000 data buffer in the comarea extension. Instead, the LASTKEY word in the comarea array is set. LASTKEY contains a value corresponding to the function key that satisfied the read trigger. LASTKEY will take on a value as follows:

0 - Enter Key
1 - 8 - Corresponding to F1-F8

If the HPTOUCH feature is implemented, VREADFIELDS is satisfied when a user selects a field. In this case, no data is transferred to VPLUS/3000's data buffer. Instead, the LASTKEY word in the comarea array is set to a negative number which indicates the field number selected. The number is negative to distinguish it from the function key's numbers which are positive 1 through 8.

The only parameter passed to VREADFIELDS is COMAREA. Within the COMAREA, the TERMOPTIONS word can be set to perform some advanced functions. These include setting up automatic reads or having a timed read.

AUTOREAD is used when VREADFIELDS is terminated by a function key instead of [ENTER]. For example, if an order processing clerk is entering order information on a screen, then presses a function key labeled "MODIFY", the program needs to read the data on the screen. Without the AUTOREAD option set, the program would not read the screen. The program needs to interrogate the LASTKEY word of the COMAREA for a i value. If a non-zero value is found, it needs to proceed with enabling AUTOREAD.

AUTOREAD is enabled by setting bit 14 of the TERMOPTIONS word to 1. For COBOL programmers, ADD and SUBTRACT may be used to set bit 14. For example, in pseudocode:

```
VREADFIELDS (COMAREA)
IF LASTKEY = 1 THEN
   TERMOPTIONS(14:1)=1  [COBOL programmers must add 2 to TERMOPTIONS]
   VREADFIELDS(COMAREA)
   TERMOPTIONS(14:1)=0  [COBOL programmers must subtract 2 from TERMOPTIONS]
ENDIF
```

Another feature which is enabled through the TERMOPTIONS word of the comarea array is setting a timed read. The TERMOPTIONS word bit (10:1) enables a timed read. Another word in the comarea array, USER'TIME, defines the time limit in seconds. Again, COBOL programmers must use the ADD and SUBTRACT to set bit 10. For example, to enable a timed read for 60 seconds in pseudocode:

```
USER'TIME = 60
TERMOPTIONS(10:1)=1  [COBOL programmers must add 32 to
                      TERMOPTIONS].
VREADFIELDS(COMAREA)
IF COM'STATUS = 160 THEN
   PERFORM TIME-OUT-PROCEDURE
ENDIF
```

TERMOPTIONS(10:1)=0 [COBOL programmers must subtract 32 from TERMOPTIONS].

## 4. Editing

Once the data is read from the screen into VPLUS/3000's data buffer, edits may be performed. The edits may be performed in as many as three passes. On the first pass, the editing process checks that all required fields are present, the data input by the user is consistent with the data type defined for the field, and the data meets rules set out by the user (for example, matching a pattern or being within range of values). Optionally, a second editing pass may be performed by the program. Edits can be designed into the program. An example of a programmatic edit is verifying that an employee number exists in the personnel data base before allowing a modify of an employee record. Optionally, a third pass of editing may be performed. The third pass of editing performs any totaling or formatting functions. For example, if a form contains hours worked for each day of the week, a first level editing pass would verify that the hours input for each day is less than 24 hours. In this example, we will assume that there are no programmatic edits in the second pass of editing. However, in the third pass of editing, a weekly total is accumulated and the numbers are right justified.

This section on "Editing" will cover the three phases of editing: VFIELDEDITS, optional programmatic edits, and VFINISHFORM.

### a. VFIELDEDITS

*Purpose: to verify the data entered.*

This intrinsic is called to verify the data entered. If the field is defined as OPTIONAL and is left blank, no further processing is performed. To perform processing on a blank field (for instance, to set it to a value), the field type should be defined as PROCESS. VFIELDEDITS performs the following checks if the field is a REQUIRED field type or a PROCESS field type and the field contains data.

1. Field type - If the field type is REQUIRED and no data is present, the field is flagged in error.

2. Data type - If the data type is invalid, the field is flagged in error. The data input by the user must match the data type defined for the field.

3. Field processing specification statements - The user may define processing specification statements within each field menu. If the data does not meet the specifications set up by the user, the field is flagged in error.

If VFIELDEDITS detects any errors on the checks listed above, the field is set in error and the NUMERRS word of the user's COMAREA is incremented by one. The only parameter passed to VFIELDEDITS is COMAREA. When the intrinsic completes, it may set the NUMERRS word within the comarea array. NUMERRS contains a count of the number of fields on the form detected in error. Note that a CSTATUS of 0 only indicates that the intrinsic executed properly, not that no errors were found; this information is contained in NUMERRS.

In looking at *Illustration 1*, if there are any errors (NUMERRS > 0) the program calls an error processing routine. The error processing routine calls VERRMSG to retrieve the appropriate error message of the

first field in error. The error processing routine then calls VPUTWINDOW which puts the error message to a window buffer maintained by VPLUS/3000. It next calls VSHOWFORM which highlights all fields in error and displays the error message in the terminal window. VREADFIELDS is called to prompt the user to input the correct data. Then VFIELDEDITS is called once again. This loop is performed until there are no errors flagged by VFIELDEDITS.

### b. Optional programmatic edits against data

After passing the first phase of editing, additional checks may be desired. For example, a programmatic check could be made to verify that the employee number entered exists in the company's data base. Since this type of check cannot be done in FORMSPEC, it needs to be implemented by the program with a call to a data base, KSAM or MPE file.

Before any user editing is performed, the program needs to retrieve VPLUS/3000's data buffer into its own program buffer. A call to VGETBUFFER or VGETFIELD or VGETtype returns the data to the program's buffer. At this point, the data may can be verified against a data base or KSAM file or MPE file. If the programmatic edits detect an error, VSETERROR is called to set the field in error.

VSETERROR is another VPLUS/3000 intrinsic available to the programmer so that when programmatic edits detect invalid data, the field may be highlighted and set in error. The pseudocode for VSETERROR looks like:

VSETERROR (COMAREA, FIELDNUM, MESSAGE, MSGLEN)

VSETERROR sets the field in error if it is not already in error and increments the NUMERRS word of the user's COMAREA by one. After the call to VSETERROR, NUMERRS reflects the number of fields detected in error on the form. The second parameter of VSETERROR is the field number to be set in error. The third and fourth parameters of VSETERROR are the customized error message and the error message length, respectively. If this is the first field in error, the customized error message appears in the terminal window on a call to VSHOWFORM.

In looking at *Illustration 1*, if the optional programmatic edits detect an error, the field is set in error by VSETERROR. The subsequent VSHOWFORM displays the programmer's custom error message and highlights the fields in error. VREADFIELDS is then called after VSHOWFORM to allow the user to input the corrected data. The data is next taken through all the passes of editing once again - beginning with VFIELDEDITS, next with the programmatic edits, and finally with VFINISHFORM.

### c. VFINISHFORM

*Purpose: to perform final editing.*

Once the validity of data has been established with VFIELDEDITS and it has passed optional programmatic editing, VFINISHFORM can be called to perform the final totaling and editing processes. VFINISHFORM executes any processing statements defined in the FINISH phase. If there are no FINISH phase statements in the form, it is not necessary to call VFINISHFORM.

FINISH phase statements are input through FORMSPEC in the field menu at the field processing specification section. Formatting and totaling are typical functions performed in this phase. An example of the formatting at FINISH phase is right justifying a number, or stripping out a decimal point or filling a numeric field with leading zeros.

The only parameter passed to VFIELDEDITS is COMAREA. Within the comarea array, VFIELDEDITS may set the NUMERRS word. If there are any field type, data type or editing errors, the field is set in error and NUMERRS is incremented.

In looking at the flowchart, if there are any errors (NUMERRS>0), VERRMSG retrieves the appropriate error message. VPUTWINDOW puts the error message in VPLUS/3000's window buffer. When VSHOWFORM is called, all fields in error are highlighted and the window displays the error message retrieved by VERRMSG. VREADFIELDS allows the user to input corrected data. Again, the flowchart passes through all levels of error checking from VFIELDEDITS through VFINISHFORM. Once the data passes all levels, the program proceeds to the next step.

## 5. Returning Data to the Program

After the data is verified by VFIELDEDITS, optional programmatic edits, and VFINISHFORM, it is returned to the program's form buffer with a call to VGETBUFFER. There are other intrinsics to return data from VPLUS/3000's data buffer to the program buffer such as VGETtype and VGETFIELD. These two intrinsics can be used to retrieve one field's worth of data to a program buffer. Refer to the *VPLUS/V Reference Manual* on their use.

This section will cover the VGETBUFFER intrinsic.

### a. VGETBUFFER

*Purpose: to return values from VPLUS/3000 data buffer to the user's program buffer.*

When VREADFIELDS is performed, it returns user input data to the VPLUS/3000 data buffer. When VFIELDEDITS and VFINISHFORM are performed, the editing and formatting is performed on VPLUS/3000's copy of the data buffer. In order to return the data to the program buffer, a call to VGETBUFFER is necessary.

The call to VGETBUFFER in pseudocode looks like:

```
VGETBUFFER (COMAREA, BUFFER, BUFLEN)
```

The number of bytes that VGETBUFFER returns into the program form buffer is determined by BUFLEN (the third parameter of the call) and DBUFLEN (word in comarea array which contains the buffer length in bytes of the current form). DBUFLEN is set by the call to VGETNEXTFORM which retrieved the current form. VGETBUFFER transfers the lesser number of bytes between BUFLEN and DBUFLEN into the program buffer. VGETBUFFER returns all the data from the screen in a left-to-right and top-to-bottom order. Display-only fields are also returned into the program buffer and therefore you will need to ensure that the program buffer is set up to accommodate this data.

There are also two other routines that transfer data from the VPLUS/3000 data buffer into the program buffer.

- VGETFIELD - transfers one field from the VPLUS/3000 data buffer to a character buffer.
- VGET{type} - transfers one field from the VPLUS/3000 data buffer to a numeric defined field, such as an integer, double integer, real, or long.

VGETFIELD and VGETtype are useful to you if you want to return one field at a time to your program buffer. If the form design is changed so that fields are moved around within the form, VGETBUFFER's

call requires that your program buffer matches the changes you have made to the screen. With VGETFIELD and VGETtype calls, the program buffer's definition is independent of the order in which the fields are organized on the screen.

Once the data resides in the program buffer, the program may write the data to a data base, KSAM or MPE file.

## 6. Closing Files

Once data collection has finished, the terminal and formsfile may be closed. Signaling the end of data collection can be accomplished several ways and is application dependent. Some programmers use [f8] to signal an exit. When VREADFIELDS has finished and LASTKEY is equal to 8, the program performs the closing routines.

### a. VCLOSETERM

*Purpose: to close the terminal file and reset the terminal out of block mode.*

This takes the terminal out of block mode and resets the straps. VCLOSETERM only partially restores the terminal's configuration. One of the things that it does not do is to restore user function key labels.

If a VPLUS/3000 application terminates abnormally, VCLOSETERM is not done. Therefore the terminal is not taken out of block mode and echo is not turned back on. Generally a hard reset and an " [ESCAPE] : " can reset the terminal back to a usable state.

### b. VCLOSEFORMF

*Purpose: to close the formsfile.*

In all languages except BASIC/3000, VCLOSEFORMF releases stack space allocated by the VOPENFORMF intrinsic. The stack space released was for the comarea extension. In BASIC/3000, since the comarea extension is declared as a global variable, the space is not released with a call to VCLOSEFORMF.

In addition, VCLOSEFORMF performs an FCLOSE on the formsfile.

# III. VPLUS/3000 ENHANCEMENTS

There have been many changes and enhancements to VPLUS/3000 since version B.04.10. This section will briefly discuss some of the changes and enhancements and how they can be of use to the programmer. The ones reviewed in this section are:

1. HPTOUCH Support
2. Cursor Positioning
3. 264X Function Key Labels
4. Enhanced VGETFORMINFO
5. Batch Mode Formspec Enhancements
6. Color Support for 2627A and 2392A Terminals
7. VCHANGEFIELD
8. VPLUS/3000 Environment Control File

## HPTOUCH Support (introduced on VPLUS/3000 B.04.10 in MPE G.01.01)

With HPTOUCH support in VPLUS/3000, a programmer may design a form that is to be used with the HP150, the 2393A or the 2397A. Users may select a field by touching the field on the screen. When a field is selected, VREADFIELDS interprets the row and column coordinates into a field number.

VREADFIELDS returns the field number as a negative number to LASTKEY. Negative numbers are returned into LASTKEY so as to distinguish from the positive number that is returned when a function key is hit. LASTKEY is set to -999 if the area touched is not a field. HPTOUCH is enabled by setting bit 0 of the SHOWCONTROL word in the user's COMAREA to 1. An example of coding the HPTOUCH feature is outlined in the *VPLUS/V Reference Manual* in Appendix E. Currently, HPTOUCH is supported on the 2393A, 2397A and HP150 terminals. For an up-to-date list of the terminals supporting HPTOUCH, refer to Appendix G of the *VPLUS/V Reference Manual.*

## CURSOR POSITIONING (introduced on VPLUS/3000 B.04.10 in MPE G.01.01)

VPLUS/3000 displays the screen and positions the cursor to the first field on the form. With some applications, positioning the cursor elsewhere would eliminate the requirement that the user press [TAB] to get to the desired field. VPLACECURSOR is a new intrinsic introduced in VPLUS/3000 version B.04.10 which allows the cursor to be positioned at an unprotected field. An unprotected field is any field in which the user is allowed to enter data. Protected fields may be headings or display-only fields. If the cursor is positioned at a display-only field, an error is returned. VSHOWFORM should be called first before positioning the cursor. The parameters of VPLACECURSOR are:

1. COMAREA

2. FIELDNUM - If the field number parameter is negative, it implies screen order number. If the field number parameter is positive, field number creation order is implied. The screen order number is a relative number of the field on the form. For example, if there are five fields on a form and the programmer wants to place the cursor on the fourth field, the FIELDNUM parameter would contain a "-4". On the other hand, if the FIELDNUM parameter is positive, it implies field number creation order. This number is assigned to the field when the field is created through FORMSPEC. This number stays with the field unless the form is renumbered through batch mode FORMSPEC or if the field is deleted from the form.

## 264X FUNCTION KEY LABELS (introduced on VPLUS/3000 B.04.10 in MPE G.01.01)

With 264X terminals, there are no function key labels like the 262X or 239X terminals. As of VPLUS/3000 version B.04.10 which was introduced in MPE version G.01.01, function key labels may now be displayed on 264X terminals. With this version of VPLUS/3000, function key labels appear in lines 23 and 24 of the terminal display on a 264X terminal. With 264X terminals, function keys may be displayed by performing the following steps:

1. Select "Y" instead of "X" in the 264X box of the TERMINAL/LANGUAGE SELECTION MENU.

2. Define the labels using the same methods as with other HP Terminals, for example,. enter the labels using the FORM FUNCTION KEY LABELS MENU, or enter the labels using the GLOBAL FUNCTION KEY LABELS MENU, or set the labels in your program with calls to VSETKEYLABEL or VSETKEYLABELS.

3. Set the LABELOPTION word in the COMAREA to 1 before a call to VOPENFORMF (as with all other terminals).

## VGETFORMINFO Enhancement (enhanced on VPLUS/3000 B.04.10 in MPE G.01.01).

Prior to this enhancement, if VFIELDEDITS detected a field in error, the total number of errors would be set in the NUMERRS word of the user's COMAREA. All fields would be highlighted upon a call to VSHOWFORM. No information regarding which fields were found to be in error is passed to the program.

With VPLUS/3000 version B.04.10, VGETFORMINFO has been enhanced to return information on all fields in error. The information may be used by the program if a special error routine is to be performed when a certain field is in error. The information buffer returned by VGETFORMINFO has been expanded to include a 16-word array. This 16-word array is a bit map of all the fields on the form and can accommodate the maximum of 256 field numbers. Field numbers may range from 1 to 256, although a form may have a maximum of 128 fields on the form. If the bit corresponding to the field is set to 1, the field is in error.

To decode and set the bits, a new intrinsic called BITMAPCNV is available and documented in the *VPLUS/V Reference Manual*, Appendix I.

## BATCH MODE FORMSPEC (enhanced on VPLUS/3000 B.04.10 in MPE G.01.01)

There have been several enhancements to batch mode FORMSPEC. Chapter 7 of the *VPLUS/V Reference Manual* describes how to use batch mode FORMSPEC. Three new commands are available to the programmer. These three commands are:

- FIELD - This command allows, several field attributes to be updated in batch mode. Any of the following can be changed: field name, field enhancements, field type, data type, initial value.
- RENUMBER - This command reassigns field numbers to their screen order numbers. This is particularly useful for forms which have deleted fields and are approaching the maximum field number (256).
- FKLABELS - This command creates or updates a form's function key labels.

To further enhance batch mode compiling in FORMSPEC, two new JCWs have been introduced: FORMSPECERRORJCW which indicates the count of the number of errors during a compile and FORMSPECWARNJCW which indicates the count of the number of warnings during a compile.

## COLOR SUPPORT for 2627A and 2397A (introduced on VPLUS/3000 B.04.15 in MPE G.01.04).

Starting with VPLUS/3000 version B.04.15 which was introduced on MPE version G.01.04, 2627A and 2397A color terminal users may have forms displayed with specific colors defined for a field. VPLUS/3000 has defined eight color pairs which can be used to enhance fields, highlight fields in error, and display the window. Digits 1 through 8 correspond to the default color pairs. The color pairs are defined in Chapter 3 of the *VPLUS/V Reference Manual.* These color pairs can be used along with the normal enhancement set of H, I, B, U, S or NONE for field, error or window enhancements. To use the color enhancements, the formsfile must have an "X" in the HP2627 box in the TERMINAL/LANGUAGE SELECTION MENU in FORMSPEC.

## VCHANGEFIELD (introduced on VPLUS/3000 version B.04.15 in MPE G.01.04).

Prior to VPLUS/3000 version B.04.15, a field defined through FORMSPEC could not be changed in the program. Therefore, if a programmer decided that a field defined as a "REQUIRED" field type should now be considered a "DISPLAY" field type, this could not be done. With VCHANGEFIELD, this is now possible. This intrinsic dynamically alters a field on a form. The changes are temporary, and are not posted to the formsfile. Some of these changes are:

- Changing field enhancements to the following: H, I, B, U, NONE. Changing field enhancements to S (security) is not available. As of version B.04.20, VCHANGEFIELD has been expanded to allow field enhancements to include a color pair in addition to H, I, B, U, or NONE.
- Changing field types to any of the following: Optional (O), Display (D), Processing (P), Required (R).

- Changing data type to any of the following: CHAR, DIG, NUMn, IMPn, DMY, MDY, YMD.

**VPLUS/3000 USER ENVIRONMENT CONTROL FILE** (introduced on VPLUS/3000 B.04.20 in MPE G.02.01).

With VPLUS/3000 version B.04.20 introduced in MPE G.02.01, there is an MPE file called VENVCNTL.PUB.SYS which is opened and read on the VOPENTERM call. Currently, the VPLUS/3000 USER ENVIRONMENT CONTROL file can control two items when VOPENTERM is executed. Two options have been implemented for the user environment control file.

- The first option is an abbreviated terminal query. When VOPENTERM is called, it goes through an involved process to identify the terminal type. With the VPLUS/3000 user environment control file, this terminal identification process can be shortened. This option should only be used if it is known that a query will return a valid terminal ID. To effect a shortened terminal query, place a "1" in column 1 of the VENVCNTL file.

- The second option is the ability to enable [AIDS]/[MODES]/[USERKEYS] without using a program. Instead of setting the SHOWCONTROL word of the user's COMAREA in your program, use the VENVCNTL file. Placing a "1" in column 2 of the VENVCNTL file enables the keys. This option is particularly useful if you do not have source code for the software packages that you are running, but would like to enable [AIDS]/[MODES]/[USERKEYS].

To implement a user environment control file for VPLUS/3000, build the file:

```
:BUILD VENVCNTL.PUB.SYS;DEV=DISC;REC=-80,1,F,ASCII
```

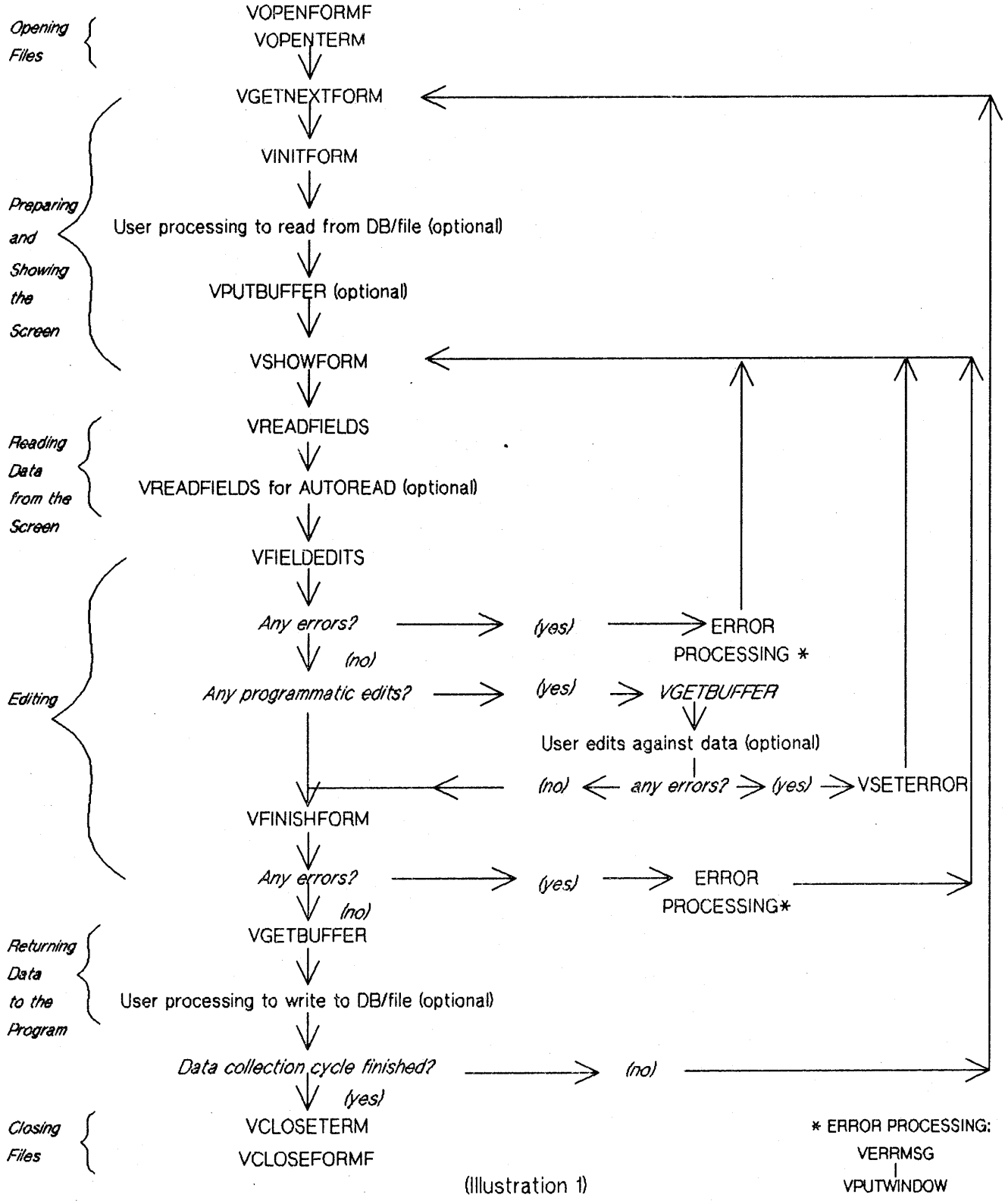
Use any type of editor tool to place a "1" in column 1 and/or column 2. When VOPENTERM executes, it looks for the user environment control file. If it doesn't find it, VPLUS/3000 operates as usual. If the file exists and columns 1 and/or 2 are set, VPLUS/3000 will perform the abbreviated terminal query and/or the enabling of keys depending on what columns are set.

It is also possible to have a VENVCNTL file in a different group and account than PUB.SYS. If the file is in another group and account, this file equation is necessary:

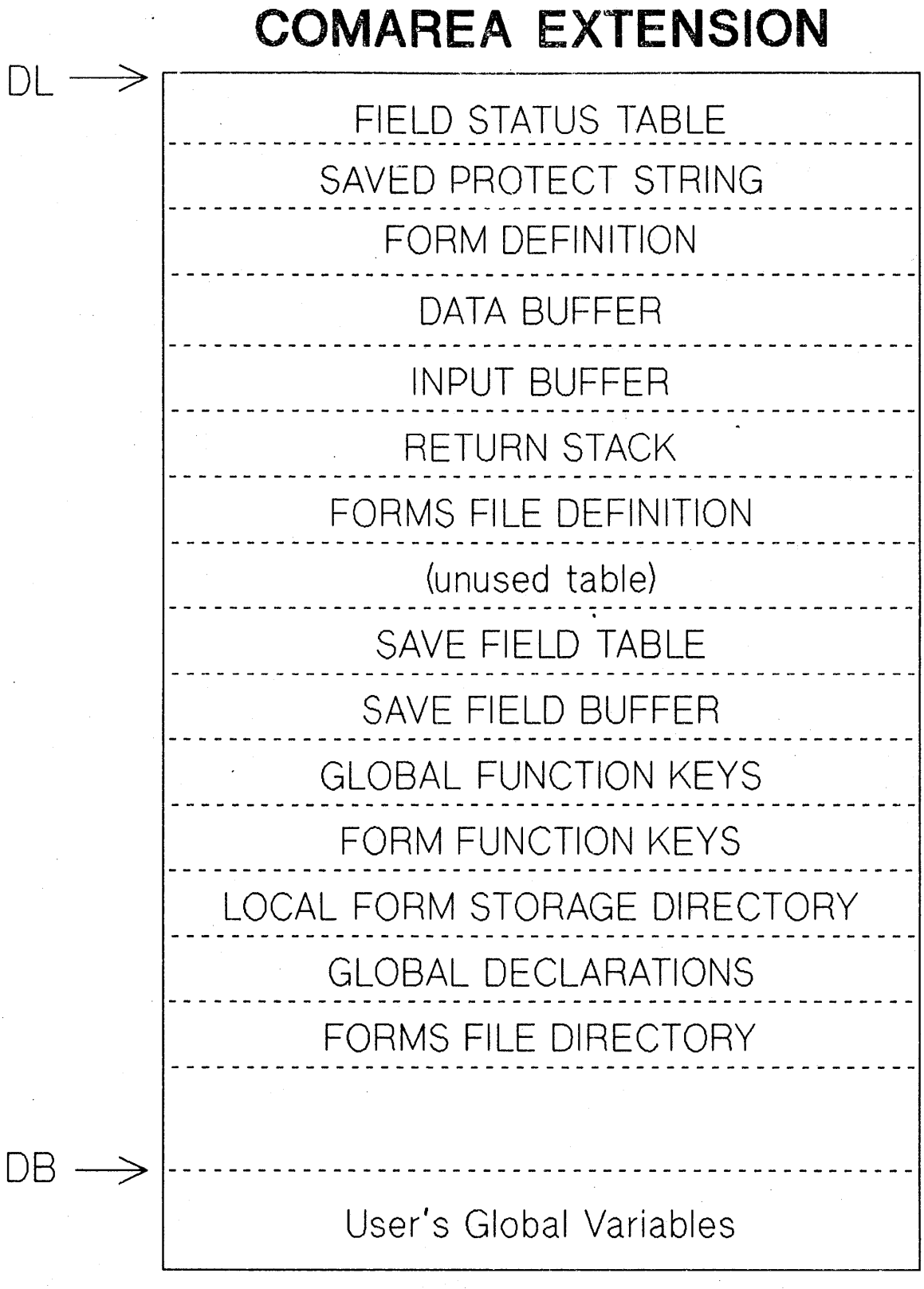
```
:FILE VENVCNTL.PUB.SYS=VENVCNTL.mygroup.myaccount
```

For more information on how to implement the features, refer to the *VPLUS/V Reference Manual.*

# FLOWCHART OF VPLUS/3000 CALLS




(Illustration 1)




(Illustration 2)