HP 3000 Computer Systems

Data Entry and Forms Management System V/3000

Reference Manual

HEWLETT PACKARD

19420 HOMESTEAD RD., CUPERTINO, CALIFORNIA 95014

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars are removed but the dates remain. No information is incorporated into a reprinting unless it appears as a prior update.

First Edition	Nov 1978
Second Edition	Jan 1980
Update Package No. 1	Apr 1980
Third Edition	Feb 1981

PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update. The edition does not change.

The software product part number printed alongside the date indicates the version and update level of the software product at the time the manual edition or update was issued. Many product updates and fixes do not require manual changes, and conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition	Nov 1978	32209A.00
Second Edition	Jan 1980	32209A
Update Package No. 1	Apr 1980	32209A.01
Update No. 1 Incorporated	Aug 1980	32209A.01
Third Edition	Feb 1981	32209B

PREFACE

This publication is the reference manual for V/3000, a comprehensive software system that operates on the HP 3000 computer system, to provide terminal data entry and control for most HP 262x, 264x, and 3075/6 terminals.

V/3000 provides the following capabilities:

- Forms design, including data editing and formatting, and form modification.
- Stand-alone data entry program.
- Reformatting of entered data.
- Programmatic interface to terminals, files, and forms.

V/3000 can be used for stand-alone data entry with no programming effort, or it can be used as a front end to transaction processing applications written in COBOL, RPG, BASIC, FORTRAN, or SPL.

V/3000 users may be terminal operators (such as data entry clerks), forms designers with or without programming experience, or programmers in any of the languages listed above. The manual is organized to start with simple operation of the stand-alone data entry program, and progress through the various levels of form design and reformatting, to the procedures used for programmatic interface with forms and terminals. Since the RPG interface differs from that of the other languages, it is described in a separate section.

In order to use the V/3000 system effectively, you should know how to operate the HP terminals used at your facility. If you use the programming interface, you may also need the manual for the programming language in which you are coding. And, since V/3000 can use KSAM files, you may also want to read the KSAM manual. The manuals in the following list contain all the information you might need as a supplement to this manual:

V/3000 Operator's Quick Reference Guide (32209-90003)
Using V/3000 (32209-90004)
Using the HP 3000 (03000-90121)
MPE Commands Reference Manual (30000-90009)
MPE Intrinsics Reference Manual (30000-90010)
MPE Software Pocket Guide (30000-90049)
KSAM/3000 Reference Manual (30000-90079)
Using Files (30000-90102)
2624A Reference Manual (02624-90002)
2626A Display Station Reference Manual (02626-90002)
2640B Display Station User's Manual (02640-90109)
2640B Display Station Reference Manual (02640-90110)
2645 Display Station User's Manual (02645-90001)
2645 Display Station Reference Manual (02645-90005)
3075A, 3076A, 3077A Data Capture Terminal Reference Manual (03075-90011)
COBOL/3000 Reference Manual (32213-90001)
Using COBOL (32213-90003)
RPG/3000 Compiler Reference Manual (32104-90001)
BASIC/3000 Interpreter Reference Manual (30000-90026)
BASIC/3000 Compiler Reference Manual (32103-90001)
FORTRAN/3000 Reference Manual (30000-90040)
SPL/3000 Reference Manual (30000-90024)

CONVENTIONS USED IN THIS MANUAL

NOTATION	DESCRIPTION
[ ]	An element inside brackets is *optional*. Several elements stacked inside a pair of brackets means the user may select any one or none of these elements. Example: $\begin{bmatrix} \text{A} \\ \text{B} \end{bmatrix}$ user may select A or B or neither.
{ }	When several elements are stacked within braces the user must select one of these elements. Example: $\left\{\begin{matrix} \text{A} \\ \text{B} \\ \text{C} \end{matrix}\right\}$ user must select A or B or C.
italics	Lowercase italics denote a parameter which must be replaced by a user-supplied variable. Example: CALL *name* *name* one to 15 alphanumeric characters.
. . .	A horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted.
underlining in dialog	When it is necessary to distinguish user input from computer output, the input is underlined.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

ILLUSTRATIONS

TABLES

INTRODUCING V/3000

SECTION I

OVERVIEW

V/3000 is a comprehensive software system that implements and controls source data entry, and also provides an interface between a terminal and any transaction processing program.

SOURCE DATA ENTRY

As a source data entry system, V/3000 provides easy forms design with data editing and validation built into the forms. It also provides a ready-to-use data entry program so that you can enter data into the system with no programming effort. This same program allows you to "browse" the entered data and modify it as it is entered.

Thus, source data entry through V/3000 can be done with no programming. If, however, you need additional or different capabilities, you can modify the existing data entry program or write your own application, incorporating V/3000 procedures.

TRANSACTION PROCESSING

As an interface to transaction processing applications, V/3000 provides a set of procedures that allow you to control forms and data on a terminal from an application program. These procedures are available to programs written in COBOL, RPG, FORTRAN, BASIC, and SPL.

V/3000 also provides a reformatting capability. You can enter specifications to control how entered data is to be reformatted, and then run a program to actually reformat the data.

The V/3000 procedures and the reformatting capability, either singly or in combination, provide a "front end" to existing transaction processing applications. Thus, V/3000 allows you to concentrate on processing problems rather than on editing data or controlling the terminal.

FEATURES

The following are the main features of the V/3000 system:

- A forms design program (FORMSPEC) that allows quick and easy forms design at a terminal using "menus".
- Advanced forms design (through FORMSPEC) that provides editing, formatting, movement, and computation of data as it is entered.
- A ready-to-run data entry program (ENTRY) that provides immediate data entry and modification with no programming effort.
- A flexible data reformatting design program (REFSPEC) that specifies reformatting of entered data.

- A batch program (REFORMAT) that reformats the data according to the REFSPEC formatting specifications and writes it to a file for use by an application.
- A set of procedures that provide a powerful language interface to terminals, using FORMSPEC definitions, from user programs written in COBOL, RPG, BASIC, FORTRAN, or SPL.

Refer to figure 1-1 for an overview of the entire V/3000 system.

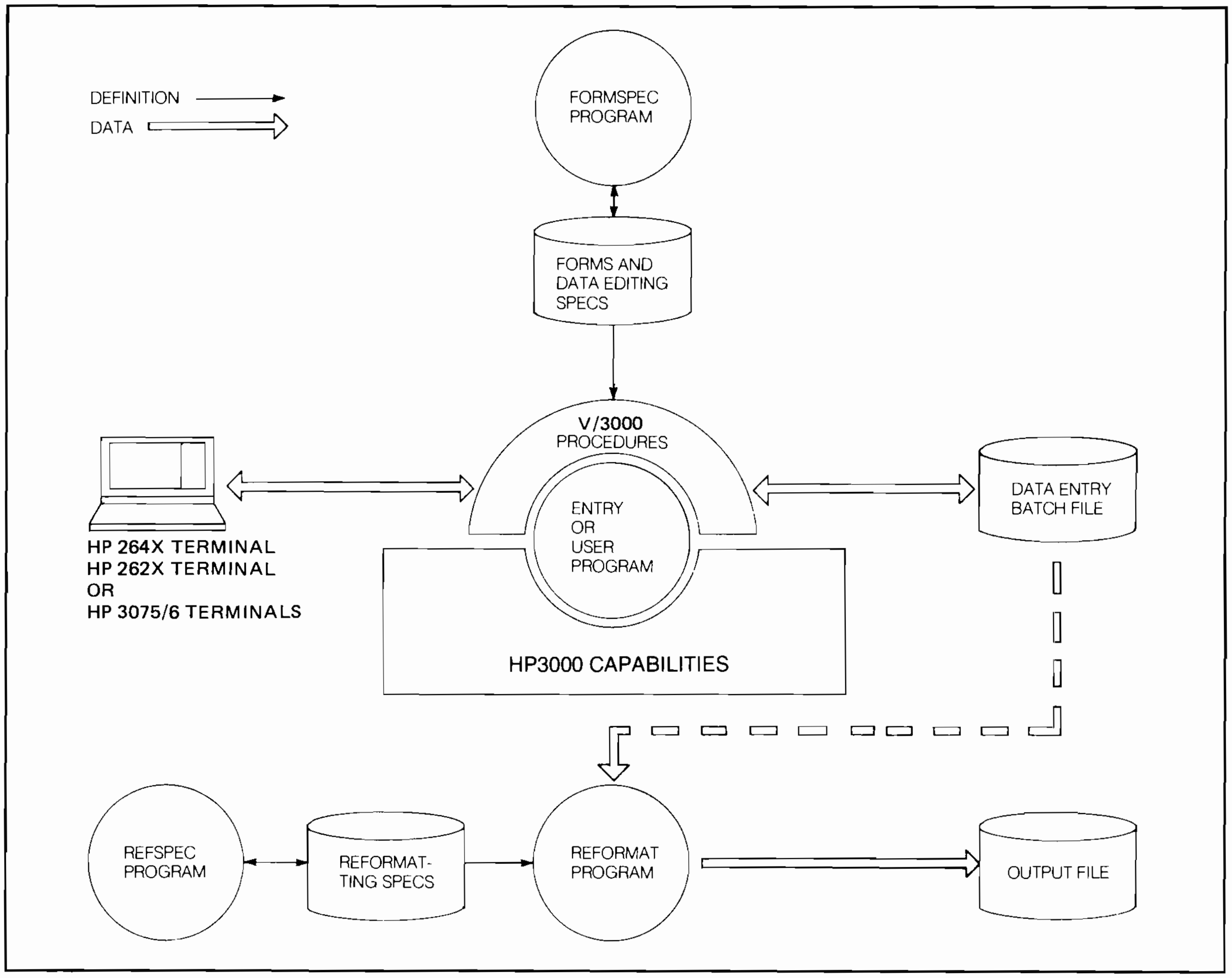


Figure 1-1. Overview of V/3000

Figure 1-1 shows data entered at the terminal, read and manipulated by V/3000 procedures in the data entry program ENTRY or a user program, according to FORMSPEC forms and editing specifications, in conjunction with the standard HP 3000 capabilities. The data entered at the terminal is written to a data entry "batch" file where it is available for use by an application program. If desired, it can be reformatted by the REFORMAT program and written to an "output file".

SUPPORTED TERMINAL FEATURES

V/3000 can be used with most HP 264x terminals. In addition the HP 2624, 2626, 3075 and 3076 terminals can be used. Applicable special features of these terminals are described below.

HP 2624 Special Features

*Modified Data Tag

You do not have to take any action to utilize this feature. Only the changes made to a field will be transmitted to the computer.

*Extended Local Edits

This feature provides for the editing of information as it is typed into the form. You specify the editing to be done by using the command LOCALEDITS in the field processing specification area.

*Relabeling Function Keys

This feature allows you to change function key labels during forms creation or modification that will be used during data entry.

*Security Display Enhancement

This feature allows you to inhibit the display of certain fields. When this feature is used, characters entered at the terminal are displayed as blanks on the screen.

HP 2626 Special Features

*Relabeling Function Keys

This feature allows you to change function key labels during forms creation or modification that will be used during data entry.

*Security Display Enhancement

This feature allows you to inhibit the display of certain fields. When this feature is used, characters entered at the terminal are displayed as blanks on the screen.

*Local Form Storage Capabilities

This feature allows for local terminal storage of up to four forms. V/3000 will control the creation and management of the workspaces as well as the mapping of the workspaces into windows. V/3000 will use the workspaces to preload as many as four forms into the terminal.

You can control which forms are loaded by making sure COM'NFNAME contains the correct name before calling VREADFIELDS. If you want more control over which forms are loaded, you can disable LOOK' AHEAD and use the intrinsic VLOADFORMS to load the terminal's workspaces.

V/3000 will place appended forms into the workspace that contains the form to which they are appended.

DEFAULT is one form in the terminal at a time.

HP 3075/6 Special Features

The HP 3075/6 terminals are Data Capture Terminals. Special features of the 3075/6 terminals that V/3000 will support are as follows:

*Terminal Input Options

- TypeV Badge Reader
- Multifunction Reader
- Magstripe Reader
- Bar Code Reader
- Standard Keyboard
- Alphanumeric Keyboard

*Terminal Output Options

- Alphanumeric Printer
- Prompting Lights
- Alphanumeric Display
- Mini-CRT

The HP-IB and RS232 Input/Output options, the Numeric Display terminal, Card Image Data and Multifield Input options will not be supported.

DESIGNING FORMS

The FORMSPEC program allows you to design forms in as simple or as complex a manner as you desire. Whatever the level of complexity of the final forms, designing the forms with FORMSPEC is easy and fast.

When all forms for an application are designed, they comprise the forms file. Any form can be modified while you are initially designing the forms, or you can modify forms after the forms file is compiled.

The form specifications are entered on formatted screens called "menus" that are issued by FORMSPEC. A combination of a "MAIN" menu and terminal function keys gives you complete control over display of the FORMSPEC menus. You can thus change existing forms, add new forms, delete forms or fields, and so forth.

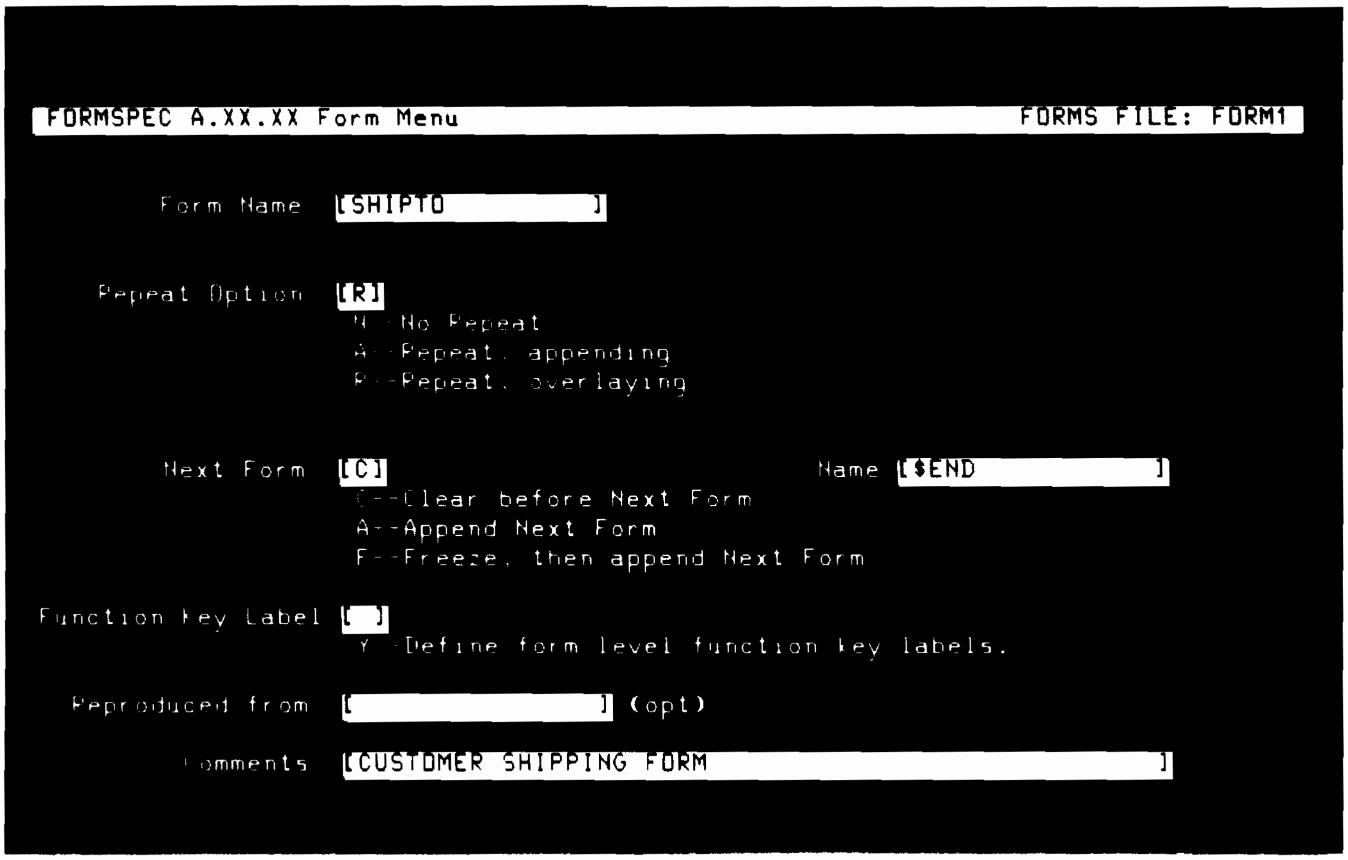
LEVELS OF FORM DESIGN

Form design with FORMSPEC can be thought of as having four levels of complexity:

- Simple Collection — Draw the form on the screen and accept any ASCII characters entered by the operator. This level uses none of the FORMSPEC editing or formatting capabilities.
- Simple Editing — Draw the form on the screen and specify edits based on field type (optional, required, display only) or data type (character, numeric, or date). No special language is required for these edits.
- Full Field Edits — Specify a full range of field edit statements that apply to individual fields in a form. These include: minimum length, range checks, pattern checks, and so forth. A subset of the FORMSPEC field specification language is used for these edits.
- Advanced Processing — Specify movement of data between fields and forms, arithmetic computation of data, formatting of data (JUSTIFY, FILL, STRIP), alteration of forms sequence, and conditional processing based on the result of editing statements. This level uses the full range of the FORMSPEC language.

SIMPLE COLLECTION

When it is first run, FORMSPEC asks you the name of the forms file, and then displays the MAIN menu so you can select what you want to do: add a form, modify or delete a form, copy a form, compile the forms file, or go to a particular menu. Suppose you want to add a form. FORMSPEC displays a Form Menu (see Figure 1-2) on which you enter the form name and how you want the form to be sequenced.



```
 FORMSPEC A.XX.XX Form Menu                                     FORMS FILE: FORM1

            Form Name  [SHIPTO          ]

        Repeat Option  [R]
                        N--No Repeat
                        A--Repeat, appending
                        R--Repeat, overlaying

            Next Form  [C]                              Name [$END            ]
                        C--Clear before Next Form
                        A--Append Next Form
                        F--Freeze, then append Next Form

   Function Key Label  [ ]
                        Y--Define form level function key labels.

      Reproduced from  [                ] (opt)

             Comments  [CUSTOMER SHIPPING FORM                               ]
```

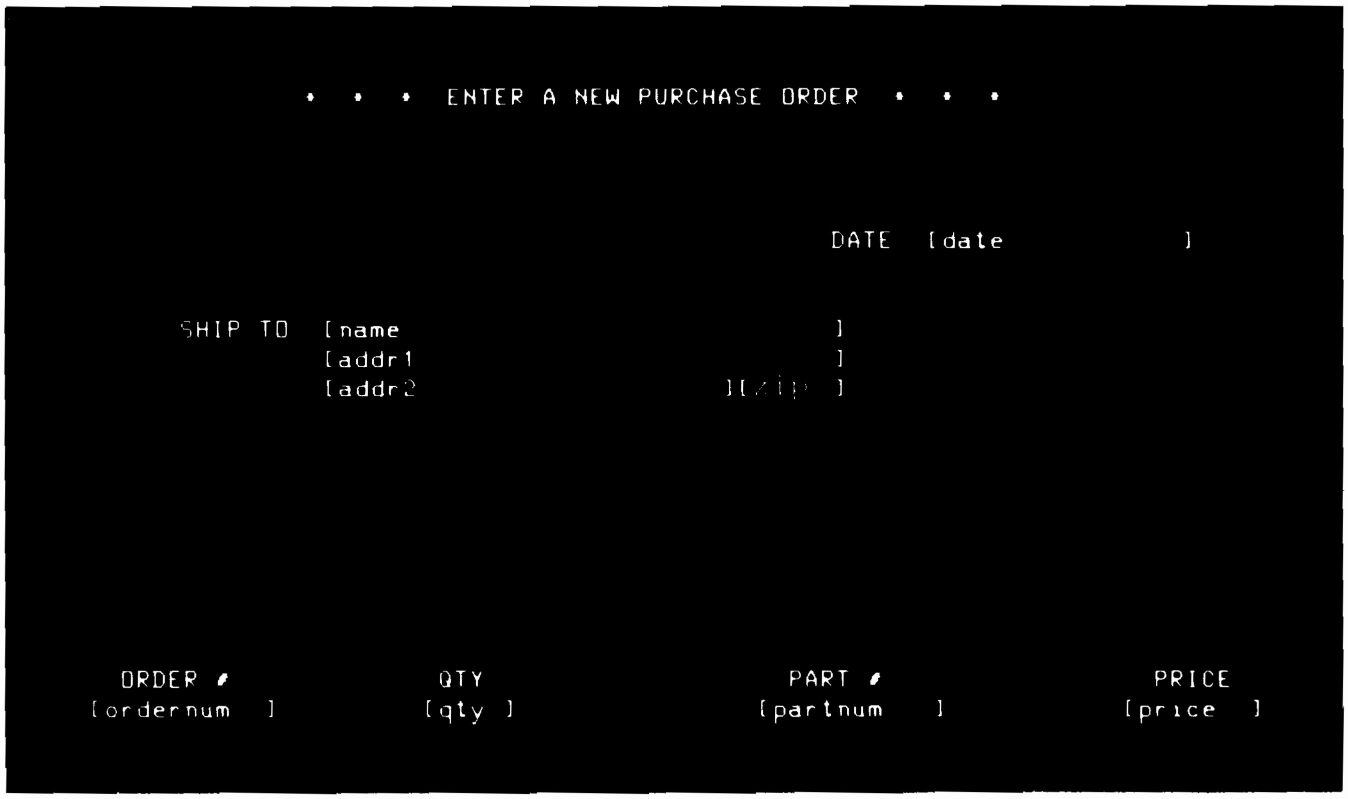
Figure 1-2. FORMSPEC Form Menu

Note that the sequence in which forms are displayed is very flexible. You can specify whether the current form is to be repeated, or repeated and appended to itself. You not only specify the name of the next form, but whether it is to replace the current form or be appended to it, and if appended, whether the current form is to be "frozen" on the screen when the next form is appended.

Next, FORMSPEC clears the screen so you can "draw" your form on a blank screen. You can use any of the terminal capabilities to insert or delete lines or characters, and to position the cursor. The special terminal capabilities that provide field enhancements such as "half bright", "inverse video", or "blinking" are entered through FORMSPEC. You do not need to use complicated escape sequences to request them.

The position of a field is defined by field delimiters. Each field is identified by typing its name within these delimiters. This name can later be changed to a longer identifier if desired (see Simple Editing below). The maximum length of each field is defined automatically by where you place the beginning and ending of the field. Field delimiters can be either brackets, or invisible control characters.

Refer to figure 1-3 for a sample screen design.



```
              * * *  ENTER A NEW PURCHASE ORDER  * * *


                                           DATE  [date           ]


      SHIP TO  [name                        ]
               [addr1                       ]
               [addr2                 ][zip ]






   ORDER #           QTY              PART #              PRICE
  [ordernum  ]      [qty ]          [partnum    ]       [price   ]
```

Figure 1-3. FORMSPEC Screen Design

At this point, the form is completely designed at the simplest level.

SIMPLE EDITING

If you want to have more control over the data entered into a form, you can use the FIELD menus issued by FORMSPEC for each field in a form. The field name and length determined by the screen design are displayed on this menu. The menu also displays the default data type, default field enhancements, and the default field type, any of which you can change by typing a new value over the displayed value. You can also assign the field a new name by which it is subsequently referenced, and you can specify an initial value for the field.

FIELD TYPE

This entry controls how data is entered in the field. For example, if a field must contain a value, simply change the field type from the default value, O for "optional", to R for "required". This causes an error message to be issued if no data is entered in the field. If you want to prevent the operator from entering data in the field because you plan to use it only to display data, change the field type to D for "display only". You may force a field to be processed even if it is blank by changing the field type to P for "process".

DATA TYPE

This entry controls the type of data entered in the field. Suppose you want to make sure that only digits are entered. Simply change the data type from the default type (CHAR for any ASCII character) to DIG.

Or, if you want to allow entry of a signed number with two decimal positions and a decimal point, you can change the data type to NUM2. If a non-numeric value is entered, an error is diagnosed.

FIELD NAME

An identifier for each field is typed within the field during screen design. Since this limits the field name to the length of the field, you may want to assign a longer identifier on this menu.

FIELD ENHANCEMENT

Unless you change it on the FIELD menu (or in a GLOBALS menu), each field is displayed in half bright, inverse video (code=HI). Other enhancements, underline, blink, and security can be combined with or replace the default enhancements, or you can eliminate field enhancements altogether.

INITIAL VALUE

When a form is first displayed, all fields are blank. You can specify in the FIELD menu, a particular value to be displayed when the form is first issued at the terminal.

Refer to figure 1-4 for an illustration of a typical FIELD menu.

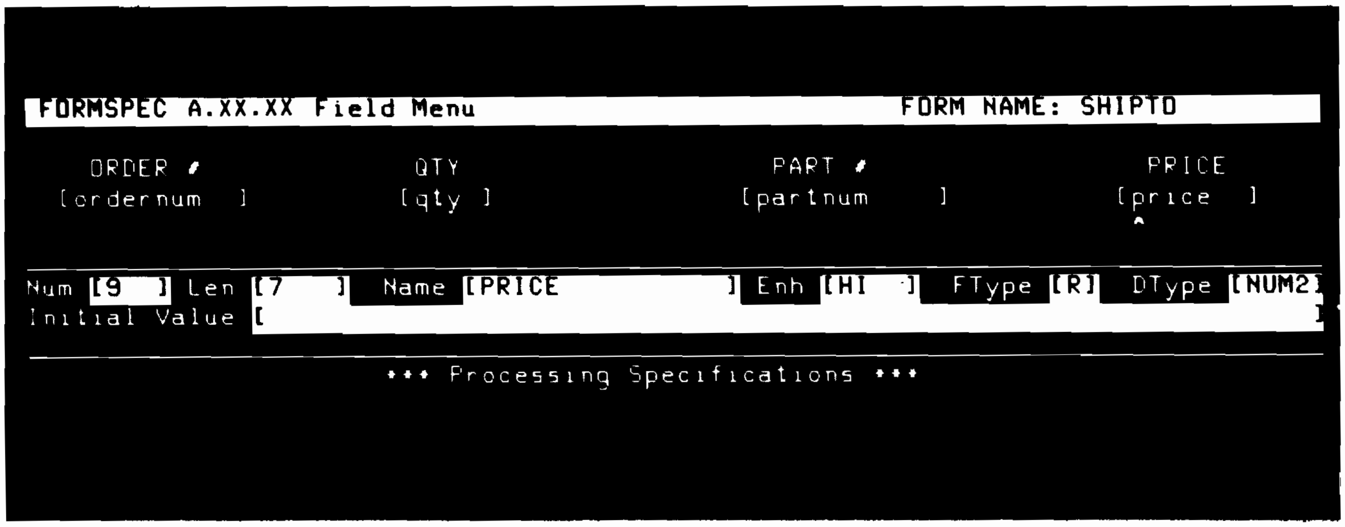

```
FORMSPEC A.XX.XX Field Menu                                      FORM NAME: SHIPTO

    ORDER #                 QTY                    PART #                   PRICE
  [ordernum  ]             [qty ]                [partnum     ]            [price   ]
                                                                             ^

Num [9  ] Len [7   ]  Name [PRICE              ] Enh [HI  ]  FType [R]  DType [NUM2]
Initial Value [                                                                    ]

                          *** Processing Specifications ***
```

Figure 1-4. Sample FIELD MENU

The field type of the item PRICE has been changed to R (required), and its data type has been changed to NUM2. By making such simple changes to FIELD menus, a form with simple edits can be completely designed.

FULL FIELD EDITS

To give you the capability to specify full field edits or advanced form and field processing, FORMSPEC provides a simple language whose statements you enter on the lower, freeform area of the FIELD menu.

These field processing specifications let you check a field value for:

Minimum Length	The value entered must be at least a specified number of characters long. For example: MINLEN 10 means at least ten characters must be entered in the field.
Equality	The value entered must be equal to, equal to or greater than, equal to or less than, less than, greater than, or not equal to a specified value. For example: GE FIELD2 means the value entered must be equal to or greater than the value entered in a different field, FIELD2.
Range Check	The value entered must be within (or not within) a range of values. For example: IN 10:20, F1:F4, 100:300 means the value must be between 10 and 20, or between the values entered in fields F1 and F4, or between 100 and 300, inclusive.
Table Check	The value entered must be in (or not in) a table of values. For example: IN 5,10,15,F7+5 means the value must be 5, 10, 15, or the current value of F7 plus 5.
Pattern Check	The value entered must match a particular pattern. For example: MATCH Aaa-ddd means the value must start with the letter "A", be followed by two letters of the alphabet, a hyphen, and three digits.
Check Digit	A check digit in the entered value is checked according to modulus 10 or 11. For example: CDIGIT 10 checks the value according to a modulus 10 check digit test.

Any of these edits is performed immediately when data is entered in the form.

ADVANCED FORM PROCESSING

In advanced processing, the sequence of specifications must be considered. As such, these statements are similar to elements of a programming language where the order in which they are entered is significant.

The advanced processing statements provide:

Data Movement	Set any field to a particular value, or to a value moved from another field. Default formatting of the data is performed during movement according to the data type of the destination field. For example: SET TO !JUNE 17, 1978! moves the date constant (delimited by exclamation points) to the current field. SET F1 TO F3 moves the value in the field F3 to the field F1.

Arithmetic Calculation	Set any numeric field to a value calculated using standard arithmetic operators (+, –, *, /, %). For example: SET TAX TO TOTAL * TAX_RATE multiplies the value in the field TOTAL by the value in the field TAX_RATE and moves the result to the field TAX. (All these fields must be numeric.)
Data Formatting	In addition to default formatting, you can specify particular formatting of the entered data. Formatting includes: STRIP, JUSTIFY, FILL, and UPSHIFT. For example: JUSTIFY RIGHT; FILL LEADING "0" moves the data to the right of the field, and then fills any leading blanks with zeros.
Forms Sequencing	Change the form sequence originally specified for the form on the FORM menu. For example: CHANGE NFORM TO $HEAD changes the next form to the head form (the first form displayed). CHANGE CFORM TO NO REPEAT stops the current form from repeating.
Conditionals	Execute a processing specification, or group of specifications, only if a particular edit is true. For example: IF F1 EQ 20 CHANGE NFORM TO CLEAR clears the current form and displays the next form when the value of field F1 equals 20.
Phases	Executes processing statements in one of four phases: CONFIG, INIT, FIELD EDITS, and FINISH.

Figure 1-5 illustrates a field menu that includes advanced processing specifications.

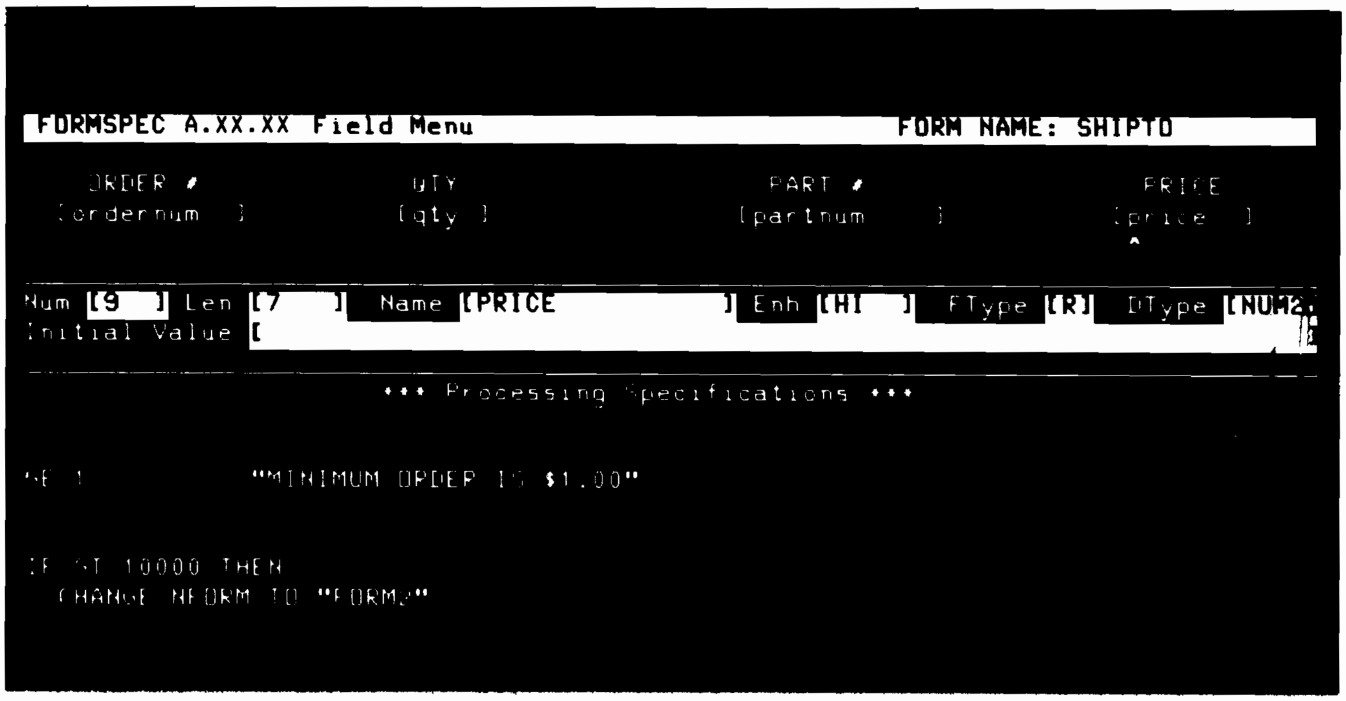

```
 FORMSPEC A.XX.XX Field Menu                              FORM NAME: SHIPTO

    ORDER #               QTY                   PART #                 PRICE
  [ordernum  ]          [qty ]                [partnum      ]        [price   ]
                                                                      ^

Num [9  ] Len [7    ]  Name [PRICE               ] Enh [HI  ]  FType [R]  DType [NUM2
Initial Value [

                         *** Processing Specifications ***

GE 1            "MINIMUM ORDER IS $1.00"


IF GT 10000 THEN
  CHANGE NFORM TO "FORM2"
```

Figure 1-5. Advanced Processing Specifications

In this example, the PRICE field (already specified as required and numeric) is further limited. The minimum value that may be entered in this field is 1. A custom error message "MINIMUM ORDER IS $1.00" will be displayed if any value less than 1 is entered by the data entry operator. Further, if a value greater than 10,000 is entered, a special next form, FORM2, will be displayed when the operator presses ENTER.

SAVE FIELDS AND OTHER GLOBAL SPECIFICATIONS

Special fields, called "save fields" can be defined for the entire forms file. These fields are global to the forms file, and can be referenced in the processing specifications of any field in any form. Save fields are useful primarily in order to pass values between forms. They are defined on a special menu, the SAVE FIELDS menu.

Certain global characteristics of a form, such as default field or error enhancements, and the placement of the error/status line on the screen are supplied by FORMSPEC as defaults. You may change these default characteristics through a GLOBALS menu.

DATA ENTRY PROGRAM

A stand-alone data entry program, ENTRY, is provided by V/3000. This program, run by a terminal operator, displays forms at the terminal, accepts and validates data entered on the forms, and writes the data to a batch file. The forms and data specifications are all entered through FORMSPEC.

ENTRY operates in two modes: Data Collection and Browse/Modify. The mode when ENTRY is first run is always data collection; the operator must request browse/modify by pressing a terminal function key. As indicated by their names, data collection mode is used to collect data from the terminal, and browse/modify to look at the collected data and modify it if necessary.

DATA COLLECTION

ENTRY displays forms on the terminal screen in the order determined by FORMSPEC. Each form is displayed with any initial values specified for the form. As each form is displayed, the operator types data into the "unprotected" fields on the form. (These are the fields that permit operator entry; they include all fields defined on a form except "display only" fields.)

After typing in the data, the operator presses the ENTER key. The ENTRY program then tests the entered data for errors and, if errors are found, it indicates each such field by an enhancement. (The particular enhancement is determined by the forms designer. Usually, the field is caused to blink.) ENTRY also displays a message that describes the first field with an error. The operator can then correct the error (or errors) and press ENTER again. ENTRY continues to check the entered data until no errors are detected. It then writes the data entered on the form, as a single record, to the batch file, and displays the next form in form sequence.

Note that form sequence is determined by the forms designer. Nevertheless, the operator using ENTRY has some control over which forms are displayed. This control is provided by a set of terminal function keys that allow the operator to:

- Request the first (or head) form in execution sequence.
- Terminate a repeating form and display the next different form.
- Continue entering data with the next sequential form after interrupting data collection.
- Clear the current form to its initial values.
- Print the current form with its initial values on the line printer.
- Request browse/modify mode to view and/or change data already written to the batch file.
- Terminate the ENTRY program.

BROWSE/MODIFY

The operator can view the data already written to the batch file by requesting browse/modify mode. (This is done simply by pressing a terminal function key; the same key returns the operator to data collection mode at the point of interruption.)

When browse/modify is requested, the previous data record written to the batch file is displayed on the form through which it was entered. The operator can examine the data, change it if desired, and then press ENTER. Any new data is written to the same batch record, overwriting data previously entered in the same form.

If the operator wants to look over the entire file, he simply presses a function key to request the first batch record. The data in this record is displayed on the form through which it was entered. When the function key NEXT is pressed, the data in the next batch record is displayed. Making changes as required and pressing ENTER, or leaving the data as is and pressing NEXT, the operator can progress through all the records in the batch file.

As with data collection, terminal function keys allow the operator further control over the sequence of browsed data. The keys for browse/modify allow the operator to:

- Display data from the first batch record on its form.
- Display data from the previous batch record on its form.
- Display data from the next batch record on its form.
- Clear the current form to the values displayed before any current modifications were entered.
- Delete the record currently displayed at the terminal.
- Print the current record on the line printer.
- Return to collection mode to enter new data.
- Terminate operation of ENTRY.

REFORMATTING DATA

Data entered through the V/3000 ENTRY program is written to a batch file. This file can then be used as input to an application program. Sometimes it is necessary to reformat the data in the batch file so that it meets the input requirements of the application program. V/3000 meets this need with a reformatting capability that allows you to:

- Combine data entered on several forms into one output record.
- Separate data entered on a single form into several output records.
- Rearrange data within a record, inserting constants, and generating check digits before writing it to the output file.
- Format data within fields by justifying, filling, stripping characters, or adjusting the sign of a numeric value.

The program REFSPEC is used to specify how you want the batch file reformatted. This program operates very much like FORMSPEC in that it issues menus, allows you to "draw" a pattern of the output record, and allows you to accept default field formatting or specify your particular formatting.

When you have completed the reformat specifications, you compile the reformat file. This file is used by the program REFORMAT to read data from the batch file, reformat it, and write it to the "output file".

REFORMAT can be run at any time after a data entry batch file has been written. It can be run from a terminal or as a batch job, requiring only the names of the batch file, the reformat file, and the output file. When REFORMAT is executed, the data in the batch file is written, according to the reformat specifications, to the output file. The output file is then ready to be used by the application.

Refer back to figure 1-1 for the flow of data between the batch file, through the reformatting specifications entered with REFSPEC, to the output file.

PROGRAM INTERFACE

A library of high-level procedures is available for use by all terminal-oriented applications. These procedures can be called by COBOL, BASIC, FORTRAN, or SPL programs. The RPG compiler has been modified so that exactly similar functions can be performed by an RPG program.

The program interface provides:

● Terminal Interface	Procedures to open and close a terminal file, to display a form at the terminal, and read data entered in fields of the displayed form.
● Forms File Interface	Procedures to open and close a forms file, to get the next form in sequence from the forms file, and to print the current form, with its contents, on the line printer.
● Data Manipulation	Procedures to initialize a form to its initial values, to perform any FORMSPEC or user-defined edits, and to perform any final form processing as defined by FORMSPEC.
● Data Entry	Procedures that open and close the batch file, write data to the batch file, or read data from the batch file.
● Access to Data	Procedures to read entered data to a program buffer, or write data from a program buffer; data from an entire form, or only from selected fields, can be read; similarly, data can be written for an entire form or only for selected fields. Data passed to or from fields can be converted to or from a variety of data types.
● Status/Error Control	Procedures to set error flags and display messages.

The V/3000 data entry program, ENTRY is implemented using these procedures. Thus, they provide a simple way to enhance or modify ENTRY. For example, if you want a special edit that is not included in FORMSPEC or you want to add the capability to access data base files, you can modify the ENTRY program to include these features. To simplify such modification, appendix A contains a listing of the ENTRY program in each of the languages: COBOL, RPG, BASIC, FORTRAN, and SPL.

The V/3000 procedures can be used for other purposes than data entry. Any application that wants to display forms on a terminal, accept data from or send data to a terminal, or validate entered data, can use the V/3000 procedures.

The V/3000 procedures can be used to:

- Transfer a form definition including processing specifications, from the forms file to the Form Definition area of memory.
- Display a form on the terminal screen.
- Transfer data from the screen to the Data Buffer area of memory.
- Transfer data and processing specifications between the Form Definition and the Data Buffer areas of memory.
- Transfer data between the Data Buffer in memory and either an application program buffer area or the batch data file used for data entry.

Refer to figure 1-6 for an overview of the various transfers between elements controlled by the V/3000 procedures. Note that many applications will use their own files rather than the V/3000 batch file for the collected data.

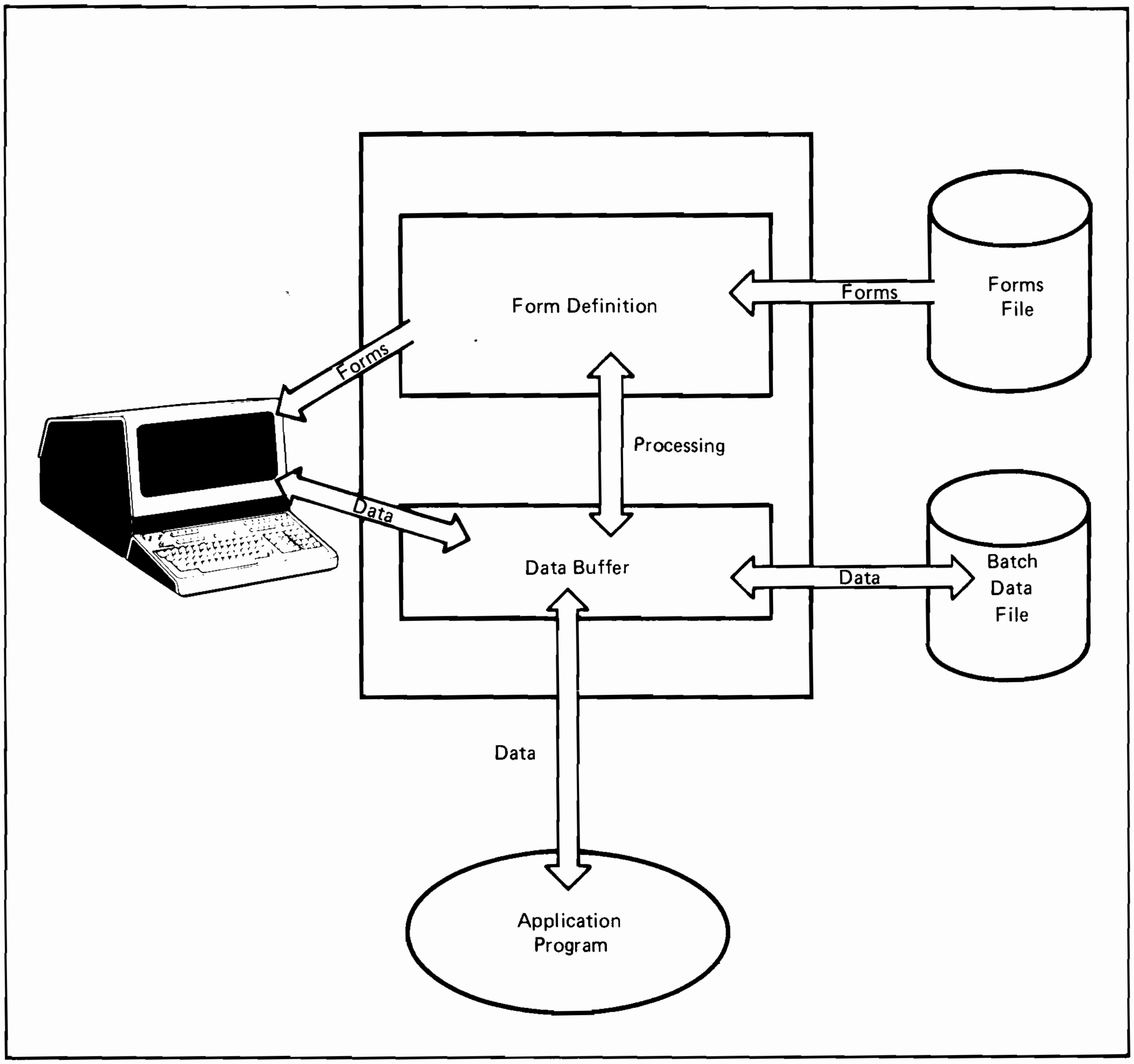


Figure 1-6. Transfers Controlled by V/3000 Procedures

USING THE MANUAL

This manual may be useful to three different types of user:

● Forms Designer	who designs the forms to be displayed, determines the order in which forms are issued, and specifies any editing or special processing to be performed on data entered through the forms.
● Applications Programmer	who designs and codes the application that uses data entered through the FORMSPEC forms, or displays data on these forms.
● Terminal Operator	who enters data on the FORMSPEC screens for processing by the V/3000 program ENTRY.

FORMS DESIGNER

The forms designer who uses FORMSPEC to design forms should read sections III and IV. These sections describe simple and advanced forms design with FORMSPEC.

The forms designer may also have the responsibility for reformatting the data entered through the forms. If so, he should also read section V that describes how to specify reformatting with program REFSPEC.

Unless the forms designer is also the applications programmer, he can skip sections II, VI, and VII.

APPLICATIONS PROGRAMMER

The first thing the applications programmer should determine is whether program ENTRY fulfills the user's application needs, and if not, whether it can be modified or must be replaced. Using ENTRY from the terminal operator's point of view is described in section II; listings of the ENTRY program in COBOL, FORTRAN, BASIC, RPG, and SPL are in appendix A.

If the ENTRY program is not suitable, the applications programmer should read section VI that describes the V/3000 procedures callable from COBOL, FORTRAN, BASIC, or SPL, or section VII that describes the RPG interface. These procedures, or the RPG actions and events, can be used in modifying the ENTRY program or in writing an entirely new program.

Since FORMSPEC provides editing and processing capabilities, the applications programmer should study FORMSPEC in order to determine how to divide the processing responsibilities between FORMSPEC and the application program. FORMSPEC is fully described in sections III and IV.

The applications programmer should also read section V that describes the specifications for reformatting data entered through FORMSPEC. Reformatting may provide a way to adapt entered data to an existing application.

TERMINAL OPERATOR

The terminal operator (or the data entry supervisor) may need to read section II. This section describes how to enter data that is processed by the ENTRY program. If the ENTRY program has been modified, or is not used at all, the terminal operator should not be given this manual. In such cases, a similar guide should be provided by the applications programmer that explains how to:

- Log on and off the HP 3000.
- Use the terminal function keys.
- Enter and modify data.
- How to run the REFORMAT program (if applicable)

ENTERING DATA WITH ENTRY

SECTION II

OVERVIEW

V/3000 provides a Data Entry Program (ENTRY) to control all data entered on forms contained in a V/3000 forms file. This section describes how an operator enters data under control of the data entry program, ENTRY. (Refer to sections III and IV for a description of forms design; and to section VI for a description of the procedures with which ENTRY is implemented and can be modified.)

In order to enter data, an operator runs the program ENTRY. Then, using the terminal in block mode, he calls up a form and enters data into the unprotected fields displayed on the screen.

PROTECTED AND UNPROTECTED FIELDS

Each form contains "protected fields" that cannot be changed by the operator, and "unprotected fields" into which the operator keys data. Protected fields may be field titles, report headings, or "display only" fields to which the system sends data. Data entered at a terminal keyboard can only be keyed into unprotected fields.

DETECTING ERRORS

If errors are detected after data is entered, the program highlights the field (or fields) in error, and prints a diagnostic message on the line of the terminal screen dedicated to error and status messages. The operator can then correct the errors and reenter the data correctly on the screen.

MODIFYING DATA

After data has been entered, the operator can review this data and, if desired, change it. The entered data can be reviewed in any order, regardless of the order it was entered.

REFORMATTING DATA

If the data is to be reformatted, the operator runs the program REFORMAT. The data will be reformatted automatically and written to a selected output file from a selected "batch" file containing the previously entered data.

NOTE

If the program ENTRY is not used or if a modified version is used, special instructions may be required. This section describes how to enter data as if the ENTRY program were used to control data entry.

USING THE TERMINAL

In order to enter data through V/3000, the terminal must be a CRT (Cathode Ray Tube) terminal that operates in block mode or an HP 3075/6 Data Capture Terminal. Any of the following terminals may be used: 2624, 2626, 2640B, 2641A, 2644A, 2645, 2647, 2648, 3075, and 3076. Depending on the terminal, the keyboard may differ slightly. In general, however, all keyboards conform to that of the HP 2645 illustrated in figure 2-1. (Refer to the User's Manual that describes your terminal for complete instructions on terminal use.)

TERMINAL KEYBOARD

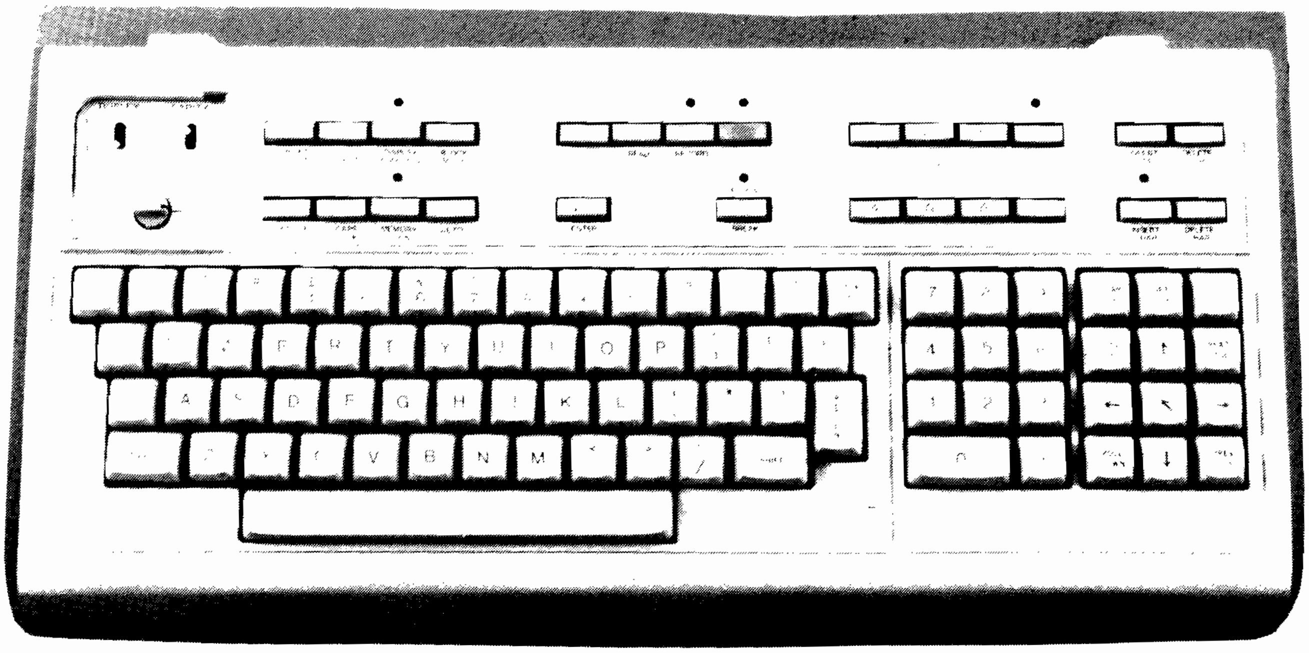


Figure 2-1. HP 2645 Terminal Keyboard

For the purpose of data entry, certain keys have special meanings. These are the eight special function keys shown in figure 2-2. In addition to these function keys, the keys listed below are also used in data entry. If you are not familiar with the terminal, you should practice using these keys before using them for data entry.

BLOCK MODE	Puts terminal in block mode so that all data can be keyed into a single form before it is actually sent to the system. When ENTRY is run, it will request that the terminal be placed in block mode if it is not already in block mode. (The 2641A, 2645, 2647 and 2648 terminals are placed in block mode automatically).
ENTER	When pressed, all data keyed into unprotected fields on a single form is sent to the sytem.
Carriage Return	Returns cursor to first position on line; it does not transmit data.

CURSOR CONTROL KEYS

At any time, the cursor (blinking underline on screen) indicates the next position in which you can key a character. When a form is first brought up on the screen, the cursor is positioned to the first unprotected field. The following keys are used to control the cursor.

TAB	Pressing the TAB key moves the cursor from its current position to the beginning of the next unprotected field. (Note that fields are ordered from left to right, top to bottom so that the first unprotected field is the first field in the upper left of the screen.) After tabbing to the last field, pressing TAB again "wraps around" to position the cursor at the first field.
BACKTAB (CNTL TAB)	Holding down the CNTL key while pressing the TAB key positions the cursor to the beginning of the preceding field. (Not available on the 2640B.)
BACKSPACE or CURSOR LEFT (←)	Moves the cursor back one character each time either of these keys is pressed.
CURSOR RIGHT (→)	This key advances the cursor forward without changing any existing characters. (The SPACE bar clears existing characters to spaces.)
CURSOR HOME (↖)	Positions the cursor to the first unprotected field in the form.

NOTE

Care should be taken using the cursor positioning keys (←, →, ↑, ↓, ↖) during data entry since they can position the cursor outside the unprotected fields where data cannot be entered.

ROLL UP/ ROLL DOWN	These keys move the entire form up (ROLL UP) or down (ROLL DOWN) on the screen. They are useful primarily when a form is longer than the screen in order to view portions of the form that have rolled off the screen.

Once the cursor is positioned, any existing character can be changed by simply typing over it. A character is cleared to a blank (space) by pressing the SPACE bar.

FIELD EDIT KEYS

Since the terminal is operating in block mode, the character control keys in the edit group can be used to insert or delete characters; other keys in the display group can be used to clear data from unprotected fields. Note that the INSERT CHAR and DELETE CHAR keys operate in one field at a time, not the entire screen.

INSERT CHAR	When lit, keying in a character inserts it before the character at which the cursor is positioned, and, if the field is full, pushes the rightmost characters out of the field, effectively losing them. Turning this key off returns to normal mode where any character keyed in replaces the character at which the cursor is positioned. The INSERT CHAR key should always be turned off after changing a particular field or it can cause problems in subsequent data entry.
DELETE CHAR	Each time this key is pressed, it deletes the character at which the cursor is positioned, and moves all successive characters one space to the left, creating blanks in the rightmost positions of the field.
CLEAR DISPLAY	Clears data in all the unprotected fields from the current position of the cursor through the last field in the form.
CLEAR FIELD (CNTL/CLEAR DISPLAY)	Holding down the CNTL key while pressing CLEAR DISPLAY clears the remaining characters in the current field.

ENTRY FUNCTION KEYS

The eight terminal function keys are assigned special functions for data entry and for subsequent data modification. Depending on whether you are entering new data into the system (data collection) or are reviewing and modifying existing data (browse/modify), these keys have slightly different meanings. The keys are illustrated in figure 2-2 followed by their function for each mode.

If you are using 2640B terminal, (and default strapping has not been changed — see appendix G), you must press the CNTL key to activate the function keys. To activate a key with CNTL, hold down the CNTL key while you press the selected function key.

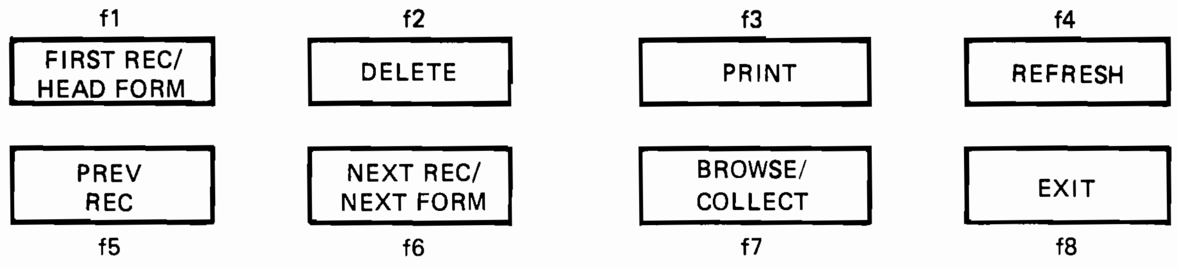


Figure 2-2. Special Function Keys for Data Entry

Key	Key Action Data Collection Mode	Key	Key Action Browse/Modify Mode
HEAD FORM f1	Display first form in sequence of forms.	FIRST REC	Display first record in batch file on form used to enter data.
		DELETE REC f2	Delete current batch record from the batch file. Note that you cannot insert a record in place of a deleted record; any new records are added to the end of the batch file.
PRINT f3	Print current form on line printer. Prints form with current data (but not any values typed on screen and not yet entered by ENTER).	PRINT	Same as in Data Collection.
REFRESH f4	Clear screen; initialize terminal; and redisplay with initial values. (Can be used to restore form should it accidentally be cleared from the screen by a local RESET or terminal power failure. If terminal power fails, a colon prompt is issued when power returns and you must type RESUME before pressing REFRESH.)	REFRESH	Same as in Data Collection except that previously entered data is displayed.
		PREVREC f5	Display previous record in batch file on form used to enter data.
NEXT FORM f6	Interrupt display of repeating form, and display next form.	NEXT REC	Display next record in batch file on form used to enter data.
BROWSE f7	Enter browse/modify mode and display previous form with entered data.	COLLECT	Return to data collection mode.
EXIT f8	Exit from ENTRY; return to MPE control.	EXIT	Same as in Data Collection.

Note that the function keys are never used to enter data; data is entered only with the ENTER key.

Key labels can be changed with FORMSPEC (as described in Section 3). However, changing a label does not affect the action associated with the key.

DISPLAY ENHANCEMENTS

The terminals used for data entry provide display enhancements to highlight portions of the forms display. These enhancements are defined during forms design and may vary depending on how the form is defined. They are, however, generally used to highlight the unprotected fields in which you can enter data, and also to highlight fields in which errors were detected. The HP 3075/6 terminals do not support display enhancements.

If not specifically changed by the forms designer, all unprotected fields are shown in inverse video, half bright. If a field is found to contain an error, it is highlighted using full bright, inverse video, and is underlined. (Note that the particular way a field in error is enhanced depends on how the form is designed, and may differ in your application from the enhancements described above.) In general, all enhancements are designed to make it easy to distinguish those fields in which data can be entered as well as those fields where data has been entered incorrectly.

Depending on the design, the enhanced unprotected fields also may be delimited by brackets to indicate exactly where data can be entered, or the unprotected fields may simply be enhanced.

PRINTING FORMS AND DATA

If you want a "hard copy" of any form, simply press the PRINT key (function f3). In collect mode, the current form with any initial data is printed on the line printer. Unless a specific initial value has been assigned to a particular field by the forms designer, all fields in the form are blank initially. In browse/modify mode, the form is printed with the data previously collected to the current record.

Basically, what you see on the screen is what is printed. However, if you key in data after the form, or the form with previously entered data, is displayed, the new data is not printed. Only initial data displayed with the form in collect mode, or data already recorded on a batch record in browse/modify mode, can be printed.

RUNNING ENTRY

In order to bring the forms onto the screen, you run the V/3000 program ENTRY. To do this, first turn on your terminal and then press RETURN. A colon prompt (:) is displayed on the screen. You log on in response to this prompt. Logging on means that you request access to the MPE operating system that controls all HP 3000 system operations. (Refer to *Using the HP 3000* for an introduction to using the HP 3000 Computer System.)

In order to log on, you must know your user name and account name (and possibly other information such as passwords or a group name). Ask your system manager for this information, and then log on with the HELLO command as follows:

```
:HELLO username.accountname
```

The words in capital letters must be entered exactly as shown; the lower case words are replaced by the specific names or numbers supplied by your system manager. Only the username and accountname are shown here; if passwords and a groupname are also needed, the system manager can explain how to enter them.

Suppose your user name is JOAN and your account name is INVENTRY, you log on with the following command:

```
:HELLO JOAN.INVENTRY
```

Then press the RETURN key to enter your log on command into the operating system.

When the system accepts the log on, it returns a "welcome" message followed by another colon prompt (:). In response to this second prompt, type the command to run program ENTRY:

```
:RUN ENTRY.PUB.SYS
```

Then press RETURN.

SPECIFYING ENTRY FILES

Once ENTRY is in control, it must know the name of the forms file in which the data entry forms are stored. It must also know the name of the "batch" file in which the data you enter is to be saved.

FORMS FILE

The forms file is a standard MPE file identified by its data file name. Forms files created by V/3000 version A.00.00 are KSAM files. It may be fully qualified by account and group name and lockword.

BATCH FILE

The batch file is a standard MPE file. If the named batch file does not exist, a new file is created automatically. If the file already exists, it is opened so that new data can be added to the end of the existing data in the file.

If you do not know the names of these files, ask your system manager.

The ENTRY program prompts for the forms file name as follows:

```
Enter Forms File name and press RETURN:
```

In response to this prompt, type the forms file name you want and then press RETURN. ENTRY then prompts for the batch file name:

```
Enter Batch File name and press RETURN:
```

Enter the batch file name. If the batch file name you enter is an existing file to which data has already been written, you may receive the following message:

```
WARNING:  Forms file recompiled since this batch was created.
Enter "Y" to continue:
```

This message is issued if the forms file has been modified and recompiled since it was last used to collect data to the batch file. Enter "Y" to continue only if you are sure that the changes to the forms file will not invalidate data already entered on the file. Otherwise, press RETURN. This causes the forms file prompt to be reissued so you can enter a new forms file and/or a new batch file name. Generally, you should use a new batch file when a forms file is recompiled.

If the batch file you named was originally used with a different forms file than the one you named, you will receive this message:

```
WARNING:  A different forms file was used to create this batch.
Enter "Y" to continue:
```

This message may mean that the wrong forms file was entered. If you press RETURN, the forms file prompt is issued again so you can enter the correct forms file name. If the forms file name was correct, but the batch file name is wrong, you must reenter the forms file name and then enter a different or new batch file name when the batch file prompt is issued. Normally, in this situation, you will not enter "Y" to continue since it makes no sense to use a batch file that does not match the forms file.

EXPANDING THE BATCH FILE

In most cases the batch file built for you by ENTRY has enough space, but if you wish to enter a large amount of data (large enough to fill over 1023 copies of a form, or 1023 records), you must either build a larger batch file before running ENTRY, or increase the size of your existing batch file.

To build a larger batch file, first use this MPE command before running the ENTRY program:

:FILE *filename*; DISC = *numrec*

where *filename* is the name of the batch file and *numrec* is the number of records you want the new batch file to have. When you run ENTRY, specify *filename* as your batch file, and the program will build it to have the number of records you requested.

You use the HP file copier to enlarge an existing batch file. Once again, issue the MPE file command:

:FILE *filename*; DISC = *numrec*

Then run the file copier program to build a new, larger file of the size you requested and copy the existing data into it:

```
:RUN FCOPY.PUB.SYS
```

When you see the FCOPY prompt, a "greater than" sign, type

FROM = *oldname*; TO=**filename*; NEW

where *oldname* is the name of your existing batch file and *filename* is to be the name of your new, larger file. Remember to include the asterisk; otherwise the new file will be no larger than the old one. After the "greater than" sign reappears, type EXIT and press RETURN.

The last step is to purge the old batch file from the system and give its name to your new batch file:

:PURGE *oldname*
:RENAME *filename*, *oldname*

where *oldname* and *filename* are the names of your old and new batch files, respectively. When you next run ENTRY, specify *oldname* as the batch file, and you may continue entering data as before.

BLOCK MODE

After the ENTRY files have been specified, the ENTRY program operates in "block mode". ENTRY either places your terminal in block mode automatically or, for HP 2640B and HP 2644A terminals, it asks you to press the BLOCK MODE key.

During log on, when you request the program ENTRY, and when you specify the forms and batch file names, your commands are entered into the system by pressing the RETURN key at the end of each line. When ENTRY operates in block mode, your data is entered only by pressing the ENTER key. This allows you to move around on the screen, pressing RETURN if you wish, and to type in or correct data. The system does not receive any keyed data until ENTER is pressed.

Once the forms and batch files have been specified, ENTRY starts displaying the forms defined in the specified forms file. The data you enter in each form displayed on the screen is stored in separate records of the specified batch file. Subsequently, you can look at or modify the data stored in the batch file as described below under "Data Modification".

FORMS SEQUENCE

The first form displayed on the screen after the forms and batch files have been specified is known as the HEAD form. Depending on the forms file definition, this form may be a "menu" type form on which you can select the particular form on which you want to enter data. If it is such a "menu" form, the data you enter determines which form is displayed next.

The next form displayed after the HEAD form also depends on the definition of the forms file; it is usually a data form on which you enter data in unprotected fields. All such data is stored in the batch file when you press ENTER and at that time, ENTRY automatically displays the next form in the sequence of forms in the forms file. This process continues until all the forms in the forms file have been displayed, or you press the EXIT key (f8).

REPEATING FORMS

The next form is not always a different form. For instance, a form used for order entry may be repeated over and over as you enter different data into the form. Each time you press ENTER to enter data for such a form, the next form displayed is identical to the preceding form except that the unprotected fields are clear or contain initial values.

You interrupt a repeating form in order to display the next form by pressing the NEXT FORM key (f6). In this case, the form that immediately follows the repeating form is displayed. If you know that the next form is the first form (the "head" form), you can also display this form by pressing the HEAD FORM key (f1).

APPENDED FORMS

Forms may also be designed so that the next form is appended to the current form. An appended form is displayed immediately below the current form on the screen. When you type data into the current form and then press ENTER, the data in the current form is written to the batch file as usual, but the form and its data remain on the screen with the next form displayed below it. Note that an appended form may also be a repeating form, or it may be different from the preceding form.

You may then enter data in the appended form, but not in the previous form. Although the previous form remains on the screen, all its fields are now protected. When all appended forms have been displayed, the next form is displayed as usual.

FROZEN FORMS

A form may be designated as "frozen". Such a form remains on the screen when subsequent forms are displayed. The next form after a frozen form is always appended to the frozen form. As data is entered into the next form (or forms), at some point, depending on the form size, there will be no more room on the screen. At this point, the top appended form is rolled off the screen leaving the frozen form at the top of the screen. Forms rolled off the screen cannot be viewed with the ROLL DOWN key. Data in the frozen form can be changed only by entering the browse/modify mode with the BROWSE key.

Figure 2-3 illustrates a possible sequence of forms, including a repeating appended form that follows a frozen form.

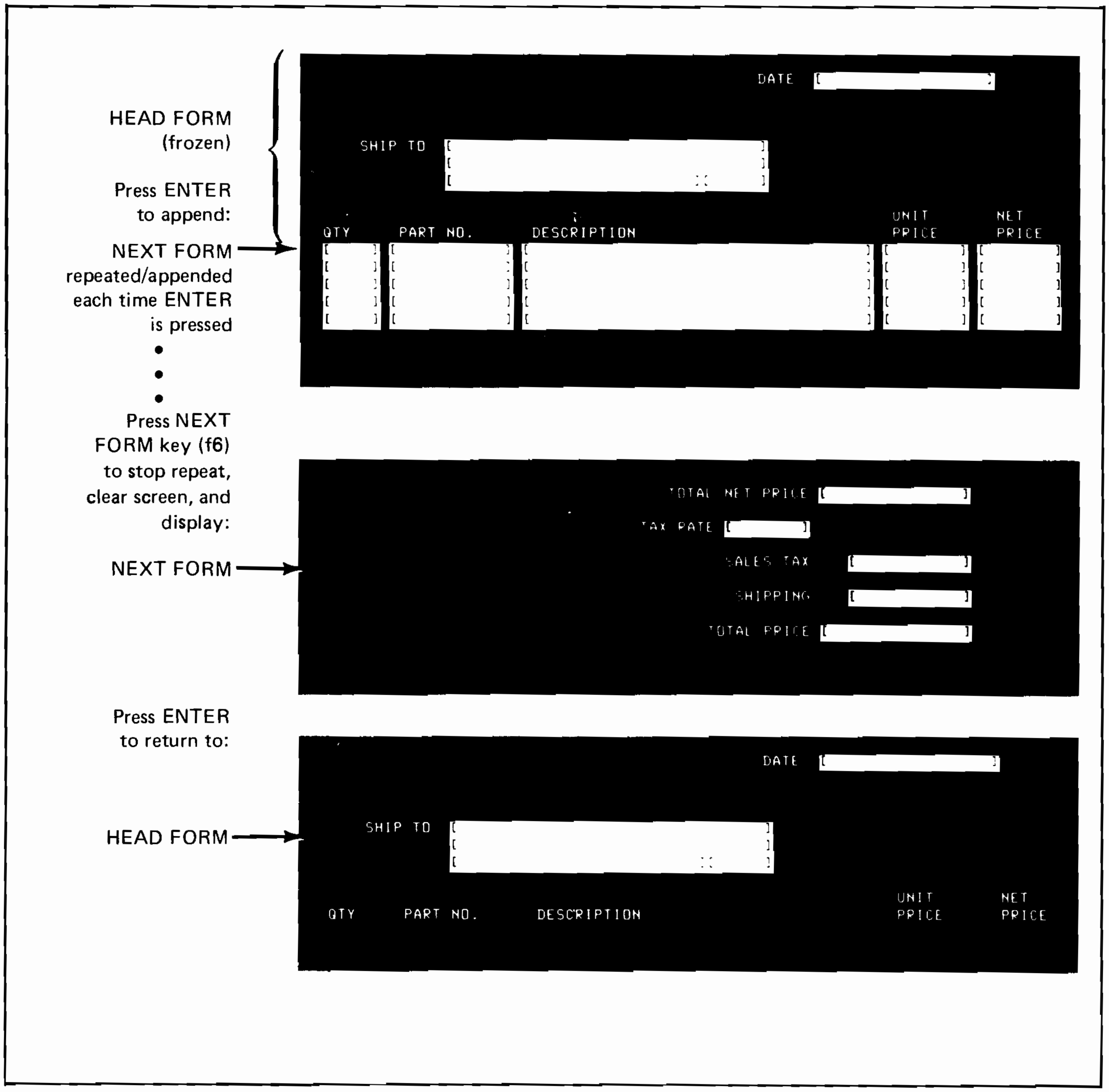


Figure 2-3. Example of Forms Sequence

ENTERING DATA

Data can be entered only into "unprotected" fields. These fields may be distinguished on the screen by display enhancements, usually half bright, inverse video.

Depending on how the forms are designed, unprotected fields may be delimited by brackets as shown in figure 2-4, or they may be designated simply by the enhanced display as shown in figure 2-5, or they may not be distinguished from the protected areas of the screen at all. The TAB key positions the cursor to the beginning of the next unprotected field.

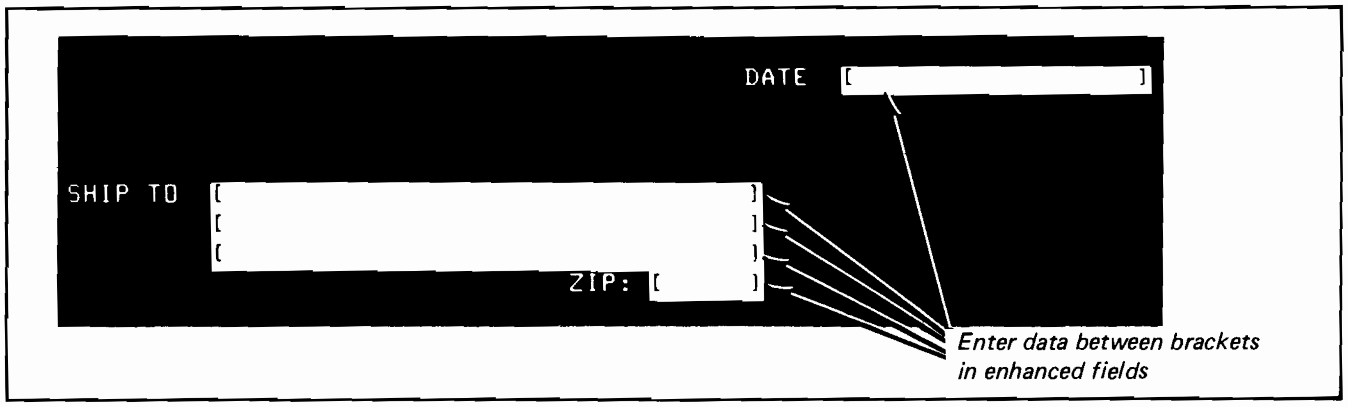


Figure 2-4. Bracketed Fields

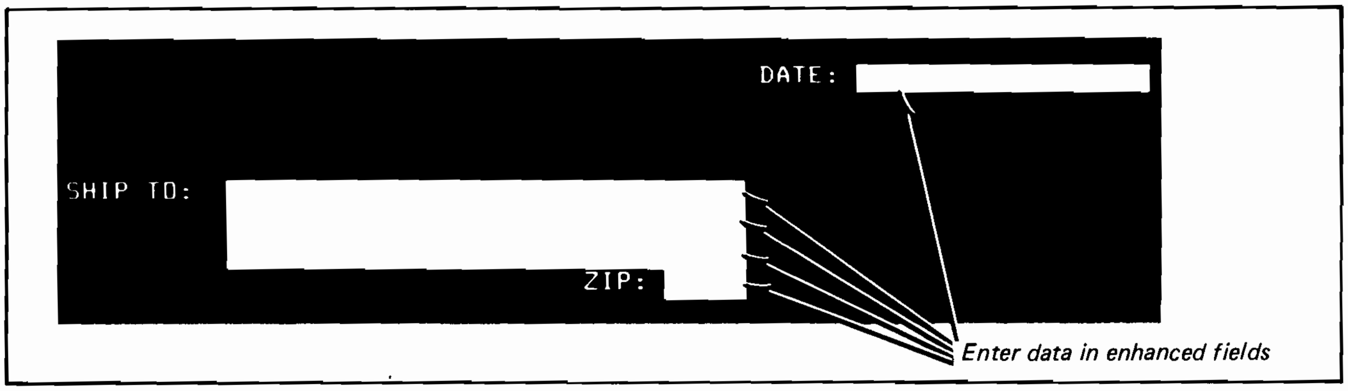


Figure 2-5. Fields Without Brackets

In either case, you enter data anywhere within the enhanced field. The brackets, although they are enhanced like the field, cannot be overwritten by data. If data keyed into a field fills the field, the cursor is positioned automatically to the beginning of the next field. If data does not fill the field, press the TAB key to go to the next field.

After keying all the data into the unprotected fields on the screen, press ENTER. The data is collected by ENTRY and tested for errors. If no errors are detected, the data is then written as a record to the batch file you named when you first ran ENTRY.

Each time you press ENTER, a new record is written to the batch file. Thus, each batch file record is associated with the data entered on a single form. If a number of appended forms are displayed on the screen, the data entered on each form is written to a separate record in the batch file. (The relation of records to forms becomes important during the browse/modify phase of data entry described below under the heading "Data Modification".)

OPTIONAL AND REQUIRED FIELDS

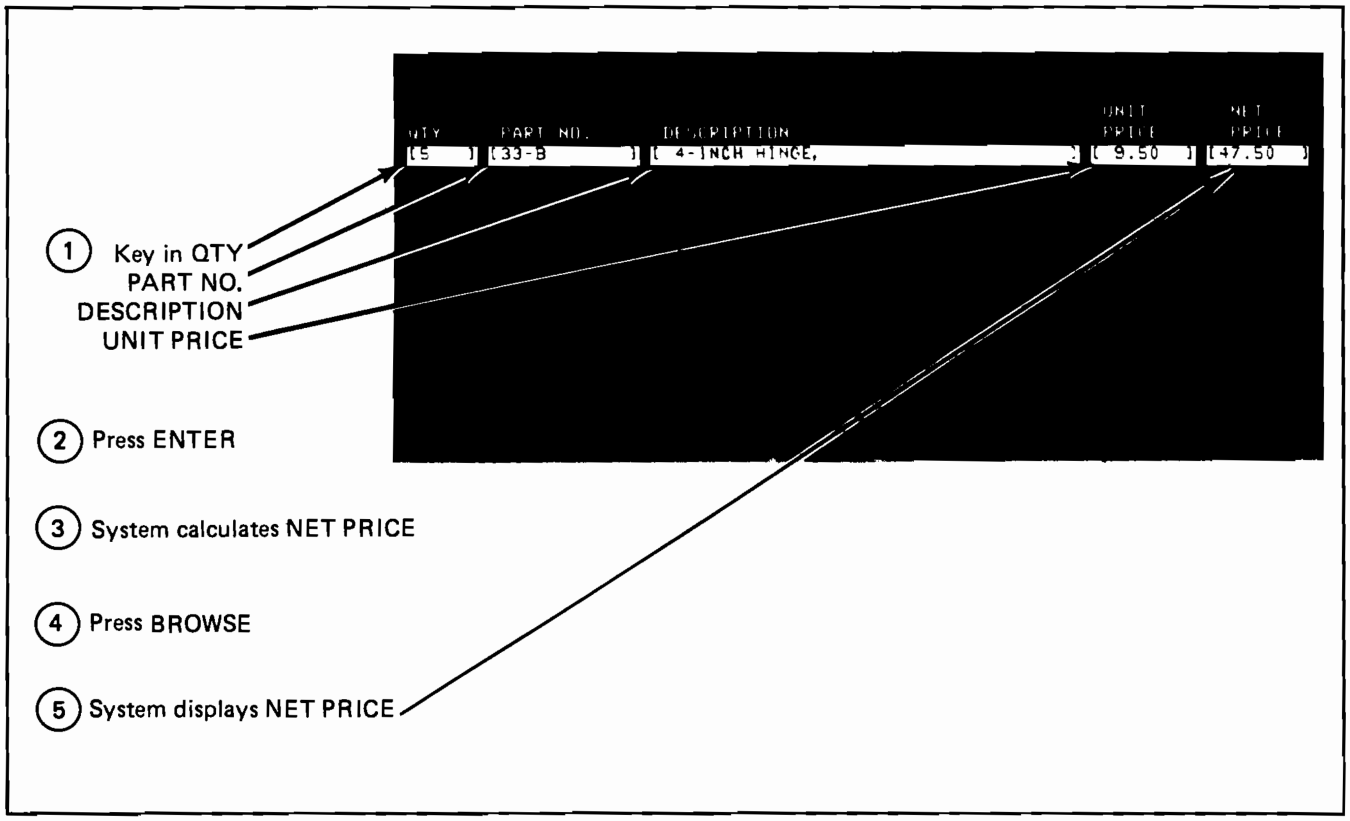
During forms design, certain fields in which you enter data may be defined as "required". If a field is required and you leave it blank, an error is diagnosed. Fields defined as optional may be left blank.

PROGRAM GENERATED DATA

When you press ENTER, values may be assigned to certain fields in the form. When this happens it is because the program has calculated values from values you entered, has moved values from other fields in this or another form, or has specified actual values to be displayed in these fields. Whatever the origin of program generated data, if it is displayed in a display-only field, you cannot change this data at the terminal.

For example, you might enter the quantity of an item and the unit price of the item. The program uses these values to calculate the net price for this line of the order and, next time you are in browse mode, will display the price in a field on the form. Figure 2-6 illustrates such a situation.

Another possibility is that a value you enter is edited to a new format. For example, a date you enter as September 15, 1978 may be displayed in the same (or a different) field as 9/15/78.



```
                                                                UNIT      NET
QTY      PART NO.        DESCRIPTION                            PRICE     PRICE
[5   ] [33-B        ] [ 4-INCH HINGE,                         ] [ 9.50  ] [47.50  ]
```

Figure 2-6. Example of Data Entry

CORRECTING ERRORS

You can correct errors either before or after pressing the ENTER key. If an error is detected by the system after ENTER is pressed, the data is not written to the batch file until all errors in the form are corrected. (Data can be changed after being written to the batch file with the browse/modify capability described below under "Data Modification".)

BEFORE ENTER

It is good practice always to look over the screen for errors before pressing ENTER. If you notice an error anywhere on the screen, you can correct it and then press the ENTER key.

If the error is in the field in which you are currently keying data, you may use the BACKSPACE key or any of the cursor positioning keys to position the cursor to the character you want to correct. Remember that the SPACE bar replaces a character with a space, so that you should use the CURSOR RIGHT (→) key to move the cursor forward within a field unless you want to clear the field. Once the cursor is positioned, you may use the INSERT CHAR or DELETE CHAR keys to make corrections.

If the error is in a different field, use the CNTL/TAB key (only on the 2645) to position to any preceding field, or the TAB key to position to a subsequent field. These keys position the cursor to the first character in the field.

Within a field, all characters after and including the current character can be cleared with the CLEAR FIELDS key (CNTL/CLEAR DISPLAY). Within the screen, all unprotected fields following the cursor (including data in the current field) are cleared with the CLEAR DISPLAY key. All unprotected fields on the screen can be cleared to spaces or the original default values by pressing the REFRESH key (f4).

AFTER ENTER

Once you have pressed ENTER, the system takes over and edits the data you have entered. If ENTRY detects any errors in the data you entered, it leaves the form and the entered data on the screen, positions the cursor to the beginning of the first field with an error, and causes all fields with errors to blink (or, depending on the forms design, to be enhanced in some other way). ENTRY also issues a message describing the first error in the "window" line. This line is dedicated to error and status messages, and is often the bottom line of the screen.

You should then correct the field with an error and press ENTER. If more than one field contains errors, you may correct them all before pressing ENTER. However, if you do not know what caused an error, correct as many fields as you can and then press ENTER. A description of the first field that still has an error will be displayed in the "window" line. This message should provide the information you need so you can correct the error and press ENTER again. Continue in this way until all errors have been corrected.

SYSTEM AND LOGIC ERRORS

Some errors cannot be corrected as described above. When a system error occurs, the program terminates and returns to MPE control. An MPE error message is displayed on the screen. System errors are caused by problems in the computer system.

Other errors are logic errors. Such errors do not terminate the program. A logic error is not necessarily the fault of the operator but may be a result of how the form is designed. A logic error might occur, for example, when data entered causes the program to perform an impossible calculation, such as division by zero.

Whether the error is a system or a logic error, in such a situation, you should consult with your system manager for the best method of correcting the error.

INTERRUPTING DATA ENTRY

If you want to interrupt data entry before the last form has been reached, press the ENTER key to record the data on the current form, and then press the EXIT key (f8). The next time you run program ENTRY with the same forms file and the same batch file, the next form is displayed automatically. Thus, you can continue entering data from the point where you left off.

If you press the EXIT key before pressing ENTER, data keyed into the current form is not recorded in the batch file. When you run ENTRY again with the same forms and batch files, the last form is redisplayed with any initial values, and you must retype the data into the form whose data entry you interrupted in the previous session.

TERMINATING A SESSION

You can terminate a data entry session by pressing the EXIT key to return to MPE control. This will act exactly as described above for interrupting data entry. (Note that you can recognize that MPE is in control when a colon prompt (:) is issued.) Then you terminate the session by entering the BYE command, as follows:

```
:BYE
```

Then press RETURN.

Remember that as soon as you exit from the data entry program ENTRY, the terminal must be released from block mode. If your terminal is an HP 2640B, or 2644A, you must release the BLOCK MODE key before entering the BYE command. For any other terminal, release from block mode is automatic.

UNEXPECTED PROGRAM INTERRUPTION

The program may interrupt unexpectedly because of a terminal power failure or because you accidentally pressed the BREAK key. In either case, MPE, the HP 3000 operating system, is in control when the power returns. Press RESET TERMINAL (twice on a 2645) and then press the RETURN key. The colon (:) prompt is displayed at the terminal. To return to ENTRY, type RESUME in response to this prompt. (If the keys you type do not appear on the screen, echo has been turned off; you must press the ESC key and then press the colon key (:) to restore echo.)

After you type RESUME, MPE issues the message READ PENDING. You then press the REFRESH key (f4). The form displayed at the time of the failure is redisplayed, and you can continue where you left off.

If you press the BREAK key accidentally, control also returns to MPE. And in this case too, you type RESUME in response to the colon prompt, and, when MPE displays READ PENDING, you press the REFRESH key to resume execution of ENTRY where you left off.

MODIFYING DATA

Data that has been written to a batch file in data collection mode can be viewed and modified in browse/modify mode.

ENTERING BROWSE/MODIFY MODE

If you are currently in data collection mode, simply press the BROWSE key (key f7) to enter browse/modify mode.

If you have exited from program ENTRY, you must run the program again. When the forms file menu is displayed on the screen, enter the name of the forms file used to enter the data and the name of the batch file to which the data was collected. Program ENTRY displays the next form on which you can enter data. To enter browse/modify mode in order to view data already entered, press the BROWSE key (f7).

When the BROWSE key is pressed, the data in the last record of the batch file to which data has been written is displayed. If you then press the PREV REC key (f5), a previous record is displayed. You can also press the NEXT REC key (f6) to display the next record in a forward direction. Of course, you can only view data that has already been entered.

You may press the FIRST REC key (f1) to display the data from the first record in the batch file. Then you can press NEXT REC to proceed sequentially through the records in the batch file. You can continue to press NEXT REC until you have viewed each record in the file that contains data.

When the last form with data has been displayed, pressing NEXT REC causes the message "There are no more batch records" to be displayed. If you press PREV REC after displaying the first form, ENTRY responds with "There are no previous records".

NOTE

The keys PREV REC and NEXT REC in browse/modify mode refer to recores in the batch file rather than to forms in the forms file.

RESUMING DATA COLLECTION

To return to data collection mode, press the COLLECT key (key f7). The next form in sequence on which data is yet to be entered will be displayed. You can then continue to enter data into the batch file that you have been reviewing in browse/modify mode.

NOTE

The BROWSE and COLLECT keys are physically the same function key (f7). If pressed in data collection mode, the key is called BROWSE; in browse/modify mode it is called COLLECT.

VIEWING THE DATA

When you browse the data in the batch file with PREV REC (f5) or NEXT REC (f6), the selected form is displayed with data previously entered at the keyboard or generated by the system. That is, you see the form with its data exactly as it was when ENTER was pressed, with no errors and with any system generated data displayed.

The data from only one record in the batch file is displayed on the screen at a time. If the data is from a set of appended forms, the forms and data are appended when you press the NEXT REC key. However, when you move backward with the PREV REC key, appended forms with data are displayed one at a time.

MODIFYING THE DATA

You can modify any unprotected field by typing over the data displayed in the field. When you press ENTER, the modified data is written to the associated record in the batch file. Existing values are overwritten by the new values.

When you change a field, the system determines if the change affects other fields in the form. For example, if you change the quantity of an item on an order form, the net price will be recalculated if, during normal data entry, the system calculates net price from the quantity and a unit price.

The modified data is rechecked according to any specified editing. If changing one field causes another field dependent on it to fail the edit check, you must correct the field that now has an error.

You may use the REFRESH key (f4) to reset the data on a form during modification. However, in browse/modify mode the system redisplays the data previously collected to the batch file wiping out any changes just entered; it does not clear the fields to blanks or reset initial values.

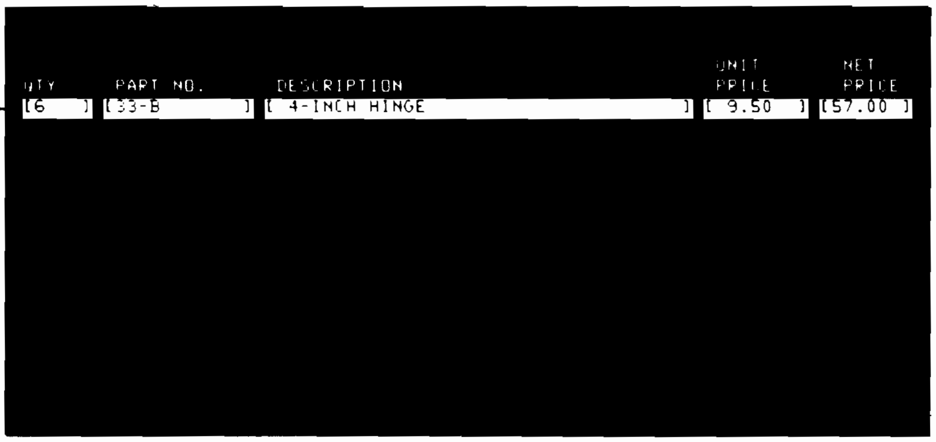
DELETING DATA

You may delete an entire record from the batch file by pressing the DELETE REC key (f2). Thereafter, the system skips over this record when you browse through the data in the batch file.

You must not delete a record expecting to insert a new record in the same position as the deleted record. ENTRY has no provision for inserting records; any new records are written to the end of the batch file.

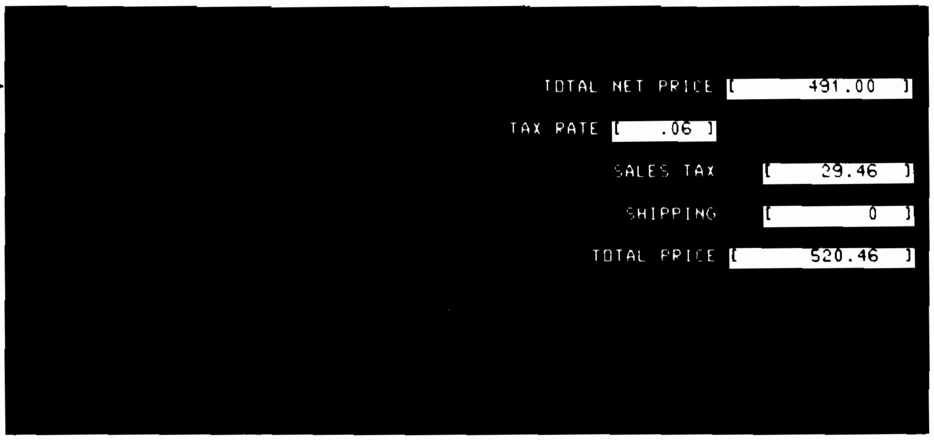
If you only want to delete a single field, key in spaces over the characters in the field or press CNTL/CLEAR. This in effect modifies the field to blanks.

Press BROWSE (f7) to enter Browse Mode and display previous record.



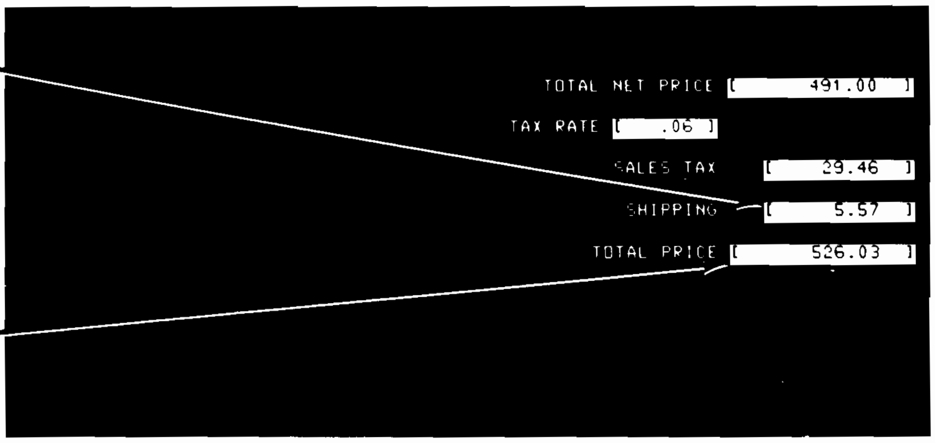
```
                                                              UNIT       NET
QTY     PART NO.         DESCRIPTION                          PRICE      PRICE
[6  ] [33-B         ] [ 4-INCH HINGE                      ] [ 9.50   ] [57.00 ]
```

Press PREV REC until particular record to be modified is displayed.



```
                    TOTAL NET PRICE [        491.00   ]
               TAX RATE [   .06 ]
                         SALES TAX    [    29.46   ]
                          SHIPPING    [        0   ]
                       TOTAL PRICE [         520.46   ]
```

Modify SHIPPING COST



```
                    TOTAL NET PRICE [        491.00   ]
               TAX RATE [   .06 ]
                         SALES TAX    [    29.46   ]
                          SHIPPING    [     5.57   ]
                       TOTAL PRICE [         526.03   ]
```

System recalculates TOTAL PRICE

Press COLLECT (f7) to continue data collection

Figure 2-7. Example of Data Modification

INTRODUCTION TO FORMS DESIGN

SECTION III

OVERVIEW

The forms displayed to a terminal operator are designed at a terminal with the interactive program, FORMSPEC. Using FORMSPEC, you design the forms, define any editing to be performed on data entered on the forms, and specify data to be calculated and displayed on the forms. The forms defined in this manner are saved in a file. At any time during forms design, FORMSPEC allows you to rearrange or modify the forms in this file.

LEVELS OF FORMS DESIGN

Forms design as implemented with FORMSPEC can be divided into four levels depending on the complexity of the design. These four levels can be grouped into simple and advanced forms design as follows:

- *Simple Forms Design* — no special edits, calculations, or data movement. Simple forms design can be further divided into data collection and editing.

Simple Collection	— Design the screen and use the default field specifications based on the screen design.
Simple Editing	— Design the screen and use only the basic field and data type edits.

- *Advanced Forms Design* — complete field edits, data movement, and data calculations. Advanced forms design consists of full field edits and advanced processing.

Full Field Edits	— Specify a full range of field edit statements but no data movement, data formatting, or conditional processing.
Advanced Processing	— Specify data movement, formatting, conditional processing of fields and forms, processing phases, and creation of form families.

This section describes only the simplified forms design that includes simple collection and simple edits. Advanced forms design using the full capabilities of FORMSPEC is described in section IV.

Note that many applications will need only the simple forms design capabilities described in this section. These provide:

- field initialization
- unconditional forms sequencing
- data editing based on field and data type.

EASE OF FORMS DESIGN

When you run the interactive program FORMSPEC, it issues a series of "menus" on which you enter the specifications to define the forms for a single application. Each menu uses a fill-in-the-blanks type of format.

A FORM menu allows you to define the characteristics of each form. This menu is followed by a blank screen on which you design the layout of the form as it will appear on the screen. This layout is easy to draw on the screen — and easy to change at this stage or later. The basic default characteristics of each field in the form are determined by the screen design. If you can use these characteristics without change, the entire form design is complete at this point.

After laying out the screen design, FORMSPEC issues a menu for each field. These menus display the form and give the default characteristics of the current field. At this point, you can accept the defaults, specify any simple edits or, if you want to use the advanced processing language, you can enter appropriate processing specifications.

This sequence is repeated until all forms in the file are defined.

FORMS MODIFICATION

At any time during forms design, you can change any form or field currently specified in the forms file. Terminal function keys or the MAIN menu allow you to return to any existing form or field specification. You can then simply change the field or form and press ENTER. The new specifications override those previously entered.

The same MAIN menu allows you to delete an entire form or to delete fields global to all forms (save fields). If you want a printed copy of an existing form, the MAIN menu allows you to list the form on an offline device.

COMPILING THE FORMS FILE

The forms defined through FORMSPEC are written to a forms file. The forms are initially stored as a "source" version. This source is modified if you change the forms file, but it must be compiled before it can be executed by an application. The compiled version, on the other hand, can be executed but not modified. The source version of the forms file is kept along with the compiled version for purposes of display and modification. Once modified, the source version must be recompiled before it can be executed. However, unless a save field is modified, only those forms which were changed are actually recompiled.

Note that the source version is the sequence of specification records entered on the menus. The compiled version is the sequence of screens issued to the operator.

FORMS FILE

The forms file consists of "global" specifications that apply to all forms in the file, followed by the individual forms specifications. Within each form specification, the form is identified, and its screen layout is designed. The screen layout defines each field into which data can be entered. Each of these different types of specification is entered on a menu issued by FORMSPEC. (Refer to figure 3-1 for an illustration of a prototype forms file.)

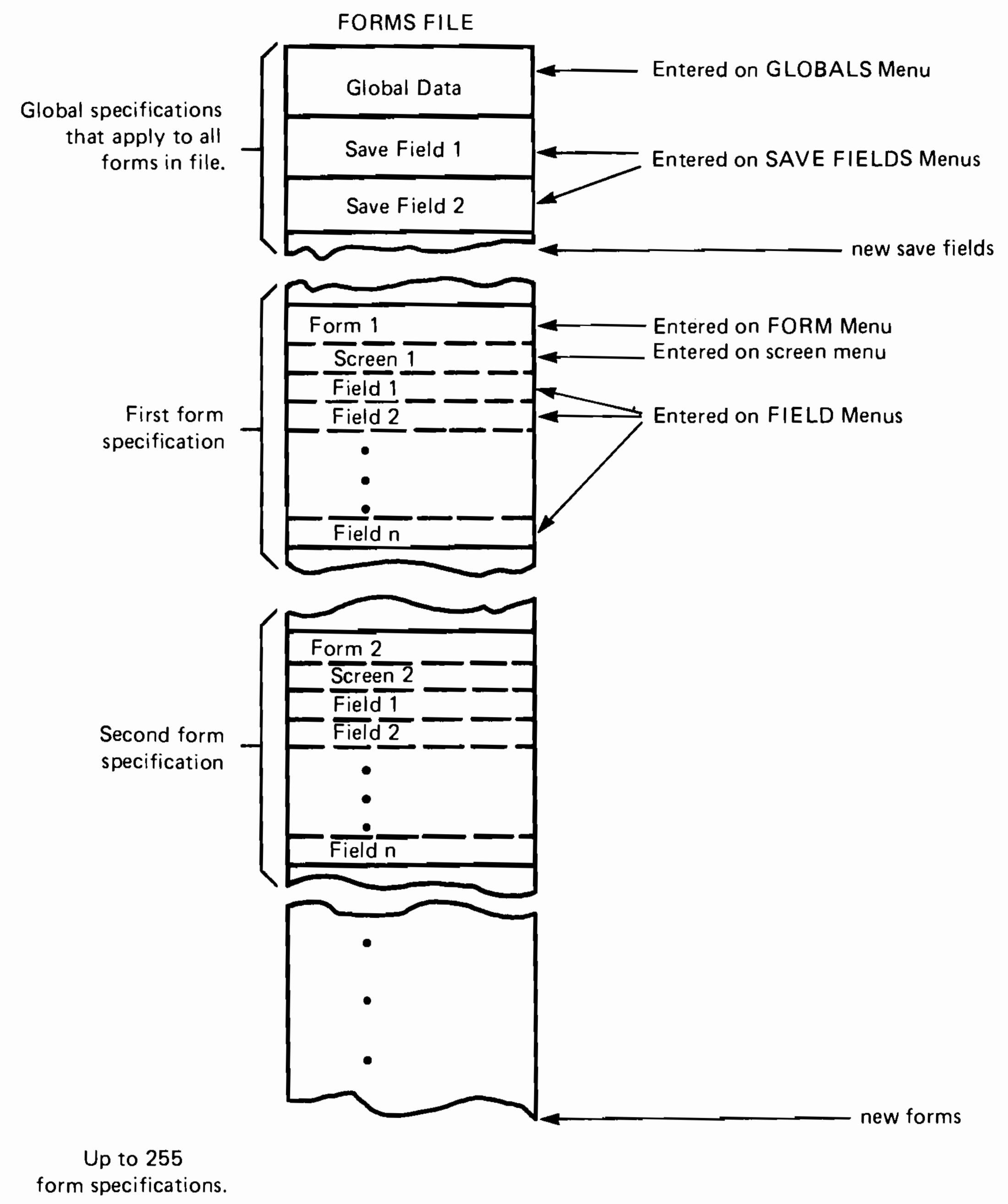


Figure 3-1. Forms File Prototype

USING FORMSPEC

You execute program FORMSPEC by entering the following command in response to the MPE colon prompt:

```
:RUN FORMSPEC.PUB.SYS
```

Since FORMSPEC runs entirely in block mode, you will be asked to press the BLOCK MODE key if your terminal is not already in block mode.

FORMSPEC prompts for information with menus. The information entered on these menus defines the forms file source version. The first menu issued whenever you run FORMSPEC is the MAIN menu. This menu allows you to specify exactly what you want to do — add, delete, copy, or list a form, go to a particular menu, compile the defined forms for execution, or select the terminal types which can use the forms file.

In addition to the control provided by the MAIN menu, a number of terminal function keys allow you to select menus either for initial definition or for modification of an existing forms file. The combination of the MAIN menu and function keys gives you the ability to change any form, field, or global specification as you define the forms file.

FORMSPEC FUNCTION KEYS

A set of eight function keys are used during creation and modification of a forms file. These keys are illustrated in figure 3-2.

```
****************************************************************************
*                                                                          *
* If you are using a 2640B terminal (and default strapping has not been changed — see appen- *
* dix G), you must press the CNTL key to activate all function keys except f2 and f3 when *
* used as start-field and stop-field delimiters. On all other terminals, the CNTL key is used *
* only to activate f2 and f3 when uses as start-field and stop-field delimiters. To activate a key *
* with CNTL, hold the CNTL key down while pressing the selected function key. *
*                                                                          *
****************************************************************************
```

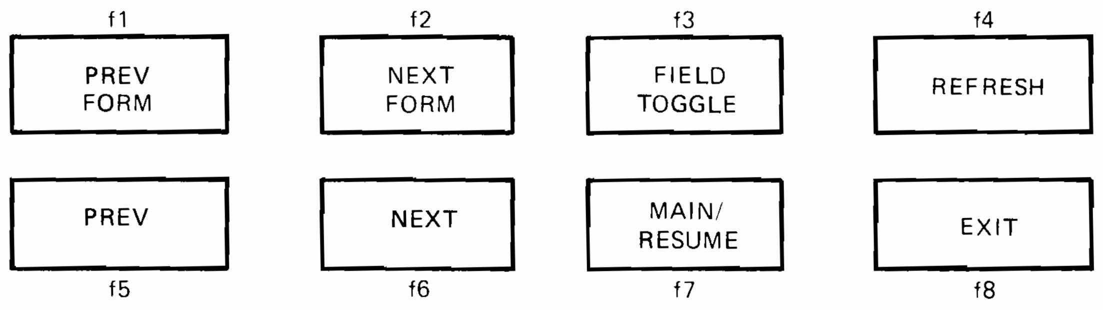


Figure 3-2. Special Function Keys for FORMSPEC

KEY LABELS

V/3000 assigns labels and actions to the eight function keys. These are defined in Table 3-1.

Table 3-1. FORMSPEC Key Labels

Key	Key Action
PREV FORM f1	Display the previous form menu. If no forms are defined, a form menu with no values is displayed.
NEXT FORM f2	Display the next form menu. If the next form is not defined, a form menu with no values is displayed. (Also used with CNTL key as alternate non-displaying delimiter for beginning of field; a left bracket, [, is standard start-field delimiter.)
FIELD TOGGLE f3	Used in FIELD menus only, to switch between standard field menu specifications and the optional processing language specifications. (Also used with CNTL key as alternate non-displaying field delimiter for end of field; a right bracket,], is the standard stop-field delimiter.)
REFRESH f4	Redisplay current menu in its initial state before any specifications were entered or existing specifications modified. Also used to recover in case of terminal RESET, BREAK key, or other unexpected program interruption.
PREV f5	Position to previous menu in sequence of menus.
NEXT f6	Position to next menu in sequence of menus.
MAIN/RESUME f7	Request MAIN menu or, if MAIN menu displayed, return to menu displayed when MAIN was requested.
EXIT f8	Terminate FORMSPEC and return to MPE control.

The function keys used by FORMSPEC should not be confused with the function keys used by an operator during data entry (see figure 2-2). Although the two groups of keys are physically the same programmable keys, their functions differ in most cases.

FUNCTION KEY LABELS

You may change key labels by using special menus provided in FORMSPEC. The new labels you assign will appear when ENTRY.PUB.SYS is executed. Note that changing key labels does not affect the action associated with that key.

You can designate new labels on a GLOBAL and/or LOCAL form level. (See "Define Function Key Labels" for GLOBAL or "Local Form Function Key Labels" for LOCAL labels below for more detail.)

TERMINATING FORMSPEC

You can terminate operation of FORMSPEC at any time by pressing the EXIT key (f8). EXIT returns you to MPE control which then issues the MPE colon prompt (:).

When you next run FORMSPEC after terminating and request the same forms file, the MAIN menu is displayed on the screen. You may select an option on the MAIN menu.

UNEXPECTED PROGRAM INTERRUPTION

In case of unexpected interruption due to hitting the BREAK key or a terminal power failure, control retuns to MPE. Press RESET TERMINAL (twice on the 2645) and then press RETURN to display the colon prompt. You type RESUME in response to this prompt. (If the keys you type do not appear on the screen, echo has been turned off; you must press the ESC key and then press the colon (:) in order to restore echo.)

After you type RESUME, MPE issues the message READ PENDING. You then press the REFRESH key (f4) to return to the menu at which you were interrupted. The menu will be cleared to initial or previously entered values. To insure against damage to the file, reenter the information on all menus pertaining to the form you were creating or modifying at the time of the program interruption.

FORMSPEC MENUS

FORMSPEC issues its menus in a predetermined sequence. As each menu is issued, type in the specifications you want and then press ENTER. If you do not want to enter a specification on a particular menu, or you want to skip over one or more menus, you can use the menu sequence control keys (f1, f2, f5, and f6). These keys allow you to select menus relative to the current menu in order to make changes or additions.

You can move forwards or backwards through the sequence of menus to locate a particular menu by pressing the NEXT (f6) or PREV (f5) keys respectively. If you want to skip the field menus associated with each form, you can use the PREV FORM (f1) or NEXT FORM (f2) keys. An alternate method is to request the MAIN menu with the MAIN key (f7), and then select a particular recorded menu.

The sequence control keys can also be used from the MAIN menu. In this case, the menu on the screen when MAIN was requested is treated as the current menu.

Although it is not technically a menu, the screen design associated with each form is treated as the menu following the FORMS menu when the NEXT MENU key is pressed, and as the menu preceding the first FIELD menu when the PREV key is pressed.

You can go to the MAIN menu at any time from any other menu by pressing MAIN (f7). You can return to the menu from which you requested the MAIN menu by pressing RESUME (also f7). If you have just entered FORMSPEC, the MAIN menu is issued automatically. In this case, pressing RESUME results in a display of the GLOBALS menu — the first menu in menu sequence.

From the GLOBALS menu, NEXT FORM (f2) causes the first FORM menu to be displayed; NEXT MENU (f6) causes a SAVE FIELD menu to be displayed.

The EXIT key (f8) provides the only way to return to MPE control from FORMSPEC.

Figure 3-3 illustrates the relation between the menu definitions presented on the FORMSPEC menu screens and the special function keys that control menu sequence.

NOTE

The number of FIELD menus issued depends on the number of fields you defined on the screen for the form. The system automatically issues a FIELD menu for each field named on the screen and displays the default values for the field. When menus for all fields on a form have been issued, the FORM menu is redisplayed. If you have no more forms to design, press the MAIN key for the MAIN menu through which you compile the forms file. Otherwise, continue defining forms by specifying the next form on the FORM menu.

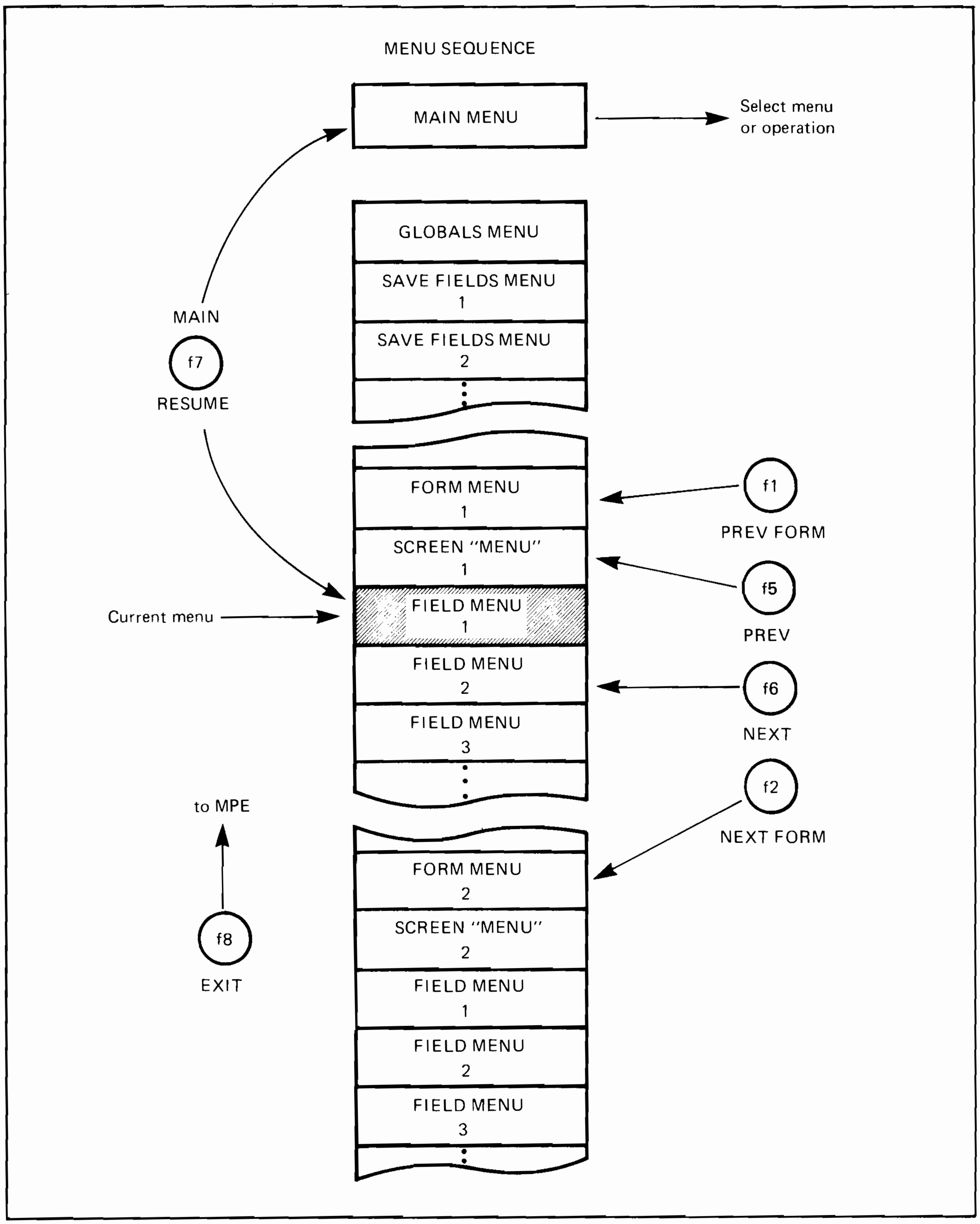


Figure 3-3. Relation Between Menus & Function Keys

GENERAL FORMS DESIGN

Before running FORMSPEC to define your forms file, it is very helpful to roughly sketch each form on paper with the following information:

- Form name
- Next form name in sequence of forms
- Screen layout

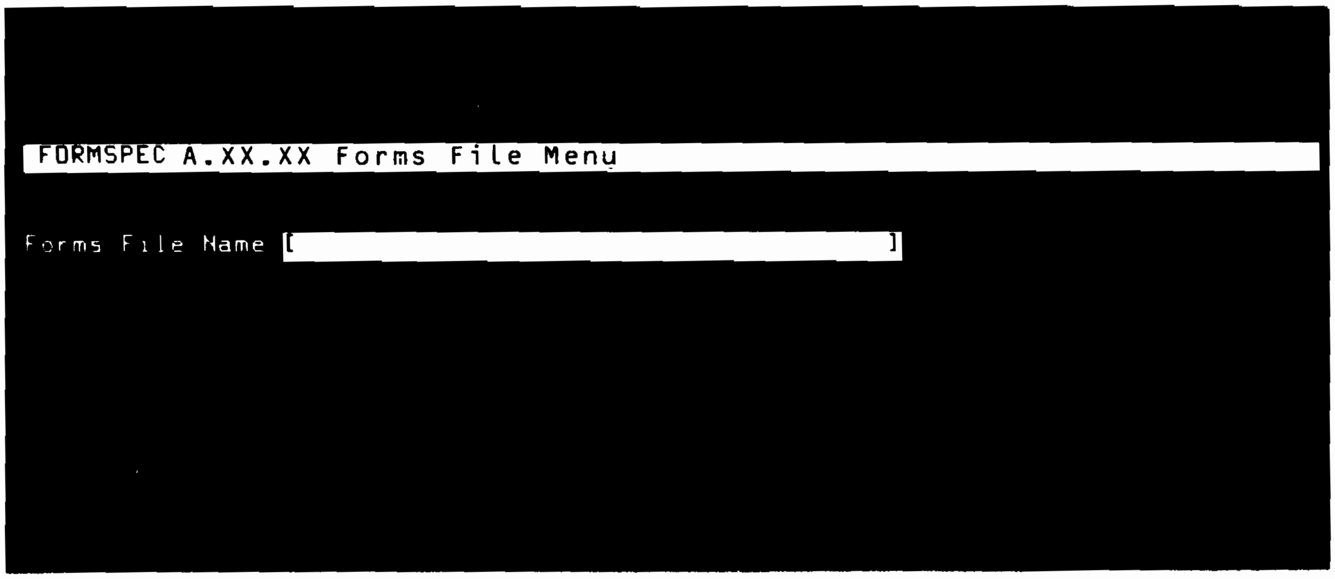
The screen layout should be sufficient to define the characteristics of all fields on the screen. The field name should be noted, and if a field has special (non-default) characteristics, these too may be noted. FORMSPEC determines the length of each field from the actual size of each field on the screen. (Refer to figure 3-19 at the end of this section for an example of how you can sketch out your forms prior to running FORMSPEC.)

Note that there is no need to use a form layout sheet since it is much easier to visualize the placement of fields and headings directly on the screen than it is to lay them out on paper.

Preparing your forms layout in this way allows you to sit down at the terminal and actually specify the complete forms file in a matter of minutes. Note that in many cases, the default values supplied by FORMSPEC can be used, thereby reducing your actual input to a minimum.

FORMS FILE MENU

The first menu displayed whenever you run FORMSPEC is the FORMS FILE menu on which you enter the name of the forms file to be created or modified. This menu is illustrated in figure 3-4.



```
FORMSPEC A.XX.XX Forms File Menu

Forms File Name [                                        ]
```

Figure 3-4. FORMS FILE Menu

Enter the name of the forms file on this menu. It may consist of up to 36 characters.

Element of File Name	Maximum Number of Characters
file name	8
group name	8
account name	8
lockword	8
2 periods, 1 slash	3
terminator (blank or special character)	1
total	36

If your file has a lockword, always enter it along with the file name. Otherwise the system will prompt for the lockword, expecting a response you cannot make until you are again in character mode.

Since the release of version A.01.01 of V/3000, FORMSPEC accepts either KSAM or standard MPE forms files. The only forms files created by FORMSPEC, however, are standard MPE files. The directory and data records are interspersed together in the file. MPE forms files use no extra data segments and are more likely to be recovered after a power failure or system crash.

The first menu issued after you specify your forms file on the FORMS FILE menu is the MAIN menu.

PURGE FORMS FILE

The MPE :PURGE command will delete a non-KSAM forms file. If, however, your forms file is a KSAM file, FORMSPEC does not provide a direct means to purge it. If you want to permanently delete a KSAM forms file created through a prior version of FORMSPEC, you must run the KSAMUTIL utility program as follows:

```
:RUN KSAMUTIL.PUB.SYS
MPE prompt
```

This program will issue a greater than sign (>) as a prompt. In response to this prompt, you enter the PURGE command. Suppose the forms file you want to purge is named ORDFORM, enter the command as follows:

```
>PURGE ORDFORM
ORDFORM.JOAN.DATAMGT & ORDKEY PURGED<--system response
```

Note that you need not specify the key file name associated with your forms file; KSAMUTIL knows this name as it indicates in the response to the PURGE command. KSAMUTIL also notes the account and group in which the forms file resides.

RENAME FORMS FILE

If you want to rename an existing non-KSAM forms file, simply use the MPE :RENAME command. In order to rename a KSAM forms file, you must run program KSAMUTIL. You may rename either the forms file or the key file with a single RENAME command; or you may rename both files with two RENAME commands. Suppose you want to rename the forms file ORDFORM and also its associated key file ORDKEY. To do this, run KSAMUTIL and then enter two RENAME commands, as shown:

```
:RUN KSAMUTIL.PUB.SYS

>RENAME ORDFORM, NEWORD --+
                          |---new names
>RENAME ORDKEY, NEWKEY  --+
```

CONVERT KSAM FORMS FILE

The utility program CONVERT.PUB.SYS will create a MPE forms file from your KSAM forms file:

```
:RUN CONVERT.PUB.SYS
PLEASE ENTER THE NAME OF THE OLD FORMS FILE
ORDFORM
PLEASE ENTER THE NEW FORMS FILE NAME
ORDNEW        <------ system response
DIRECTORY LENGTH=1300
```

The directory is found in the extension of the COMAREA (see page 6-8), and consists of information on where data records in the forms file are located. The length of the directory is 1300 words for standard forms files and 500 words for fast forms files.

EXPAND FORMS FILE

Your forms file may be large enough to require more than the default limit of 1028 records. If you wish to increase the size of your file, you may use either the MPE :FILE or the MPE :BUILD command.

To use the :FILE command to create a form named ORDFORM, issue the file equation:

```
                    :FILE ORDFORM;DISC=2500
                                          ↑
                        desired disc space, in number of records
```

Run FORMSPEC and specify ORDFORM as the forms file. You now have an empty file with room for 2500 records, into which you may add or copy forms using the FORMSPEC main menu options. Note that the FORMSPEC copy command will not copy save fields from other forms files.

To use the :BUILD command to expand your form file, ORDFORM, give the following command:

```
        :BUILD ORDFORM2;DISC=2600;REC=-256,1,F,ASCII;CODE=VFORM
                              ↑                              ↑
                        desired disc space          VFAST, if fast forms
                                                    file
```

Run the file copier to copy the contents of ORDFORM into the larger file:

```
        :RUN FCOPY.PUB.SYS
        >FROM=ORDFORM;TO=ORDFORM2
        ↑
      FCOPY prompt

        EOF FOUND IN FROMFILE AFTER RECORD 1015  }
                                                 }  FCOPY
                                                 }  response
        1016 RECORDS PROCESSED *** 0 ERRORS      }
```

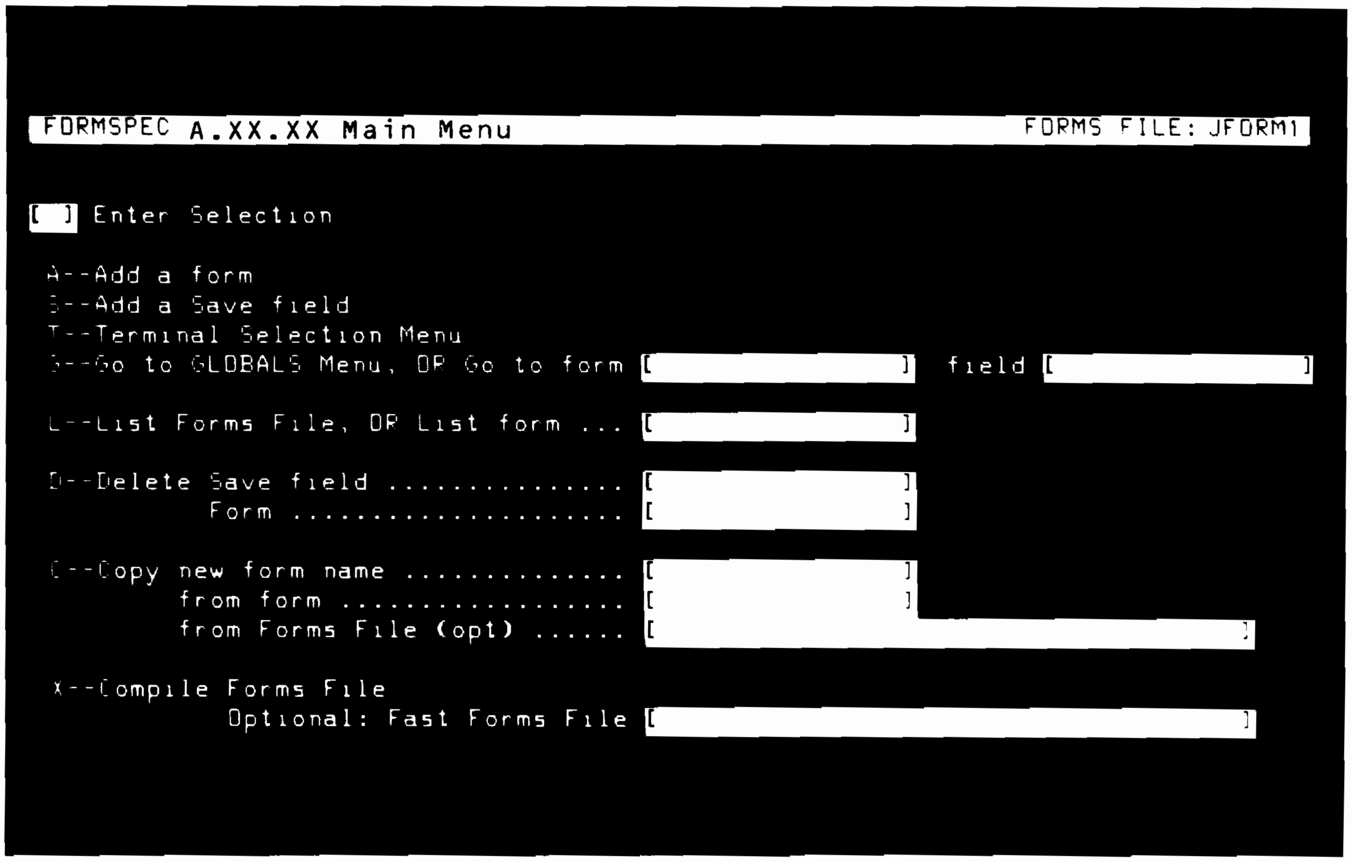
NOTE

The procedures described above do not apply to KSAM files.

MAIN MENU

This menu is the main control menu for all FORMSPEC operations. If the forms file is new, you will usually select "Add a form". FORMSPEC then issues the menus that allow you to define your forms file. If the forms file already exists, you may enter any selection depending on what you want to do. You can add new forms to the file, select a particular menu in order to change it, purge a form, and so forth.

The MAIN menu is illustrated in figure 3-5.



```
FORMSPEC A.XX.XX Main Menu                                  FORMS FILE: JFORM1

[ ] Enter Selection

 A--Add a form
 S--Add a Save field
 T--Terminal Selection Menu
 G--Go to GLOBALS Menu, OR Go to form [              ]  field [              ]

 L--List Forms File, OR List form ... [              ]

 D--Delete Save field ............... [              ]
           Form ..................... [              ]

 C--Copy new form name .............. [              ]
         from form .................. [              ]
         from Forms File (opt) ...... [                                ]

 X--Compile Forms File
           Optional: Fast Forms File [                                ]
```

Figure 3-5. MAIN Menu

ADDING A FORM

When you create a new forms file, you always add a form; if you are modifying an existing file, you may want to add a form. To add a form, simply enter "A" in the MAIN menu selection box. In response, FORMSPEC issues a FORM menu so you can define the form. For details of form definition using this menu, see the description below under "FORM MENU".

New forms are added to the end of the forms file. If a new form is not the first form, you must make sure it is referenced as the "next form" by some other form in the file. The GLOBALS menu (see description later in this section) allows you to specify which form is to be executed first.

After adding forms to the file, the file must be compiled if it is a new file, or recompiled if it is an existing file. Only those forms of the existing file which have been changed are actually recompiled.

ADDING AND DELETING SAVE FIELDS

Save fields are fields that can be used by the entire forms file. They are part of the global information stored at the beginning of a forms file. If you specify "S" in the MAIN menu selection box, a blank SAVE FIELD menu is displayed. You can then enter a save field description. If you want to modify a save field, locate the particular SAVE FIELD menu specification with the PREV or NEXT keys. You can then change the specification and recompile the forms file.

If you want to delete a save field, specify "D" in the MAIN menu selection box and then specify the save field name. The field will be deleted. Note that you should be sure there are no references to the deleted field in the processing specifications of any other field.

Save fields are only used in advanced forms design. As such, the SAVE FIELD menu and a description of how save fields are used is deferred to section IV. Because of the special nature of save fields, the SAVE FIELD menu is not displayed automatically, but is issued only if requested through the MAIN menu, or displayed through pressing the PREV or NEXT keys.

After adding, changing, or deleting save fields, you must recompile the forms file in order to execute it with the changes.

TERMINAL SELECTION MENU

This menu allows you to specify which terminal or set of terminals will be used with the forms file. You obtain this menu by entering a "T" in the selection box.

This menu is not required for the HP 264x terminals.

GLOBAL SPECIFICATIONS

Certain characteristics of the entire forms file are defined as global specifications. These specify the default screen enhancements for fields and errors, the first form to be executed (only if it is not the first form defined), where the message "window" line appears on the screen, and provides for user-defined labels for the function keys.

FORMSPEC supplies default values for these characteristics and, unless you want to change the defaults, you need never be concerned with global specifications. If you do want to change the global characteristics, you enter "G" on the MAIN menu selection box, but do not specify a form or field name. The GLOBALS menu is displayed so you can make the desired changes. (Refer to "GLOBALS MENU" below for detailed information.)

You must compile the forms file in order to include the global specifications when the file is executed.

DISPLAYING AND MODIFYING FORMS

You can change a form description at any time, either as you are initially defining the form or after the forms file has been compiled. In either case, you locate the form description through the MAIN menu selection box or with the function keys that control menu sequence.

CHANGING A FORM

To display the general description of a form, you request the FORM menu by entering "G" in the MAIN menu selection box and specifying the form name, or you can locate the FORMS menu with the PREV FORM (f1) or NEXT FORM (f2) keys. When the FORM menu is displayed, you can then change any specification on the menu. (Refer to "FORM MENU" below for a discussion of the form description.)

CHANGING A FIELD

You can display or change individual field descriptions on the FIELD menu or directly on the screen design menu. The FIELD menu can be located by entering "G" as the menu selection in the MAIN menu and then specifying both the form and field names, or by using the PREV (f5) or NEXT (f6) keys. A screen menu can be located only with the function keys (f5 or f6).

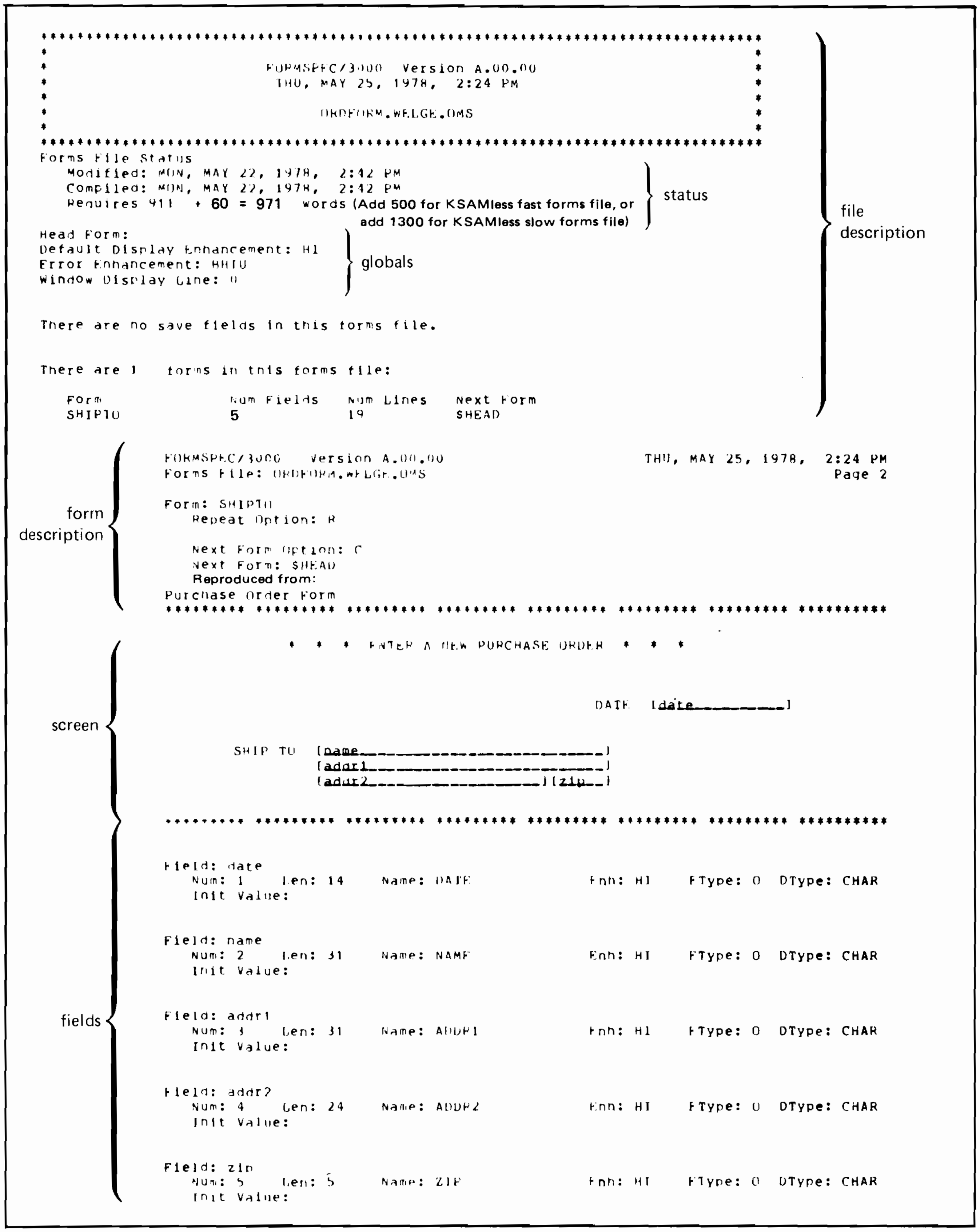
The basic field descriptions are entered and changed on the screen design "menu". Any new fields are added or existing fields deleted by changing the screen design. All non-default field editing specifications, and such specifications as the field name are entered and changed on the FIELD menu. (For details, refer to the "Screen Design" and FIELD Menu" descriptions below.)

After modifying the forms file, you must compile it before it can be executed with the modifications.

LISTING FORMS

You can print a description of any form in your forms file by entering an "L" in the MAIN menu selection box, and then entering the form name. If you want to list all the forms in the file with a description of the file in general, simply enter "L" and do not specify a form name. The listings are printed on the standard list device (usually, the line printer).

Figure 3-6 illustrates the listing for a forms file with only one form. Note that the listing includes the current status of the forms file, including when it was last modified and compiled, and the number of words needed.



```
****************************************************************************
*                                                                          *
*                     FORMSPEC/3000  Version A.00.00                       *
*                     THU, MAY 25, 1978,  2:24 PM                          *
*                                                                          *
*                         ORDFORM.WELGE.DMS                                *
*                                                                          *
****************************************************************************
Forms File Status
   Modified: MON, MAY 22, 1978,  2:12 PM
   Compiled: MON, MAY 22, 1978,  2:12 PM
   Requires 911  + 60 = 971  words (Add 500 for KSAMless fast forms file, or
                                    add 1300 for KSAMless slow forms file)
Head Form:
Default Display Enhancement: HI
Error Enhancement: HBIU
Window Display Line: 0

There are no save fields in this forms file.

There are 1  forms in this forms file:

   Form              Num Fields     Num Lines     Next Form
   SHIP10            5              19            $HEAD

           FORMSPEC/3000   Version A.00.00                       THU, MAY 25, 1978,  2:24 PM
           Forms File: ORDFORM.WELGE.DMS                                             Page 2

           Form: SHIP10
              Repeat Option: R

              Next Form Option: C
              Next Form: $HEAD
              Reproduced from:
           Purchase Order Form
           ********* ********* ********* ********* ********* ********* ********* **********

                           *  *  *  ENTER A NEW PURCHASE ORDER  *  *  *


                                                          DATE  [date__________]


                   SHIP TO  [name___________________________]
                            [addr1__________________________]
                            [addr2___________________][zip__]

           ********* ********* ********* ********* ********* ********* ********* **********


           Field: date
              Num: 1     Len: 14     Name: DATE              Enh: HI     FType: O   DType: CHAR
              Init Value:


           Field: name
              Num: 2     Len: 31     Name: NAME              Enh: HI     FType: O   DType: CHAR
              Init Value:


           Field: addr1
              Num: 3     Len: 31     Name: ADDR1             Enh: HI     FType: O   DType: CHAR
              Init Value:


           Field: addr2
              Num: 4     Len: 24     Name: ADDR2             Enh: HI     FType: O   DType: CHAR
              Init Value:


           Field: zip
              Num: 5     Len: 5      Name: ZIP               Enh: HI     FType: O   DType: CHAR
              Init Value:
```

Figure 3-6. Forms File Listing

DELETING A FORM

An entire form can be deleted from the forms file by entering "D" in the selection box on the MAIN menu, and then specifying the form name. Although not physically removed from the file, the specified form can not be referenced after the forms file is compiled, and will not be displayed when the forms file is executed.

When you delete a form, be sure to modify any other form that references the deleted form. For example, if "FORM2" is given as the next form name for FORM1, and FORM2 is deleted, you must change the next form name specification on the menu for FORM1.

After deleting a form, you must recompile the file or the form will still be in the compiled version of the file.

COPYING FORMS

A very useful feature allows you to copy an entire form from the current forms file or another forms file. The copied form can then be displayed and modified to suit your design. To copy a form, you enter "C" in the selection box. Then specify the name you want to give the new form, the name of the existing form to be copied, and, if the form to be copied is not in the current file, the name of the file containing this form. Note that the copy feature will not copy save fields from one file to another; you must recreate the necessary save fields if you copy a form from a forms file other than the current file.

The form being copied must be given a name unique to the current forms file.

A new form is created by FORMSPEC that is an exact replica of the existing form. If you want to modify the copied form in any way, simply display the form menus and make the changes. Note that new fields are added or existing fields deleted by making these changes directly to the screen design. You can use the MAIN menu to go to a particular menu on the copied form, or you can use the NEXT MENU key (f6) to step through the menus in the copied form.

COMPILING THE FORMS FILE

You can compile the current forms file by entering "X" in the selection box on the MAIN menu.

If the file has already been compiled, the forms that have been modified are recompiled and the new compiled version replaces the previous version. There is never more than one compiled version of a forms file at a time. You may however modify the source version without affecting the compiled version. For example, if the compiled version is being used for data entry, it is not altered in any way as you modify the source version. When the source is completely modified, you can compile it and use the newly compiled version for data entry. In that case, the previous version is lost.

FAST FORMS FILE

When your forms design is complete, you can compile the forms file to a "fast forms file". A fast forms file is similar to the forms file except that it is created with the smallest record size that can hold the longest form in the file. Because record size is minimal, such a forms file can improve performance at run time. A fast forms file can only be executed; it cannot be modified. You may however always modify the source forms file and then recompile it to a fast forms file.

To compile to a fast forms file, enter "X" in the selection box exactly as you would for any compilation (or recompilation), but also supply the fast forms file name.

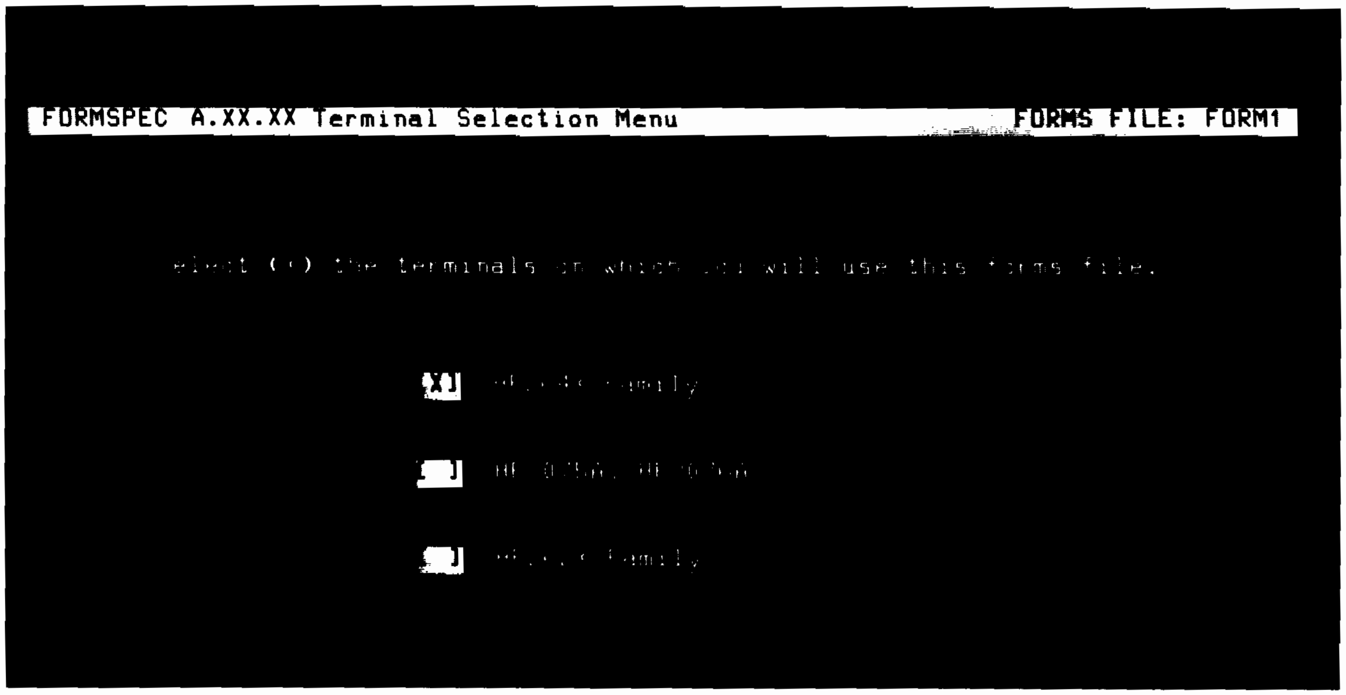
RESUMING DESIGN FROM MAIN MENU

If you requested the MAIN menu by pressing the MAIN key during forms design, you can return to the menu you were in by pressing the RESUME key (f7). Note that this is the same key as the MAIN key, but it acts differently when you are in the MAIN menu than it does in any other menu.

You can use PREV key (f5) to return to the FORM FILE menu where you can specify a different forms file to be modified. The NEXT key (f6) can be used to specify new GLOBAL specifications. The previous form or the next form are the forms preceding or following the form you were designing when MAIN was requested. The previous or next form is requested by pressing PREV FORM (f1) or NEXT FORM (f2) respectively. (Refer to figure 3-3 for a summary of how the forms sequence keys are used.)

TERMINAL SELECTION MENU

The TERMINAL SELECTION menu allows you to specify which terminals will be used with the forms file. You may select one family or any combination of families including all three families. The HP 264x family is the terminal default value. When the selection is defaulted, the forms file will run on the 264x family and the 262x family, but the following features of the 262x family will not be supported: local edits and security display enhancement.



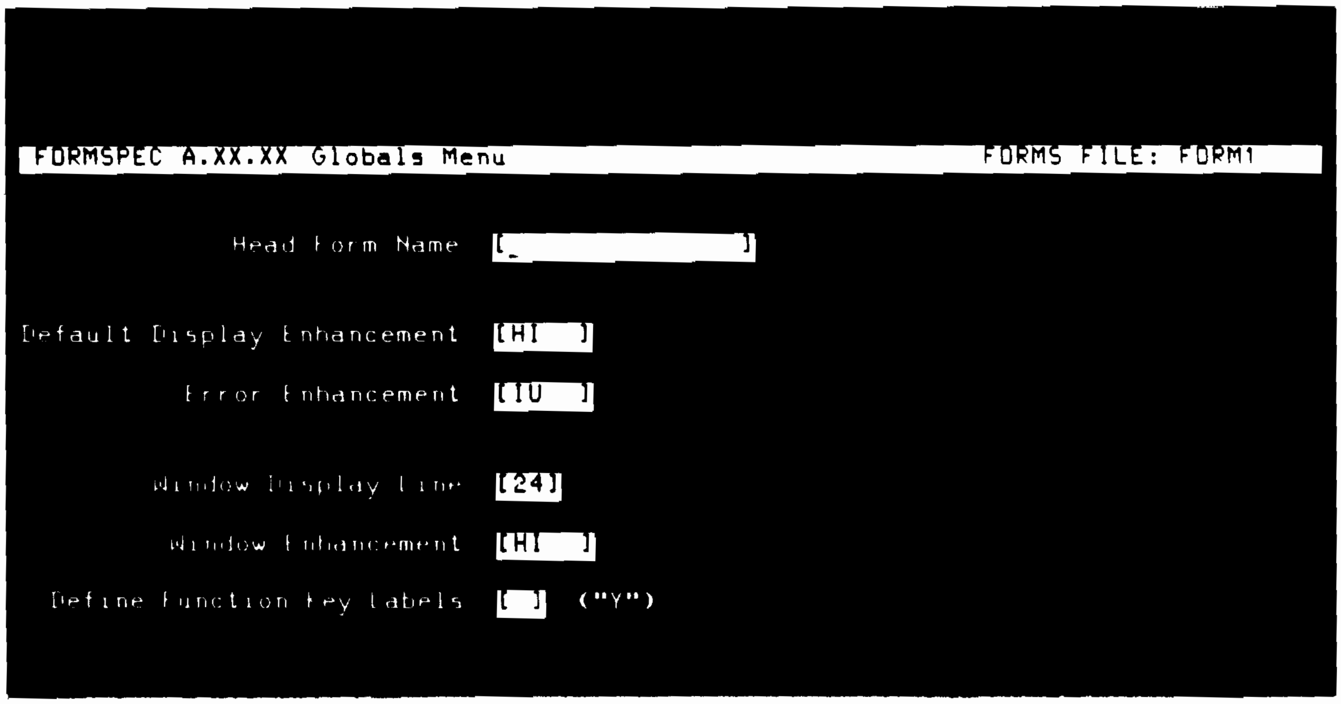
```
FORMSPEC A.XX.XX Terminal Selection Menu                    FORMS FILE: FORM1

      Select (X) the terminals on which you will use this forms file.

                          [X]  HP 264x family

                          [ ]  HP 3075A, HP 3076A

                          [ ]  HP 262x family
```

Figure 3-7. TERMINAL SELECTION Menu

GLOBALS MENU

The first part of any forms file contains "global" specifications. These specifications apply to the entire forms file rather than to individual forms or to fields within the forms. Of these global specifications, only the information entered on the GLOBALS menu is described in this section. Save fields, entered on SAVE FIELD menus, are global in scope. But, since they are generally used in conjunction with advanced field processing, they are discussed in the next section.

In most cases, you can accept the default values provided for the global specifications. These include how fields, errors, and the message window are enhanced, where the message window is placed, and the first form to be executed. If, however, you want to change these specifications, you enter "G" in the MAIN menu selection box, and do not specify a form name or a field name. The GLOBALS menu is then displayed so you can change the default global specifications. If you indicated the HP 3075/6 terminals on the TERMINAL SELECTION menu, the following message will appear at the bottom of the GLOBALS menu: "Press NEXT to select HP 3075/6 device specifications." (See HP 3075/6 DEVICE SPECIFICATIONS menu below for more detail.) The GLOBALS menu is illustrated in figure 3-8.



```
FORMSPEC A.XX.XX  Globals Menu                          FORMS FILE: FORM1

                    Head Form Name  [_            ]

     Default Display Enhancement  [HI  ]

               Error Enhancement  [IU  ]

             Window Display Line  [24]

              Window Enhancement  [HI  ]

       Define Function Key Labels  [ ]  ("Y")
```

Figure 3-8. GLOBALS Menu

HEAD FORM NAME

Enter the name of the first form you want displayed when the forms file is executed. If you leave this field blank, the first form you define in the forms file will be the first form displayed.

When the forms file is compiled, this name must identify an existing form. If you enter the name using lower case letters, FORMSPEC upshifts the letters to upper case and then looks for a matching form name.

Default = first form in forms file.

DEFAULT DISPLAY ENHANCEMENT

When defining a form, you can specify individual display enhancements for any field on the FIELD menu. If you do not specify such enhancements on the FIELD menu, FORMSPEC assigns default field enhancements. If you do not want to use these default display enhancements, you can specify your own default enhancements here. Enter one or more of the following enhancement codes in any combination:

Enhancement	Code
Inverse video	I
Half bright	H
Underline	U
Blink	B
None	NONE

The enhancement codes can be entered in any combination, in any order. For example, for blinking, half bright, underlined, you can specify "BHU", "HUB", "UBH", and so forth. If you want to remove all display enhancements, enter "NONE".

Default = Half bright, Inverse video (HI).

The HP 3075/6 terminals do not support display enhancements.

ERROR ENHANCEMENT

When an operator entering data on a form makes an error, the field with the error is highlighted on the screen by special display enhancements. You can change the default error enhancement by entering one or more of the display enhancement codes (B, H, I, U) in any combination, or you can enter "NONE" if you do not want fields with errors to be enhanced at all. For example, if you want fields with errors to be displayed in half bright, underlined, and to blink, enter "BHU". If you want to retain the default error enhancement, leave the box blank.

Default = Inverse video, full bright, Underline (IU).

WINDOW DISPLAY LINE

Specify the line on the screen to be reserved for error and status messages. Screen lines are numbered from 1 (top) through 24 (bottom). You may enter 0 (zero) if you do not want a window. Note that in this case, no error or status messages are displayed on the screen during execution. Any fields in which errors are detected will still be enhanced when there is no window line, unless you specify "NONE" for error enhancements.

Default = Bottom line (24)

WINDOW ENHANCEMENT

The window line is normally enhanced with inverse video, half bright. If you want a different enhancement for this line, you must specify it in this box. You may enter any of the standard enhancement codes (I, H, U, B) in any combination, or you can specify no enhancements by entering "NONE".

Default = Inverse video, Half bright (HI).

DEFINE FUNCTION KEY LABELS

To change function key labels, enter a "Y" to obtain the GLOBAL FUNCTION KEY LABELS menu. If you want to retain the default GLOBAL labels, leave this box blank.

When you press ENTER, FORMSPEC checks and records the global values (default or specified) and then issues the next menu. The next menu is the SAVE FIELD menu. To get to the first form menu, press the NEXT FORM key (f2).

GLOBAL FUNCTION KEY LABELS MENU

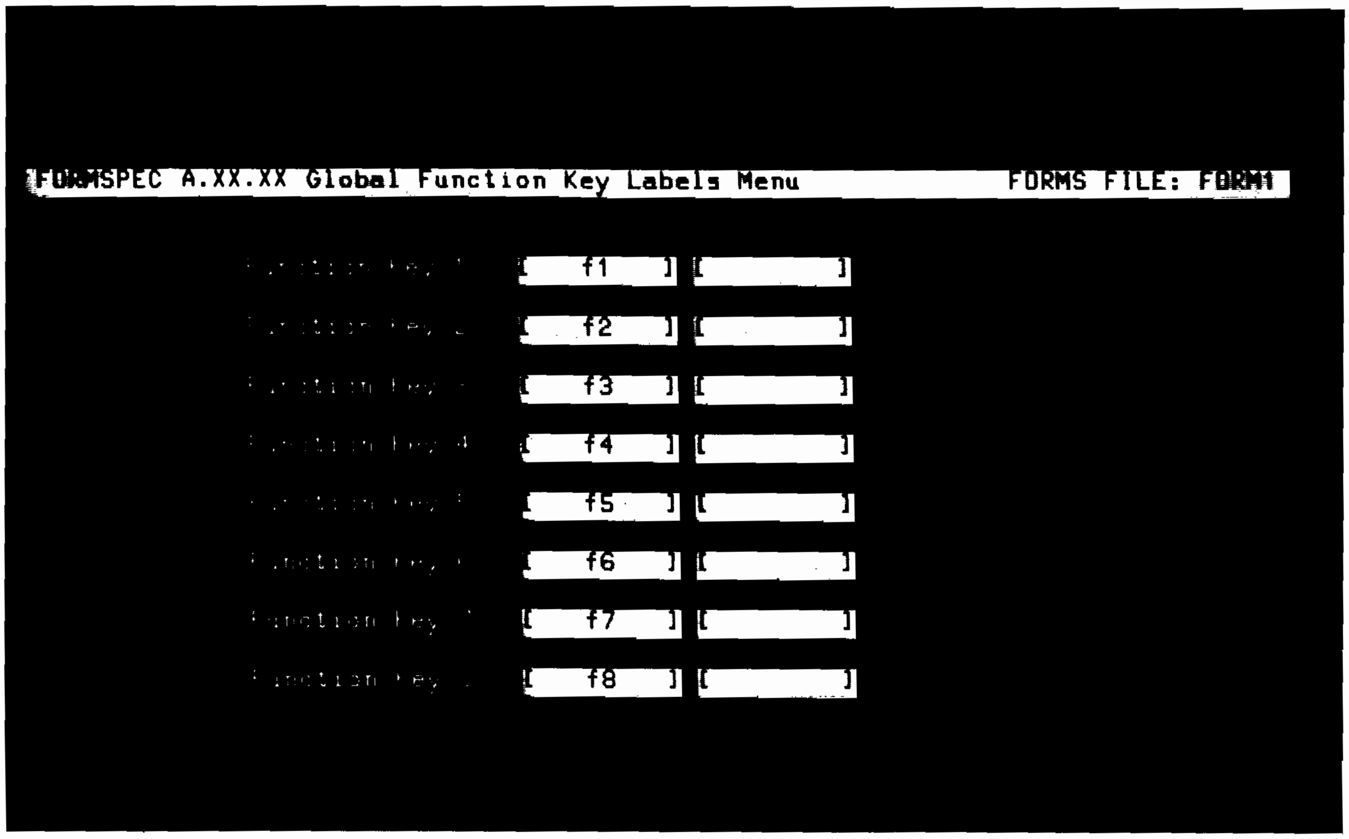
This menu is used to specify new function key labels which will appear when ENTRY.PUB.SYS is executed. Labels you specify will replace original labels provided by V/3000.

Each label consists of two lines of eight characters each. To specify a label enter the first line in the first field and the second line of the label in the second field in the menu. For example, to change the label for Key 1 to read ORDER NUMBER, enter:

Function Key 1 [ORDER ΔΔΔ]   [NUMBER ΔΔ]

If the labels are to be GLOBAL, they will be displayed for all forms that do not have LOCAL labels defined.

In the case of a frozen form with another form appended to it, the labels displayed will be those for the appended form. If the appended form has no LOCAL labels, the file's GLOBAL labels will be used.



```
FORMSPEC A.XX.XX Global Function Key Labels Menu              FORMS FILE: FORM1

              Function Key 1     [   f1   ] [        ]

              Function Key 2     [   f2   ] [        ]

              Function Key 3     [   f3   ] [        ]

              Function Key 4     [   f4   ] [        ]

              Function Key 5     [   f5   ] [        ]

              Function Key 6     [   f6   ] [        ]

              Function Key 7     [   f7   ] [        ]

              Function Key 8     [   f8   ] [        ]
```

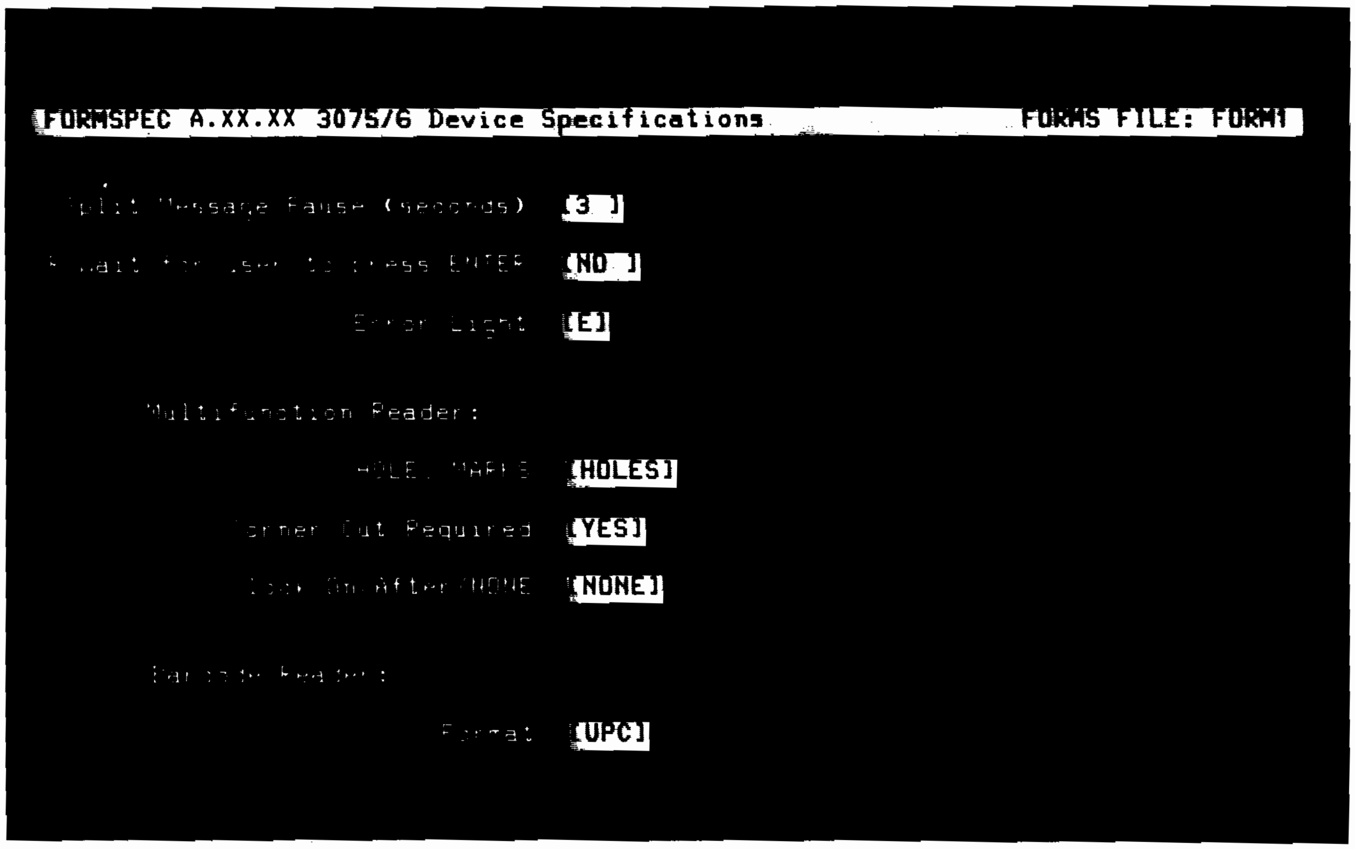
Figure 3-9. GLOBAL FUNCTION KEY LABELS Menu

HP 3075/3076 DEVICE SPECIFICATION MENU

If you select the HP 3075/6 terminals on the Terminal Selection menu, the Device Specification menu will appear after the GLOBAL menu.

On this menu you can specify how you want messages longer than 24 characters to be presented on single-line display. In addition, you can specify which light is to be lit if an error is detected, and the device configuration for the Multifunction and Bar code readers if they were specified as valid input devices.

If you want to change these specifications, use the TAB key to access the appropriate box.



```
FORMSPEC A.XX.XX 3075/6 Device Specifications                    FORMS FILE: FORM1

    Split Message Pause (seconds)  [3 ]

  Or wait for user to press ENTER  [NO ]

                      Error Light  [E]

       Multifunction Reader:

                      HOLES/MARKS  [HOLES]

              Corner Cut Required  [YES]

               Lock On After NONE  [NONE]

       Barcode Reader:

                           Format  [UPC]
```

Figure 3-10. HP 3075/6 DEVICE SPECIFICATIONS Menu

SPLIT MESSAGE PAUSE (Seconds)

(Applies to single-line display only)

You may specify the number of seconds between screen presentations for messages longer than twenty-four characters. Enter a value between 1 and 99 in this box and a NO in the "Wait for ENTER" box. User and V/3000's messages will be broken so as not to display part of a word.

Default: 3 (seconds)

WAIT FOR USER TO PRESS ENTER

(Applies to single-line display only)

You may also specify that screen presentations for messages longer than twenty-four characters be displayed after the ENTER key is pressed. Enter a YES in this box and a zero (0) in the "Split Message Pause" box.

Default: NO

ERROR LIGHT

You may specify which light is to be lit if an error is detected on the form by the calling procedure or V/3000. This light will remain lit during the re-presentation of the form. You may specify any one of the following: @, A through P.

Default: E

MULTIFUNCTION READER

If you select the Multifunction Reader as an input device, you can specify how it should be configured to accept valid card and badge input. The following options are available:

Holes/Marks	— card or badge will have holes or will be mark sensed. Default: HOLES
Corner Cut Required	— card or badge must be presented with the corner in accordance with the diagram on the terminal. Default: YES
Clock After/NONE	— Card or badge will have clock marks after each column of data, or no clock marks. Valid input is CAD (clock after data) or NONE (no clocking). Default: NONE

Invalid combinations of the above are Holes with Clock on data and Marks with NONE.

BAR CODE READER

You can specify which bar code format will be read. Choose one of the following:

I25	— Industrial 2 out of 5
MAT	— INTERMEC 2 out of 5 MATRIX code
I39	— INTERMEC code 39

DEFAULT: I39

When you specify the bar code reader as an input device, V/3000 will configure the terminal to require check digits.

Example of how typical screen design will look on the HP 3075/6:

```
   This is the first line of the form.

       This is line 3 of the form and it extends all the way to column 80..........

                   This line starts in column 17

                          This line starts in column 25

                                  This line starts in column 33
```

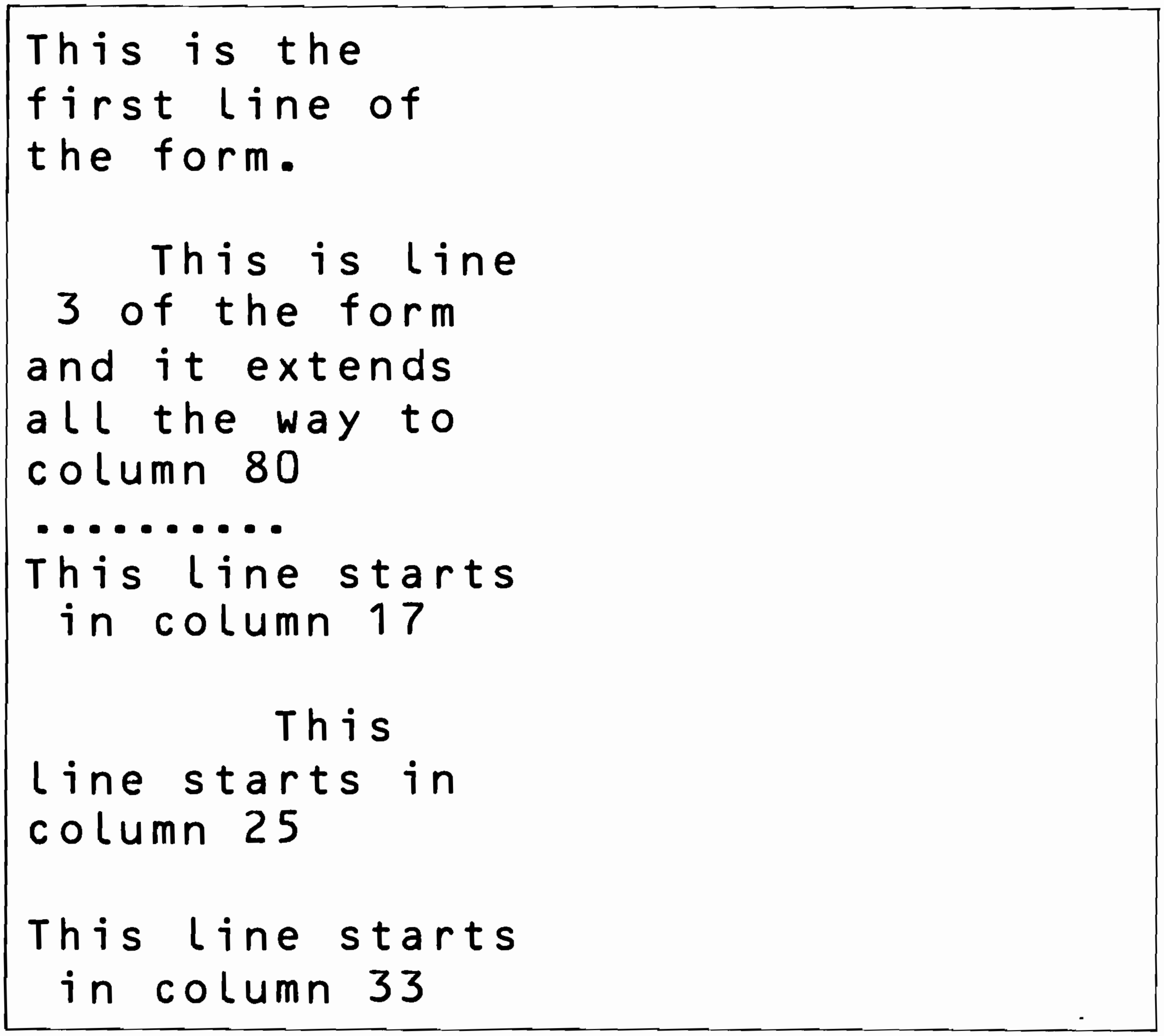
Figure 3-11. 26XY Screen Design

```
This is the first line of the
form.

    This is line 3 of the form
and it extends all the way to
column 80..........

                This line starts
 in column 17

                       This
line starts in column 25

This line starts in column 33
```

Figure 3-12. Standard Character Set Mini-CRT Screen

```
This is the first line
of the form.
    This is line 3 of
the form and it extends
all the way to column 80
 ..........
                This
line starts in column 17
This line starts in
column 25
        This line starts
in column 33
```

Figure 3-13. 1 Line Alpha Display Screen



```
This is the
first line of
the form.

    This is line
 3 of the form
and it extends
all the way to
column 80
..........
This line starts
 in column 17

        This
line starts in
column 25

This line starts
 in column 33
```

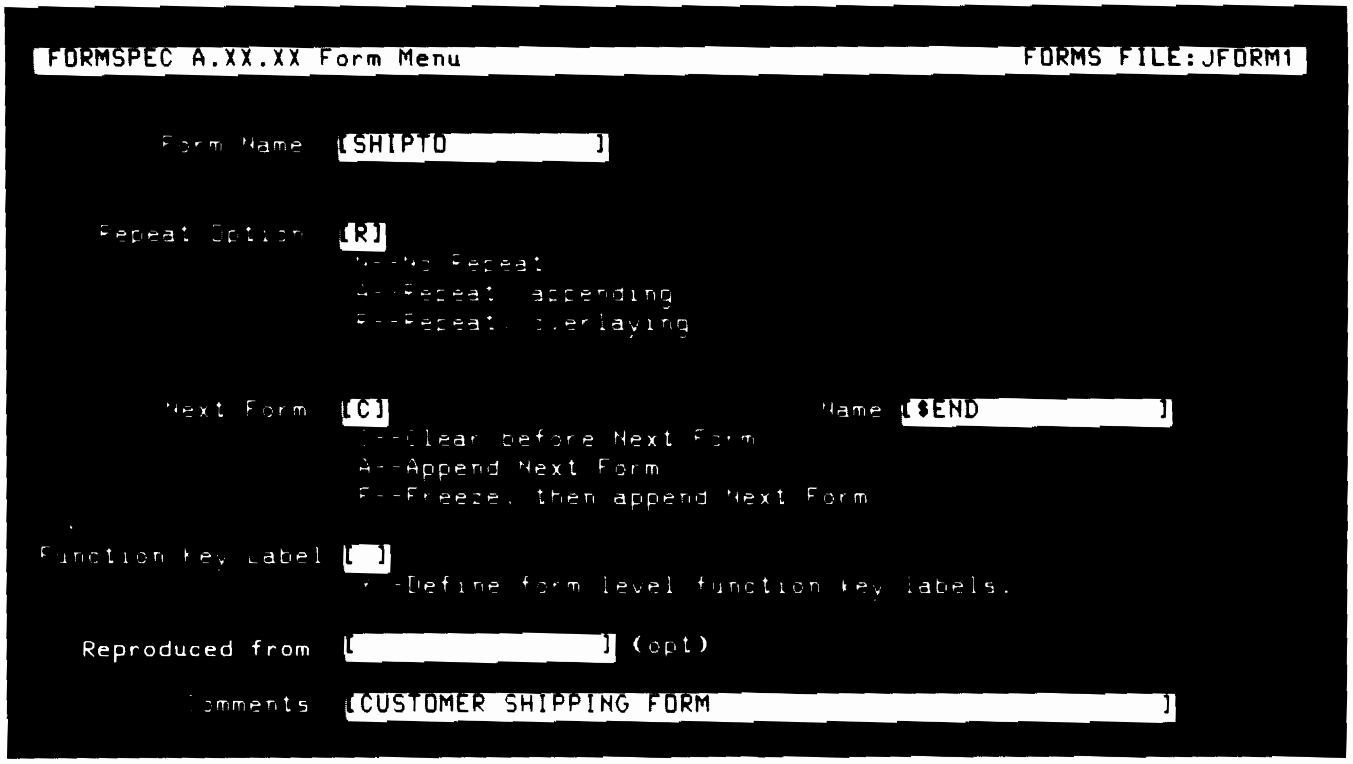
Figure 3-14. Large Character Mini-CRT Screen

FORM MENU

The name and characteristics of each form are specified on the FORM menu. As soon as the FORM menu specifications are entered, FORMSPEC issues a blank screen on which you design the layout of the form. This screen design phase establishes the field names and their characteristics. Based on the screen design, FORMSPEC then issues a FIELD menu for each field established on the screen.

The FORM menu is issued once for each form you define. When you have no more forms to define, press the MAIN key (f7) to display the MAIN menu in order to compile the forms file.

The FORM menu is illustrated in figure 3-15.



```
 FORMSPEC A.XX.XX Form Menu                                  FORMS FILE:JFORM1

        Form Name   [SHIPTO         ]


    Repeat Option   [R]
                     N--No Repeat
                     A--Repeat appending
                     R--Repeat overlaying


        Next Form   [C]                              Name [$END            ]
                     C--Clear before Next Form
                     A--Append Next Form
                     F--Freeze, then append Next Form

Function Key Label  [ ]
                     Y--Define form level function key labels.

  Reproduced from   [               ] (opt)

         Comments   [CUSTOMER SHIPPING FORM                                ]
```

Figure 3-15. FORM Menu

You must specify at least a form name on the FORM menu. Default values are provided for the remaining specifications. If these defaults are not what you want, you can overwrite them with other values. If satisfied with the defaults, press ENTER to go directly to screen design.

FORM NAME

Enter a name to identify the form. The name can be from 1 to 15 characters, either letters of the alphabet (A-Z), digits (0-9), or the underline (_). The first character must be alphabetic. The name can be entered using upper or lower case letters, but all lower case letters are shifted to upper case by FORMSPEC. Thus, if you enter the name "Form2", FORMSPEC changes it to "FORM2".

The form name must be unique within the forms file, and it cannot be one of the reserved words listed in table 3-2. Note that the form name can be changed if you modify the form description. If you change a form name, you should make sure that references to this form as "Next Form" in another form description are also changed.

Table 3-2. FORMSPEC Reserved Word List

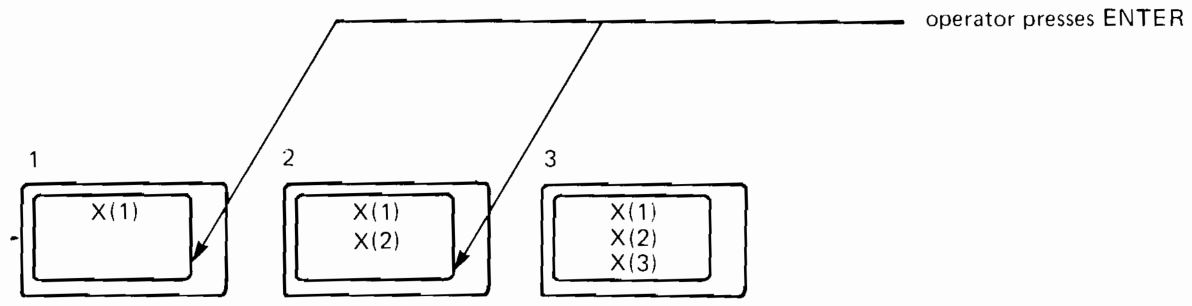
ALL	FILL	LT	STOP
APPEND	FINISH	MAGSTRIPE	STRIP
BARCODE	FREEZE	MARKS	THEN
CAD	GE	MAT	TO
CDIGIT	GT	MATCH	TRAILING
CENTER	HOLES	MFR	TYPEV
CFORM	I25	MINLEN	UPSHIFT
CHANGE	I39	NE	$EMPTY
CLEAR	IF	NFORM	$END
COD	IN	NIN	$HEAD
CONFIG	INIT	NOCUT	$LENGTH
CUT	JUSTIFY	NONE	$REFRESH
DEVICE	KEYBOARD	NOREPEAT	$RETURN
DISPLAY	LARGECHAR	OF	$STATE
ELSE	LE	PRINTER	$TODAY
EQ	LEADING	REPEAT	
FAIL	LEFT	RIGHT	
FIELD	LIGHT	SET	
	LOCALEDITS	STDCHAR	

REPEAT OPTION

When the forms in the forms file are displayed, a form may be repeated and appended to itself (A); it may be repeated overlaying the previous display of the same form (R); or it can be a non-repeating form that is displayed on the screen once (N).

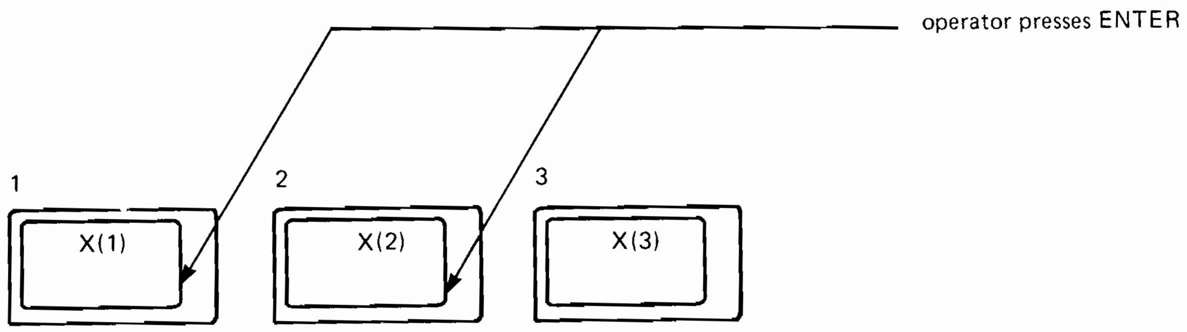
Either repeat option (A or R) causes the form to be repeated until the application program changes the repeat option, or, in ENTRY, the operator presses the NEXT key (f6) to request the next different form.

To illustrate, assume that form "X" is a repeat/append form (A). This form is displayed, the operator types in data, and presses ENTER. The form with the data remains on the screen, and the same form with no data (except initial values) is displayed immediately below the form with data. The next time ENTER is pressed, the form is displayed a third time immediately below the second form. This continues until the repeat option is changed.



Appended forms are particularly useful when the form is a single line that has an indeterminate number of iterations. An order entry blank, for instance, could be designed as a repeat/append form.

A form that repeats without the append option (R) is cleared each time the operator presses ENTER to enter data. For example, assume form X is a repeating form that overlays itself:



Note that this type of repeating form overlays itself even if there are other forms on the screen.

Default = No repeat (N).

NEXT FORM

You must enter the name of the next form to be displayed after the current form. You may also specify whether the next form is to be appended to the current form, "A"; and, if appended, whether the current form is to remain on the screen when the screen fills up, "F". If the screen is to be cleared before the next form is displayed, leave the default value, "C".

NEXT FORM NAME

The next form name is entered as a standard form name or you can enter one of the following system-defined values:

$RETURN	— Display the previous (but different) form.
$HEAD	— Display the first form (either the first in the file or as defined on the GLOBALS menu).
$END	— Terminate forms display with this form.
$REFRESH	— Display current form cleared to initial values as the next form.

Note that each of these values has meaning only when the forms file is executed. Any value entered for Next Form Name can be changed through the CHANGE NFORM specification described in section IV.

Default = $HEAD

FREEZE/APPEND OPTION

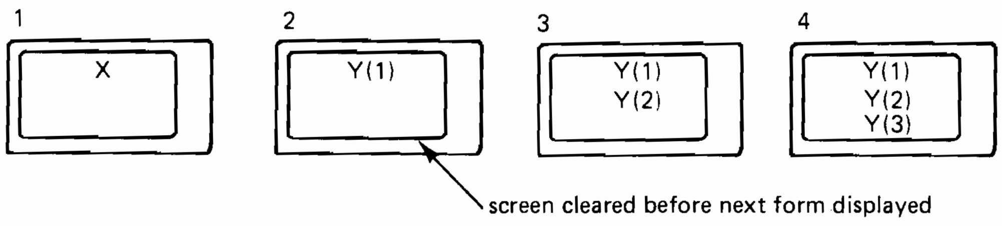
This option allows you to specify how the next form will interact with the current form. The next form can be appended to the current form, and if so, the current form can be frozen on the screen when subsequent forms are displayed. Note that this option can also be changed by the CHANGE NFORM field processing specification IV.

Default = Clear screen before displaying next form (C).

Examples. The following examples illustrate how the Freeze/Append option interacts with the current form. Assume a current form "X" and a next form "Y". The next form (Y) is defined on its own FORM menu as a repeating form appended to itself (repeat option = A for Append).

1. Current Form X — Repeat Option = N
   Next Form Option = C

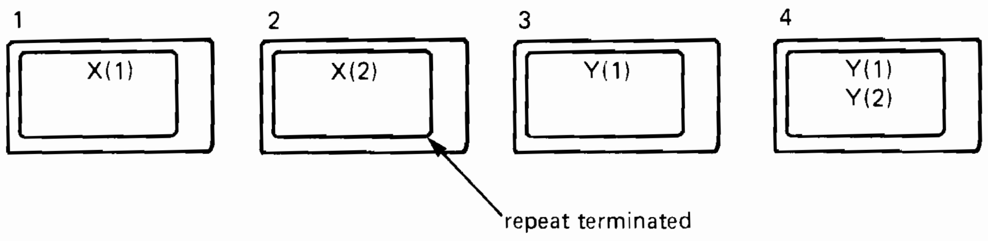
   Next Form Y — Repeat Option = A



Form X is displayed, then after ENTER, the screen is cleared and form Y is displayed. After the next ENTER, Y is repeated below itself until the operator presses NEXT (f6) or the application changes the repeat option.

2. Current Form X — Repeat Option = R
   Next Form Option = C

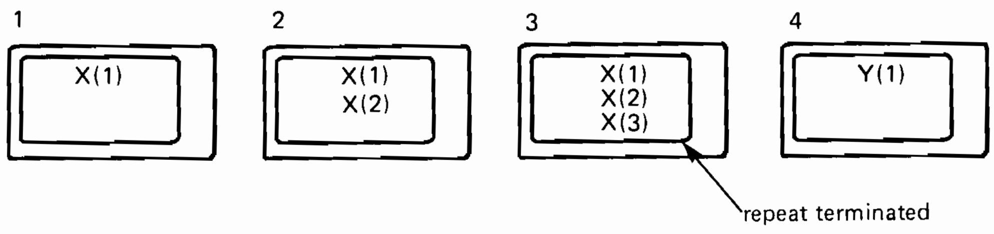
   Next Form Y — Repeat Option = A



X is displayed until the repeat option is terminated, then Y is displayed and appended to itself until the repeat is terminated again.

3. Current Form X — Repeat Option = A
   Next Form Option = C

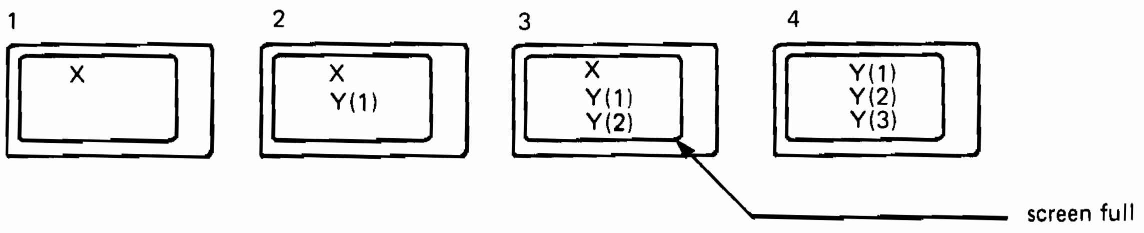
   Next Form Y — Repeat Option = A



X is displayed and then appended to itself until the repeat option is terminated. Then the screen is cleared and Y is displayed.

4. Current Form X — Repeat Option = N
   Next Form Option = A

   Next Form Y — Repeat Option = A

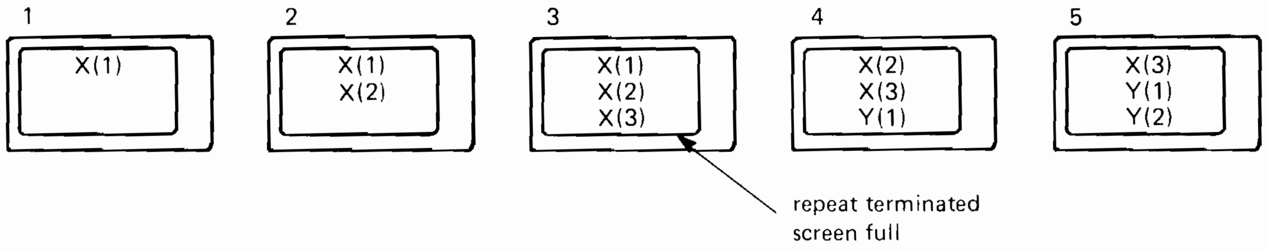


X remains on the screen while Y forms are appended to it until no room is left on the screen. At that point, form X (or part of it) is rolled off the screen and deleted to make room for the third iteration of Y.

If the next form Y(1) filled the screen below X, Y(1) would be rolled off and deleted before Y(2) is displayed.

5. Current Form X — Repeat Option = A
Next Form Option = A

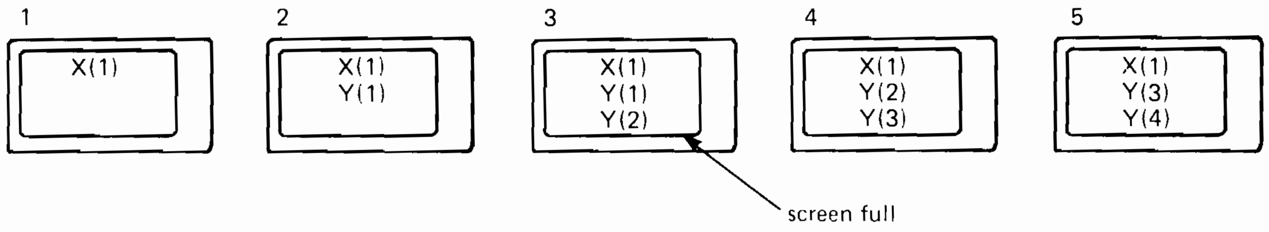
Next Form Y — Repeat Option = A



Form X is repeated and appended until the repeat is terminated. When the first Y form is displayed, there is no more room for the first X and it is rolled off the top of the screen and deleted. As new Y forms are appended, the X forms continue to be rolled off and deleted.

6. Current Form X — Repeat Option = N
Next Form Option = F

Next Form Y Repeat Option = A

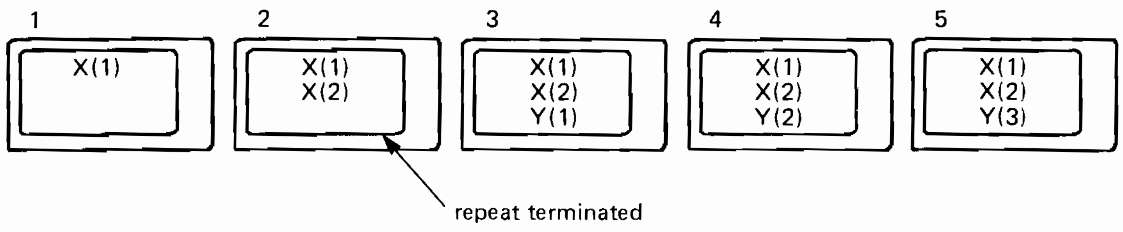


Form X remains frozen on the screen as form Y is appended to it. When the screen is filled, X remains frozen on the screen while the oldest Y form, Y(1) is rolled off to make room for Y(3). Note that if Y(1) fills the screen below X, it will be rolled off the screen and deleted when Y(2) is displayed.

Note that the F specification determines what happens when the screen is full. The A specification determines what happens when the next form is displayed. If an appended form does not fit on the screen with a frozen form, the freeze specification is cleared and the frozen form is rolled off the screen and deleted until the entire next appended form fits.

7. Current Form X — Repeat Option = A
   Next Form Option = F

   Next Form Y — Repeat Option = A



Form X is appended to itself. When the repeat is terminated, the first Y form is appended to the last X. When the next Y form is displayed, the first Y is rolled off the screen and deleted leaving the remaining X forms on the screen.

Note that if the frozen X forms should fill the screen, the first X is rolled off and deleted to make room for the latest X, and when the first Y form is appended, the X forms are no longer frozen.

LOCAL FORM FUNCTION KEY LABELS

To change function key labels enter a "Y" in the box to obtain the FORM FUNCTION KEY LABELS menu. If you want to retain the default labels on a local level, leave this box blank.

REPRODUCED FROM

This optional field is used in the creation of forms families, a feature discussed at the end of Section 4.

COMMENTS

Enter any comment that helps to document the form. The comment is displayed with the source file, but is not included in the displayed form at execution time. A comment can be up to 50 characters long.

When you press ENTER to enter your form definition, the screen is cleared so that you can lay out your screen design.

SCREEN DESIGN

Each form is associated with a picture that is displayed when the form is requested during execution. This picture is called the "screen". You design the screen on a blank display screen using all the terminal capabilities associated with block mode.

Once you have entered your headings and other text, and delimited the data fields, the screen is basically designed. Each field has default characteristics assigned to it by FORMSPEC, so that unless you want to specify special characteristics for the field, screen design completes the form design for the form.

Each screen consists of two kinds of information:

- Data — Fields in which data is entered by or displayed to the operator.
- Text — The headings and other displayed information that appears on the screen but is never altered during execution.

You must distinguish between these two kinds of information within the screen definition.

Note that you can use the terminal escape sequences or function keys to enhance the text. If you alter the meaning of the function keys while designing the screen, you must restore them to their default values before continuing forms design on the next menu.

Once your screen is designed, you can use the terminal SET TAB key to mark the beginning of each field. This is a help when you design subsequent forms with fields that need to be lined up between forms.

FIELD DELIMITERS

The data fields are delimited either by brackets ([ ]) or by the ESC key combined with brackets, or by a combination of these. The ESC key prevents the brackets from being displayed on the screen. In either case, the delimiter used to mark the beginning of one field can be used to mark the end of the previous field.

PRINTED DELIMITERS

If you delimit the data fields with brackets, they are not included in the length of the field, but they do take on the display enhancements assigned to the field. Brackets make it easy to see where a field begins and ends during definition, but they take up space on the screen. Thus, if you must concatenate two data fields, you should not use brackets to delimit the fields.

NON-PRINTING DELIMITERS

You can also delimit fields by pressing the ESC key with either the left bracket ([) or right bracket (]) keys. The advantage of these delimiters is that they take up no space on the screen and thus can be used to delimit contiguous fields. The disadvantage of using these keys is that they are not displayed and therefore do not show up during screen design.

NOTE

The maximum number of fields allowed on any one form is 128.

FIELD IDENTIFIER

Each field is identified by a "tag" entered in the field itself. Shifted to upper case, this tag is the name by which a field is referenced, unless you choose to specify a different name for the field in the FIELD menu.

FIELD TAG

Regardless of the technique used to delimit the fields, each field must be identified by a "field tag". This tag consists of letters of the alphabet (upper or lower case), digits, or the the underline (_). The first character must be alphabetic. Since the field tag must be specified within the field delimiters, it is limited to a length less than or equal to the number of characters in the field. Thus, if you have a one-character field, its tag may not have more than one character. You may rename any tag on the FIELD menu. Thus, you can give a one-character tag a longer name when the menu for the field is displayed.

Note that for the field tag, upper case letters differ from lower case letters. Thus, "f1" and "F1" are two different tags. All other names used by FORMSPEC make no distinction between upper and lower case letters, but shift all letters to upper case. Since each tag is upshifted when used as the default field name, tags that differ on the screen may result in identical field names. When this occurs, you must rename one of the identical field names on a FIELD menu.

Each field tag must be unique before it is upshifted, thus, a form may have up to 52 1-character fields.

You can fill up the field with dots (periods). This gives you a visual representation of field size while you are designing the field. Using dots in the field is particularly useful when the field is delimited by ESC/[and ESC/] since these delimiters are not displayed.

MIXING PRINTING AND NON-PRINTING DELIMITERS

To use one printing and one non-printing delimiter to fix the boundaries of a field, use the delimiters as follows:

If the printing delimiter is to come first, use

```
[ ESC[fieldtag. . .ESC]
```

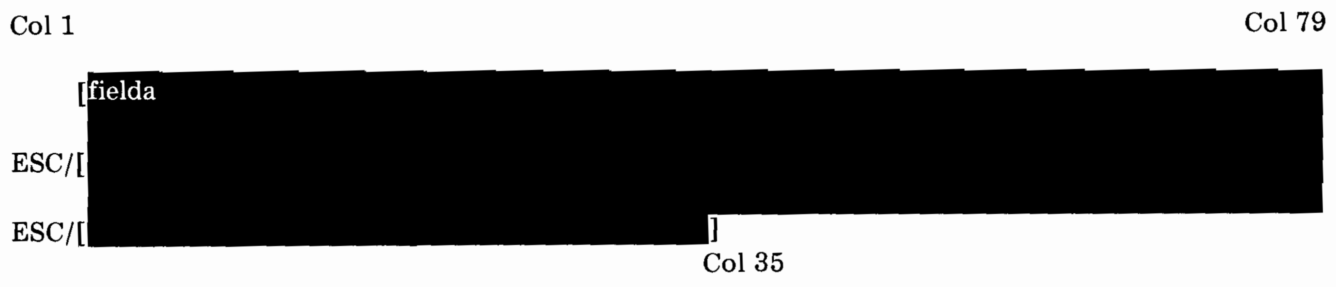
If the printing delimiter is to come last, use

```
ESC[fieldtag. . .ESC] ]
```

FIELD LENGTH

The maximum length of each field is determined by the number of characters you give the field when you design it. If you change the field length on the screen, the new length is automatically reflected in the FIELD menus. Note that if the form is to be appended, do not use column 80 for either text or a field.

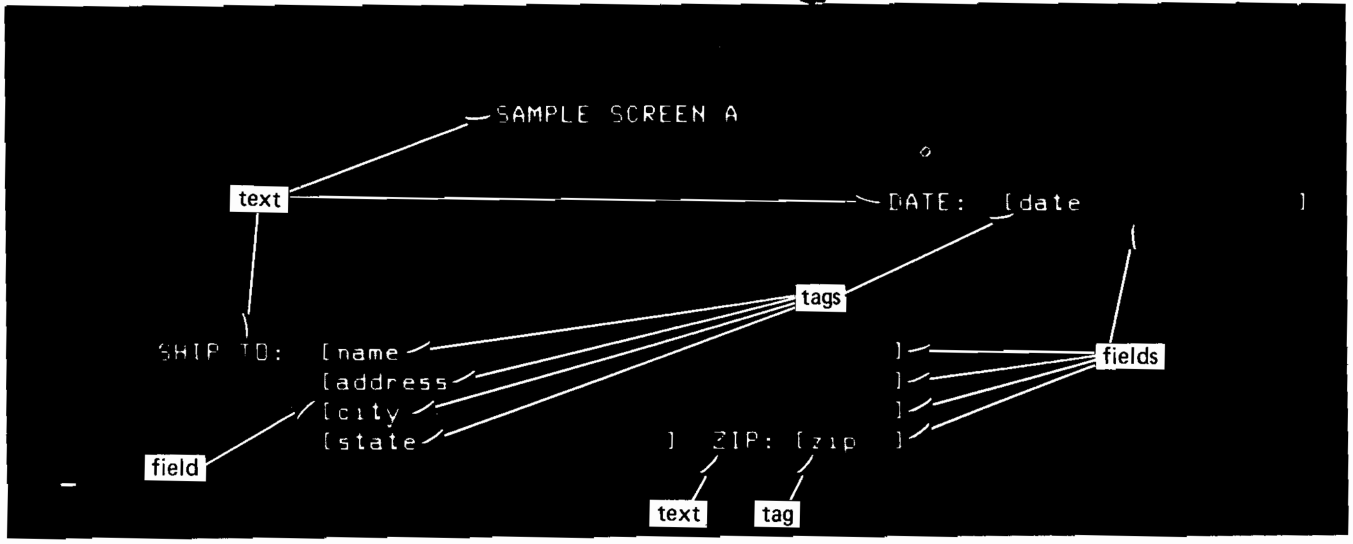
If you want to design a field that is longer than one line, you start the field with a bracket (or ESC/[) and terminate it with a closing bracket (or ESC/]). At the beginning of each intermediate line of the form, you enter an ESC/[. For example, the following field extends across three lines:



Assuming the first and second lines of the field each contain 78 characters and the third line contains 34 characters, the entire field is 180 characters long. This count excludes the printing brackets that delimit the beginning and end of the field.

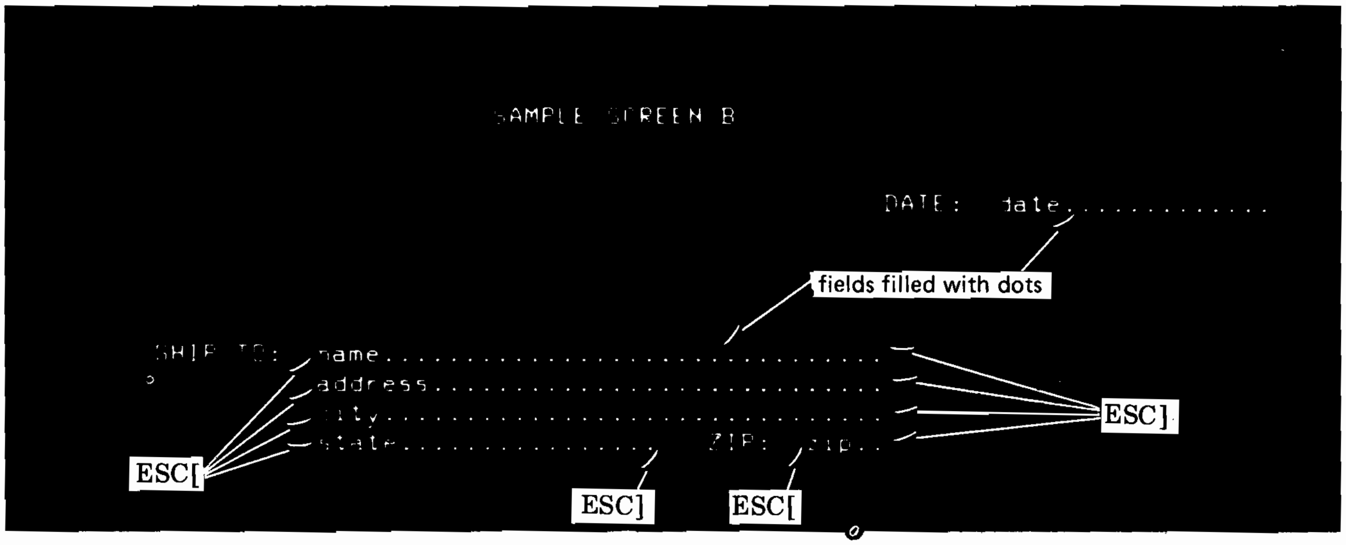
Refer to figure 3-16 for examples of different ways to design a screen.

A. Using Brackets as Field Delimiters:



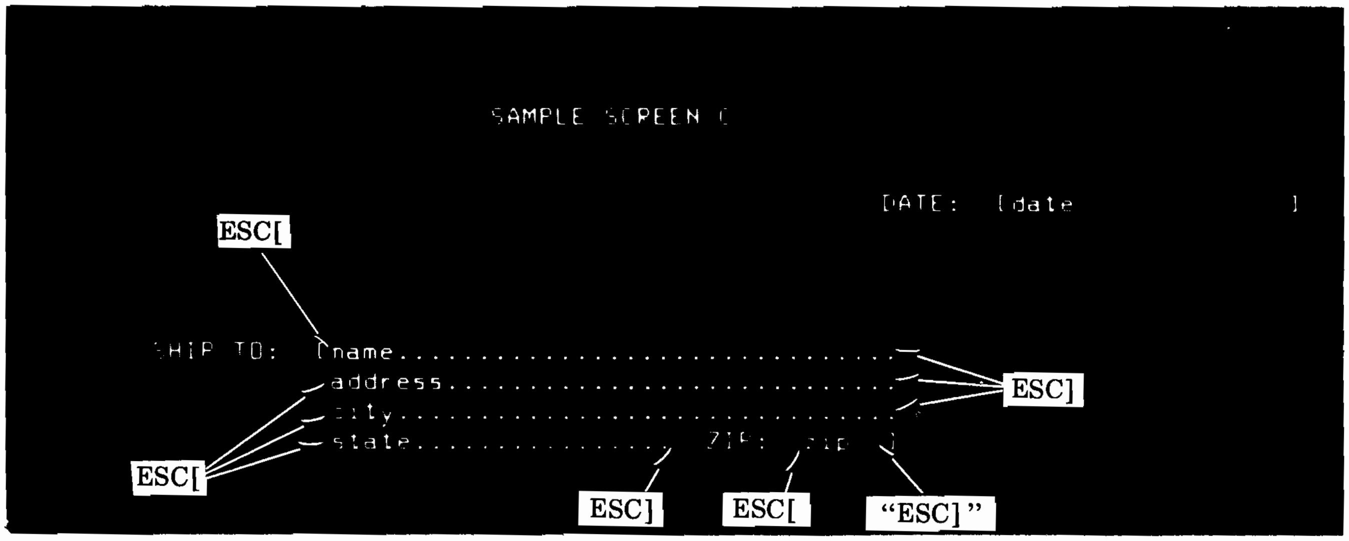
```
                    SAMPLE SCREEN A

                                        DATE:  [date                 ]

   SHIP TO:  [name                              ]
             [address                           ]
             [city                              ]
             [state             ]  ZIP: [zip    ]
```

B. Using ESC[/ ESC] as Field Delimiters:



```
                    SAMPLE SCREEN B

                                        DATE:  date.............

   SHIP TO:  name.................................
             address..............................
             city.................................
             state..................  ZIP: zip..
```

C. Combining Brackets with ESC[/ ESC]



```
                    SAMPLE SCREEN C

                                        DATE:  [date                 ]

   SHIP TO:  [name.................................
             address..............................
             city.................................
             state..................  ZIP: zip   ]
```

Figure 3-16. Sample Screen Designs

FORM FUNCTION KEY LABELS MENU

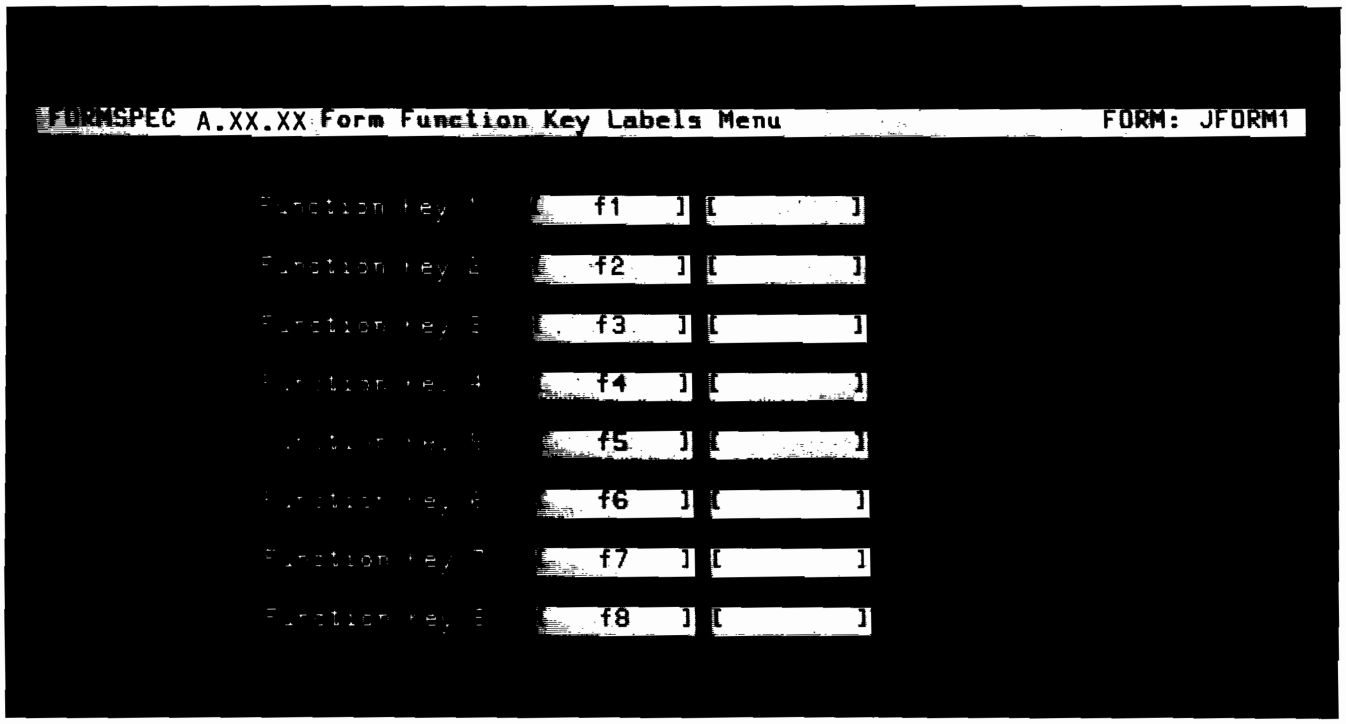
This menu is used to specify new function key labels which will appear when ENTRY.PUB.SYS is executed. Labels you specify will replace original labels provided by V/3000.

Each label consists of two lines of eight characters each. To specify a label enter the first line in the first field and the second line of the label in the second field in the menu. Both lines can be specified on this menu. For example, to change the label for Key 1 to read ORDER NUMBER, enter:

Function Key 1 [ORDERΔΔΔ][NUMBERΔΔ]

If the labels are to be LOCAL, they will be displayed only while that form is on the screen. If the next form does not have LOCAL labels, the file's GLOBAL labels will be displayed.

In the case of a frozen form with another form appended to it, the labels displayed will be those for the appended form. If the appended form has no LOCAL labels, the file's GLOBAL labels will be used.



```
FORMSPEC A.XX.XX Form Function Key Labels Menu                FORM: JFORM1

          Function Key 1    [   f1   ] [        ]
          Function Key 2    [   f2   ] [        ]
          Function Key 3    [   f3   ] [        ]
          Function Key 4    [   f4   ] [        ]
          Function Key 5    [   f5   ] [        ]
          Function Key 6    [   f6   ] [        ]
          Function Key 7    [   f7   ] [        ]
          Function Key 8    [   f8   ] [        ]
```

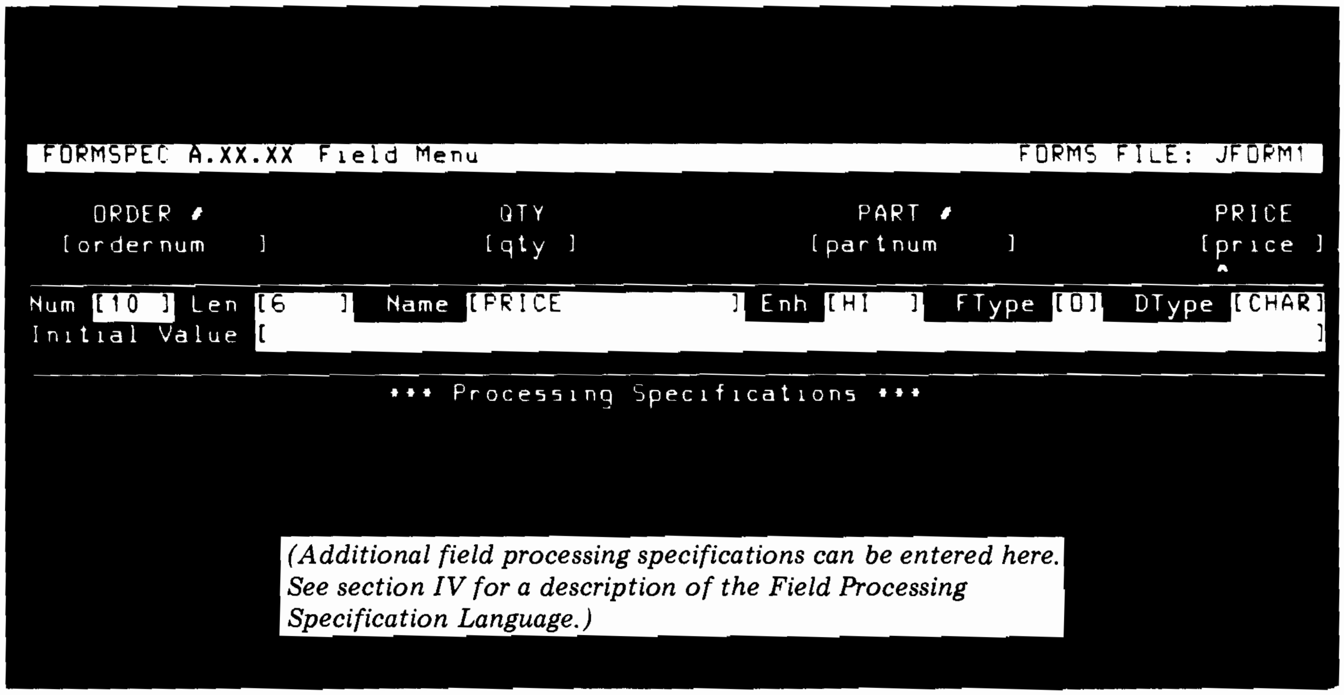
Figure 3-17. FORM FUNCTION KEY LABEL Menu

FIELD MENU

In case you want to assign special field characteristics, FORMSPEC issues a FIELD menu for each field delimited during screen design. Each menu displays up to three lines of the form, and marks the particular field to which the menu applies. It then displays the default characteristics of the field.

You can change the field default characteristics by entering new values and pressing ENTER. If you do not want to change the defaults, simply press NEXT (f6) to display the next FIELD menu. If you plan to use the simplest type of forms design in which all data is accepted and no special edits are performed, you can skip all the FIELD menus by pressing NEXT FORM (f2).

The FIELD menu is illustrated in figure 3-18.



```
FORMSPEC A.XX.XX Field Menu                                    FORMS FILE: JFORM1

      ORDER #                      QTY                    PART #              PRICE
   [ordernum    ]                 [qty ]               [partnum     ]        [price ]
                                                                               ^
Num [10 ] Len [6    ] Name [PRICE            ] Enh [HI  ] FType [O] DType [CHAR]
Initial Value [                                                                 ]

                        *** Processing Specifications ***
```

(Additional field processing specifications can be entered here. See section IV for a description of the Field Processing Specification Language.)

Figure 3-18. FIELD Menu

The FIELD menus are presented in the order the fields appear on the screen, from left to right, top to bottom.

The information you enter on the FIELD menu is divided into two categories: required specifications and optional processing specifications. Only the required specifications are discussed here. The optional processing specifications are described in section IV. The required specifications are:

FIELD NUMBER

Although your form may have a maximum of 128 fields, a number between 1 and 999 is assigned by FORMSPEC to each new field. This number is permanently assigned to the field and is not changed even if other field characteristics are changed. If, however, the field tag is changed, this effectively deletes the field associated with the old tag. The field number is deleted along with the field, and a new number is assigned to the field associated with the new tag.

The "Num" field may not be changed.

FIELD LENGTH

This is the length of the field as determined by the number of characters entered between the field delimiters during screen design.

The length of a field cannot be changed on this menu. If you want to change field length, you must display the screen menu and actually change the field on the screen.

The "Len" field cannot be changed.

FIELD NAME

The field tag assigned during screen design is shifted up to all upper case letters and displayed in this field. You can enter another name in this field. For example, you may want a longer name than would fit in the field; or if two tags are no longer unique when shifted to all capitals, you can rename one of them here. In any case, the name in this field (an upshifted tag or a new name) identifies the field in subsequent references.

You can enter any name up to 15 characters long. Like other FORMSPEC names, it must start with a letter and may be followed by upper or lower case letters (A-Z), numbers (0-9), or an underline (_). It must not be one of the reserved words listed in table 3-2.

Default = the upper-case field tag.

DISPLAY ENHANCEMENT

The default display enhancement for all fields on this form is displayed in this box. You can change the enhancement for the particular field with any combination of the following codes:

Enhancement	Code
Half bright	H
Inverse video	I
Underline	U
Blink	B
Security	S
None	NONE

Default = the defaults established when the screen was entered.

Only four of the five enhancements may be used at one time.

FIELD TYPE

The field type is specified as D, R, O, or P to indicate one of the following options:

Display	— Field cannot be modified by the operator. Intended for display only, the field can receive data as specified by optional processing specifications (see section IV). It is a protected field on the screen, but data written to display only fields is sent to the batch file exactly as if it had been entered by the operator.
Required	— Field cannot be left blank. Operator must enter non-blank values in field.
Optional	— Field may be left blank by operator. Edit checks and field phase processing specifications for the field are skipped if the field is left blank.
Process	— Exactly like an optional field, except that edit checks are performed and field phase processing specifications are executed even if the field is blank.

Default = O; optional.

Required, Optional, and Process fields are all three treated as "input" fields. An input field is one in which the operator can enter or change data, as opposed to a display field which is protected from the operator. Note that initial values may be displayed in any field, but such values can be, and usually will be, changed by the operator.

DATA TYPE

Each field must be specified as one of three data types: character, numeric, or date. Based on the data type, FORMSPEC can determine the basic validity of the data entered, how it is formatted for data movement, and the type of operations that can be performed on the field. The data type is specified as follows:

CHARACTER TYPE

CHAR

Data entered in field is assumed to be a string of any ASCII characters. No validity checking is performed on data at time of entry. No arithmetic operations can be performed on data of this type. (The default data type.)

The operator can enter any characters in a CHAR type field. For example, "$12.59", "A-15-75", "**123**" are all legitimate entries.

NUMERIC TYPES

NUM[*n*]

Data entered in a field of this type must be numeric. The maximum number of decimal places can be indicated by the optional digit, *n*, where *n* is a value between 0 and 9. If you omit the value, *n*, the data item is assumed to have a floating decimal point.

Validity checking is performed on this data type. An optional sign (+ or -) is allowed. Commas are allowed, but if included they must be correctly placed. An optional decimal point may be included in the data.

The operator must enter numeric data in this field. If *n* specifies a maximum number of decimal positions, the operator must include a decimal point when the value has decimal places, and may omit the decimal point when it does not. For example in a NUM2 field, the operator can enter the value "500", or he can enter a value with one decimal place such as "5.5", or a value with two decimal places such as "5.95". Commas and a sign may also be included. For example, the following are legitimate entries for a NUM2 field: "1,390", "-327.00", "+100,000.00", and so forth; but the value "5.678" is disallowed as it has too many decimal places.

DIG

Data entered in a field of this type must be a positive integer. No sign, no commas, and no decimal point can be included. As with NUM data, validity checks are performed on this data type.

For example, "10030", "2", "307" are all legitimate entries in a DIG field, but "10.5", "100,000", or "-10" are not.

IMP*n*

A value entered in an IMP field has an assumed decimal point. The assumed decimal point is *n* places from the right of the value. (Note that the value, *n*, must be specified.) Although permitted, a decimal point should not be entered in this type field, but commas and a sign are allowed. As with NUM type data, validity checks are performed on data of this type.

For example, if the data type is IMP2, and the user enters the value "500", the value is treated as "5.00".

The following table illustrates how each numeric data type interprets an entered value.

Entered Value	Interpreted Value (based on data type)				
	NUM2	IMP2	NUM	NUMO	DIG
10.95	10.95	10.95	10.95	(error)	(error)
1095	1095.00	10.95	1095	1095	1095
10.9	10.90	10.90	10.9	(error)	(error)
10.956	(error)	(error)	10.956	(error)	(error)
-100	-100.00	-1.00	-100	-100	(error)
1,000	1,000.00	(error)	1,000	1,000	(error)
1,00000	(error)	1,000.00	(error)	(error)	(error)

DATE TYPES

MDY

A date entered in an MDY field must be in the order: month, day, year. The data can be entered in any format, such as "FEB 6, 1978" or "02/06/78" or "FEBRUARY 6 78" and so forth.

DMY

A date entered in a DMY field must be in the order: day, month, year. It can be any format, such as "2 MAR 1978" or "02-03-78" or "2/3/78" and so forth.

YMD

A date entered in a YMD field must be in the order: year, month, day. It can be in any format such as "1978, APRIL 12" or "78/4/12" or "78-04-01", or "780402". (Note, that quotes are not part of the date and should not be entered in the field.)

FORMSPEC checks that the entered data is in the correct format and that it is a valid date. Arithmetic operations are not allowed on date fields.

To illustrate how the three date type specifications interpret entered dates, the following tables show legal dates followed by dates that would be diagnosed as illegal.

LEGAL DATES:

MDY	DMY	YMD
February 7, 1978	7 February 1978	1978, February 7
FEB 7 1978	7 FEB 1978	1978 FEB 7
02/07/78	07/02/78	78/02/07
2/7/78	7/2/78	78/2/7
02-7-78	07-2-78	78-2-07
2 7 78	7 2 78	78 2 7
020778	070278	780207

ILLEGAL DATES:

MDY	DMY	YMD	REASON
Febrary 7, 1978	7 Febrary, 1978	1978 Febrary 7	misspelled month
FEBR 7 1978	7 FEBR 1978	1978 FEBR 7	4-letter abbreviation
7 FEB 1978	FEB 7 1978	7 FEB 1978	wrong order
1778	7278	7872	month & day must be 2 digits when there is no separator.

Default Data Type = CHAR

INITIAL VALUE

You may specify an initial value for the field. This value will be displayed in the field when the form is first displayed at the terminal. It is also displayed when the REFRESH key is pressed during data collection and the form is cleared to its initial values.

The value entered here is treated like operator input. That is, it must match the data type of the field and must not be longer than the field length.

(Refer to section IV for a discussion of how FORMSPEC uses these data types to perform validity checks on data entered in the fields, and how data is treated during movement from one field to another.)

EXAMPLE OF FORMSPEC DESIGN

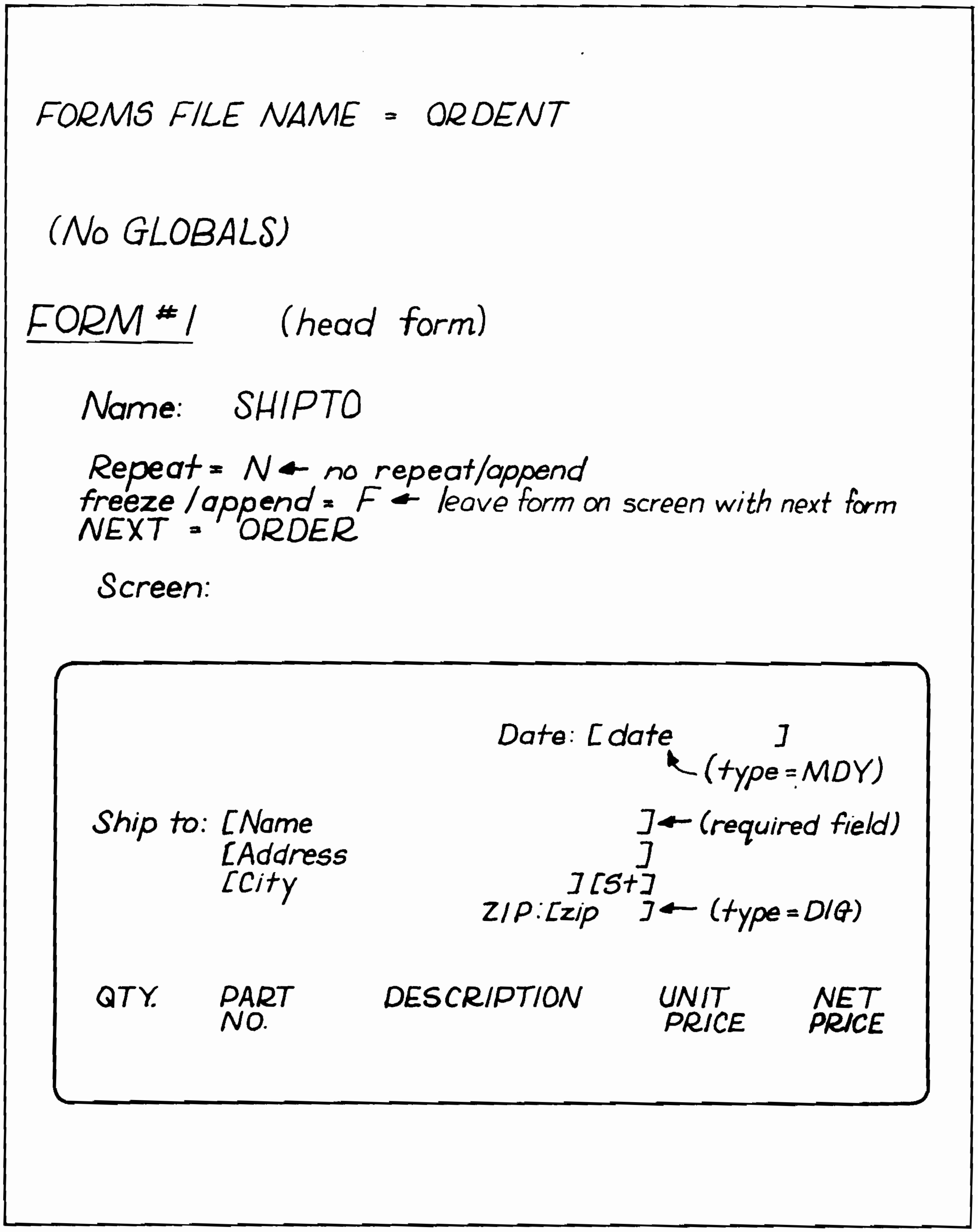
It is much easier to design the screen directly at the terminal where you can visualize how the final form will look, than it is to lay it out rigorously on graph paper before running FORMSPEC. You may, however want to plan the forms sequence and sketch out each screen before running FORMSPEC.

Because FORMSPEC provides default values for each field entered on a screen, the screen design itself is usually sufficient to define the fields. You may indicate on the sketch for each screen any non-default field characteristics.

SAMPLE OF PRELIMINARY FORMS DESIGN

Figure 3-19 illustrates one mehtod for planning a complete forms file. This forms description, when entered through the FORMSPEC menus, will generate the set of forms used as a data entry example in section II. Note that the first form is "frozen" on the screen, the second form is appended to the first and is repeated until the operator presses NEXT to display the next and last form, TOTALS.

When TOTALS is displayed, all previous forms are cleared from the screen. Up until that point the first form is frozen on the screen. Should the second form be repeated so many times that there is no more room on the screen, its first appearance is rolled off while the first form remains on the screen.



FORMS FILE NAME = ORDENT

(No GLOBALS)

FORM #1 (head form)

Name: SHIPTO

Repeat = N ← no repeat/append
freeze/append = F ← leave form on screen with next form
NEXT = ORDER

Screen:

```
                                   Date: [date        ]
                                               ↖(type = MDY)

Ship to: [Name                              ]← (required field)
         [Address                           ]
         [City                       ][St]
                                 ZIP:[zip   ]← (type = DIG)

QTY.     PART      DESCRIPTION       UNIT      NET
         NO.                         PRICE     PRICE
```

Figure 3-19. Sample Forms File Layout

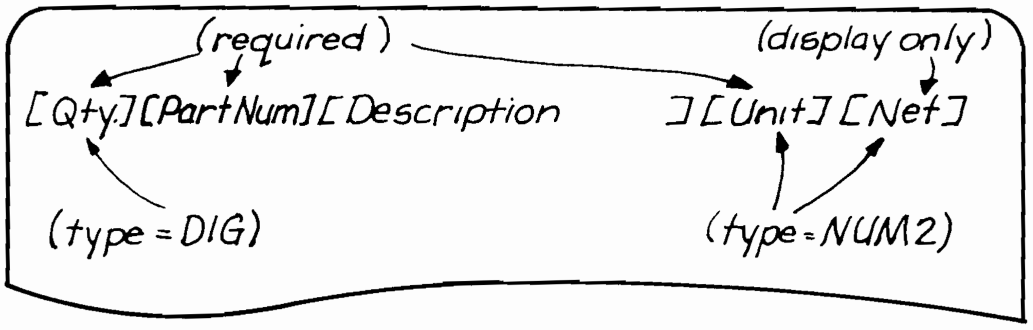
FORM #2

Name = ORDER

Repeat = A ← form is repeated and appended
freeze/append = C ← When next form is requested
by operator or program, current
form is cleared.

NEXT = TOTALS

Screen:



FORM #3

Name = TOTALS

Repeat = N
freeze/append = C ← current form cleared after ENTER
NEXT = $HEAD ← returns to first form

Screen:

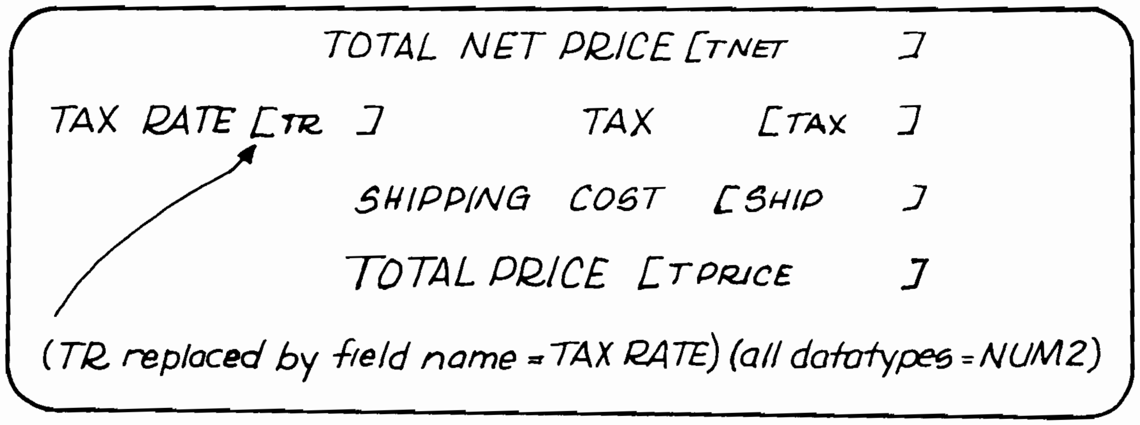


Figure 3-19. Sample Forms File Layout (continued)

ADVANCED FORMS DESIGN

SECTION IV

OVERVIEW

The processing specifications for advanced forms design allow you to:

- Compare a field value to a specified single value, range of values, list of values, or a pattern.
- Test a check digit.
- Specify up to 20 global save fields.
- Assign a value to any field in the form, including save fields.
- Move values between fields in a form or, using save field, between fields in different forms.
- Perform arithmetic calculations on data in numeric fields.
- Format data in a field either during data movement, or using the set of formatting commands.
- Dynamically alter the forms sequencing originally assigned to a form.
- Execute any of the processing specifications conditionally, depending on the value of a field.

LEVELS OF ADVANCED DESIGN

Advance forms design can be separated into two levels: full field edits and advanced processing.

- *Full Field Edits* — use a comprehensive set of editing statements that apply to a single field. These statements allow you to test a value entered in a field for length, and to compare the value to a single value, a range of values, the values in a table, or against a pattern, and to test a check digit in a numeric field. Each of these edits applies to the field in the FIELD menu in which it appears — the editing statements do not cross field boundaries.
- *Advanced Processing* — includes statements that control data movement between fields and across forms, arithmetic calculation and formatting of data, dynamic alteration of forms sequence (including changes to repeat, append, and freeze specifications). Any of these statements and the edit statements can be combined into conditional statements so that processing is performed only in specified circumstances. This type of processing, as in a real processing language, is affected by the order in which the statements are specified. The order of statement execution can be defined explicitly through phase statements.

All advanced forms design uses field processing specifications that are specified in the lower nonformatted area of the FIELD menu. Since these processing specifications are the only place in which save fields are used, adding and deleting save fields is also described in this section.

FIELD PROCESSING SPECIFICATIONS

The field processing specifications perform field editing, data movement, and data formatting. They can also be used to alter the sequence of forms dynamically. Any of the processing specifications can be executed conditionally, based on the result of a field edit.

The processing specifications are entered in the lower nonformatted area of the FIELD menu. Although they apply primarily to particular fields, some specifications apply to the form in general.

The specifications can, if desired, be executed in three phases: initialization, field edits, and finish. If used, phases allow initialization of all fields in the form before data is entered in the fields, editing of data in each field as it is entered, and after all fields are edited, any final processing of the form.

SAVE FIELDS

Save fields are not part of a particular form like other fields, but are "global" to all forms. The save fields can be used by most field processing statements, but are particularly useful for passing data between fields on different forms. A maximum of 20 save fields can be defined for any forms file.

ENTERING PROCESSING SPECIFICATIONS

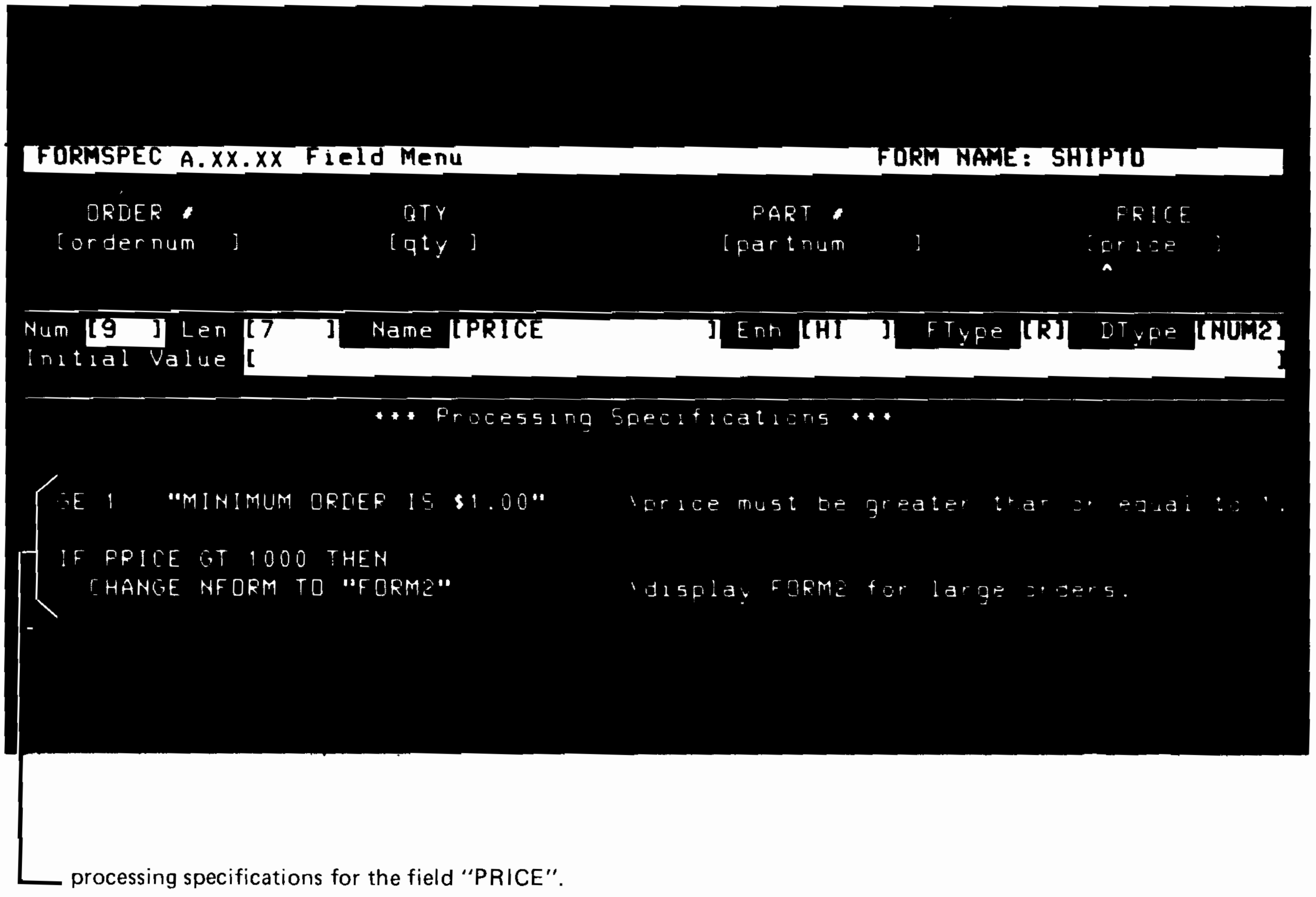
After defining the required field specifications on the FIELD menu (as described in section III), if you want additional processing, press the FIELD TOGGLE function key (f3). You can then enter the field processing statements in the lower (nonformatted) area of the FIELD menu. (Refer to figure 4-1 for an illustration of the FIELD menu that includes processing specifications.)

When additional field processing is requested, FORMSPEC performs the following steps:

1. Leaves the FIELD menu on the screen with its current values for the required specifications.

2. Places the terminal in nonformatted block mode. (This allows you to type anywhere on the screen, not just in unprotected fields, and use the full terminal capabilities for block mode entry such as the INSERT LINE, DELETE LINE, INSERT CHAR, DELETE CHAR keys, the SET and CLEAR TAB keys, and the cursor positioning keys.)

3. Positions the cursor at the beginning of the first line on which you can enter the field processing specifications.



```
FORMSPEC A.XX.XX Field Menu                              FORM NAME: SHIPTO

      ORDER #               QTY                    PART #                 PRICE
    [ordernum   ]          [qty ]                [partnum      ]         [price   ]
                                                                          ^

Num [9   ] Len [7    ]  Name [PRICE                  ] Enh [HI  ]  FType [R]  DType [NUM2]
Initial Value [                                                                         ]

                         *** Processing Specifications ***

 GE 1    "MINIMUM ORDER IS $1.00"      \price must be greater than or equal to 1.

 IF PRICE GT 1000 THEN
   CHANGE NFORM TO "FORM2"             \display FORM2 for large orders.
```

processing specifications for the field "PRICE".

Figure 4-1. FIELD Menu with Processing Specifications

You can then enter any of the field processing statements described in this section. They can be typed on any line in the processing specifications area, and must only conform to the specified syntax rules. When you have typed all the additional specifications for this field, press the ENTER key to enter them, with the required specifications for this field, to the forms file.

NOTE

In nonformatted mode, it is possible accidentally to wipe out the required field description with a key such as CLEAR DISPLAY or DELETE LINE. Doing this alters the field description in an unpredictable manner. To recover, press REFRESH (f4) and then reenter the field description specifications.

When ENTER is pressed, the required field description is checked first. If there are no errors, the field processing specifications are checked for syntax errors. When the FIELD menu passes these checks, the next menu is displayed. You can continue with field definitions or go back to change previously defined fields.

CORRECTING EXISTING SPECIFICATIONS

Whenever you return to a FIELD menu after ENTER is pressed, the terminal will be in formatted mode with the cursor positioned to the first unprotected field in the upper part of the screen. To change the required field specifications, tab to the field you want to change and type in the new specification. To change the field processing specifications in the lower part of the screen, you must press FIELD TOGGLE (f3) to put the terminal in nonformatted mode. The cursor will be positioned to the beginning of the processing specification area.

If you want to change a required field description in the upper part of the FIELD menu when you are in the lower part entering field processing specifications, you must press FIELD TOGGLE (f3) to return to formatted mode. The cursor will be positioned at the first unprotected field in the upper part of the menu, and you can then change the field description exactly as if the menu were newly displayed.

STATEMENT SYNTAX

The field processing specification statements consist of statement names followed by parameters.

The description of each statement uses syntax conventions defined at the beginning of the manual (page vi). In order to understand the notation used in the statement formats in this section, you should review these conventions.

Multiple statements can be placed on the same line by following the last parameter of the statement by a blank and the next statement. If you want, you can separate statements on the same line with an optional semicolon (;). For example:

```
MINLEN 5   GT 1000   LT 2500        3 statements without separator
MINLEN 5;  GT 1000;  LT 2500        3 statements separated by;
```

Statements may begin anywhere on the line, except for nested IF statements where indentation is significant. Multiple blanks are ignored within a line, except at the beginning of a line in a nested IF statement.

COMMENTS

Comments may be included in the text by preceding the comment with a back slash (\). Anything typed between the back slash and the end of the line is ignored when the statement is parsed. For example:

$$\underbrace{\text{GT 12}}_{\textit{statement}} \qquad \underbrace{\backslash\text{This field must have a value greater than 12.}}_{\textit{comment}}$$

CONTINUING LINES

Statements that are not completed before the first back slash or the end of the line can be continued anywhere on the next line. A continuation character, the ampersand (&), is used only when a string literal must be continued on the next line. The ampersand concatenates two or more string literals to form one string. For example:

```
EQ  "ABCDEF"&       \This field must be a string consisting only of
    "UVWXYZ"        \the uppercase letters ABCDEFUVWXYZ.
```

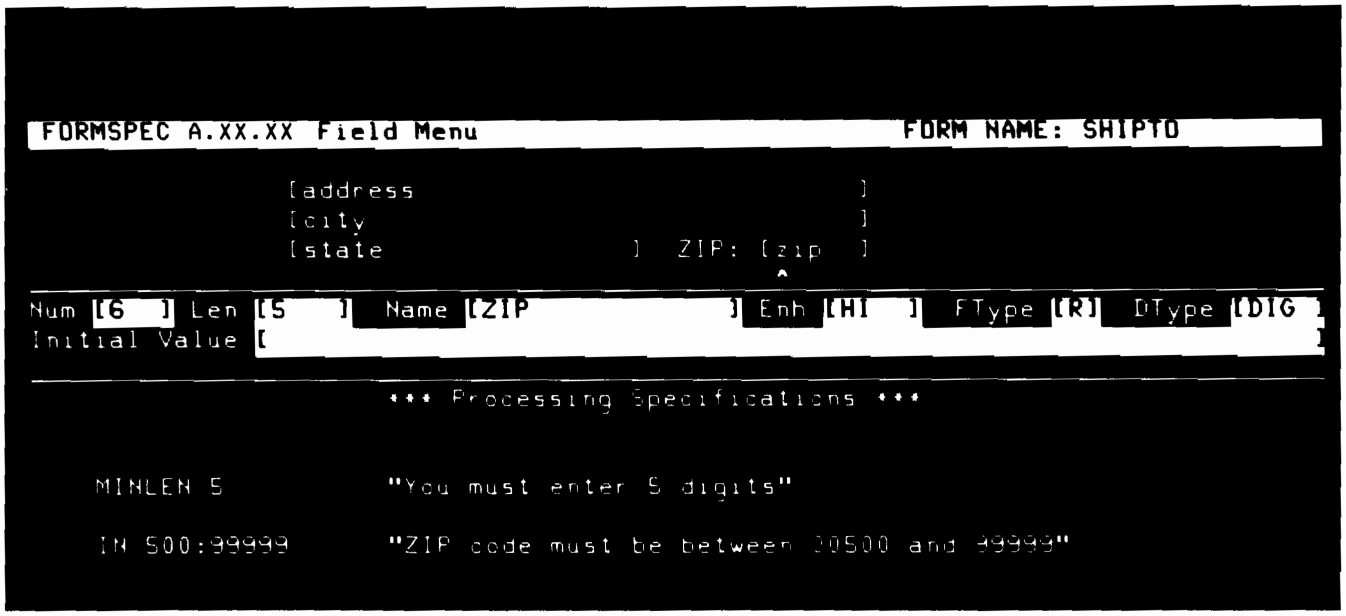
CUSTOM ERROR MESSAGES

Whenever a field edit statement detects an error at run time, an error message is issued automatically. The message is displayed in the window line.

You may choose to write your own message to be issued when a field fails a particular edit specification. To do this, you specify the message in quotes immediately following the statement to which it applies. When and if the statement causes a field to fail, this message is displayed instead of a system error message. For example:

```
MATCH udddd (A:G)       "Field has wrong format for Product Number"
```

Figure 4-2 illustrates a FIELD menu in which the field processing specifications contain custom error messages.



```
 FORMSPEC A.XX.XX Field Menu                          FORM NAME: SHIPTO

                     [address                           ]
                     [city                              ]
                     [state                ]  ZIP: [zip  ]
                                                    ^
Num [6  ] Len [5  ] Name [ZIP              ] Enh [HI  ] FType [R] DType [DIG ]
Initial Value [                                                              ]

                      *** Processing Specifications ***

    MINLEN 5              "You must enter 5 digits"

    IN 500:99999          "ZIP code must be between 00500 and 99999"
```

Figure 4-2. FIELD Menu With Custom Error Messages

The required specifications for the field determine the first checks made by the application program on the data in the field. For example, if the field is required (Ftype=R), the program checks that the operator has entered a value in the field. Editing is performed according to the data type. For example, if the data type is DIG, then the program checks that only the digits 0-9 were entered in the field.

Next, the program tests the field processing specifications in the order they are entered. As an example, consider the specifications in figure 4-2. After making the checks based on the required specifications, data entered in the field is tested to be sure that at least 5 digits were entered (MINLEN 5), and then the value is checked to be sure that it is within the range 500 through 99999 (IN 500:99999).

Table 4-1 is a summary of all the statements that may be entered in the Processing Specifications area of the FIELD menu. Each of these statements is fully described later in this section.

Table 4-1. Summary of Processing Statements

CONFIGuration Statements		
LOCALEDITS	alphabetic	integer_fill
	alpha_numeric	justify
	constant	must_fill
	dec_digits n	required
	dec_type_EUR	sign_dec
	dec_type_US	sign_dec_fill
	imp_dec	transmit_only
	imp_dec_fill	unrestricted
	integer	upshift
LIGHT	@	
	A	
	B	
	. . .	
	N	
	O	
	P	
DEVICE	PRINTER	
	KEYBOARD	
	MAGSTRIPE	
	TYPEV	
	MFR	
	CAD MARKS	CUT
	NONE HOLES	NOCUT
	CAD HOLES	
	DISPLAY	
	LARGECHAR	
	STDCHAR	(DEFAULT)
	BARCODE	
	125	
	139	
	MAT	

Table 4-1 is continued on the following page.

Table 4-1. Summary of Processing Statements (continued)

Edit Statements

```
MINLEN value* ["msg"]                 ⎧ GT ⎫
                                      ⎪ LT ⎪
                                      ⎨ GE ⎬   value ["msg"]
                                      ⎪ LE ⎪
                                      ⎪ EQ ⎪
                                      ⎩ NE ⎭

        ⎧ value*[,value*]. . .                                    ⎫
⎧ IN  ⎫ ⎨ lowvalue*:highvalue*[,lowvalue*:highvalue*]. . .        ⎬  ["msg"]
⎩ NIN ⎭ ⎩ value*[,lowvalue*:highvalue*]. . .                      ⎭

MATCH pattern ["msg"]                 CDIGIT ⎧ 10 ⎫ ["msg"]
                                             ⎩ 11 ⎭
```

Formatting Statements

```
      ⎧ TRAILING ⎫                                     ⎧ LEFT   ⎫
STRIP ⎨ LEADING  ⎬ "characters"                JUSTIFY ⎨ RIGHT  ⎬
      ⎩ ALL      ⎭                                     ⎩ CENTER ⎭

      ⎧ LEADING  ⎫                             UPSHIFT
FILL  ⎩ TRAILING ⎭ "character"
```

SET Statement

```
    ⎧    TO          ⎧ value*                          ⎫       ⎫
    ⎪                ⎩ index OF value*[,value*]. . .   ⎭       ⎪
SET ⎨                                                          ⎬
    ⎪ ⎧ field      ⎫ ⎡TO        ⎧ value*                         ⎫⎤ ⎪
    ⎩ ⎩ save field ⎭ ⎣          ⎩ index OF value*[,value*]. . .  ⎭⎦ ⎭
```

CHANGE Statement

```
       ⎧ CFORM TO ⎧ NOREPEAT      ⎫                                              ⎫
       ⎪          ⎨ REPEAT        ⎬                                              ⎪
       ⎪          ⎩ REPEAT APPEND ⎭                                              ⎪
CHANGE ⎨                                ⎡ "formname"                       ⎤ ⎫   ⎬
       ⎪                                ⎢ fieldname                        ⎥ ⎪   ⎪
       ⎪          ⎧ ⎡ CLEAR         ⎤   ⎢ index OF value*[,value*] . . .   ⎥ ⎪   ⎪
       ⎪ NFORM TO ⎨ ⎢ APPEND        ⎥   ⎢ $RETURN                          ⎥ ⎬   ⎪
       ⎪          ⎩ ⎣ FREEZE APPEND ⎦   ⎢ $HEAD                            ⎥ ⎪   ⎪
       ⎪                                ⎢ $END                             ⎥ ⎪   ⎪
       ⎩                                ⎣ $REFRESH                         ⎦ ⎭   ⎭
```

Conditional Statement

```
IF [value*] editstatement THEN [statement] . . .
    [statement] . . .
⎡ELSE [statement] . . . ⎤
⎣   [statement] . . .   ⎦
```

Control Statements

```
FAIL ["msg"]          STOP
```

*value can be a constant, field, save field, expression, or index retrieve operand.

OPERANDS

Wherever "value*" is specified as an operand in a field processing statement, a field name, constant, or a save field name can be used. If the operand is numeric, any of these can be combined into an arithmetic expression whose result is used as the operand.

Another operand, index retrieve, is used to retrieve items from a list by an index value. It must be enclosed within parentheses in edit statements; in SET and CHANGE statements, it can be specified without the enclosing parentheses.

FIELD

The name of any existing field within the same form can be specified as a "value" operand. When a field name is specified, the value in that field at run time is used to check the current field.

A field name in an editing statement must describe a field of the same data type (character, numeric, or date) as the current field.

CONSTANT

Constants are divided into categories corresponding to the three main data types: character, numeric, and date. In addition, there are four system defined constants:

$EMPTY	equivalent to all blanks; in comparison, $EMPTY is less than any other value. *(any date type)*
$LENGTH	equivalent to the length of the field at compile time. *(numeric data types).*
$STATE	equivalent to a table of constants each of which is a 2-character upper or lower case state code; refer to appendix F for the codes. (*character type; used only in table checks*)
$TODAY	equivalent to today's date in the form dd/dd/dd where the order of month day year is determined by the field type. *(date type only).*

Character. Character constants are strings of any character enclosed in single or double quotes.

For example:

```
"This is a constant."
'and this is, too!'
$EMPTY (all blanks)
$STATE (any 2-character state code)
```

Numeric. Numeric constants are digit strings with an optional leading sign and optional decimal point. Commas are not allowed. If the constant has no decimal point, a decimal is assumed at the right of the rightmost (least significant) digit. $EMPTY (all blanks) can be used to indicate a numeric constant less than any non-$EMPTY value. $LENGTH can be used to indicate the number of characters in the current field at compile time. The following examples illustrate numeric constrants:

.23
-123
500
-3729
$EMPTY (all blanks)
$LENGTH (number of characters in field at compile time.)

Date. A date constant can be any legal date format, but it must be in the order MDY. To distinguish it from other constants, a date constant must be delimited by exclamation points (!). Note that the date constant in the order MDY is used to check date values in any of the three legal orders: MDY, DMY, or YMD. A special date constant $TODAY is equivalent to the date at execution time.

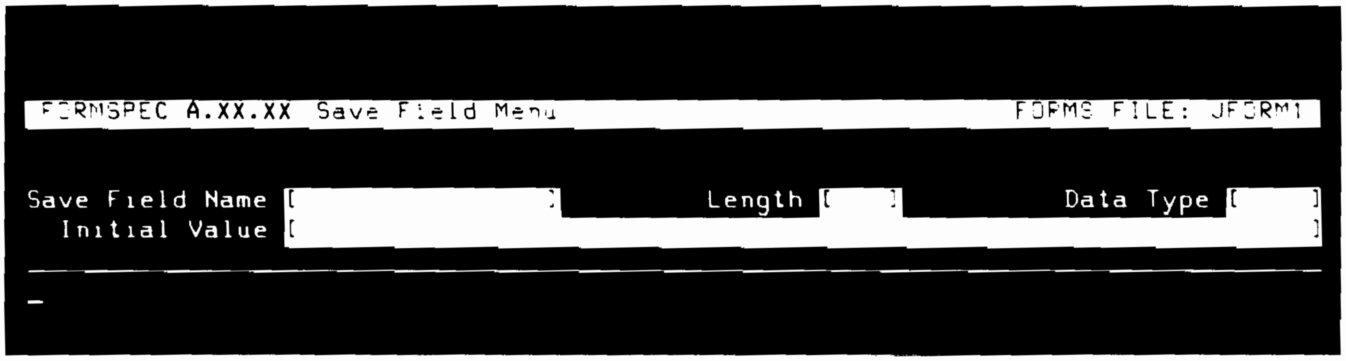
Examples of date constants are:

!MAY 30, 1973!
!12/24/78!
!NOVEMBER 7, 78!
$TODAY (date at execution time in order of destination field)
$EMPTY (all blanks)

SAVE FIELD

A save field is a global field whose value can be used anywhere in the forms file where field references are allowed. Save fields can be thought of as standard fields that are global to all forms in the file. Like other fields, they always contain ASCII characters.

Save fields are specified on the SAVE FIELD menu which is issued when requested on the MAIN menu. Any defined save fields are stored at the beginning of the forms file just after the global specifications. (Refer to section III for the order of specifications in the forms file.) The SAVE FIELD menu is illustrated in figure 4-3.



```
FORMSPEC A.XX.XX Save Field Menu                          FORMS FILE: JFORM1

Save Field Name [                  ]      Length [   ]          Data Type [     ]
  Initial Value [                                                                ]
```

Figure 4-3. SAVE FIELD Menu

A save field name appears exactly the same as a field name. To help distinguish the global save fields from local fields, you may want to establish a naming convention. For example, if all save field names (and no local field names) start with the letters SF, then any field name starting with SF is immediately recognizable as a save field.

Up to 20 save fields can be defined in one forms file. The SAVE FIELD menu is usually requested through the MAIN menu. To add a save field, press MAIN (f7) to request the MAIN menu and then enter "S" in the selection box. A blank SAVE FIELD menu is then displayed. You can also display a blank SAVE FIELD menu by pressing PREV (f5) from the first form menu.

To delete a save field, you also request the MAIN menu and then enter "D" in the selection box, and the name of the field to be deleted in the field name box.

The following information is associated with each save field defined:

Save Field Name	A standard field identifier composed of up to 15 characters, beginning with an upper or lowercase letter. The remaining characters can be upper or lower case letters, digits, or underlines. For example, SF3, D2-15, or ALPHA. Note that if you use a lowercase letter in a name, FORMSPEC shifts it to an uppercase letter. (The name must not be one of the reserved words listed in table 3-2.)
Save Field Data Type	The data types allowed are the same as for any field. The general types are character (CHAR), numeric (NUM[n], IMPn, or DIG), or date (MDY, YMD, or DMY). (Refer to the FIELD menu description in section III, for a complete discussion of these data types.)
Save Field Length	Like other fields, save fields must be assigned a length, representing the maximum number of characters allowed in the field. Note that you must supply a field length, unlike field specifications where field length is determined from the screen design. Since save fields may be used as accumulators, it is important that the save field length be long enough to avoid rounding and/or truncation when summed values are moved to the field.
Initial Value	An initial value may be assigned to any save field. This is an optional specification; if omitted, all fields are set to blanks ($EMPTY). If specified, the initial value must be the same type as the specified data type. Values are entered exactly as if they were entered by a terminal operator; that is, characters are not delimited by quotes, and dates are not delimited by exclamation points. For example:

12.5	type is NUM1
3790	type is DIG
FEB 3, 78	type is MDY
John	type is CHAR

When the forms file is opened, initial values (default or specified) are assigned to the save fields.

ARITHMETIC EXPRESSION

Any numeric constant, field, or save field can be combined into an arithmetic expression. The expression is evaluated to generate a constant which can be used to check a field or replace the value of a field.

The operators used to form an arithmetic expression are:

+	(add)
-	(subtract)
*	(multiply)
/	(divide)
%	(percent of)

The operators determine the order of evaluation in the standard operator hierarchy where + and - are evaluated after *, /, and %. Operators at the same level are evaluated from left to right. Parentheses may be used to further define the hierarchy. Expressions within parentheses are evaluated first and if parentheses are nested, the innermost are evaluated first.

The operator % in the expression a%b is equivalent to (a * .01) * b.

When $EMPTY is used in an arithmetic expression, the result is always $EMPTY. Thus, $EMPTY with any operator and operand = $EMPTY.

The following examples illustrate arithmetic expressions:

F2 + 1	Add 1 to the value in field F2.
QUANTITY * UNITPRICE	Multiply value in QUANTITY by value in UNITPRICE.
TOTALCENTS/100	Divide value in TOTALCENTS by 100.
20% (TOTALPRICE - TAX)	Find 20 percent of the value resulting from subtracting TAX from TOTALPRICE.
$EMPTY + 20	The result is always $EMPTY, regardless of the operator or other operand.

NOTE

When ENTRY executes an arithmetic expression it expects each operand in the expression to have a value; if any operand does not have a value, the result is null. It appears to the operator as if the expression had not been executed and no message is issued.

CONFIGURATION COMMANDS

You can use the configuration commands to specify local editing to be done on the HP 2624 terminal, and to specify the input device and prompting lights for use on the HP 3075 or HP 3076 terminals.

These commands may only be used in the CONFIG phase of the field processing specifications area.

DEVICE

Enables the input media device

```
DEVICE {device1,device2, . . .device n}
```

This command allows you to override the GLOBAL attribute values for the Multifunction Reader and Bar Code Reader. You must specify which devices are to be used for input/output. Any of the following may be specified: printer, keyboard, magstripe, typeV, multifunction reader, display, and bar code reader. There are no default devices. The following example illustrates this command:

```
                              ***Processing Specifications***

CONFIG
DEVICE   printer, keyboard, display, barcode
```

LIGHT

Specifies which light will be lit during field presentation.

```
LIGHT {light1,light2, . . .light n}
```

This command allows you to specify which, if any, of the seventeen available prompting lights is to be turned on during a field presentation. As many lights as desired can be specified.

Specifying @ will turn on the light associated with the shift key. The remaining valid characters of A through P are associated with the same characters on the alphanumeric keyboard. The following example illustrates this command:

```
                    ***Processing Specifications***

CONFIG
LIGHT  A,B,D,G,N,P
LIGHT   @
```

LOCALEDITS

Used to stipulate edits to be done within the terminal.

```
LOCALEDITS {edit1,edit2,. . .edit n}
```

This command allows you to specify during forms file creation or modification a set of local edits that can be implemented at run time. V/3000 will identify the terminal at run time. If the terminal supports local edits, V/3000 will load to the terminal the local edits you specified. If the terminal does not support local edits, V/3000 will ignore them. An example set of edits can be found in Appendix G.

V/3000 "edit" processing specifications and "terminal" edit processing statements are separate and are not checked for compatibility.

You can specify one or a string of terminal edits to be performed. The following example illustrates this command.

```
                              ***Processing Specifications***

CONFIG
LOCALEDITS constant             No characters may be entered from the keyboard.

CONFIG
LOCALEDITS dec_type_US          United States number format.
LOCALEDITS alphabetic           Upper and lower case characters.
```

INDEX RETRIEVE OPERAND

An index retrieve operand (enclosed within parentheses) can be used in any edit statement where a constant, field, save field, or expression is legal. An index retrieve operand without enclosing parentheses can be used in SET and CHANGE statements.

An index retrieve operand is specified in the format:

```
index OF element [,element] . . .
```

where

index	is a numeric type field or save field, or a numeric expression whose value is a positive integer. The index value indicates which element is selected.
element	is a constant, field, save field, or expression. Each element in the list of elements separated by commas must be the same type (character, numeric, or date).

When an operand of this type is specified, its effective value is selected from a list of values according to its position in the list, where the first element of the list is 1. First, the index expression is evaluated, then this value is used to select a final value from the list.

For example:

EQ (N OF "ABC", "DEF", "GHI", FIELDX)	If the value of N is 1, the character string "ABC" is the selected operand; if the value of N is 4, the value of FIELDX is selected. If the value of N is negative, zero, or greater than the number of elements, an error message is returned. Note that the operand is enclosed in parentheses because EQ is an edit statement.
SET TO F3 OF F1, 15, 20, 25, 35, 40	If the value of F3 is 1, the selected operand is the value of field F1; if F3 is 4, the selected operand is 25. Parentheses are not required in a SET statement.
CHANGE NFORM TO X OF "FORMA", "FORMB", "FORMC", OPSELECT	X may be a value between 1 and 4. If its value is 4, the field name OPSELECT is chosen. The value of OPSELECT must be type character to conform to the other elements in the list, and to make sense it should contain a form name. No parentheses are needed in a CHANGE statement.

If an index has a value beyond the number of elements in the list or is $EMPTY, an error is returned.

FULL FIELD EDITS

Should only field editing be needed, the edit statements listed in table 4-2 and described below provide full field edits. Edit statements apply to the current field only. During table and range checks leading and trailing blank are stripped. (The current field is the field defined in the FIELD menu in which the edit statements are entered.)

EDIT STATEMENTS

The FORMSPEC edit statements allow you to check whether the value of the current field:

- is at least a specified minimum length;
- is greater than, equal to, or less than a specified value;
- is one of a list (table) of values, or is not in the list;
- is within a range of specified values;
- matches a general pattern of characters;
- has a valid check digit.

Edit statements use the set of statements listed in table 4-2. They are performed in the order they are entered by the forms designer. As soon as an edit fails during execution, the field is marked in error and further processing on that field is abandoned.

Table 4-2. Edit Statements

Edit Type	Statement	Meaning
Length	MINLEN value* ["message"]	Checks that field data has at least the number of characters specified by value* (excluding leading and trailing blanks).

Single Value Checks	{GT | LT | GE | LE | EQ | NE} value* ["message"]	Checks that field data has specified relation to another value*. Where:
		GT = Greater than
		LT = Less than
		GE = Greater than/equal to
		LE = Less than/equal to
		EQ = Equal to
		NE = Not equal to

Table Checks and/or Range Checks	{IN | NIN} {value* [,value*] . . . | lowvalue*:highvalue* [,lowvalue*:highvalue*] . . .} . . . ["message"]	Checks that field data is identical to one of (IN) or differs from all (NIN) values* in list. Checks that field data is within (IN) or is not within (NIN) any of the specified ranges.
Pattern Match	MATCH pattern ["message"]	Checks that field data matches, character by character, a specified pattern.
Check Digit	CDIGIT {10 | 11} ["message"]	Uses modulus 10 or 11 type check on numeric field with a check digit to insure that number is keyed in correctly.

*The operand "value" can be a constant, a field name, a save field name, an index retrieve operand within parentheses, or for numeric values, an arithmetic expression.

NOTE

Edit statements are allowed only in the field edit phase, (Refer to phase description at the end of this section)

LENGTH CHECK

To specify the minimum length of a field value, enter the statement:

```
MINLEN value ["message"]
```

A value longer than the field cannot be entered because of the physical limit imposed by the unprotected field area. For this reason, maximum field length need never be specified as an edit check. You may, however, specify the minimum number of characters to be entered with the MINLEN statement.

The value specifies the minimum number of characters allowed in the field. Note that this length does not include leading and trailing blanks.

The value can be a field, constant, save field, or a numeric expression, or it may be an index retrieve operand within parenthesis.

The system defined value $LENGTH may be specified. This value allows you to indicate that the field must be filled. The advantage of using $LENGTH rather than the current field length is that $LENGTH allows you to change the field length without changing the MINLEN specification. $LENGTH is equal to the length of the field when the forms file is compiled.

1. For example, the minimum number of characters entered in a 6-character field could be specified as:

   ```
   MINLEN 2
   ```

   If the operator spaces over three characters, types in one nonblank character, and leaves the rest of the field blank, an error is diagnosed.

2. In another example, MINLEN is used to specify that the field must be filled. The following statement forces the field to be filled, leaving no leading or trailing blanks:

   ```
   MINLEN $LENGTH
   ```

You can also use an indexed operand as follows:

```
MINLEN (LEN OF F1,F2,F3,F4,F5)
```

LEN must contain an integer value between 1 and 5, and the current value of the fields F1 through F5 must be positive. The minimum length depends on the value of LEN and the respective values of F1 through F5.

SINGLE VALUE COMPARISONS

To compare the field value against a single value, use the statement:

$$\left\{\begin{array}{l}\text{GT}\\\text{LT}\\\text{GE}\\\text{LE}\\\text{EQ}\\\text{NE}\end{array}\right\}\quad \text{value} \qquad [\text{``message''}]$$

You may use these statements to compare the contents of the current field with a specified value. The specified value can be a constant, field, save field, or an arithmetic expression. You may also use an index retrieve operand enclosed in parentheses. Any field can be compared to the special comparison value $EMPTY that represents the lowest value for any data type.

The main rule governing these comparisons is that only values of the same data type (character, numeric, or date) may be compared. The comparison is performed from left to right after leading and trailing blanks have been stripped from the field.

CHARACTER COMPARISONS

If the two values being compared are not the same length, the shorter is padded with blanks on the right until it is the same length as the longer value. Then the comparison is made.

In a character comparison, values are considered equal only if each character matches. One value is less than another if, at the point of mismatch, one character is numerically less than its counterpart. ASCII order is used to determine the numeric value of a character. (Refer to appendix C for the ASCII collating sequence.)

For example:

a	is greater than A
A	is greater than △
AA	is greater than A△
A△	is greater than △a

where △ represents a blank.

NUMERIC COMPARISONS

When numeric values are compared, they are first converted to the HP 3000 internal representation of the number. Thus, a field of type IMPn can be compared with NUMn, and NUMn or IMPn fields can be compared to type DIG fields. For example, assume a field of type NUM2 in which the value 123 has been entered. This value can be successfully compared to the constant "123", or a field of type IMP, NUM, or DIG so long as the field contains the value 123 at run time.

For example:

123.000 (NUM3) is equal to 123 (DIG)
123.000 (NUM3) is equal to 12300 (IMP2)

DATE COMPARISONS

Two dates may be compared even if they differ in format and order. This means that the date February 9, 1978 specified in the current field as 9 FEB 1978 (DMY) can be successfully matched against the constant !02/09/78! whose order is MDY. Remember: a date constant must always be in the order MDY.

For example:

2/3/78 (MDY) is equal to Feb 3, 1978 (MDY)
2/3/78 (MDY) is equal to 78/2/3 (YMD)

Some examples of single value comparisons are:

EQ F3	The current field value must exactly match the value in F3 at run time. (Any leading or trailing blanks are stripped from the value in both fields before the comparison is made.)
GT 143.56	The current field value must be greater than 143.56
NE $TODAY	A date entered in this field cannot be today's date.
GE "Cd"	The current field may contain any value greater than "Cd", such as "D" or "Cde" but it cannot be "C" or "CD". Note that all uppercase characters have a lower value than any lowercase character.
GE !5/7/73!	Any date including and after May 7, 1973 may be in current field. The date may be in any of three formats: MDY, DMY, or YMD.

TABLE CHECKS

To verify that the field value is either in a list (table) of values, or is not in that list, enter the following statement:

```
{IN }  value1    [,value2] . . .    ["message"]
{NIN}
```

The values specified in the list may be any mixture of field names, constants, save fields, expressions, or an index retrieve operand in parentheses. Each element in the list must result in a single value to be matched exactly. There is an implicit OR between the elements of the list so that the statement can be understood as:

Is current value equal to (IN) value1 OR value2 OR . . .

Is current value not equal to (NIN) value1 AND value2 AND . . .

One system defined table, $STATE, is provided that consists of a list of all 50 2-character state codes (see appendix F for a list of the codes). This list is in alphabetic order and can be compared successfully with all upper or lowercase codes. $STATE may be used anywhere a table is legal in the statement syntax.

As with other comparison, values must match the field data type, that is, a numeric field can only be compared to a list of numeric values; a date field to a list of date values, and a character field to a list of character values.

Some examples of table verification are:

IN 12, 14, 16, 18, 20	Current value must be one of the five listed numbers.
NIN "CA", "ME", "NY"	Current value must not be any of the three listed values.
IN $STATE, MX	Value must be a legitimate state code or by "MX". Note that the system constant $STATE can be part of a list including other values.

RANGE CHECKS

To check whether the field value is within, or is not within, a specified range of values, use the following statement:

```
{IN }
{NIN}  lowvalue:highvalue[,lowvalue:highvalue] . . . ["message"]
```

Range checks are similar to table checks except that you specify a list of ranges rather than a list of exact values. The range is inclusive, that is the field value must be within the range that includes the lowvalue and the highvalue.

The low and high values may be any combination of field names, constants, save fields, arithmetic expressions, or index retrieve operands. As with table checks, an implicit OR is understood between the ranges in the list. Thus, the statement can be interpreted as:

Is current value in (IN) range1 OR range2 OR . . .

Is current value not within (NIN) range1 AND range2 AND . . .

The low value must not be greater than the high value. If it is, an error is issued when the form is executed (not when it is compiled).

Some examples of range checks are:

NIN 12:45	Current numeric value must not have any value between 12 and 45 inclusive.
IN F2/2:F2*2	The field can have any numeric value ranging from half the value in field F2 through twice the value in F2. Note that the current field and F2 must be numeric in order to use an arithmetic expression in the range check.
IN $TODAY:!12/31/79!	Any date between today's date and December 31, 1979 may be entered.

Range and table checks can be combined in one statement, as illustrated in the following examples:

NIN -12.5:-2,25,1000:FIELD3	Value may NOT be in range -12.5 through -2, nor equal to 25, nor in the range from 1000 through the current value of FIELD3.
IN "ADE":"BB", "s" "t"	The field may have any value in the range "ADE" through "BB" or it can be "s" or "t". Thus, it could be "Abcd", or "B" or "t", but it cannot be "AB" or "S". Note that all lowercase letters are greater than any uppercase letters.

CHECK DIGIT

To test the check digit in a numeric field (modulus 10 or 11), use the following statement:

```
CDIGIT   {11}   ["message"]
         {10}
```

Check digit verification is a special check on a numeric or alphanumeric field in which the last (rightmost) character is a check digit. Verification can be either modulus 10 or modulus 11. You can use the REFSPEC program described in section V to add a check digit to a field value.

Modulus checks are used when the risk of error keying in numbers must be reduced to a minimum. Depending on the modulus selected, single digit errors, and single or double transpositions can be checked using a check digit. Modulus 10 detects single transpositions and incorrect keying of a single digit. Modulus 11 detects these, plus double transpositions. (Refer to appendix D for an exact description of the modulus 10 and 11 checks.)

In general, a check digit is arrived at by performing calculations on a number and then using the result of these calculations as the final digit or "check digit" in that number. For example, suppose a 5-digit charge account number is to be assigned to a new account. The specified calculations (modulus 10 or 11) are performed on the 5-digit number, the result is added as a check digit to the number, and a 6-digit number is assigned as the new account number.

Thereafter, when this number is keyed in, if the CDIGIT edit test is selected, then the last digit is checked against the same calculations used to generate the number. If the check digit is not equal to the result of the calculations, the number was keyed incorrectly, and an error is diagnosed.

Note that a number derived using modulus 10 calculations can only be checked by CDIGIT 10; and a number derived using modulus 11 can only be checked by CDIGIT 11.

Letters of the alphabet can be checked by either a modulus 10 or 11 check. Digits are assigned to the letters so they can be treated like numbers. Thus a field with a check digit can contain a mix of numbers and letters, but must not contain any special characters. Any initial plus or minus sign is ignored.

PATTERN MATCH

To test a field value against a pattern of characters, use the following statement:

```
MATCH pattern ["message"]
```

The pattern match allows you to check field values against a general pattern. It can be used to check the actual value, but actual values are more easily checked in the Single Value, Table, or Range tests. This test is generally used to test the type of character entered in a particular position. Before data is checked against a pattern, any leading and trailing blanks are stripped.

The pattern consists of a series of special characters that indicate the type of data that can be entered in that position. These characters are:

a	upper or lower case alphabetic character (A-Z, a-z)
u	uppercase alphabetic character (A-Z)
l	lowercase alphabetic character (a-z)
b	blank
d	a digit (0-9)
?	any character

The beginning of the pattern is defined by the first non-blank character after MATCH and, in the simplest case, is terminated by the first blank encountered.

A pattern may contain embedded blanks only if it is enclosed within braces { }. Also, a pattern can span more than one line if it is enclosed within braces { }. If a pattern is within braces, it is terminated by the first blank outside the braces.

The match pattern can include specific characters in addition to the types listed above. To illustrate:

```
MATCH Aaa-ddd
```

This pattern means that the value must start with the letter "A", be followed by any two upper or lowercase letters of the alphabet, followed by a hyphen and then three digits. For example, the value "Acs-123" or "AAA-999 are acceptable, but the values "Bcs-999", or "A12-345" are not.

TRANSPARENCY

A special operator can be used to indicate that a pattern character is to be used as an actual value. For example, suppose you want the lowercase letter "a" to be an exact value in the pattern, you can do this by preceding it, or any of the other special pattern characters, with an exclamation point (!). For example:

```
MATCH !addd      Value must start with the letter "a" followed by any three digits.
```

The exclamation point (called the transparency operator) is also used to allow inclusion of any of the pattern operators listed below, and described in table 4-3.

!	(transparency)
,	(choice)
:	(range)
{ }	(grouping)
[ ]	(optional)
+	(repetition — 1 or more)
*	(repetition — 0 or more)

CHOICE

You can indicate a selection of acceptable patterns as part of the MATCH pattern. Each possible choice is separated by a comma (,). For example:

MATCH A,AB,BCD,ddd	The characters "A", "AB", "BCD", or any three digits are acceptable.

RANGE

A range of acceptable characters for a single character position can be indicated with the colon (:). All characters within the range are acceptable. This acts as a shorthand for listing a series of single characters in ASCII sequence. For example:

MATCH C:J	This is equivalent to the pattern specified by MATCH C,D,E,F,G,H,I,J.
MATCH 1:7	Any of the digits 1, 2, 3, 4, 5, 6, or 7 is accepted by this pattern.
MATCH !a:f	Since "a" is a pattern character, it is preceded by an exclamation point. Other such characters within the range are implicitly preceded by this operator. Thus, the range is equivalent to specifying MATCH !a,!b,c,!d,e,f.

It is important to differentiate between a pattern range which is a range of single characters, and the range check described earlier. In a pattern check, MATCH 10:90 means the value must be a "1" followed by a digit between "0" and "9", followed by a "0". In a range check, IN 10:90 means the value must be in the range "10" through "90".

GROUPING AND OPTIONAL

You can group pattern specifications by enclosing the pattern in braces { }. Brackets [ ] make the enclosed pattern optional. Braces indicate data must correspond to at least one item in the group; brackets indicate any item in the group is optional. For example:

MATCH {A,AB,BCD} ddd	One of the choices within braces must be matched. "A123" or "AB999" or "BCD562", among others, are acceptable matches for this pattern.

MATCH [A,AB,BCD]ddd	All choices within brackets can be omitted, or one may be matched. For example: "AB345", "BCD567", "A441", or "123" are all acceptable matches.
MATCH [u,d] ! +[1:5]	Some acceptable values are "A+" or "5+3" or "+5" or simply "+".
MATCH [B, dd]dd[%,d]	Accepts such values as "B12" or "12345" or "50%" or "10", among others.

Since blanks may be included within braces { }, you can put blanks in a pattern to enhance its clarity by enclosing the entire pattern within braces. For example:

MATCH {[B, dd] dd [%, d]}	Identical to preceding pattern except that it is enclosed within braces so that blanks can be included.

Enclosing a pattern within braces also allows the pattern to span lines. For example:

```
MATCH { [B, dd]          Identical to the example above, except that each pattern
        dd               component is listed on a separate line.
        [%, d] }
```

REPETITION

Repetition of any character or sets of characters can be indicated by an asterisk (*) or by a plus sign (+) following any pattern character or pattern group within braces { }. Plus (+) means that at least one occurrence of the pattern is required for the match; the asterisk (*) means that zero or more occurrences can be matched. These repetition indicators cannot follow items enclosed within brackets [ ]. Some examples:

MATCH d+	The plus sign indicates repetition of the digit, with at least one occurrence required for the match. Thus "2" or "7654321" or "55" are acceptable, but leaving it blank is not. (However, if the field type is O for optional, a blank is accepted because processing specifications are ignored.)
MATCH Xd+	This pattern accepts the letter X followed by one or more digits. "X1" or "X2345", and so forth are acceptable, but not "X".
MATCH M{A,C,d} +	A plus sign after a pair of braces indicates repetition of any item within the braces, in any order. Some acceptable values are "MA", "MCCC", or "M123CAA9".

MATCH d*	The asterisk indicates "optional" repetition that allows zero or more occurrences of the pattern. Thus, the digit can be omitted, or repeated any number of times. Nothing, or "3" or "123456", and so forth are acceptable matches.
MATCH [d+]	This pattern is another way of expressing the pattern shown above as d*.
MATCH a*	Accepts any alphabetic or empty string.
MATCH Xu*	This pattern accepts "X" alone, or followed by any number of upper case letters. For example, "XABC" or "XX" or "X" are all acceptable.
MATCH M{A,C,d}*	Any of the enclosed characters can be repeated in any order, or can be omitted. Thus, "M" is acceptable, as are "MAA", "MCCAC12", "MA63CCA5", and so forth.

OPERATOR HIERARCHY

The pattern operators are evaluated in the following order, where x and y are any patterns:

Highest	!x	Transparency
↓	x:y	Range
	x+ or x*	Repetition
	xy	Concatenation
Lowest	x,y	Choice

EXAMPLES

Some further examples of the MATCH statement are:

MATCH 1dddd	Accepts an integer between 10000 and 19999. Could also be expressed as: IN 10000:19999.
MATCH [d]d!:dd[AM,PM]	Accepts a time such as "3:00PM" or "12:00".
MATCH {1:9,1 {0:2}}!: { 0:5 } db*[{A,P}[M]]	12-hour clock time.
MATCH {1:7}{0:7} *	Accepts an octal number greater than zero with at least one digit and no leading zeros, such as "2047", or "1", or "24".
MATCH ddd-dd-dddd	Accepts any Social Security number, such as "044-24-0474".
MATCH [(ddd)] b*ddd-dddd	Accepts a phone number with an optional area code.

Table 4-3 summarizes the operators allowed in a MATCH pattern.

Table 4-3. MATCH Pattern Operators

Operator	Function	Example
!	Transparency operator allows use of any special MATCH characters as an element in the pattern.	MATCH !u,!d,!,,!! accepts any of the values "u", "d", ",", or "!".
,	Choice of subpatterns any one of which satisfies the match.	MATCH A,B,dd accepts values such as "A", "B", and "22".
:	Range of single characters in ascending ASCII order, any one of which satisfies the match.	MATCH 2:6 accepts only the values "2", "3", "4", "5", or "6".
{ }	Grouping (required) requires 1 occurrence of any pattern within braces.	MATCH {A,B} dd {%,d} accepts "A223", "B34%", "A795", and so forth.
[ ]	Grouping (optional) allows zero or 1 occurrence of any item in pattern within brackets [ ].	MATCH [A,B] dd[%,d] accepts "24", "A99", "10%", "123", and so forth.
+	Repetition (required) requires one or more occurrence of a preceding item, or a pattern within braces { }.	MATCH Xd+ accepts values such as "X1", "X22", "X3334789", and so forth, but not "X". MATCH {d,a} + accepts values such as "11", "A23", "acb", "33ABC9".
*	Repetition (optional) allows zero or more occurrences of a preceding item or a pattern within braces { }.	MATCH Xd* accepts values such as "X", "X1", "X22", and "X3334789". MATCH {d,a} * accepts a null value, or such values as "11", "A23", "acb", or "33ABC9".

ADVANCED PROCESSING

The advanced processing discussed in the rest of this section includes:

- Data movement
- Data formatting
- Altering forms sequence
- Conditional processing
- Processing phases
- Form Families

For advanced processing the sequence of specifications must be considered. If you need only the field edits described so far, you need not be overly concerned with the order in which statements are specified. Advanced processing statements, on the other hand, can be thought of as elements of a language where the order in which they are entered is important.

You may want to read the paragraphs on phases at the end of the section before reading the statement descriptions. In any case, you should be aware that all field processing statements can be executed in three phases: initialization, field edits, and final forms processing. Phases allow all initial processing for the form to precede all field edits, which in turn can precede all final processing.

DATA MOVEMENT

Data movement falls into two basic categories: setting the current field to a value, and moving data between fields. Each of these categories uses the SET statement.

SET STATEMENT

The SET statement format is:

$$\text{SET}\quad\left\{\begin{array}{l}\text{destination TO source}\\ \text{destination}\\ \text{TO source}\end{array}\right\}$$

where:

source	=	field save field constant arithmetic expression index retrieve
destination	=	field save field

When data is moved between constants, fields, and save fields, certain restrictions apply and certain conversions may take place. These depend entirely upon the data types of the source and destination. In general, any field, save field, or constant can be converted to a character type field; but numeric and date fields accept only data of a similar type. DIG fields accept only positive sources. If a source is $EMPTY, the destination is set to all blanks.

Table 4-4 under Data Formatting shows the conversion that is performed when data is moved between fields.

ASSIGNING VALUE TO CURRENT FIELD

To set the current field to the value of another field, a save field, a constant, an arithmetic expression, or a value in a list located through an index, use the following SET statement:

```
SET TO source
```

where:

source	=	field
		save field
		constant
		arithmetic expression
		index retrieve

In general, any source value can be moved to a character type field. Numeric and date fields accept only data of similar type. If the field type is DIG, the source must be a positive value. (See table 4-4).

By default, all fields are initialized to blanks in the initialization phase. (See phase description at end of this section.) You can specify a particular initial value for any field by including an initial value in the FIELD menu required specifications. More elaborate initialization can be done with this subset of the SET statement.

When you assign an initial value to a field in the FIELD menu, you can specify only a constant. The constant is entered exactly like operator input at a terminal. When you assign a value through the SET statement, you have more leeway. The values assigned are dynamic in that they may depend on values in other fields or in save fields, or they may be derived from an arithmetic expression, or through an indexed retrieval. If you do assign a constant through the SET statement, it must follow the rules for constants described earlier in this section. That is, a character string must be surrounded by quotes, a date string by exclamation points. Also, a date constant must be in the order MDY regardless of its destination format.

Some examples using the SET TO source statement follow.

1. Assume a date field of the form MDY.

SET TO $TODAY	Sets the field to today's date in the format dd/dd/dd in the order of the field's date type.
SET TO !FEB 10, 1978!	Sets the field to the specified date.
SET TO DAT1	Sets the field to whatever value is in the field DAT1 at run time.

2. Assume the current field is type DIG. The following statement sets this field to a digit selected from a list of digits by the index value, COUNT:

   SET TO COUNT OF 7, 9, 16, 24, 31, 72, 15, 12

   If COUNT=5, the value assigned to the field is 31, if COUNT=3, the value 16 is assigned, and so forth.

3. Values may be passed from one form to another through save fields. Assume that when FORMA is executed, SF3 is set to the value of F1. Further assume you are designing FORMB and want to set the current field to the value of the field F1 in FORMA;

SET TO SF3	Value passed from a field in a different form through the save field SF3.

MOVING DATA BETWEEN FIELDS

To move data to a field or a save field from another field or save field, or to move a constant, an arithmetic expression, or a value retrieved from a list to a particular field, use the following versions of the SET statement:

```
SET     { destination              }
        { destination TO source    }
```

where:

destination	=	field save field
source	=	field save field constant arithmetic expression index retrieve

When a source is not included, whatever value is in the current field is moved to the specified destination. (The current field is the field in which the SET statement appears.)

The following examples illustrate movement between fields:

1. Move the value resulting from an arithmetic expression to a numeric data field AMOUNT:

   ```
   SET AMOUNT TO 6 % (3 * COST)
   ```

   This statement multiplies the value of the field COST by 3 and then sets AMOUNT to 6 percent of the result.

2. Set the save field SF3 to the current value of the field in which the SET statement appears:

   ```
   SET SF3
   ```

   Assume the current field is a character type with the value "SMITH". The SET STATEMENT moves the value "SMITH" to the save field SF3. SF3 must be a character type save field.

DATA FORMATTING

When data is moved between fields, certain automatic formatting is performed. If the automatic formatting is not exactly what you want, or if you want to display data in a different format, you can specify formatting statements to apply to the current field.

You may also use the Reformatting Capability (described in section V) to reformat the data in the batch file for subsequent use by an application program.

AUTOMATIC FORMATTING

In general, automatic formatting performed during data movement is governed by the data type of the destination. The following discussion illustrates data movement for various data types. Refer to table 4-4 for a summary of the conversion performed during data movement.

CHARACTER TYPE

If the destination is a character field, data moved to it is not shifted. If the source is too large for the destination, the data is truncated on the right when it is moved. If the source has fewer characters, the destination is padded with blanks.

For example:

Source		Destination	
"ARMSTRONG"	(9 characters)	"ARMSTRONG△"	(10 characters)
"△△ARMSTRONG"	(11 characters)	"△△ARMSTRON"	(10 characters)
"ARMSTRONG△△△△△"	(14 characters)	"ARMSTRONG"	(9 characters)

NUMERIC TYPE

When data is moved between numeric fields, the following formatting is performed:

Sign	Any plus sign is stripped from the source before the number is moved to its destination. If the source is negative, a minus sign is inserted to the left of the first digit in the destination.
Decimal Point	If the source has an implied or actual decimal point (IMPn or NUM[n] data type), the fractional part is rounded and/or truncated or zero filled to conform to the number of decimal places specified for the destination.
	If the destination has no decimal position (NUM0,IMP0, or DIG), any fractional part is rounded and/or truncated.
	If the destination is NUM (floating decimal point), the number is right justified and stripped of zeros after the decimal point.
	If the source specifies no decimal places (type is NUM0, IMP0, or DIG) and if the destination has an implied or actual decimal point, the fractional part is zero filled.
	Note that if the length of the destination is too small, any decimal places are rounded and truncated until the source fits.

Commas	All commas in the source are removed.
Leading Zeros	Leading zeros are stripped in all cases.

The result is then placed, right justified in the destination field.

For example

Source		Destination	
123	(DIG)	ΔΔ123	(DIGIT, length is 5)
123	(IMP2)	1.23	(NUM2, length is 4)
12.3	(NUM1)	ΔΔ12.3	(NUM, length is 6)
12.3	(NUM1)	12.30	(NUM2, length is 5)
12.3	(NUM1)	12.	(NUM2, length is 3)
12.3	(NUM1)	1230	(IMP2, length is 4)
12.3	(NUM1)	123	(IMP1, length is 3)
+3357	(NUM)	335700	(IMP2, length is 6)
–3357	(IMP3)	–3.4	(NUM1, length is 4)
001,000	(NUM)	1000.00	(NUM2, length is 7)

DATE TYPE

Any date, regardless of the format of the source, is moved to the destination as dd/dd/dd. The order depends on whether the destination is specified as MDY, DMY, or YMD.

Source	Destination	
FEB 5, 1978	02/05/78	(defined as MDY)
2/5/78	78/02/05	(defined as YMD)
February 5, 1978	05/02/78	(defined as DMY)
September 16, 1978	09/16/78	(MDY)
October 23, 1978	23/10/78	(DMY)

Table 4-4. Conversion During Data Movement

From	To					
	CHAR	**NUM**	**NUM n**	**IMPn**	**DIG**	**DATE**
CHAR	Truncate or Pad with blanks on right	illegal	illegal	illegal	illegal	illegal
NUM	Truncate or Pad with blanks on right	Right justify; Strip leading zeros; Pad leading blanks; try to fit 9 decimal places, rounding truncating, inserting a decimal point, as needed.	Right justify; Strip leading zeros; Pad leading blanks; Round or truncate fractions, insert decimal points as needed.	Right justify; Strip leading zeros even if fractional; Pad leading blanks, remove any decimal point.	Right justify; Strip leading zeros; Pad leading blanks. (value must be positive)	illegal
NUM n						illegal
IMPn						illegal
DIG						illegal
DATE	Truncate or Pad with blanks on right.	illegal	illegal	illegal	illegal	Convert to dd/dd/dd in order of destination; Left justify; Pad trailing blanks.

FORMATTING SPECIFICATIONS

When the position or form of operator entered data is not correct for the applications that handle the data, you can use the REFORMAT facilities described in section V to adjust the data. In some cases, it is more desirable to reformat data as it is being collected. In such cases, you can use the FORMSPEC formatting statements to reformat the current data field during data collection. These statements are: STRIP, JUSTIFY, FILL, UPSHIFT, as well as the default formatting performed by the SET statement.

The formatting statements have immediate effect on the data in the field. When the operator enters data into a field for which formatting is specified, the data is reformatted as soon as ENTER is pressed. Thereafter, the field is displayed in its reformatted form. Thus, FORMSPEC formatting is done before data is written to the batch file. Reformatting through the REFORMAT program is done after data is collected, edited, and written to the batch file.

DEFAULT FORMATTING

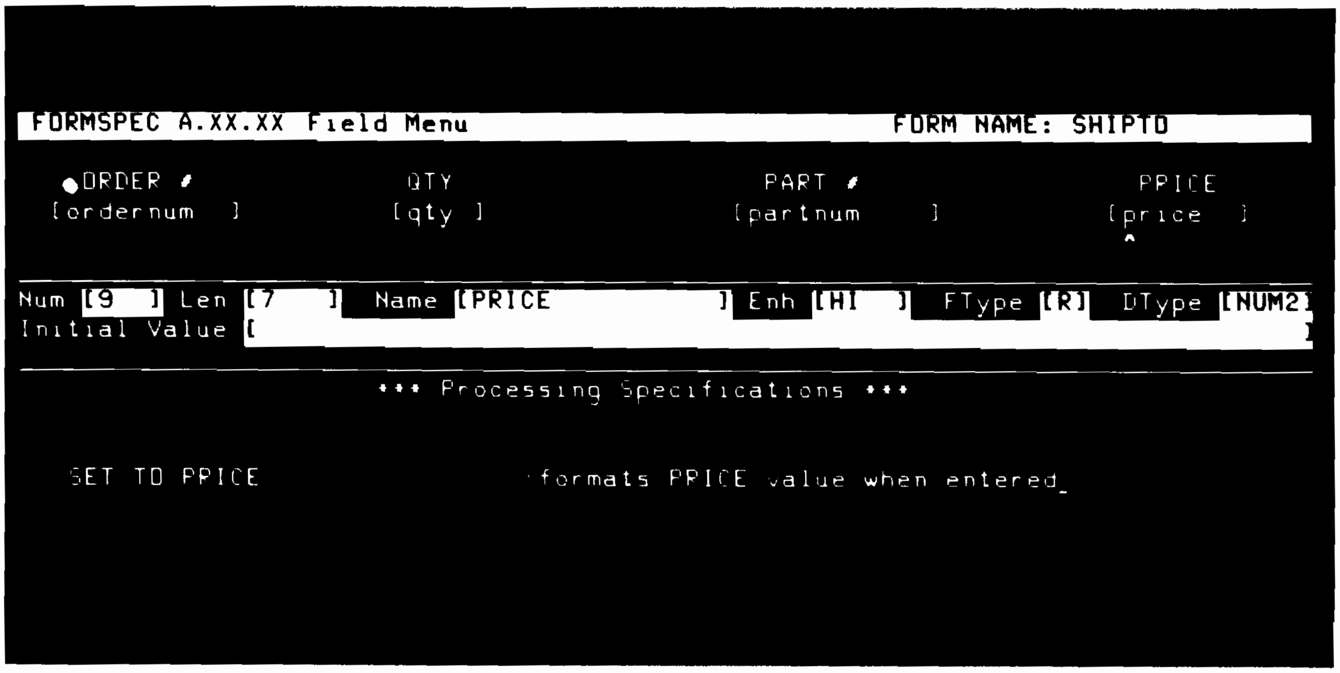
Besides the explicit formatting described below, data is reformatted whenever it is moved between fields according to the rules for automatic reformatting. If you want data entered by the operator to be formatted automatically, you can specify the following version of the SET statement.

```
SET TO thisfield
```

where:

thisfield	is the name of the field in which SET appears.

For example, you may want to insure that a monetary value is always right justified with a decimal point inserted preceding two decimal positions. To do this, define the field as type NUM2 and then use the SET statement to force data entered in this field to be formatted. The following FIELD menu illustrates this use of SET:



```
FORMSPEC A.XX.XX Field Menu                                   FORM NAME: SHIPTO

  ●ORDER #              QTY                    PART #                   PRICE
  [ordernum  ]          [qty ]                 [partnum     ]           [price   ]
                                                                         ^

Num [9  ] Len [7   ] Name [PRICE              ] Enh [HI  ] FType [R] DType [NUM2]
Initial Value [                                                                    ]

                        *** Processing Specifications ***

  SET TO PRICE                    \formats PRICE value when entered_
```

1. Suppose the operator enters a price in the field as follows:

   [123.5ΔΔΔΔ]

   The SET statement causes it to be formatted as:

   [ΔΔΔ123.50]

2. In another situation, this version of SET TO can be used to format a date. Suppose the date field, DATE1, is type MDY and the operator enters a date in the form:

   [FEB 12, 78]

   A SET statement of the form SET TO DATE1 in the FIELD menu describing DATE1 formats the operator entered data as:

   [02/12/78ΔΔ]

FORMATTING STATEMENTS

The following statements format data entered into or moved to any field. Note that only the characters between the leftmost non-blank and the rightmost non-blank characters are affected. That is, leading and trailing blanks are not included when the data is formatted.

STRIP

To delete any specified character in the field, use the STRIP statement.

The format of this statement is:

```
        ⎧ TRAILING ⎫
STRIP   ⎨ LEADING  ⎬      "characters"
        ⎩ ALL      ⎭
```

The specified "characters" are one or more ASCII characters within quotes. If TRAILING is specified, all occurrences of each character at the end of the field are replaced with blanks. STRIP LEADING replaces with blanks all occurrences of each character at the beginning of the field. STRIP ALL deletes all occurrences of each character, compressing the data to the left. Since the statement does not apply to leading or trailing blanks, the statements STRIP LEADING " " or STRIP TRAILING " " are not useful.

For example:

Statement	Data Entered	After Formatting
STRIP ALL " - "	[548-72-2002]	[548722002ΔΔ]
STRIP LEADING "0"	[Δ000205000]	[ΔΔΔΔ205000]

JUSTIFY

To move data within a field to the left or right boundary of the field, or center it within the field, use the JUSTIFY statement.

The format of this statement is:

```
JUSTIFY     { LEFT   }
            { RIGHT  }
            { CENTER }
```

Depending on the specification, the data in the current field is shifted to the left, to the right, or is centered in the field.

For example:

Statement	Data Entered	After Formatting
JUSTIFY LEFT	[ΔΔSMITHΔΔΔΔ]	[SMITHΔΔΔΔΔΔ]
JUSTIFY RIGHT	[ΔΔSMITHΔΔΔΔ]	[ΔΔΔΔΔΔSMITH]
JUSTIFY CENTER	[ΔΔSMITHΔΔΔΔ]	[ΔΔΔSMITHΔΔΔ]

If the data cannot be centered exactly, the extra blank is on the right.

FILL

To replace all the blanks between the field boundaries and the first or last non-blank data character with a designated character, use the FILL statement.

The format of this statement is:

```
FILL     { TRAILING }     "character"
         { LEADING  }
```

FILL TRAILING replaces those blanks following the data with the specified character; FILL LEADING replaces the blanks preceding the data with the specified character.

For example:

Statement	Data Entered	After Formatting
FILL TRAILING "*"	[Δ250ΔΔΔ]	[Δ250***]
JUSTIFY RIGHT FILL LEADING "0"	[Δ250ΔΔΔ]	[0000250]

Note that more than one formatting statement can be specified. Since the statements are executed in order of appearance, the FILL statement in the second example above affects the data justified by the preceding JUSTIFY statement.

UPSHIFT

To shift every lowercase character in a field to its uppercase equivalent, use the UPSHIFT statement.

The format of this statement is:

```
UPSHIFT
```

This statement causes all alphabetic characters to be replaced with their uppercase counterparts. If edit statements expect data to be uppercase, but it could be entered in lowercase, this statement should be executed prior to the edit statement.

For example:

Suppose the field contains a state code that is to be checked against a list of uppercase state codes. The following statements insure that codes entered in lower case pass the edit:

```
UPSHIFT
IN "NY", "NJ", "PA"
```

ALTERING FORM SEQUENCE

To change the specification of either the current form or the next form, use the CHANGE statement.

CHANGE STATEMENT

The format of this statement is:

```
                                                     ┌ "formname"    ┐
                                                     │ field name    │
                                 ┌ CLEAR          ┐  │ indexretrieve │
CHANGE NFORM TO             {    │ APPEND         │  │ $RETURN       │  }
                                 └ FREEZE APPEND  ┘  │ $HEAD         │
                                                     │ $END          │
                                                     └ $REFRESH      ┘

                              { NOREPEAT      }
CHANGE CFORM TO               { REPEAT        }
                              { REPEAT APPEND }
```

where:

NFORM	indicates the next form to be displayed after the current form at execution time.
CFORM	indicates the current form.
CLEAR	indicates that the screen is to be cleared when the next form is displayed.
APPEND	indicates that the next form is to be appended to the current form.
FREEZE APPEND	indicates that the current form is kept on the screen when the next form is appended to it even after the screen is full, at which point the top next form is rolled off the screen.
NOREPEAT	indicates that the current repeating form is to be stopped.
REPEAT	indicates that the current form is to be repeated.
REPEAT APPEND	indicates that the current form is to be repeated and appended to itself.
"formname"	is the name of any existing form in the forms file.
field name	is the name of a character-type field that contains a form name.
indexretrieve	is an item in a list of existing form names. It is specified as: index OF formname [,formname] . . . where index is an integer, and formname is a quoted form name or any field or save field whose value is a form name. The formname must identify an existing form.
$RETURN	indicates the last different form displayed before the current form at execution time.

$HEAD	indicates the first form displayed at execution time.
$END	indicates that the current form is the last form to be displayed.
$REFRESH	indicates that current form is to be refreshed (cleared of entered data) and displayed as the next form.

This statement may be entered in a processing specification for any field in the form. It causes the specified changes to the current or next form to take effect when the next form is requested, and it causes the specified next form to be displayed when the current form is completed at execution time.

If several NFORM statements are specified in a form, only the last statement executed is effective.

When forms sequence is defined in the FORM menu (refer to section III), the current form may be repeated, or repeated and appended to itself, or neither, when the next form is requested. Also, the current form may be cleared when the next form is requested, or it may remain on the screen with the next form appended to it. The CHANGE statement allows you to alter these form specifications dynamically.

Additionally, the CHANGE statement allows you to specify a different next form than the one specified on the FORM menu.

Note that the form changes do not occur when the field is entered with the ENTER statement, but only after the current form has been finished.

For example:

CHANGE NFORM TO CNT OF "FORM3", "FORM4", "FORM5", "FORM6", "FORM7"	Depending on the current value of field CNT, the next form displayed is one of the forms in the list. For instance, if CNT is 3, FORM5 is the next form.
CHANGE NFORM TO $END	After this form is finished, no more forms will be displayed.
CHANGE NFORM TO APPEND "FB"	FB, the next form, is to be appended to the current form.

CONDITIONAL PROCESSING

To execute any of the processing statements only under certain conditions, you can use the IF statement. Any of the field processing statements described so far can be executed conditionally depending on the run-time interpretation of a specified condition.

IF STATEMENT

The general syntax of the IF statement is:

```
    IF condition THEN [statement]
       [statement]
          •
          •
       [statement]
  ┌ ELSE [statement] ┐
  │    [statement]   │
  │       •          │
  │       •          │
  └    [statement]   ┘
```

An IF statement consists of two groups of statements: the THEN part and the ELSE part. Either may have no statements associated with it. The THEN part may include statements on the same line as THEN, plus statements indented from the IF on immediately following lines. An ELSE statement at the same level of indentation as the IF corresponds to that IF. Like the THEN part, the ELSE part can have statements on the same line, plus statements on immediately following lines that are indented from it. Nested IF statements must be indented from each enclosing IF, but otherwise follow the same rules

SYNTAX RULES

- No more than one IF or ELSE statement may appear on the same line.
- When non-IF statements follow either the THEN or the ELSE, they may be on the same line as the THEN or ELSE. Statements can be separated from each other by an optional semicolon (;).
- The entire ELSE portion of the statement may be omitted. In such a case, no statement is executed if the condition is false. If ELSE is included, it must be the first statement following the THEN part that is at the same level of indentation as its corresponding IF.
- Nested IF statements are allowed. They must maintain nested indenting.
- When nested IF statements are specified, they must be indented. The indenting is essential for multiple statements to identify the scope of the THEN part, as well as the ELSE part.
- An IF statement (including the THEN and ELSE part) must not cross phase boundaries. (Refer to the description of phases later in this section.)

The following formats illustrate some variations on the IF statement according to these rules:

IF condition THEN statement	Simple IF statement. If condition is true, the statement is executed; if false, the statement is not executed.
IF condition THEN statement statement statement	If condition is true, all three statements are executed in the order specified. If condition is false, none of the three statements is executed.
IF condition THEN statementA ELSE statementB	If condition is true, statementA is executed; otherwise, statementB is executed.
IF condition THEN statementA statementB statementC ELSE statementD	Statements A, B, and C are executed if the condition is true; statementD is executed if the condition is false.
IF condition THEN statementA statementB statementC ELSE statementD	If condition is true, statementA, statementB, and statementC are executed; otherwise statementD is executed.
IF condition1 THEN IF condition2 THEN statementA statementB	Statements A and B are executed only if conditions 1 and 2 are both true.
ELSE IF condition3 statementC	StatementC is executed only if conditions 1 and 3 are true, but condition2 is false.
statementD	statementD is executed only if condition1 is true, regardless of whether conditions 2 and 3 are true.
ELSE statementE	statementE is executed only if condition1 is false.

CONDITIONS

The format of a condition used in an IF statement is:

```
┌ constant        ┐
│ field           │
│ save field      │        edit statement
│ expression      │
└ (index retrieve)┘
```

A condition specified in an IF statement can be any edit statement previously described.

If the data in the field passes the editing specified by the statement, then the condition is true. If it does not, then the condition is false. Note that this differs from interpretation of a stand alone edit statement which causes the field to return an error and stops field processing if the data fails the edit.

Only edit statements can be used in conditions.

The edit statement can be preceded by a constant, a field name, a save field name, an expression, or an index retrieve operand within parentheses. If an operand precedes the edit statement, that value is tested; otherwise, the value of the current field is tested.

Some examples are:

IF QUANTITY GT 100 THEN . . .	If the current value of the field QUANTITY is greater than 100, any statements in the THEN part at the same level are executed.
IF NE $EMPTY THEN . . .	If there is a value in the current field (it is not blank), then any statements in the THEN part are executed.
IF SAV1 IN 12:50,100:120 THEN	If the value of the save field is within the range 12 through 50 or 100 through 120, then any statements associated with THEN are executed.
IF MINLEN 1 THEN	If at least one character was entered in the current field, execute any statements associated with THEN.

CONTROL STATEMENTS

Two statements provide control over forms processing. One forces a failure of the current field edit, the other stops all processing of the current phase of the current form.

FAIL STATEMENT

To force failure of a field edit, use the FAIL statement. The format of this statement is:

```
FAIL [“message”]
```

When this statement is executed, it forces data entered in the current field to fail any specified edits. When FAIL is executed, an error flag is set for the field. If you include the message parameter, that message is issued; otherwise, a system message is issued.

A FAIL statement is normally used in an IF statement where it is executed conditionally.

For example, suppose you want to insure that an entered value is in a table of values. You can use the FAIL statement to insure that no further edits are performed and an error message issued if the value is not found:

```
IF NIN $STATE THEN
   SET TO $EMPTY
   FAIL “Must enter legitimate state code”
```

If the value is found, the FAIL statement is not performed.

STOP STATEMENT

To stop processing the current phase of the current form, use the STOP statement. The format of this statement is:

```
STOP
```

When STOP is executed, no further processing is performed on the current field, or on any subsequent fields in the form.

For example, if you want to terminate processing of the entire forms file when “END” is entered in the current field:

```
IF EQ “END” THEN
   CHANGE NFORM TO $END; STOP       These statements terminate execution of the forms
                                    file if “END” is the value of the current field.
```

PHASES

Processing statements may be associated with one of four phrases of form execution:

• Configuration	Configure the terminal for specific field.
• Form initialization	Determine initial values of fields.
• Field edits	Edit and validate data entered in field.
• Finish form	Complete processing of form.

It is important to note that while specifications are entered field by field, execution of the phases applies to the entire form. During the initialization phase, all fields in the form are initialized; and during the field edit phase, all fields (in screen order: left to right, top to bottom) are validated and edited. The finish phase usually is performed only when all fields have passed the edit tests of the field phase. Finish statements apply to the entire form. (Refer to figure 4-4 for an illustration of the flow of form execution under ENTRY program control.)

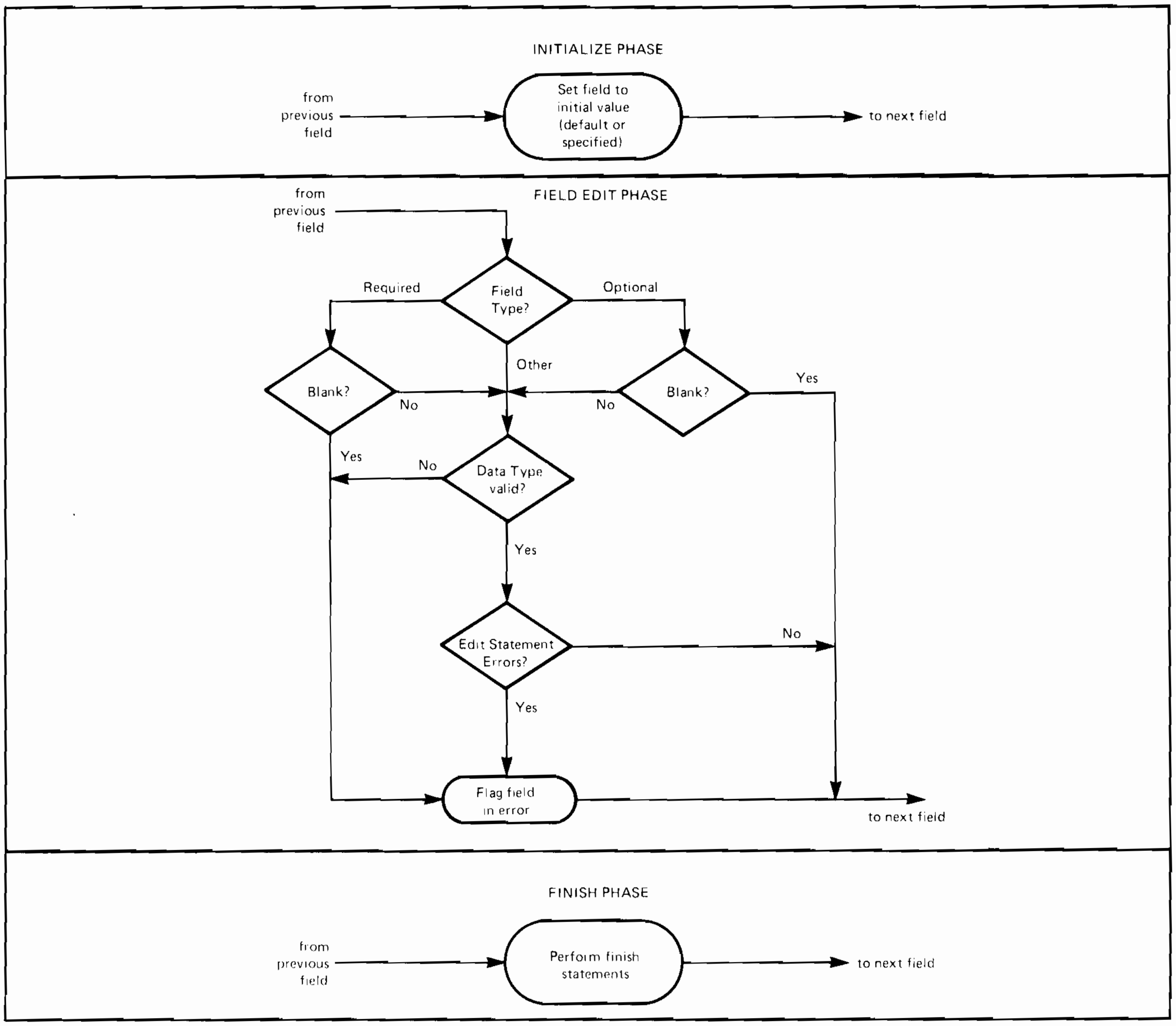


Figure 4-4. Flow of Phase Execution

You can specify that a statement be executed during a particular phase with the phase specification statements described below. If you do not specify a phase, all statements are executed in the field edits phase.

CONFIGURATION PHASE

If you use the configuration phase, the CONFIG statement must be the first statement in the processing specification area on the field menu. The CONFIG statement is processed only for the field in which it appears.

Three new commands can be used in the configuration phase: DEVICE, LIGHT, and LOCALEDITS. (See the Command area for more detail.)

INITIALIZATION PHASE

The processing that precedes the display of the form is usually performed in this phase.

For each field in the form, the field is first initialized to any value specified in the Initial Values box of the FIELD menu. (By default, all fields are set to $EMPTY, all blanks). If any initialization statements were included in the field processing specifications, these are executed next. Then, the next field is initialized, and so forth, until all fields in the form are initialized.

A SET TO source statement may be used to initialize field values.

FIELD EDIT PHASE

The field edit phase is usually performed after the form is displayed on the screen, and the operator has typed in data and pressed ENTER. During this phase, the entered data is checked according to the required field description and any field processing specifications entered by the designer.

Each field in turn is examined after ENTER is pressed. The first action is to check the field type. If it is optional and blank, the rest of the field edit phase for this field is skipped. If the field is requdired, it must contain a non-blank value. If it has a value, the data is checked to see if it conforms to the field's data type. If it does, then any field processing statements are executed in the order they were entered.

Edit failures detected in this phase cause the field to be flagged in error so the operator can be informed. All further processing on this field is stopped when an error is detected, and control passes to the next field.

The ENTRY program loops through all processing statements in the field edit phase until all errors have been corrected.

The Edit statements (table 4-2) can be executed *only* in the field Edit phase.

FINISH PHASE

The finish phase usually occurs after all data has been entered and validated for the entire form. The ENTRY program executes statements in this phase only when no errors were detected after ENTER was pressed. Finish statements direct irreversible processing during which the global environment can be altered.

Under ENTRY control, any processing performed by the preceding two phases can be undone prior to the FINISH phase. For instance, if the operator presses REFRESH, the initial values replace any that were entered. In the finish phase, such changes can no longer be done.

Under ENTRY control, the batch file is written as soon as the finish phase is complete. ENTRY assumes all fields have been tested and corrected; fields with errors are not enhanced in the finish phase.

PHASE SPECIFICATION STATEMENTS

If you want statements to be executed in a particular phase, you must precede these statements by one of the following phase headings in the order shown:

CONFIG	Perform the following configuration during form presentation.
INIT	Perform the following statements during initialization.
FIELD	Perform the following statements during the field edit phase.
FINISH	Perform the following statements during the finish phase.

For example:

```
CONFIG
LOCALEDITS alphabetic
LIGHT B
DEVICE printer

INIT
SET TO 20.00

FIELD
IN 20:200
IF IN 100:200 THEN JUSTIFY RIGHT

FINISH
SET SAVE—FIELD—1
```

If you omit these phase headings altogether, all statements are executed in the field edit phase.

"FINISH" must always be specified to indicate that a statement is to be executed in that phase. "FIELD" can be omitted if there are no initialize statements and the field statements are followed by the FINISH statements. "INIT" can be omitted if the initialize statements are followed by a FIELD statement.

Table 4-5 shows the cases in which a phase must be specifically assigned. Brackets surround the phase headings that can be omitted. The CONFIG phase must always be preceded by "CONFIG" before listing specifications.

Table 4-5. Phase Specification

Initialize				INIT statements	INIT statements	[INIT] statements	[INIT] statements
Field Edit		[FIELD] statements	[FIELD] statements			FIELD statements	FIELD statements
Finish	FINISH statements		FINISH statements		FINISH statements		FINISH statements

FORM FAMILIES

A family of forms is a collection of forms which share a common screen definition but may have different field characteristics, field processing specifications, or repeat and freeze options. Use of form families may improve V/3000 performance in some transaction processing applications; since only the internal form characteristics are different between members of a family, you do not have to wait for V/3000 to repaint the screen when you change to another form in the same family.

The figure below illustrates some form family concepts.

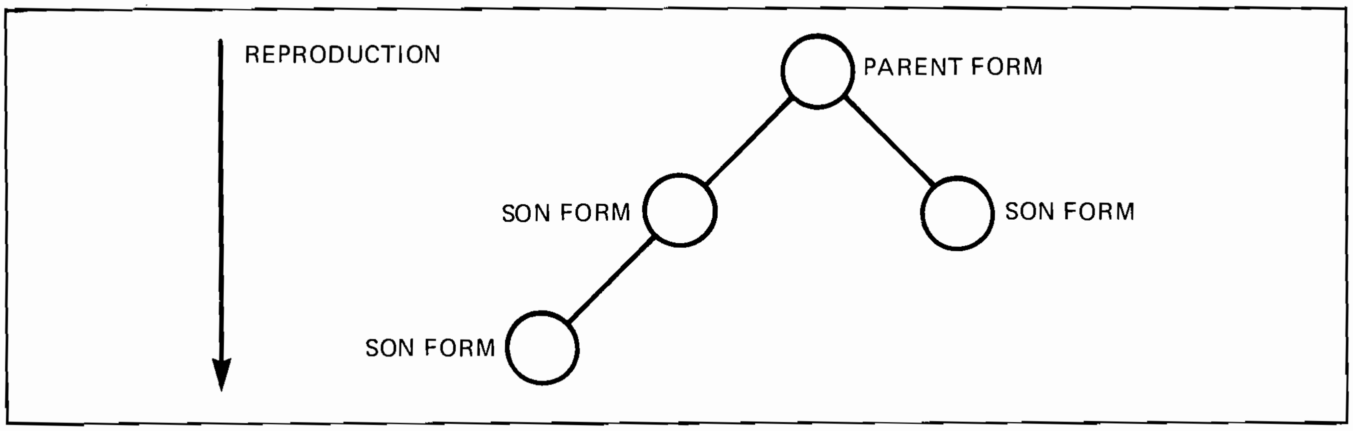


Figure 4-5. Form Family Relationships

Parent Form	Also called father or root form. A parent form is a standard V/3000 form created in FORMSPEC either by designing it or by copying it from another form. The screen design of the parent form determines the screen definition of all son forms that are reproduced from it. Changing the screen of the parent form causes the screens of the son forms to change.
Son Form	Reproduced from the parent form, the screen definition of the son form may not be modified. You may change repeat and next form characteristics, field characteristics and processing specifications. As the above figure indicates, you may reproduce a son form from another son form, although V/3000 still considers the original root form to be the parent of both forms.
Reproduce	To generate a related form by entering the name of an existing form in the "Reproduced from" field of the form menu.

CREATING SON FORMS

You create son forms by following these steps:

1. Select the "Add a form" option of the Main Menu.

2. When the Form Menu comes onto the screen, enter the name of the son form you wish to create, and, in the optional "Reproduced from" field, enter the name of the form from which you wish to generate the son form. You may change any of the other Form Menu fields.

3. When the screen menu appears, press NEXT (f6). Since the screen definition must be the same for parent and sons, even pressing ENTER will cause the error message "Cannot change screen, the form is a family member." to appear in the window.

4. You may change any unprotected values in the field menus to adjust the internal characteristics of the son form to your needs. A son form will continue to have all the field characteristics of the parent if you do not modify the son's field menus.

To modify the screen definition of a form family, you need to change only the screen of the parent form, and recompile the forms file. This will change the screens of the son forms automatically. You can delete an entire form family by merely deleting the parent form.

Careful naming of forms and the use of the optional comment field of the Form Menu will make relationships between forms of the same family more apparent.

To make it easier to distinguish between forms of the same family in ENTRY or in other application programs, you may wish to add a field reserved for the form name to the screen definition of the parent form, and add processing specifications which set the value of that field to the name of the current form.

FORM FAMILY EXAMPLE

Suppose you had a V/3000 application program which, after accepting a customer's identification number, would retrieve items of pertinent customer information from a data base and display them on a form. It would be convenient to have two forms, one on which the customer ID is required, and the other on which the data entry operator would view or alter the customer data. If the form family feature is not used, the operator who would be alternating between entering customer ID's and inspecting or modifying data, would constantly have to wait for the form to be repainted each time he or she needs information on another customer.

The solution to this problem of long operator waiting time is to create two forms with the same screen definition and different field types.

Suppose the root or parent form of this example is named CUSTPAR, and that its screen definition is as shown in the figure below. All of the fields in this form are display only, except for custid, which is required.

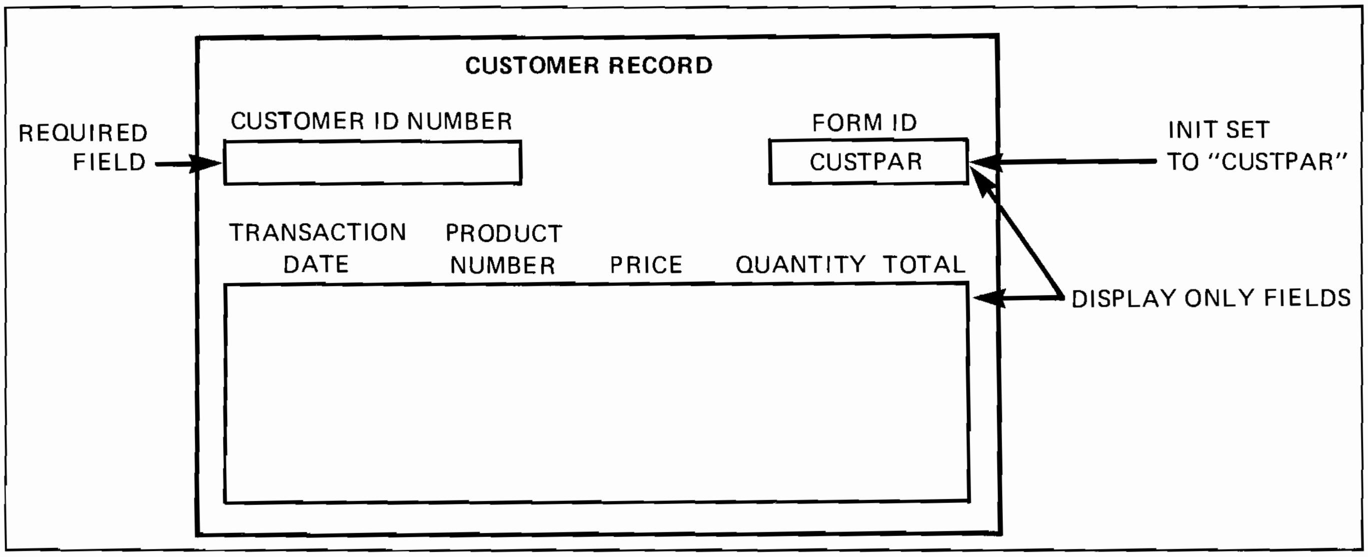


Figure 4-6a. Parent Form

Create a son form, CUSTSON, by entering "CUSTPAR" in the "Reproduced from" field of the form menu. Press NEXT (f6) until the field menu for custid appears and make it a display only field. Make all the data fields in the table optional. Compile the forms file.

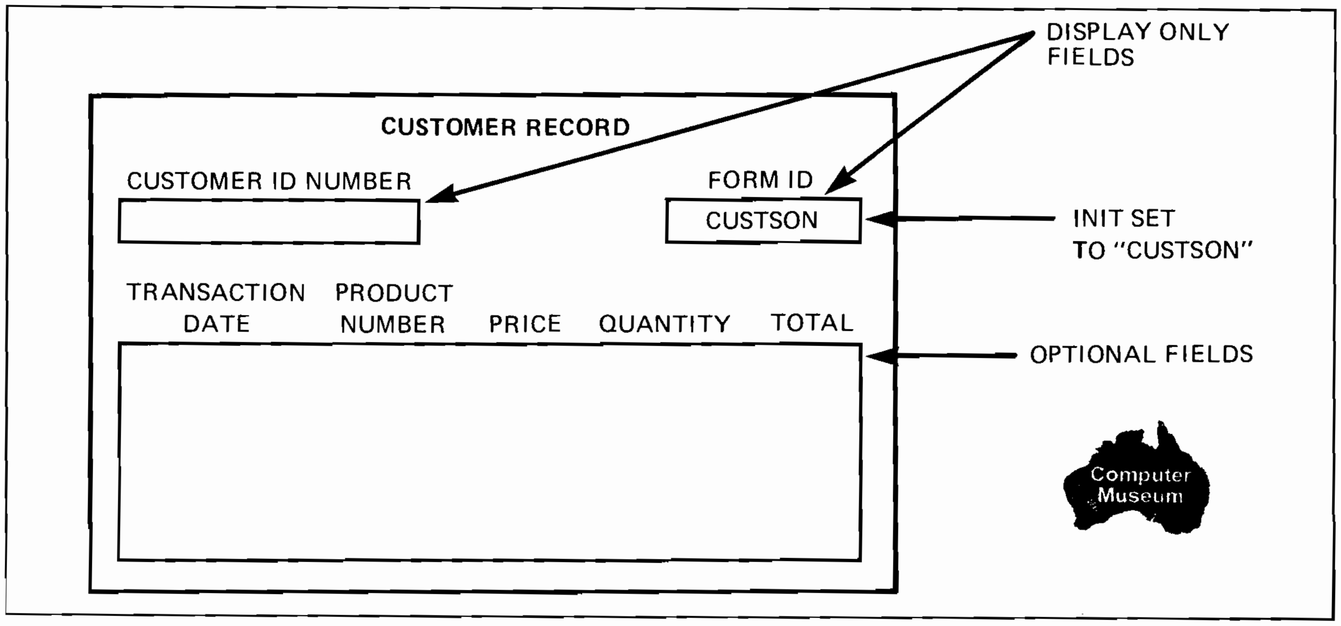


Figure 4-6b. Son Form

Now when the operator types the customer ID and presses ENTER, the application program will retrieve the related information from the data base and put it on the screen immediately. Once the form is on the screen, it will appear to remain there as long as the operator uses the application program, rather than being repainted one time for every customer about whom the operator calls up data.

REFORMATTING SPECIFICATIONS

SECTION V

OVERVIEW

Information gathered during data collection is stored in one or more "batch" files. The data entered for each field in a form is stored exactly as it was entered. All blanks, punctuation, or special characters are included in the data. The only exception is that any formatting specified on the FIELD menu or any automatic formatting resulting from data movement (refer to section IV) is completed before the data is written to the batch file. Thus, if a value has been entered in the middle of a field, the blanks on either side are stored as part of the data; or if a number has been right justified and zero filled, it is stored in the batch file in that form.

All data from all fields in a single form are concatenated together without delimiters into a single record in the batch file.

If the configuration of the data in a batch file is not suitable for input to an application program, the data can be reformatted with the REFORMAT program. This program writes the data from the batch file to another file, the "output" file, according to formatting specified through the reformat design program, REFSPEC. (Refer to figure 5-1.)

You may want to reformat data entered in the batch file in order to:

- combine data from several forms into one record in the output file;
- separate data from a single form into two or more output records;
- generate several output files from the data entered through a single forms file;
- add literals (such as sort codes or record separators) to each output record;
- use only a selected portion of the data entered in any field;
- write only selected fields from any form to an output record;
- fill, justify, or strip characters from data entered in any field;
- add a check digit to alphabetic or numeric data.

This section describes how to use the reformat design program REFSPEC. It is an extension of forms design and is used by the forms designer in conjuction with an application programmer to specify how data is to be reformatted. Use of the program REFORMAT is described at the end of this section.

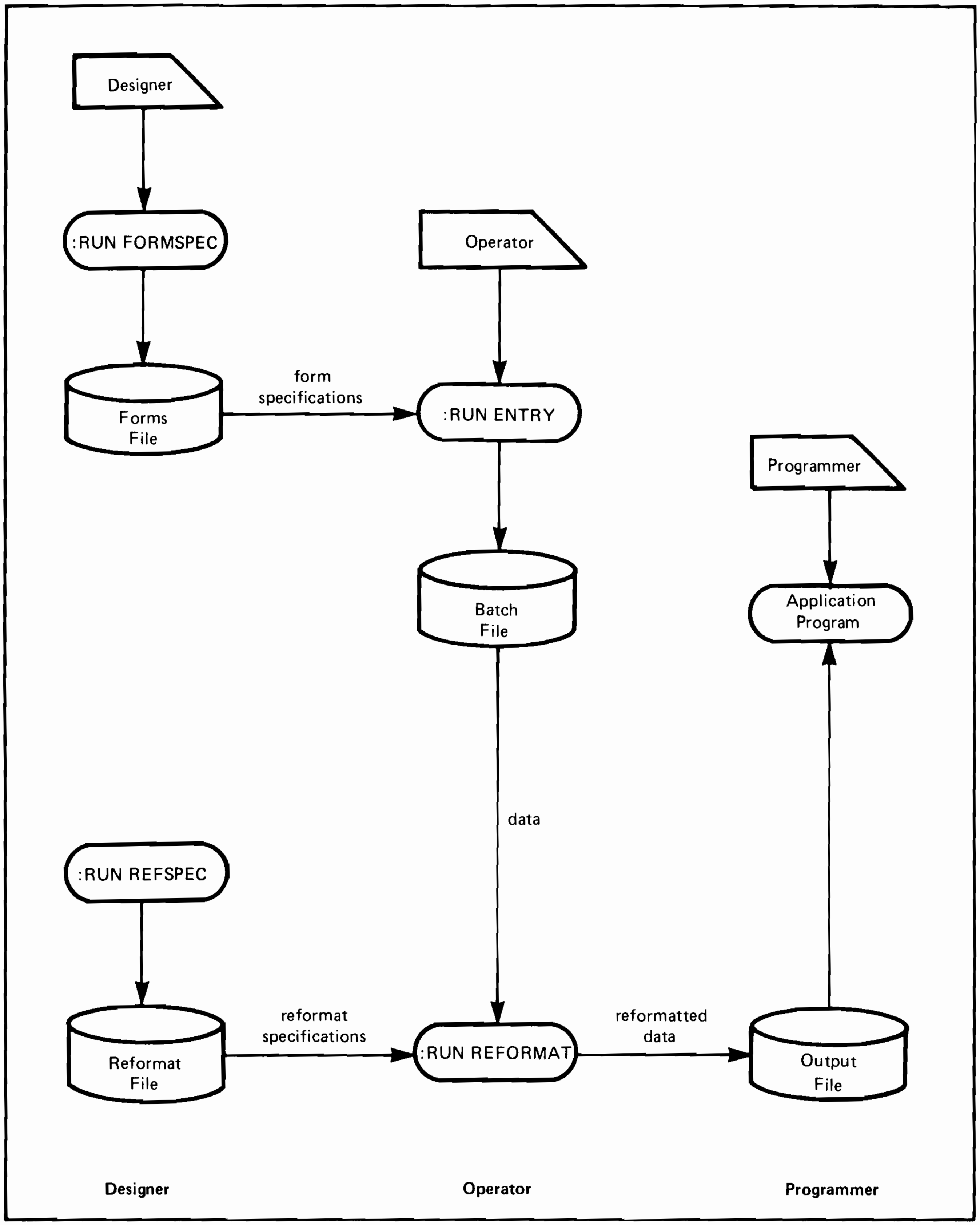


Figure 5-1. Relation Among Files Used for Reformatting

FILES

One batch file containing collected data can be reformatted into one output file. This output file is then used as input to the application program that processes the collected data. Data from more than one batch file can be written to one output file in sequence — that is, data from a batch file can be appended to data from another batch file in an existing output file with a :FILE command equation. Without a file equation, data from each batch file overwrites data in an existing output file.

The reformatting specifications themselves are stored in a "reformat" file. The specifications in this file determine how the data in the batch file is to be stored in the output file.

One output file is generated for each reformat file. If you wish to separate the data into more than one output file, you must establish different reformat files, and run the reformatter program separately, to generate each output file.

REFORMAT FILE

The reformat file consists of the following components:

• Forms File Name	Identifies the forms file that contains the forms through which data in the batch file was entered.
• Global Information	Specifications that apply to the entire output file, such as a string used for an end-of-record mark, or record length and whether the record length is fixed or variable. Any or all global specifications can be omitted since defaults are provided.
• Input Forms Sequence	Specify the form names on which data to be reformatted was entered. Only data from the forms in an input forms sequence will be written to the output file. At least one input forms sequence must be included, and as many may be included as there are forms in the forms file.
• Output Record Definition	Specify each field to be written to the output file, and define the beginning of each output record. Usually, one output record definition follows each input forms sequence.
• Field Specifications	Special reformatting at the field level can be specified for each field listed in the preceding output record definition.

This information is entered on menus very much like those issued for FORMSPEC. Figure 5-2 illustrates a prototype reformat file.

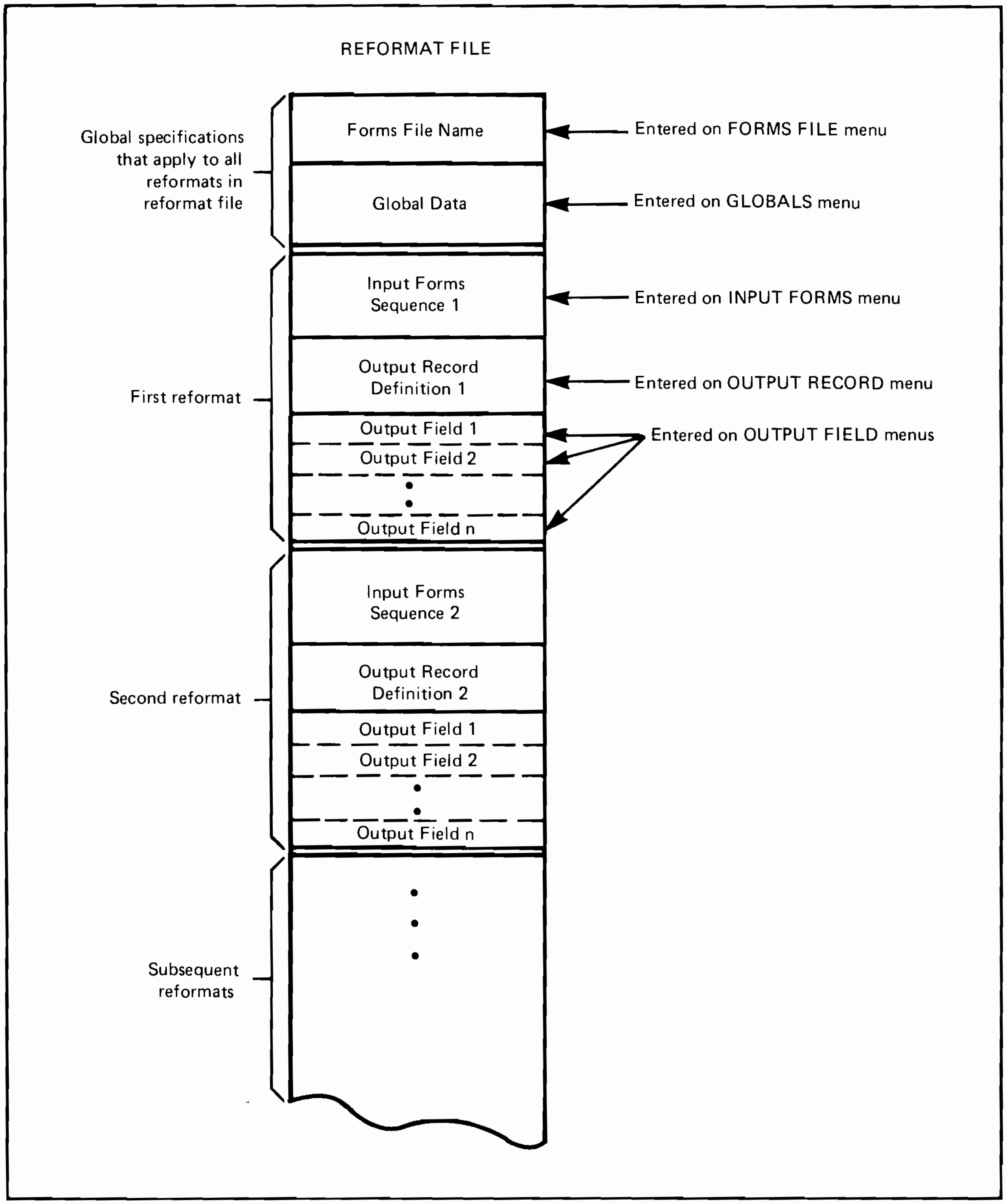


Figure 5-2. Prototype of Reformat File

RELATION OF FORMS TO OUTPUT RECORDS

Every record in a batch file consists of data entered on a single form. It may be that the data on one form represents a logical group of information. However, the data from a sequence of forms may make up such a logical grouping, or the data entered on one form may make several logical groupings. The reformatter allows you to rearrange the entered data into output records with different groupings.

Before running REFSPEC to set up your reformat file, it is important to understand the relations between the forms (and the data entered on the forms) and the output records generated by the reformatter. The forms to be reformatted are identified in the input forms sequences of the reformat file; the output records are defined in the associated output record definitions.

INPUT FORMS SEQUENCE

Each input forms sequence lists the form or forms on which the data to be reformatted was entered.

In order to generate the output file from the reformatting specifications, the REFORMAT program reads each batch record in turn. Associated with each record is the form on which the data was entered. For the first batch record, the REFORMAT program searches the input forms sequences in the reformat file for a matching "reformat identifier". This identifier is the name of the first form in each input forms sequence, and must be unique in the reformat file. If a matching reformat identifier is not found, the batch record is skipped.

Other form names may follow the reformat identifier in the same input forms sequence. These form names need not be unique. If a sequence of forms is specified, the batch records following the record that matches the reformat identifier must exactly match the form names in the input forms sequence. If they do not, the batch record is skipped, and the next batch record is checked against all reformat identifiers in the reformat file.

Some rules to remember:

- Every batch file record contains data entered on a single form.
- The form name on which the data was entered is stored in the batch record with the data.
- Records in the batch file are processed sequentially starting with the first record and continuing through to the end.
- The first form name in each input forms sequence is the reformat identifier, and must be unique.
- In order to be reformatted, the form name of a batch record must match one of the reformat identifiers or be in a sequence following a form that matches a reformat identifier.
- Form names in an input forms sequence following the reformat identifier can appear in other input forms sequences as reformat identifiers or as part of a sequence following the reformat identifier.
- If more than one form is named in an input forms sequence, the sequence of batch records starting with the reformat identifier must exactly match the sequence of forms.

OUTPUT RECORD DEFINITION

The output record definition determines how information from batch file records is stored in the output file. Any field in the batch file that is to be written to an output record must be uniquely identified. The sequence of the output field names in the output record definition determines the order in which fields are written to the output record. Fields from the batch file may thus be reordered, or repeated, or omitted from the output record.

Constants, in the form of literals, system constants, or the numeric equivalent of ASCII characters, can be interspersed freely between fields or portions of fields in the output record definition.

The output record definition also determines where each output record begins. The designer can mark one or more fields (or constants) as the starting point of a record. If no field is so marked in an output record definition, then the first field in the definition is appended to the last field in the previous definition as part of the same record. Thus a batch record can be divided into several output records, or be made part of a larger record.

Note that you can specify a fixed record length, or a maximum variable record length, for all output records through the GLOBALS menu. The REFSPEC program does not allow the designer to specify a record in any one output record definition that is longer than this maximum length, however, it does not check the length of a record formed from several output record definitions.

Some rules to remember:

- A unique output field name must be assigned to any batch field to be written to the output file. (Usually, the input field name is unique and can be used as the output field name.)
- Batch fields are written to the output file in the order of the output field names.
- Only those fields specified in an output record definition are written to the output file.
- A particular batch field can be specified many times in a single output record definition as long as each occurrence is given a unique output field name.
- A start-of-record marker in an output record definition determines the start of an output record.
- A start-of-record marker can be associated with any field or constant in an output record definition.
- If no start-of-record marker is included in an output record definition, the first field of this definition follows the last field of the preceding definition when it is written to the output file.

FIELD SPECIFICATIONS

An OUTPUT FIELD menu is issued for each field named in the preceding output record definition. The designer can reformat the output field, or leave the field as it was written to the batch file.

The following set of examples illustrates some ways in which the data from a batch file can be reorganized on an output file.

COMBINING DATA FROM SEVERAL FORMS

Suppose data is entered on forms A, B, and C. The reformat file shown in figure 5-3 processes these records in sequence and writes them to one output record.

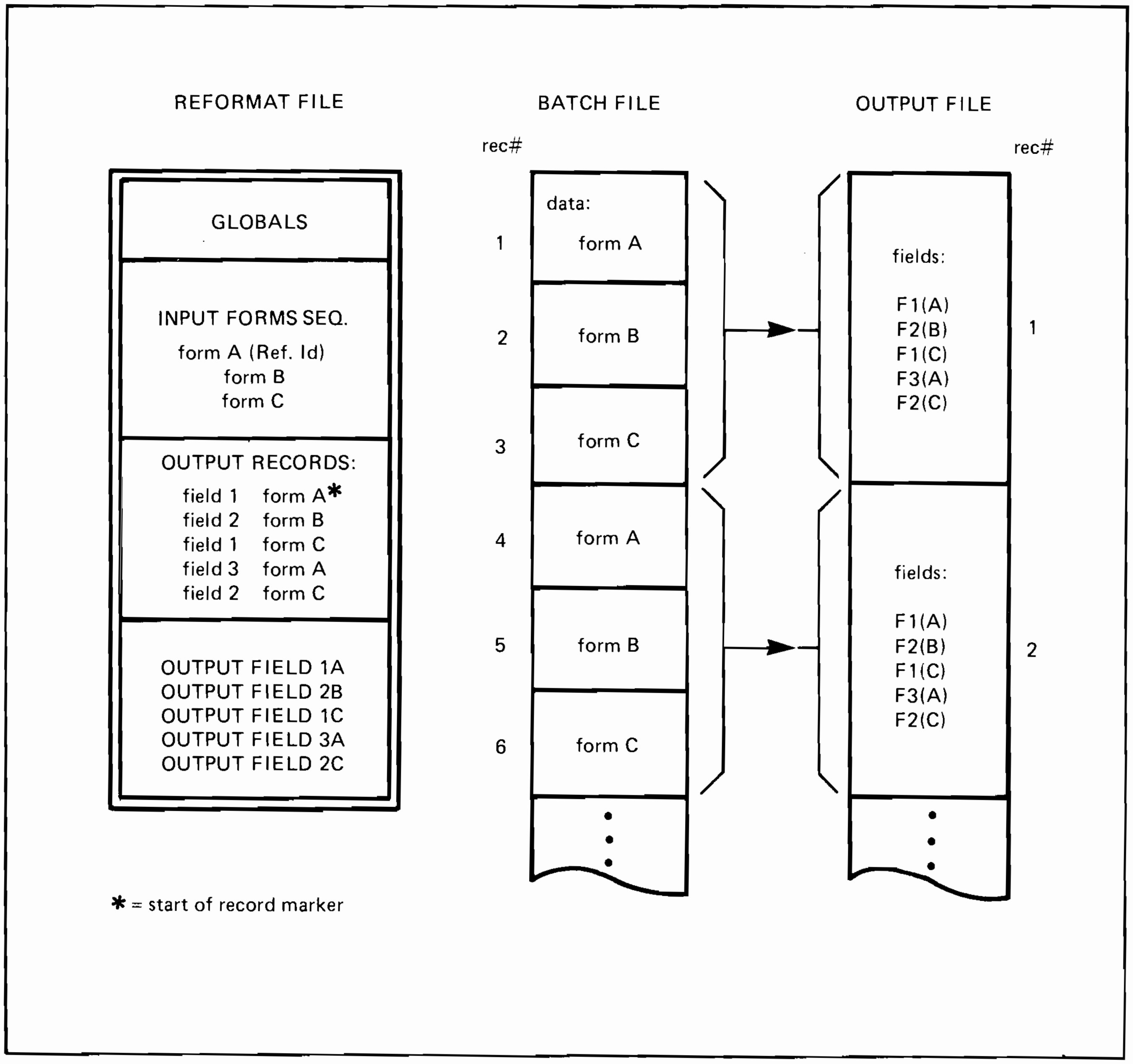


Figure 5-3. Combining Data From 3 Forms into 1 Output Record

Data from the batch file illustrated in figure 5-3 can be reformatted in many different ways. For example, figure 5-4 shows how this same data could be combined into two output records with data from forms A and B appearing in both records. Note that when the same field appears more than once, it must be given a unique output field name for each occurrence (this is not specifically shown in figure 5-4).

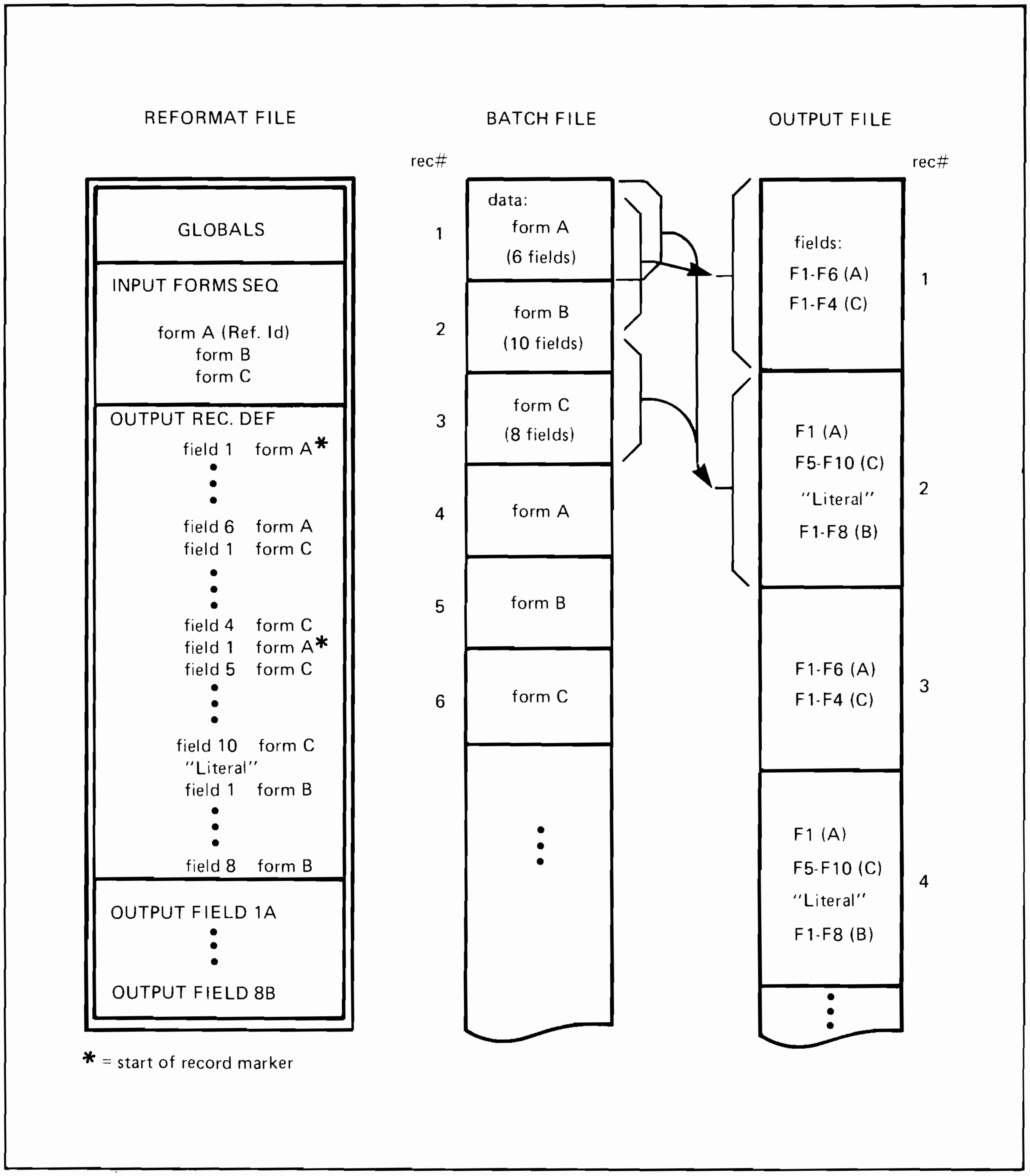


Figure 5-4. Reformat Data from 3 Forms into 2 Output Records

SEPARATING DATA INTO SEVERAL RECORDS

In the following example (figure 5-5), data entered on form A is separated into three output records, each a fixed length of 40 characters. Data from a subsequent form is written to a single 40 character record. If any group of data is less than the 40-character record length, the record is padded with blanks. The data cannot be formatted to exceed the record length.

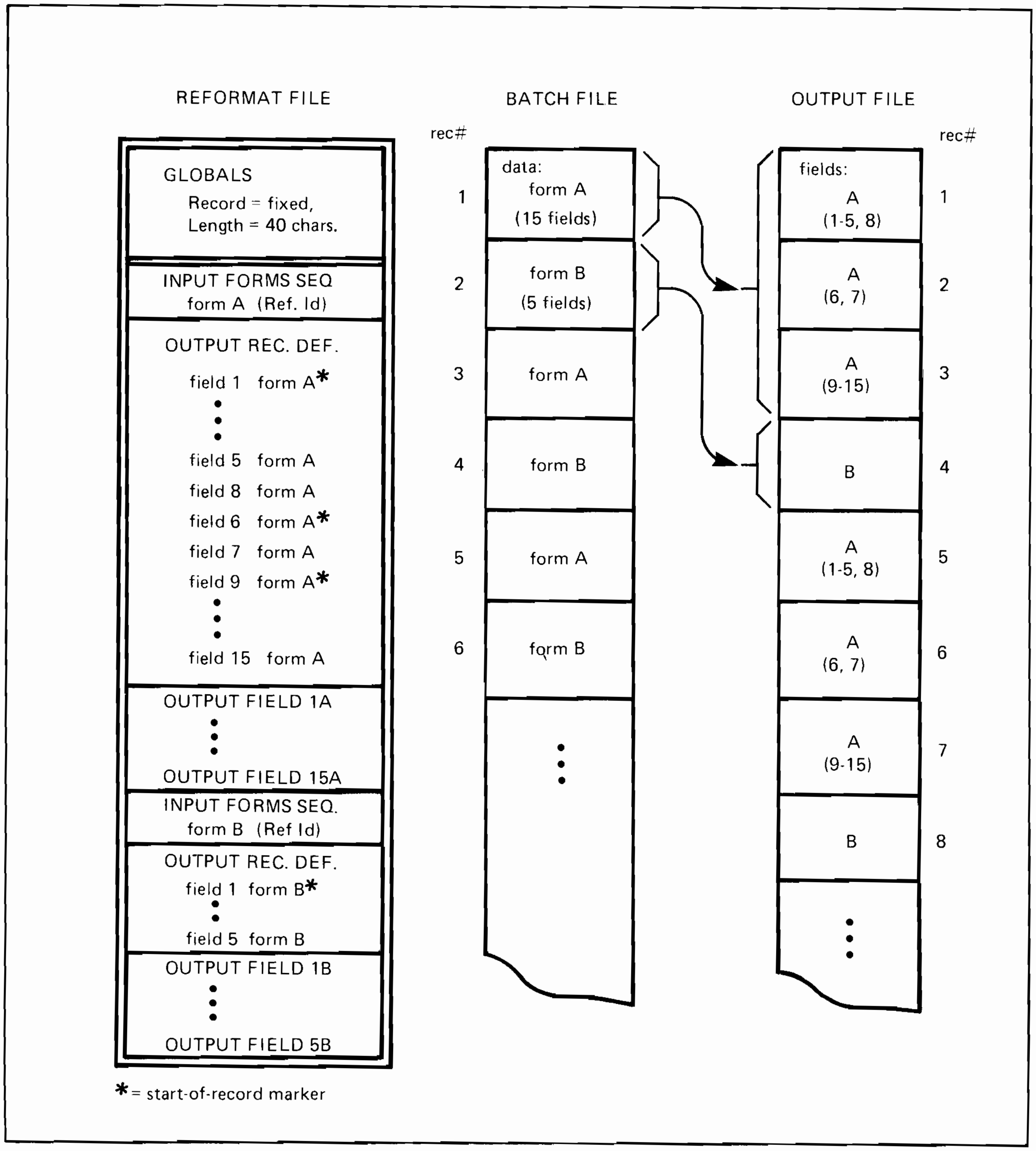


Figure 5-5. Separating Data from 1 Form into Several Records

REFORMATTING REPEATING FORMS

In figure 5-6, form B is a repeating form that may occur a variable number of times. The reformat file in this example causes data entered on forms A, B, and C to be written to a single variable length output record. Note that if repeating form B causes so much data to be written to the output record that the maximum record length is exceeded, data (possibly significant) is truncated when the REFORMAT program is run.

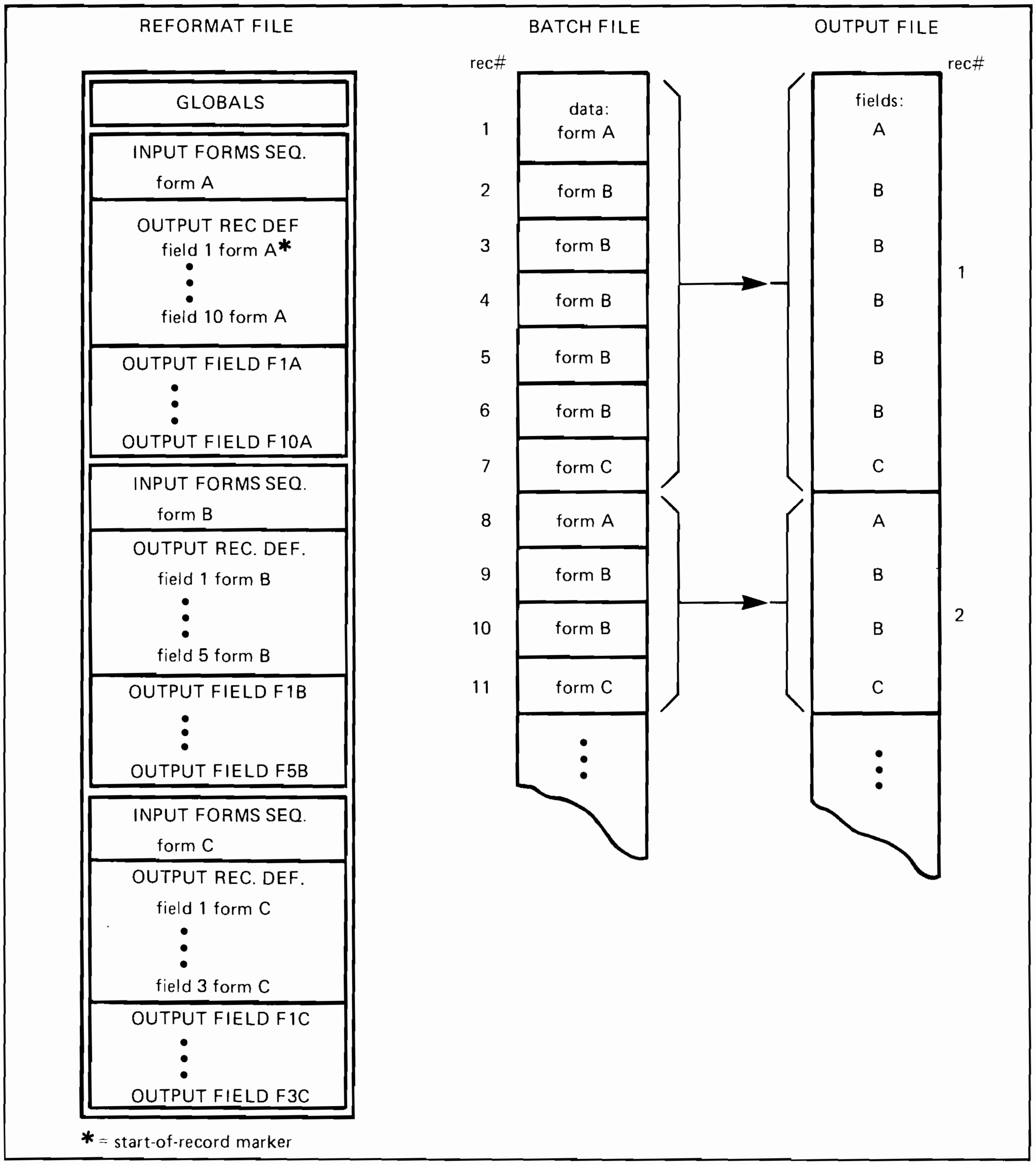

REFORMAT FILE
BATCH FILE
OUTPUT FILE
GLOBALS
INPUT FORMS SEQ. form A
OUTPUT REC DEF field 1 form A* field 10 form A
OUTPUT FIELD F1A OUTPUT FIELD F10A
INPUT FORMS SEQ. form B
OUTPUT REC. DEF. field 1 form B field 5 form B
OUTPUT FIELD F1B OUTPUT FIELD F5B
INPUT FORMS SEQ. form C
OUTPUT REC. DEF. field 1 form C field 3 form C
OUTPUT FIELD F1C OUTPUT FIELD F3C
* = start-of-record marker
rec#
1 data: form A
2 form B
3 form B
4 form B
5 form B
6 form B
7 form C
8 form A
9 form B
10 form B
11 form C
rec#
fields: A B B B B B C
1
A B B C
2

Figure 5-6. Reformatting Data from Repeating Forms

Suppose data entered on one form can follow data entered on either of two different forms, and you want to generate separate records depending on the sequence in which the forms appear. In such a case, you can set up a reformat file as shown in figure 5-7. Note that form B (a repeating form) appears as a reformat identifier and also as a succeeding form in other input forms sequences.

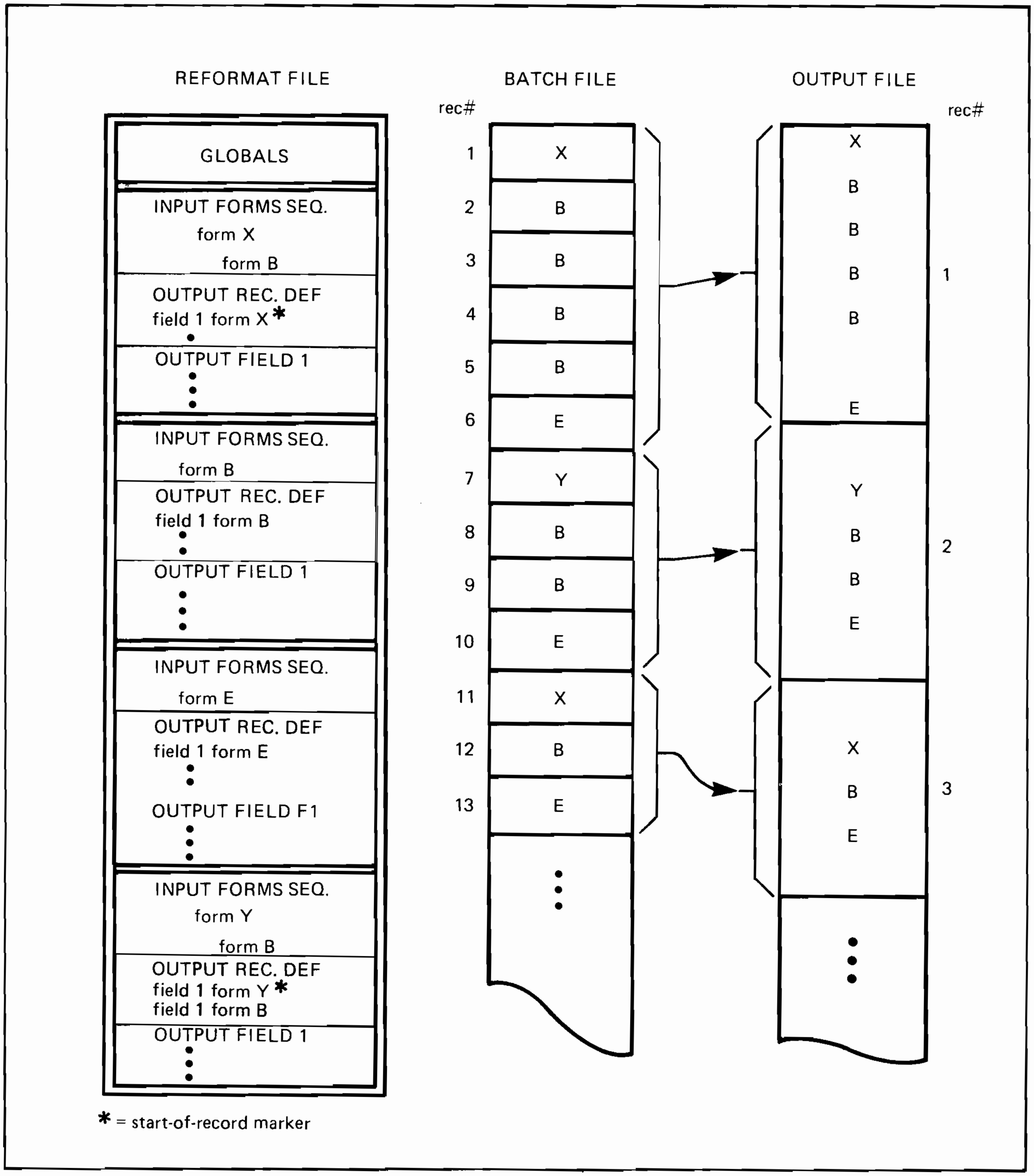


Figure 5-7. Reformatting Data Based on Form Sequence

SEPARATING DATA FROM ONE BATCH FILE INTO SEVERAL OUTPUT FILES

A separate reformat file must be established for each different output file. In figure 5-8, data entered on forms A and B is written as a single record to one output file. Data from forms A and C is written as two records to a second output file; data from form B is not written to output file 2.

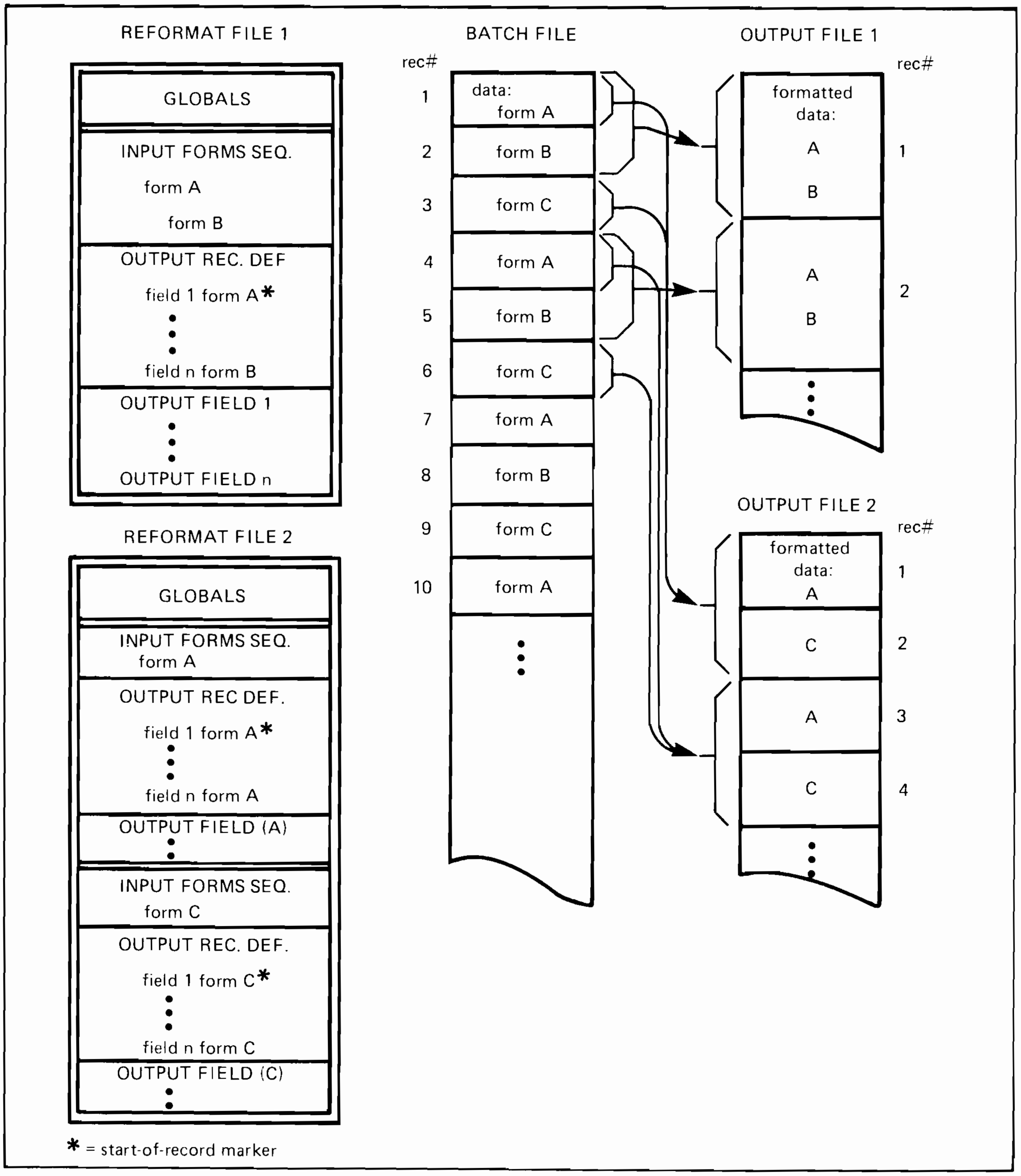


Figure 5-8. Generating 2 Output Files from 1 Batch File

USING REFSPEC

You execute program REFSPEC by entering the following command in response to the MPE colon prompt:

```
:RUN REFSPEC.PUB.SYS
```

Since REFSPEC runs entirely in block mode, you will be asked to press the BLOCK MODE key if your terminal is not placed in block mode automatically.

Like FORMSPEC, REFSPEC prompts for information on menu screens. The information you enter on these menus defines how data in the batch file is to be reformatted and written to an output file. The first menu issued whenever you run REFSPEC is the REFORMAT FILE menu.

If you specify a new reformat file, REFSPEC creates this file and then issues the FORMS FILE menu so you can specify the forms file through which the data to be reformatted was entered. If you specify an existing reformat file, it is already associated with a forms file so the FORMS FILE menu is not issued, and the MAIN menu is the next menu.

REFSPEC FUNCTION KEYS

A set of seven function keys, similar to those used for forms design, are used to control execution of REFSPEC. These keys are illustrated in figure 5-9.

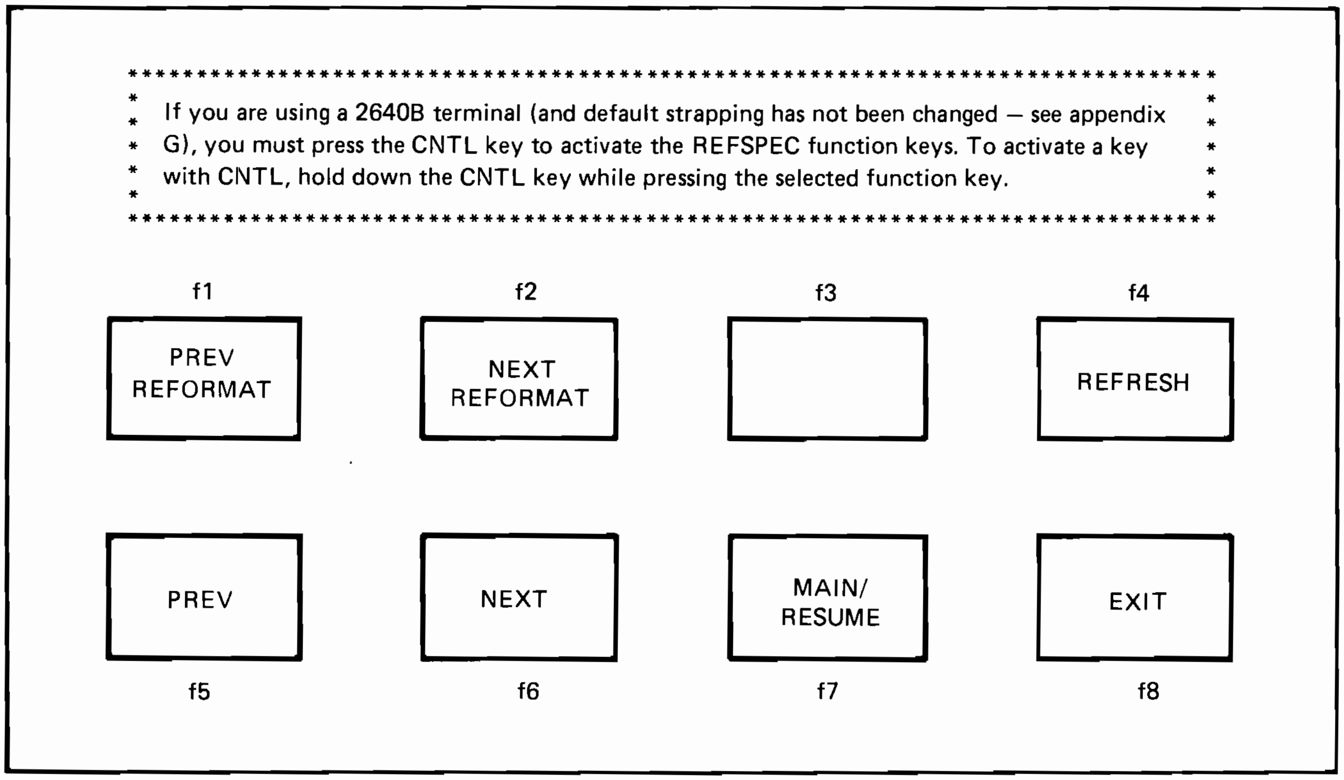

If you are using a 2640B terminal (and default strapping has not been changed — see appendix G), you must press the CNTL key to activate the REFSPEC function keys. To activate a key with CNTL, hold down the CNTL key while pressing the selected function key.

Figure 5-9. Function Keys for REFSPEC

These keys have the following meaning:

PREV REFORMAT f1	Go to previous INPUT FORMS menu (skipping any intervening menus).
NEXT REFORMAT f2	Go to next INPUT FORMS menu (skipping any intervening menus).
REFRESH f4	Redisplay current menu in its initial state before any specifications were entered, or existing specifications were modified.
PREV f5	Position to previous menu screen in sequence of menus.
NEXT f6	Position to next menu screen in sequence of menus.
MAIN/RESUME f7	Request MAIN menu or, if MAIN menu displayed, return to menu displayed when MAIN was requested.
EXIT f8	Terminate REFSPEC and return to MPE control.

NOTE

The function keys used by REFSPEC should not be confused with the function keys used by an operator during data entry (see figure 2-2). Although the two groups of keys are physically the same programmable keys, their functions differ significantly.

TERMINATING REFSPEC

You can terminate operation of REFSPEC at any time by pressing the EXIT function key (f8). EXIT returns you to MPE control and issues the colon prompt (:).

When you next run REFSPEC after terminating and request the same reformat file, the FORMS FILE menu is skipped and the MAIN menu is displayed on the screen. You may then select an option on the MAIN menu or you can press RESUME (f6) to display the FORMS FILE menu. If you specify a new reformat file on the REFORMAT FILE menu, the next menu issued is the FORMS FILE menu.

UNEXPECTED PROGRAM INTERRUPTION

As with FORMSPEC, if the program halts unexpectedly because of a terminal power failure or the operator pressing the BREAK key, control returns to MPE. Press RESET TERMINAL (twice on a 2645) and then press RETURN; a colon prompt (:) is issued. Type RESUME in response to this prompt. (If the keys you type do not appear on the screen, echo has been turned off; you must then press the ESC key and then press the colon (:) in order to restore echo.)

After you type RESUME, MPE issues the message READ PENDING. You then press the REFRESH key (f4) to return to the REFSPEC menu where you were interrupted. The menu will be cleared to initial or previously entered values. To insure against damage to the file, reenter the information on all menus pertaining to the reformat you were creating or modifying at the time of program interruption.

REFSPEC MENUS

REFSPEC issues menu screens in a predetermined sequence. As each menu is issued, enter the reformatting specifications you want in the reformat file, and then press ENTER. When ENTER is pressed, the specifications are written to the reformat file and the next menu in the sequence is issued.

If you do not want to enter any specifications (on a menu such as the GLOBALS menu where defaults are supplied), simply press NEXT (f6) to display the next menu.

When the reformatting is completely defined, request the MAIN menu by pressing the MAIN function key (f7). You can then compile the reformat file so that it can be executed.

The REFSPEC menu screens are shown in figure 5-10 in the order they are issued. First the REFORMAT FILE menu requests the name of the reformat file. For a new reformat file, the next menu asks for the name of the forms file whose data is to be reformatted. For an existing reformat file, the FORMS FILE menu is skipped and the MAIN menu issued.

You can move forwards or backwards through the sequence of menus to locate a particular menu by pressing NEXT (f6) or PREV (f5), respectively. You can skip an entire reformat specification by pressing NEXT REFORMAT (f2) or PREV REFORMAT (f1). (A "reformat" consists of an input forms sequence and its associated output record definition.) These keys allow you to go directly to the next or previous INPUT FORMS menu. An alternate method is to request the MAIN menu by pressing the MAIN key (f7) and then select a particular menu.

The EXIT key (f8) returns you to MPE control from REFSPEC.

Figure 5-10 illustrates the relation between the menu definitions presented on the REFSPEC menu screens and some of the special function keys.

NOTE

The number of OUTPUT FIELD menus issued depends on the number of reformat fields defined in the preceding OUTPUT RECORD menu. One OUTPUT RECORD menu is issued for each INPUT FORMS menu. You may be able to specify reformatting with as few as one INPUT FORMS menu, or you may need as many as there are forms in the forms file.

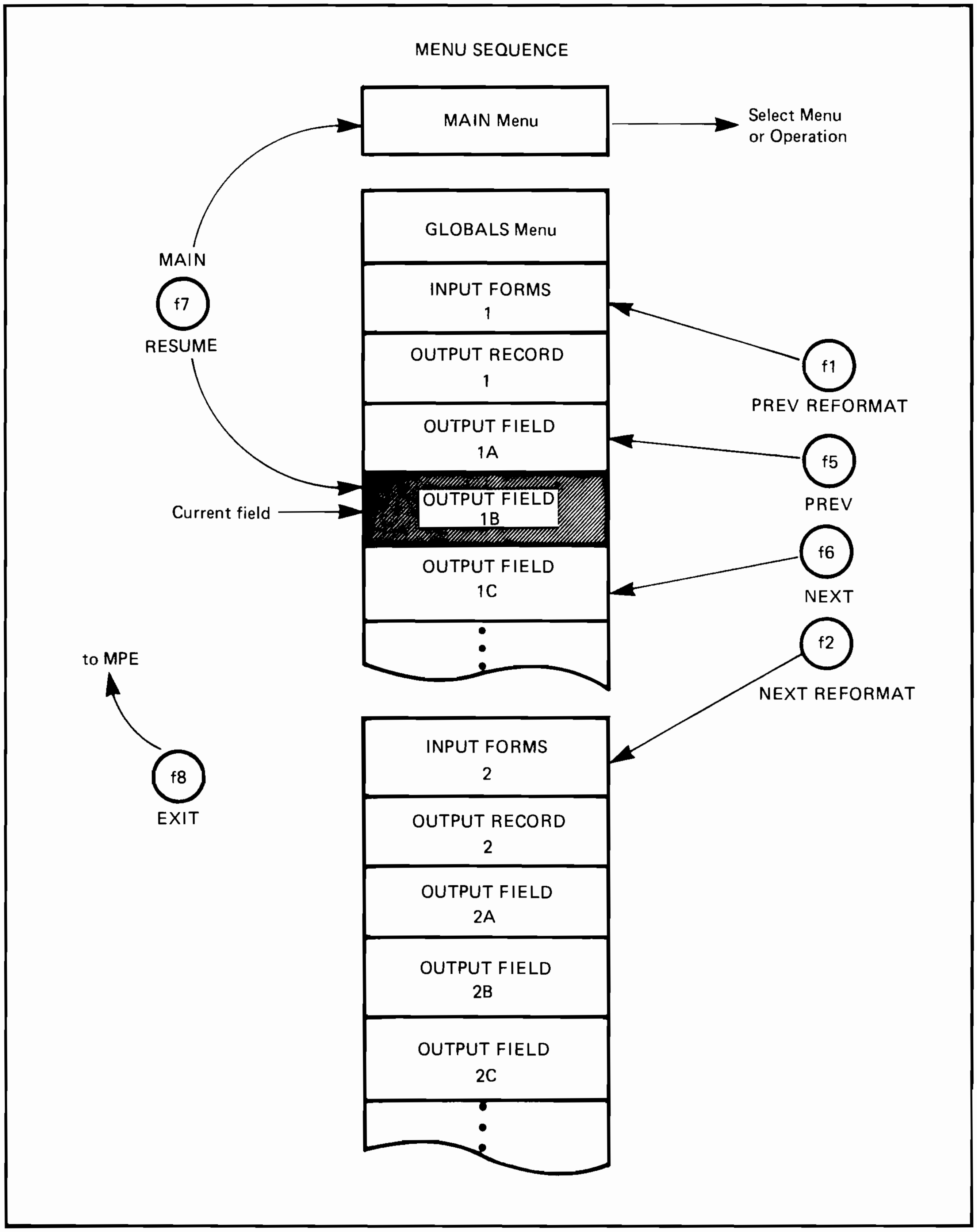
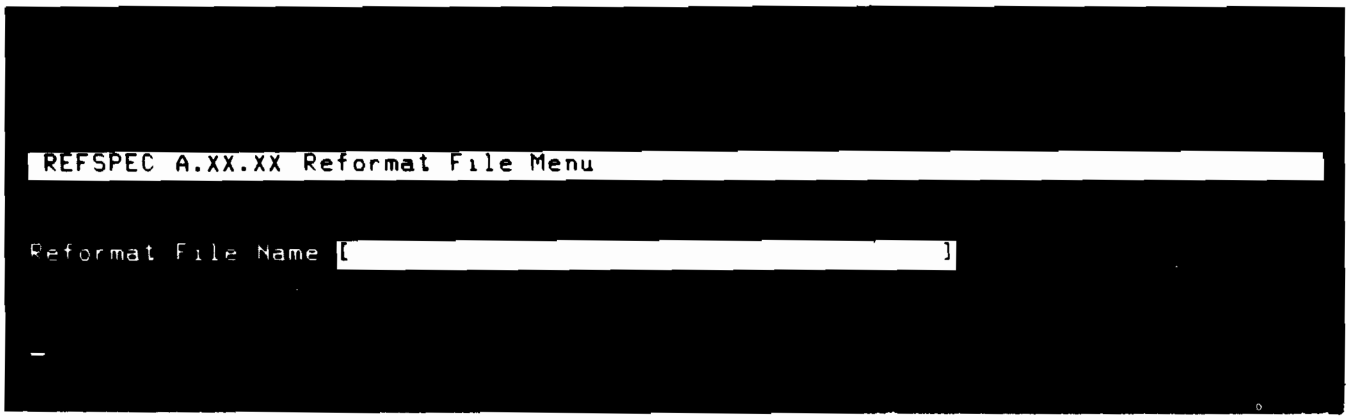


Figure 5-10. Relation of REFSPEC Menus to Function Keys

REFORMAT FILE MENU

The first menu displayed after you run REFSPEC is the REFORMAT FILE menu. The only information entered on this menu is the name of the reformat file.

This menu is illustrated in figure 5-11.



```
REFSPEC A.XX.XX Reformat File Menu

Reformat File Name [                                        ]

_
```

Figure 5-11. REFORMAT FILE Menu

If your reformat file has a lockword, always enter it along with the file name. Otherwise the system will prompt for the lockword, expecting a response from a terminal in character mode, when your terminal is in block mode.

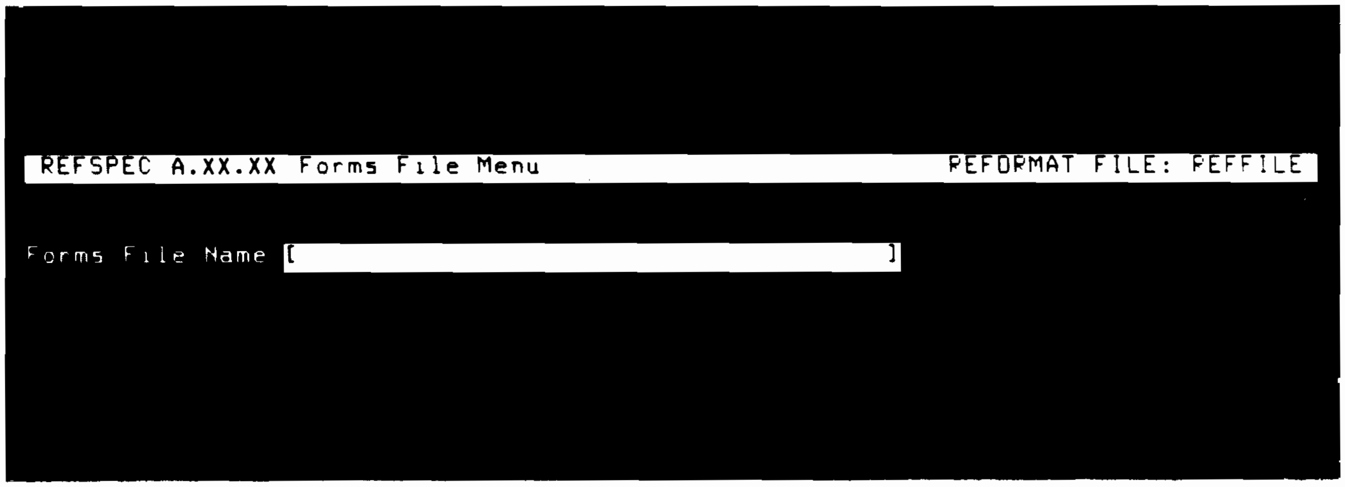
If the name identifies a new reformat file, REFSPEC creates the file and then issues the next menu, the FORMS FILE menu (see figure 5-12).

If the name you enter identifies an existing reformat file, REFSPEC assumes you want to continue creating the file or that you want to modify it. In this case, it skips the FORMS FILE menu and issues the MAIN menu (see figure 5-13).

FORMS FILE MENU

This menu asks you to enter the name of an existing forms file. This is the file that was used to create the batch file (or files) to be reformatted. Only one forms file can be specified for each reformat file.

Figure 5-12 illustrates the FORMS FILE menu.



```
REFSPEC A.XX.XX Forms File Menu                    REFORMAT FILE: REFFILE

Forms File Name [                                    ]
```

Figure 5-12. FORMS FILE Menu

The name of the data file may be a fully qualified MPE file name.

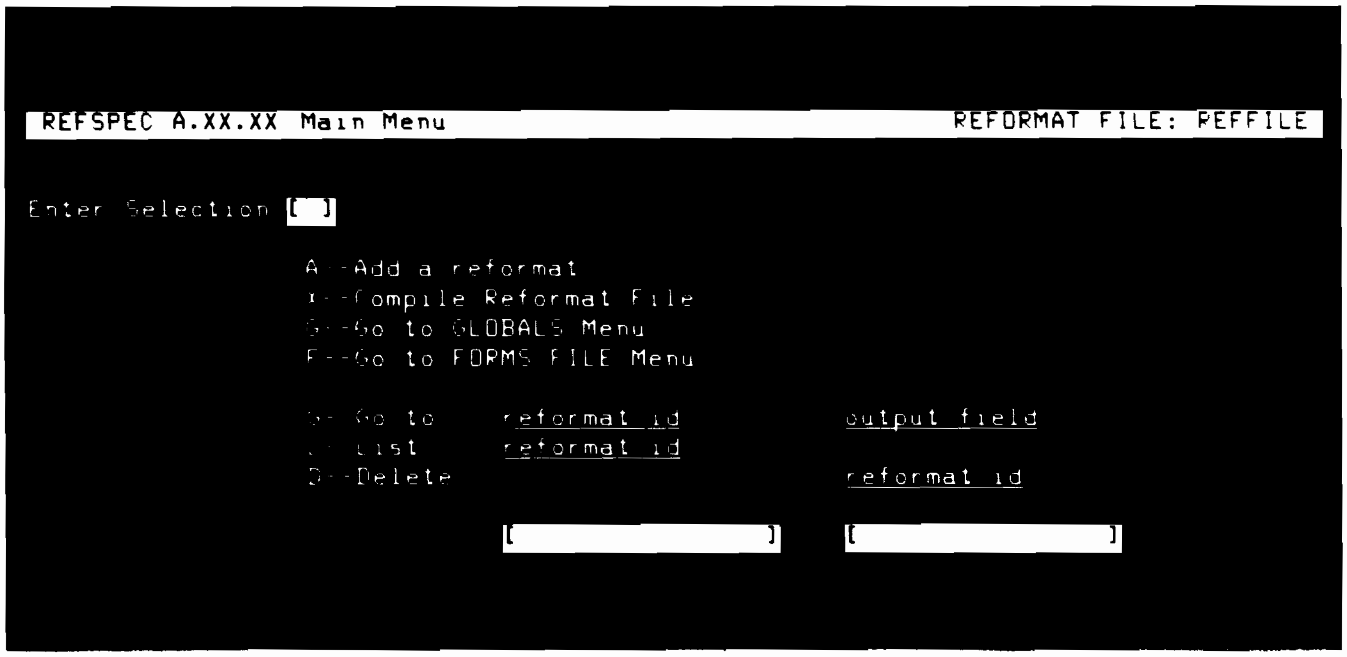
Since the release of version A.01.01 of V/3000, REFSPEC accepts either KSAM or standard MPE reformat files. The only forms files created by REFSPEC, however, are MPE files. The keys and forms records are interspersed together in the file. MPE reformat files do not create extra data segments and are more likely to be recovered after a power failure or system crash.

After you have identified the forms file, the MAIN menu is issued.

MAIN MENU

This menu is the main control menu for all REFSPEC operations. If the reformat file is new, you will usually select "Add a reformat". REFSPEC then issues the menus that allow you to define the reformat file. If the reformat file already exists, you may enter any selection depending on what you want to do. You can compile the reformat file, add new reformats, select a particular menu in order to change it, purge reformats and so forth.

The MAIN menu is illustrated in figure 5-13.



```
 REFSPEC A.XX.XX Main Menu                              REFORMAT FILE: REFFILE

 Enter Selection [ ]

                   A--Add a reformat
                   X--Compile Reformat File
                   G--Go to GLOBALS Menu
                   F--Go to FORMS FILE Menu

                   S--Go to    reformat id       output field
                   L--List     reformat id
                   D--Delete                     reformat id

                               [           ]     [           ]
```

Figure 5-13. MAIN Menu

ADDING A REFORMAT

When you are creating a new reformat file, you will always add at least one reformat; if you are modifying an existing reformat file, you may want to add a reformat. To add a reformat, simply enter "A" in the MAIN menu selection box. In response, REFSPEC issues the INPUT FORMS menu.

New reformats are added to the end of the reformat file.

COMPILING THE REFORMAT FILE

In order for the REFORMAT program to use the reformat file, the file must be compiled. You compile a reformat file by entering an "X" in the MAIN menu selection box.

If the file has already been compiled, it is recompiled and the newly compiled version replaces the previous version. There is never more than one compiled version of a reformat file at a time. When you modify the source version, the compiled version is not affected.

ADDING GLOBAL SPECIFICATIONS

Certain characteristics of the reformat definition are global; that is, they apply to the entire output file. For example, record length and the end of record separators are global definitions. (Refer to the GLOBALS menu description below.)

REFSPEC supplies default values for all global characteristics and, unless you want to change these defaults, you need never be concerned with global values. If you do want to specify non-default global characteristics, enter "G" in the MAIN menu selection box, but do not specify a reformat identifier or output field name. The GLOBALS menu will be displayed so you can enter global specifications.

CHANGING THE FORMS FILE NAME

Normally, you will not need to modify the forms file name once it has been specified. However, in case the forms file is renamed with the MPE or KSAMUTIL RENAME command, you can go to the FORMS FILE menu and rename the forms file to correspond to its new name. To do this, enter "F" in the MAIN menu selection box and, when the FORMS FILE menu is displayed, type in the new forms file name.

MODIFYING REFORMAT SPECIFICATIONS

You can change a reformat specification at any time, either as you initially define the specification, or after the reformat file has been compiled. In either case, you locate the particular specification through the MAIN menu selection box or with the function keys that control menu sequence.

CHANGING AN INPUT FORMS SEQUENCE

Each input forms sequence consists of the form names associated with batch records to be reformatted and written to the output file. (Since each batch file record is associated with a form name, the list of form names is in effect a list of batch records.) The first form name in any input forms sequence is called the "reformat identifier". Each reformat identifier is unique to the reformat file and is used to identify the input forms sequence.

If you want to modify an input forms sequence, enter "G" in the MAIN menu selection box, and then specify the reformat identifier that identifies that sequence; or locate the input forms sequence you want with the PREV REFORMAT (f1) or NEXT REFORMAT (f2) function keys. The INPUT FORMS menu is displayed with the form names previously specified. You can then change any of these names, add new names, or delete existing names. When you have made the changes, press ENTER. You can change the reformat identifier exactly as you would change any other form name, but the ch anged identifier must still be unique to the reformat file.

Note that if you change an INPUT FORMS menu, you must validate the associated OUTPUT RECORD and you may have to validate the OUTPUT FIELD menus. To do this, you request the applicable OUTPUT RECORD and OUTPUT FIELD menus and press the ENTER key for each. (If these menus were affected by your change to the INPUT FORMS menu, you must make the appropriate changes before pressing ENTER.) Until these menus are validated, REFSPEC prints a warning when they are displayed.

CHANGING AN OUTPUT RECORD DEFINITION

Every input forms sequence has an associated output record definition. The output record definition defines how the individual fields or portions of fields from the batch file are to be written to the output file. You can change some of these field specifications directly on the freeform OUTPUT RECORD menu, and others on a particular OUTPUT FIELD menu. OUTPUT FIELD menus can be located by entering

"G" in the MAIN menu selection box and then specifying a reformat id and output field name, or by using the NEXT (f6) or PREV (f5) function keys. An output record definition can be located only through these function keys.

The basic output record descriptions are entered and changed on the OUTPUT RECORD menu. The output fields can be rearranged, new fields or constants added, or existing fields or constants deleted only on this menu. To change specific field reformatting specifications, you must go to the OUTPUT FIELD menu for that field. (For details, refer to the OUTPUT RECORD menu and OUTPUT FIELD menu descriptions below.)

After changing a reformat, you must compile the reformat file to enter these changes in the executable version of the file.

DELETING A REFORMAT

You can purge an entire reformat with its associated output record definition and field specifications. Enter "D" in the MAIN menu selection box and then specify the reformat identifier that identifies the reformat to be deleted. The reformat specification is not physically removed from the reformat file, but it can no longer be referenced by the REFORMAT program.

After deleting a reformat, you must compile the reformat file or the reformat will still be in the file.

LISTING A REFORMAT

You can print a listing of the entire reformat file or only a single reformat on an offline device (usually, the line printer). To list a single reformat, enter "L" in the MAIN menu selection box and then specify the reformat identifier for the reformat you want listed. If you want to list all reformats in the reformat file, simply enter "L" in the selection box without specifying a reformat identifier.

Figure 5-14 illustrates the listing of a reformat.

```
REFSPEC  VERSION Y.04.00                                    WED, SEP 20, 1978,   9:01 AM
REFORMAT FILE: RTEST2.CRANE.DMS                                                   PAGE 4

REFORMAT SECONDFORMNAME2

   INPUT FORMS (IN SEQUENCE):
      SECONDFORMNAME2


OUTPUT RECORD DEFINITION

INPUT FIELD          SUBST LEN  FORM NAME           OUTPUT FIELD       STRT  LEN    STRT
                     STRT                                              COL          REC
****************************************************************************************
FIELD1               1     10   SECONDFORMNAME2     FIELD1             1     20      *
FIELD2               1     10   SECONDFORMNAME2     A                  28    10
FIELD2               1     10   SECONDFORMNAME2     B                  45    20
FIELD2               1     10   SECONDFORMNAME2     C                  1     20      *
FIELD2               1     10   SECONDFORMNAME2     D                  28    20
FIELD1               1     10   SECONDFORMNAME2     E                  55    10
FIELD1               1     10   SECONDFORMNAME2     O                  1     10      *
****************************************************************************************
```

Figure 5-14. Reformat Listing

```
INPUT  FIELD: FIELD1              START: 1     LENGTH: 10     FORM: SECONDFORMNAME2
OUTPUT FIELD: FIELD1              START: 1     LENGTH: 20     DATA TYPE: IMP2

   STRIP:         ALL: "ABCD"         LEADING:                   TRAILING:
   CHECK DIGIT:
   JUSTIFY:
   SIGN:            L                 PLUS SIGN? Y
   FILL:          ALL:                LEADING:                   TRAILING:

INPUT  FIELD: FIELD2              START: 1     LENGTH: 10     FORM: SECONDFORMNAME2
OUTPUT FIELD: A                   START: 28    LENGTH: 10     DATA TYPE: IMP9

   STRIP:         ALL:                LEADING: "EFGH"            TRAILING:
   CHECK DIGIT:
   JUSTIFY:
   SIGN:          Z                   PLUS SIGN?
   FILL:          ALL:                LEADING:                   TRAILING:

INPUT  FIELD: FIELD2              START: 1     LENGTH: 10     FORM: SECONDFORMNAME2
OUTPUT FIELD: B                   START: 45    LENGTH: 20     DATA TYPE: NUM

   STRIP:         ALL:                LEADING:                   TRAILING: "IJKL"
   CHECK DIGIT:
   JUSTIFY:
   SIGN:          L                   PLUS SIGN? Y
   FILL:          ALL:                LEADING:                   TRAILING:

REFSPEC  VERSION X.04.00                                  WED, SEP 20, 1978,  9:01 AM
REFORMAT FILE: RTEST2.CRANE.UMS                                                PAGE 5

INPUT  FIELD: FIELD2              START: 1     LENGTH: 10     FORM: SECONDFORMNAME2
OUTPUT FIELD: C                   START: 1     LENGTH: 20     DATA TYPE: NUM3

   STRIP:         ALL: "MNOP"         LEADING: "QRST"            TRAILING:
   CHECK DIGIT:
   JUSTIFY:
   SIGN:          Z                   PLUS SIGN? Y
   FILL:          ALL:                LEADING:                   TRAILING:

INPUT  FIELD: FIELD2              START: 1     LENGTH: 10     FORM: SECONDFORMNAME2
OUTPUT FIELD: D                   START: 28    LENGTH: 20     DATA TYPE: IMP9

   STRIP:         ALL: "UVWX"         LEADING:                   TRAILING: "YZ12"
   CHECK DIGIT:
   JUSTIFY:
   SIGN:          Z                   PLUS SIGN?
   FILL:          ALL:                LEADING:                   TRAILING:

INPUT  FIELD: FIELD1              START: 1     LENGTH: 10     FORM: SECONDFORMNAME2
OUTPUT FIELD: E                   START: 55    LENGTH: 10     DATA TYPE: NUM

   STRIP:         ALL:                LEADING:                   TRAILING:
   CHECK DIGIT:
   JUSTIFY:
   SIGN:          R                   PLUS SIGN? Y
   FILL:          ALL:                LEADING:                   TRAILING:

INPUT  FIELD: FIELD1              START: 1     LENGTH: 10     FORM: SECONDFORMNAME2
OUTPUT FIELD: D                   START: 1     LENGTH: 10     DATA TYPE: CHAR

   STRIP:         ALL:                LEADING:                   TRAILING:
   CHECK DIGIT:
   JUSTIFY:
   SIGN:                              PLUS SIGN?
   FILL:          ALL:                LEADING:                   TRAILING:  -
```

Figure 5-14. Reformat Listing (Cont.)

RESUMING DESIGN FROM MAIN MENU

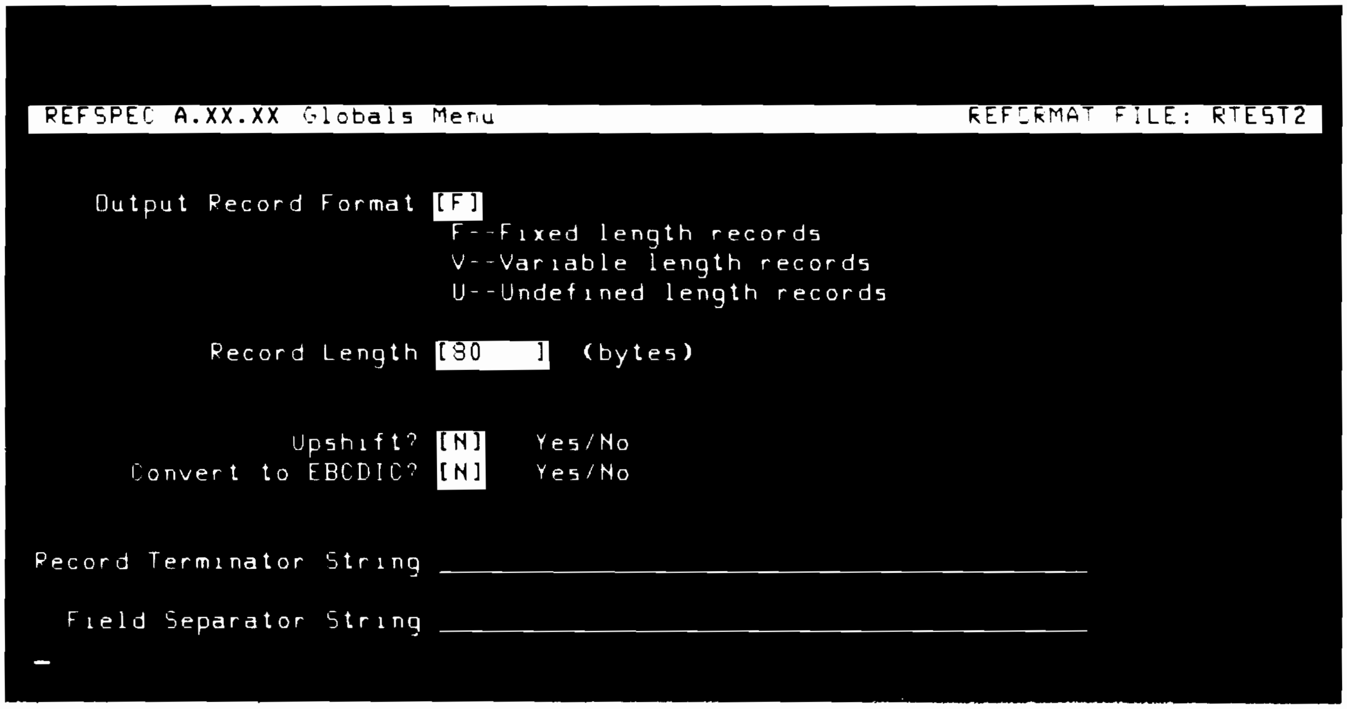
If you requested the MAIN menu by pressing the MAIN key during reformat design, you can return to the menu you were in by pressing the RESUME key (f7). Note that this is the same key as the MAIN key, but it acts differently in the MAIN menu than it does in other menus.

You can also go to the previous menu or the next menu by pressing PREV (f5) or NEXT (f6) respectively. When requested from the MAIN menu, the previous menu is the one preceding the menu you were in when you requested the MAIN menu, and the next menu is that following the menu you were in. Similarly, you can request the previous or next reformat directly by pressing PREV REFORMAT (f1) or NEXT REFORMAT (f2). The previous or next reformat is relative to the reformat you were designing when MAIN was requested.

GLOBALS MENU

This menu requests information that applies to the total reformat process. All information entered on the GLOBALS menu relates to the output file produced when the REFORMAT program is run. Since only one output file is generated for each REFORMAT execution, this information appears only once in the reformat file.

The GLOBALS menu is illustrated in figure 5-15.



```
 REFSPEC A.XX.XX Globals Menu                                    REFORMAT FILE: RTEST2

       Output Record Format [F]
                              F--Fixed length records
                              V--Variable length records
                              U--Undefined length records

              Record Length [80    ]  (bytes)


                   Upshift? [N]    Yes/No
          Convert to EBCDIC? [N]    Yes/No

 Record Terminator String ________________________________________

   Field Separator String ________________________________________
 _
```

Figure 5-15. GLOBALS Menu

When the GLOBALS menu is displayed, default values are shown for each option. If you are satisfied with the default values provided by REFSPEC, you can simply press the ENTER key to request the next menu. The possible values and their defaults are listed below.

Record Format — Specifies whether the output record is fixed, variable, or undefined in length. Enter:

F - Fixed-length records
V - Variable-length records
U - Undefined-length records

Default = F (Fixed length)

Record Length — A positive integer that specifies the total number of characters in the output record, including all fields and separators.

If the output records are fixed length and the total number of characters is less than this length, the record is padded with blanks at the end. If there are more characters in the field than will fit in the record, the record is written up to the specified length and the excess characters are discarded. In this case, a warning message is issued to the operator who runs program REFORMAT.

For variable length records, the actual record length is the sum of all the fields written to the record, including separators. In this case, the record length specified here is the maximum length allowed for any record.

Default = 80 characters.

Upshift

Indicates whether letters of the alphabet are to be shifted up to all uppercase letters when data is written to the output file. Specified as:

Y — Shift letters to uppercase.
N — Leave letters as entered by operator.

Default = N (do not upshift).

If nothing is specified, an error is returned.

Convert to EBCDIC

Enter "Y" if you want the output file to be written in EBCDIC code rather than ASCII; enter "N" if you want the data left in ASCII code.

Default = N (do not convert).

Record Terminator

Indicates a character or string of characters to be appended to the end of every record.

The record terminator may be specified as any of the following:

- Quoted String — Any ASCII characters, including blanks, enclosed within single or double quotes. For example: "eof" or "   " or '**'.
- ASCII Code — Numeric equivalent to an ASCII character preceded by a $. Code may be any decimal number in range 0-127 (refer to appendix C for decimal equivalent to ASCII code). For example: $34 is the numeric equivalent of quotation mark; $65 is the equivalent of "A".
- System Constant — The following system defined constants may be used as a terminator:

  $LF — line feed
  $CR — carriage return
  $GS — group separator
  $US — unit separator
  $RS — record separator

Default = No terminator

If a record terminator is not specified, no special terminator is placed at the end of output records.

You can combine any of the above terminators into a single terminator by specifying them one after the other. For example: "end" $LF $CR or $120 "!" $CR or "ABC" "DEF".

Such multiple terminators are concatenated together to form a single string that is inserted between fields. Note that blanks are optional except between quoted strings where a separating blank is required. If a blank does not separate quoted strings, the quote is included; for example: "ABC""DEF" becomes ABC"DEF in the record.

Field Separator A user-defined value to be inserted between all consecutive fields in the output record. It will not appear after the last field in the record, or before or after a constant. If two consecutive fields are assigned specific column positions so that the second does not immediately follow the first, the field separator is placed after the first field. For example, suppose the field separator is "**", and the fields are defined as:

```
FIELD1  value  = ABCD  start column = 5
FIELD2  value  = XYZ   start column = 15
```

The output record is written as:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
				A	B	C	D	*	*					X	Y	Z

The separator may be a quoted string, an ASCII code, or a system constant, as described above for the Record Terminator.

Default = No separator.

Unless a separator is specified, fields in an output record are not separated, but are written as one continuous string of characters.

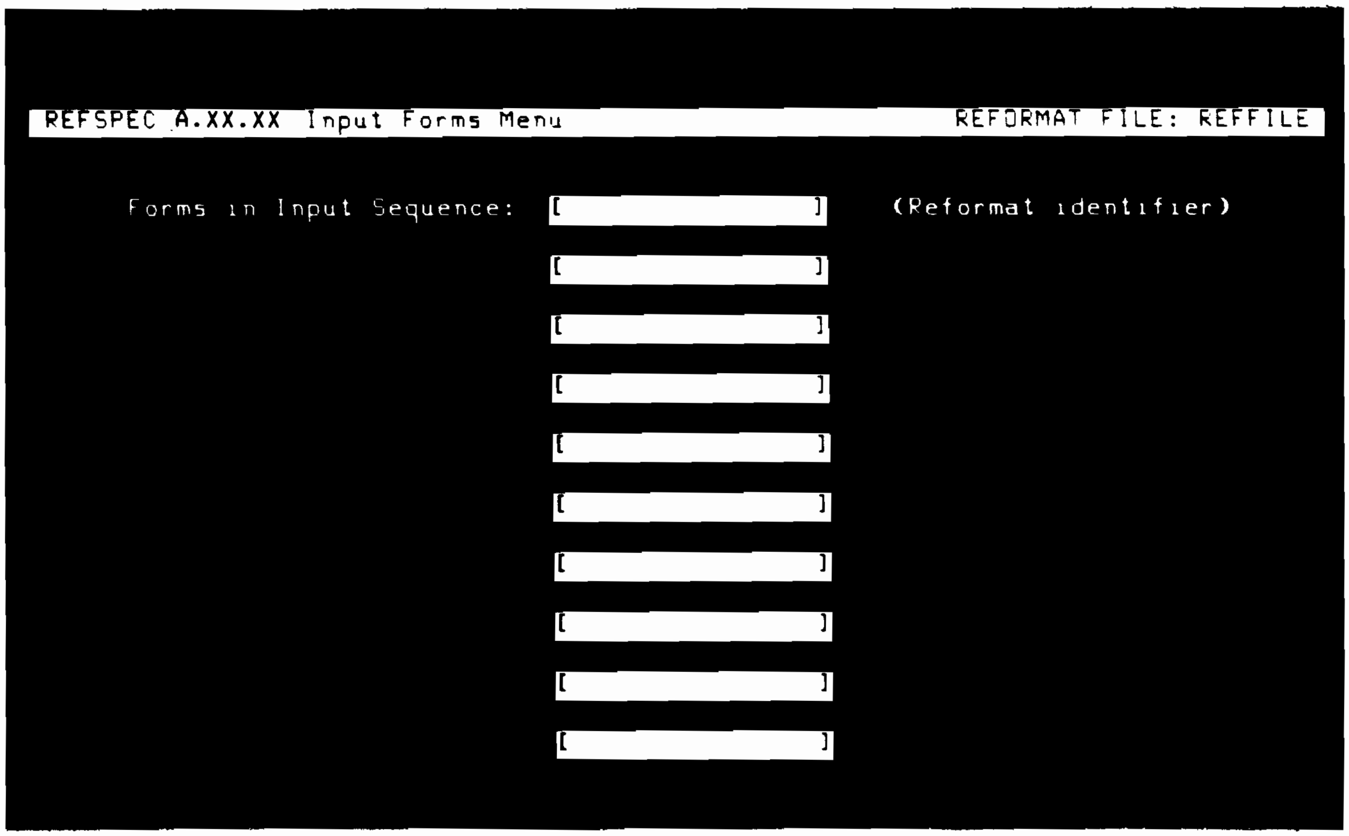
After global specifications are entered on the GLOBALS menu, the next menu issued is the INPUT FORMS menu.

INPUT FORMS MENU

This menu is used to specify one or more forms whose data is to be written to the output file. You can specify as many input forms sequences as there are forms in the forms file, or you can specify as few as one input forms sequence.

Each input forms sequence may contain a single form name or it may contain up to 10 form names. The first form in the sequence (or the only form) is the "reformat identifier". All reformat identifiers must be unique to the reformat file. Subsequent form names in the sequence need not be unique.

Figure 5-16 illustrates the INPUT FORMS MENU.



```
REFSPEC A.XX.XX Input Forms Menu                              REFORMAT FILE: REFFILE

          Forms in Input Sequence: [            ]   (Reformat identifier)
                                   [            ]
                                   [            ]
                                   [            ]
                                   [            ]
                                   [            ]
                                   [            ]
                                   [            ]
                                   [            ]
                                   [            ]
```

Figure 5-16. INPUT FORMS Menu

The form names in any input forms sequence must be existing forms in the forms file named on the FORMS FILE menu for this reformat file. (Refer to figure 5-12.)

Care must be taken when you specify a sequence of forms. The form names following the reformat identifier must be in exactly the same order as the forms appear in the batch file. (Each data record in the batch file is entered on a single form and the name of this form is included with the data in the batch file.)

To illustrate, assume the following input forms sequence:

FORMA (reformat identifier)
FORMB
FORMD

And assume the first three records of data in the batch file were entered on forms in the sequence:

FORMA
FORMB
FORMC

The data entered on FORMA of this sequence will not be written to the output file. This is because of the method used by the REFORMAT program to match batch file records with input forms sequences in the reformat file.

The REFORMAT program:

- Reads records from the batch file in sequential order from the beginning. (Assume the first record was entered on FORMA.)
- Checks all reformat identifiers in the reformat file until it finds FORMA. (Assume FORMA is found as the identifier for the sequence FORMA, FORMB, FORMD.)
- Reads the next record from the batch file. (Assume the next record was entered on FORMB.)
- Checks form names in the input forms sequence following FORMA. (Assume the next form in the sequence is FORMB, and the check is satisfactory.)
- Reads the next record from the batch file. (Assume the next record was entered on FORMC.)
- Checks next form name in the input forms sequence after FORMB. (Assume this form is FORMD; The form names do not match.)
- Skip data record written on FORMA.

The REFORMAT program then takes the next record in the batch file, in this case, the record written on FORMB, and searches the entire reformat file for a reformat identifier, FORMB. If such a reformat identifier is found, it checks the rest of the input forms sequence (as described above) to make sure that all forms in the sequence match the forms on which the batch records following FORMB were entered.

When a match is successful, each batch record is discarded as it is processed. If the match is unsuccessful, only the first batch record in a sequence is discarded.

OUTPUT RECORD MENU

An OUTPUT RECORD menu is issued for every input forms sequence. It specifies which fields in the input forms are to be written to an output record. The output record definition may generate one output record, many output records, or only a portion of an output record.

The order in which fields are specified is the order in which they are written to the output record.

The OUTPUT RECORD menu is not a true menu screen. Rather than selecting from predefined options, you enter the reformat fields in a free-form area on the screen. Once the output record definition is specified, you will have a complete list of the output record image for the forms specified in the corresponding input forms sequence. At a glance, you can see the order in which fields will appear and where each record begins. If only a portion of a field is to be written, this too is evident from the output record definition. As such, the output record definition can be thought of as the "screen image" of the reformat similar to the screen image generated for each form in the forms file.

The output record definition is entered on the OUTPUT RECORD menu shown in figure 5-17.

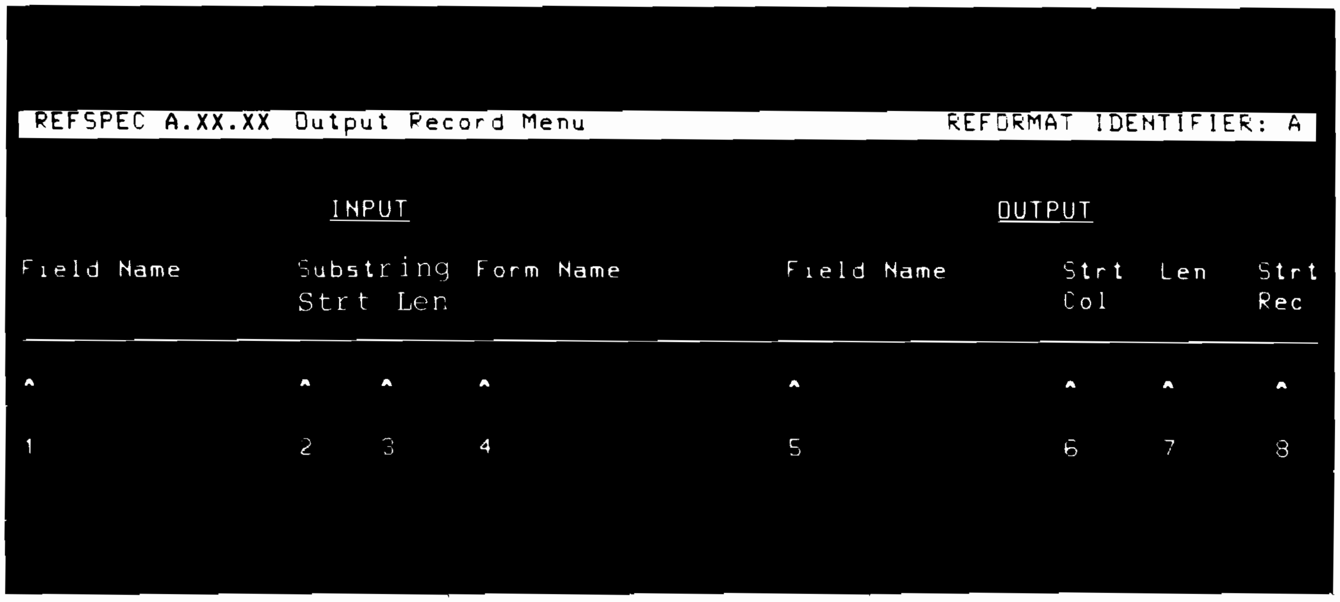

```
 REFSPEC A.XX.XX Output Record Menu                               REFORMAT IDENTIFIER: A

                     INPUT                                            OUTPUT

Field Name          Substring  Form Name        Field Name           Strt   Len    Strt
                    Strt  Len                                        Col           Rec
_____________________________________________________________________________________
^                   ^     ^    ^                ^                    ^      ^      ^

1                   2     3    4                5                    6      7      8
```

Figure 5-17. OUTPUT RECORD Menu

When this menu is displayed, the cursor is positioned to the first position in the first line in which you can enter an output record specification. Tabs are set so that when you press the TAB key, the cursor is moved to each subsequent position.

Note that the only required entry is the field name; other specifications depend on the particular definition.

The specifications under the heading INPUT identify the fields (or portions of fields) to be accessed from the batch file. The specifications under OUTPUT determine how the field is to be written to the output record.

You can specify a constant (literal string, system constant, or numeric equivalent of an ASCII character) instead of a field name. A constant may start anywhere on the line, but cannot extend past column 75.

You may enter as many specification lines as there is room in your terminal memory. (Check the terminal memory capacity in the reference manual for your terminal.)

NOTE

Since the output record specifications are entered on a free-form area of the screen, you can type anywhere on the screen. You should be careful not to delete or overwrite lines accidentally. The field specifications should be entered in the positions to which the tabs are set.

INPUT SPECIFICATIONS

Field Name (Required)	Identifies the field in the batch file to be written to the output file. The field name must identify an existing field in one of the forms specified in the preceding input sequence. The field name need not be unique to the output definition. But, if it is not unique, a different output field name must be entered for each identical input field name.
Subst Strt (Optional)	If you want to retrieve a portion of an input field that is not at the beginning of the field, specify the first character position that you want written to the output field as the Subst Strt value. Character positions in a field are numbered from 1. For example, suppose you want only the last five characters of a 20-character field written to the output file, enter 16 under Subst Strt. Used in combination with the Len parameter, you can specify any group of consecutive characters in an input field. Default = First character in field (1).
Len (Optional)	Allows you to retrieve only a portion of an input field. For example, if you want only the first five characters in a field, leave the Subst Strt field blank and enter 5 under Len. By combining the Len specification with the Subst Strt specification, you can select any consecutive group of characters in the input field. For example, to select characters 5 through 14 of the input field, set Subst Strt to 5 and Len to 10. Default = length of input field — Subst Strt + 1
Form Name (Optional)	Identifies the form in which the input field appears. This parameter can be included when a field with the same name appears in more than one form in the forms file. If omitted, and the field name is not unique, the first form in the input forms sequence in which the field appears is assumed. Default = First form in which field appears.

OUTPUT SPECIFICATIONS

Output Field Name (Optional)	In case the input field name is not unique, you can give the field (or portion of the field) a unique identifier in this position. Any legal 15-character identifier can be specified. (Note that any lowercase letters are shifted to all uppercase by REFSPEC.) Default = Input field name.

Strt Col (Optional)

If you want the field to start in a particular column in the output record rather than immediately after the preceding field, you can specify a starting column. Column numbers in this case correspond to character (byte) positions in the output record. Columns are numbered starting with 1.

When you specify a new starting column, REFSPEC checks to make sure the starting column does not overwrite a previous field in the output record. If the starting column is not correctly specified, REFSPEC issues an error message.

One reason for changing the starting column is to insure that a field starts on a word boundary. For example, suppose the first field to be written is five characters long. You can either increase the length of this field in the "Len" position to an even number of characters, or you can specify a new starting column for the next field.

Default = Consecutive fields with no intervening spaces.

Len (Optional)

You can specify the number of characters for the output field in this position. If you specify a length shorter than the input field, character type data is truncated when it is written to the output field; numeric data may not be converted if the field is too short.

If you plan to add a check digit to the field, you must specify the output field length as at least one character longer than the input field length.

Default = Input field length (or substring length).

Strt Rec (Optional)

In order to separate the output into records, you must specify the beginning of each record. Any character entered in the Strt Rec position of the OUTPUT RECORD menu marks the associated field or constant as the beginning of a new record. For consistency, you should select a standard character, such as an asterisk.

If a starting record is not marked in the output record definition, the fields in the current definition follow the fields in the previous definition as part of the same record.

Default = Field is not first field.

Constant (Optional)

You can specify a constant instead of a field name. The constant can be any of the following:

- a character string (literal) enclosed within single or double quotes;
- the numeric equivalent of an ASCII character preceded by a dollar sign ($);
- one of the system-defined constants:

$CR	carriage return
$LF	line feed
$GS	group separator
$US	unit separator
$RS	record separator

The following are acceptable constants:

"Part No."   'ABC' $65 $34"NAME"$34 $RS

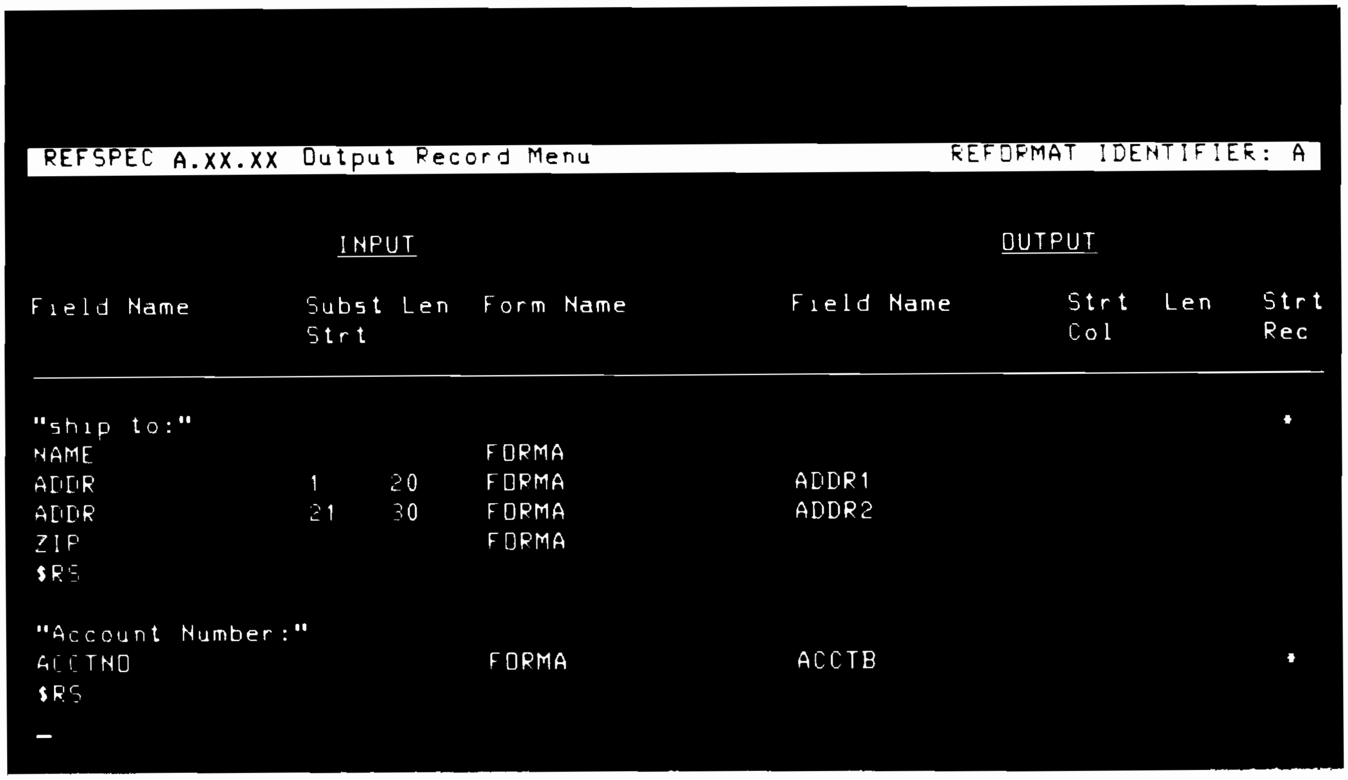
Constants may not be specified on the same line as a field name. More than one constant can be specified on a line. If more than one constant is specified, they are concatenated into a single literal value. For example: "Part No." $GS $65.

Blanks are not significant except within quoted strings where they must be included.

A constant or group of concatenated constants must not extend past column 75 of the screen. Apart from this restriction, they can appear anywhere on the line.

Default = No constant.

Figure 5-18 illustrates an OUTPUT RECORD menu specification.



```
 REFSPEC A.XX.XX  Output Record Menu                         REFORMAT IDENTIFIER: A

                      INPUT                                      OUTPUT

Field Name          Subst Len  Form Name           Field Name          Strt   Len   Strt
                    Strt                                               Col          Rec
-----------------------------------------------------------------------------------------
"ship to:"                                                                           *
NAME                           FORMA
ADDR                1     20   FORMA               ADDR1
ADDR                21    30   FORMA               ADDR2
ZIP                            FORMA
$RS

"Account Number:"
ACCTNO                         FORMA               ACCTB                             *
$RS
_
```

Figure 5-18. Sample OUTPUT RECORD Menu

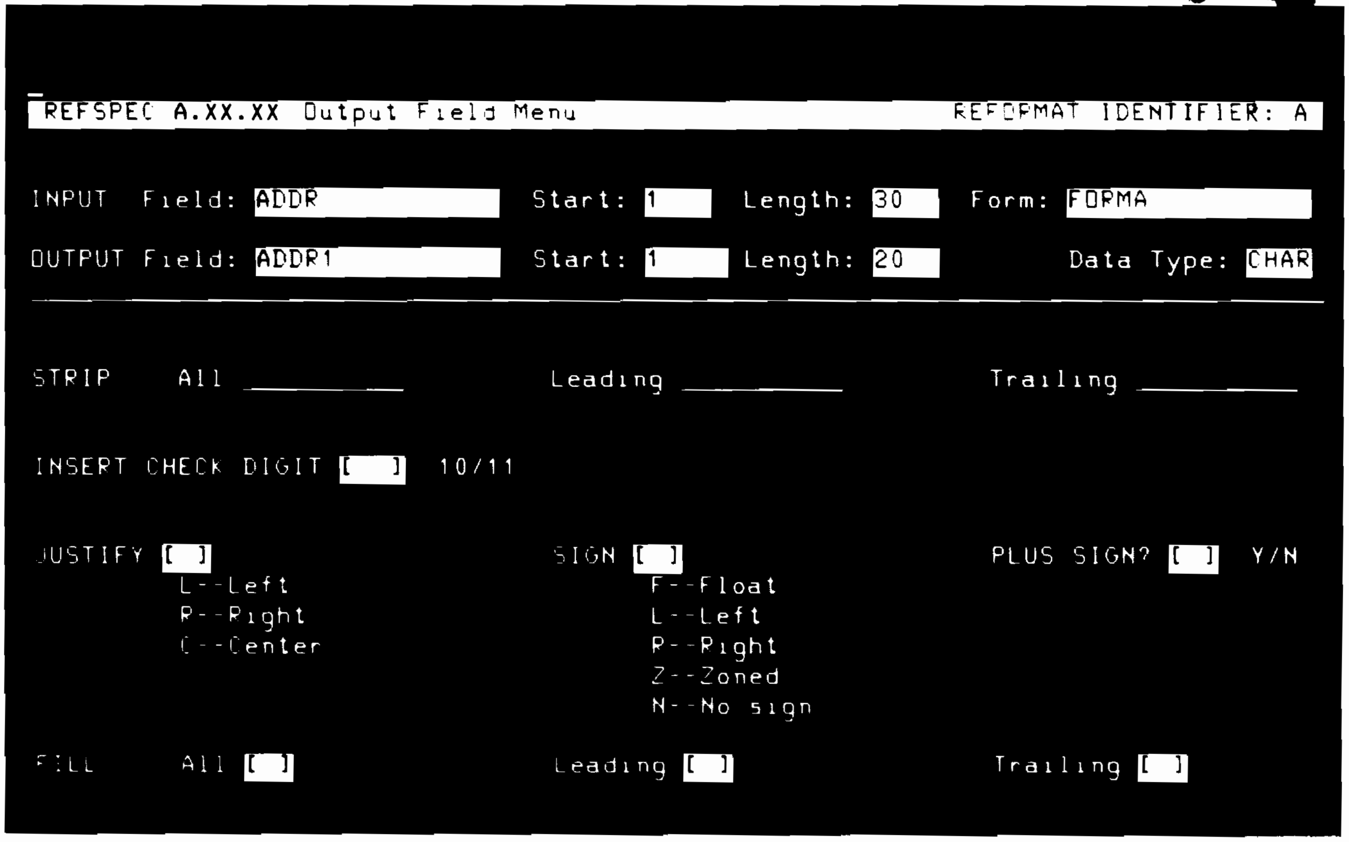
When the output record definition has been defined, press ENTER. A FIELD menu will be issued for each field name associated with an input field name in the output record definition.

OUTPUT FIELD MENU

A separate OUTPUT FIELD menu is displayed for every input field named in the preceding OUTPUT RECORD menu. OUTPUT FIELD menus are issued in the order the field names were entered in the output record definition. No OUTPUT FIELD menu is issued for constants.

The OUTPUT FIELD menu displays the information for a field entered on the preceding output record definition. In addition, it allows you to change the data type of a field from CHAR to any of the legal FORMSPEC data types. It also allows some formatting in addition to that performed by default when data is moved from the input field in the batch file to the output field in the output file.

Figure 5-19 shows the format of an OUTPUT FIELD menu.



```
 REFSPEC A.XX.XX Output Field Menu                         REFORMAT IDENTIFIER: A

 INPUT  Field: ADDR            Start: 1    Length: 30   Form: FORMA

 OUTPUT Field: ADDR1           Start: 1    Length: 20        Data Type: CHAR
 ______________________________________________________________________________

 STRIP     All ________       Leading ________          Trailing ________

 INSERT CHECK DIGIT [  ]  10/11

 JUSTIFY [ ]                 SIGN [ ]                   PLUS SIGN? [ ]  Y/N
         L--Left                 F--Float
         R--Right                L--Left
         C--Center               R--Right
                                 Z--Zoned
                                 N--No sign

 FILL      All [ ]           Leading [ ]                Trailing [ ]
```

Figure 5-19. OUTPUT FIELD Menu

The first line of the menu displays the input field values defined for this field on the preceding OUTPUT RECORD menu. None of these values can be changed.

The second line displays the output field values defined for this field. Except for the data type, these values were established on the preceding OUTPUT RECORD menu and cannot be changed on this menu. The data type of all fields to be written to the output file is defaulted to CHAR. This is done because only character type fields are written to the output record exactly as they were entered. If you want the field to be formatted to the format of a particular data type, you must change the data type specification on the OUTPUT FIELD menu. (Refer to table 5-1 on the following page for the standard formatting by data type.)

The remaining items in the OUTPUT FIELD menu allow you to specify reformatting to be performed on the data item when it is moved to the output field. Note that these reformatting specifications cannot override the standard conversion and reformatting performed on the item based on the destination data type.

Table 5-1. Standard Formatting by Data Type

Destination Data Type	Standard Formatting
CHAR	*Conversion:* none.
	Data movement: Move data to output field, character by character from leftmost character (including blanks) through last character (including blanks).
	If the data is shorter than the output field, pad with blanks on the right. If the data is longer than the output field, truncate on the right.
	Assume a value with length=10 ΔΔΔCHARΔΔΔ
	moved to CHAR field, length=10 ΔΔΔCHARΔΔΔ
	to CHAR field, length=15 ΔΔΔCHARΔΔΔΔΔΔΔΔ
	to CHAR field, length=5 ΔΔΔCH
DATE	*Conversion and Movement:* Convert entered date to dd/dd/dd, where the order of digits depends on the date type. A date is written to MDY type field as mm/dd/yy; to DMY type field as dd/mm/yy; to YMD field as yy/mm/dd.
	Write converted date, character by character, to the output field starting with the leftmost character. If the date is shorter than the field, fill with blanks on the right; if longer than the field, issue error message.
	Assume an MDY date, length=12 JAN 31, 1978
	moved to MDY field, length=12 01/31/78ΔΔΔΔ
	to DMY field, length=8 31/01/78
	to YMD field, length=8 78/01/31
	to any date field, length=7 ERROR
DIGIT	*Conversion and Movement:* Strip any leading zeros. Right justify converted data in the output field. If the data is shorter than the field, pad it with blanks on the left. If the data is too long for the output field, strip any leading blanks, one at a time until the data fits. If, after all leading blanks are removed, the data does not fit, issue error message.
	Assume an integer, length=7 0012345
	moved to DIGIT field, length=7 ΔΔ12345
	to DIGIT field, length=10 ΔΔΔΔΔ12345
	to DIGIT field, length=5 12345
	to DIGIT field, length=4 ERROR

Table 5-1. Standard Formatting by Data Type (Continued)

Destination Data Type	Standard Formatting
NUMn or NUM	*Conversion and Movement*: Strip any commas, sign, or leading zeros. Float any minus sign to position preceding the first nonblank character. Right justify data in the output field. If the data is shorter than the field, pad with blanks on the left. If the data is too long, strip leading blanks one at a time until data fits. If, after all blanks are removed, the data is still too long, strip trailing fractional zeros, one by one. Then, if necessary, round fractional digits, one place at a time, until the value fits in the output field. (Note that rounding may change the value of the integer part.) If the value still does not fit, issue error message. Assume value with length=10 1,234.510Δ moved to NUM3 field, length=10 ΔΔ1234.510 to NUM3 field, length=11 ΔΔΔ1234.510 to NUM3 field, length=6 1234.5 to NUM3 field, length=4 1235 to NUM3 field, length=3 ERROR Assume signed value, length=10 +1234.510Δ moved to NUM2 field, length=10 ΔΔΔ1234.51 to NUM2 field, length=3 ERROR Assume signed value, length=7 -12.10Δ moved to NUM2 field, length=7 Δ-12.10 to NUM2 field, length=3 -12 to NUM2 field, length=2 ERROR
IMPn	*Conversion and Movement*: Strip any decimal point, commas, sign, or leading zeros. Float any minus sign to the position preceding the first non-blank digit. Right justify data in the output field. If the data is shorter than the field, pad with blanks on the left. If the data is too long, strip leading blanks, one by one. If the data does not fit and only the fractional part remains, strip leading fractional zeros. (Trailing fractional zeros are never stripped from an IMPn field.) If the data still does not fit, issue error message. Assume a value, length=10 -123.0120ΔΔ moved to IMP4 field, length=10 ΔΔΔ-1230120 to IMP4 field, length=8 -1230120 to IMP4 field, length=7 ERROR Assume a value, length=7 .0120ΔΔ to IMP4 field, length=3 120 to IMP4 field, length=2 ERROR

DATA TYPE

You may change the output data type to a type other than CHAR. If, for example, the data type of the input item is NUM2, you can retain this data type by changing the output data type from CHAR to NUM2. This causes the value to be reformatted according to the standard rules for data type reformatting.

To illustrate, when a NUM2 item entered as 000123.12 is moved to a NUM2 field, leading zeros are replaced by blanks and it is right justified in the field. If moved to a CHAR field the value is moved to the output field exactly as entered. For another example, assume a date of type MDY is entered as MAR 6, 1978. If the output data type is left as CHAR, the date is written to the output field as MAR 6, 1978. If, on the other hand, the output data type is changed to MDY, the date is written as 03/06/78.

When you change a data type to one other than CHAR and different from the original data type, you must take care since not all data types are interchangeable, and those that are may require conversion of the data. Allowed changes are:

• Any type to character	No conversion — data is left exactly as it was entered.
• Numeric to numeric	Convert numeric value to conform to the destination type (DIGIT, NUM, NUMn, or IMPn).
• Date to Date	Convert date to dd/dd/dd; the exact order depends on the destination date type (MDY, DMY, or YMD).

FIELD FORMATTING

The remaining OUTPUT FIELD menu specifications affect the actual data in the field. These specifications (STRIP, JUSTIFY, SIGN, FILL, and CHECKDIGIT) are performed in addition to the standard formatting performed when data is moved from an input (batch) field to an output field.

The standard formatting is described in table 5-1. In general, the data is first converted to the destination data type, if necessary. Then all other formatting is performed. Note that the data is converted only if the output field type is specifically changed to a type other than CHAR.

REFSPEC performs all formatting (user-defined and standard) in the following order:

1. Convert data to destination data type (unless type is CHAR). If numeric type data does not fit in the output field after conversion, the output field will be set to all blanks.
2. STRIP (user-specified)
3. Move data to output field, justified left, and inserting check digit if specified, and padding with blanks as needed. If character type data does not fit, it is truncated on the right. If a data field does not fit, the output field is set to all blanks.

   Perform any CHECKDIGIT (user-specified) and/or JUSTIFY (user-specified).
4. SIGN (user-specified)
5. FILL (user-specified)

Note that step 3 includes data type formatting with data movement and user-specified justification and check digit insertion. These steps are performed simultaneously so that significant data is not lost due to justification. For example, if you justify data in a field to the left, the data is justified before movement; but if you justify data to the right, the justification is performed after the data is moved.

Any changes (truncation, conversion, and so forth) made to the data in the output field do not affect the original data in the input field.

STRIP

This option lets you remove a particular character or group of characters from data entered in the field. The following three options are provided:

STRIP ALL "characters"	Strips each occurrence of each specified character. The remaining characters are shifted left to fill the space created by stripping the specified characters.
STRIP LEADING "characters"	Strips each occurrence of each specified character only if it appears before any other non-blank characters in the field. Stripped characters are replaced by blanks. It is meaningless to strip leading blanks; if you want to shift data left, use JUSTIFY LEFT.
STRIP TRAILING "characters"	Strips each occurrence of each specified character only if it appears after all other non-blank characters. Stripped characters are replaced by blanks.

In the following examples using STRIP, both the input field and the output field are 12 characters long and are the same data type; a blank is shown by the character Δ.

Data Type	Input Value	Specification	Output Value
CHAR	ΔACBAXCBYZBB	none	ΔACBAXCBYZBB
		STRIP ALL "ABC"	ΔXYZΔΔΔΔΔΔΔΔ
		STRIP LEADING "ABC"	ΔΔΔΔΔXCBYZBB
		STRIP TRAILING "ABC"	ΔACBAXCBYZΔΔ
DIGIT	Δ1234567ΔΔΔΔ	none	ΔΔΔΔΔ1234567
		STRIP ALL "35"	ΔΔΔΔΔ12467ΔΔ
		STRIP LEADING "0"	ΔΔΔΔΔ1234567
		STRIP TRAILING "7"	ΔΔΔΔΔ123456Δ
NUM2	Δ12,345.67ΔΔ	none	ΔΔΔ-12345.67
		STRIP ALL "35"	ΔΔΔ-124.67ΔΔ
		STRIP LEADING "01"	ΔΔΔΔ-2345.67
		STRIP TRAILING "7"	ΔΔΔ-12345.6Δ
IMP2	Δ12,34567ΔΔΔ	none	ΔΔΔΔ-1234567
		STRIP ALL "35"	ΔΔΔΔ-12467ΔΔ
		STRIP LEADING "01"	ΔΔΔΔΔ-234567
		STRIP TRAILING "7"	ΔΔΔΔ-123456Δ

Note that in the last IMP2 example, stripping the last character moves the implied decimal to the position between "4" and "5".

JUSTIFY

This specification lets you move data to the right or left boundary of the output field or to center it in the field. These three options are specified as:

JUSTIFY RIGHT	Moves the data to the right until the last character in the output field is non-blank, padding with blanks on the left as the data is moved.
JUSTIFY LEFT	Moves data to the left until the first character in the output field is non-blank, padding with blanks on the right as the data is moved.
JUSTIFY CENTER	Positions the data so that there is an equal number of blanks to the right and left of the nonblank data. If the total number of blanks in the field is not even, the extra blank is on the right.

If JUSTIFY is not specified, numeric type data is justified right and date type data is justified left. No justification is performed on character type data.

The following examples illustrate the three types of justification. It is assumed the input and output data types are the same.

Data Type	Input Value	Specification	Output Value
CHAR	ΔΔABCDEFΔΔ	JUSTIFY R	ΔΔΔΔABCDEF
		JUSTIFY L	ABCDEFΔΔΔΔ
		JUSTIFY C	ΔΔABCDEFΔΔ
DATE	1/30/78ΔΔΔ	none	1/30/78ΔΔΔ
		JUSTIFY R	ΔΔ01/30/78
		JUSTIFY C	Δ01/30/78Δ
DIGIT NUM or IMP	ΔΔ123456ΔΔ	none	ΔΔΔΔ123456
		JUSTIFY L	123456ΔΔΔΔ
		JUSTIFY C	ΔΔ123456ΔΔ

SIGN

Wtih the SIGN specification, you can indicate where you want either a plus or minus sign placed in the output field. Any plus sign is removed automatically when a number is moved to the output field unless you specifically request that it be included with the PLUS option of SIGN. SIGN has the following options:

SIGN LEFT	The sign is placed in the first character position of the field. If this position contains a blank, the sign replaces it. Otherwise, data may be moved to the right of a leading zero stripped to accommodate the sign. If needed, a fractional digit is rounded. If the sign still does not fit, an error occurs and the field is set to blanks.
SIGN RIGHT	The sign is placed in the last character position in the field. As with SIGN LEFT, every attempt is made to fit the sign in the field, but if it results in too many characters, an error occurs and the field is set to blanks.
SIGN FLOAT	The sign is placed immediately preceding the first non-blank digit in the field. As with SIGN left, a leading zero may need to be stripped or a fractional digit rounded to accommodate the sign. If it still does not fit, an error occurs and the field is set to blanks.
SIGN ZONE	The sign is represented as an "overpunch" character in the last digit of the field. (See table 5-2 for the value of a zoned sign.) No movement of data is required for this option.
NO SIGN	Any sign in the input field is stripped from the field when it is moved to the output record.
PLUS	All the preceding specifications apply to either a minus or a plus sign. The default is to insert only minus signs. If you want to retain a plus sign, you must enter Y in the PLUS option as well as indicate where you want the sign positioned.

If SIGN is not specified, a minus sign is floated for numeric type data and any plus sign is stripped; character type data is written exactly as entered.

Table 5-2. Overpunch Character for Zoned Sign

Positive Values		**Negative Values**	
Signed Digit	**Character**	**Signed Digit**	**Character**
+0	{	–0	}
+1	A	–1	J
+2	B	–2	K
+3	C	–3	L
+4	D	–4	M
+5	E	–5	N
+6	F	–6	O
+7	G	–7	P
+8	H	–8	Q
+9	I	–9	R

The following examples illustrate use of the SIGN option. The input and output fields are assumed to be the same data type.

Data Type	Input Value	Specification	Output Value
NUMO	-123456ΔΔΔ	none	ΔΔΔ-123456
	+123456ΔΔΔ	none	ΔΔΔΔ123456
	-123456ΔΔΔ	NO SIGN	ΔΔΔΔ123456
	-1234567ΔΔ	SIGN ZONE	ΔΔΔ123456P
	+1234567ΔΔ	SIGN ZONE, PLUS	ΔΔΔ123456G
	-1234567ΔΔ	SIGN RIGHT	ΔΔ1234567-

FILL

This specification allows you to replace leading, trailing, or all blanks in a field by a particular character. The specified character may be any printable character. It may not be a non-printing control character, nor may it be more than one character.

If FILL is not specified, no default replacement is made.

FILL ALL character	Replaces all blanks in the data with the specified character.
FILL LEADING character	Replaces all leading blanks with the specified character.
FILL TRAILING character	Replaces all trailing blanks with the specified character.
	Defaults: none

You can specify both FILL LEADING and FILL TRAILING with no redundancy. But, you must not specify either FILL LEADING or FILL TRAILING with FILL ALL. Since the FILL ALL fills all the blanks, any other FILL specification for the field is diagnosed as an error.

The following examples illustrate use of the FILL option. The input and output data types are the same.

Data Type	Input Value	Specification	Output Value
CHAR	ΔΔABCΔDEFΔΔ	FILL ALL *	**ABC*DEF**
		FILL LEADING *	**ABCΔDEFΔΔ
		FILL TRAILING *	ΔΔABCΔDEF**
NUM2	ΔΔΔΔΔ123.75	FILL LEADING *	*****123.75

FILL LEADING and FILL ALL are not sensitive to the sign of a signed number. For example, the first set of specifications below produces a meaningless result because the sign is floated before the field is filled with zeros; the second set of specifications produces the desired result.

Input Value	Specification	Output Value
-ΔΔΔ999	FLOAT SIGN	
	FILL LEADING 0	000-999
-ΔΔΔ999	SIGN LEFT	-ΔΔΔ999
	FILL ALL 0	-000999

CHECK DIGIT

You can request that a check digit be added to the end of any digit or alphabetic value by entering 10 or 11 in the CHECKDIGIT option of the OUTPUT FIELD menu. The check digit is calculated by modulo 10 or modulo 11 depending on which you specify.

Check digits can be added only to fields that are type DIG or type CHAR and that contain only digits or letters of the alphabet.

If ADD CHECK DIGIT is not specified as "10" or "11", no check digit is added.

NOTE

> This specification adds a check digit. If you want to verify a number that contains a check digit, this can be requested in the original form design using FORMSPEC (refer to section IV).

If the data is right justified, the non-blank digits in the field are moved left one character position to make room for the check digit. Note that when a check digit is to be added to a field, the length of the output field must be specified as at least one character longer than the input field. (Field length is increased on the OUTPUT RECORD menu, not the OUTPUT FIELD menu.)

Refer to appendix D for a description of how check digits are calculated if modulo 10 or modulo 11 is specified.

Examples:

CHECKDIGIT 10	Calculate check digit according to modulo 10 formula and add it following rightmost non-blank digit.
CHECKDIGIT 11	Calculate check digit according to modulo 11 formula and add it following rightmost non-blank digit.

If the product generated by a CHECKDIGIT 11 calculation evaluates to 10, this is considered invalid and the following message is issued when REFORMAT is executed:

"Check digit is invalid for modules 11 calculation."

If the product generated by a CHECKDIGIT 11 calculation evaluates to 11, a zero is appended to the basic number.

RUNNING PROGRAM REFORMAT

Once the reformat specifications have been defined and are stored in a compiled reformat file, you can run program REFORMAT to actually reformat the data in the batch file and write it to the output file.

Program REFORMAT is not an interactive program. That is, it does not prompt for any information. It is usually run as a batch job, though it can be run from a terminal directly or as a "streamed" job.

Program REFORMAT needs the names of three files: the "reformat" file, that contains the reformat specifications, the "batch" file containing the data to be reformatted, and the "output" file to which the reformatted data is written. To specify these files, use MPE :FILE commands before running REFORMAT, as follows:

```
:FILE REFFILE=reformatfile
:FILE BATCH=batchfile
:FILE OUTFILE=output file
```

For example, assuming a reformat file named REF1, a batch file named BAT1, and an output file named OUT1, the following commands are needed to run REFORMAT:

```
:FILE REFFILE=REF1
:FILE BATCH=BAT1
:FILE OUTFILE=OUT1
:RUN REFORMAT.PUB.SYS
```

These file equations show the required files for running REFORMAT. You will also probably want to list the reformatted data and, if you are running REFORMAT from a terminal, you will want error messages listed on the line printer rather than on the terminal screen.

A list of the reformatted data is very important when you run REFORMAT the first time, so you can see whether your reformat specifications are doing what you expect. To list reformatted data, include the file TESTLIST in a file equation that sends the listing to the line printer:

```
:FILE TESTLIST;DEV=LP
```

Another important file, particularly when you are running in a session, is the error message file. This file REFLIST normally is written to $STDLIST. Since $STDLIST in a session is the terminal, you may want to equate this file to the line printer with the command:

```
:FILE REFLIST;DEV=LP
```

USING A USER-DEFINED COMMAND

These commands will run REFORMAT using the three files specified in the :FILE commands.

You may want to combine these commands into one user-defined command (UDC). You enter a UDC through the EDITOR program, as follows:

```
:EDITOR
/A
1  REFORMAT REFSPECS, DATA, OUTPUT=$STDLIST
2  FILE REFFILE=!REFSPECS
3  FILE BATCH=!DATA
4  FILE OUTFILE=!OUTPUT
5  FILE REFLIST;DEV=LP
6  FILE TESTLIST;DEV=LP
7  RUN REFORMAT.PUB.SYS
8  *
/K REF1
/E
:SETCATALOG REF1
```

The UDC definition is recorded by the SETCATALOG statement. Now, all you need to do in order to run REFORMAT, is to enter the following command:

```
:REFORMAT reformatfile, batchfile, outputfile
```

Suppose your reformat file is named REF1, your batch file is named BAT1, and your output file is OUT1, run REFORMAT as follows:

```
:REFORMAT REF1, BAT1, OUT1
```

STREAMING PROGRAM REFORMAT

You may also want to stream the REFORMAT program. You must first enter the commands to run the job through the EDITOR, and then run the :STREAM command to actually run the program.

For example:

```
:EDITOR
/A
1  !JOB USER.ACCOUNT
2  !FILE REFFILE=REF1
3  !FILE BATCH=BAT1
4  !FILE OUTFILE=OUT1
5  !FILE TESTLIST;DEV=LP
6  !FILE REFLIST;DEV=LP
7  !RUN REFORMAT.PUB.SYS
8  !EOJ
/K REFSTREM,UNN
/E
```

To stream this job, use the following command:

```
:STREAM REFSTREM
```

In many cases, a streamed job is part of a group of streamed jobs. When program REFORMAT is part of a series of program executions, you should precede the RUN REFORMAT command by a CONTINUE command. Otherwise, any fatal error in REFORMAT (such as an inappropriate file name) prevents subsequent programs from executing.

For example:

```
!JOB
!FILE . . .
!FILE . . .
!FILE . . .
!CONTINUE ◄──────────────── use this command so that,
!RUN REFORMAT.PUB.SYS◄──────in case of REFORMAT error,
!SPLGO MYPROG   ◄────────────subsequent programs will run
!EOJ
   .
   .
   .
```

USING THE V/3000 PROCEDURES

SECTION VI

OVERVIEW

A set of callable procedures is provided by V/3000. These procedures are used by the V/3000 Data Entry Program (ENTRY) to control data entry. They can also be used by any user-written program, either for data entry or for other terminal related applications. The V/3000 procedures manage the interface between a user program, the terminal, a forms file, the entered data, and, for data entry, the batch file to which entered data is written.

Note that appendix A contains listings of program ENTRY in each of the languages that use the V/3000 procedures. These listings provide useful examples of how to use these procedures for data entry.

MULTIPURPOSE

While the principal purpose of these procedures is for data entry, they can be extended to any application that wants to display forms on a terminal supported by V/3000. With the exception of the batch file management procedures that are used specifically for data entry, the V/3000 procedures can be used in conjunction with the forms specification program, FORMSPEC, for data base inquiry, data base update, or any other application that displays or collects data through a terminal form. Such applications can also make use of the input data validation capabilities provided with V/3000.

MULTILANGUAGE

The V/3000 procedures can be called from programs written in any of the languages: COBOL, FORTRAN, BASIC, and SPL. RPG programs have their own interface with terminals and forms (described in section VII) and do not call the V/3000 procedures directly. Each of the languages, COBOL, FORTRAN, BASIC, or SPL calls the V/3000 procedures with the same parameters, and these parameters are essentially the same type and size. So that the procedures can adjust to any peculiarities of the calling language, one parameter specifies the language of the calling program.

ERROR DETECTION

If a system or program error causes a procedure to fail, an error code is returned to the calling program. Once an error has been detected, subsequent procedure calls do not perform any functions until the program detects the error and performs its own error routine. In order for processing to continue after an error is detected, the calling program must reset the error flags. This method of error handling means that procedure errors do not cause unexpected program termination.

Table 6-1 lists the V/3000 procedures in alphabetic order and summarizes their functions. A full description of each procedure appears later in this section.

Table 6-1. Summary of V/3000 Procedures

PROCEDURE	FUNCTION
VCLOSEBATCH	Closes batch file.
VCLOSEFORMF	Closes forms file.
VCLOSETERM	Closes terminal file.
VERRMSG	Returns message associated with error code.
VFIELDEDITS	Edits field data and performs other field processing.
VFINISHFORM	Performs final processing specified for form.
VGETBUFFER	Reads contents of data buffer into user program.
VGETFIELD	Reads field from data buffer into user program.
VGETFILEINFO	Returns forms file information.
VGETFORMINFO	Returns form information.
VGETFIELDINFO	Returns field information.
VGETKEYLABELS	Returns global function key labels.
VGETNEXTFORM	Reads next form into form definition area of memory; window and data buffer are not affected.
VGETtype	Reads field from data buffer to user program, converting data to specified type.
VINITFORM	Sets data buffer to initial values for form.
VLOADFORMS	Loads forms named in FORMS into terminal workspace.
VOPENBATCH	Opens batch file for processing.
VOPENFORMF	Opens forms file for processing.
VOPENTERM	Opens terminal file for processing.
VPOSTBATCH	Posts end of file mark in batch file after last record referenced.
VPRINTFORM	Prints current form and data on offline list device.
VPUTBUFFER	Writes data from user program to data buffer.
VPUTFIELD	Writes data from user program to field in data buffer.
VPUTtype	Writes data of specified type from user program to data buffer, converting data to ASCII.
VPUTWINDOW	Writes message from user program to window area in memory for later display.
VREADBATCH	Reads record from batch file into data buffer.
VREADFIELDS	Reads input from terminal into data buffer.
VSETERROR	Sets error flag for data field in error; and moves error message to window area.
VSETKEYLABEL	Temporarily sets a new label for a function key.
VSETKEYLABELS	Temporarily sets new labels for function keys.
VSHOWFORM	Updates terminal screen, merging the current form, any data in buffer, and any message in window.
VUNLOADFORM	Unloads a form from the terminal workspace.
VWRITEBATCH	Writes data from data buffer to batch file.

HOW PROCEDURES ARE USED

The V/3000 procedures control the interface between forms stored in a forms file, a terminal screen, data entered on the screen, and the user program. Figure 6-1 illustrates the relation between the V/3000 procedures, a terminal, forms file, user program, batch file, and the buffer areas in memory used by the procedures.

The procedures use a buffer area in memory for the form definition and another buffer area for the data. A third area in memory is used as a buffer for the "window" area, the line on the form to which error and other messages are sent.

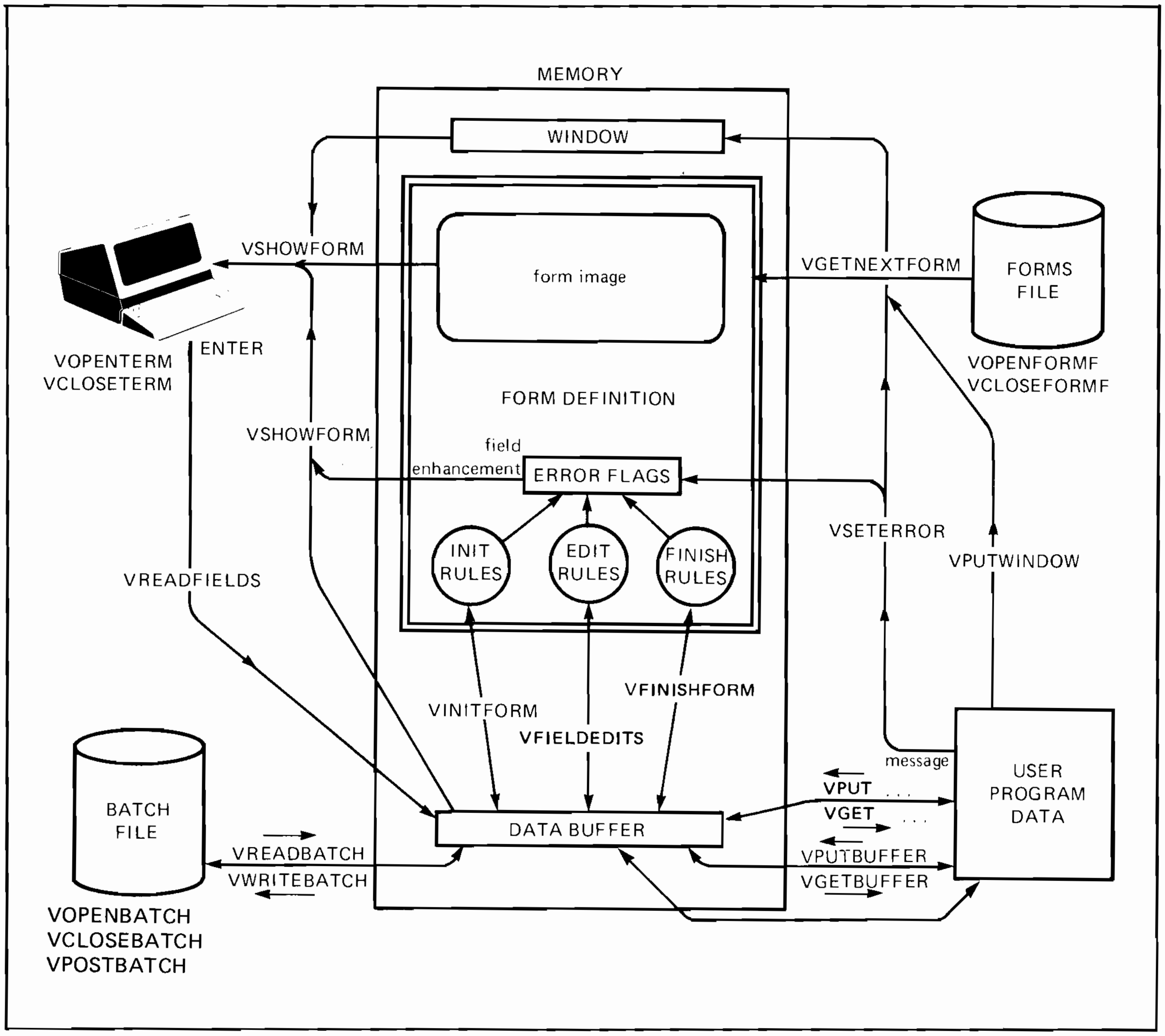


Figure 6-1. Operation of V/3000 Procedures

FORM DEFINITION AREA

A form displayed on the terminal screen consists of protected areas (headings, labels, titles, display-only fields) and the unprotected areas (fields) into which data can be entered by an operator.

Any form written to the terminal screen by VSHOWFORM is already resident in memory as a "form image". This form image contains the description of all the protected areas on the form, except for "display only" fields. It also contains the visual enhancements for all fields, protected and unprotected. Associated with each form is a set of data specifications — field attributes, edits, and processing specifications defined by FORMSPEC. These specifications are read from the forms file with the form by VGETNEXTFORM, and are memory resident with the form image during execution of the form.

DATA BUFFER AREA

Besides the form definition, there is a data buffer area in memory that contains data for all the fields (unprotected and display only) defined for the form. These fields reside in the buffer in the order they are defined on the screen, from left-to-right, top-to-bottom. When the ENTER key is pressed at the terminal, VREADFIELDS transfers the operator-entered data from the screen to the data buffer. Before data is entered, if there are any initializations, VINITFORM sets the appropriate fields in the buffer to initial values. In the field phase, VFIELDEDITS verifies and possibly modifies the operator-entered data in the data buffer according to any edit specifications defined for the fields. VFINISHFORM performs any final form modifications specified in the "finish" phase. Data for display-only fields may be written to the buffer by VINITFORM, VFIELDEDITS, or VFINISHFORM. Any changes to the data buffer are displayed at the terminal by the next VSHOWFORM.

During data collection, data in the data buffer is written to the batch file by VWRITEBATCH. The data in the batch file can be read from the batch file to the data buffer by VREADBATCH. Typically, data is read from the batch file for display at the terminal during browse and modify operations.

The data buffer can be read by a user program with VGETBUFFER, or a single field can be read with VGETFIELD or VGETtype. Conversely, a program can write data to the buffer with VPUTBUFFER or to a single field in the buffer with VPUTFIELD or VPUTtype.

WINDOW AREA

If the field editing specifications detect an error in operator entry or data movement, an error flag is set. VERRMSG returns the message associated with the error, and VPUTWINDOW puts the message in the window. If an error is detected by a user program, VSETERROR sets the error flag, and also puts any message associated with the error in the window. Any non-error messages generated by a user program are put in the window by VPUTWINDOW. A subsequent call to VSHOWFORM displays the contents of the window at the terminal.

ERROR HANDLING

There are basically two types of error that can occur as a result of calling V/3000 procedures. The first type consists of errors in the procedure call itself or in an attempt to access a file used by the called procedure. The second type are errors detected by editing data entered into FORMSPEC forms. These two error types are handled differently by V/3000.

PROCEDURE CALL OR FILE ERRORS

If a call to a V/3000 procedure causes an error so that the procedure cannot be executed correctly, or if an MPE file error occurs as a result of an attempt to access a file with a V/3000 procedure, the COMAREA word CSTATUS is set to a non-zero value. In addition, the number associated with an MPE file error is stored in the COMAREA word, FILERRNUM.

When CSTATUS is not zero, any subsequent V/3000 procedures called by the program return to the program without executing. For this reason, the calling program is not required to check CSTATUS after each call. However, it is good programming practice to check CSTATUS, report the error, and then reset CSTATUS.

EDITING ERRORS

Field processing may be specified in the forms description and checked by VFIELDEDITS, VINITFORM, or VFINISHFORM; or editing may be provided by user routines in the application program.

Each field in a FORMSPEC form has an error flag associated with it. When one of the V/3000 procedures that performs field processing (VFIELDEDITS, VINITFORM, or VFINISHFORM) detects a field error, it sets the error flag for that field. It also increments NUMERRS, the word in COMAREA that contains the total number of fields with errors in each form. If a user-provided editing routine detects a field error, the program must call the V/3000 procedure VSETERROR in order to set the field error flag and increment NUMERRS. CSTATUS is not set when an editing error is detected and subsequent procedures may be executed.

Should new data be written to a field in the data buffer that had an error, the error flag for the field is cleared and NUMERRS is decremented. VPUTBUFFER, VPUTFIELD, or VPUTtype are the procedures that can correct field errors and decrement NUMERRS. The procedure VREADFIELDS clears NUMERRS to zero when it reads new data into the buffer from the terminal.

ERROR MESSAGES

Messages associated with all V/3000-detected errors can be retrieved by a call to VERRMSG. This procedure uses the error message file whose MPE file number is kept in the COMAREA word, ERRFILENUM. The error message file contains internal error numbers linked to particular field errors and may be a general V/3000 message (see appendix B), or it may be a custom message specified during forms design by FORMSPEC. In either case, the message is returned to the calling program by VERRMSG.

VERRMSG determines the type of the error by examining CSTATUS and NUMERRS. If CSTATUS is not zero, its value indicates a particular procedure call error. If CSTATUS is zero and NUMERRS is set, VERRMSG knows the error is an editing error and uses internal values to locate the error in the error message file. If an editing error is detected by a user routine, the program must provide its own message when it calls VSETERROR to set the error flag for the field.

CALLING V/3000 PROCEDURES

In order to provide consistency between calls from different programming languages, the following rules apply to all parameters:

- Parameters are passed by reference; this means that a literal value cannot be used as a parameter.
- No condition codes are returned; the status of the call is returned in a status word included as part of the COMAREA parameter specified in every procedure call.
- Return type procedures are not allowed; any values returned by the procedure are sent to the COMAREA or to a passed parameter.
- No optional parameters are allowed.

The following examples show the format of calls to the V/3000 procedures from each language.

Language	Procedure Call Format
COBOL	CALL "procedurename" USING parameter1[, parameter2] . . .
FORTRAN	CALL procedurename (parameter1[,parameter2] . . .)
BASIC	label CALL procedurename(parameter1[,parameter2] . . .)
SPL	procedurename(parameter1 [,parameter2] . . .);

where:

procedurename	identifies the procedure being called.
parameter	at least one parameter is required for each procedure; the particular parameters are listed in the formats for the individual procedure descriptions. Note that when more than one parameter is specified, each is separated by a comma; and for COBOL calls by a comma and a space.

PARAMETER TYPES

The data types that are allowed in V/3000 procedures are shown in table 6-2. Note that not all types are allowed for all languages.

Table 6-2. Data Types Allowed for Various Languages

DATA TYPE	LANGUAGE			
	COBOL	FORTRAN	BASIC	SPL
Character	DISPLAY PIC X(n)	CHARACTER	STRING	BYTE ARRAY
1-Word Integer	COMP PIC S9 thru PIC S9(4)	INTEGER	INTEGER	INTEGER
Unsigned 1-Word Integer	COMP PIC 9 thru PIC 9(4)	LOGICAL	INTEGER (with value ≤32767)	LOGICAL
2-Word Integer	COMP PIC S9(5) thru PIC S9(9)	DOUBLE INTEGER	INTEGER INTEGER*	DOUBLE INTEGER
Real	–	REAL	REAL	REAL
Long	–	LONG	LONG	LONG

*In BASIC, a double integer can be represented by two consecutive integers; the first contains the high-order digits of values above 32767 or is zero, the second contains the low order digits of values above 32767 or the entire value up to 32767.

The V/3000 parameters use only data types that are available in all four languages: character, integer, logical, and double integer. The remaining data types, real and long, are available to programs in all languages except COBOL for transferring or converting data; they are not used as parameters in the V/3000 calls.

Each parameter is described according to its generic type (character, integer, logical, or double integer). This table is provided for those languages that do not call their data types by these particular names. For example, if you are coding in COBOL and a parameter is specified as logical, you can determine from this table that it is an unsigned computational item that uses from 1 to 4 digits.

COMMUNICATION AREA

Every program that calls V/3000 procedures must allocate a data area in the program for communication with the procedures. This area (called the COMAREA) is the first, and often the only, parameter in every call to a V/3000 procedure. Table 6-3 briefly outlines the contents of this communication area. It is essential to successful operation of your program that the COMAREA be defined exactly as shown in this table.

Table 6-3. Outline of COMAREA Contents

DATA TYPE	(SPL) WORD	OFFSET	NAME	FUNCTION
Integer	1	0	CSTATUS	status, error returns
	2	1	LANGUAGE	language of calling programs
	3	2	COMAREALEN	length (in words) or COMAREA
	4	3	USRBUFLEN	COMAREA extension length (BASIC)
	5	4	CMODE	current mode (collect or browse)
	6	5	LASTKEY	code of last key pressed
	7	6	NUMERRS	number of errors in current form
	8	7	WINDOWENH	code for window enhancement
	9	8	MULTIUSAGE	next form flag (parent/son)
	10	9	LABEL'OPTION	function key label indicator
Character	11	10	CFNAME	current form name
	19	18	NFNAME	next form name
Integer	27	26	REPEATAPP	repeat flag (freeze/append)
	28	27	FREEZAPP	next form flag (freeze/append)
	29	28	CFNUMLINES	number of lines in current form
	30	29	DBUFLEN	data buffer length (in characters)
	31	30	Reserved for system use.	
	32	31	LOOK'AHEAD	form preload indicator
	39	38	FORM'STORE'SIZE	number of forms in buffer
Logical	33	32	DELETEFLAG	delete current batch record
	34	33	SHOWCONTROL	control flags for VSHOWFORM
	35	34	Reserved for system use	
Integer	36	35	PRINTFILNUM	file # of forms file print file
	37	36	FILERRNUM	MPE file error number from FCHECK
	38	37	ERRFILENUM	MPE file number, error message file
	39-42	38-41	Reserved for system use	
Double Integer	43/44	42/43	NUMRECS	number of records in batch file
	45/46	44/45	RECNUM	record # of current batch record
	47-48	46-47	Reserved for system use	
Logical	49	48	FILEN	MPE file number of terminal
	50-54	49-53	Reserved for system use	

Table 6-3. Outline of COMAREA Contents (continued)

DATA TYPE	(SPL) WORD	OFFSET	NAME	FUNCTION
Logical	55	54	RETRIES*	max number of retries
	56	55	OPTIONS*	suppress msgs. and autoread
	57	56	ENVIRON	term environment: Sys LDEV
	58	57	USER'TIME*	user defined time out length
	59	58	IDENTIFIER	type of terminal
	60	59	LAB'INFO	num of function keys; length
	61-70	60-69	Reserved for system use	
THIS AREA ONLY REFERENCED WHEN USING HP 3075/6 TERMINALS:				
Integer	71	70	NUM'FLDS	num of fields on current form
	72	71	SPLIT'PAUSE	length of pause (in seconds)
	73	72	LEFT'MODULE	type of terminal options
	74	73	RIGHT'MODULE	type of terminal options
	75	74	KEYBOARD	type of keyboard
	76	75	DISPLAY	type of display
	77	76	KEYBOARD'OVER	whether to override or not
Character	78	77	ERROR'LIGHT	error detection light
Logical	79-80	78-79	USER'LIGHTS'ON	lights on/off indicator
	81-86	80-85	Reserved for system use	

* *not supported on the 3075/6 terminals.*

The location of each item in COMAREA is given as a word offset in the table; for SPL programs the word offset starts with zero, for all other programs it starts with 1.

The COMAREA must be at least 60 words long (120 characters or bytes). For BASIC programs, this area must be extended to include space for the form and data buffers and for internal tables. For non-BASIC programs, the DL area is used for these buffers and internal tables, and the communication area need not be extended. However, non-BASIC programs must be careful not to use the DL area for other purposes when using V/3000.

The COMAREA items listed in table 6-3 are defined as follows:

CSTATUS	Integer to which the procedure status is returned. Set to zero if the call is successful; to a non-zero value if an error occurs. If the error is an MPE file error, a file error number is also returned to FILERRNUM. (Refer to appendix B for a list of the error codes that may be returned to CSTATUS with their meaning.) It is up to the user to provide error-handling routines and to reset CSTATUS.
LANGUAGE	Integer that indicates the language of the calling program: 0 = COBOL 1 = BASIC 2 = FORTRAN 3 = SPL LANGUAGE must be set by the calling program before any procedure is called.
COMAREALEN	Integer that indicates the total length of COMAREA (use 60 for HP 264x and HP 262x terminals; 85 for HP 307x terminals). COMAREALEN should be specified for each COMAREA to simplify future changes to the length of COMAREA.
USRBUFLEN	For BASIC programs only, USRBUFLEN must be set to the number of words to be appended to COMAREA for the form and data buffers and internal tables. (Note that the number of words required for this extension is printed when a form is listed through FORMSPEC.) USRBUFLEN need not be set by other languages since V/3000 automatically uses the DL area for buffers and internal tables. USRBUFLEN does not include the COMAREA length specified in COMAREALEN, but it must be contiguous to and immediately follow COMAREA.
CMODE	Indicates whether current mode of data entry is collect 0 = Collect Mode 1 = Browse Mode CMODE must be set before calling VGETNEXTFORM so that it can determine whether to save form sequence for $RETURN.
LASTKEY	Integer set to a number between -1 and 26 by VREADFIELDS to indicate last function key pressed at the terminal.
NUMERRS	Integer set to the number of errors found when a form is edited according to FORMSPEC edit specifications.

WINDOWENH

Integer in which the right character (byte) contains an ASCII code for the "window line" enhancement. The code specifies any combination of enhancements according to the following table; zero indicates no enhancement.

	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Half-Bright									x	x	x	x	x	x	x	x
Underline					x	x	x	x					x	x	x	x
Inverse Video			x	x			x	x			x	x			x	x
Blinking		x		x		x		x		x		x		x		x
Stop enhancement	x															

For example, to indicate Half-Bright, Inverse Video, WINDOWENH is set to "J".

MULTIUSAGE

Integer that indicates whether the next form is in the same family as the previous form, but not the parent form.

1 = Son or brother to previous form
0 = Otherwise

LABEL'OPTION

Integer that indicates whether function key labels are to be used.

0 - Default function key labels are used.
1 - User-defined labels are to be used (space will be allocated in the user stack by VOPENFORMF to store the labels).

LABEL'OPTION must be set prior to a call to VOPENFORMF so that it can be determined whether to display user-defined function key labels or not.

CFNAME

15-character array containing the name of the current form. Required and updated by VGETNEXTFORM.

NFNAME

15-character array containing the name of the next form. Required by VGETNEXTFORM and updated when necessary by VGETNEXTFORM. NFNAME may be set by user program.

NOTE

All COMAREA entries start on word boundaries. Thus, the 15-character items CFNAME and NFNAME are each followed by one filler character that is not part of the name.

REPEATAPP

Integer that indicates whether the current form is a repeating form and, if so, whether it is to be appended to itself:

0 = Normal sequence, neither repeat nor append
1 = Repeat current form
2 = Repeat current form, appending it to itself

FREEZAPP	Integer that indicates whether the screen is to be cleared when the next form is displayed, or whether next form is to be appended to current form, and, if appended, whether current form is to be frozen on the screen. If the current form is frozen, it remains on the screen and the next form is rolled off when the screen is full; otherwise, the current form is rolled off the screen when the screen is full. FREEZAPP is specified as: 0 = Clear screen, neither freeze nor append current form 1 = Append next form to current form 2 = Freeze current form and append next form to it
CFNUMLINES	Integer that specifies the number of lines in the current form. CFNUMLINES is required by VPRINTFORM in order to print the form offline.
DBUFLEN	Integer that specifies the number of characters in the data buffer for the current form. Set by VGETNEXTFORM, this length is the sum of all the concatenated data fields in the form, including any display-only fields.
LOOK'AHEAD	Integer that indicates whether preloading of forms is to occur or not. 0 - ON--preload the forms. 1 - OFF--do not preload the forms. Note LOOK'AHEAD applies to the terminals that have local form storage capabilities.
DELETEFLAG	Logical flag that indicates whether the current batch record has been or is to be deleted. DELETEFLAG is returned by VREADBATCH and used by VWRITEBATCH: FALSE (all zeros) = Current batch record not deleted TRUE (all 1's) = Delete current batch record
SHOWCONTROL	Logical word used to override VSHOWFORM optimizations. It allows user to display a form, data buffer enhancements, or the window line separately by setting the following bits:

bit	15	= 1		Write form only to terminal
	14	= 1		Write data and enhancements from data buffer
	13	= 1		Write window line only
	12	=	0:	STOP after presenting a form without fields.
			1:	Do not STOP after presenting a form without fields.
	11	=	0:	Do not put a right closing bracket (]) on all input fields.
			1:	Put a right closing bracket (]) on all input fields.

PRINTFILNUM	MPE file number to which a form is printed by VPRINTFORM.
FILERRNUM	MPE file error number (FCHECK number) returned by V/3000 procedures when an MPE file error occurs. (Refer to appendix B for a list of the errors that return a number to FILERRNUM.)
ERRFILENUM	MPE file number of the error message file used by VERRMSG.

FORM'STORE'SIZE	Integer that indicates the number of forms to be allowed in the form store buffer. -1 - Workspace configuration under user control. 0 - No local form storage. 1 . . 4 - One to four forms can be stored locally. FORM'STORE'SIZE allows the user to control the amount of stack space used to control local form storage.
NUMRECS	Double integer that contains the number of non-deleted records in the current batch file.
RECNUM	Double integer set to the current record number in the batch file. (Note that record numbers start with zero.) RECNUM must be set by the program before writing to or reading from the batch file. It is used by the VREADBATCH and VWRITEBATCH procedures.
FILEN	MPE file number used to identify the terminal.
RETRIES	Maximum number of retries. value = 0 use default value (4 retries) value > 0 use this value as maximum value < 0 do not perform any retries
OPTIONS	Terminal control options: bits: 0 - 8 reserved for system use 9 - 10 01=enables ENTER/FCN key timeout in VREADFIELDS. 11 or 00=disables ENTER/FCN KEY timeout in VREADFIELDS. (default) 13 - 14 01=enables AUTOREAD in VREADFIELDS. 11 or 00=disables AUTOREAD. (default) 15 0=display mode message. (default) 1=suppress mode message.
ENVIRON	First byte is the logical device number of the terminal. The remaining byte is reserved for system use.
USER'TIME	If enabled, the value in this word is used as the number of seconds to wait for the ENTER key (or function key) to be pressed.
IDENTIFIER	V/3000 identifies the terminal type being used.
LAB'INFO	First byte is the number of labels the terminal supports. Second byte is the length of labels (characters).

In addition to the above COMAREA descriptions, the HP 3075/6 terminals use the following.

SPLIT'PAUSE	Length of time in seconds to pause between the presentation of lines of text on the single line alpha display. Default is 3 seconds. -1 - wait for user to hit key 0 - do not pause >0 - pause specified number of seconds

LEFT'MODULE	MPE determines which, if any, module is present. 0 - no module 1 - printer 2 - multifunction reader 3 - RS232 interface 4 - typeV badge reader 5 - magstripe reader 6 - bar code reader 7 - HP-IB interface Note V/3000 does not communicate with 3 and 7.
RIGHT'MODULE	MPE determines which, if any, module is present. 0 - no module 1 - printer 2 - multifunction reader 3 - RS232 interface 4 - typeV badge reader 5 - magstripe reader 6 - bar code reader 7 - HP-IB interface Note V/3000 does not communicate with 3 and 7.
KEYBOARD	Type of keyboard used with terminal. 0 - HP 3077 - no keyboard. 1 - standard keyboard (12 function keys with values of -1,0,17 . . . 26). 2 - alphanumeric keyboard (28 function keys with values of -1,0 . . . 26).
DISPLAY	Indicates the terminal type of display. 0 - numeric display 1 - alphanumeric display 2 - mini-CRT display V/3000 does not support the terminal which has numeric display.
KEYBOARD'OVER	Integer determining whether to override input on the keyboard or not. Default is 0. -1 - Override and enable the keyboard without regard to forms design. 0 - Do not override. Allow input from the devices specified during the form design (in the PROCESSING SPECIFICATION area).
ERROR'LIGHT	Indicates which light to be turned on when an error is detected. The default is "E", but this can be changed to "@" or any letter from "A" to "P". The second byte is reserved for system use.

LIGHTS'ON	Two integers to indicate whether or not lights are to be turned on during run time. The turning on of lights here does not affect the lights turned on during form design (in the PROCESS SPECIFICATION area of the form).

0 - OFF
1 - ON

The default value is OFF, but this may be changed as follows:

A "1" in bit position	0, turns light of key	"@" ON
	1	"A" ON
	2	"B" ON
	. . .	. . .
	14	"N" ON
	15	"O" ON
WORD 2 bit position	0	"P" ON

(the remaining bits are reserved for system use).

NOTE

Before calling the first V/3000 procedure, you should initialize the entire COMAREA to zero. Do not change COMAREA values between calls except under documented conditions. Then, if you are coding in a language other than COBOL, you must set LANGUAGE to the code for the language you are using. If you are coding in BASIC, you must set USRBUFLEN to the number of words needed for the COMAREA extension, and if your forms file is non-KSAM, you must add 1300 words to the extension length, or 500 words if it is a fast forms file. Finally, in order to provide for possible future extensions to COMAREA, set COMAREALEN to its current total length.

PROCEDURE DESCRIPTIONS

The procedures in the following pages are described in alphabetic order for easy reference. However, this is not the order in which they are normally used. Table 6-4 is provided in order to group the procedures according to the functions they perform. Note that terminal access is performed by only six procedures: VOPENTERM, VCLOSETERM, VSHOWFORM, VREADFIELDS, VLOADFORMS, and VUNLOADFORMS. The remaining procedures interact with the form definition, data buffer, and window areas of memory.

Table 6-4. Procedures by Function Group

FUNCTION	PROCEDURE	
Access to Terminal	VOPENTERM VCLOSETERM VSHOWFORM	VREADFIELDS VLOADFORMS VUNLOADFORM
Access to Forms File	VOPENFORMF VCLOSEFORMF VGETNEXTFORM VPRINTFORM	VGETFILEINFO VGETFORMINFO VGETFIELDINFO
Data Processing	VINITFORM VFIELDEDITS VFINISHFORM	
Data Entry	VOPENBATCH VCLOSEBATCH VWRITEBATCH	VREADBATCH VPOSTBATCH
Programmatic Access to Data	VGETBUFFER VPUTBUFFER VGETFIELD	VPUTFIELD VGETtype VPUTtype
Access to Error/Status Window, Function Key Label	VSETERROR VPUTWINDOW VERRMSG	VGETKEYLABELS VSETKEYLABELS VSETKEYLABEL

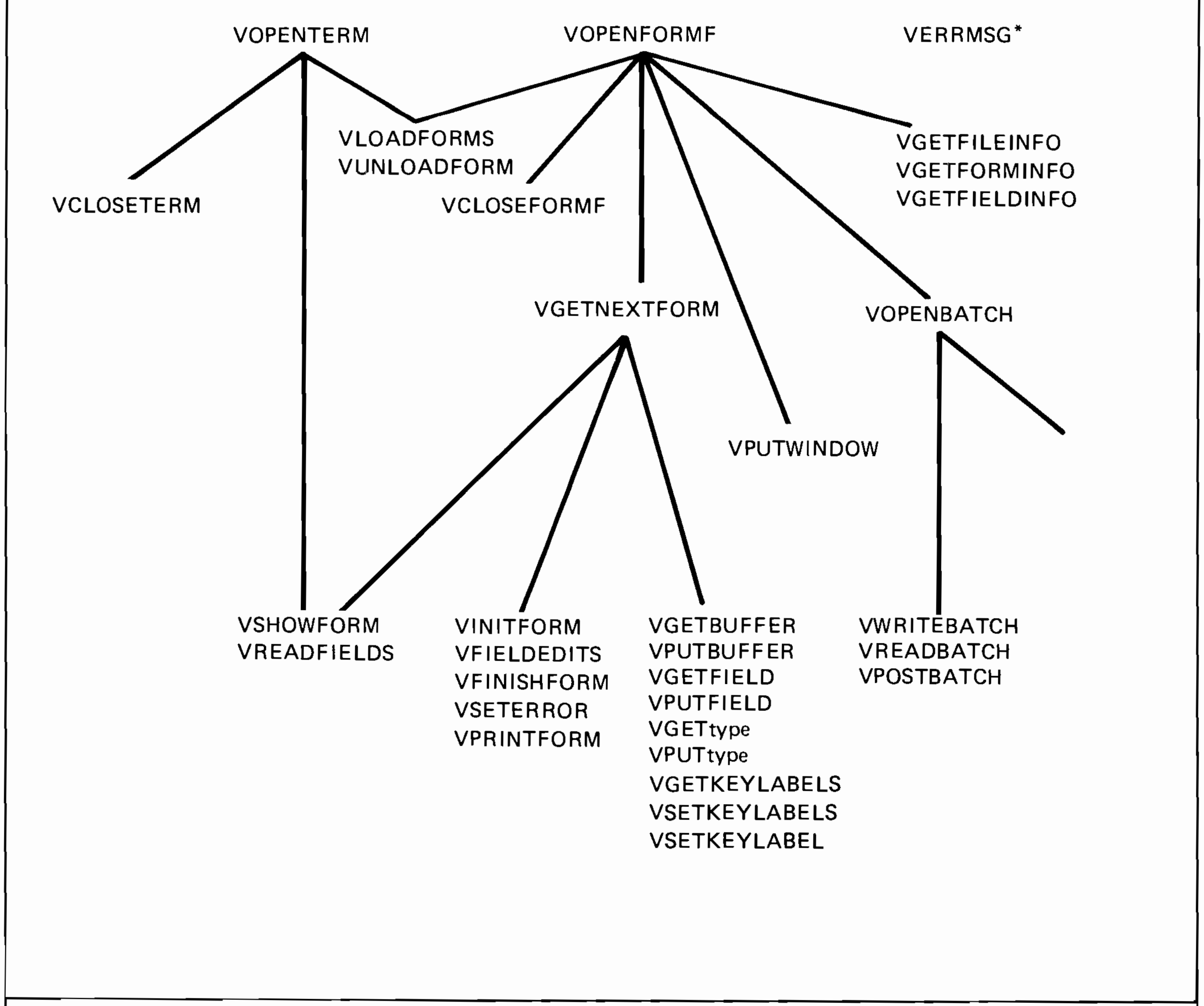
All procedures require that the CSTATUS, LANGUAGE, and COMAREALEN items in COMAREA be set before the procedure is called. LANGUAGE and COMAREALEN only need to be set once before the first procedure is called since they are not modified by any subsequent procedures. CSTATUS, however, should be reset before calling any procedure after an error occurs, except for the procedure VERRMSG which uses CSTATUS in retrieving the associated error message.

No running examples are provided with the procedure descriptions. However, appendix A contains a tested data entry program in each of the languages, COBOL, FORTRAN, BASIC, RPG, and SPL. This sample program uses many of the procedures described in this section.

DEPENDENCY BETWEEN PROCEDURES

Certain procedures must be called before other procedures can be executed. For example, the terminal file must be opened before any other procedures are called if prompts are to be sent to the terminal. Each of the three files, terminal, forms, and batch must be opened before operations can be performed on these files and before the files can be closed.

Figure 6-2 illustrates the standard dependencies among procedures.



*VERRMSG does not require any previous procedure calls

Figure 6-2. Procedure Dependencies

VCLOSEBATCH

Closes an open batch file.

```
CALL "VCLOSEBATCH" USING COMAREA
```

PARAMETERS

comarea Must be name of COMAREA specified in VOPENBATCH call that opened this file. If not already set, the following COMAREA fields must be set before calling VCLOSEBATCH:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VCLOSEBATCH may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

The open batch file identified by COMAREA is closed when this procedure is called.

EXAMPLES

COBOL:	CALL "VCLOSEBATCH" USING COMAREA.
BASIC:	200 CALL VCLOSEBATCH(C(*))
FORTRAN:	CALL VCLOSEBATCH(COMARE)
SPL:	VCLOSEBATCH(COMAREA);

VCLOSEFORMF

Closes an open forms file. Once closed, the forms file is not available for further processing.

```
CALL "VCLOSEFORMF" USING COMAREA
```

PARAMETERS

comarea — Must be COMAREA named when forms file was opened by VOPENFORMF. If not already set, the following COMAREA fields must be set before calling VCLOSEFORMF:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VCLOSEFORMF may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

The open forms file is closed when this procedure is executed.

EXAMPLES

COBOL:	CALL "VCLOSEFORMF" USING COMAREA.
BASIC:	200 CALL VCLOSEFORMF(C(*))
FORTRAN:	CALL VCLOSEFORMF(COMARE)
SPL:	VCLOSEFORMF(COMAREA);

VCLOSETERM

Closes an open terminal file.

```
CALL "VCLOSETERM" USING COMAREA
```

PARAMETERS

comarea	Must be COMAREA named when file was opened by VOPENTERM. If not already set, the following COMAREA fields must be set before calling VCLOSETERM:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VCLOSETERM may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

The terminal file opened with the specified COMAREA is closed when this procedure is executed.

EXAMPLES

COBOL:	CALL "VCLOSETERM" USING COMAREA.
BASIC:	200 CALL VCLOSETERM(C(*))
FORTRAN:	CALL VCLOSETERM(COMARE)
SPL:	VCLOSETERM(COMAREA);

VERRMSG

Returns a message corresponding to the error number of an edit error or a procedure call error.

```
CALL "VERRMSG" USING COMAREA, BUFFER, BUFLEN, ACTUALEN
```

PARAMETERS

comarea The following COMAREA fields should be set before calling VERRMSG:

LANGUAGE	Set to code identifying language of calling program.
COMAREALEN	Set to total number of words in COMAREA.
ERRFILENUM	Contains MPE file number of V/3000 error message file; should be initialized to zero so that VERRMSG can open the error message file.

(Note that CSTATUS should not be cleared before calling this procedure.)

VERRMSG may set the following COMAREA fields:

ERRFILENUM	If initialized to zero, VERRMSG opens V/3000 error message file and sets ERRFILENUM to MPE file number of the opened file.

buffer Character string in user program to which message is returned by VERRMSG. In order to contain the message, this buffer must be defined as at least 72 characters long.

buflen Integer variable set by user program to length of buffer.

actualen Integer to which VERRMSG returns the number of characters in the message sent to buffer.

If an error occurs in a procedure call or is detected by a V/3000 edit, a call to VERRMSG returns the message associated with the error. For a procedure call error, CSTATUS is set to a non-zero value. If CSTATUS indicates an error, VERRMSG returns the text explaining the type and cause of the error. If VINITFORM, VFIELDEDITS, or VFINISHFORM detect an error, CSTATUS is set to zero and NUMERRS is set to a non-zero value. If NUMERRS is set, VERRMSG returns a custom error message if there is one; otherwise, it returns the V/3000 error message associated with the error number in the V/3000 error message file.

The message describing the error is returned to the user program in the area defined by the buffer parameter. The message length can be no longer than buflen; the actual length of the message is returned in actualen. You can then call VPUTWINDOW to move this message to the window area of memory for later display at the terminal.

ERRFILENUM contains the MPE file number of the V/3000 error message file. This file contains the error numbers and their associated messages for both procedure call and edit errors. It does not contain custom error messages, which are retrieved by VERRMSG through the form definition. VERRMSG opens this file if ERRFILENUM equals zero the first time VERRMSG is called; it then sets ERRFILENUM to the MPE file number of the file.

EXAMPLES

Assume that FILERRNUM contains an MPE file error code, the following calls to VERRMSG return to your program, a message describing the error and the length of this message in characters.

COBOL:	CALL "VERRMSG" USING COMAREA, BUFFER, BUFLEN, MSGLEN.
BASIC:	225 CALL VERRMSG(C1(*),M$,L,M)
FORTRAN:	CALL VERRMSG(COMARE,BUFFER,BUFLEN,MSGLEN)
SPL:	VERRMSG(COMAREA,BUFFER,BUFLEN,MSGLEN);

VFIELDEDITS

Edits data entered in each field of form and, if indicated, modifies data in the data buffer. If necessary, sets error flags.

```
CALL "VFIELDEDITS" USING COMAREA.
```

PARAMETERS

comarea Must be COMAREA name specified when forms file was opened. If not set already, the following fields must be set before calling VEDITFIELDS:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VFIELDEDITS may set the following COMAREA fields:

NUMERRS	Set to total number of fields in which errors were detected.
CSTATUS	Set to non-zero value if call unsuccessful.
NFNAME	Set to new next form name if name changed by processing specifications.
REPEATAPP	Set to new repeat code if code changed by processing specifications.
FREEZAPP	Set to new freeze code if code changed by processing specifications.

This procedure checks each field into which data is entered. It checks that the data type and field type are correct, and if any special processing specifications were defined for this field, it checks that the data conforms to these specifications. If any data formatting or data movement was specified for a field, VFIELDEDITS performs these functions on the data in the data buffer.

For each field that does not pass the edit checks, VFIELDEDITS sets an error flag. These error flags cannot be accessed directly by an application program, but are used by the following V/3000 procedures:

VGETNEXTFORM	initializes all the error flags to zero.
VFIELDEDITS, VINITFORM, VFINISHFORM	each may set field error flags on, increment NUMERRS, and set an internal error number used by VERRMSG.
VSHOWFORM	enhances the fields whose error flags are set.
VREADFIELDS	resets the flags to zero after displaying the form.

After setting error flags for all fields with errors, VFIELDEDITS saves the error number for the first field with an error. (Fields are counted in screen order, starting at the top left and moving left to right, then top to bottom.)

If requested by a call to VERRMSG, the text associated with the error number in the V/3000 error message file, or the associated custom error message, is returned to the user program. If requested by a call to VPUTWINDOW, the message is written to the window area of memory. A subsequent call to VSHOWFORM can be used to display this message on the terminal screen and enhance the fields with errors. Finally, VFIELDEDITS sets NUMERRS to the total number of fields in which errors were found.

EXAMPLES

The following calls edit data entered on a form:

COBOL:	CALL "VFIELDEDITS" USING COMAREA.
BASIC:	150 CALL VFIELDEDITS(C1(*))
FORTRAN:	CALL VFIELDEDITS(COMARE)
SPL:	VFIELDEDITS(COMAREA);

VFINISHFORM

Performs any processing specifications defined for the final phase of field editing.

```
CALL "VFINISHFORM" USING COMAREA
```

PARAMETERS

comarea Must be the name specified in a prior call to VOPENFORMF to open the forms file. The following fields in COMAREA must be set before calling VFINISHFORM (if not already set):

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VFINISHFORM may set the following fields:

CSTATUS	Set to non-zero value if call unsuccessful.
NUMERRS	Set to total number of fields in which errors in the form were detected.
NFNAME	Set to name of next form if processing specifications altered form name.
REPEATAPP	Set to new repeat code if processing specifications altered code.
FREEZAPP	Set to new next form code if processing specifications altered code.

All special processing defined as part of the "finish" phase of field editing is performed by this procedure. Altering the next form to be displayed is a typical finish operation, and updating a save field is another. (Refer to the discussion of phases in section IV.)

Like VFIELDEDITS, VFINISHFORM sets an error flag for each field that has an error as a result of processing the form. (Refer to the VFIELDEDITS discussion.)

EXAMPLES

The following examples perform all finish operations on a form:

COBOL:	`CALL "VFINISHFORM" USING COMAREA.`
BASIC:	`160 CALL VFINISHFORM(C1(*))`
FORTRAN:	`CALL VFINISHFORM(COMARE)`
SPL:	`VFINISHFORM(COMAREA);`

VGETBUFFER

Reads entire contents of data buffer from memory to user program.

```
CALL "VGETBUFFER" USING COMAREA, BUFFER, BUFLEN
```

PARAMETERS

comarea Before calling VGETBUFFER, the following COMAREA fields must be set (if not already set):

CSTATUS	Set to zero.
LANGUAGE	Set to code identifying the language of the calling program.
COMAREALEN	Set to total number of words in COMAREA.

VGETBUFFER may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.

buffer Character string in user program to which the data in the data buffer is written.

buflen Integer variable that specifies the number of characters to be transferred to the user buffer.

This procedure transfers data from the data buffer in memory to the area in the user program specified by the buffer parameter. The data transferred includes all the data entered in unprotected fields and display only fields on a form. The data in the data buffer is stored in the screen order of the fields on the form. Fields are stored adjacent to one another with no separators.

The number of characters moved from the data buffer is based on the number of characters specified in the buflen parameter, or the number of characters in DBUFLEN, whichever is less. (DBUFLEN contains the actual number of characters in the data buffer; it is usually set by a prior call to VREADFIELDS. DBUFLEN is an integer in COMAREA, see table 6-3.)

For example, if there are 20 characters in the data buffer (DBUFLEN=20), and the user requests 50 characters in the buflen parameter, only 20 characters are transferred. Conversely, if the user requests 10 characters through the buflen parameter, but there are 20 characters in the data buffer (DBUFLEN=20), only 10 characters are transferred.

Specific fields can be moved from the buffer with VGETFIELD or VGETtype. Note that VGETBUFFER performs no conversion. If you want the data converted as it is moved to the user program, you must move it one field at a time and use VGETtype, where type specifies the data type to which the field is converted.

EXAMPLES

The following examples transfer the contents of the data buffer in memory to the user program. The current value of DBUFLEN is specified as the user buffer length.

COBOL:

```
01  ORDER-ENTRY.
       03  PART-NO    PIC X(7).
       03  DESCR      PIC X(12).
       03  QTY        PIC S9(4).
       03  UNIT-PR    PIC S9(4)V9(2).
       03  TOTL-PR    PIC S9(6)V9(2).
       •
       •
       •
CALL "VGETBUFFER" USING COMAREA, ORDER-ENTRY, DBUFLEN.
```

BASIC:

```
340 L1=D1 <-------------- D1 is the "DBUFLEN" word in C1
350 CALL VGETBUFFER(C1(*),D$,L1)
```

FORTRAN:

```
CALL VGETBUFFER(COMARE,D1,DBFLEN)
```

SPL:

```
BYTE ARRAY D1(0:36);
  •
  •
  •
VGETBUFFER(COMAREA,D1,DBUFLEN);
```

VGETFIELD

Reads specified field from data buffer in memory to user program.

```
CALL "VGETFIELD" USING COMAREA, FIELDNUM, FIELDBUF, BUFLEN,
                  ACTUALEN, NEXTFLDNUM.
```

PARAMETERS

comarea	If not already set, the following COMAREA fields must be set before calling VGETFIELD:

CSTATUS	Set to zero.
LANGUAGE	Set to code identifying language of the calling program.
COMAREALEN	Set to total number of words in COMAREA.

VGETFIELD may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful, or if requested field has an error.

fieldnum	Integer variable containing the number assigned to the field by FORMSPEC.
fieldbuf	Character string in user program to which data entered in specified field is written.
buflen	Integer variable that specifies the number of characters in fieldbuf.
actualen	Integer to which VGETFIELD returns the number of characters actually moved to fieldbuf.
nextfldnum	Integer to which VGETFIELD returns the number of the next field in screen order. If there are no more fields, zero is returned. If the field number was set to zero or a negative number by a user program, this is an error. In this case, VGETFIELD returns the number of the first field in screen order.

VGETFIELD transfers the contents of a particular field from the data buffer to a variable in the user program. All the fields defined for a particular form in FORMSPEC are assigned numbers. The number assigned to a field by FORMSPEC never changes as long as the field exists. This is true regardless of changes to the field position in the buffer or to the field length. This is in contrast to VGETBUFFER which retrieves data according to the current field layout.

If the number of characters specified by buflen is less than the field size, the rightmost characters are truncated. If the requested field has an error, its value is returned, but CSTATUS is set to an error number indicating the field error flag is set.

Following a successful transfer, actualen contains the exact number of characters transferred to the user buffer, fieldbuf; nextfldnum is set to the number of the next field in screen order, or to zero after the last field is processed.

Note that VGETFIELD does not convert the data it moves. If you want to convert the field, you must use VGETtype, where type specifies the data type to which the field is converted.

EXAMPLES

Assume that field number "2" is to be read, and that the length of this field is 12 characters. The following calls read this field into a variable in the user program:

COBOL:

```
DATA DIVISION.
77  ORD-LEN           PIC  S9(4)COMP.
77  ITEM-LEN          PIC  S9(4) COMP.
77  FIELD-NUM         PIC  S9(4) COMP.
77  ACTUAL-LENGTH     PIC  S9(4) COMP.
77  NEXT-FIELD        PIC  S9(4) COMP.
01  ORDER-ENTRY.
    03  PART-NO       PIC  X(8).
    03  UNIT-PR       PIC  9(4)V9(2).
    03  QUANTITY      PIC  S9(4) COMP.
    03  TOTAL-PR      PIC  9(5)V9(2).
    03  PART-DESCR    PIC  X(12).<------field number "2"
•
•
PROCEDURE DIVISION.
•
•
    MOVE 12 TO ITEM-LEN.
    MOVE 2 TO FIELD-NUM.
    CAL "VGETFIELD" USING COM1,
                          FIELD-NUM,
                          PART-DESCR OF ORDER-ENTRY,
                          ITEM-LEN,
                          ACTUAL-LENGTH,
                          NEXT-FIELD.
```

BASIC:

```
350 F1=2
355 L1=12
360 CALL VGETFIELD(C1(*),F1,P$,L1,A1,N1)
```

FORTRAN:

```
FIELD=2
LEN=12
CALL VGETFIELD(COMARE,FIELD,PARTDES,LEN,LENFLD,NXTFLD)
```

SPL:

```
INTEGER
    FIELD,
    LEN;
BYTE ARRAY PARTDES(0:11);
•
•
FIELD:=2;
LEN:=12;
VGETFIELD(C1,FIELD,PARTDES,LEN,ACTUAL'LEN,NEXT'FLD);
```

VGETinfo

Three procedures which give access to an internal table containing information on the forms file, on forms, or on fields.

The procedures are:

VGETFILEINFO	Provides forms file information.
VGETFORMINFO	Provides form information.
VGETFIELDINFO	Provides field information.

```
CALL "VGETFIELDINFO" USING COMAREA, INFOBUF, INFOBUFLEN
```

PARAMETERS

comarea If not set already, the following COMAREA fields must be set before calling VGETinfo procedures.

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VGETinfo procedures will set the following field:

CSTATUS	Set to non-zero value if call unsuccessful, and zero if successful.

infobuf An integer array. The buffer through which you pass the input parameters (the request for specific information) and to which the procedure passes the output. This buffer has two sections. The first section, called the table control block, indicates how much of the internal information table you want copied into the buffer, and the second or main section details specifically which information you want. The information returned to the procedure is copied into the second or main section of the buffer.

infobuflen Integer variable set by user program to length of INFOBUF in words.

TABLE CONTROL BLOCK

The table control block is the first section of the INFOBUF parameter. It has two required parameters for VGETFILEINFO and VGETFORMINFO, and three for VGETFIELDINFO:

NUMROWS	Integer containing the number of table entries, that is, the number of forms or fields you are inquiring about. (You may inquire about only one forms file — the one that is open.)
ROWLEN	Integer containing the number of words of information on file, form, or field attributes you want for each entry.
FORMNAME	Necessary only for VGETFIELDINFO. Character array in which the name of the form containing the field about which you are inquiring is stored.

MAIN SECTION OF INFOBUF

The second or main section of the buffer consists mostly of space reserved for that portion of the internal information table which the procedure will copy into the INFOBUF array. It may also contain one key word such as form name or form number for each form for which you are requesting information, or one key word such as field name, field screen order number, or field number for each field about which you are inquiring. Keys allowed in this area are given in the table below.

PROCEDURE	PERMISSIBLE KEYS IN MAIN SECTION OF BUFFER
VGETFILEINFO	None
VGETFORMINFO	Form Name or Form Order Number
VGETFIELDINFO	Field Name or Field Screen Order Number or Field Number

The main section of the buffer INFOBUF is like a template; it should consist of one or more "rows" of length equal to ROWLEN, rows which are blank except for permissible keys in their proper word position. For example, if in this area you have entered only the name "BILLFORM" beginning at the third word of INFOBUF, and you call VGETFORMINFO, the procedure will be guided to copy the information about the specific form, "BILLFORM", into INFOBUF.

The tables below are a guide to the structure and contents of the internal information table from which specified rows are copied into INFOBUF. Permissible keys are marked with an asterisk.

Table 6-5a. Internal Table — File Information

DATA TYPE	WORD	CONTENTS	COMMENT
Integer	1	NUMROWS	Table Control Block (required)
	2	ROWLEN	
Integer Array	3-4	File version number	
Integer	5	Number of forms in file	
	6	Maximum number of fields	In any one form
	7	Maximum data buffer size in words	Of any one form
	8	Number of save fields in file	
Character Array	9-16	Name of head form in file	Last byte of word 16 unused
	17-18	Global error enhancement	
	19-20	Global window enhancement	
Integer	21	Position of window	Given in number of lines from top of screen

Calling one of the VGETinfo procedures is like requesting that a portion of an internal table containing file, form, or field information be copied into INFOBUF. The NUMROWS parameter can be visualized as the number of rows of the information table you wish to inspect, and the ROWLEN parameter can be seen as the number of specific columns of file, form, or field attributes you want copied into INFOBUF and passed back to your program.

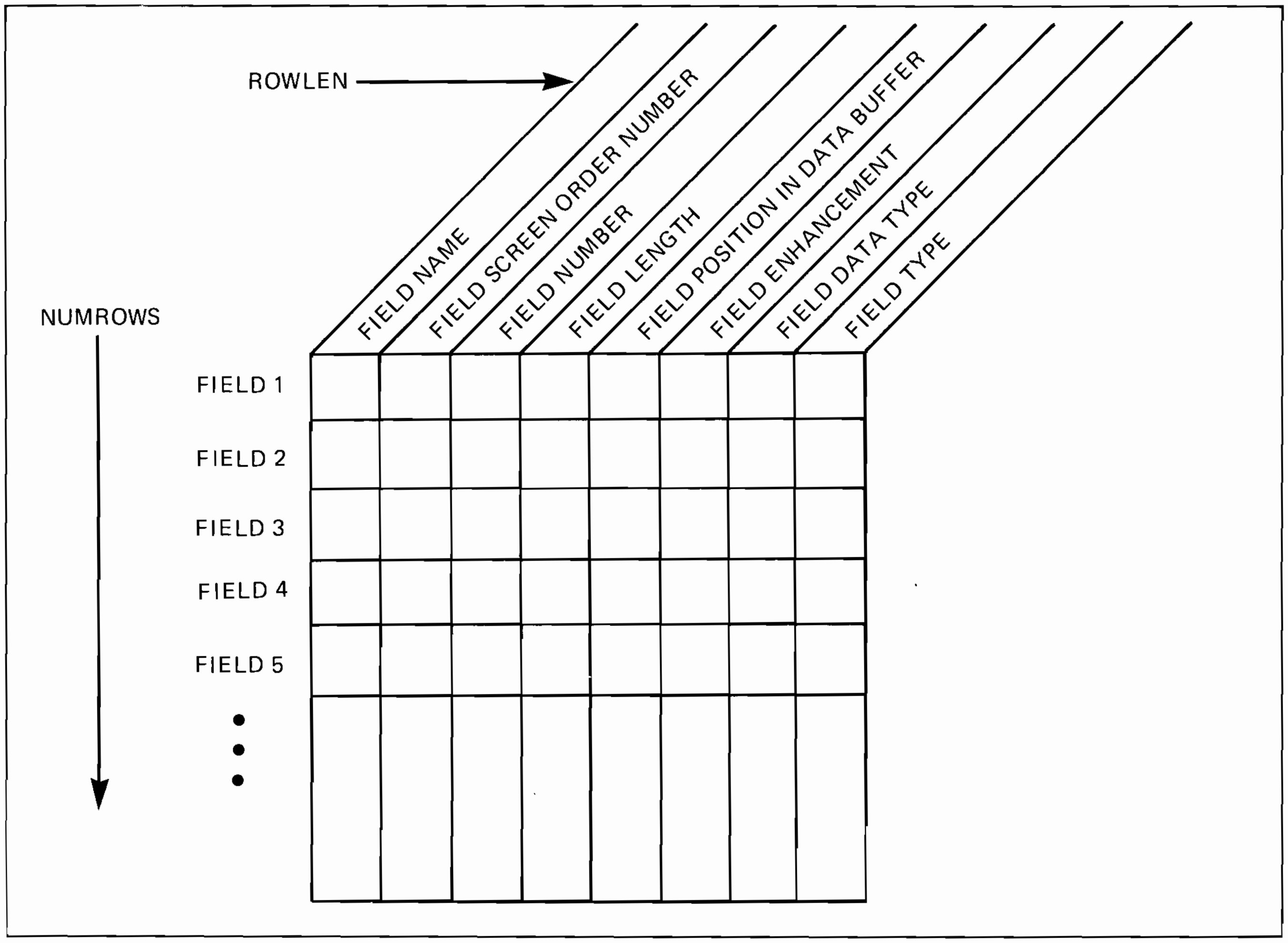


Figure 6-3. Visualization of VGETFIELDINFO Internal Information Table
(Fields are sequenced by screen order number.)

Table 6-5b. Internal Table — Form Information

DATA TYPE	WORD	CONTENTS	COMMENT
Integer	1 2	NUMROWS ROWLEN	Table Control Block
Character Array	3-10	Form name*	Last byte of word 10 unused (15 characters maximum)
Integer	11 12 13	Form number* Number of fields in form Data buffer length in words	
Character Array	14-21	Name of next form	Last byte of word 21 unused
Character	22 22	Repeat Option Next Form Option	First byte of word 22 (will be N, A, or R) Last byte of word 22 (will be C, A, or F)

*Permissible Key

Table 6-5c. Internal Table — Field Information

DATA TYPE	WORD	CONTENTS	COMMENT
Integer	1 2	NUMROWS ROWLEN	Table Control Block
Character Array	3-10 11-18	Name of form Name of field*	Last byte of word 10 unused Last byte of word 18 unused
Integer	19 20 21 22	Field screen order number* Field number* Length of field Position of field in data buffer	 In order of creation In characters (bytes) In bytes, offset from zero
Character Array	23-24 25-26	Field enhancement Data type of field	 May be CHAR, DIG, IMP, IMPn, MDY (DMY, YMD), NUM, or NUMn
Character	27	Field type	First byte of word 27 May be "O", "R", "P", or "O"

*Permissible Key

To avoid the passing of incorrect or unwanted information to your program, always reinitialize INFOBUF to all spaces before filling it and calling a VGETinfo procedure.

PASSING INFOBUF WITHOUT ENTERING KEYS

All keys in the main section of INFOBUF are optional; you may fill in only the elements of the table control block if you wish. Whenever there are no keys to guide the procedure in selecting which row of the information table to copy, the procedure will select the first row not already copied during the current procedure call.

In terms of the table visualization in Figure 6-3, if you specify in NUMROWS that you want to see 10 entries of the information table and you enter the keys for fields 3, 5, and 6 (in screen order), VGETFIELDINFO will copy into INFOBUF rows 3, 5, 6, and 7 through 13 of the information table, to make a total of 10 rows. If you enter no keys into the main section of the buffer, VGETFIELDINFO will copy the contents of rows 1 through 9 of the information table into INFOBUF. In either case, the length of each row is determined by the value you specified for ROWLEN in the table control block.

TABLE WRAPAROUND

In the internal information table referenced by the VGETinfo procedures, a request for the row following the final row of information will access the first row of the table. For example, if you ask in a call to VGETFILEINFO for information on 2 files (NUMROWS=2), when only one forms file is open, the procedure will place two copies of the information about that one forms file into INFOBUF.

For another example which takes advantage both of table wraparound and of the fact that keys are optional, if your form has 20 fields and you want information on fields 1, 2, 18, 19, and 20 (in screen order), set NUMROWS equal to 5 and ROWLEN equal to 17 in the input control block and enter as a key only the name of field 18 at the eleventh word of INFOBUF. The call to VGETFIELDINFO will cause 17 words of information on each of the 5 fields to be copied into the buffer, in this order: field information on field 18 first, followed by information on fields 19, 20, 1, and 2.

EXAMPLES

The following COBOL example illustrates the data declaration of INFOBUF and INFOBUFLEN, and the entering of the parameters in the table control block and in the main section of INFOBUF. Note that before VGETFIELDINFO is called, it is initialized to all spaces. Never initialize this buffer to zeros. The called procedure, VGETFIELDINFO, will copy into INFOBUF 20 words of information on fields 2 through 6 of the form "FORM1".

```
DATA DIVISION.
        •
        •
        •
WORKING-STORAGE SECTION.
01   INFOBUF.
     05   OFLD-NUMROWS                  PIC S9(4) COMP.
     05   OFLD-ROWLEN                   PIC S9(4) COMP.
     05   OFLD-FORMNAME                 PIC X(15).
     05   FILLER                        PIC X.
     05   OFLD-E-TABLE OCCURS 10 TIMES.
          10   OFLD-E-FIELDNAME         PIC X(15).
          10   FILLER                   PIC X.
          10   OFLD-E-SNO               PIC S9(4) COMP.
          10   OFLD-E-FLDNUM            PIC S9(4) COMP.
          10   OFLD-E-LEN               PIC S9(4) COMP.
          10   OFLD-E-DBUFPOS           PIC S9(4) COMP.
          10   OFLD-E-ENH               PIC X(4).
          10   OFLD-E-DATATYPE          PIC X(4).
          10   OFLD-E-TYPE              PIC X.
          10   FILLER                   PIC X(7).
01   INFOBUFLEN                         PIC S9(4) COMP VALUE 210.
        •
        •
        •
PROCEDURE DIVISION.
        •
        •
        •
     MOVE SPACES TO INFOBUF.
        •
        •
        •
     MOVE 5 TO OFLD-NUMROWS.
     MOVE 20 TO OFLD-ROWLEN.
     MOVE "FORM1      " TO OFLD-FORMNAME.
     MOVE 2 TO OFLD-E-SNO(1).
     CALL "VGETFIELDINFO" USING COMAREA,INFOBUF,INFOBUFLEN.
```

The following examples call the VGETFORMINFO procedure.

BASIC:	340 CALL VGETFORMINFO(C(*),I(*),I1) 350 REM I(*) is INFOBUF and I1 is INFOBUFLEN
FORTRAN:	CALL VGETFORMINFO(COMAREA,INFOBUF,INFOBUFLEN)
SPL:	VGETFORMINFO(COMAREA,INFOBUF,INFOBUFLEN);

VGETKEYLABELS

Retrieves the global function key labels for the function keys.

```
CALL "VGETKEYLABELS" USING COMAREA,FORM-OR-GLOB,NUM-OF-LABELS,LABELS
```

PARAMETERS

comarea If not already set, the following COMAREA fields must be set before calling VGETKEYLABELS.

CSTATUS	set to zero.
COMAREALEN	set to total number of words in COMAREA.

form-or-glob Integer value specifying which type of labels are to be retrieved.

0 - Retrieve global labels
1 - Retrieve current form labels

num-of-labels Integer value indicating how many labels are to be retrieved. This value should be from 1 to 8, inclusive.

labels A byte array in which the labels will be passed back to the user program. The length of the array must be at least NUM'OF'LABELS * 16. (Each 16 byte string will be displayed as a label of 2 lines of 8 bytes each.)

This procedure is used to retrieve global or current form labels for function keys. All function key labels or any number of labels can be retrieved with this procedure by setting the number of labels parameter.

EXAMPLES

The following examples illustrate a call to VGETKEYLABELS in each of the programming languages.

COBOL:

```
77  FORM-OR-GLOB       PIC  S9(4)COMP.
77  NUM-OF-LABELS      PIC  S9(4)COMP.
77  KEY-LABELS         PIC   X(32).
.
.
MOVE 1 TO FORM-OR-GLOB.
MOVE 2 TO NUM-OF-LABELS.
CALL "VGETKEYLABELS" USING COMAREA,FORM-OR-GLOB,
        NUM-OF-LABELS,KEY-LABELS.
```

BASIC:

```
10  INTEGER F,N
20  DIM L$[32]
30  F=1
40  N=2
50  CALL VGETKEYLABELS(C[*],F,N,L$)
```

FORTRAN:

```
CHARACTER*32 LABELS
INTEGER FORMORGLOB,NUMLABELS
FORMORGLOB=1
NUMLABELS=2
CALL VGETKEYLABELS(COMAREA,FORMORGLOB,NUMLABELS,
        LABELS)
```

SPL:

```
INTEGER
        FORM'OR'GLOB,
        NUM'OF'LABELS;
BYTE ARRAY
        LABELS(0:31);
.
.
FORM'OR'GLOB:=1;
NUM'OF'LABELS:=2;
VGETKEYLABELS(COMAREA,FORM'OR'GLOB,NUM'OF'LABELS,
        LABELS);
```

VGETNEXTFORM

Reads the next form from an open forms file into the form definition area of memory.

```
CALL "VGETNEXTFORM" USING COMAREA
```

PARAMETERS

comarea Must be communication area assigned to open forms file from which form is to be retrieved. The following items in COMAREA must be set before calling VGETNEXTFORM:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.
NFNAME	Set to name of next form, or $END, $HEAD, $RETURN, or $REFRESH. (Also set by prior call to VGETNEXTFORM, VOPENFORMF, VREADBATCH, VINITFORM, VFIELDEDITS, or VFINISHFORM.)
REPEATAPP	Set to 1 if current form is to be repeated, to 2 if repeated and appended. Note that REPEATAPP must be set to zero in order to retrieve and display the next form. (Also may be set by a prior call to VGETNEXTFORM, VINITFORM, VFIELDEDITS, or VFINISHFORM.)
FREEZAPP	Set to 1 if next form is to be appended, to 2 if this form is to be frozen before next form is appended. Set to zero in order to clear the current form before displaying the next form. (Also may be set by prior call to VGETNEXTFORM, VINITFORM, VFIELDEDITS, or VFINISHFORM.)

VGETNEXTFORM sets the following items in COMAREA:

NUMERRS	Set to zero.
CFNAME	Set to name of form just read from file.
CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.
MULTIUSAGE	Set to 1 if next form is son or brother of previous form, zero otherwise.

In addition, if a new form has been retrieved (REPEATAPP = 0), VGETNEXTFORM sets the following items:

CFNUMLINES	Set to number of lines in form just read (now the current form).
NFNAME	Set to name of next form to be read from file.
REPEATAPP	Set to value read from forms file for this (current) form.
FREEZAPP	Set to value read from forms file for this (current) form.
DBUFLEN	Set to length (in characters) of the current form just read from the forms file.

VGETNEXTFORM checks the value of REPEATAPP passed in COMAREA. If this value indicates the current form is to be repeated, or repeated and appended to itself, it does not read the next form, nor does it update the values of NFNAME, REPEATAPP, FREEZAPP, CFNUMLINES, or DBUFLEN. Note that a repeating form is repeated until REPEATAPP is cleared to zero, either by the user program or, for ENTRY, when the operator presses the NEXT FORM key (f6) to request the next form, or by the FORMSPEC processing language.

If the current form is not to be repeated, VGETNEXTFORM checks NFNAME to determine which form to read from the forms file. NFNAME may contain one of the following values:

form name	Identifies the form to be read from the forms file.
$REFRESH	Clear and reset the terminal screen, and redisplay the form, window, and data buffer.
$RETURN	Retrieve previous different form; if current form is the head form, the current form is retrieved.
$HEAD	Retrieve first form displayed when the forms file is executed (the "head" form).
$END	Terminate execution of VGETNEXTFORM; return to calling program without resetting any COMAREA items.

EXAMPLES

The following examples call VGETNEXTFORM to retrieve the next form from the forms file and reset the COMAREA according to the values in the next form.

COBOL:	`CALL "VGETNEXTFORM" USING COMAREA.`
BASIC:	`11 CALL VGETNEXTFORM(C1(*))`
FORTRAN:	`CALL VGETNEXTFORM(COMARE)`
SPL:	`VGETNEXTFORM(COMAREA);`

VGETtype

Reads ASCII-coded data field from data buffer into user program, converting numeric value to specified type.

When using HP V/3000 for data entry, all data is read from the unprotected fields on the screen as ASCII characters and concatenated to form a data buffer. The VGETBUFFER intrinsic will transfer the entire buffer to an application program or VGETFIELD can be used to obtain the contents of an individual field. However, neither of these intrinsics performs any conversion of the data values. Therefore, a field defined as DIG in FORMSPEC and containing the value 123 would be transferred as the string of characters "123". The VGETtype intrinsics, VGETINT, VGETDINT, VGETREAL, and VGETLONG, have been provided to perform conversion from ASCII to the four indicated data types.

This procedure may be specified as:

VGETINT	converts value to integer
VGETDINT	converts value to double integer
VGETREAL	converts value to real value
VGETLONG	converts value to long value

```
CALL "VGETINT" USING COMAREA, FIELDNUM, VARIABLE.
```

PARAMETERS

comarea If not set already, the following COMAREA fields must be set before calling VGETtype:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VGETtype may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful, or if requested field has an error.

fieldnum Integer variable containing the number assigned to the field by FORMSPEC.

variable Variable of type specified in VGETtype into which converted value is placed.

This procedure reads the field identified by its field number from the data buffer. (Note that this field number is a unique number assigned to each field by FORMSPEC and is totally independent of the field position in the data buffer.) The field's value must be numeric, but its data type need not be. That is, numbers in a character type field can be converted.

The numeric value, stored in the buffer in ASCII coded form, is converted to the specified type and then moved to the variable in the user's program. (Refer to table 6-6 for the format of each type.) If errors occur during conversion, CSTATUS is set to an error code. If the requested field has an error, its value is moved to the variable, but CSTATUS is set to an error code.

Table 6-6. Numeric Type Conversion

Type	Format
INT	Single-word fixed-point; 2's complement representation of positive and negative values; range from –32768 through +32767.
DINT	Double-word fixed-point format; 2's complement representation of positive and negative range between approximately –2 billion and +2 billion. (Not used by BASIC programs.)
REAL	Double-word floating-point format with sign bit in bit 0; an exponent (biased by +256) in bits 1 through 9, and a positive fraction in the remaining 22 bits in HP 3000 format. (Not used by COBOL programs.)
LONG	Floating-point format using four words; sign bit in bit 0; an exponent (biased by +256) in bits 1 through 9, and a positive fraction in the remaining 54 bits in HP 3000 format. (Not used by COBOL programs.)

The following chart will help you correlate V/3000 data types with VGETtype intrinsics and programming language data types.

Language	Intrinsic	Data Type
FORTRAN	VGETINT VGETDINT VGETREAL VGETLONG	INTEGER INTEGER*4 REAL DOUBLE PRECISION
BASIC	VGETINT VGETREAL VGETLONG	INTEGER REAL LONG
SPL	VGETINT VGETDINT VGETREAL VGETLONG	INTEGER DOUBLE INTEGER REAL LONG
COBOL	VGETINT VGETDINT	S9-S9 (4) COMP S9 (5) - S9 (9) COMP

1. If errors occur during conversion, the status word in the communications area is set to an error value.

2. If the requested field has been flagged as having an error (perhaps by VFIELDEDITS or VSETERROR), the conversion is performed, but the status word is also set to an error value.

3. An attempt to convert a number larger than 32767 using VGETINT will return an error value (504) in the status word and will leave the receiving value unchanged.

4. All commas are stripped before conversion is performed.

5. Fields of type CHAR may be converted as long as they contain numeric characters (including ". , - + ") Otherwise an error value will be returned in the status word.

6. VGETINT and VGETDINT will only convert the integer portion of a given field. The fractional portion is truncated before conversion. Remember that in a field of type IMPn, the right-most "n" characters will be treated as a fraction.

7. If VGETFIELD is used to pass a field containing a decimal point to a COBOL program, the decimal point will also be passed and no arithmetic may be performed on the field.

8. Negative numbers can be zoned correctly for COBOL only by using the VGETINT and VGETDINT intrinsics. VGETBUFFER and VGETFIELD will transfer the negative sign, but COBOL will treat the value as positive, ignoring the sign character. An EXAMINE statement using TALLY can determine that the negative sign is present and then the program can treat the value accordingly.

9. Normal rules of truncation in COBOL are followed. For example, conversion of 12345 using VGETINT with a receiving field of S9(4) will truncate the value of 2345.

10. VGETINT may be used to convert positive integers to type LOGICAL in SPL.

EXAMPLES

The following calls convert a value read from field "5" in the data buffer, converts it to integer representation, and stores it in the user programs:

COBOL:

```
77  FIELD-NUM          PIC  S9(4)COMP.
77  COUNT              PIC  S9(4)COMP.
•
•
•
MOVE 5 TO FIELD-NUM.
CALL "VGETINT" USING COMAREA, FIELD-NUM, COUNT.
```

BASIC:

```
210 F1=5
220 CALL VGETINT(C1(*),F1,C)
```

FORTRAN:

```
FIELD=5
CALL VGETINT(COMARE,FIELD,K1)
```

SPL:

```
INTEGER
    FIELD,
    COUNT;
•
•
•
FIELD:=5;
VGETINT(COMAREA,FIELD,COUNT);
```

VINITFORM

Initializes fields in data buffer according to specifications defined in the initialize phase of field definition.

```
CALL "VINITFORM" USING COMAREA
```

PARAMETERS

comarea	Must be a COMAREA assigned to an open forms file. The following COMAREA items must be set before calling VINITFORM (if not already set):

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VINITFORM may set the following COMAREA values:

CSTATUS	Set to non-zero value if call unsuccessful.
NUMERRS	Set to total number of fields in which errors were detected.
NFNAME	Set to new next form name, if name changed by processing specifications.
REPEATAPP	Set to new repeat code, if code changed by processing specifications.
FREEZAPP	Set to new next form code, if code changed by processing specifications.

NOTE

Both VOPENFORMF and VGETNEXTFORM must have been executed prior to calling VINITFORM.

Certain values may be assigned to fields as initial values. These values are determined by special processing specifications that are explicitly or implicitly defined as part of the initialize phase of field processing using FORMSPEC. These values include any initial values specified on the FIELD menus for the form. If no initial values were specified, all fields are initialized to blanks by VINITFORM. If the form being initialized is a son or brother to the previous form, data from the previous form is transferred to this form (with conversion if necessary) before initializations occur.

EXAMPLES

The following calls set initial values in the data buffer area of memory, according to initialize specifications defined for each field in the current form:

COBOL:	CALL "VINITFORM" USING COMAREA.
BASIC:	140 CALL VINITFORM(C1(*))
FORTRAN:	CALL VINITFORM(COMARE)
SPL:	VINITFORM(COMAREA);

VLOADFORMS

Allows forms named in FORMS to be loaded into terminal workspaces.

```
CALL "VLOADFORMS" USING COMAREA,NUM-OF-FORMS,FORMS-LOADED,FORMS
```

PARAMETERS

comarea

If not already set, the following COMAREA fields must be set prior to a call to VLOADFORMS:

CSTATUS	Set to zero
COMAREALEN	Set to toal number of words in COMAREA

num-of-forms

Integer value indicating the number of forms to be loaded.

-1 - Workspace configuration under user control.
0 - No local form storage.
1. .4 - Number of forms to be loaded.

forms-loaded

Integer value indicating the number of forms that were successfully loaded.

forms

The names of the forms to be loaded. Each name is 15 characters. The dummy byte should be placed at the 16th character slot of the array.

This intrinsic is to be used with user application programs being written for use on terminals with local form storage as a feature.

VLOADFORMS will load the forms named in FORMS into the terminal workspaces. It may not be possible to load all the forms named because of limit of available space. The number loaded will be returned in the parameter FORMS-LOADED and their names will be the first FORMS-LOADED in the FORMS array.

Note this procedure loads forms immediately while LOOK'AHEAD only happens in VREADFIELDS.

EXAMPLES

The following examples illustrate a call to VLOADFORMS in each of the programming languages:

COBOL:

```
77  NUM-OF-FORMS        PIC  S9(4)COMP.
77  FORMS-LOADED        PIC  S9(4)COMP.
77  FORMS               PIC    X(16).
.
.
MOVE 1 TO NUM-OF-FORMS.
MOVE "FORMA                   " TO FORMS.
CALL "VLOADFORMS" USING COMAREA, NUM-OF-FORMS,
        FORMS-LOADED,FORMS.
```

BASIC:

```
10  INTEGER N,F
15  DIM F$[16]
20  N=1
30  F$="FORMA                 "
40  CALL VLOADFORMS(C[*],N,F,F$)
```

FORTRAN:

```
INTEGER NFORM,FLOAD
CHARACTER*16 FORMS
NFORM=1
FORMS="FORMA           "
CALL VLOADFORMS(COMAREA,NFORM,FLOAD,FORMS)
```

SPL:

```
INTEGER
        NUM'OF'FORMS,
        FORMS'LOADED;
BYTE ARRAY
        FORMS(0:15);
.
.
NUM'OF'FORMS:=1;
MOVE FORMS:="FORMA           ";
VLOADFORMS(COMAREA,NUM'OF'FORMS,FORMS'LOADED,FORMS);
```

VOPENBATCH

Opens existing batch file for processing; or, if specified file is new, creates a batch file and then opens it for processing.

```
CALL "VOPENBATCH" USING COMAREA, BATCHFILE
```

PARAMETERS

comarea Must identify the COMAREA specified when the forms file was opened. If not set already, calling program must set the following COMAREA fields:

CSTATUS	Set to zero.
LANGUAGE	Set to code that identifies language of the calling program.
COMAREALEN	Set to total number of words in COMAREA.

VOPENBATCH sets the following COMAREA fields:

NFNAME	Set to the name of the form corresponding to the record identified by RECNUM.
RECNUM	Set to zero if new file opened; to the next sequential record number if existing file opened.
NUMRECS	Set to zero if new file opened; to the number of non-deleted records in file if existing file opened.

VOPENBATCH may set the following fields:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

batchfile Character string of up to 36 characters (including a terminator) that identifies the batch file being opened. Specified name can be any fully qualified MPE file name.

VOPENBATCH opens the specified batch file for processing by the calling program. The batch file may be an existing file or a new file.

EXISTING FILE

If the named file already exists, VOPENBATCH initializes RECNUM to the record number of the next record in the file, and NUMRECS to the total number of existing batch records. Thus, a user resuming collection does not overwrite data collected into previous batch records. VOPENBATCH sets NFNAME to the name of the form associated with the batch record to be collected. This record is identified by RECNUM, VOPENBATCH keeps track of forms sequence in order to associate a form with a record. For example, if a batch file is closed after record 6 of FORMA was collected, and FORMA is a repeating form, the batch file starts with record 7 FORMA when it is next opened by VOPENBATCH.

VOPENBATCH also resets the global environment (save fields and so forth) to the environment existing when collection stopped. (All this information is derived from the file labels preceding each file.)

NEW FILE

If the named file does not exist, VOPENBATCH creates a new file with the specified name, and sets RECNUM and NUMRECS to zero. A new batch file created by VOPENBATCH has the following characteristics:

- Non-Ksam file
- ASCII-coded data
- Fixed-length records
- No carriage control
- :FILE command allowed (use actual file designator)
- Update access
- Exclusive (non-shared) access
- No dynamic locking
- No multi-record access
- Normal buffering

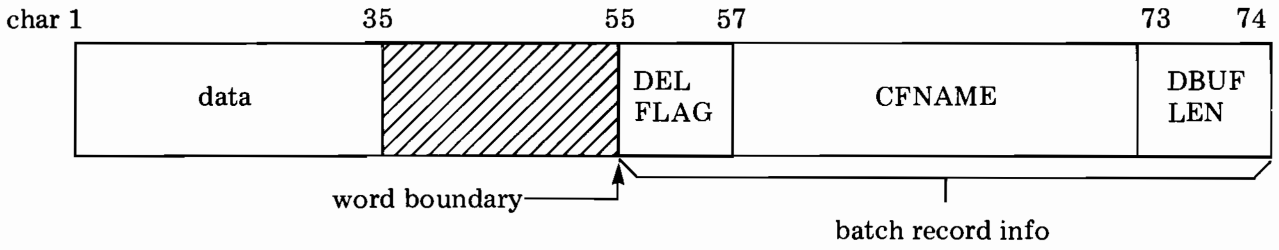
RECORD FORMAT

The size of the fixed-length batch file records must be large enough to hold the largest data buffer used by the forms file associated with the batch file, plus 10 words for batch record control information. If the largest data buffer is an even number of bytes, an additional word is added before the control information. This batch record information consists of:

1 word (logical)	— Delete flag (TRUE if record deleted)
8 words (character)	— Current form name + extra character
1 word (logical)	— Data buffer length

Total = 10 words (20 characters)

The above batch record information is written at the end of each record in the batch file, starting on a word boundary. To illustrate, assume the record size is 74 characters, and the data only requires 35 characters (characters are numbered from 1):



LABELS

In addition, VOPENBATCH creates sufficient user labels (each 128 words long) to hold any save field buffers, plus 88 words for the collection environment and the forms file version.

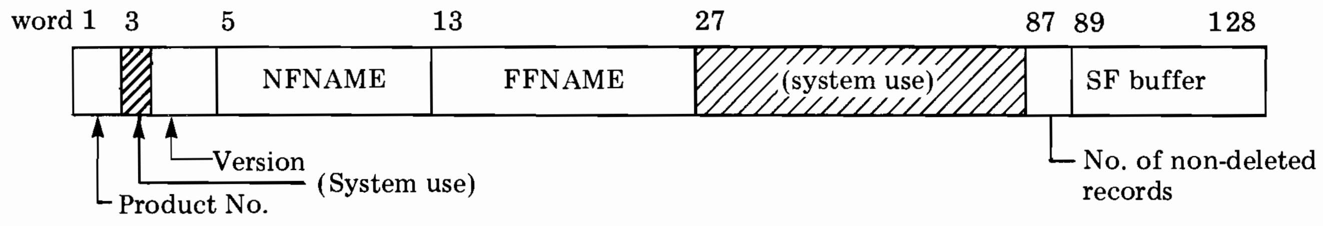
The length of the save field buffers depends on how many (if any) save fields were defined for the form and the length of each. The collection environment consists of the forms file name and version number, the next form name, and 61 words of system information. This information is stored as follows:

1 word —	Product number
2 words —	Forms file version (date and time of compilation)
8 words —	Next form name + 1 extra character
14 words —	Forms file name
60 words —	Reserved for system use
2 words —	Number of non-deleted records in batch file

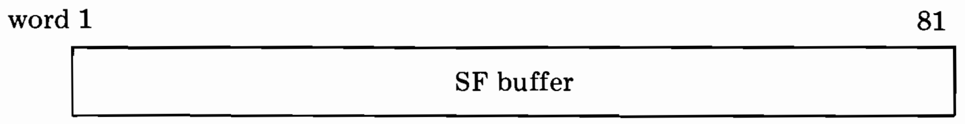
Total = 88 words (176 characters)

For example, assume that the save fields buffer requires 242 characters. The following user labels are created by VOPENBATCH:

LABEL 0 — Contains first 40 words — 80 characters — of Save Fields buffer



LABEL 1 — Contains remaining 81 words (162 characters) of Save Fields buffer

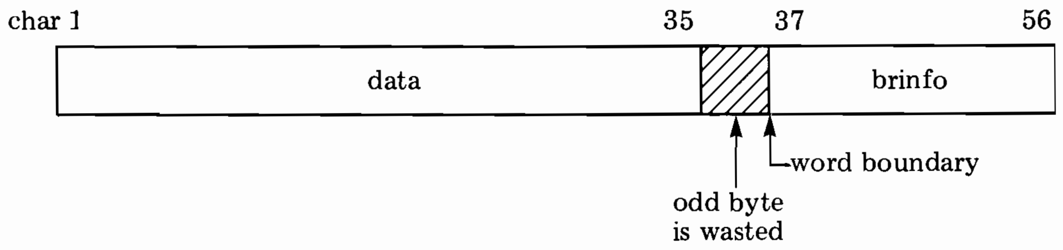


Note that the length of the save fields buffers is determined by taking the number of characters in each save field, summing them together, and then rounding the total up to an even number.

CREATING YOUR OWN BATCH FILE

Should you want to create your own batch file rather than calling VOPENBATCH, you must build the file using the above record format and user label requirements. Also, if you create a batch file with variable-length or undefined-length records rather than fixed-length records, browsing of the batch file is not allowed.

If you do, nevertheless, want to create a batch file with variable-length records for data collection only, the batch file information is stored immediately following the data in each record. Assume a variable-length record with 35 characters of data:



The total size of this record is 56 characters; depending on the size of the data, other records have varying lengths. The maximum size of the variable-length records must be the size of the largest data buffer (in words) plus 10 words for the batch record information.

Undefined-length records are formatted like fixed-length records.

EXAMPLES

The following examples open a batch file:

COBOL:	CALL "VOPENBATCH" USING COMAREA, BATCH.
BASIC:	170 CALL VOPENBATCH(C1(*),B1$)
FORTRAN:	CALL VOPENBATCH(COMARE,BATCH)
SPL:	VOPENBATCH(COMAREA,BATCH):

If the requested batch file name is an existing file, it is opened and the operator can continue to enter data into the record following the last record that contains data. If the requested batch file name is a new file, VOPENBATCH creates a batch file and data entered at the terminal is written to the first record in the file (record 0).

VOPENFORMF

Opens forms file for access.

```
CALL "VOPENFORMF" USING COMAREA, FORMFILE
```

PARAMETERS

comarea

The COMAREA name must be unique for each open forms file. If not already set, the calling program must set the following items in COMAREA:

CSTATUS	Set to zero.
LANGUAGE	Set to code that identifies language of the calling program.
COMAREALEN	Set to total number of words in COMAREA.
USRBUFLEN	BASIC programs only; set to number of words needed for COMAREA extension.
LABEL'OPTION	Set to one.
FORM'STORE'SIZE	Set to minus one, or zero, or one to four.

VOPENFORMF sets the following COMAREA items:

LASTKEY	Set to zero.
NUMERRS	Set to zero.
RECNUM	Set to zero.
DBUFLEN	Set to zero.
CMODE	Set to zero (collect mode).
REPEATAPP	Set to zero (no repeat/append).
FREEZAPP	Set to zero (clear current form).
PRINTFILNUM	Set to zero.
DELETEFLAG	Set to FALSE (all zeros).
CFNAME	Set to bank.
NFNAME	Set to name of head form.
CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.
WINDOWENH	Set to enhancement code defined in forms file.

If the HP 3075/6 terminals are used:

ERROR'LIGHT	Set to default or user specified value.
SPLIT'MSG'PAUSE	Set to default or user specified value.

formfile

Character string of up to 36 characters (including a terminator) that identifies the forms file being opened. Specified name can be any fully qualified MPE file name.

VOPENFORMF opens the specified forms file for processing by the calling program. If the program is written in a language other than BASIC, a DL area is obtained for use as the COMAREA extension. In this case, the program must not use the DL area for other functions. If the calling program is BASIC, it must provide its own COMAREA extension immediately following the COMAREA and specify the size of this extension in USRBUFLEN.

EXAMPLES

The following examples illustrate the call to VOPENFORMF in each of the programming languages.

COBOL:	CALL "VOPENFORMF" USING COMAREA, FNAME.
BASIC:	100 CALL VOPENFORMF(C1(*),F1$)
FORTRAN:	CALL VOPENFORMF(COMARE,FNAME)
SPL:	BYTE ARRAY FNAME(0:34);

```
BYTE ARRAY FNAME(0:34);
•
•
•
FNAME:="FORMSA     ";
VOPENFORMF(COM1,FNAME):
```

VOPENTERM

Opens a V/3000 supported terminal.

```
CALL "VOPENTERM" USING COMAREA, TERMFILE
```

PARAMETERS

comarea

The comarea name must be unique for each open forms file. The calling program should initialize the entire COMAREA to zero before calling VOPENTERM. In addition, the following COMAREA items must be set before the call:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.
FILEN	Set to MPE file number.
IDENTIFIER	Set to appropriate V/3000 terminal type.
LAB'INFO	Set to appropriate number and length of labels.

If the HP 3075/6 terminals are used:

LEFT'MODULE	Set to appropriate value defining terminal.
RIGHT'MODULE	Set to appropriate value defining terminal.
KEYBOARD	Set to appropriate value defining terminal.
DISPLAY	Set to appropriate value defining terminal.

termfile

Character string of up to 36 characters (including a terminator) that identifies the terminal. If set to a blank, $STDIN is used. Otherwise, any fully qualified MPE file name can be assigned to the terminal as its formal file designator. If specified, the name must be terminated by a special character (a blank is suggested). Before using a terminal identified by name, the formal designator may be equated to an actual designator with a :FILE command.

This procedure opens the terminal as a file. If you are running your program as a session with your terminal as the open terminal file, the terminal name should be left blank so that $STDIN is opened. The HP 3075/6 terminals are treated as character mode terminals.

If you are using a 2640B or 2644 terminal and the terminal is not in block mode, VOPENTERM asks you to press the BLOCK MODE key; other terminals are set to block mode automatically. In order to operate V/3000, the terminal must be mode, strapped for page.

EXAMPLES

The following examples open a terminal whose COMAREA is identified as COM1 (C1(60) for BASIC) and assigns the terminal to $STDIN.

COBOL:

```
77  T1 PIC X(8) VALUE SPACES.◄──────────── termfile item
01  COM1.◄──────────── comarea
    03  STATUS PIC S9(4) COMP VALUE ZERO.
    03  LANGUAGE PIC S9(4) COMP VALUE ZERO.
    03  COMAREALEN PIC S9(4) COMP VALUE 60.
    03  CFNAME PIC X(15).
    03  FILLER PIC X.
    •
    03  NFNAME PIC X (15).
    03  FILLER PIC X.
    03  NFNAME PIC X (15).
    03  FILLER PIC X.
    03  FILLER PIC 9 (4) COMP.

PROCEDURE DIVISION.
•
•
OPENTERMINAL.
    MOVE SPACES TO T1.
    CALL "VOPENTERM" USING COMAREA, T1.
```

BASIC:

```
90   T1$="  "
100  CALL VOPENTERM(C1(*),T1$)
```

FORTRAN:

```
T1="  "
CALL VOPENTERM(COMARE,T1)
```

SPL:

```
T1:="    ";
VOPENTERM(COMAREA,T1);
```

VPOSTBATCH

Protects a user specified portion of the batch file data from a system crash by posting an end of file mark after the last record referenced and updating the batch file labels.

```
CALL "VPOSTBATCH" USING COMAREA
```

PARAMETERS

comarea If not set already, the following COMAREA fields must be set before a call to VPOSTBATCH:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VPOSTBATCH will set the following field:

CSTATUS	Set to non-zero if call unsuccessful, zero if successful.

VPOSTBATCH posts an end of file mark after the last record referenced in the batch file and updates the environmental information found in the file label.

If a system crash or power failure occurs while the batch file is open, all data before the end of file mark will be preserved, and data collection will continue from that point. In the ENTRY program VPOSTBATCH is called after every 20 records, though you may extend or shorten this posting interval by increasing or decreasing the value of a globally declared integer. (This integer is named "PARMVAL" in the COBOL, FORTRAN, and SPL versions, and "P1" in the BASIC version of ENTRY.)

Two cautions:

a. Never call VPOSTBATCH while you are in BROWSE mode, or at any time when the last record referenced is not the last record in the batch file. If you call this procedure when the last record referenced is in the middle of the file, VPOSTBATCH will post a mark before the actual end of the file, causing all data after this mark to be lost.

b. The COMAREA field NUMRECS, which contains the number of undeleted records in the file, may not be restored correctly after a system crash if batch records had been deleted since the last call to the VPOSTBATCH procedure.

EXAMPLES

COBOL:	CALL "VPOSTBATCH" USING COMAREA.
BASIC:	290 CALL VPOSTBATCH(C1 (*))
FORTRAN:	CALL VPOSTBATCH(COMARE)
SPL:	VPOSTBATCH(COMAREA);

VPRINTFORM

Prints the current form on an offline list device.

```
CALL "VPRINTFORM" USING COMAREA, PRINTCNTL, PAGECNTL
```

PARAMETERS

comarea The following COMAREA items must be set before calling VPRINTFORM (unless they are already set):

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.
PRINTFILNUM	Set to file number of list file to which form is to be printed. If set to zero, VPRINTFORM opens the device "LP" as the list file and sets PRINT-FILNUM to the file number of the opened list file.

VPRINTFORM may set the following COMAREA values:

PRINTFILNUM	If VPRINTFORM opened the list file, set to the file number of the open file.
CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

printcntl Integer that, if set to 1, causes VPRINTFORM to underline each field in the form listing. If set to any other value, fields are not underlined.

pagecntl Integer value that determines the carriage control operation performed after a form is listed. May be any of the carriage control codes used by the MPE FWRITE procedure, including the following:

61(octal) — Page eject
320(octal) — No line feed or carriage return
zero — Carriage return/line feed

VPRINTFORM prints the current form to a list file. It is analagous to VSHOWFORM, in that it prints the form and the current data buffer values, except that VPRINTFORM prints the form on a hard-copy device rather than on the terminal. Enhancements obviously cannot be shown directly, and the window line is not printed. The form must have been read into the form definition area of memory by a prior call to VGETNEXTFORM.

The carriage control character specified in the pagecntl parameter is effective after the form is printed.

If the calling program opens the list file, it must supply the file number of this file in PRINTFILNUM. If PRINTFILNUM is zero, VPRINTFORM opens a list file and sets PRINTFILNUM to the file number of the file. VPRINTFORM opens the list file, with the formal and actual file designator FORMLIST, assigns it to the device class LP, and specifies its length as 80 characters. This is equivalent to using the file equation:

```
:FILE FORMLIST;DEV=LP;REC=-80
```

A user may change any of these characteristics with a :FILE command.

EXAMPLES

Each of the following calls prints the current form with all its specified characteristics on a list device; if not already open, it opens the device file.

COBOL:	CALL "VPRINTFORM" USING COMAREA, UNDERLINE, PAGE.
BASIC:	135 CALL VPRINTFORM(C1(*),U,P)
FORTRAN:	CALL VPRINTFORM(COMARE,UNDRLN,PAGE)
SPL:	VPRINTFORM(COMAREA,UNDERLINE,PAGE);

VPUTBUFFER

Writes data from a user program to the data buffer in memory.

```
CALL "VPUTBUFFER" USING COMAREA, BUFFER, BUFLEN
```

PARAMETERS

comarea	Before calling VPUTBUFFER, the following COMAREA fields must be set (if not set already):

CSTATUS	Set to zero.
LANGUAGE	Set to code identifying language of calling program.
COMAREALEN	Set to total number of words in COMAREA.

VPUTBUFFER may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.
NUMERRS	May be decremented as a result of new data replacing data in field with error.

buffer	Character string in user program containing the data to be written to the data buffer in memory.
buflen	Integer variable that specifies the number of characters to be written to the data buffer in memory. The comarea item DBUFLEN, that contains the size of the data buffer in memory, may be used as this parameter.

This procedure transfers data from a buffer in the user program to the data buffer in memory. The length of the data moved is based on the number of characters specified in the buflen parameter and the number of characters in the largest data buffer in the forms file, whichever is less. The length of the buffer assigned to the current form is not considered since the user may intend the data for another form with a longer buffer length.

For example, assume there are three forms in the forms file:

Form A	DBUFLEN = 100 characters
Form B	DBUFLEN = 200 characters
Form C	DBUFLEN = 75 characters

In this case, the maximum data buffer length is 200 characters. If the current form is form A and the user calls VPUTBUFFER with a user buffer length (buflen parameter) of 200, he may intend to call VGET-NEXTFORM to get form B and then VSHOWFORM to display form B with the 200 characters of data moved to the data buffer with VPUTBUFFER.

Fewer characters than the data buffer can hold may be transferred; the remaining space in the data buffer is not changed.

The data moved to the data buffer is exactly as it appears in the user program buffer. (If you want the data converted to ASCII in the data buffer, you must use VPUTtype, where type is the data type of the field in the user program.) When the data is displayed, it is moved to each field in the form in sequence from left to right, top to bottom. If any field being replaced by user data contained an error, VPUTBUFFER clears the error flat for the field and decrements NUMERRS.

EXAMPLES

The following calls transfer 24 characters from the user program area, DAT1 to the data buffer; the longest DBUFLEN is assumed to be 80.

COBOL:

```
01   DAT1.
     03   FIRSTNAME PIC X(6)
     03   LASTNAME  PIC X(18).

•
•
•
ACCEPT DAT1.
CALL "VPUTBUFFER" USING COMAREA, DAT1, DBUFLEN.
```

BASIC:

```
235  L1=24
240  CALL VPUTBUFFER(C1(*),D1$,L1)
```

FORTRAN:

```
CALL VPUTBUFFER(COMARE,DAT1,DBLEN)
```

SPL:

```
BYTE ARRAY DAT1(0:23);
•
•
•
VPUTBUFFER(COM1,DAT1,LEN);
```

VPUTFIELD

Writes a value from the user program into a specified field in the data buffer in memory.

```
CALL "VPUTFIELD" USING COMAREA, FIELDNUM, FIELDBUF, BUFLEN,
                  ACTUALEN, NEXTFLDNUM
```

PARAMETERS

comarea — If not already set, the following COMAREA fields must be set before calling VPUTFIELD:

CSTATUS	Set to zero.
LANGUAGE	Set to code identifying language of calling program.
COMAREALEN	Set to total number of words in COMAREA.

VPUTFIELD may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.
NUMERRS	May be decremented if new value is moved to a field which has error flag set.

fieldnum — Integer variable containing the number assigned to the field in the data buffer by FORMSPEC.

fieldbuf — Character string in user program containing the value to be written to the specified data buffer field.

buflen — Integer containing the number of characters to be written from *fieldbuf* to the field identified by *fieldnum.*

actualen — Integer that specifies the number of characters actually written.

nextfldnum — Integer to which VPUTFIELD returns the number of the next field in screen order. If there are no more fields, it returns zero.

The value in the user program is written to the field in the data buffer identified by its field number. Note that the field number is a unique number assigned to the field by FORMSPEC when the form is first created. This number never changes regardless of any changes to the field's position in the form, or to its length. The field number must not be confused with the field's position in the data buffer, which corresponds to its position in the form.

If the field is shorter than the value transferred to it, the value is truncated on the right.

If the field whose value is being replaced contained an error,VPUTFIELD clears the field's error flag, and decrements NUMERRS.

Note that VPUTFIELD does not convert the data. To convert data to ASCII, you must use VPUTtype, where type specifies the data type of the field in the user program.

EXAMPLES

The following calls write a 10-character value from the user program to field number "1" in the data buffer.

COBOL:

```
MOVE 1 TO FIELDNUM.
MOVE 10 TO FIELD-LEN.
MOVE "GADGET    " TO PART-DES.
CALL "VPUTFIELD" USING COMAREA, FIELDNUM, PART-DES
                       FIELD-LEN, DESC-LEN, NEXT-FIELD.
```

BASIC:

```
250 F1=1
255 C=10
257 I$="GADGET    "
260 CALL VPUTFIELD(C1(*),F1,I$,C,A,N)
```

FORTRAN:

```
FLDNUM=1
ICOUNT=10
XITEM="GADGET    "
CALL VPUTFIELD(COMARE,FLDNUM,XITEM,ICOUNT,INUM,FLDNXT)
```

SPL:

```
INTEGER
  FLD'NUM,
  COUNT,
  ACTUAL'LEN,
  NXT'FLD'NUM;
  BYTE ARRAY PART'DES(0:9):="GADGET    ";
  •
  •
  FLD'NUM:=1;
  COUNT:=10;
  VPUTFIELD(COMAREA,FLD'NUM,PART'DES,COUNT,ACTUAL'LEN,
            NXT'FLD'NUM);
```

VPUTtype

Writes a numeric value of specified type from the user program to a field in the data buffer in memory, converting the value to ASCII.

This procedure may be specified as follows:

VPUTINT	writes integer value as ASCII value
VPUTDINT	writes double-integer value as ASCII value
VPUTREAL	writes real value as ASCII value
VPUTLONG	writes long value as ASCII value

```
CALL "VPUTINT" USING COMAREA, FIELDNUM, VARIABLE
```

PARAMETERS

comarea If not set already, the following COMAREA fields must be set before calling VPUTtype:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VPUTtype may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.
NUMERRS	May be decremented if new value replaces the value of a field with an error.

fieldnum Integer variable containing the field number assigned by FORMSPEC to the field in the data buffer to which the value is written.

variable Variable of specified type in the user program that contains the value to be converted to ASCII and written to a field in the data buffer.

Depending on how it is specified, this procedure converts integer, double, real, or long values to ASCII and writes the converted value to a particular field in the data buffer, right justified. (Note that the exact format depends on the data type of the destination field.) The destination field is identified by the field number assigned by FORMSPEC. The field to which the value is written must be defined as a numeric field (type NUM, INT, or DIG).

Note that the number never changes as long as the field exists. It is not changed even if the position of the field in the form is changed, or its length or other characteristics are changed. The field number should not be confused with the position of the field in the data buffer which is based on the field position within the form. Thus, the field number provides a way to locate fields regardless of their position.

If the specified field had an error, VPUTtype clears the field's error flag, and decrements NUMERRS.

Refer back to table 6-6 under VGETtype for the format of each of the data types that may be converted. Note that COBOL does not have type real or long, and BASIC does not have a double integer data type.

EXAMPLES

The following calls convert an integer value of 25 in the user program to ASCII and write it to field "4" in the data buffer in memory:

COBOL:

```
77  FIELD-NUM PIC S9(4)   COMP.
77  ITEM   PIC  S9(4)  COMP.
•
•
•
MOVE  4 TO FIELD-NUM.
MOVE 25 TO ITEM.
CALL "VPUTINT" USING COMAREA, FIELD-NUM, ITEM.
```

BASIC

```
260  F1=4
263  I=25
265 CALL VPUTINT(C1(*),F1,I)
```

FORTRAN:

```
FIELD=4
ITEM=25
CALL VPUTINT(COMAREA,FIELD,ITEM)
```

SPL:

```
INTEGER FIELD,ITEM;
•
•
•
FIELD:=4;
ITEM:=25;
VPUTINT(COMAREA,FIELD,ITEM);
```

VPUTWINDOW

Moves a message to the window area of memory.

```
CALL "VPUTWINDOW" USING COMAREA, MESSAGE, LENGTH
```

PARAMETERS

comarea

The following COMAREA fields must be set before calling VPUTWINDOW (unless already set):

LANGUAGE	Set to code identifying language of calling program.
COMAREALEN	Set to total number of words in COMAREA.

VPUTWINDOW may set the following COMAREA fields:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

message

Character string containing the message to be displayed to the window area of memory.

length

Integer that specifies the number of characters in the message. If set to zero, any message in the window is cleared to blanks. The maximum length is 150 characters, but only 80 can be printable characters.

This procedure moves the specified message to the window area of memory. A subsequent call to VSHOWFORM can be used to display the message in the window area of the terminal screen, with the window enhanced as indicated by WINDOWENH. A message written by VPUTWINDOW overwrites any previous message in the window area, including any message written by a previous call to VPUTWINDOW or VSETERROR.

If the message is longer than the defined window length, the message is truncated on the right. If shorter, the rest of the window line is cleared.

Note that the forms file may be defined with no window line for error and status messages. In this case, the message is ignored.

EXAMPLES

The following calls write the message "ENTER ORDERS ON THIS FORM" to the window area of memory:

COBOL:

```
MOVE "ENTER ORDERS ON THIS FORM" TO MESSAGE.
MOVE 25 TO MSG-LENGTH.
CALL "VPUTWINDOW" USING COMAREA, MESSAGE, MSG-LENGTH.
```

BASIC:

```
310  M1$="ENTER ORDERS ON THIS FORM"
320  L1=25
330  CALL VPUTWINDOW(C1(*),M1$,L1)
```

FORTRAN:

```
MSG="ENTER ORDERS ON THIS FORM"
LEN=25
CALL VPUTWINDOW(COMARE,MSG,LEN)
```

SPL:

```
BYTE ARRAY MSG(0:24):="ENTER ORDERS ON THIS FORM";
INTEGER LEN;
•
•
LEN:=25;
VPUTWINDOW(COMAREA,MSG,LEN);
```

VREADBATCH

Reads contents of current batch record into data buffer in memory.

```
CALL "VREADBATCH" USING COMAREA
```

PARAMETERS

comarea Must be name used when the batch file was opened with VOPENBATCH. The following COMAREA fields must be set before calling VREADBATCH (if not set already):

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.
RECNUM	Set to number of record in open batch file from which data is to be read (records are numbered from zero).

VREADBATCH sets the following COMAREA fields:

NFNAME	Set to name of form associated with the data read from batch file; (used by VGETNEXTFORM to retrieve this form from forms file).
DBUFLEN	Set to length of data buffer (in characters) based on length of data read from batch record.
DELETEFLAG	Set to TRUE (all 1's) if delete flag in batch record indicates record is deleted; set to FALSE (all zeros) otherwise.
CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

This procedure is used primarily in browse/modify mode. It enables an operator to view the data in the batch file. VREADBATCH may also be used to bring the data from a batch file into the data buffer so that it can be retrieved from a user program with the VGETBUFFER, VGETFIELD, or VGETtype procedures.

VREADBATCH reads the record (RECNUM) in the batch file opened by VOPENBATCH. The record is read into memory where VREADBATCH extracts the batch record control information (refer to the VOPENBATCH description). This information includes the current form name which is moved to NFNAME, the delete flag which is moved to DELETEFLAG, and the data length in characters which is moved to DBUFLEN. The actual data is in the data buffer in memory.

If the batch file is not on a direct-access device (that is, it is on a device such as magnetic tape), or if the batch file was created with variable-length records, the browse/modify mode cannot be used.

EXAMPLES

The following calls read the batch record specified by RECNUM; updates the COMAREA according to the batch record information stored with the data, and puts the data in the data buffer in memory:

COBOL:	CALL "VREADBATCH" USING COMAREA
BASIC:	175 CALL VREADBATCH(C1(*))
FORTRAN:	CALL VREADBATCH(COMARE)
SPL:	CALL VREADBATCH(COMAREA):

VREADFIELDS

Accepts all user input from an open terminal, including data entered by pressing ENTER, or operator requests made by pressing a function key.

```
CALL "VREADFIELDS" USING COMAREA
```

PARAMETERS

comarea Must be COMAREA named when terminal file was opened. The following COMAREA items must be set before calling VREADFIELDS (if not already set):

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

VREADFIELDS sets the following COMAREA values:

NUMERRS	Set to zero.
LASTKEY	Set to code for terminal function key pressed by operator.
CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

VREADFIELDS accepts operator-entered data when the ENTER key is pressed. It maps the data into the data buffer in memory. The data is mapped in screen order, from left to right, top to bottom. If there are any display-only fields already in the buffer, the fields read by VREADFIELDS are interspersed among the display-only fields according to the screen order. VREADFIELDS then sets LASTKEY to zero. For the HP 3075/6 terminals, this intrinsic issues the form and performs a read on a field by field basis, but editing is still done on a form basis.

If a function key was pressed, VREADFIELDS sets LASTKEY to the corresponding number: 1 for the f1 key, 2 for the f2 key, and so forth. On the HP 3075/6 VREADFIELDS sets LASTKEY as follows: -1 is the ATTENTION key, 0 is the ENTER key, 1 is the "A" key, 2 is the "B" key . . . and 26 is the "Z" key.

Application programs must supply the code to perform the functions requested by pressing any of the function keys. The particular function assigned to a key is determined only by how the program processes the key code passed to it by VREADFIELDS. (Refer to appendix A for the code used by ENTRY to process the functions assigned to these keys for V/3000 data entry.)

When entering data on the HP 3075/6, if a function key is pressed, VREADFIELDS returns to the calling procedure and does NOT continue presentation of remaining fields.

EXAMPLES

Each of the following calls accepts operator input from the terminal, transfers any data to the data buffer, and sets LASTKEY:

COBOL:	CALL "VREADFIELDS" USING COMAREA.
BASIC:	130 CALL VREADFIELDS(C1(*))
FORTRAN:	CALL VREADFIELDS (COMARE)
SPL:	VREADFIELDS(COMAREA);

VSETERROR

Sets the error flag of a specified field and increments NUMERRS. If this is the first field (in screen order) with an error, it moves a message to the window area of memory for later display.

```
CALL "VSETERROR" USING COMAREA, FIELDNUM, MESSAGE, MSGLEN
```

PARAMETERS

comarea — If not already set, the following COMAREA fields must be set before calling VSETERROR:

CSTATUS	Set to zero.
LANGUAGE	Set to code identifying language of calling program.
COMAREALEN	Set to total number of words in COMAREA.

VSETERROR may set the following COMAREA fields:

NUMERRS	Contains number of fields in form with errors; incremented by VSETERROR.
CSTATUS	Set to non-zero value if call unsuccessful.

fieldnum — Integer variable containing the field number of the data field to be flagged for error.

message — Character string containing an error message to be stored in the window area of memory for subsequent display by VSHOWFORM.

length — Integer containing the length in characters of the message parameter. If length is set to –1, the current content of the window is not changed. If length is set to zero, the current content of the window is cleared to blanks.

This procedure can be called by any program that wants to perform its own edits, either in addition to, or instead of VFIELDEDITS. VSETERROR sets the error flag associated with the specified field. If this is the first time this field has been diagnosed as having an error, VSETERROR increments NUMERRS; otherwise, it does not change NUMERRS. Thus, if field number "1" has an error detected by a prior call to VFIELDEDITS, a call to VSETERROR for that field does not increment NUMERRS.

If this is the first field in the form (in screen order) that has an error, the specified message is moved to the window area of memory for later display. If you do not want to change the current contents of the window, set the length parameter to –1. To clear a message, set length to zero.

Note that the field number identifies a field regardless of its position in the form. Thus, field number "1" could be the third field in screen order counting from left to right, top to bottom.

EXAMPLES

The following examples set error flags for field number "3" in the currently open form, and set up the message "THIS FIELD IS REQUIRED" to be displayed if no value is entered in the field and this is the first field (in screen order) in which an error is detected:

COBOL:

```
DATA DIVISION.
    77  FLDNUM       PIC 9(4) COMP.
    77  MESSAGE      PIC X(80).
    77  MLENGTH      PIC S9(4) COMP.
    •
    •

PROCEDURE DIVISION.
    MOVE 3 TO FIELDNUM.
    MOVE 22 TO MLENGTH.
    MOVE "THIS FIELD IS REQUIRED" TO MESSAGE.
    CALL "VSETERROR" USING COMAREA, FLDNUM, MESSAGE, MLENGTH.
```

BASIC:

```
220  F1=3
225  L1=22
230  M$="THIS FIELD IS REQUIRED"
250  CALL VSETERROR(C1(*),F1,M$,L1)
```

FORTRAN:

```
FF=3
ML=22
MSG="THIS FIELD IS REQUIRED"
CALL VSETERROR(COMAREA,FF,MSG,ML)
```

SPL:

```
INTEGER FF,ML;
BYTE ARRAY MESSAGE(0:21);="THIS FIELD IS REQUIRED";
•
•
FF:=3;
ML:=22;
VSETERROR(COMAREA,FF,MESSAGE,ML);
```

The following examples show how VSETERROR can be used to set error flags for a field in error without writing a message to the window:

COBOL:

```
MOVE 3 TO FIELDNUM.
MOVE -1 TO MLENGTH.
CALL "VSETERROR" USING COM1, FIELDNUM, MESSAGE, MLENGTH.
```

BASIC:

```
220  F1=3
230  L1=-1
CALL VSETERROR(C1(*),F1,M$,L1)
```

FORTRAN:

```
FF=3
ML=-1
CALL VSETERROR(COM1,FF,MSG,ML)
```

SPL:

```
FIELD:=3;
LEN:=-1;
VSETERROR(COM1,FIELD,MESSAGE,LEN);
```

VSETKEYLABEL

Allows for temporarily setting, programmatically, a new label for a function key.

```
CALL "VSETKEYLABEL" USING COMAREA,FORM-OR-GLOB,KEY-NUM,LABEL
```

PARAMETERS

comarea	If not already set, the following COMAREA must be set prior to a call to VSETKEYLABEL: CSTATUS — Set to zero. COMAREALEN — Set to total number of words in COMAREA.
form-or-glob	Integer value specifying which type of label is to be temporarily set. 0 - Set GLOBAL label. 1 - Set current form label.
key-num	Integer specifying which function key is to be set.
label	A byte array containing the text for the label.

VSETKEYLABEL is only a temporary setting of a new label for a individual function key. Use of this procedure will not change the label definition made in FORMSPEC. (Note only one function key can be set with this procedure.)

The temporary label will be displayed after the next call to VSHOWFORM. If the temporary label is GLOBAL, it will remain active until the forms file is closed or it is replaced by a new GLOBAL label. If the temporary label is for the current form only, it will be replaced when the next form is retrieved or when a new current form label is loaded.

EXAMPLES

The following examples illustrate a call to VSETKEYLABEL in each of the programming languages:

COBOL:

```
77  FORM-OR-GLOB       PIC  S9(4)COMP.
77  KEY-NUM            PIC  S9(4)COMP.
77  KEY-LABEL          PIC    X(16).
.
.
MOVE 1 TO FORM-OR-GLOB.
MOVE 1 TO KEY-NUM.
MOVE " LABEL        1         " TO KEY-LABEL.
CALL "VSETKEYLABEL" USING COMAREA,FORM-OR-GLOB,
        KEY-NUM, KEY-LABEL.
```

BASIC:

```
10  INTEGER F,N
20  DIM L$[16]
30  F=1
40  N=2
50  L$=" LABEL        1        "
60  CALL VSETKEYLABEL(C[*],F,N,L$)
```

FORTRAN:

```
INTEGER FORMORGLOB,KEYNUM
CHARACTER*16 LABEL
FORMORGLOB=1
KEYNUM=1
LABEL=" LABEL      1       "
CALL VSETKEYLABEL(COMAREA,FORMORGLOB,KEYNUM,LABEL)
```

SPL:

```
INTEGER
      FORM'OR'GLOB,
      KEY'NUM;
BYTE ARRAY
      KEY'LABEL(0:15);
.
.
FORM'OR'GLOB:=1;
KEY'NUM:=1;
MOVE KEY'LABEL:=" LABEL      1       ";
VSETKEYLABEL(COMAREA,FORM'OR'GLOB,KEY'NUM,
      KEY'LABEL);
```

VSETKEYLABELS

Allows for temporarily setting, programmatically, labels for function keys.

```
CALL "VSETKEYLABELS" USING COMAREA,FORM-OR-GLOB,NUM-OF-LABELS,LABELS
```

PARAMETERS

comarea — If not already set, the following COMAREA fields must be set before calling VSETKEYLABELS:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMARA.

form-or-glob — Integer value specifying which type of labels are to be temporarily replaced.

0 - Replace GLOBAL labels.
1 - Replace current form labels.

num-of-labels — Integer value indicating how many labels are to be temporarily set. This value may be zero, where 0 indicates that the labels defined in FORMSPEC should be set.

labels — A byte array in which the labels will be defined. The length of the array must be at least NUM'OF'LABELS * 16.

VSETKEYLABELS is only a temporary setting of new labels for the function keys. Use of it will not change the label definitions made in FORMSPEC.

The temporary labels will be displayed after the next call to VSHOWFORM. If the temporary labels are GLOBAL, they will remain active until the forms file is closed or replaced by new GLOBAL labels. If the temporary labels are current form labels, they will be replaced when the next form is retrieved or when new current form labels are loaded.

EXAMPLES

The following examples illustrate a call to VSETKEYLABELS in each of the programming languages:

COBOL:

```
77 FORM-OR-GLOB       PIC  S9(4)COMP.
77 NUM-OF-LABELS      PIC  S9(4)COMP.
77 KEY-LABELS         PIC   X(32).
.
.
MOVE 1 TO FORM-OR-GLOB.
MOVE 2 TO NUM-OF-LABELS.
MOVE " LABEL      1        LABEL        2        "
      TO KEY-LABELS.
CALL "VSETKEYLABELS" USING COMAREA,FORM-OR-GLOB,
      NUM-OF-LABELS,KEY-LABELS.
```

BASIC:

```
10 INTEGER F,N
20 DIM L$[32]
30 F=1
40 N=2
50 L$=" LABEL      1        LABEL       2        "
60 CALL VSETKEYLABELS(C[*],F,N,L$)
```

FORTRAN:

```
INTEGER FORMORGLOB,NUMLABELS
CHARACTER*32 LABELS
FORMORGLOB=1
NUMLABELS=2
LABELS=" LABEL      1        LABEL       2        "
CALL VSETKEYLABELS(COMAREA,FORMORGLOB,NUMLABELS,
        LABELS)
```

SPL:

```
INTEGER
        FORM'OR'GLOB,
        NUM'OF'LABELS;
BYTE ARRAY
        LABELS(0:31);
.
.
FORM'OR'GLOB:=1;
NUM'OF'LABELS:=2;
MOVE LABELS:=" LABEL      1        LABEL       2        ";
VSETKEYLABELS(COMAREA'FORM'OR'GLOB,NUM'OF'LABELS,
        LABELS);
```

VSHOWFORM

Displays on terminal screen, the current form from the form definition buffer, any data in the data buffer, and any messages from the window buffer.

```
CALL "VSHOWFORM" USING COMAREA
```

PARAMETERS

comarea The COMAREA name must be one used by VOPENTERM to open the terminal file. If not already set, the following COMAREA items must be set prior to calling VSHOWFORM:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.
WINDOWENH	Set to window enhancement code; may be set before call to code for non-default enhancement; otherwise, set by VOPENFORMF to default enhancement specified in forms file. If set to zero, window is not enhanced. (Refer to WINDOWENH discussion under "Communication Area" earlier in this section.)

Normally, VSHOWFORM writes only changed information from the window, the data buffer, or the form. For exceptional situations, you can override this optimization by setting the following COMAREA value:

SHOWCONTROL	If set to non-zero value, override VSHOWFORM optimizations. (Particular SHOWCONTROL settings are described below.)

VSHOWFORM sets the following COMAREA values:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.

VSHOWFORM displays, on an open terminal screen, the form currently stored as a form image in the form definition area of memory and function key label. Any enhancements specified for the form are used for the display. Data currently in the data buffer in memory is moved to the appropriate fields of the displayed form. Any message in the window buffer of memory is displayed in the line of the form selected as the status line.

In order to optimize VSHOWFORM performance, only changed information is written to the terminal. Thus, if the window has not been changed by VPUTWINDOW since the last execution of VSHOWFORM, or if the current form is being repeated in place, these areas are not rewritten. Also, when a form is repeating in place, only the changed values in the data buffer area of memory are written by VSHOWFORM. This will be sufficient for most applications; however, these three optimizations (form, data, window) can be overridden by setting SHOWCONTROL.

SHOWCONTROL is a logical value in COMAREA that, when set to a non-zero value disables the VSHOWFORM optimizations. This means that VSHOWFORM will write data or a form or the window to the terminal whether or not it has changed. SHOWCONTROL can be set as follows:

bit 15 = 1	display form
14 = 1	display data including enhancements
13 = 1	display window line

Any combination of these bits may be set. For example, if you want to force a write of all data:

0 7 8	13	14	15
∅————————∅	0	1	0

SHOWCONTROL = octal 2

Or, if you want to force the window to be written:

0 7 8	13	14	15
∅————————∅	1	0	0

SHOWCONTROL = octal 4

Note that a prior call to VGETNEXTFORM is used to read the form into the form definition area of memory. This form definition includes not only the form image, but also field editing specifications and all enhancements.

The field enhancements are specified with the form definition. If a field has an error, VSHOWFORM changes its enhancement to the error enhancement defined for the form by FORMSPEC.

The data buffer may contain data as a result of initialization by VINITFORM, retrieval of operator-entered data by VREADFIELDS, data formatting or movement caused by editing specified with each field and executed by VINITFORM, VFIELDEDITS, or VFINISHFORM. Data may also be transferred directly to the data buffer from a user program with VPUTBUFFER or from a batch file with VREADBATCH.

The window contains any message set by VSETERROR or VPUTWINDOW.

EXAMPLES

The following calls display a form with optional data and enhancements on the terminal screen opened with the COMAREA, COM1:

COBOL:	CALL "VSHOWFORM" USING COMAREA
BASIC:	120 CALL VSHOWFORM(C1(*))
FORTRAN:	CALL VSHOWFORM(COMARE)
SPL:	VSHOWFORM(COMAREA);

VUNLOADFORM

Allows for the form named in WHICH-FORM to be unloaded from the terminal.

```
CALL "VUNLOADFORM" USING COMAREA, WHICH-FORM
```

PARAMETERS

comarea If not already set, the following COMAREA must be set prior to a call to VUNLOADFORM:

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.

which-form The name of the form to be removed from local form storage. Each name is 15 characters. The dummy byte should be placed at the 16th character slot of the array.

This intrinsic is to be used with user application programs being written for use on terminals with local form storage as a feature.

VUNLOADFORM is used to remove one form at a time from the terminal. It is to be used with VLOAD-FORMS to control which forms are in the terminal.

EXAMPLES

The following examples illustrate a call to VUNLOADFORM in each of the programming languages:

COBOL:

```
77 WHICH-FORM          PIC X(16).
.
.
MOVE "FORMA                 " TO WHICH-FORM.
CALL "VUNLOADFORM" USING COMAREA,WHICH-FORM.
```

BASIC:

```
 10 DIM W$[16]
 20 W$="FORMA             "
100 CALL VUNLOADFORM(C[*],W$)
```

FORTRAN:

```
CHARACTER*16 WFORM
WFORM="FORMA                 "
CALL VUNLOADFORM(COMAREA,WFORM)
```

SPL:

```
BYTE ARRAY WHICH'FORM(0:15);
.
.
WHICH'FORM:="FORMA              ";
VUNLOADFORM(COMAREA,WHICH'FORM);
```

VWRITEBATCH

Writes a record to the batch file from the data buffer in memory, or deletes a record from the batch file.

```
CALL "VWRITEBATCH" USING COMAREA
```

PARAMETERS

comarea Must be name specified when batch file was opened with VOPENBATCH. You must set the following COMAREA fields before calling VWRITEBATCH (if not set already):

CSTATUS	Set to zero.
COMAREALEN	Set to total number of words in COMAREA.
RECNUM	Set to number in batch file to which data is to be written (record numbers start with zero). RECNUM is initialized to zero by VOPENBATCH.
DELETEFLAG	Set to TRUE (all 1's) if record is deleted.

(CFNAME and DBUFLEN should be set by a prior call to VGETNEXTFORM.)

VWRITEBATCH may set the following COMAREA values:

CSTATUS	Set to non-zero value if call unsuccessful.
FILERRNUM	Set to file error code if MPE file error.
NUMRECS	Incremented each time a new record is written; decremented if record is deleted.

VWRITEBATCH writes the contents of the data buffer to the record specified by RECNUM in an open batch file. (RECNUM must be maintained by the calling program.) VWRITEBATCH writes the following information to the batch record:

- Contents of the data buffer
- Batch record control information (from COMAREA):

DELETEFLAG	TRUE if record is deleted
CFNAME	Name of the form associated with this data.
DBUFLEN	Length (in characters) of the data buffer.

(Refer to "Record Format" in the VOPENBATCH description for a diagram of the batch record.)

VWRITEBATCH is used in both the data collection and browse/modify modes.

DATA COLLECTION MODE

The data in the data buffer is entered on a particular form displayed at a terminal and then read into the data buffer by VREADFIELDS.

VWRITEBATCH can then be called to write the data buffer and the record control information to the batch record specified by RECNUM.

BROWSE/MODIFY MODE

When data is modified, an existing batch file record is rewritten. The calling program must set RECNUM to the record number of this record.

To mark a batch record as deleted, the DELETEFLAG must be set to TRUE (all 1's) by the user program. Then a call to VWRITEBATCH sets a corresponding flag in the batch record to mark the record as deleted. Since a deleted record still exists in the batch file, it can be viewed through FCOPY or a user-written procedure.

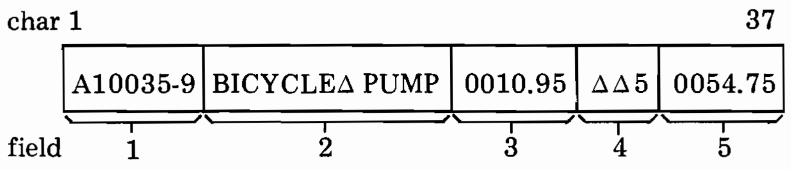
In the V/3000-supplied program ENTRY, a function key pressed by the operator not only determines which record is to be viewed, but also specifies which record is to be deleted.

EXAMPLES

The following calls write the contents of the data buffer to the batch record identified by RECNUM:

COBOL:	CALL "VWRITEBATCH" USING COMAREA.
BASIC:	165 CALL VWRITEBATCH(C1(*))
FORTRAN:	CALL VWRITEBATCH(COMARE)
SPL:	VWRITEBATCH(COMAREA);

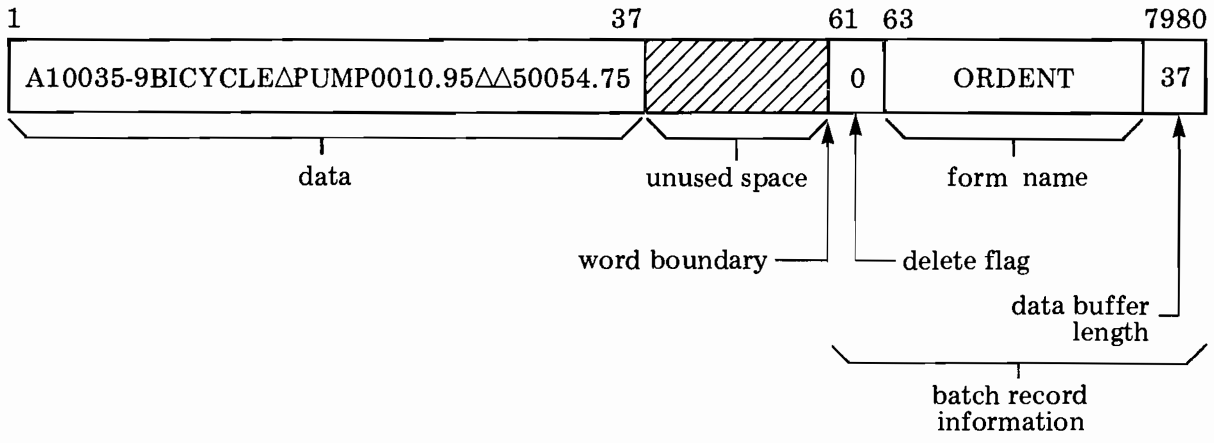
Assume the following data is in the data buffer and that it was entered on form, ORDENT. (Note that the data entered on separate fields of a form is concatenated in the data buffer, with no separators.)



Assume the COMAREA is set as follows:

```
RECNUM  = 5
CFNAME  = ORDENT
DBUFLEN = 37
```

And, assume the batch file opened by VOPENBATCH has fixed-length records, 80 characters long. The call to VWRITEBATCH writes the following record as the sixth record in the batch file:



RPG INTERFACE TO V/3000

SECTION VII

OVERVIEW

The RPG interface to V/3000 provides functions similar to those provided by the V/3000 procedures for other languages. Since RPG does not make it easy to call the V/3000 procedures, the interface is designed to operate in a manner familiar to RPG programmers. Using the interface, the RPG programmer can:

- Get a form from the forms file and display it at the terminal.
- Display a message in window area of the terminal screen.
- Display initial values specified through FORMSPEC in fields displayed on the screen.
- Accept input from the terminal.
- Determine if the input contains errors and, if so, flag the fields in error and display an error message.
- Write data accepted from the terminal to the user program or to a batch file.
- Transfer data from the user program or a batch file to the terminal screen.
- Transfer data between the user program and a batch file.

These features provide a complete interface between a user program, the terminal, a forms file, the entered data, and, for data entry, the batch file to which entered data is written.

Note that appendix A contains a listing of the RPG version of program ENTRY. Since program ENTRY controls standard data entry using V/3000, the RPG interface described in this section need be used only to modify ENTRY to suit particular applications, or to provide an interface between an existing application and the facilities provided by FORMSPEC.

OPERATION OF THE RPG INTERFACE

So that the RPG interface to V/3000 appears natural to RPG programmers, all input to, output from, and commands that control the data entry facility are treated as input/output to a file. This file is identified by the special device class name:

```
WORKSTN
```

In effect, you request the interface to V/3000 by specifying "WORKSTN" in the device field of a File Description Specification. "WORKSTN" is *not* a system configured name.

WORKSTN FILE

The file assigned to this device can have any legal file name, but must be specified as an update file. The file is usually designated as a demand file, but may be a primary file. RPG executes update primary files within the normal RPG cycle. For example, an update primary file might be used when an application is limited to a single cycle such as showing a form, reading data entered on the form, and writing the data to a file. For most applications, however, the WORKSTN file is specified as update demand. For update demand files, the program stays within the calculation section, and uses demand reads and exception output.

There may be only one file per program assigned to the device file WORKSTN. This file must be defined with variable length records.

OTHER FILES

Three other files are specified as continuation files in the same File Description Specification. These three files are:

• Forms file	— contains the forms designed through program FORMSPEC. (File is required.)
• Batch file	— file to which data entered on form is written during standard V/3000 data entry. (May be omitted.)
• Trace file	— file that contains a record of every action and event during execution of RPG interface, and on which runtime errors are recorded. (May be omitted.)

ACTIONS AND EVENTS

The operation of the RPG V/3000 interface is specified in terms of "actions" and "events". Actions are output records written to the file; they carry out all the V/3000 functions. Events are input records read from the file; they specify input from the terminal, or from the V/3000 buffer.

Actions include, but are not restricted to:

Get next form to display
Display form and data
Read input from terminal
Write data buffer to batch file

Events include:

ENTER key was pressed
Function key was pressed
V/3000 editing error occurred

RUNTIME ERRORS

RPG allows the programmer to specify the action to take in case of a runtime error. The action can be one of six options specified on the Control Record Specification. If no option is specified, the system asks the operator to specify an option by means of a function key.

A stack dump is provided as one of the RPG error options. Since the WORKSTN programs are run as sessions, the program can redirect the dump to a file. If no file is specified, the default file is $STDLIST.

A further aid is the trace file. When runtime problems occur, the trace file can be used to determine where the problem occurred.

Error messages are displayed at the terminal unless suppressed. Each message is displayed for three seconds unless this interval is increased or decreased.

BREAK KEY DISABLED

On many 264x terminals, the break key is physically positioned near the ENTER key. As a result, it is easy for the terminal operator to press the BREAK key by accident. Since it is difficult to recover from a break, the BREAK key is disabled when a WORKSTN file is used.

If you feel you need the BREAK key, you can enable it with a notation on the File Description Specification for the WORKSTN file.

USING THE RPG INTERFACE

All actions and events have a two-digit code. These codes are integers in the range 00 through 99. Codes from zero through 49 denote events; codes from 50 through 99 denote actions. Events codes are specified in columns 1-2 of each record.

Any numeric action code can be replaced by a six-character mnemonic in columns 1-6 of the output record. The mnemonic differs from the numeric code only in that it is easier to remember. On the other hand, performance may be improved by using numeric codes rather than the action names.

Events and actions are treated programmatically as input and output records. Events can be identified by record indicators on Input Specifications. Actions are specified as fields on Output Specifications.

DATA ENTRY THROUGH RPG

Data Entry using V/3000 is essentially a three step process:

1. Enter data at the terminal.
2. Transfer the data to the V/3000 buffer; perform FORMSPEC edits.
3. Transfer data from the V/3000 buffer to a batch file or to a user program buffer.

Output to the terminal reverses these steps.

If data editing is specified by FORMSPEC, this editing is performed on the data in the V/3000 buffer. Since many RPG applications may not use the FORMSPEC editing features, the RPG interface is designed so that the application can skip step 2, and write data directly to a user program, or vice versa.

EVENTS

The user program requests the return of an event by performing a read operation on the WORKSTN file. Event codes indicate the type of input passed to WORKSTN. The input is either received from the terminal or is in response to a particular action code. Input data is transferred to the user program according to the WORKSTN file input specifications.

Events 00 through 08 indicate which terminal key was pressed by the operator. If ENTER was pressed (event 00), then data has been entered and can be read. If any function key was pressed (events 01-08), the RPG programmer must specify the action to be performed in response. (Refer to appendix A for an example of how the RPG program ENTRY handles the function keys.)

The remaining event codes respond to actions specified in the RPG program. These actions involve the V/3000 data buffer, so event codes greater than 08 will probably be used only by those applications that take full advantage of the V/3000 data handling features.

All event records contain the name of the current form in columns 3-17. This allows the program to associate the data with the form on which it was entered.

Refer to table 7-1 for the complete list of the event codes, the functions they perform, and the actions to which they may be a response.

Table 7-1. Event Codes

Code	Function	Response to Action Code
00	Terminal operator pressed the ENTER key. Data is included in the record.	54 (RDTERM)
01	Terminal operator pressed f1 key; no data.	54 (RDTERM)
02	Terminal operator pressed f2 key; no data.	54 (RDTERM)
03	Terminal operator pressed f3 key; no data.	54 (RDTERM)
04	Terminal operator pressed f4 key; no data.	54 (RDTERM)
05	Terminal operator pressed f5 key; no data.	54 (RDTERM)
06	Terminal operator pressed f6 key; no data.	54 (RDTERM)
07	Terminal operator pressed f7 key; no data.	54 (RDTERM)
08	Terminal operator pressed f8 key; no data.	54 (RDTERM)
09	Read number of fields that failed V/3000 or user edits.	59 (EDITS) or 61 (NUMERR)
10	Read data from V/3000 data buffer.	64 (GETDTA)
11	Return record number of current batch record, mode of operation (0 = collect, 1 = browse), repeat/append and freeze/append status, and next form name.	Any action (or no action) except 54, 59, 61, 64 or 74.
12	Return length and contents of a particular field in the V/3000 data buffer.	74 (GETFLD)

ACTIONS

For simple applications that bypass V/3000 editing, RPG programmers need use only actions numbered 50 through 57. These actions are designed to allow direct interaction between a terminal and a user program. Any editing must be specified in the user program rather than by FORMSPEC.

For more complex applications the RPG programmer can use the full set of actions that provide access to all the V/3000 capabilities.

Refer to table 7-2 for a list of the action codes with their functions.

Table 7-2. Action Codes

Code	Mnemonic	Function	Corresponding V/3000 Procedure
50	CHGNXT	Specify next form name and whether form is repeat/append, freeze/append, or normal.	–
51	GETNXT	Get the next form from the forms file, and set repeat/append, and freeze/append status. If repeat mode set, the form is not actually retrieved. Follow this action by action 53 to display form at terminal.	VGETNEXTFORM
52	PUTMSG	Specify a message, with any enhancements to be displayed on the error/status line on the terminal screen (the "window"). The message is not displayed until action 53 is executed.	VPUTWINDOW
53	SHOW	Display current form, any initial data, and any message on the terminal screen.	VSHOWFORM
54	RDTERM	Read input from terminal to the V/3000 data buffer. This action must be followed by a read to transfer the data to the user program; the record read is an event type 00-08.	VREADFIELDS VGETBUFFER
55	SHOMSG	Display message specified in user program with any enhancements, in error/status line of terminal screen (the "window"). (Action combines actions 52 and 53.) If data in the V/3000 buffer has changed, the new data is displayed.	VPUTWINDOW VSHOWFORM
56	CORERR	Indicate fields that failed user edits. The output record contains the number of the bad field, and a message with its length and any enhancements. The current form and the message are displayed on the screen and the user's response is read. Fields with errors are enhanced if enhancements were specified. Follow this action with a read of the user response (event type 00–08). This action combines actions 62, 53, and 54.)	VSETERROR VSHOWFORM VREADFIELDS VGETBUFFER
57	SHODTA	Display data from the user program buffer to fields on ** screen. (This action combines actions 63 and 53.)	VPUTBUFFER VSHOWFORM
The actions to this point can all be used to transfer data between a user program and the terminal according to forms specified by FORMSPEC. The remaining actions are intended for users who want to make use of the full V/3000 capability.			
58	INIT	Initialize fields in the current form according to initialization specified for the field through FORMSPEC. If errors, move message to window.	VINITFORM VERRMSG VPUTWINDOW
59	EDITS	Perform edits specified for the fields in the current form; display message describing the first error in the terminal window. This action should be followed by a read (event type 09); the record read contains the number of fields that failed the edit.	VFIELDEDITS VERRMSG VPUTWINDOW
60	PRINT	Print current form with its data on line printer. Form must have been read from the forms file by a prior call to action 51.	VPRINTFORM
61	NUMERR	Ask for error status. Follow output of this action by a read (event type 09); the record read indicates the number of fields that failed edits.	–

Table 7-2. Action Codes (Continued)

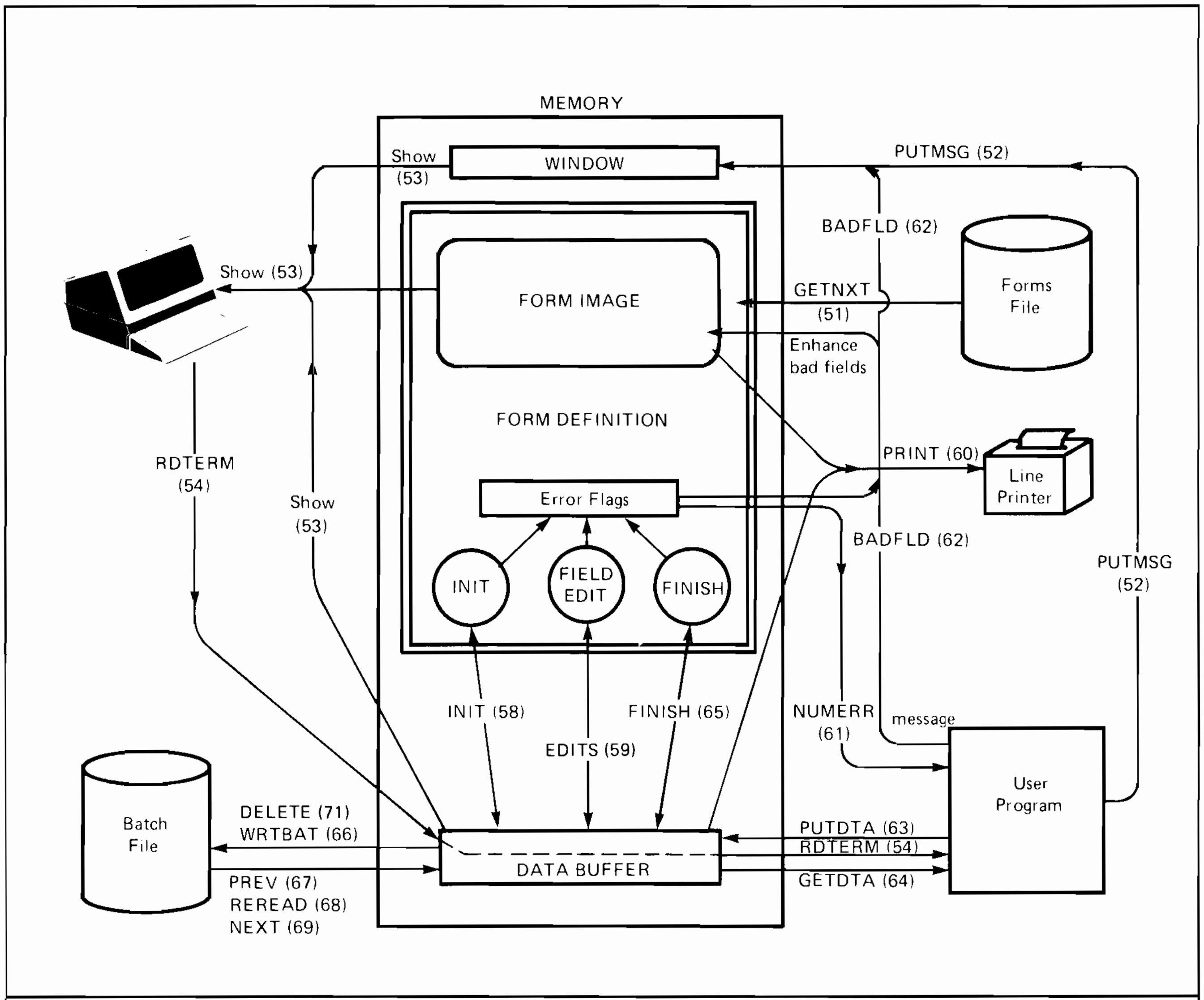
Code	Mnemonic	Function	Corresponding V/3000 Procedure
62	BADFLD	Indicate that field failed a user edit. The output record contains the number of the bad field and a message for the window.	VSETERROR
63	PUTDTA	Replace the data in the V/3000 data buffer with values specified in the user program buffer.** The output record contains data for the V/3000 data buffer and the length of the data.	VPUTBUFFER
64	GETDTA	Write the data in the V/3000 data buffer to the user program buffer. This action should be followed by a read (event type 10); the record read contains the data from the V/3000 buffer and its length.	VGETBUFFER
65	FINISH	Perform all the final processing that is specified as part of the "finish" phase for the form. If errors, move message to window.	VFINISHFORM VERRMSG VPUTWINDOW
66	WRTBAT	Write the contents of the V/3000 data buffer to the batch record corresponding to the current record number. If in collect mode, the current record number is then advanced by one. If in browse mode, the current record number is unaffected.	VWRITEBATCH
67	PREV	Read data from the previous batch record to the VIEW data buffer.*** If not in browse* mode, place program in browse mode. Save the current location in the batch file and the current form name.	VREADBATCH
68	REREAD	Reread data from current batch record into V/3000 data buffer. Program must be in browse mode (requested by prior call to action 67).	VREADBATCH
69	NEXT	Read data from next batch record into VIEW data buffer.*** Program must be in browse mode (requested by prior call to action 67).	VREADBATCH
70	RESUME	Return to collect mode from browse mode. Program must be in browse mode (prior call to action 67). Next form name and location in batch file is restored.	–
71	DELETE	Delete current batch record. Program must be in browse mode (prior call to action 67).	–
72	RDBTNU	Read batch record identified by its record number.	–
73	CLRMSG	Clear the message window. If CLRMSG is followed by the letter "I", clear the message from both the terminal screen and from the window buffer in memory; otherwise, clear only the window buffer.	VPUTWINDOW VSHOWFORM
74	GETFLD	Locate contents of specified field in V/3000 data buffer. Follow this action by a read (event 12) to transfer the data and its length to the user program.	VGETFIELD
75	PUTFLD	Transfer data from user program buffer to specified field in V/3000 data buffer.	VPUTFIELD

*Browse mode is the mode of operation in which existing data in the batch file may be examined and modified. New batch records cannot be added during browse mode.

**The user program buffer may contain old data from a previous action or event. To clear the buffer before updating, specify "ADD" in columns 16 to 18 of the first output specification for this action (you must also place an "A" in column 66 of the WORKSTN File Description Specification).

***Any attempt to read past either end of the batch file will cause the current record number to be set to the out-of-bounds value (-1 for PREV, and End-of-Batch +1 for NEXT), although no read will actually be attempted. **No other indication is given to the user program that the batch file is out-of-bounds.**

Refer to figure 7-1 for an overview of the relation between the terminal, the forms file, the batch file, the user program and the areas of memory used by V/3000. The figure illustrates how the RPG action codes are used to transfer information between these areas.



CHGNXT(50)	specify next form name and define form sequence
SHOMSG(55)	PUTMSG (52) and SHOW (53)
CORERR(56)	BADFLD (62) + SHOW (53) + RDTERM (54)
SHODTA(57)	PUTDTA (63) + SHOW (53)

Figure 7-1. Relation of RPG Action Codes to V/3000

RECORD TYPES

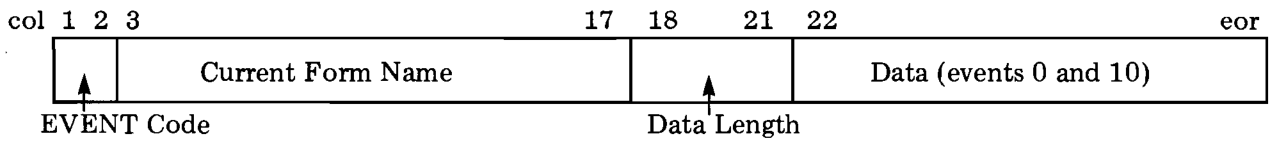
All records used for the RPG interface to V/3000 must be variable length ("V" in column 19 of the File Description Specification for WORKSTN.) Input records contain the EVENT codes plus forms information. Output records contain the ACTION codes or mnemonics, plus other information that depends on the particular action.

INPUT RECORD FORMATS

Depending on the EVENT code, input records have one of four possible formats. All input records contain the EVENT code in the first two locations, and the current form name in the 15 characters immediately following.

EVENTS 00-08, AND 10

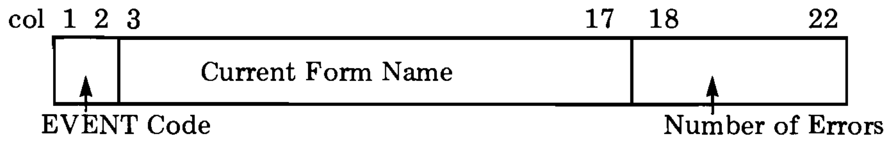
uses the input record format:



The data starting in location 22 is separated into fields according to the definition of the current form. The Data length is the total number of characters required by all the fields in the current form. Only events 0 and 10 actually have data.

EVENT 09

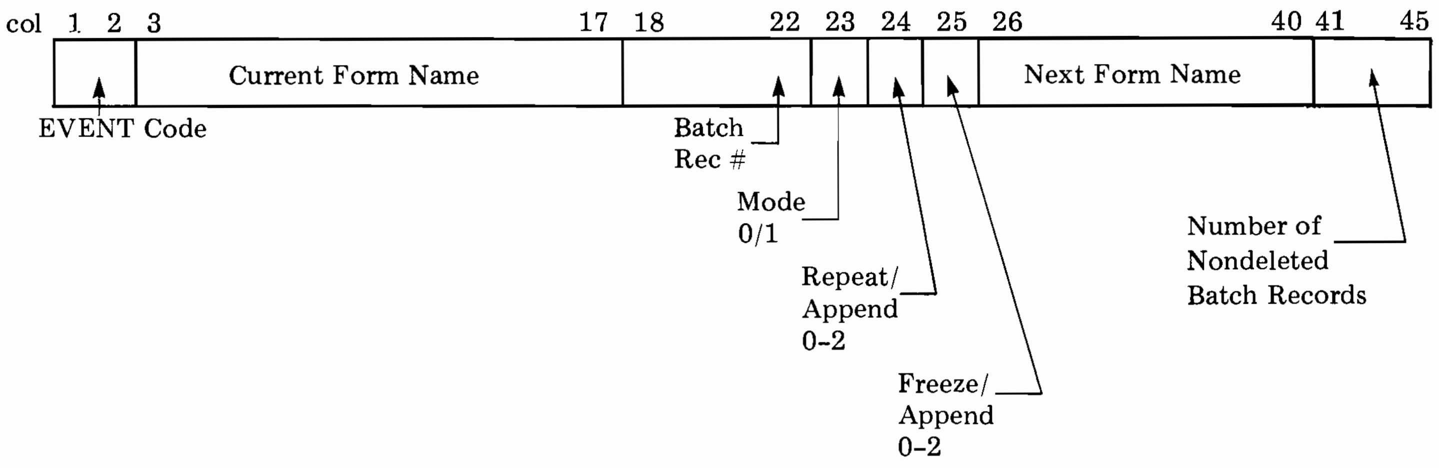
uses the input record format:



The number of errors specifies the number of fields in which errors were detected, either by the FORMSPEC edit checks or by user program editing.

EVENT 11

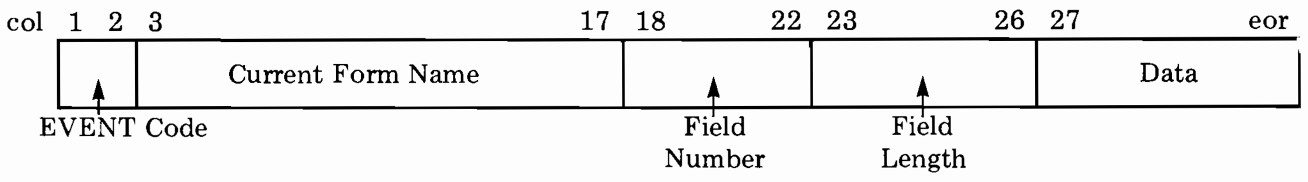
uses the input record format:



Event 11 is a general status event which is available after any action that does not return another event. Event 11 is also available at the beginning of execution when it returns the number of the next batch record (i.e. End-of-batch +1). This is particularly useful in determining the batch file upper bounds when an existing batch file is used.

EVENT 12

uses the input record format:



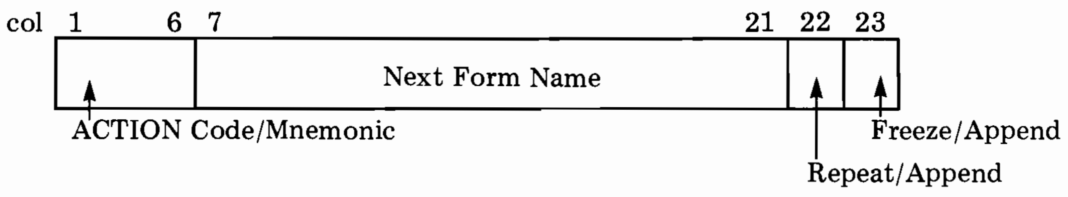
The field number is a unique identifier for the field being retrieved; it does not change if the position of the field in the form changes. The data in the specified field is returned in this record.

OUTPUT RECORD FORMATS

Output records are in one of five formats depending on the ACTION code or mnemonic. All output records contain an ACTION code in locations 1 and 2, or an ACTION mnemonic in locations 1 through 6.

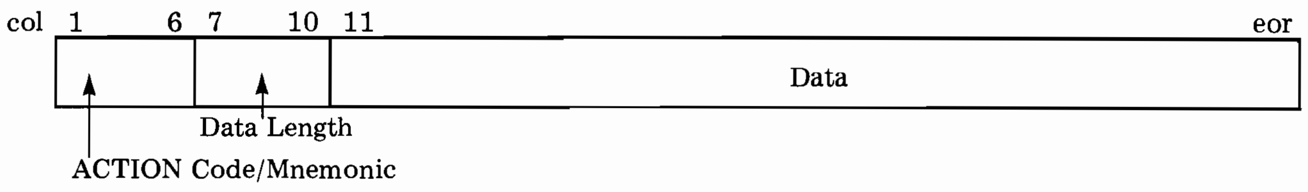
ACTION 50 (CHGNXT)

uses the output record format:



ACTIONS 57 (SHODTA) AND 63 (PUTDTA)

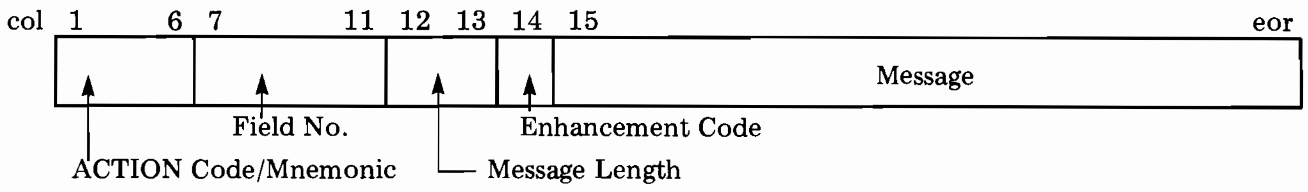
use the output record format:



The data starting in location 11 is separated into fields according to the field definition of the current form. The data length is the total number of characters for the data in this form.

ACTIONS 56 (CORERR) AND 62 (BADFLD)

use the output record format:

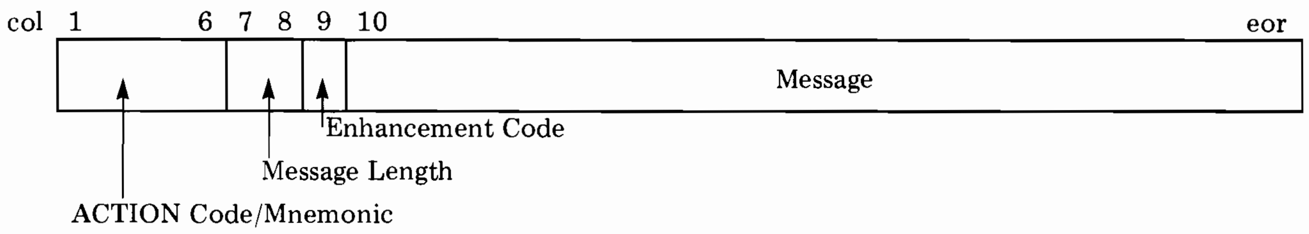


The field number identifies the field in which an error was detected so that this field can be enhanced at the terminal. The message length specifies the number of characters in the message to be sent to the window for subsequent display. For best results, the message should have no more than 79 displayable characters. The enhancement code may be one of the following:

@,A-0	Specify particular enhancement (refer to WINDOWENH description, section VI)
zero	No enhancement
blank	Do not change existing enhancement

ACTIONS 52 (PUTMSG) AND 55 (SHOMSG)

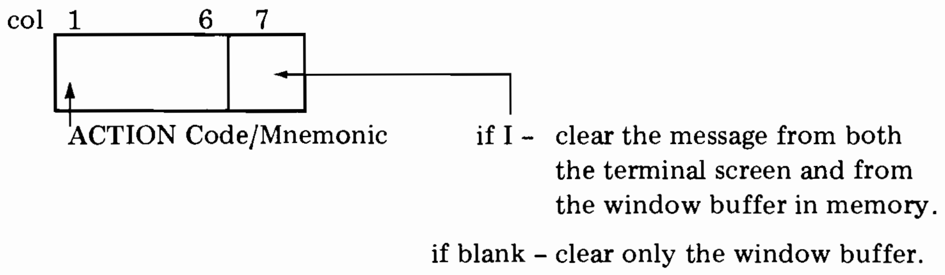
use the output record format:



The message length specifies the number of characters in the message, and the enhancement code determines the message enhancement (see list of codes above for actions 56 and 62).

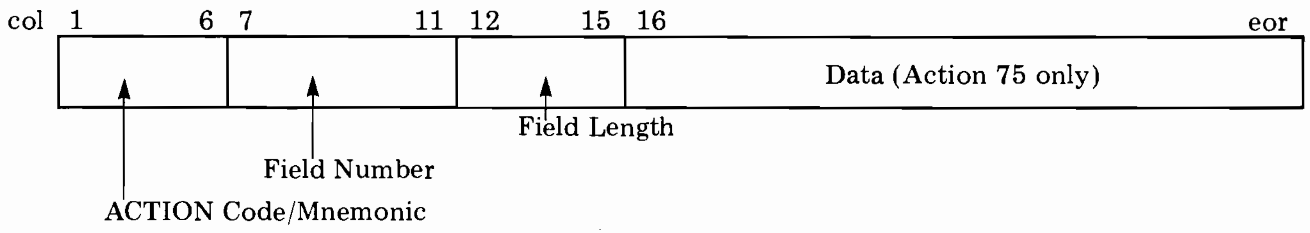
ACTION 73 (CLRMSG)

use the output record format:


col 1 6 7
ACTION Code/Mnemonic
if I - clear the message from both the terminal screen and from the window buffer in memory.
if blank - clear only the window buffer.

ACTIONS 74 (GETFLD) AND 75 (PUTFLD)

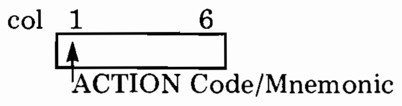
use the output record format:



This record contains the field number of the selected field in the V/3000 data buffer, the number of characters in the field, and for PUTFLD, the data to be written to the V/3000 buffer. Note that each field in a form is assigned a number by FORMSPEC that does not change regardless of any changes to the position of the field in the form or to its description.

ACTIONS 51 (GETNXT), 53 (SHOW), 54 (RDTERM), 58-61 (INIT,EDITS,PRINT,NUMERR) AND 64-72 (GETDTA,FINISH,WRTBAT,PREV,REREAD,NEXT, RESUME,DELETE,RDBTNU)

use the output record format:



Note that this output record contains only the ACTION code. It is used by all actions that do not use the five preceding output formats.

HANDLING RUNTIME ERRORS

RPG allows the programmer to specify one of six error response options. The specified option determines what RPG does in case an error occurs during program execution. The selected option is entered on a Control Record specification (H spec). When an error occurs for which no option is specified, the system asks the terminal operator to specify an option.

In a data entry operation with a form displayed on the terminal screen and the terminal operating in block mode, the request cannot take the usual form. Instead, the RPG interface displays a message describing the error in the screen message window, and waits for the terminal operator to enter an error response by pressing a function key.

Function keys f1 through f6 are used for this purpose. The terminal operator should press one of the following keys to select an error response:

f1	Continue execution
f2	Skip the input record containing the error and continue execution.
f3	Terminate the program by executing the normal termination code.
f4	Terminate the program immediately.
f5	Terminate normally, and print an error dump.
f6	Terminate immediately, and print an error dump.

ERROR DUMP

The RPG error dump is normally written to the standard list device, ($STDLIST), which in a session, is the terminal. Since an error dump to a terminal is of limited use, particularly in a data entry environment, the dump can be redirected to a file. The name of this file is specified in columns 7-14 of the Control Record Specification (H spec). If omitted, $STDLIST is used and the dump is sent to the terminal.

ERROR MESSAGE DISPLAY

Some error and warning messages are displayed on the terminal screen. Each message is displayed for three seconds unless this interval is changed by entering a digit between 0 and 9 in column 51 of the File Description specification. For example, to increase the time during which the message appears on the screen to 5 seconds, enter "5" in column 51 of the File Description spec. If the interval is set to zero, messages are not displayed at the terminal.

BREAK KEY

The BREAK key is placed close to the ENTER key on some terminals. As a result, it is easy for the terminal operator to press BREAK rather than ENTER. To avoid this possibility, the BREAK key is disabled when a WORKSTN file is in use. You can override this default setting and enable the BREAK key by entering a "B" in column 52 of the File Description specification (F spec) for the WORKSTN file.

If BREAK is enabled, the terminal operator should be told how to recover from an accidental break by typing RESUME. If the operator was entering data, he or she should then press function key 4 (REFRESH under ENTRY program control) to clear previously typed data from the form. The operator can then retype the data in the form and press ENTER to enter the data.

TRACE FILE

The trace file is a tool to help find program errors. This file is specified as a continuation line on the File Description Specification. If a trace file is specified, every action and event causes at least one record to be written to the file. Then, if the program does not operate as expected, you can examine the trace file to isolate the error. If a runtime error occurs, the message describing the error is written to the trace file as well as displayed at the terminal. If the trace file becomes full, tracing stops, but program execution continues.

EXAMPLE USING RPG INTERFACE

The following simple example uses the RPG interface to control data entry through forms design by the FORMSPEC program of V/3000. (For a complete example using all the capabilities of V/3000 to control data entry, refer to the RPG program in appendix A.)

Assume the program is to perform the following actions:

- Display an initialized form.
- Read data entered at the terminal.
- Perform V/3000 and user edits.
- Write the edited data to the batch file.

The first step is to use the V/3000 FORMSPEC program to define the forms and the editing associated with them. The next step is to write the RPG program. A general plan of the program is shown below. Note that all output is exception output, and all reads are demand reads.

Note that in this example, the function keys f1 through f7 are treated as if they were f8 (EXIT). Thus, if any function key is pressed, the program terminates.

PROGRAM PLAN

*1. Get the next form from the forms file using ACTION 51 (GETNXT). (If the form is a repeating form, the next form is not actually fetched.)

2. Move any initial values to this form using ACTION 58 (INIT).

*3. Display the form with the initial values and message window contents using ACTION 53 (SHOW).

4. Indicate that data entered on form is to be read with ACTION 54 (RDTERM).

5. Read entered data from the WORKSTN file (demand read). The data is input as an EVENT of one of the types 00 through 08:

   If EVENT is type 01 through 08 (EXIT), terminate.
   If EVENT is type 00 (ENTER), continue.

   (For simplicity in this example, events 01 through 07 are treated as if they were event 08.)

6. Perform all V/3000 edits specified for the fields on the form using ACTION 59 (EDITS).

7. Read the WORKSTN file. The record returned is an EVENT type 09 and specifies the number of errors found in the entered data.

*If errors are found, a message describing the first error found is moved to the window buffer, and all the fields with errors are enhanced. Return to step 3 to display the form with the error message and enhancements. Continue through steps 3-7 until no errors are found.

If no errors are found, continue.

8. Transfer the data from the V/3000 data buffer to the user program with ACTION 64 (GETDTA).

9. Read the WORKSTN file. The record returned is an EVENT type 10, containing the data from the V/3000 buffer.

*10. Perform any editing specified in the user program. If errors are found, do step 11; otherwise, go on to step 12.

11. Use ACTION 56 (CORERR) to indicate which fields failed the user edit, display the form with fields in error enhanced, and if the program buffer contains a message for the window, display this message with the form. Follow ACTION 56 by a demand read. The record read should be an EVENT type 00 indicating ENTER was pressed and new data has been entered.

*Repeat steps 10 and 11 until all data has passed the user edits.

12. Transfer the edited data to the V/3000 buffer using ACTION 63 (PUTDTA).

13. Write the data in the V/3000 buffer to the batch file with ACTION 66 (WRTBAT).

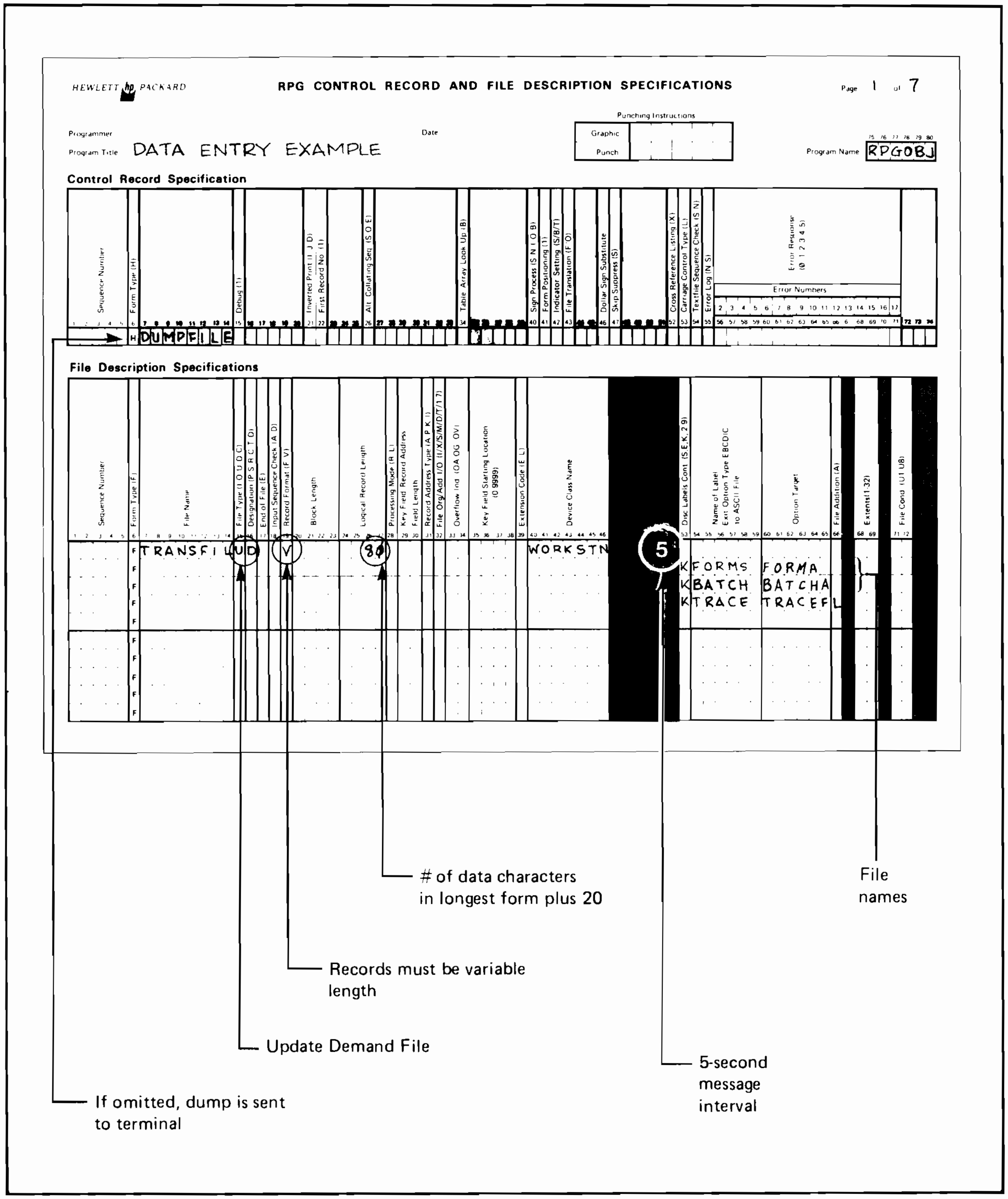
14. If the next form name or the repeat/append or repeat/freeze status is to be changed, use ACTION 50 (CHGNXT) to make these changes.

*Return to step 1.

During execution of this program, all data entered by an operator at the terminal into forms designed through FORMSPEC will be written to the batch file, one record per form. You can use the reformatter (described in section V) to insure that the data is in the exact form needed by your application.

Before running the program, all files must be described. These files include the file allocated to the device WORKSTN, the forms file on which the forms are stored, the batch file to which the data is written, a trace file to trace the actions and events, and a dump file to which a program dump is sent in case of runtime errors.

These files are entered on the Control Record and File Description Specification sheets. (Refer to figure 7-2).



HEWLETT hp PACKARD — RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS — Page 1 of 7

Programmer — Date — Punching Instructions: Graphic / Punch

Program Title: DATA ENTRY EXAMPLE — Program Name: RPGOBJ

Control Record Specification

Form Type (H)	Debug / File name
H	DUMPFILE

File Description Specifications

Form Type (F)	File Name	File Type (I O U D C)	Designation (P S R C T D)	Record Format (F V)	Logical Record Length	Device Class Name		Disc Labels Cont (S E K 29)	Name of Label Exit Option Type EBCDIC to ASCII File	Option Target	File Addition (A)
F	TRANSFI	U	D	V	80	WORKSTN	5				
F								K	FORMS	FORMA	
F								K	BATCH	BATCHA	
F								K	TRACE	TRACEF	L

# of data characters in longest form plus 20
File names
Records must be variable length
Update Demand File
5-second message interval
If omitted, dump is sent to terminal

Figure 7-2. File Description Specification for Sample Program

CONTROL RECORD SPECIFICATION

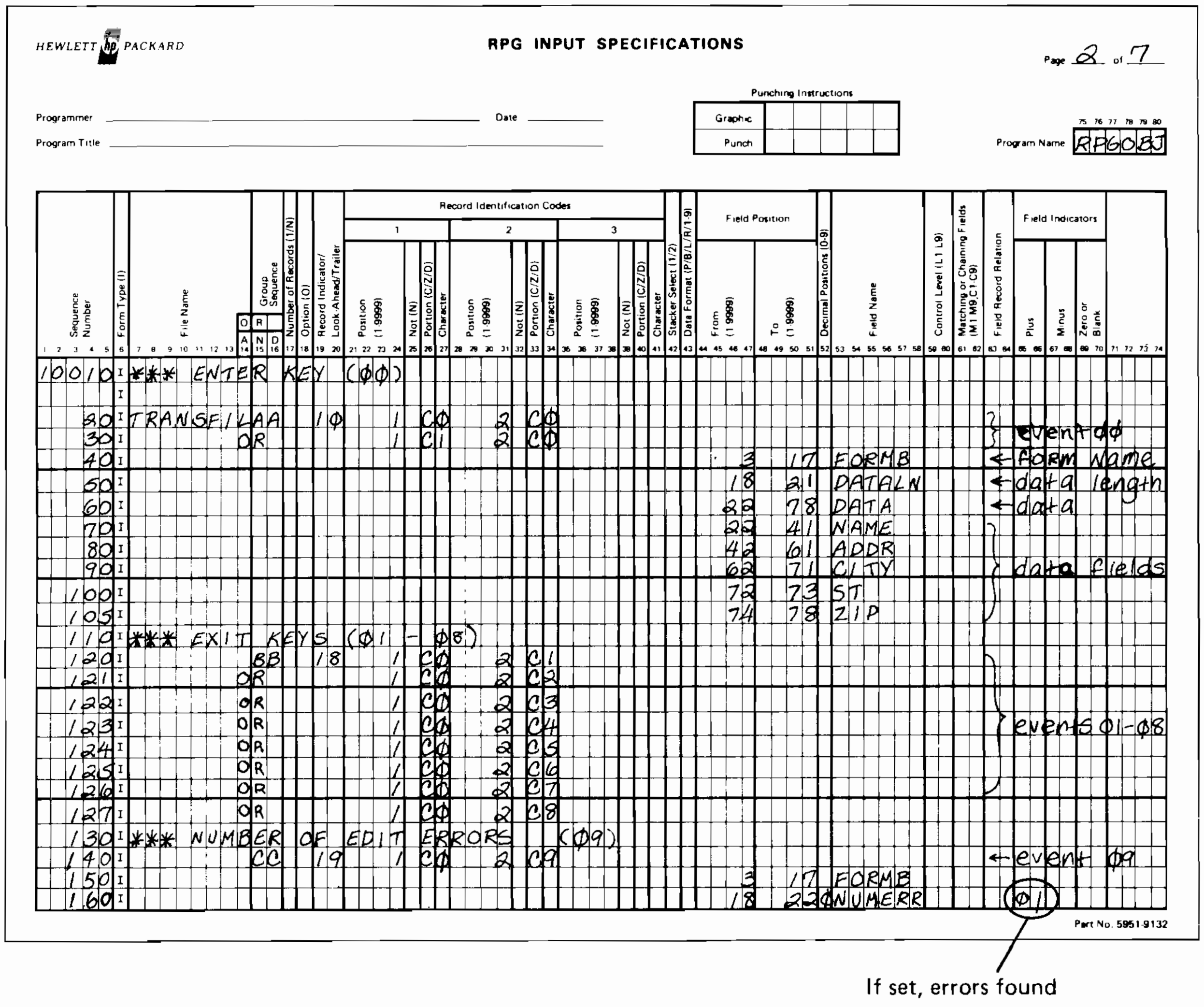
Columns 7 - 14	"DUMPFILE" - Name of file to which dump is sent in case of a runtime error. (If omitted, dump is directed to the terminal.)

FILE DESCRIPTION SPECIFICATION

Columns 7 - 14	"TRANSFIL" - Name of RPG file assigned to device class WORKSTN.
Column 15	"U" — Indicates file is type update. No other type is allowed for interface with V/3000.
Column 16	"D" - Indicates file is demand. For this application, file must be demand file; for other applications that use a simple RPG cycle, P for primary may be specified.
Column 19	"V" - Indicates record length is variable. All records in WORKSTN file must be variable length.
Columns 24 - 27	Record length is dependent on the data buffer length needed to hold all the data from the longest form, plus 20 characters for control information. (Assume the longest form for this application requires 60 characters of data, so the record length is specified as 80.)
Columns 40 - 46	"WORKSTN" — Device class name must be WORKSTN for the V/3000 interface.
Column 51	"0" through "9" indicates number of seconds during which message is displayed at terminal (default = 3). If zero, message is suppressed.
Column 52	"B" - Enables BREAK key; leaving this column blank or entering any other value leaves the BREAK key disabled.
Column 53	"K" - Indicates file description continuation, one for each additional file.
Columns 54 - 59	"FORMS" - Specifies that a forms file is used. This file is required.
	"BATCH" - Specifies that a batch file is used. If this keyword is not specified, no batch file is used.
	"TRACE" - Specifies that a trace file is used. If this keyword is not specified, no trace file is used.
Columns 60 - 74	Name of file whose use is indicated in columns 54 - 59; specified as an MPE file name. If more than 15 characters are needed for the name, a :FILE command should be used.

INPUT SPECIFICATIONS

The input specifications in figure 7-3 are for events 00 (ENTER key), 01 through 08 (function keys f1 - f8), and 09 (number of fields with errors). Note that events 01 through 07 are treated by this program as if they were event 08. No data is read by events 01 through 08; in this program, they signal that a key has been pressed at the terminal, and are treated as EXIT.



HEWLETT PACKARD — RPG INPUT SPECIFICATIONS — Page 2 of 7

Programmer ______ Date ______

Program Title ______

Punching Instructions: Graphic / Punch

Program Name RPG0BJ

Sequence Number	Form Type	File Name	AND/OR	Group Sequence	Record Indicator	Position (1)	Portion C/Z/D (1)	Character (1)	Position (2)	Portion C/Z/D (2)	Character (2)	From	To	Decimal Positions	Field Name	Field Indicators Plus	
10010	I	*** ENTER KEY (ØØ)															
	I																
20	I	TRANSFIL		AA	1Ø	1	C	Ø	2	C	Ø						} event ØØ
30	I		OR			1	C	1	2	C	Ø						
40	I											3	17		FORMB		← form name
50	I											18	21		DATALN		← data length
60	I											22	78		DATA		← data
70	I											22	41		NAME		
80	I											42	61		ADDR		
90	I											62	71		CITY		data fields
100	I											72	73		ST		
105	I											74	78		ZIP		
110	I	*** EXIT KEYS (Ø1 - Ø8)															
120	I			BB	18	1	C	Ø	2	C	1						
121	I		OR			1	C	Ø	2	C	2						
122	I		OR			1	C	Ø	2	C	3						
123	I		OR			1	C	Ø	2	C	4						events Ø1-Ø8
124	I		OR			1	C	Ø	2	C	5						
125	I		OR			1	C	Ø	2	C	6						
126	I		OR			1	C	Ø	2	C	7						
127	I		OR			1	C	Ø	2	C	8						
130	I	*** NUMBER OF EDIT ERRORS (Ø9)															
140	I			CC	19	1	C	Ø	2	C	9						← event Ø9
150	I											3	17		FORMB		
160	I											18	22	Ø	NUMERR	Ø1	

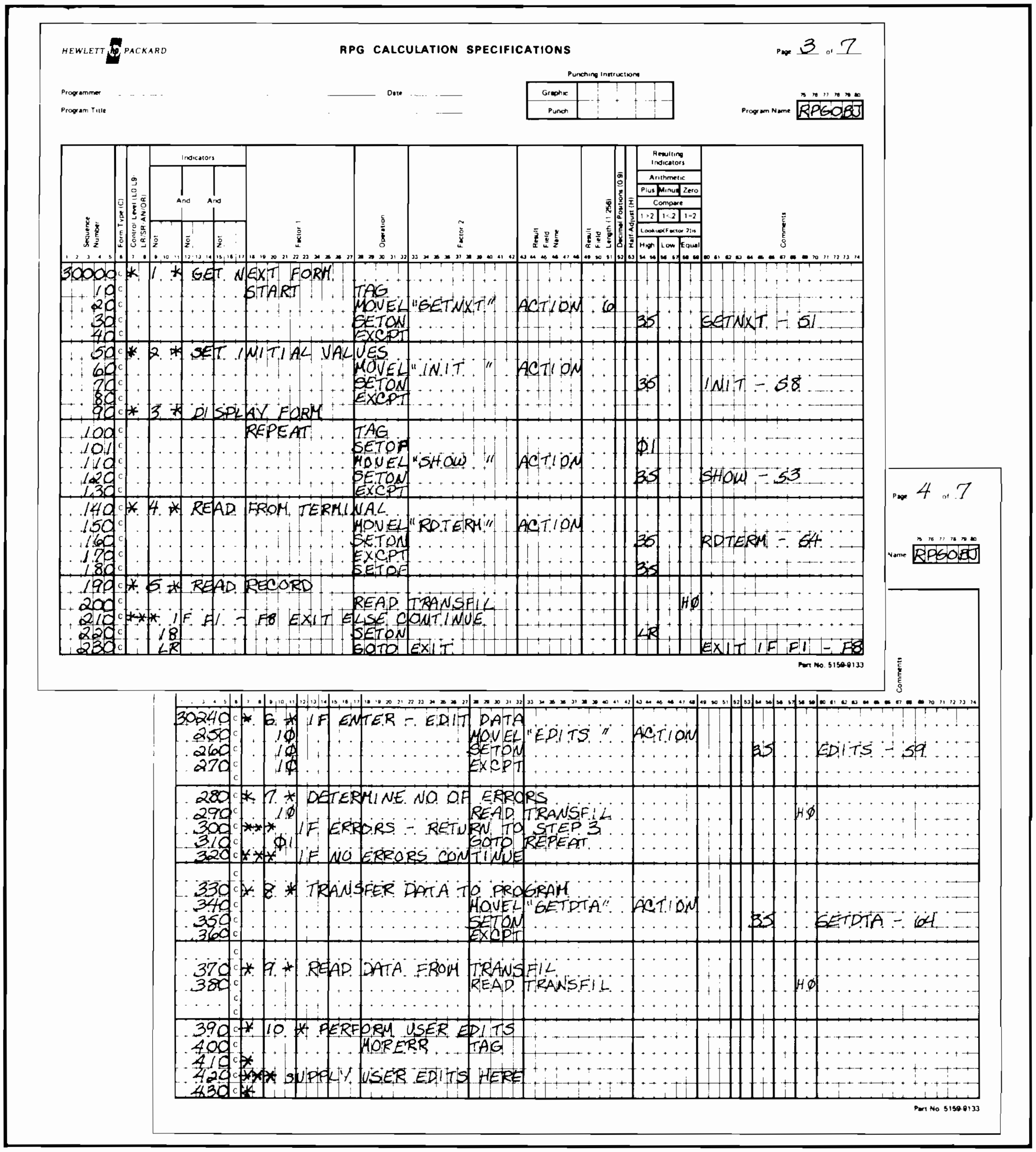
Part No. 5951-9132

If set, errors found

Figure 7-3. Input Specifications for Sample Program

CALCULATION SPECIFICATIONS

Since the file TRANSFIL assigned to device file WORKSTN is specified as an update demand file, the program control remains in the calculation specifications. The program terminates when the f8 key (event 08) is pressed. Refer to figure 7-4 for the calculation specifications used by the program. Note that each numbered step corresponds to the steps described above under the heading "Program Plan".


HEWLETT PACKARD RPG CALCULATION SPECIFICATIONS Page 3 of 7

Programmer Date Punching Instructions Graphic Punch

Program Title Program Name RP60BJ

```
30000C* 1 * GET NEXT FORM
   10C           START     TAG
   20C                     MOVEL"GETNXT" ACTION  6
   30C                     SETON                     35         GETNXT - 51
   40C                     EXCPT
   50C* 2 * SET INITIAL VALUES
   60C                     MOVEL"INIT  " ACTION
   70C                     SETON                     35         INIT - 58
   80C                     EXCPT
   90C* 3 * DISPLAY FORM
  100C           REPEAT    TAG
  101C                     SETOF                     Ø1
  110C                     MOVEL"SHOW  " ACTION
  120C                     SETON                     35         SHOW - 53
  130C                     EXCPT
  140C* 4 * READ FROM TERMINAL
  150C                     MOVEL"RDTERM" ACTION
  160C                     SETON                     35         RDTERM - 64
  170C                     EXCPT
  180C                     SETOF                     35
  190C* 5 * READ RECORD
  200C                     READ TRANSFIL                 HØ
  210C*** IF F1 - F8 EXIT ELSE CONTINUE
  220C  18                 SETON                     LR
  230C  LR                 GOTO EXIT                            EXIT IF F1 - F8
```

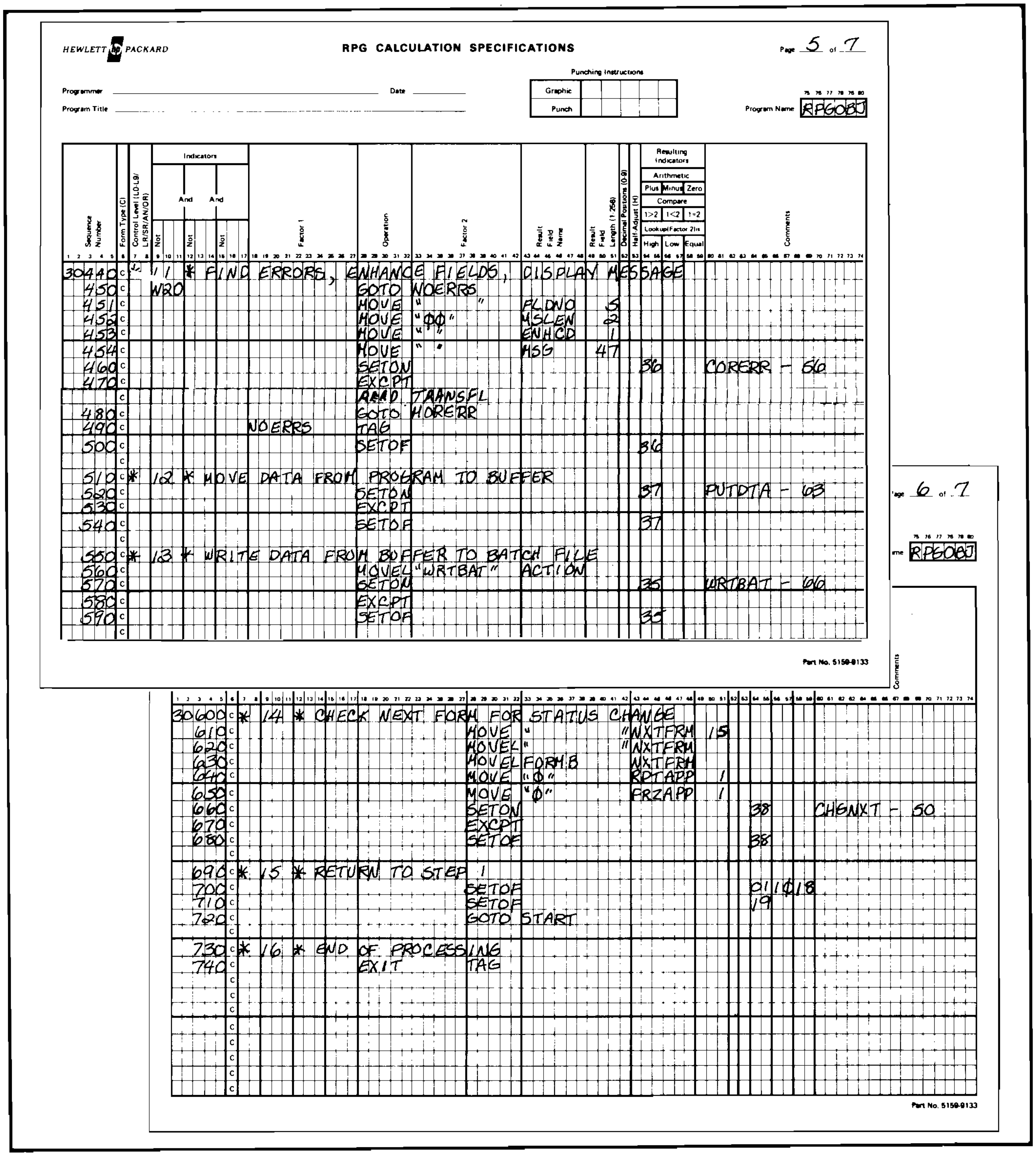
Part No. 5159-9133

Page 4 of 7 Name RP60BJ

```
30240C* 6 * IF ENTER - EDIT DATA
  250C  1Ø                 MOVEL"EDITS " ACTION
  260C  1Ø                 SETON                     35         EDITS - 59
  270C  1Ø                 EXCPT
     C
  280C* 7 * DETERMINE NO OF ERRORS
  290C  1Ø                 READ TRANSFIL                 HØ
  300C***     IF ERRORS - RETURN TO STEP 3
  310C  Ø1                 GOTO REPEAT
  320C***     IF NO ERRORS CONTINUE
     C
  330C* 8 * TRANSFER DATA TO PROGRAM
  340C                     MOVEL"GETDTA" ACTION
  350C                     SETON                     35         GETDTA - 64
  360C                     EXCPT
     C
  370C* 9 * READ DATA FROM TRANSFIL
  380C                     READ TRANSFIL                 HØ
     C
     C
  390C* 10 * PERFORM USER EDITS
  400C           MORERR    TAG
  410C*
  420C*** SUPPLY USER EDITS HERE
  430C*
```

Part No. 5159-9133

Figure 7-4. Calculation Specifications for Sample Program



HEWLETT PACKARD — RPG CALCULATION SPECIFICATIONS — Page 5 of 7

Programmer ____ Date ____ Program Title ____ Punching Instructions: Graphic / Punch — Program Name RPGOBJ

```
  30440C* 11 * FIND ERRORS, ENHANCE FIELDS, DISPLAY MESSAGE
    450C  W20                GOTO NOERRS
    451C                     MOVE "     "   FLDNO   5
    452C                     MOVE "ØØ"      MSLEN   2
    453C                     MOVE " "       ENHCD   1
    454C                     MOVE " "       MSG    47
    460C                     SETON                     36    CORERR - 56
    470C                     EXCPT
       C                     READ TRANSFL
    480C                     GOTO MORERR
    490C           NOERRS    TAG
    500C                     SETOF                     36
       C
    510C* 12 * MOVE DATA FROM PROGRAM TO BUFFER
    520C                     SETON                     37    PUTDTA - 63
    530C                     EXCPT
    540C                     SETOF                     37
       C
    550C* 13 * WRITE DATA FROM BUFFER TO BATCH FILE
    560C                     MOVEL"WRTBAT"  ACTION
    570C                     SETON                     35    WRTBAT - 66
    580C                     EXCPT
    590C                     SETOF                     35
       C
```

Part No. 5150-9133

Page 6 of 7 — Program Name RPGOBJ

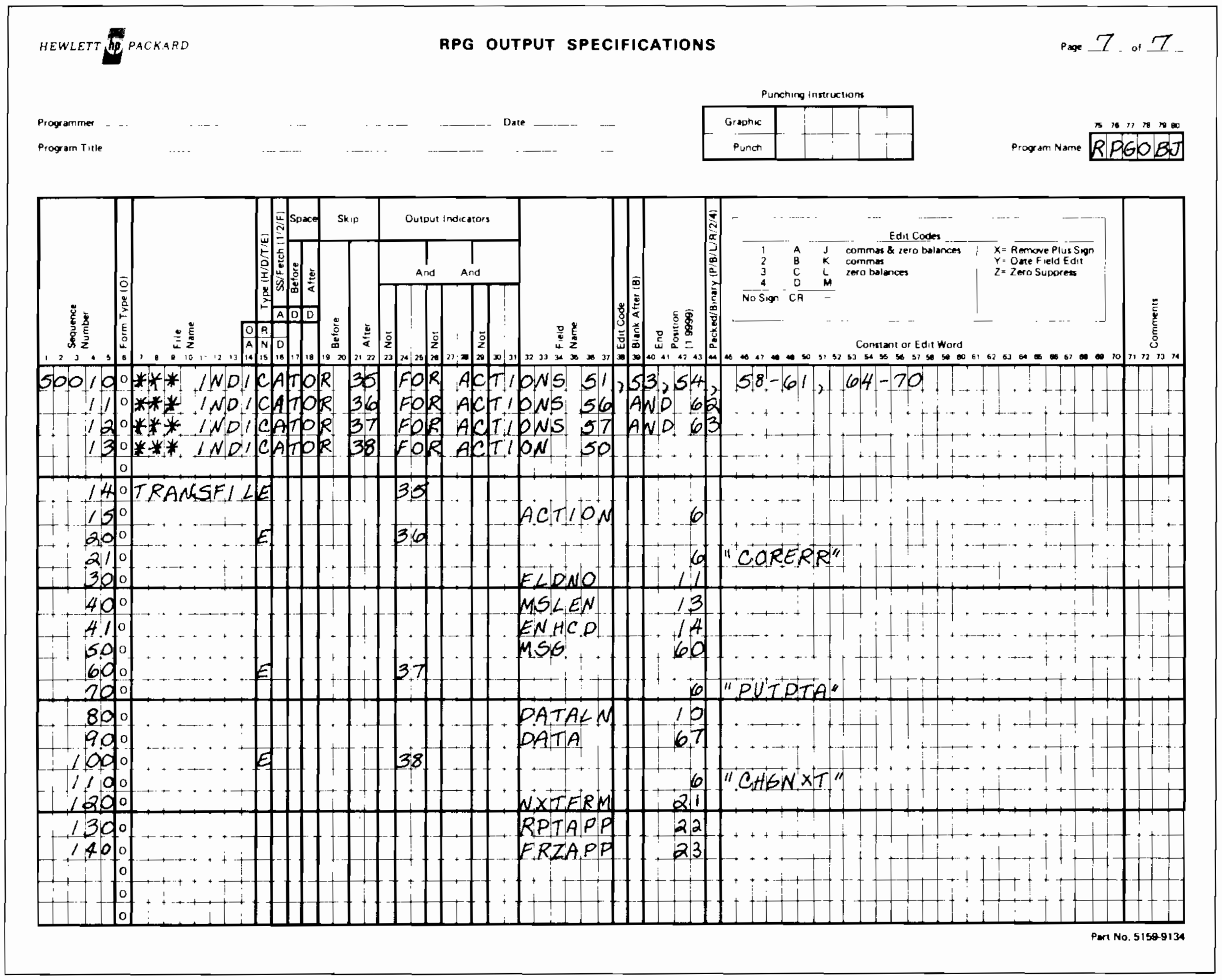
```
  30600C* 14 * CHECK NEXT FORM FOR STATUS CHANGE
    610C                     MOVE "        "NXTFRM 15
    620C                     MOVEL"        "NXTFRM
    630C                     MOVELFORMB     NXTFRM
    640C                     MOVE "Ø"       RPTAPP  1
    650C                     MOVE "Ø"       FRZAPP  1
    660C                     SETON                     38    CHGNXT - 50
    670C                     EXCPT
    680C                     SETOF                     38
       C
    690C* 15 * RETURN TO STEP 1
    700C                     SETOF                     011Ø18
    710C                     SETOF                     19
    720C                     GOTO START
       C
    730C* 16 * END OF PROCESSING
    740C           EXIT      TAG
       C
       C
       C
       C
       C
       C
       C
```

Part No. 5150-9133

Figure 7-4. Calculation Specifications for Sample Program (Cont.)

OUTPUT SPECIFICATONS

Four types of output record are established for this program. They are illustrated in figure 7-5. Refer to "Output Record Formats" in the preceding text for a description of the formats assigned to each action code.



HEWLETT PACKARD RPG OUTPUT SPECIFICATIONS Page 7 of 7

Programmer Date

Program Title

Punching Instructions: Graphic / Punch

Program Name RPGOBJ

Edit Codes:

1	A	J	commas & zero balances	X= Remove Plus Sign
2	B	K	commas	Y= Date Field Edit
3	C	L	zero balances	Z= Zero Suppress
4	D	M		
No Sign	CR	–		

```
50010O*** INDICATOR 35 FOR ACTIONS 51,53,54, 58-61, 64-70
   11O*** INDICATOR 36 FOR ACTIONS 56 AND 62
   12O*** INDICATOR 37 FOR ACTIONS 57 AND 63
   13O*** INDICATOR 38 FOR ACTION  50
     O
   14OTRANSFILE       35
   15O                        ACTION      6
   20O       E        36
   21O                                    6 "CORERR"
   30O                        FLDNO      11
   40O                        MSLEN      13
   41O                        ENHCD      14
   50O                        MSG        60
   60O       E        37
   70O                                    6 "PUTDTA"
   80O                        DATALN     10
   90O                        DATA       67
  100O       E        38
  110O                                    6 "CHGNXT"
  120O                        NXTFRM     21
  130O                        RPTAPP     22
  140O                        FRZAPP     23
     O
     O
     O
```

Part No. 5159-9134

Figure 7-5. Output Specifications for Sample Program

COMPILED VERSION

The following pages contain a listing of the compiled version of the sample program illustrated in figures 7-2 through 7-5.

```
$CONTROL LIST,SOURCE,NOWARN,USLINIT,QUOTE="
H

FTRANSFILUD  V      80            WORKSTN
F                                              KFORMS NEWFORMS
F                                              KBATCH BATCHF
F                                              KTRACE TRACEF
I*** PROCESS ENTER KEY (00)

ITRANSFILAA  10    1 C0    2 C0
I       OR         1 C1    2 C0
I                                        3  17 FORMB
I                                       18  21 DATALN
I                                       22  78 DATA
I                                       22  41 NAME
I                                       42  61 ADDR
I                                       62  71 CITY
I                                       72  73 STATE
I                                       74  78 ZIP
I*** EXIT KEYS (01 - 08)
I        66   18    1 C0    2 C1
I       OR          1 C0    2 C2
I       OR          1 C0    2 C3
I       OR          1 C0    2 C4
I       OR          1 C0    2 C5
I       OR          1 C0    2 C6
I       OR          1 C0    2 C7
I       OR          1 C0    2 C8
I*** NUMBER OF EDIT ERRORS (09)
I        CC   19    1 C0    2 C9
I                                        3  17 FORMB
I                                       18  220NUMERR      01
C* 1 *  GET NEXT FORM

C           START     TAG
C                     MOVEL"GETNXT"  ACTION  6
C                     SETON                    35    GETNXT - 51
C                     EXCPT
C* 2 *  SET INITIAL VALUES
C                     MOVEL"INIT  "  ACTION
C                     SETON                    35    INIT - 58
C                     EXCPT
C* 3 *  DISPLAY FORM
C           REPEAT    TAG
C                     SETOF                    01
C                     MOVEL"SHOW  "  ACTION
C                     SETON                    35    SHOW - 53
C                     EXCPT
C* 4 *  READ FROM TERMINAL
C                     MOVEL"RDTERM"  ACTION
C                     SETON                    35    RDTERM - 54
```

```
C                           EXCPT
C                           SETOF                   35
C* 5 *  READ RECORD
C                           READ TRANSFIL                H0
C*** IF F1 - F8 THEN EXIT ELSE CONTINUE
C   18                      SETON                   LR
C   LR                      GOTO EXIT                      EXIT IF F1
C* 6 *  IF ENTER, EDIT DATA
C   10                      MOVEL"EDITS "   ACTION
C   10                      SETON                   35     EDITS - 59
C   10                      EXCPT
C* 7 *  DETERMINE NUMBER OF EDIT ERRORS
C   10                      READ TRANSFIL                H0
C*** IF ERRORS, RETURN TO STEP 3
C   01                      GOTO REPEAT
C*** IF NO ERRORS - CONTINUE
C* 8 *  TRANSFER DATA TO PROGRAM
C                           MOVEL"GETDTA"   ACTION
C                           SETON                   35     GETDTA - 64
C                           EXCPT
C* 9 *  READ DATA FROM TRANSFIL
C                           READ TRANSFIL                H0
C* 10 *  PERFORM USER EDITS
C               MORERR      TAG
C*
C*** SUPPLY EDIT ROUTINES HERE ***
C*
C* 11 *  FIND ERRORS, ENHANCE FIELDS, DISPLAY MESSAGE
C  N20                      GOTO NOERRS
C                           MOVE "       "  FLDNO    5
C                           MOVE "00"       MSLEN    2
C                           MOVE " "        ENHCD    1
C                           MOVE " "        MSG     47
C                           SETON                   36     CORERR - 56
C                           EXCPT
                            READ TRANSFIL
C                           GOTO MORERR
C               NOERRS      TAG
C                           SETOF                   36
C* 12 *  MOVE DATA FROM PROGRAM TO BUFFER
C                           SETON                   37     PUTDTA - 63
C                           EXCPT
C                           SETOF                   37
C* 13 *  WRITE DATA FROM BUFFER TO BATCH FILE
C                           MOVEL"WRTBAT"   ACTION
C                           SETON                   35     WRTBAT - 66
C                           EXCPT
C                           SETOF                   35
C* 14 *  CHECK NEXT FORM FOR STATUS CHANGE
C                           MOVE "         "NXTFRM 15
C                           MOVEL"         " NXTFRM
C                           MOVELFORMB      NXTFRM
C                           MOVE "0"        RPTAPP   1
C                           MOVE "0"        FRZAPP   1
C                           SETON                   38     CHGNXT - 50
C                           EXCPT
C                           SETOF                   38
C* 15 *  RETURN TO STEP 1
```

```
C                          SETOF                         011018
C                          SETOF                         19
C                          GOTO START
C* 16 *  END OF PROCESSING
C              EXIT        TAG
O*** IND 35 FOR ACTIONS 51,53,54,58-61,64-70



OTRANSFILE              35
O                               ACTION      6
O          E            36
O                                           6  "CORERR"
O                               FLDNO      11
O                               MSLEN      13
O                               ENHCD      14
O                               MSG        60
O          E            37
O                                           6  "PUTDTA"
O                               DATALN     10
O                               DATA       67
O          E            38
O                                           6  "CHGNXT"
O                               NXTFRM     21
O                               RPTAPP     22
O                               FRZAPP     23
```

ENTRY PROGRAM

This appendix contains a listing of the ENTRY program in each of the following languages:

The programs listed in this appendix have been updated and reprinted for update package No. 1 and are tested examples in RPG, COBOL, FORTRAN, BASIC, and SPL to aid programmers who want to write their own special version. They perform exactly the same functions as the ENTRY program supplied with V/3000.

```
**************************************************************
*                                                            *
*            COBOL--ENTRY--V/3000 Data Entry Program         *
*                                                            *
*                          9/1/79                            *
*                                                            *
**************************************************************
*
* This program controls source data entry for any forms file.
* It opens a forms file, based on user input; it opens a batch
* file, also named by the user.  If all is ok, it displays the
* headform, accepts input, edits the data, and if no errors,
* writes it to the batch file.  The program continues to do this
* until $END is reached, or until the EXIT function key has been
* pressed.
*
* This program also controls browsing through the data collected,
* This supports modification of that data.
*
* The function keys have defined meanings as follows:
*
*     f1         f2         f3         f4
*    HEAD      DELETE     PRINT      REFRESH
*
*     f5         f6         f7         f8
*    PREV       NEXT      BROWSE/     EXIT
*                         COLLECT
*
*
********************************************************

 IDENTIFICATION DIVISION.
 PROGRAM-ID. COBOLENTRY.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. HP3000III.
 OBJECT-COMPUTER. HP3000III.
 INPUT-OUTPUT SECTION.
```

```
$PAGE "                  DECLARATIONS"
*********************************************************************
*                                                                   *
*                    GLOBAL DECLARATIONS                            *
*                                                                   *
*********************************************************************
 DATA DIVISION.

 FILE SECTION.

 WORKING-STORAGE SECTION.

 01 COMAREA.
    05 COM-STATUS             PIC S9(4) COMP VALUE ZERO.
    05 COM-LANGUAGE           PIC S9(4) COMP VALUE ZERO.
    05 COM-COMAREALEN         PIC S9(4) COMP VALUE 60.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 COM-MODE               PIC S9(4) COMP VALUE ZERO.
    05 COM-LASTKEY            PIC S9(4) COMP VALUE ZERO.
    05 COM-NUMERRS            PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 COM-CFNAME             PIC X(15) VALUE SPACES.
    05 FILLER                 PIC X VALUE SPACES.
    05 COM-NFNAME             PIC X(15) VALUE SPACES.
    05 FILLER                 PIC X VALUE SPACES.
    05 COM-REPEATOPT          PIC S9(4) COMP VALUE ZERO.
    05 COM-NFOPT              PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 COM-DBUFLEN            PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 COM-DELETEFLAG         PIC S9(4) COMP VALUE ZERO.
    05 COM-SHOWCONTROL        PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 COM-NUMRECS            PIC S9(6) COMP VALUE ZERO.
    05 COM-RECNUM             PIC S9(6) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 COM-TERMFILENUM        PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER                 PIC S9(4) COMP VALUE ZERO.
    05 FILLER               PIC S9(4) COMP VALUE ZERO.
    05 FILLER               PIC S9(4) COMP VALUE ZERO.
    05 FILLER               PIC S9(4) COMP VALUE ZERO.
    05 FILLER               PIC S9(4) COMP VALUE ZERO.
    05 FILLER               PIC S9(4) COMP VALUE ZERO.
 01 MISC-VALUES.
    05 COMAREALEN            PIC S9(4) COMP VALUE 60.
    05 COBOL-LANG            PIC S9(4) COMP VALUE 0.
    05 COLLECT-MODE          PIC S9(4) COMP VALUE 0.
    05 BROWSE-MODE           PIC S9(4) COMP VALUE 1.
    05 MAXWINDOWLEN          PIC S9(4) COMP VALUE 150.
    05 NAMELEN               PIC S9(4) COMP VALUE 15.
    05 NORM                  PIC S9(4) COMP VALUE 0.
    05 NOREPEAT              PIC S9(4) COMP VALUE 0.
    05 REPEAT                PIC S9(4) COMP VALUE 1.
    05 REPEATAPP             PIC S9(4) COMP VALUE 2.
    05 FORWARDS              PIC S9(4) COMP VALUE 1.
    05 BACKWARDS             PIC S9(4) COMP VALUE -1.
    05 ENTRY-ERRER-NUM       PIC S9(4) COMP VALUE 0.
 01 FUNC-KEYS.
    05 ENTERKEY              PIC S9(4) COMP VALUE  0.
    05 HEADKEY               PIC S9(4) COMP VALUE  1.
    05 DELETEKEY             PIC S9(4) COMP VALUE  2.
    05 PRINTKEY              PIC S9(4) COMP VALUE  3.
    05 REFRESHKEY            PIC S9(4) COMP VALUE  4.
    05 PREVKEY               PIC S9(4) COMP VALUE  5.
    05 NEXTKEY               PIC S9(4) COMP VALUE  6.
    05 BROWSEKEY             PIC S9(4) COMP VALUE  7.
    05 EXITKEY               PIC S9(4) COMP VALUE  8.
 01 ENTRY-ERRERS.
    05 PREV-NOT-ALLOWED      PIC S9(4) COMP VALUE  1.
    05 NO-PREV-RECS          PIC S9(4) COMP VALUE  2.
    05 NOT-REPEATING         PIC S9(4) COMP VALUE  3.
    05 DELETE-NOT-DEFINED    PIC S9(4) COMP VALUE  4.
    05 NO-BATCH-RECS         PIC S9(4) COMP VALUE  5.
    05 NO-BATCH              PIC S9(4) COMP VALUE  6.
    05 NO-NEXT-RECS          PIC S9(4) COMP VALUE  7.
 01 ERRERS                   PIC S9(4) COMP VALUE ZERO.
 01 BATCH                    PIC S9(4) COMP VALUE ZERO.
 01 UNDERLINE                PIC S9(4) COMP VALUE 1.
 01 MESSAGE-BUF.
    05 FILLER                PIC X(4).
    05 STATUS-LINE-INFO      PIC X(26).
    05 IDENT                 PIC X(14).
    05 REC                   PIC X(6).
    05 FILLER                PIC X(13).
    05 MODEE                 PIC X(6).
    05 MODE-TYP              PIC X(7).
 01 MESSAGE-BUF-LEN          PIC S9(4) COMP VALUE 76.
 01 MSGLEN                   PIC S9(4) COMP.
 01 PAGE-EJECT               PIC S9(4) COMP VALUE 49.
 01 LAST-REC-NUM             PIC S9(6) COMP.
 01 DIRECTION                PIC S9(4) COMP.
 01 LOCAL-COM-REC            PIC S9(6) COMP VALUE ZERO.
```

```
01 CNT                       PIC S9(4) COMP.
01 MESSAGE-LENGTH            PIC S9(4) COMP.
01 LOCAL-BUF.
   05 LOCAL-MESSAGE-BUF      PIC X(101).
   05 LOC-BUF REDEFINES LOCAL-MESSAGE-BUF.
      10 PREP                PIC X(17).
      10 ER-MESS             PIC X(76).
      10 TRLR                PIC X(7).
      10 FILLER              PIC X.
01 INDEXX                    PIC S9(4) COMP.
01 FIRST-TIME                PIC S9(4) COMP.
01 SAVED-FORM-NAME.
   05 FILLER                 PIC X(15) VALUE SPACES.
01 NO-BATCH-FILE             PIC S9(4) COMP VALUE 71.
01 VERSIONS-DIFF             PIC S9(4) COMP VALUE 70.
01 DIF-FF                    PIC S9(4) COMP VALUE 73.
01 WARN-FLAG                 PIC S9(4) COMP VALUE 0.
01 TERMFILENAME              PIC X(11) VALUE SPACES.
01 FILENAME                  PIC X(11) VALUE SPACES.
01 READ-FLAG                 PIC S9(4) COMP.
01 MSG-NUM                   PIC S9(4) COMP.
01 ENTRY-PROG-ID            PIC X(51) VALUE
   "HP32209A.01.01 ENTRY(COB) HEWLETT-PACKARD CO. 1980.".
01 ENTRY-PROD-ID             PIC X(20) VALUE
   "ENTRY(COBOL) A.01.01".
01 POST-NO                   PIC S9(6) COMP.
01 POST-TEST                 PIC S9(6) COMP.
01 PARMVAL                   PIC S9(6) COMP VALUE 20.
```

```
$PAGE "  M A I N   P R O G R A M  "
*************************************************************
*                                                           *
*                  MAIN PROGRAM                             *
*                                                           *
*************************************************************
 PROCEDURE DIVISION.

 START.

      PERFORM INIT THRU INIT-EXIT.

      PERFORM COLLECT THRU COLLECT-EXIT.

      PERFORM PROG-EXIT.

      STOP RUN.


$PAGE "             FORMAT-STATUS-LINE"
*************************************************************
*                                                           *
*                  FORMAT-STATUS-LINE                       *
*                                                           *
*************************************************************
 FORMAT-STATUS-LINE.


     MOVE SPACES TO MESSAGE-BUF.
     MOVE ENTRY-PROD-ID TO STATUS-LINE-INFO.
     MOVE "BATCH RECORD #" TO IDENT.
     ADD 1 TO COM-RECNUM.
     MOVE COM-RECNUM TO REC.
     SUBTRACT 1 FROM COM-RECNUM.
     MOVE "MODE: " TO MODEE.
     IF COM-MODE = COLLECT-MODE THEN
        MOVE "COLLECT" TO MODE-TYP
     ELSE
        MOVE "BROWSE" TO MODE-TYP.

     MOVE 76 TO MSGLEN.

     CALL "VPUTWINDOW" USING COMAREA, MESSAGE-BUF, MSGLEN.
```

```
$PAGE "          ENTRY-ERRER"
*************************************************************
*                                                           *
*                    ENTRY-ERRER                            *
*                                                           *
*************************************************************
 ENTRY-ERRER.

     IF ERRERS = 1 THEN GO TO ENT-ERR-EXIT.

      MOVE 1 TO ERRERS.
      MOVE SPACES TO MESSAGE-BUF.
      GO TO E1, E2, E3, E4, E5, E6, E7, DEPENDING
             ON ENTRY-ERRER-NUM.


** 0 IS NOT DEFINED **
 E0.
      GO TO PUT-WINDOW.

** PREV NOT DEFINED      **
 E1.
      MOVE
      " The PREV key is only defined for browse mode."
      TO MESSAGE-BUF.
      GO TO PUT-WINDOW.

** NO PREV RECORDS       **
 E2.
      MOVE
      " There are no previous batch records."
      TO MESSAGE-BUF.
      GO TO PUT-WINDOW.

** NOT REPEATING         **
 E3.
      MOVE
      " The NEXT key is not defined for a non-repeating form."
      TO MESSAGE-BUF.
      GO TO PUT-WINDOW.

** DELETE NOT DEFINED    **
 E4.
      MOVE
      " The DELETE key is only defined for browse mode."
      TO MESSAGE-BUF.
      GO TO PUT-WINDOW.

** NO BATCH RECS         **
 E5.
      MOVE
      " There are no batch records to browse."
      TO MESSAGE-BUF.
      GO TO PUT-WINDOW.
```

```
** NO BATCH                    **
 E6.
      MOVE
      " No batch file was specified, so browse is not allowed."
      TO MESSAGE-BUF.
      GO TO PUT-WINDOW.

** NO NEXT REC    **
 E7.
      MOVE
      " There are no more batch records."
      TO MESSAGE-BUF.



 PUT-WINDOW.
      MOVE 76 TO MSGLEN.

      CALL "VPUTWINDOW" USING COMAREA, MESSAGE-BUF, MSGLEN.

 ENT-ERR-EXIT.
      EXIT.



$PAGE "           ERRER"
*************************************************************
*                                                           *
*                          ERRER                            *
*                                                           *
*************************************************************
 ERRER.

     IF ERRERS = 1 THEN GO TO ERRER-EXIT.

     MOVE 1 TO ERRERS.

     MOVE SPACES TO MESSAGE-BUF.
     CALL "VERRMSG" USING COMAREA, MESSAGE-BUF,
                          MESSAGE-BUF-LEN, MSGLEN.

     MOVE 0 TO COM-STATUS.
     CALL "VPUTWINDOW" USING COMAREA,
                             MESSAGE-BUF, MSGLEN.

 ERRER-EXIT.
     EXIT.
```

```
$PAGE "          PRINT-MSG"
*************************************************************
*                                                           *
*                  PRINT-MSG                                *
*                                                           *
*************************************************************
 PRINT-MSG.
     MOVE SPACES TO MESSAGE-BUF.
     CALL "VERRMSG" USING COMAREA, MESSAGE-BUF,
                                 MESSAGE-BUF-LEN, MSGLEN.
     DISPLAY MESSAGE-BUF.
     MOVE 0 TO COM-STATUS.




$PAGE "                INITIALIZATION PROCEDURE"
*************************************************************
*                                                           *
*                         INIT                              *
*                                                           *
*************************************************************
 INIT.

 DISPLAY ENTRY-PROG-ID.


** INITIALIZE COMAREA -- IS ALL ZERO'S TO START **
     MOVE COBOL-LANG TO COM-LANGUAGE.
     MOVE COMAREALEN TO COM-COMAREALEN.
     MOVE 1 TO COM-RECNUM.
     MOVE 1 TO BATCH.

** NOW, OPEN THE FORMS FILE **
 OPEN-FORM.
     MOVE 0 TO COM-STATUS.
     MOVE SPACES TO FILENAME.
     MOVE 1 TO MSG-NUM.
     PERFORM PRINT-TO-TERM THRU PRINT-TERM-EXIT.
     PERFORM READ-FROM-TERM.
     IF READ-FLAG = 1 THEN STOP RUN.
     CALL "VOPENFORMF" USING COMAREA, MESSAGE-BUF.
     IF COM-STATUS IS NOT EQUAL TO 0 THEN
        PERFORM WRITE-MSG
        GO TO OPEN-FORM.

** NOW, OPEN THE BATCH FILE **

 OPEN-BATCH.
     MOVE 0 TO COM-STATUS.
     MOVE 2 TO MSG-NUM.
     PERFORM PRINT-TO-TERM THRU PRINT-TERM-EXIT.
     PERFORM READ-FROM-TERM.
     IF READ-FLAG = 1 THEN
        MOVE 0 TO BATCH
        GO TO OPEN-TERM.
     CALL "VOPENBATCH" USING COMAREA, MESSAGE-BUF.
     IF COM-STATUS = 0 THEN GO TO OPEN-TERM.
     IF COM-STATUS = VERSIONS-DIFF OR
        COM-STATUS = DIF-FF THEN
        NEXT SENTENCE
     ELSE
        PERFORM WRITE-MSG
        GO TO OPEN-TERM.
     IF COM-STATUS = DIF-FF THEN MOVE 3 TO MSG-NUM
     ELSE MOVE 4 TO MSG-NUM.
     PERFORM PRINT-TO-TERM THRU PRINT-TERM-EXIT.
     MOVE 5 TO MSG-NUM.
     PERFORM PRINT-TO-TERM THRU PRINT-TERM-EXIT.
     PERFORM READ-FROM-TERM.
     IF READ-FLAG IS NOT EQUAL TO 1 THEN
        IF MESSAGE-BUF = "Y" THEN
           MOVE 0 TO COM-STATUS.

**
** ALL OK TO HERE, SO LET'S OPEN THE TERMINAL **
**
 OPEN-TERM.
     IF BATCH = 0 OR COM-STATUS = 0 THEN
        NEXT SENTENCE
     ELSE
        MOVE 0 TO COM-STATUS
        CALL "VCLOSEBATCH" USING COMAREA
        CALL "VCLOSEFORMF" USING COMAREA
        GO TO OPEN-FORM.
     MOVE "A264X " TO TERMFILENAME.
     CALL "VOPENTERM" USING COMAREA, TERMFILENAME.
     IF COM-STATUS = 0 THEN GO TO INIT-EXIT.
     PERFORM WRITE-MSG.
     STOP RUN.

 PRINT-TO-TERM.
     MOVE SPACES TO MESSAGE-BUF.
     GO TO FF-NAME, BF-NAME, FF-DIF, FF-MOD, Y-OR-N
     DEPENDING ON MSG-NUM.

 FF-NAME.
     MOVE "Enter FORMS FILE name and press return: "
        TO MESSAGE-BUF.
     GO TO PRINT-IT.
 BF-NAME.
     MOVE "Enter BATCH FILE name and press return: "
        TO MESSAGE-BUF.
     GO TO PRINT-IT.
 FF-DIF.
     MOVE "WARNING: A DIFFERENT FORMS FILE WAS USED TO CREATE THIS
-    " BATCH."
```

```
         TO MESSAGE-BUF.
         GO TO PRINT-IT.
     FF-MOD.
         MOVE "WARNING: FORMS FILE WAS RECOMPILED SINCE THIS BATCH WAS
-        " CREATED."
         TO MESSAGE-BUF.
         GO TO PRINT-IT.
     Y-OR-N.
         MOVE "Enter Y to continue: "
            TO MESSAGE-BUF.
     PRINT-IT.
         DISPLAY MESSAGE-BUF.
     PRINT-TERM-EXIT.
         EXIT.

     READ-FROM-TERM.
         MOVE 0 TO READ-FLAG.
         MOVE SPACES TO MESSAGE-BUF.
         ACCEPT MESSAGE-BUF.
         IF MESSAGE-BUF = "                    " THEN
            MOVE 1 TO READ-FLAG.
     WRITE-MSG.
         MOVE SPACES TO MESSAGE-BUF.
         CALL "VERRMSG" USING COMAREA, MESSAGE-BUF,
                              MESSAGE-BUF-LEN, MSGLEN.
         DISPLAY MESSAGE-BUF.
     INIT-EXIT.
         EXIT.



    $PAGE "          EXIT"
    ************************************************************
    *                                                          *
    *                 PROG-EXIT                                *
    *                                                          *
    ************************************************************
     PROG-EXIT.

    ** FIRST CLOSE TERMINAL **
           CALL "VCLOSETERM" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 THEN PERFORM PRINT-MSG.

    ** NOW, BATCH FILE **
           IF BATCH = 1 AND ERRERS = 0 THEN
              CALL "VCLOSEBATCH" USING COMAREA
              IF COM-STATUS IS NOT EQUAL TO 0 THEN
                 PERFORM PRINT-MSG.

    ** NOW, CLOSE FORMS FILE **
           CALL "VCLOSEFORMF" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 THEN PERFORM PRINT-MSG.
```

```
$PAGE "            BROWSE"
*************************************************************
*                                                           *
*                    BROWSE                                 *
*                                                           *
*************************************************************
 BROWSE.

      MOVE COM-RECNUM TO LOCAL-COM-REC.
      SUBTRACT 1 FROM COM-RECNUM.
      MOVE BACKWARDS TO DIRECTION.
 BROWSE-START.
      IF COM-NUMRECS = 0 THEN
         GO TO BROWSE-EXIT.
      IF COM-RECNUM = LAST-REC-NUM THEN
         MOVE NO-NEXT-RECS TO ENTRY-ERRER-NUM
         PERFORM ENTRY-ERRER THRU ENT-ERR-EXIT
         SUBTRACT 1 FROM COM-RECNUM
         MOVE BACKWARDS TO DIRECTION.
      IF COM-RECNUM < 0 THEN
         MOVE NO-PREV-RECS TO ENTRY-ERRER-NUM
         PERFORM ENTRY-ERRER THRU ENT-ERR-EXIT
         MOVE 0 TO COM-RECNUM
         MOVE FORWARDS TO DIRECTION.
      CALL "VREADBATCH" USING COMAREA.
      IF COM-STATUS IS NOT EQUAL TO 0 THEN
         PERFORM ERRER THRU ERRER-EXIT.

      IF COM-DELETEFLAG IS NOT EQUAL TO 0 THEN GO TO REC-DELETED.

      IF COM-RECNUM = LOCAL-COM-REC AND COM-LASTKEY
         IS NOT EQUAL TO REFRESHKEY THEN GO TO SAME-FORM.

      IF DIRECTION = BACKWARDS
         OR COM-LASTKEY = REFRESHKEY THEN
            MOVE NORM TO COM-NFOPT
            MOVE NORM TO COM-REPEATOPT
      ELSE
         IF COM-CFNAME IS NOT EQUAL TO COM-NFNAME THEN
            MOVE NORM TO COM-REPEATOPT.
      IF COM-LASTKEY = REFRESHKEY THEN
         MOVE "$REFRESH       " TO COM-NFNAME.
      CALL "VGETNEXTFORM" USING COMAREA.
      IF COM-STATUS IS NOT EQUAL TO 0 THEN
         PERFORM ERRER THRU ERRER-EXIT.
      MOVE COM-RECNUM TO LOCAL-COM-REC.
      GO TO CHECK-ERRER.
 SAME-FORM.
 CHECK-ERRER.
      IF ERRERS = 0 THEN
         PERFORM FORMAT-STATUS-LINE.
 TEST-ERRERS.
      MOVE 0 TO ERRERS.
      CALL "VSHOWFORM" USING COMAREA.
```

```
           IF COM-STATUS IS NOT EQUAL TO 0 THEN
              PERFORM ERRER THRU ERRER-EXIT.
           MOVE 0 TO COM-SHOWCONTROL.
           CALL "VREADFIELDS" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 THEN
              PERFORM ERRER THRU ERRER-EXIT.
           IF ERRERS = 0 AND COM-LASTKEY = 0 THEN GO TO EKEY.
           IF ERRERS = 0 THEN
              GO TO  HKEY,  DKEY, PKEY, RKEY,
                     PRKEY, NKEY, CKEY, EXKEY
                 DEPENDING ON COM-LASTKEY.

     ** ENTER KEY **
      EKEY.
           MOVE FORWARDS TO DIRECTION.
           CALL "VFIELDEDITS" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 OR COM-NUMERRS
              IS NOT EQUAL TO 0 THEN
              PERFORM ERRER THRU ERRER-EXIT.
           IF ERRERS = 1 THEN GO TO KEY-EXIT.
           CALL "VFINISHFORM" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 OR COM-NUMERRS
              IS NOT EQUAL TO 0 THEN
              PERFORM ERRER THRU ERRER-EXIT.

           IF COM-REPEATOPT = NOREPEAT AND COM-NFOPT
              IS NOT EQUAL TO NORM OR COM-REPEATOPT = REPEATAPP THEN
              CALL "VSHOWFORM" USING COMAREA
              IF COM-STATUS IS NOT EQUAL TO 0 THEN
                 PERFORM ERRER THRU ERRER-EXIT.
              IF ERRERS = 0 THEN
              CALL "VWRITEBATCH" USING COMAREA
              IF COM-STATUS IS NOT EQUAL TO 0 THEN
                 PERFORM ERRER THRU ERRER-EXIT.
              IF ERRERS = 0 THEN ADD 1 TO COM-RECNUM.

           GO TO KEY-EXIT.
     ** HEAD KEY **
      HKEY.
            MOVE FORWARDS TO DIRECTION.
            MOVE 0 TO COM-RECNUM.
            MOVE NORM TO COM-REPEATOPT.
            MOVE NORM TO COM-NFOPT.
            GO TO KEY-EXIT.
      DKEY.
     ** DELETE KEY **
           MOVE FORWARDS TO DIRECTION.

           MOVE 1 TO COM-DELETEFLAG.
           CALL "VWRITEBATCH" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 THEN
               PERFORM ERRER THRU ERRER-EXIT.
      MOVE 0 TO COM-DELETEFLAG.
      IF ERRERS = 0 THEN
         ADD 1 TO COM-RECNUM
         MOVE NORM TO COM-REPEATOPT
         MOVE NORM TO COM-NFOPT
         GO TO KEY-EXIT.

 PKEY.
** PRINT KEY **
     CALL "VPRINTFORM" USING COMAREA, UNDERLINE, PAGE-EJECT.
     IF COM-STATUS IS NOT EQUAL TO 0 THEN
        PERFORM ERRER THRU ERRER-EXIT.
     GO TO KEY-EXIT.

 RKEY.
**REFRESH KEY **
     GO TO KEY-EXIT.

 PRKEY.
** PREVIOUS KEY **
     MOVE BACKWARDS TO DIRECTION.
     SUBTRACT 1 FROM COM-RECNUM.
     GO TO KEY-EXIT.

 NKEY.
** NEXT KEY **
     MOVE FORWARDS TO DIRECTION.
     ADD 1 TO COM-RECNUM.

     IF COM-REPEATOPT = NOREPEAT
            AND COM-NFOPT IS NOT EQUAL TO NORM OR
               COM-REPEATOPT = REPEATAPP THEN
                      CALL "VSHOWFORM" USING COMAREA
                      IF COM-STATUS IS NOT EQUAL TO 0
                      THEN PERFORM ERRER THRU ERRER-EXIT.
            GO TO KEY-EXIT.

       CKEY.
      ** COLLECT KEY **
           GO TO BROWSE-EXIT.

       EXKEY.
      ** EXIT KEY **
           GO TO BROWSE-EXIT.

       KEY-EXIT.
           IF ERRERS = 0 AND COM-LASTKEY IS NOT EQUAL TO PRINTKEY
                    THEN NEXT SENTENCE ELSE GO TO TEST-ERRERS.
           GO TO BROWSE-START.

       REC-DELETED.
           IF DIRECTION = BACKWARDS THEN
               SUBTRACT 1 FROM COM-RECNUM ELSE
                      ADD 1 TO COM-RECNUM.
        GO TO BROWSE-START.

    BROWSE-EXIT.
        EXIT.




$PAGE "          COLLECT"
*************************************************************
*                                                           *
*                        COLLECT                            *
*                                                           *
*************************************************************
 COLLECT.


     MOVE COLLECT-MODE TO COM-MODE.
     MOVE 1 TO FIRST-TIME.
     MOVE 0 TO COM-NUMERRS.
     MOVE 0 TO COM-DELETEFLAG.
     MOVE 0 TO ERRERS.
 FORM-IS-EXIT.
     IF COM-LASTKEY = ENTERKEY OR
        COM-LASTKEY = NEXTKEY THEN
        IF COM-REPEATOPT = NOREPEAT AND COM-NFOPT
           IS NOT EQUAL TO NORM OR
           COM-REPEATOPT = REPEATAPP THEN NEXT SENTENCE
        ELSE GO TO NOT-ERRER.
     CALL "VSHOWFORM" USING COMAREA.
     IF COM-STATUS IS NOT EQUAL TO 0 THEN
        PERFORM ERRER THRU ERRER-EXIT.
 NOT-ERRER.
     CALL "VGETNEXTFORM" USING COMAREA.
     IF FIRST-TIME = 1 AND COM-STATUS IS NOT EQUAL TO 0 THEN
        CALL "VERRMSG" USING COMAREA, MESSAGE-BUF,
        MESSAGE-BUF-LEN, MSGLEN
        MOVE 1 TO ERRERS
        GO TO COLLECT-EXIT.
     IF COM-STATUS IS NOT EQUAL TO 0 THEN
        PERFORM ERRER THRU ERRER-EXIT.
     MOVE 0 TO FIRST-TIME.
     CALL "VINITFORM" USING COMAREA.
     IF COM-STATUS IS NOT EQUAL TO 0 OR
        COM-NUMERRS IS NOT EQUAL TO 0 THEN
        PERFORM ERRER THRU ERRER-EXIT.
     IF ERRERS = 0 THEN PERFORM FORMAT-STATUS-LINE.

 COLLECT-ERRERS.
     MOVE 0 TO ERRERS.
     CALL "VSHOWFORM" USING COMAREA.
     IF COM-STATUS IS NOT EQUAL TO 0 THEN
        PERFORM ERRER THRU ERRER-EXIT.
     MOVE 0 TO COM-SHOWCONTROL.
```

```
           IF COM-DBUFLEN > 0 OR
              COM-REPEATOPT IS NOT EQUAL TO  NOREPEAT OR
              COM-NFOPT = NORM THEN GO TO READ-IT.
           IF ERRERS IS NOT EQUAL TO 0 OR
              BATCH IS NOT EQUAL TO 1 THEN
                 GO TO COLL-KEY-EXIT.
      *
      *    THE SPECIAL CASE: NOREPEAT FORM WITHOUT ANY INPUT FIELD,
      *    SO SIMPLY WRITE BATCH, DON'T BOTHER TO READ FIELDS.
           CALL "VWRITEBATCH" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 THEN
              PERFORM ERRER THRU ERRER-EXIT.
           IF ERRERS = 0 THEN
           ADD 1 TO COM-RECNUM
           DIVIDE COM-RECNUM BY PARMVAL GIVING POST-NO
           REMAINDER POST-TEST
           IF POST-TEST = 0 THEN
               CALL "VPOSTBATCH" USING COMAREA.
           GO TO NOT-NORMAL-FORM.

       READ-IT.
           CALL "VREADFIELDS" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 THEN
              PERFORM ERRER THRU ERRER-EXIT.
           IF ERRERS IS NOT EQUAL TO 0 THEN GO TO NOT-NORMAL-FORM.

           IF COM-LASTKEY = 0 THEN GO TO ENT-KEY.

           GO TO  HEAD-KEY, DEL-KEY, PRNT-KEY, REF-KEY,
                  PREV-KEY, NXT-KEY, BRS-KEY,  EXIT-KEY
                  DEPENDING ON COM-LASTKEY.

       ENT-KEY.

      ** ENTER KEY **

           CALL "VFIELDEDITS" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 OR
               COM-NUMERRS IS NOT EQUAL TO 0 THEN
                   PERFORM ERRER THRU ERRER-EXIT.
           IF ERRERS = 1 THEN GO TO COLL-KEY-EXIT.
           CALL "VFINISHFORM" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 OR
               COM-NUMERRS IS NOT EQUAL TO 0 THEN
                   PERFORM ERRER THRU ERRER-EXIT.

           IF ERRERS = 0 AND BATCH = 1 THEN NEXT SENTENCE
               ELSE GO TO COLL-KEY-EXIT.
           CALL "VWRITEBATCH" USING COMAREA.
           IF COM-STATUS IS NOT EQUAL TO 0 THEN
               PERFORM ERRER THRU ERRER-EXIT.
```

```
         IF ERRERS = 0 THEN
             ADD 1 TO COM-RECNUM
             DIVIDE COM-RECNUM BY PARMVAL GIVING POST-NO
             REMAINDER POST-TEST
             IF POST-TEST = 0 THEN
                 CALL "VPOSTBATCH" USING COMAREA
             GO TO COLL-KEY-EXIT.

     HEAD-KEY.

    ** HEAD KEY **

         MOVE NORM TO COM-REPEATOPT.
         MOVE NORM TO COM-NFOPT.
         MOVE "$HEAD          " TO COM-NFNAME.
         GO TO COLL-KEY-EXIT.

     DEL-KEY.

    ** DELETE KEY **

         MOVE DELETE-NOT-DEFINED TO ENTRY-ERRER-NUM.
         PERFORM ENTRY-ERRER THRU ENT-ERR-EXIT.
         GO TO COLL-KEY-EXIT.

     PRNT-KEY.

    ** PRINT KEY **

         CALL "VPRINTFORM" USING COMAREA, UNDERLINE, PAGE-EJECT.
         IF COM-STATUS IS NOT EQUAL TO 0 THEN
            PERFORM ERRER THRU ERRER-EXIT.
         GO TO COLL-KEY-EXIT.

     REF-KEY.

    ** REFRESH KEY **

         MOVE "$REFRESH       " TO COM-NFNAME.
         GO TO COLL-KEY-EXIT.

     PREV-KEY.

    ** PREVIOUS KEY **

         MOVE PREV-NOT-ALLOWED TO ENTRY-ERRER-NUM.
         PERFORM ENTRY-ERRER THRU ENT-ERR-EXIT.
         GO TO COLL-KEY-EXIT.

     NXT-KEY.

    ** NEXT KEY **

         IF COM-REPEATOPT = NORM THEN
```

```
              MOVE NOT-REPEATING TO ENTRY-ERRER-NUM
              PERFORM ENTRY-ERRER THRU ENT-ERR-EXIT
              GO TO COLL-KEY-EXIT
           ELSE
              MOVE NORM TO COM-REPEATOPT
              GO TO COLL-KEY-EXIT.

       BRS-KEY.

      ** BROWSE KEY **

           IF BATCH = 0 THEN
              MOVE NO-BATCH TO ENTRY-ERRER-NUM
              PERFORM ENTRY-ERRER THRU ENT-ERR-EXIT
              GO TO COLL-KEY-EXIT
           ELSE
              IF COM-NUMRECS = 0 THEN
                 MOVE NO-BATCH-RECS TO ENTRY-ERRER-NUM
                 PERFORM ENTRY-ERRER THRU ENT-ERR-EXIT
                 GO TO COLL-KEY-EXIT.
           MOVE COM-RECNUM TO LAST-REC-NUM.
           MOVE COM-CFNAME TO SAVED-FORM-NAME.
           MOVE BROWSE-MODE TO COM-MODE.
           MOVE SAVED-FORM-NAME TO COM-NFNAME.
           MOVE LAST-REC-NUM TO COM-RECNUM.
           MOVE NORM TO COM-REPEATOPT.
           MOVE NORM TO COM-NFOPT.
           PERFORM BROWSE THRU BROWSE-EXIT.
           MOVE COLLECT-MODE TO COM-MODE.
           MOVE SAVED-FORM-NAME TO COM-NFNAME.
           MOVE LAST-REC-NUM TO COM-RECNUM.
           MOVE NORM TO COM-REPEATOPT.
           MOVE NORM TO COM-NFOPT.
           MOVE 0 TO COM-DELETEFLAG.
           IF COM-LASTKEY = EXITKEY THEN
              MOVE SAVED-FORM-NAME TO COM-CFNAME
              GO TO COLLECT-EXIT.
           GO TO COLL-KEY-EXIT.
       EXIT-KEY.

      ** EXIT KEY **

           GO TO COLLECT-EXIT.
       COLL-KEY-EXIT.
```

```
NOT-NORMAL-FORM.
    IF ERRERS = 0 AND COM-LASTKEY IS NOT EQUAL TO PRINTKEY THEN
       NEXT SENTENCE ELSE GO TO COLLECT-ERRERS.
    IF COM-NFNAME IS NOT EQUAL TO "$END           " OR
       COM-REPEATOPT IS NOT EQUAL TO NORM THEN
       GO TO FORM-IS-EXIT.

COLLECT-EXIT.
    EXIT.
```

```
$CONTROL LIST,SOURCE,WARN,MAP,USLINIT,QUOTE=",SEG=4
H
F*
F*
F*  ENTRYRPG VERSION A.01.01  9/1/80
F*
F*
F* THE "B" IN COL. 52 OF THE F'SPEC INDICATES THAT BREAK
F* IS TO BE ENABLED.  AFTER DEBUGGING THE "B" SHOULD BE
F* REPLACED WITH " " TO ELIMINATE PROBLEMS CAUSED WHEN
F* THE BREAK KEY IS HIT INSTEAD OF THE ENTER KEY.
FTERMINALUD  V     300            WORKSTN    3B
F                                              KFORMS NEWFORMS
F                                              KBATCH BATCHF
F                                              KTRACE TRACEF
I*
I* DECLARE ALL INPUT RECORDS.  INPUT INDICATOR NUMBER IS
I* TEN GREATER THAN THE EVENT NUMBER.  ALL RECORDS ARE
I* IDENTIFIED BY THE EVENT NUMBER IN COL'S 1-2.
I*
ITERMINALAA  10    1 C0    2 C0
I                                         1   20EVENT
I                                         3  17 CURFRM
I                                        18  210DATALN
I                                        22 200 DATA
I* EVENT 01
I        BB  11    1 C0    2 C1
I                                         1   20EVENT
I* EVENT 02
I        CC  12    1 C0    2 C2
I                                         1   20EVENT
I* EVENT 03
I        DD  13    1 C0    2 C3
I                                         1   20EVENT
I* EVENT 04
I        EE  14    1 C0    2 C4
I                                         1   20EVENT
I* EVENT 05
I        FF  15    1 C0    2 C5
I                                         1   20EVENT
I* EVENT 06
I        GG  16    1 C0    2 C6
I                                         1   20EVENT
I* EVENT 07
I        HH  17    1 C0    2 C7
I                                         1   20EVENT
I* EVENT 08
I        II  18    1 C0    2 C8
I                                         1   20EVENT
I* EVENT 09
I        JJ  19    1 C0    2 C9
I                                         1   20EVENT
I                                         3  17 CURFRM
I                                        18  220NUMERS      30
```

```
I* EVENT 10
I        KK  20   1 C1   2 C0
I                                        1   20EVENT
I                                        3  17 CURFRM
I                                       18  210DATALN
I                                       22 200 DATA
I* EVENT 11
I        LL  21   1 C1   2 C1
I                                        1   20EVENT
I                                        3  17 CURFRM
I                                       18  22 BTNUMA
I                                       18  220BTNUMN        3133
I                                       23  230MCODE       34
I                                       24  24 RPTAPN
I                                       25  25 FRZAPN
I                                       26  40 NEXTFM
I                                       41  45 NONDEL
I* EVENT 12
I        MM  22   1 C1   2 C2
I                                        1   20EVENT
I                                        3  17 CURFRM
I                                       18  220FLDNUM
I                                       23  260FLDLNG
I                                       27 150 FLDATA
C*
C*
C* THE FILES WILL BE OPENED BY RPG, SO I NEED NOT BE
C* CONCERNED WITH THEM.
C*
C* INITIALIZE SOME FIELDS.  (REALLY, DEFINE THEM.)
C*
C                     MOVEL"  "      MESSAG 79
C                     MOVE " "       ENHAN   1
C                     Z-ADD79        MSGLEN  20
C                     Z-ADD0         BADFLD  50
C*
C*
C*
C* THE MAIN PROGRAM CALLS COLLECT.  COLLECT DOES THE DATA
C* COLLECTION.  COLLECT WILL CALL BROWSE TO ENTER BROWSE
C* MODE.  COLLECT RETURNS TO THE MAIN PROGRAM TO STOP.
C*
C* WHEN THE BATCH FILE IS OPENED, AN EVENT 11 IS MADE
C* AVAILABLE.  READ IT TO FIND THE BATCH RECORD NUMBER.
C* THE BATCH RECORD NUMBER WILL BE ZERO IF THE BATCH FILE
C* IS NEW.  IF THE RECORD NUMBER IS GREATER THAN ZERO, THE
C* FILE IS OLD.  IF THE PROGRAMMER WISHES, COL'S 41-45 MAY
C* BE CHECKED TO FIND THE NUMBER OF NONDELETED BATCH RECORDS.
C*
C                     READ TERMINAL                 HO
C*
C                     EXSR COLLCT
C*
C* RETURN FROM COLLECT AT END OF EXECUTION.
```

```
C*
C*
C                          SETON                           LR
C*
C*
C* FILES WILL BE CLOSED BY RPG.
C*
C*
C*
C* SUBROUTINE STDMESSAGE WILL FORMAT THE MESSAGE WINDOW
C* WITH THE NORMAL INFORMATION.
C*
CSR           STDMSG    BEGSR
C*
C* DISPLAY ENHANCEMENT FOR THE MODE PORTION OF THE MESSAGE
C* IS HALF-BRIGHT, INVERSE VIDEO FOR COLLECT MODE.  FOR
C* BROWSE MODE IT IS HALF-BRIGHT, INVERSE VIDEO, BLINKING.
C*
CSR                     MOVE "dJ"     MODENH  4
C*                           ©  ESCAPE CHARACTER
CSR 01                  MOVE "C"        MODENH
CSR                     SETON                          82
CSR                     EXCPT
CSR                     SETOF                          82
CSR                     ENDSR
C*
C*
C*
C* SUBROUTINE COLLECT IS CALLED FROM THE OUTER BLOCK.
C* COLLECT CALLS BROWSE TO ENTER BROWSE MODE.
C* COLLECT WILL RETURN TO THE MAIN PROGRAM TO END EXECUTION.
C*
CSR           COLLCT    BEGSR
CSR                     MOVE "COLLECT" MODE    7
CSR           BEGCOL    TAG
C*
C* SET ALL INPUT INDICATORS OFF.  THIS MUST BE DONE HERE
C* SINCE WE DO NOT GO THROUGH THE NORMAL INPUT CYCLE.
C*
CSR                     SETOF                          101112
CSR                     SETOF                          131415
CSR                     SETOF                          161718
CSR                     SETOF                          182021
CSR                     SETOF                          223033
CSR                     SETOF                          3334
C*
C* GET AND INITIALIZE THE NEXT FORM.
C*
C* WE SET INDICATOR 02 ON WHEN WE DON'T WANT TO GET THE
C* NEXT FORM.  THIS WILL BE USED WHEN THE PREVIOUS COMMAND
C* WAS ONE SUCH AS "PRINT".
C*
C* INDICATOR 99 CONDITIONS THE UNIVERSAL OUTPUT RECORD FOR
C* ACTIONS.  TO USE THIS RECORD, THE FIELD "ACTION" MUST
```

```
C* CONTAIN THE ACTION NAME OR NUMBER.
C*
CSRN02                MOVE "GETNXT"  ACTION  6
CSRN02                SETON                         99
CSRN02                EXCPT
CSRN02                MOVE "INIT   "  ACTION
CSRN02                EXCPT
CSR                   SETOF                         0299
C*
C* USER PROGRAM INITIALIZATIONS  (EG. VALUES FROM A DATABASE)
C* WOULD BE DONE AT THIS POINT.
C*
C* SET UP STANDARD MESSAGE WINDOW.
C*
C* WE SET INDICATOR 03 ON WHENEVER WE DO NOT WANT THE STANDARD
C* MESSAGE PLACED IN THE WINDOW.  THIS IS NORMALLY USED WHEN
C* A COMMAND PUTS A WARNING MESSAGE IN THE WINDOW, AND WE
C* DON'T WANT THE MESSAGE OVERWRITTEN.
C*
CSRN03                EXSR STDMSG
CSR                   SETOF                         03
C*
C* THIS IS THE POINT WE JUMP TO IF EDIT ERRORS WERE FOUND.
C*
CSR        ERPT       TAG
CSR                   SETON                         99     MAY BE OFF
CSR                   SETOF                         091030
C*
C* SHOW THE FORM AND GET INPUT FROM THE USER.
C*
CSR                   MOVE "SHOW   "  ACTION
CSR                   EXCPT
CSR                   MOVE "RDTERM"  ACTION
CSR                   EXCPT
CSR                   SETOF                         99
CSR                   READ TERMINAL                     HO
C*
C* NOW EXECUTE THE SECTION CONDITIONED BY THE INPUT IND.
C*
C* EVENT 0 - ENTER KEY WAS PRESSED.  REPEAT EDITS UNTIL NO
C* ERRORS ARE FOUND.  THEN DO FINISH PHASE PROCESSING AND
C* WRITE THE BATCH RECORD.  NOTICE THAT USER PROGRAM EDITS
C* COULD BE ADDED TO THIS SECTION.
C*
CSR 10                MOVE "EDITS "  ACTION
CSR 10                SETON                         99
CSR 10                EXCPT
CSR 10                SETOF                         99
CSR 10                READ TERMINAL                     HO
C*
C* EVENT 9 IS READ.  NUMERS WILL CONTAIN THE NUMBER OF EDIT
C* ERRORS FOUND.  IF NUMERS > 0 INDICATOR 30 WILL BE SET.
C*
CSR 10 30             GOTO ERPT
```

```
C*
C* USER PROGRAM EDITS COULD BE INCLUDED AT THIS POINT.
C*
C* IF NO ERRORS, FINISH-UP AND WRITE THE BATCH RECORD.
C*
C* FIRST, SHOW THE FORM TO CLEAR ANY ERROR ENHANCEMENTS.
C*
CSR 10                MOVE "SHOW   "  ACTION
CSR 10                SETON                     99
CSR 10                EXCPT
C*
CSR 10                MOVE "FINISH"   ACTION
CSR 10                EXCPT
CSR 10                MOVE "WRTBAT"   ACTION
CSR 10                EXCPT
CSR 10                SETOF                     99
C*
C* READ THE BATCH RECORD NUMBER.  THE NUMBER IS ALWAYS THAT
C* OF THE NEXT RECORD TO BE WRITTEN.
C*
CSR 10                READ TERMINAL                HO BATCH REC #
C*
C* EVENT 1 - "HEAD" KEY PRESSED.  SET THE NEXT FORM NAME TO
C* $HEAD, CLEAR REPEAT AND FREEZE.
C*
CSR 11                MOVE "        "NEXTFM
CSR 11                MOVEL"$HEAD   " NEXTFM
CSR 11                MOVE "0"        RPTAPN  1
CSR 11                MOVE "0"        FRZAPN  1
C*
C* INDICATOR 50 IS THE OUTPUT INDICATOR FOR THE SPECIAL RECORD
C* DEFINED FOR THE CHANGE NEXT-FORM-NAME ACTION.  IN GENERAL,
C* SPECIAL RECORDS ARE DEFINED FOR ALL ACTIONS WHICH REQUIRE
C* THEM.  THE OUTPUT INDICATORS FOR THESE RECORDS ARE THE ACTION
C* NUMBERS OF THE ACTIONS.
C*
CSR 11                SETON                     50
CSR 11                EXCPT
CSR 11                SETOF                     50
C*
C* EVENT 2 IS NOT DEFINED IN COLLECT MODE.  WRITE MESSAGE
C* TO THE OPERATOR.
C*
CSR 12                SETON                     800302
C*
C* INDICATOR 80 IS THE OUTPUT INDICATOR FOR THE FUNCTION
C* NOT DEFINED IN COLLECT/BROWSE MODE MESSAGE.  THE FIELD
C* "MODE" CONTAINS THE CURRENT MODE, "FUNCT" CONTAINS THE
C* FUNCTION KEY NAME.  ("F1", "F2" ... )
C*
CSR 12                MOVE "F2"       FUNCT   2
CSR 12                EXCPT
CSR 12                SETOF                     80
C*
```

```
C* EVENT 3 - "PRINT" KEY.  PRINT THE CURRENT FORM.
C*
CSR 13                MOVE "PRINT "  ACTION
CSR 13                SETON                     9902
CSR 13                EXCPT
CSR 13                SETOF                     99
C*
C* EVENT 4 -  "REFRESH"   NEXT FORM NAME <-- $REFRESH.
C*
CSR 14                MOVE "        "NEXTFM
CSR 14                MOVEL"$REFRESH"NEXTFM
CSR 14                SETON                     50
CSR 14                EXCPT
CSR 14                SETOF                     50
C*
C* EVENT 5 - NOT DEFINED
C*
CSR 15                MOVE "F5"      FUNCT
CSR 15                SETON                     800302
CSR 15                EXCPT
CSR 15                SETOF                     80
C*
C* EVENT 6 - "NEXT"  GO TO THE NEXT FORM (STOP REPEATING).
C*
CSR 16      RPTAPN    COMP "0"                      81
CSR 16N81             MOVE "        "NEXTFM
CSR 16N81             MOVEL"       " NEXTFM
CSR 16N81             MOVE " "       FRZAPN  1
CSR 16N81             MOVE "0"       RPTAPN  1
CSR 16N81             SETON                     50
CSR 16N81             EXCPT
CSR 16N81             SETOF                     50
CSR 16 81             SETON                     03
CSR 16 81             EXCPT
CSR 16                SETOF                     81
C*
C* EVENT 7 - "BROWSE"  ENTER BROWSE MODE.
C*
C* WHEN IN BROWSE MODE, "OBTNUM" WILL CONTAIN THE NUMBER OF
C* THE NEXT BATCH RECORD TO BE WRITTEN.  THIS NUMBER IS USED
C* TO SHOW WHEN WE ARE AT THE UPPER END OF THE BATCH FILE.
C*
CSR 17                Z-ADDBTNUMN    OBTNUM  50
CSR 17                EXSR BROWSE
C*
C* IF 33 IS SET ON RETURN FROM BROWSE, THE BATCH FILE WAS
C* EMPTY.  IN THIS CASE, DO NOT GET THE NEXT FORM.
CSR 17 33             SETON                     02
C*
CSR 17                MOVE "COLLECT" MODE
C*
C* EVENT 8 - "EXIT"  EXIT THE PROGRAM.
C* IF F8 WAS PRESSED IN BROWSE, IT WILL STILL BE ON.
C*
```

```
C* IF NOT DONE (F8 NOT PRESSED) THEN GO BACK TO BEGINNING.
C*
CSRN18                GOTO BEGCOL
C*
C* OTHERWISE, END OF COLLECT.
C*
CSR                   ENDSR                          END OF COLLECT
C*
C*
C*
C*
C* SUBROUTINE "BROWSE" IS CALLED FROM "COLLECT" TO HANDLE
C* BROWSE MODE.
C*
CSR          BROWSE   BEGSR
CSR                   MOVE "BROWSE " MODE
CSR                   MOVE "PREV   "  ACTION
C*
C* SET REPEAT-APPEND AND FREEZE-APPEND FLAGS OFF
CSR                   MOVE "        "NEXTFM
CSR                   MOVEL"        " NEXTFM
CSR                   MOVE "0"        FRZAPN  1
CSR                   MOVE "0"        RPTAPN  1
CSR                   SETON                     50
CSR                   EXCPT
CSR                   SETOF                     50
C*
C* INDICATOR 01 IS SET WHEN IN BROWSE MODE.
C*
CSR                   SETON                     9901
CSR                   EXCPT
CSR                   SETOF                     99
C*
C* ALL BATCH FILE OPERATIONS RETURN EVENT 11 WHICH CONTAINS
C* THE CURRENT BATCH RECORD NUMBER.
C*
C* INDICATOR 34 WILL BE SET WHEN EVENT 11 INDICATES BROWSE MODE
C*
CSR                   SETOF                     34
C*
CSR                   READ TERMINAL                  HO BATCH REC #
CSR 33N34             GOTO ENDBRW
CSR          BEGBRW   TAG
C*
C* SET OFF ALL INPUT INDICATORS.
C*
C* IF WE ARE OFF THE BOTTOM OF THE BATCH, GO TO THE FIRST
C* NON-DELETED RECORD.
C*
CSR 31                MOVE "NEXT   "  ACTION
CSR 31                SETON                     99
CSR 31                EXCPT
CSR 31                SETOF                     99
CSR 31                READ TERMINAL                    HO
CSR                   SETOF                     101112
CSR                   SETOF                     131415
CSR                   SETOF                     161718
CSR                   SETOF                     192021
CSR                   SETOF                     223031
CSR                   SETOF                     3334
C*
C* GET THE FORM THAT CORRESPONDS TO THE CURRENT BATCH RECORD.
C*
CSRNO2                MOVE "GETNXT"  ACTION
CSRNO2                SETON                     99
CSRNO2                EXCPT
CSR                   SETOF                     0299
C* SET REPEAT-APPEND AND FREEZE-APPEND FLAGS OFF
CSR                   MOVE "         "NEXTFM
CSR                   MOVEL"        " NEXTFM
CSR                   MOVE "0"       FRZAPN  1
CSR                   MOVE "0"       RPTAPN  1
CSR                   SETON                     50
CSR                   EXCPT
CSR                   SETOF                     50
C*
C* SET UP STANDARD MESSAGE WINDOW.
C*
CSRNO3                EXSR STDMSG
CSR                   SETOF                     03
C*
C* THIS IS THE POINT WE JUMP TO IF EDIT ERRORS WERE FOUND.
C*
CSR         ERPTB     TAG
CSR                   SETON                     99      MAY BE OFF
CSR                   SETOF                     091030
C*
C* SHOW THE FORM AND GET INPUT FROM THE USER.
C*
CSR                   MOVE "SHOW  "  ACTION
CSR                   EXCPT
CSR                   MOVE "RDTERM"  ACTION
CSR                   EXCPT
CSR                   SETOF                     99
CSR                   READ TERMINAL                 H0
C*
C* NOW EXECUTE THE SECTION CONDITIONED BY THE INPUT IND.
C*
C* EVENT 0 - ENTER KEY WAS PRESSED.  REPEAT EDITS UNTIL NO
C* ERRORS ARE FOUND.  THEN DO FINISH PHASE PROCESSING AND
C* WRITE THE BATCH RECORD.  NOTICE THAT USER PROGRAM EDITS
C* COULD BE ADDED TO THIS SECTION.
C*
CSR 10                MOVE "EDITS "  ACTION
CSR 10                SETON                     99
CSR 10                EXCPT
CSR 10                SETOF                     99
CSR 10                READ TERMINAL                 H0
```

```
C*
C* EVENT 9 IS READ.  NUMERS WILL CONTAIN THE NUMBER OF EDIT
C* ERRORS FOUND.  IF NUMERS > 0 INDICATOR 30 WILL BE SET.
C*
CSR 10 30             GOTO ERPTB
C*
C* IF NO ERRORS, FINISH-UP AND WRITE THE BATCH RECORD.
C*
CSR 10                MOVE "FINISH"  ACTION
CSR 10                SETON                    9902
CSR 10                EXCPT
CSR 10                MOVE "WRTBAT"  ACTION
CSR 10                EXCPT
CSR 10                SETOF                    99
CSR 10                READ TERMINAL                H0
C*
C* EVENT 1 - "HEAD"  GO TO FIRST BATCH RECORD.
C*
C* IF BATCH RECORD 00000 HAS BEEN DELETED, THE TERMINAL
C* OPERATOR WILL GET A WARNING MESSAGE AND THE FIRST
C* NONDELETED RECORD WILL BE FOUND.
C*
CSR 11                Z-ADD0         BTNUMN
CSR 11                MOVE "00000"   BTNUMA
CSR 11                SETON                    72
CSR 11                EXCPT
CSR 11                SETOF                    72
CSR 11                READ TERMINAL                H0
C*
C* EVENT 2 - "DELETE"  DELETE THE CURRENT BATCH RECORD.
C*
CSR 12                MOVE "DELETE"  ACTION
CSR 12                SETON                    99
CSR 12                EXCPT
CSR 12                SETOF                    99
CSR 12                READ TERMINAL                H0
C*
C* EVENT 3 "PRINT"  PRINT THE CURRENT FORM.
C*
CSR 13                MOVE "PRINT "  ACTION
CSR 13                SETON                    9902
CSR 13                EXCPT
CSR 13                SETOF                    99
C*
C* EVENT 4 "REFRESH"  REREAD THE CURRENT BATCH RECORD.
C*
CSR 14                MOVE "        "NEXTFM
CSR 14                MOVEL"$REFRESH"NEXTFM
CSR 14                MOVE "0"       RPTAPN
CSR 14                MOVE "0"       FRZAPN
CSR 14                SETON                    50
CSR 14                EXCPT
CSR 14                SETOF                    50
CSR 14                MOVE "GETNXT"  ACTION
CSR 14                SETON                  99
CSR 14                EXCPT
CSR 14                MOVE "REREAD"   ACTION
CSR 14                EXCPT
CSR 14                SETOF                  99
C*
C* EVENT 5 - "PREVIOUS"  READ THE PREVIOUS BATCH RECORD.
C*
CSR 15                MOVE "PREV  "   ACTION
CSR 15                SETON                  99
CSR 15                EXCPT
CSR 15                SETOF                  99
CSR 15                READ TERMINAL              HO
C*
C* EVENT 6 - "NEXT"  READ THE NEXT BATCH RECORD.
C*
CSR 16                MOVE "NEXT  "   ACTION
CSR 16                SETON                  99
CSR 16                EXCPT
CSR 16                SETOF                  99
CSR 16                READ TERMINAL              HO
C*
C* EVENT 8 - "EXIT"  EXIT BROWSE AND PROGRAM.
C*
CSR 18                SETON                  17
C*
C* EVENT 7 - "COLLECT"  RETURN TO COLLECT MODE.
C*
C* IF WE ARE OFF THE BOTTOM OF THE BATCH, GO TO THE FIRST
C* NON-DELETED RECORD.  IF WE ARE OFF THE TOP END, RETURN
C* TO COLLECT MODE.
C*
CSR         BTNUMN    COMP OBTNUM            32  32
CSR 32                SETON                  17
C*
C* EITHER JUMP TO BEGINNING OF BROWSE OR RETURN TO COLLECT.
C*
CSRN17N18             GOTO BEGBRW
C*
CSR         ENDBRW    TAG
CSR 17                MOVE "RESUME"   ACTION
CSR 17                SETON                  99
CSR 17                EXCPT
CSR 17                SETOF                  99
CSR 17                READ TERMINAL              HO
CSR                   SETOF                  01
CSR                   ENDSR                      END OF BROWSE
O*
O*
O*
O* ANY ACTION, WHERE FIELD "ACTION" CONTAINS THE ACTION NAME.
O*
OTERMINALE        99
O                         ACTION     6
```

```
O*
O*
O* ACTIONS WITH SPECIAL RECORDS.  OUTPUT INDICATOR SELLECTS
O* THE PROPER RECORD.  OUTPUT INDICATOR = ACTION NUMBER.
O*
O        E        50
O                                      6 "CHGNXT"
O                           NEXTFM    21
O                           RPTAPN    22
O                           FRZAPN    23
O*
O        E        57
O                                      6 "SHODTA"
O                           DATALN    10
O                           DATA     188
O*
O        E        63
O                                      6 "PUTDTA"
O                           DATALN    10
O                           DATA     188
O*
O        E        56
O                                     '6 "CORERR"
O                           BADFLD    11
O                           MSGLEN    13
O                           ENHAN     14
O                           MESSAG    94
O*
O        E        62
O                                      6 "BADFLD"
O                           BADFLD    11
O                           MSGLEN    13
O                           ENHAN     14
O                           MESSAG    94
O*
O        E        52
O                                      6 "SHOMSG"
O                           MSGLEN     8
O                           ENHAN      9
O                           MESSAG    89
O*
O        E        55
O                                      6 "PUTMSG"
O                           MSGLEN     8
O                           ENHAN      9
O                           MESSAG    89
O*
O        E        72
O                                      6 "RDBTNU"
O                           BTNUMN    11
O*
O        E        74
O                                      6 "GETFLD"
O                           FLDNUM    11
```

```
O                       FLDLNG    15
O*
O        E        75
O                                  6 "PUTFLD"
O                       FLDLNG    15
O                       FLDATA   140
O*
O*
O* SPECIAL MESSAGE RECORDS
O*
O        E        80
O                                  6 "PUTMSG"
O                                  8 "31"
O                                  9 "J"
O                       FUNCT     11
O                                 27 " NOT DEFINED IN "
O                                 40 " MODE."
O                       MODE      34
O*
O*
O        E        81
O                                  6 "PUTMSG"
O                                  8 "24"
O                                  9 "J"
O                                 33 "REPEAT WAS NOT SET.
O*
O*
O        E        82
O                                  6 "PUTMSG"
O                                  8 "83"
O                                  9 "J"
O                                 18 "RPG ENTRY"
O                                 30 " A.01.01    "
O                                 53 "BATCH RECORD #"
O                       BTNUMA    58
O                                 86 "MODE:        "
O                       MODENH    84
O                       MODE      92
O*
O*
O*
O*
```

```
C***************************************************************C
C                                                               C
C                ENTRY--V/3000 Data Entry Program               C
C                                                               C
C                       FORTRAN Version                         C
C                                                               C
C                          9/1/79                               C
C                                                               C
C***************************************************************C
C
C  This program controls source data entry for any forms file.
C  It opens a forms file, based on user input; it opens a batch
C  file, also named by the user.  If all is ok, it displays the
C  head form, accepts input, edits the data, and if no errors,
C  head form, accepts input, edits the data, and if no errors,
C  writes it to the batch file.  The program continues to do this
C  until $END is reached, or until the EXIT function key has been
C  pressed.
C
C  This program also controls browsing through the data collected,
C  and supports modification of that data.
C
C  The function keys have defined meanings as follows:
C
C          f1        f2        f3         f4
C         HEAD     DELETE     PRINT     REFRESH
C
C          f5        f6        f7         f8
C         PREV      NEXT     BROWSE/     EXIT
C                            COLLECT
C
C
C***************************************************************C



$PAGE "           MAIN PROGRAM        "
C***************************************************************C
C                                                               C
C                  MAIN PROGRAM                                 C
C                                                               C
C***************************************************************C
      COMMON /COM1/ COMAREA
      COMMON /COM2/ ERRORS, BATCH, UNDERLINE
      COMMON /COM3/ VERSION
      COMMON /COM5/ LASTRECNUM
      INTEGER COMAREA (60)
      LOGICAL ERRORS, BATCH, UNDERLINE
      CHARACTER*16 VERSION
C
      CALL INIT
      CALL COLLECT
      CALL EXIT
```

```
C
      STOP
      END




$PAGE "          FORMAT STATUS LINE"
C*******************************************************************C
C                                                                   C
C                         FORMAT STATUS LINE                        C
C                                                                   C
C*******************************************************************C
      SUBROUTINE FMTSTATUSLINE
      COMMON /COM1/ COMAREA
      COMMON /COM3/ VERSION
      INTEGER COMAREA(60), COLLECTMODE
      CHARACTER*16 VERSION
      CHARACTER*11 NUMBER
      INTEGER*4 COMNUMRECS, COMRECNUM
      EQUIVALENCE (COMNUMRECS,COMAREA(43)), (COMRECNUM,COMAREA(45))
      INTEGER COMMODE
      EQUIVALENCE (COMMODE,COMAREA(5))
      CHARACTER*150 MESSAGEBUF
      CHARACTER*1 ESC
      INTEGER MSGLEN, LEN, DASCII
      DATA ESC /%33C/
      DATA COLLECTMODE /0/
C
C     BUILD STATUS MESSAGE
C
      I = 1
      MESSAGEBUF [I:7] = " ENTRY "
      I = I + 7
      MESSAGEBUF [I:16] = VERSION
      I = I + 16
      MESSAGEBUF [I:1] = ESC
      I = I + 1
      MESSAGEBUF [I:5] = "&a31C"
      I = I + 5
      MESSAGEBUF [I:14] = "Batch Record #"
      I = I + 14
      LEN = DASCII (®COMRECNUM+1J®,®10®,NUMBER)
      MESSAGEBUF [I:LEN] = NUMBER [1:LEN]
      I = I + LEN
      MESSAGEBUF [I:1] = ESC
      I = I + 1
      MESSAGEBUF [I:11] = "&a65CMode: "
      I = I + 11
C
      IF (COMMODE.NE.COLLECTMODE) GOTO 10
      MESSAGEBUF [I:7] = "Collect"
      I = I + 7
      GOTO 20
```

```
   10 MESSAGEBUF [I:1] = ESC
      I = I + 1
      MESSAGEBUF [I:9] = "&dKBrowse"
      I = I + 9
   20 CONTINUE
C
      MSGLEN = I - 1
      CALL VPUTWINDOW (COMAREA,MESSAGEBUF,MSGLEN)
      RETURN
      END




$PAGE "          ENTRYERROR"
C*****************************************************************C
C                                                                 C
C                         ENTRY ERROR                             C
C                                                                 C
C*****************************************************************C
      SUBROUTINE ENTRYERROR (N)
      COMMON /COM1/ COMAREA
      COMMON /COM2/ ERRORS, BATCH, UNDERLINE
      INTEGER COMAREA(60)
      LOGICAL ERRORS, BATCH, UNDERLINE
      CHARACTER*60 MESSAGE(10), MSG
      INTEGER MAXERRS
      DATA MAXERRS /7/
      DATA MESSAGE
     +  /" The PREV key is only defined for browse mode.",
     +    " There are no previous batch records.",
     +    " The NEXT key is not defined for a non-repeating form.",
     +    " The DELETE key is only defined for browse mode.",
     +    " There are no batch records to browse.",
     +    " No batch file was specified, so browse is not allowed.",
     +    " There are no more batch records."/
C
      IF (ERRORS) RETURN
      ERRORS = .TRUE.
      IF (N.LT.1.OR.N.GT.MAXERRS) MSG = " Undefined ENTRY error."
      IF (N.GE.1.AND.N.LE.MAXERRS) MSG = MESSAGE(N)
      MSGLEN=60
      CALL VPUTWINDOW (COMAREA, MSG, MSGLEN)
C
      RETURN
      END
```

```
$PAGE "          ERROR"
C******************************************************************C
C                                                                  C
C                           ERROR                                  C
C                                                                  C
C******************************************************************C
      SUBROUTINE ERROR
      COMMON /COM1/ COMAREA
      COMMON /COM2/ ERRORS, BATCH, UNDERLINE
      INTEGER COMAREA(60)
      INTEGER COMSTATUS
      EQUIVALENCE (COMSTATUS,COMAREA(1))
      LOGICAL ERRORS, BATCH, UNDERLINE
      CHARACTER*150 MESSAGEBUF
      INTEGER MESSAGEBUFLEN, MSGLEN
C
      IF (ERRORS) RETURN
      ERRORS = .TRUE.
      MESSAGEBUF = " "
      CALL VERRMSG (COMAREA, MESSAGEBUF, MESSAGEBUFLEN, MSGLEN)
      MSGLEN = MSGLEN + 1
      COMSTATUS = 0
      CALL VPUTWINDOW (COMAREA, MESSAGEBUF, MSGLEN)
C
      RETURN
      END


$PAGE "          PRINTMSG"
C******************************************************************C
C                                                                  C
C                         PRINTMSG                                 C
C                                                                  C
C******************************************************************C
      SUBROUTINE PRINTMSG
      COMMON /COM1/ COMAREA
      INTEGER COMAREA(60)
      INTEGER COMSTATUS
      EQUIVALENCE (COMSTATUS,COMAREA(1))
      CHARACTER*150 MESSAGEBUF
C
      INTEGER MESSAGEBUFLEN, MSGLEN
      MESSAGEBUFLEN = 150
      CALL VERRMSG (COMAREA, MESSAGEBUF, MESSAGEBUFLEN, MSGLEN)
      DISPLAY MESSAGEBUF [1:MSGLEN]
      COMSTATUS = 0
C
      RETURN
      END
```

```
$PAGE "           ENTRY INITIALIZATION PROCEDURE"
C******************************************************************C
C                                                                  C
C                              INIT                                C
C                                                                  C
C******************************************************************C
      SUBROUTINE INIT
      IMPLICIT INTEGER (A-Z)
      COMMON /COM1/ COMAREA
      COMMON /COM2/ ERRORS, BATCH, UNDERLINE
      COMMON /COM3/ VERSION
      INTEGER COMAREA(60)
      CHARACTER*16 VERSION
      LOGICAL ERRORS, BATCH, UNDERLINE
      EQUIVALENCE (COMAREA(1),COMSTATUS),
     +            (COMAREA(2),COMLANGUAGE),
     +            (COMAREA(3),COMCOMAREALEN),
     +            (COMAREA(56),COMTERMOPTIONS)
C
      CHARACTER*150 MESSAGEBUF
      LOGICAL LMESSAGEBUF (75)
      EQUIVALENCE (MESSAGEBUF,LMESSAGEBUF)
C
      CHARACTER*36 FILENAME
      LOGICAL LFILENAME (18)
      EQUIVALENCE (FILENAME,LFILENAME)
C
      CHARACTER*1 ANSWER
      LOGICAL LANSWER
      EQUIVALENCE (ANSWER,LANSWER)
C
      CHARACTER*11 TERMFILENAME
      CHARACTER*1 ESC
      DATA ESC /%33C/
      DATA NOBATCHFILE /71/, VERSIONSDIFF /70/, DIFFF /73/
      DATA ENTERKEY /0/, NEXTKEY /6/
C
C   << INITIALIZE COMAREA  >>
C
      DO 2 I = 1, 60
    2 COMAREA (I) = 0
C
      COMLANGUAGE = 2
      COMCOMAREALEN = 60
      BATCH = .TRUE.
      ERRORS = .FALSE.
C
C   << ENTRY IDENTIFICATION >>
C
      VERSION = "ENTRY(FT)A.01.01 "
      DISPLAY "HP32209A.01.01 ENTRY (FTN) HEWLETT-PACKARD CO. 1980."
C
C   << GET FORMS FILE >>
C
```

```
   10 COMSTATUS = 0
      DISPLAY " Enter Forms file name and press RETURN: "
      FILENAME = " "
      FILENAMELEN = READ (LFILENAME,®-36®)
      IF (FILENAMELEN.EQ.0) CALL TERMINATE
C
      CALL VOPENFORMF (COMAREA, FILENAME)
C
      IF (COMSTATUS.EQ.0) GOTO 20
       MESSAGEBUFLEN = 150
       CALL VERRMSG (COMAREA, MESSAGEBUF, MESSAGEBUFLEN, MSGLEN)
       DISPLAY MESSAGEBUF [1:MSGLEN]
       GOTO 10
C
C  << NOW OPEN BATCH FILE >>
C
   20 DISPLAY " Enter Batch file name and press RETURN: "
      FILENAME = " "
      FILENAMELEN = READ (LFILENAME,®-36®)
      IF (FILENAMELEN.NE.0) GOTO 30
       BATCH = .FALSE.
       GOTO 50
C
   30 CALL VOPENBATCH (COMAREA, FILENAME)
C
      IF (COMSTATUS.EQ.0) GOTO 50
       IF (COMSTATUS.NE.VERSIONSDIFF.AND.COMSTATUS.NE.DIFFF) GOTO 40
        IF (COMSTATUS.EQ.DIFFF) DISPLAY
     +   "Warning: Different Forms file used to create this batch."
        IF (COMSTATUS.EQ.VERSIONSDIFF) DISPLAY
     +   "Warning: Forms file recompiled since last run."
        DISPLAY "Enter 'Y' to continue"
        ANSWERLEN = READ (LANSWER,®-1®)
        IF (ANSWERLEN.EQ.1.AND.(ANSWER.EQ."Y".OR.ANSWER.EQ."y"))
     +   COMSTATUS = 0
        GOTO 50
   40  MESSAGEBUFLEN = 150
       CALL VERRMSG (COMAREA, MESSAGEBUF, MESSAGEBUFLEN, MSGLEN)
       DISPLAY MESSAGEBUF [1:MSGLEN]
   50 CONTINUE
C
C  << OPEN THE TERMINAL >>
C
      IF (BATCH.AND.COMSTATUS.NE.0) GOTO 70
       TERMFILENAME = "A264X "
C
       CALL VOPENTERM (COMAREA, TERMFILENAME)
C
       IF (COMSTATUS.EQ.0) GOTO 60
        MESSAGEBUFLEN = 150
        CALL VERRMSG (COMAREA, MESSAGEBUF, MESSAGEBUFLEN, MSGLEN)
        DISPLAY MESSAGEBUF [1:MSGLEN]
        CALL QUIT (6)
   60  COMTERMOPTIONS [11:2] = 1
```

```
       RETURN
C
C   << NORMAL ERROR >>
C
   70 COMSTATUS = 0
      CALL VCLOSEBATCH (COMAREA)
      CALL VCLOSEFORMF (COMAREA)
      GOTO 10
      END



$PAGE "           EXIT"
C******************************************************************C
C                                                                  C
C                             EXIT                                 C
C                                                                  C
C******************************************************************C
      SUBROUTINE EXIT
      COMMON /COM1/ COMAREA
      COMMON /COM2/ ERRORS, BATCH, UNDERLINE
      INTEGER COMAREA(60), COMSTATUS
      EQUIVALENCE (COMSTATUS,COMAREA(1))
      LOGICAL ERRORS, BATCH, UNDERLINE
C
C   << FIRST, CLOSE TERMINAL >>
C
      CALL VCLOSETERM (COMAREA)
      IF (COMSTATUS.NE.0) CALL PRINTMSG
C
C   << NOW, BATCH FILE >>
C
      IF (.NOT.BATCH.OR.ERRORS) GOTO 10
      CALL VCLOSEBATCH (COMAREA)
      IF (COMSTATUS.NE.0) CALL PRINTMSG
   10 CONTINUE
C
C   << NOW, CLOSE FORMS FILE >>
C
      CALL VCLOSEFORMF (COMAREA)
      IF (COMSTATUS.NE.0) CALL PRINTMSG
      RETURN
      END
```

```
$PAGE "      BROWSE"
C************************************************************C
C                                                            C
C                           BROWSE                           C
C                                                            C
C************************************************************C
      SUBROUTINE BROWSE
      IMPLICIT INTEGER (A-Z)
      COMMON /COM1/ COMAREA
      COMMON /COM2/ ERRORS, BATCH, UNDERLINE
      COMMON /COM5/ LASTRECNUM
      INTEGER*4 LASTRECNUM
      INTEGER COMAREA (60)
      LOGICAL COMDELETEFLAG, ERRORS, BATCH, UNDERLINE
      CHARACTER*16 COMCFNAME, COMNFNAME
      EQUIVALENCE (COMAREA(1),COMSTATUS),
     +            (COMAREA(2),COMLANGUAGE),
     +            (COMAREA(3),COMCOMAREALEN),
     +            (COMAREA(5),COMMODE),
     +            (COMAREA(6),COMLASTKEY),
     +            (COMAREA(7),COMNUMERRS),
     +            (COMAREA(11),COMCFNAME),
     +            (COMAREA(19),COMNFNAME),
     +            (COMAREA(27),COMREPEATOPT),
     +            (COMAREA(28),COMNFOPT),
     +            (COMAREA(30),COMDBUFLEN),
     +            (COMAREA(33),COMDELETEFLAG),
     +            (COMAREA(34),COMSHOWCONTROL)
      INTEGER*4 COMNUMRECS, COMRECNUM, LOCALCOMREC
      EQUIVALENCE (COMAREA(43),COMNUMRECS),
     +            (COMAREA(45),COMRECNUM)
C
C  << MISCELLANEOUS VALUES >>
C
      DATA COMAREALEN /60/,
     +     FTNLANG /4/,
     +     COLLECTMODE /0/,
     +     BROWSEMODE /1/,
     +     MAXWINDOWLEN /150/,
     +     NAMELEN /15/,
     +     NORM /0/,
     +     NOREPEAT /0/,
     +     REPEAT /1/,
     +     REPEATAPP /2/,
     +     FORWARDS /1/,
     +     BACKWARDS /-1/,
     +     PAGEEJECT /%61/
C
C  << FUNCTION KEY ASSIGNMENTS >>
C
      DATA ENTERKEY /0/,
     +     HEADKEY /1/,
     +     DELETEKEY /2/,
     +     PRINTKEY /3/,
     +     REFRESHKEY /4/,
     +     PREVKEY /5/,
     +     NEXTKEY /6/,
     +     BROWSEKEY /7/,
     +     EXITKEY /8/
C
C   << ENTRY ERROR EQUATES >>
C
      DATA PREVNOTALLOWED /1/,
     +     NOPREVRECS /2/,
     +     NOTREPEATING /3/,
     +     DELETENOTDEFNED /4/,
     +     NOBATCHRECS /5/,
     +     NOBATCH /6/,
     +     NONEXTRECS /7/
C
C   << BROWSE DEFINITIONS >>
C
      UNDERLINE = .TRUE.
      LOCALCOMREC = COMRECNUM
C
C   << BROWSE IT >>
C
      COMRECNUM = COMRECNUM - 1
      DIRECTION = BACKWARDS
   10 IF (COMNUMRECS.EQ.0J) RETURN
      IF (COMRECNUM.NE.LASTRECNUM) GOTO 20
       CALL ENTRYERROR (NONEXTRECS)
       COMRECNUM = COMRECNUM - 1
       DIRECTION = BACKWARDS
   20 CONTINUE
      IF (COMRECNUM.GE.0J) GOTO 30
       CALL ENTRYERROR (NOPREVRECS)
       COMRECNUM = 0
       DIRECTION = FORWARDS
   30 CONTINUE
      CALL VREADBATCH (COMAREA)
      IF (COMSTATUS.NE.0) CALL ERROR
      IF (.NOT.COMDELETEFLAG) GO TO 60
C
C   << RECORD DELETED >>
C
       IF (DIRECTION.EQ.BACKWARDS) GOTO 50
        COMRECNUM = COMRECNUM + 1
        GO TO 10
   50   COMRECNUM = COMRECNUM - 1
        GO TO 10
C
C   << RECORD NOT DELETED >>
C
   60 CONTINUE
      IF (COMRECNUM.EQ.LOCALCOMREC.AND.COMLASTKEY.NE.REFRESHKEY)
     +    GOTO 90
       IF (DIRECTION.NE.BACKWARDS.AND.COMLASTKEY.NE.REFRESHKEY)
     +      GOTO 65
          COMREPEATOPT = NORM
          COMNFOPT = NORM
          GOTO 70
 65       IF (COMCFNAME[1:15].EQ.COMNFNAME[1:15]) GOTO 70
           COMREPEATOPT=NORM
 70      CONTINUE
         IF (COMLASTKEY.EQ.REFRESHKEY) COMNFNAME="$REFRESH"
         CALL VGETNEXTFORM (COMAREA)
         IF (COMSTATUS.NE.0) CALL ERROR
         LOCALCOMREC = COMRECNUM
 90     CONTINUE
 98     ERRORS = .FALSE.
        CALL VSHOWFORM (COMAREA)
        IF (COMSTATUS.NE.0) CALL ERROR
        COMSHOWCONTROL = 0
C
        CALL VREADFIELDS (COMAREA)
C
        IF (COMSTATUS.NE.0) CALL ERROR
        IF (ERRORS) GOTO 1000
        CASE = COMLASTKEY + 1
        GOTO (100,200,300,400,500,600,700,800,900), CASE
C
C   << ENTER KEY PRESSED >>
C
 100    DIRECTION = FORWARDS
        CALL VFIELDEDITS (COMAREA)
        IF (COMSTATUS.NE.0.OR.COMNUMERRS.NE.0) CALL ERROR
        IF (ERRORS) GOTO 140
         CALL VFINISHFORM (COMAREA)
         IF (COMSTATUS.NE.0.OR.COMNUMERRS.NE.0) CALL ERROR
         IF (COMREPEATOPT.NE.NOREPEAT.OR.COMNFOPT.EQ.NORM
     +      .AND.COMREPEATOPT.NE.REPEATAPP) GOTO 120
          CALL VSHOWFORM (COMAREA)
          IF (COMSTATUS.NE.0) CALL ERROR
 120     CONTINUE
         IF (ERRORS) GOTO 130
          CALL VWRITEBATCH (COMAREA)
          IF (COMSTATUS.NE.0) CALL ERROR
          IF (.NOT.ERRORS) COMRECNUM = COMRECNUM + 1
 130     CONTINUE
 140    CONTINUE
        GOTO 1000
C
C   << HEAD KEY >>
C
 200    DIRECTION = FORWARDS
        COMRECNUM = 0
        COMREPEATOPT = NORM
        COMNFOPT = NORM
        GOTO 1000
C
C   << DELETE KEY >>
```

```
C
  300 DIRECTION = FORWARDS
      COMDELETEFLAG = .TRUE.
      CALL VWRITEBATCH (COMAREA)
      IF (COMSTATUS.NE.0) CALL ERROR
      COMDELETEFLAG = .FALSE.
      IF (.NOT.ERRORS) COMRECNUM = COMRECNUM + 1
      COMPEATOPT = NORM
      COMNFOPT = NORM
      GOTO 1000
C
C  << PRINT KEY >>
C
  400 CALL VPRINTFORM (COMAREA, UNDERLINE, PAGEEJECT)
      IF (COMSTATUS.NE.0) CALL ERROR
      GOTO 1000
C
C  << REFRESH KEY >>
C
  500 GOTO 100Q
C
C  << PREV KEY >>
C
  600 DIRECTION = BACKWARDS
      COMRECNUM = COMRECNUM - 1
      GOTO 1000
C
C  << NEXT KEY >>
C
  700 DIRECTION = FORWARDS
      COMRECNUM = COMRECNUM + 1
      IF (COMREPEATOPT.NE.NOREPEAT.OR.COMNFOPT.EQ.NORM
     +    .AND.COMREPEATOPT.NE.REPEATAPP) GOTO 720
       CALL VSHOWFORM (COMAREA)
       IF (COMSTATUS.NE.0) CALL ERROR
  720 CONTINUE
      GOTO 1000
C
C  << COLLECT KEY >>
C
  800 RETURN
C
C  << EXIT KEY >>
C
  900 RETURN
C
C
C
 1000 IF (.NOT.ERRORS.AND.COMLASTKEY.NE.PRINTKEY) GOTO 10
      GOTO 98
      END
```

```
$PAGE "           COLLECT           "
C***************************************************************C
C                                                               C
C                            COLLECT                            C
C                                                               C
C***************************************************************C
      SUBROUTINE COLLECT
      IMPLICIT INTEGER (A-Z)
      COMMON /COM1/ COMAREA
      COMMON /COM2/ ERRORS, BATCH, UNDERLINE
      COMMON /COM5/ LASTRECNUM
      INTEGER*4 LASTRECNUM
      INTEGER COMAREA (60)
      LOGICAL COMDELETEFLAG, ERRORS, BATCH, UNDERLINE
      CHARACTER*16 SAVEDFORMNAME
      LOGICAL FIRSTTIME
      CHARACTER*16 COMCFNAME, COMNFNAME
      EQUIVALENCE (COMAREA(1),COMSTATUS),
     +            (COMAREA(2),COMLANGUAGE),
     +            (COMAREA(3),COMCOMAREALEN),
     +            (COMAREA(5),COMMODE),
     +            (COMAREA(6),COMLASTKEY),
     +            (COMAREA(7),COMNUMERRS),
     +            (COMAREA(11),COMCFNAME),
     +            (COMAREA(19),COMNFNAME),
     +            (COMAREA(27),COMREPEATOPT),
     +            (COMAREA(28),COMNFOPT),
     +            (COMAREA(30),COMDBUFLEN),
     +            (COMAREA(33),COMDELETEFLAG),
     +            (COMAREA(34),COMSHOWCONTROL)
      INTEGER*4 COMNUMRECS, COMRECNUM, LOCALCOMREC
      EQUIVALENCE (COMAREA(43),COMNUMRECS),
     +            (COMAREA(45),COMRECNUM)
      CHARACTER*150 MESSAGEBUF
      INTEGER POSTTEST
      DATA PARMVAL /20/
C
C   << MISCELLANEOUS VALUES >>
C
      DATA COMAREALEN /60/,
     +     FTNLANG /4/,
     +     COLLECTMODE /0/,
     +     BROWSEMODE /1/,
     +     MAXWINDOWLEN /150/,
     +     NAMELEN /15/,
     +     NORM /0/,
     +     NOREPEAT /0/,
     +     REPEAT /1/,
     +     REPEATAPP /2/,
     +     FORWARDS /1/,
     +     BACKWARDS /-1/,
     +     PARMVAL /20/,
     +     PAGEEJECT /%61/
C
```

```
C   << FUNCTION KEY ASSIGNMENTS >>
C
          DATA ENTERKEY /0/,
         +     HEADKEY /1/,
         +     DELETEKEY /2/,
         +     PRINTKEY /3/,
         +     REFRESHKEY /4/,
         +     PREVKEY /5/,
         +     NEXTKEY /6/,
         +     BROWSEKEY /7/,
         +     EXITKEY /8/
C
C   << ENTRY ERROR EQUATES >>
C
          DATA PREVNOTALLOWED /1/,
         +     NOPREVRECS /2/,
         +     NOTREPEATING /3/,
         +     DELETENOTDEFNED /4/,
         +     NOBATCHRECS /5/,
         +     NOBATCH /6/,
         +     NONEXTRECS /7/
C
C   << COLLECT DEFINITIONS >>
C
          DATA FIRSTTIME /.TRUE./
C
C   << COLLECT IT >>
C
          COMMODE = COLLECTMODE
          COMDELETEFLAG = .FALSE.
    10    IF (COMLASTKEY.NE.ENTERKEY.AND.COMLASTKEY.NE.NEXTKEY) GOTO 30
            IF (COMREPEATOPT.NE.NOREPEAT.OR.COMNFOPT.EQ.NORM
         +         .AND.COMREPEATOPT.NE.REPEATAPP) GOTO 30
              CALL VSHOWFORM (COMAREA)
              IF (COMSTATUS.NE.0) CALL ERROR
    30    CONTINUE
          CALL VGETNEXTFORM (COMAREA)
          IF (.NOT.FIRSTTIME.OR.COMSTATUS.EQ.0) GOTO 50
           CALL VERRMSG (COMAREA, MESSAGEBUF, MESSAGEBUFLEN, MSGLEN)
           ERRORS = .TRUE.
           RETURN
    50    CONTINUE
          IF (COMSTATUS.NE.0) CALL ERROR
          FIRSTTIME = .FALSE.
          CALL VINITFORM (COMAREA)
          IF (COMSTATUS.NE.0.OR.COMNUMERRS.NE.0) CALL ERROR
          IF (.NOT.ERRORS) CALL FMTSTATUSLINE
    60    ERRORS = .FALSE.
          CALL VSHOWFORM (COMAREA)
          IF (COMSTATUS.NE.0) CALL ERROR
          COMSHOWCONTROL = 0
          IF (COMDBUFLEN.GT.0.OR.COMREPEATOPT.NE.NOREPEAT
         +    .OR.COMNFOPT.EQ.NORM) GOTO 70
           IF (ERRORS.OR..NOT.BATCH) GOTO 1000
         CALL VWRITEBATCH (COMAREA)
         IF (COMSTATUS.NE.0) CALL ERROR
         IF (ERRORS) GOTO 1000
         COMRECNUM = COMRECNUM + 1
         POSTTEST = JMOD(COMRECNUM , PARMVAL)
         IF (POSTTEST.EQ.0)
      +  CALL VPOSTBATCH (COMAREA)
         GOTO 1000
C
   70 CONTINUE
      CALL VREADFIELDS (COMAREA)
C
      IF (COMSTATUS.NE.0) CALL ERROR
      IF (ERRORS) GOTO 1000
      CASE = COMLASTKEY + 1
      GOTO (100,200,300,400,500,600,700,800,900),CASE
C
C  << ENTER KEY >>
C
  100 CALL VFIELDEDITS (COMAREA)
      IF (COMSTATUS.NE.0.OR.COMNUMERRS.NE.0) CALL ERROR
      IF (ERRORS) GOTO 150
       CALL VFINISHFORM (COMAREA)
       IF (COMSTATUS.NE.0.OR.COMNUMERRS.NE.0) CALL ERROR
       IF (ERRORS.OR..NOT.BATCH) GOTO 120
        CALL VWRITEBATCH (COMAREA)
        IF (COMSTATUS.NE.0) CALL ERROR
        IF (ERRORS) GOTO 120
        COMRECNUM = COMRECNUM + 1
        POSTTEST = JMOD(COMRECNUM , PARMVAL)
        IF (POSTTEST.EQ.0)
     +  CALL VPOSTBATCH (COMAREA)
  120  CONTINUE
  150 CONTINUE
      GOTO 1000
C
C  << HEAD KEY >>
C
  200 COMREPEATOPT = NORM
      COMNFOPT = NORM
      COMNFNAME = "$HEAD "
      GOTO 1000
C
C  << DELETE KEY >>
C
  300 CALL ENTRYERROR (DELETENOTDEFNED)
      GOTO 1000
C
C  << PRINT KEY >>
C
  400 CALL VPRINTFORM (COMAREA, UNDERLINE, PAGEEJECT)
      IF (COMSTATUS.NE.0) CALL ERROR
      GOTO 1000
C
```

```
C   << REFRESH KEY >>
C
  500 COMNFNAME = "$REFRESH "
      GOTO 1000
C
C   << PREV KEY >>
C
  600 CALL ENTRYERROR (PREVNOTALLOWED)
      GOTO 1000
C
C   << NEXTKEY >>
C
  700 IF (COMREPEATOPT.NE.NORM) GOTO 710
       CALL ENTRYERROR (NOTREPEATING)
       GOTO 720
  710 COMREPEATOPT = NORM
  720 CONTINUE
      GOTO 1000
C
C   << BROWSE KEY >>
C
  800 IF (BATCH) GOTO 810
       CALL ENTRYERROR (NOBATCH)
       GOTO 830
  810 IF (COMNUMRECS.NE.0J) GOTO 820
       CALL ENTRYERROR (NOBATCHRECS)
       GOTO 830
  820 LASTRECNUM = COMRECNUM
      SAVEDFORMNAME = COMCFNAME
      COMMODE = BROWSEMODE
      COMREPEATOPT = NORM
      COMNFOPT = NORM
      CALL BROWSE
      COMMODE = COLLECTMODE
      COMNFNAME = SAVEDFORMNAME
      COMRECNUM = LASTRECNUM
      COMREPEATOPT = NORM
      COMNFOPT = NORM
      COMDELETEFLAG = .FALSE.
      IF (COMLASTKEY.NE.EXITKEY) GOTO 830
      COMCFNAME = SAVEDFORMNAME
      RETURN
  830 CONTINUE
      GOTO 1000
C
C   << EXIT KEY >>
C
  900 RETURN
```

```
1000 CONTINUE
2000 IF (.NOT.ERRORS.AND.COMLASTKEY.NE.PRINTKEY) GOTO 3000
     GOTO 60
3000 IF (COMNFNAME[1:15].EQ."$END            "
    +     .AND.COMREPEATOPT.EQ.NORM) RETURN
     GOTO 10
     END
```

```
$PAGE "HP32209A.01.02 V/3000 S40S209A, ENTRY"
$COPYRIGHT "                                                                ",&
$ "                                                                     ",&
$ "       (C)    COPYRIGHT HEWLETT-PACKARD.    1980                     ",&
$ "This program may be used with one computer system at a time          ",&
$ "and shall not otherwise be recorded,  transmitted or stored          ",&
$ "in a retrieval system. Copying or other reproduction of this         ",&
$ "program except for archival purposes is prohibited without           ",&
$ "the prior written consent of the Hewlett-Packard Company.            "
 <<                                                                    >>
$CONTROL USLINIT, LIST, MAP, CODE
 <<*******************************************************************>>
 <<                                                                   >>
 <<                  ENTRY--V/3000 Data Entry Program                   >>
 <<                                                                   >>
 <<                             9/1/79                                >>
 <<                                                                   >>
 <<*******************************************************************>>
 <<
    This program controls source data entry for any forms file.
    It opens a forms file, based on user input; it opens a batch
    file, also named by the user.  If all is ok, it displays the
    head form, accepts input, edits the data, and if no errors,
    writes it to the batch file.  The program continues to do this
    until $END is reached, or until the EXIT function key has been
    pressed.

    This program also controls browsing through the data collected,
    and supports modification of that data.

    The function keys have defined meanings as follows:

            f1         f2         f3         f4
           HEAD      DELETE      PRINT     REFRESH

            f5         f6         f7         f8
           PREV       NEXT      BROWSE/     EXIT
                                COLLECT

 >>
 $PAGE "            ENTRY DECLARATIONS"
 <<*******************************************************************>>
 <<                                                                   >>
 <<                    ENTRY Global Declarations                      >>
 <<                                                                   >>
 <<*******************************************************************>>
 BEGIN
 DEFINE
     VERSION                  = "A.02.00" #
    ,ID'MSG=("HP32209",VERSION," ENTRY (C) HEWLETT-PACKARD CO. 1980")#
    ;
```

```
DEFINE
    COM'STATUS           = COMAREA (0) #
   ,COM'LANGUAGE         = COMAREA (1) #
   ,COM'COMAREALEN       = COMAREA (2) #
   ,COM'MODE             = COMAREA (4) #
   ,COM'LASTKEY          = COMAREA (5) #
   ,COM'NUMERRS          = COMAREA (6) #
   ,COM'LABEL'OPTION     = COMAREA (9) #
   ,COM'CFNAME           = COMAREA'B (10*2) #
   ,COM'NFNAME           = COMAREA'B (18*2) #
   ,COM'REPEATOPT        = COMAREA (26) #
   ,COM'NFOPT            = COMAREA (27) #
   ,COM'DBUFLEN          = COMAREA (29) #
   ,COM'DELETEFLAG       = COMAREA (32) #
   ,COM'SHOWCONTROL      = COMAREA (33) #
   ,COM'NUMRECS          = COMAREA'D (21) #
   ,COM'RECNUM           = COMAREA'D (22) #
   ,COM'TERMFILENUM      = COMAREA (48) #
   ,COM'TERMOPTIONS      = COMAREA (55) #
   ,com'term'type        = comarea (58) #
   ,com'keyboard'type    = comarea (74) #
   ,com'form'stor'size   = comarea (38) #
   ;
DEFINE
    CHECK'ERROR = IF COM'STATUS <> 0 THEN
                     ERROR #
   ,CHECK'EDIT'ERROR = IF COM'STATUS <> 0 OR COM'NUMERRS <> 0 THEN
                          ERROR #
   ;
EQUATE   << MISCELLANEOUS VALUES >>
    COMAREALEN     = 85
   ,SPL'LANG       = 3
   ,COLLECT'MODE   = 0
   ,BROWSE'MODE    = 1
   ,MAXWINDOWLEN   = 150
   ,NAMELEN        = 15
   ,NORM           = 0
   ,NOREPEAT       = 0
   ,REPEAT         = 1
   ,REPEATAPP      = 2
   ,ESC            = 27
   ,FORWARDS       = 1
   ,BACKWARDS      = -1
   ;
EQUATE   << FUNCTION KEY ASSIGNMENTS >>
    ENTERKEY       = 0
   ,HEADKEY        = 1
   ,DELETEKEY      = 2
   ,PRINTKEY       = 3
   ,REFRESHKEY     = 4
   ,PREVKEY        = 5
   ,NEXTKEY        = 6
   ,BROWSEKEY      = 7
   ,EXITKEY        = 8
   ;
```

```
EQUATE   << ENTRY ERROR EQUATES >>
    PREV'NOT'ALLOWED        = 1
   ,NO'PREV'RECS            = 2
   ,NOT'REPEATING           = 3
   ,DELETE'NOT'DEFINED      = 4
   ,NO'BATCH'RECS           = 5
   ,NO'BATCH                = 6
   ,NO'NEXT'RECS            = 7
   ;
INTEGER ARRAY
    COMAREA (0:COMAREALEN-1) := COMAREALEN (0)
   ;
BYTE ARRAY
    COMAREA'B (*) = COMAREA
   ;
DOUBLE ARRAY
    COMAREA'D (*) = COMAREA
   ;
LOGICAL
    ERRORS := FALSE
   ,BATCH
   ,UNDERLINE := TRUE
   ;
ARRAY
    MESSAGE'WBUF (0:MAXWINDOWLEN/2)
   ;
BYTE ARRAY
    MESSAGE'BUF (*) = MESSAGE'WBUF
   ;
INTEGER
   PARMVAL := 20
   ,MESSAGE'BUF'LEN := MAXWINDOWLEN
   ,MSGLEN
   ,PAGE'EJECT := %61
   ;
DOUBLE
    LAST'REC'NUM
   ;
$PAGE "            V/3000 INTRINSIC DECLARATIONS"
<<*************************************************************>>
<<                                                             >>
<<                     V/3000 INTRINSICS                         >>
<<                                                             >>
<<*************************************************************>>
INTRINSIC
    VCLOSEBATCH
   ,VCLOSEFORMF
   ,VCLOSETERM
   ,VERRMSG
   ,VFIELDEDITS
   ,VFINISHFORM
   ,VGETNEXTFORM
   ,VINITFORM
   ,VOPENBATCH
   ,VOPENFORMF
```

```
        ,VOPENTERM
        ,VPOSTBATCH
        ,VPRINTFORM
        ,VPUTWINDOW
        ,VREADBATCH
        ,VREADFIELDS
        ,VSHOWFORM
        ,VWRITEBATCH
        ,VGETKEYLABELS
        ,VSETKEYLABELS
        ,VSETKEYLABEL
        ;

<<*******************************************************************>>
<<                                                                   >>
<<                     DO'COLLECT'LABELS                            >>
<<                                                                   >>
<<*******************************************************************>>
PROCEDURE DO'COLLECT'LABELS;

  BEGIN

    BYTE ARRAY LABELS(0:127);

    INTEGER NUMBER'OF'LABELS,GLOB'FORM;

    MOVE LABELS := (
        << FUNCTION KEY 1 >>      "  HEAD    FORM  "
        << FUNCTION KEY 2 >>     ,"                "
        << FUNCTION KEY 3 >>     ," PRINT          "
        << FUNCTION KEY 4 >>     ,"REFRESH         "
        << FUNCTION KEY 5 >>     ,"                "
        << FUNCTION KEY 6 >>     ,"  NEXT    FORM  "
        << FUNCTION KEY 7 >>     ," BROWSE         "
        << FUNCTION KEY 8 >>     ,"  EXIT          "
                   );

    GLOB'FORM := 0;  << GLOBAL LABELS >>

    NUMBER'OF'LABELS := 8;

    VSETKEYLABELS(COMAREA,GLOB'FORM,NUMBER'OF'LABELS,LABELS);

  END;

<<*******************************************************************>>
<<                                                                   >>
<<                     DO'BROWSE'LABELS                              >>
<<                                                                   >>
<<*******************************************************************>>
PROCEDURE DO'BROWSE'LABELS;
```

```
SPL


  BEGIN

    BYTE ARRAY LABELS(0:127);

    INTEGER NUMBER'OF'LABELS,GLOB'FORM;

    MOVE LABELS := (
        << FUNCTION KEY 1 >>      " FIRST    REC    "
        << FUNCTION KEY 2 >>     ," DELETE   REC    "
        << FUNCTION KEY 3 >>     ," PRINT           "
        << FUNCTION KEY 4 >>     ,"REFRESH          "
        << FUNCTION KEY 5 >>     ,"  PREV    REC    "
        << FUNCTION KEY 6 >>     ,"  NEXT    REC    "
        << FUNCTION KEY 7 >>     ," COLLECT         "
        << FUNCTION KEY 8 >>     ,"  EXIT           "
                   );

    GLOB'FORM := 0;  << GLOBAL LABELS >>

    NUMBER'OF'LABELS := 8;

    VSETKEYLABELS(COMAREA,GLOB'FORM,NUMBER'OF'LABELS,LABELS);

  END;

$PAGE "          FORMAT'STATUS'LINE"
<<*****************************************************************>>
<<                                                                 >>
<<                       FORMAT'STATUS'LINE                        >>
<<                                                                 >>
<<*****************************************************************>>
PROCEDURE FORMAT'STATUS'LINE;
   BEGIN

   INTEGER CNT;

   INTRINSIC ASCII, DASCII;

   if com'term'type = 15 or  << HP3075 >>
      com'term'type = 16 then << hp3076 >>
     move message'buf := (" ENTRY ", version, " "), 2
   else
     MOVE MESSAGE'BUF := (" ENTRY ", VERSION, ESC, "&a31C"), 2;
   MSGLEN := TOS - @MESSAGE'BUF;

   MOVE MESSAGE'BUF(MSGLEN) := "Batch Record #", 2;
   MSGLEN := TOS - @MESSAGE'BUF;
   MSGLEN := MSGLEN + DASCII (COM'RECNUM+1D, 10, MESSAGE'BUF (MSGLEN));

   if com'term'type = 15 or << hp3075 >>
      com'term'type = 16 then << hp3076 >>
     move message'buf(msglen) := (" Mode: "), 2
   else
     MOVE MESSAGE'BUF (MSGLEN) := (ESC, "&a65CMode: "), 2;
   MSGLEN := TOS - @MESSAGE'BUF;
   IF COM'MODE = COLLECT'MODE THEN
      MOVE MESSAGE'BUF (MSGLEN) := "Collect", 2
   ELSE
     if com'term'type = 15 or << hp3075>>
        com'term'type = 16 then << hp3076 >>
       move message'buf(msglen):= ("Browse"), 2
     else
       MOVE MESSAGE'BUF (MSGLEN) := (ESC, "&dKBrowse"), 2;

   MSGLEN := TOS - @MESSAGE'BUF;

   VPUTWINDOW (COMAREA, MESSAGE'BUF, MSGLEN);
   END;   << FORMAT'STATUS'LINE >>
$PAGE "          ENTRY'ERROR"
<<*************************************************************>>
<<                                                             >>
<<                         ENTRY'ERROR                         >>
<<                                                             >>
<<*************************************************************>>
PROCEDURE ENTRY'ERROR (ENTRY'ERROR'NUM);
VALUE ENTRY'ERROR'NUM;
INTEGER ENTRY'ERROR'NUM;
   BEGIN

   IF ERRORS THEN
      RETURN;

   ERRORS := TRUE;

   CASE ENTRY'ERROR'NUM OF
      BEGIN

      << 0 IS NOT DEFINED >>
         ;

      << PREV'NOT'DEFINED:     >>
         MOVE MESSAGE'BUF :=
            " The PREV key is only defined for browse mode.", 2;

      << NO'PREV'RECS:         >>
         MOVE MESSAGE'BUF :=
            " There are no previous batch records.", 2;

      << NOT'REPEATING:        >>
         MOVE MESSAGE'BUF :=
            " The NEXT key is not defined for a non-repeating form.", 2;

      << DELETE'NOT'DEFINED:   >>
         MOVE MESSAGE'BUF :=
            " The DELETE key is only defined for browse mode.", 2;
      << NO'BATCH'RECS:          >>
         MOVE MESSAGE'BUF :=
            " There are no batch records to browse.", 2;

      << NO'BATCH:               >>
         MOVE MESSAGE'BUF :=
            " No batch file was specified, so browse is not allowed.",2;

      << NO'NEXT'REC    >>
      MOVE MESSAGE'BUF :=
         " There are no more batch records.", 2;

      END;

   MSGLEN := TOS - @MESSAGE'BUF;

   VPUTWINDOW (COMAREA, MESSAGE'BUF, MSGLEN);

   END;   << ENTRY'ERROR >>
$PAGE "          ERROR"
<<*****************************************************************>>
<<                                                                 >>
<<                              ERROR                              >>
<<                                                                 >>
<<*****************************************************************>>
PROCEDURE ERROR;
   BEGIN

   IF ERRORS THEN       << WILL ONLY HANDLE FIRST ERROR! >>
      RETURN;

   ERRORS := TRUE;

   MESSAGE'BUF := " ";
   VERRMSG (COMAREA, MESSAGE'BUF(1), MESSAGE'BUF'LEN, MSGLEN);
   MSGLEN := MSGLEN + 1;

   COM'STATUS := 0;
   VPUTWINDOW (COMAREA, MESSAGE'BUF, MSGLEN);

   END;  << ERROR >>
$PAGE "          ENTRY INITIALIZATION PROCEDURE"
<<*****************************************************************>>
<<                                                                 >>
<<                                INIT                             >>
<<                                                                 >>
<<*****************************************************************>>
PROCEDURE INIT;
   BEGIN

   EQUATE
       VERSIONS'DIFF = 70
      ,DIF'FF        = 73
      ,FILENAMELEN   = 36
      ;
```

```
EQUATE
    GET'FF'NAME    = 0
   ,GET'BF'NAME    = 1
   ,DIF'FF'WARN    = 2
   ,VERS'DIF'WARN  = 3
   ,Y'TO'CONT      = 4
   ,PRODUCT'ID     = 5
   ;
INTEGER
    INDEX
   ,READ'LEN
   ;
BYTE ARRAY
    FILENAME (0:FILENAMELEN)
   ;
INTRINSIC
    TERMINATE
   ,QUIT
   ,PRINT
   ,READ
   ;

SUBROUTINE HANDLE'PROMPT'ERR (QUIT'NUM);
VALUE QUIT'NUM;
INTEGER QUIT'NUM;
   BEGIN
   MOVE MESSAGE'BUF := "Terminal access failed unexpectedly.", 2;
   MSGLEN := TOS - @MESSAGE'BUF;
   PRINT (MESSAGE'WBUF, -MSGLEN, 0);
   QUIT (QUIT'NUM);
   END;    << HANDLE'PROMPT'ERR >>

SUBROUTINE WRITE'MSG;
   BEGIN
   VERRMSG (COMAREA, MESSAGE'BUF, MESSAGE'BUF'LEN, MSGLEN);
   PRINT (MESSAGE'WBUF, -(MSGLEN), %60);
   IF <> THEN   << CANT WRITE TO PROMPT FILE! >>
      HANDLE'PROMPT'ERR (%60);
   END;  << WRITE'MSG >>

SUBROUTINE PRINT'TO'TERM (MSG'NUM, CCTL);
VALUE MSG'NUM, CCTL;
INTEGER MSG'NUM;
LOGICAL CCTL;
   BEGIN

   CASE MSG'NUM OF
      BEGIN

      << 0, FF'NAME'PROMPT >>
      MOVE MESSAGE'BUF:=" Enter Forms File name and press RETURN: ",
                            2;
```

```
         << 1, BF NAME PROMPT >>
         MOVE MESSAGE'BUF:=" Enter Batch File name and press RETURN: ",
                              2;

         << 2, DIFFERENT FF WARNING >>
         MOVE MESSAGE'BUF:=("WARNING: A different forms file was used ",
                              "to create this batch."),2;

         << 3, FF MOD WARN >>
         MOVE MESSAGE'BUF:=("WARNING: Forms File was recompiled since ",
                              "this batch was created."), 2;

         << 4, Y'TO'CONTINUE >>
         MOVE MESSAGE'BUF := (" Enter ""Y"" to continue: "), 2;

         << 5, PRODUCT'ID >>
         MOVE MESSAGE'BUF := ID'MSG, 2;

         END;   << CASE >>

      MSGLEN := TOS - @MESSAGE'BUF;
      PRINT (MESSAGE'WBUF, -MSGLEN, CCTL);
      IF <> THEN
         HANDLE'PROMPT'ERR (2);

      END;   << PRINT'TO'TERM >>
   INTEGER SUBROUTINE READ'FROM'TERM (READBUF, READLEN);
   VALUE READLEN;
   BYTE ARRAY READBUF;
   INTEGER READLEN;
      BEGIN

      << BLANK BUF FIRST >>
      READBUF := " ";
      MOVE READBUF (1) := READBUF (0), (READLEN-1);

      READ'FROM'TERM := READ (READBUF, -READLEN);
      IF <> THEN
         HANDLE'PROMPT'ERR (3);
      END;   << READ'FROM'TERM >>

$PAGE
   << INITIALIZE COMAREA;  IS ALL 0'S TO START >>
   COM'LANGUAGE := SPL'LANG;
   COM'COMAREALEN := COMAREALEN;

   << SET COM'LABEL'OPTION TO 1 TO ENABLE FUNCTION KEY LABEL   >>
   << SUPPORT FOR TERMINALS SUPPORTING FUNCTION KEY LABELS     >>
   COM'LABEL'OPTION := 1;

   << Set form storage buffer size (2626 terminal only) to 4 >>
   COM'FORM'STOR'SIZE := 4;
```

```
BATCH := TRUE;  << INIT >>

PRINT'TO'TERM (PRODUCT'ID, %60);   << ENTRY IDENTIFICATION >>

WHILE TRUE DO
   BEGIN

   DO     << UNTIL COM'STATUS = 0 >>
      BEGIN
      COM'STATUS := 0;
      PRINT'TO'TERM (GET'FF'NAME, %320);
      READ'LEN := READ'FROM'TERM (FILENAME, FILENAMELEN);
      IF READ'LEN = 0 THEN   << ALL DONE >>
         TERMINATE;

      VOPENFORMF (COMAREA, FILENAME);
      IF COM'STATUS <> 0 THEN
         WRITE'MSG;   << WRITES VERRMSG >>
      END
   UNTIL COM'STATUS = 0;   << KEEP GOING TILL OK >>

   << NOW, OPEN BATCH FILE >>
   PRINT'TO'TERM (GET'BF'NAME, %320);
   READ'LEN := READ'FROM'TERM (FILENAME, FILENAMELEN);
   IF READ'LEN = 0 THEN   << NO BATCH FILE! >>
      BATCH := FALSE  << ALL OK >>
   ELSE
      BEGIN
      VOPENBATCH (COMAREA, FILENAME);
      IF COM'STATUS <> 0 THEN
         IF COM'STATUS = VERSIONS'DIFF OR
             COM'STATUS = DIF'FF THEN
             BEGIN
             PRINT'TO'TERM ((IF COM'STATUS=DIF'FF THEN DIF'FF'WARN
                                 ELSE VERS'DIF'WARN), 0);
             PRINT'TO'TERM (Y'TO'CONT, %320);
             READ'LEN := READ'FROM'TERM (MESSAGE'BUF, 1);
             IF READ'LEN > 0 THEN
                IF READ'LEN=1 AND (MESSAGE'BUF = "Y" OR
                                      MESSAGE'BUF = "y") THEN
                   COM'STATUS := 0;    << GO AHEAD >>
             END
         ELSE  << IS REAL ERROR >>
             WRITE'MSG;
      END;

   << ALL OK HERE, SO OPEN TERMINAL >>
   IF NOT BATCH OR COM'STATUS = 0 THEN
      BEGIN

      << OPEN TERMINAL IN BLOCKMODE ... >>
      MOVE FILENAME := "A264X ";
      VOPENTERM (COMAREA, FILENAME);
      IF COM'STATUS <> 0 THEN
         BEGIN
```

```
            WRITE'MSG;
            QUIT (6);
            END;
         COM'TERMOPTIONS.(11:2) := 1;    << DONT HARD RESET TERM >>
         RETURN;    << ALL DONE INITIALIZING >>
         END
      ELSE    << IS NORMAL ERROR >>
         BEGIN
         COM'STATUS := 0;
         VCLOSEBATCH (COMAREA);
         VCLOSEFORMF (COMAREA);
         END;

      END;    << WHILE TRUE >>
   END;  << INIT >>
$PAGE "          EXIT"
<<*****************************************************************>>
<<                                                                 >>
<<                           EXIT                                  >>
<<                                                                 >>
<<*****************************************************************>>
PROCEDURE EXIT;
   BEGIN

   BYTE ARRAY LOCAL'MESSAGE'BUF (0:80);
   INTEGER LOCAL'MSGLEN;

   INTRINSIC PRINT;
   SUBROUTINE PRINT'MSG;
      BEGIN
      VERRMSG (COMAREA, LOCAL'MESSAGE'BUF, MESSAGE'BUF'LEN,
               LOCAL'MSGLEN);
      PRINT (LOCAL'MESSAGE'BUF, -LOCAL'MSGLEN, 0);
      COM'STATUS := 0;
      END;

   << FIRST, CLOSE TERMINAL >>
   VCLOSETERM (COMAREA);
   IF COM'STATUS <> 0 THEN
      PRINT'MSG;

   << NOW, BATCH FILE >>
   IF BATCH THEN
      IF ERRORS THEN
         PRINT (MESSAGE'WBUF, -MSGLEN, 0)  << MSG FROM COLLECT >>
      ELSE  <<  OK TO GO AHEAD >>
         BEGIN
         VCLOSEBATCH (COMAREA);
         IF COM'STATUS <> 0 THEN
            PRINT'MSG;
         END;
   << NOW, CLOSE FORMS FILE >>
   VCLOSEFORMF (COMAREA);
   IF COM'STATUS <> 0 THEN
      PRINT'MSG;

   END;   << EXIT >>
$PAGE "          BROWSE"
<<*****************************************************************>>
<<                                                                 >>
<<                            BROWSE                               >>
<<                                                                 >>
<<*****************************************************************>>
PROCEDURE BROWSE;
   BEGIN

   EQUATE
       FORWARDS   = 1
      ,BACKWARDS = -1
      ;
   INTEGER
       PAGE'EJECT := %61
      ,DIRECTION
      ;
   LOGICAL
       UNDERLINE := TRUE
      ;
   DOUBLE
       LOCAL'COM'REC
      ;

    DO'BROWSE'LABELS;


   LOCAL'COM'REC := COM'RECNUM;
   COM'RECNUM := COM'RECNUM - 1D;
   DIRECTION := BACKWARDS;

   WHILE TRUE DO      << UNTIL EXIT OR COLLECTKEY >>
      BEGIN

      IF COM'NUMRECS = 0D THEN
         RETURN;

      IF COM'RECNUM = LAST'REC'NUM THEN
         BEGIN
         ENTRY'ERROR (NO'NEXT'RECS);
         COM'RECNUM := COM'RECNUM - 1D;
         DIRECTION := BACKWARDS;
         END;

      IF COM'RECNUM < 0D THEN
         BEGIN
         ENTRY'ERROR (NO'PREV'RECS);
         COM'RECNUM := 0D;
         DIRECTION := FORWARDS;
         END;

      VREADBATCH (COMAREA);
      CHECK'ERROR;

      IF COM'DELETEFLAG = FALSE THEN    << NOT DELETED >>
         BEGIN

         IF COM'RECNUM <> LOCAL'COM'REC OR COM'LASTKEY = REFRESHKEY THEN
            BEGIN
            IF DIRECTION = BACKWARDS OR COM'LASTKEY = REFRESHKEY THEN
               COM'REPEATOPT := COM'NFOPT := NORM
            ELSE   << MUST BE FORWARDS >>
               IF COM'CFNAME <> COM'NFNAME, (15) THEN
                  COM'REPEATOPT := NORM;   << CLEAR SINCE NOT REPT >>

            IF COM'LASTKEY = REFRESHKEY THEN
               MOVE COM'NFNAME := "$REFRESH       ";


            VGETNEXTFORM (COMAREA);
            CHECK'ERROR;

            LOCAL'COM'REC := COM'RECNUM;
            END;

         IF NOT ERRORS THEN
            FORMAT'STATUS'LINE;

         DO  << WHILE ERRORS >>
            BEGIN

            ERRORS := FALSE;

            VSHOWFORM (COMAREA);
            CHECK'ERROR;

            COM'SHOWCONTROL := 0;     << RESET JUST IN CASE >>

            VREADFIELDS (COMAREA);
            CHECK'ERROR;

            if com'lastkey <> 0 then
              if com'term'type = 15 or      << HP3075 >>
                 com'term'type = 16 then    << Hp3076 >>
                if com'keyboard'type = 1 then  << Numeric keyboard >>
                  com'lastkey := com'lastkey - 16;

            IF NOT ERRORS THEN
               CASE COM'LASTKEY OF
                  BEGIN

                  << ENTERKEY: >>
                     BEGIN
                     DIRECTION := FORWARDS;
      VFIELDEDITS (COMAREA);
      CHECK'EDIT'ERROR;

      IF NOT ERRORS THEN
         BEGIN
         VFINISHFORM (COMAREA);
         CHECK'EDIT'ERROR;

         IF COM'REPEATOPT=NOREPEAT AND COM'NFOPT <> NORM
            OR COM'REPEATOPT=REPEATAPP THEN
            BEGIN
            VSHOWFORM (COMAREA);
            CHECK'ERROR;
            END;

         IF NOT ERRORS THEN
            BEGIN
            VWRITEBATCH (COMAREA);
            CHECK'ERROR;

            IF NOT ERRORS THEN
               COM'RECNUM := COM'RECNUM+1D;
            END;

         END;

      END;

   << HEADKEY: >>
      BEGIN
      DIRECTION := FORWARDS;
      COM'RECNUM := 0D;
      COM'REPEATOPT := COM'NFOPT := NORM;
      END;

   << DELETEKEY: >>
      BEGIN
      DIRECTION := FORWARDS;

      COM'DELETEFLAG := TRUE;
      VWRITEBATCH (COMAREA);
      CHECK'ERROR;

      COM'DELETEFLAG := FALSE;
      IF NOT ERRORS THEN
         COM'RECNUM := COM'RECNUM + 1D;

      COM'REPEATOPT := COM'NFOPT := NORM;
      END;

   << PRINTKEY: >>
      BEGIN
      VPRINTFORM (COMAREA, UNDERLINE, PAGE'EJECT);
      CHECK'ERROR;
      END;
```

```
                  << REFRESHKEY: >>
                     ;

                  << PREVKEY: >>
                     BEGIN
                     DIRECTION := BACKWARDS;
                     COM'RECNUM := COM'RECNUM - 1D;
                     END;

                  << NEXTKEY: >>
                     BEGIN
                     DIRECTION := FORWARDS;
                     COM'RECNUM := COM'RECNUM + 1D;

                     IF COM'REPEATOPT=NOREPEAT AND COM'NFOPT <> NORM
                        OR COM'REPEATOPT=REPEATAPP THEN
                        BEGIN
                        VSHOWFORM (COMAREA);
                        CHECK'ERROR;
                        END;
                     END;

                  << COLLECTKEY: >>
                     RETURN;

                  << EXIT: >>
                     RETURN;

                  END;   << CASE >>

              END
            UNTIL NOT ERRORS AND COM'LASTKEY <> PRINTKEY;

            END    << IN NOT COM'DELETEFLAG >>
          ELSE     << REC WAS DELETED >>
            COM'RECNUM := IF DIRECTION = BACKWARDS THEN COM'RECNUM - 1D
                             ELSE COM'RECNUM + 1D;

          END;  << WHILE TRUE DO >>

        END;   << BROWSE >>
      $PAGE "          COLLECT"
      <<*************************************************************>>
      <<                                                             >>
      <<                        COLLECT                              >>
      <<                                                             >>
      <<*************************************************************>>
      PROCEDURE COLLECT;
        BEGIN

        LOGICAL
           FIRST'TIME := TRUE
          ;
        BYTE ARRAY
           SAVED'FORM'NAME (0:NAMELEN-1)
```

```
       ;

    DO'COLLECT'LABELS;

    COM'MODE := COLLECT'MODE;
    COM'DELETEFLAG := FALSE;

    DO   << UNTIL COM'NFNAME <> EXIT AND COM'DO <> NORM >>
       BEGIN

       IF COM'LASTKEY=ENTERKEY OR COM'LASTKEY=NEXTKEY THEN
          IF COM'REPEATOPT=NOREPEAT AND COM'NFOPT <> NORM OR
             COM'REPEATOPT=REPEATAPP THEN
             BEGIN

             VSHOWFORM (COMAREA);
             CHECK'ERROR;
             END;


       VGETNEXTFORM (COMAREA);
       IF FIRST'TIME AND COM'STATUS <> 0 THEN   << IS FIRST TIME >>
          BEGIN
          VERRMSG (COMAREA, MESSAGE'BUF, MESSAGE'BUF'LEN, MSGLEN);
          ERRORS := TRUE;   << DONT WANT TO CLOSE BATCH IF ERROR! >>
          RETURN;
          END;

       CHECK'ERROR;
       FIRST'TIME := FALSE;

       VINITFORM (COMAREA);
       CHECK'EDIT'ERROR;

       IF NOT ERRORS THEN
          FORMAT'STATUS'LINE;

       DO   << WHILE ERRORS >>
          BEGIN

          ERRORS := FALSE;

          VSHOWFORM (COMAREA);
          CHECK'ERROR;

          COM'SHOWCONTROL := 0;      << CLEAR >>

          IF COM'DBUFLEN <= 0 AND    << DONT READ!!! >>
             COM'REPEATOPT=NOREPEAT AND COM'NFOPT <> NORM THEN
             BEGIN
             IF NOT ERRORS AND BATCH THEN
                BEGIN
                VWRITEBATCH (COMAREA);
                CHECK'ERROR;
```

```
               IF NOT ERRORS THEN
                  BEGIN
                  COM'RECNUM := COM'RECNUM + 1D;
                  IF (COM'RECNUM MOD DOUBLE(PARMVAL) = 0D) THEN
                     VPOSTBATCH (COMAREA);
                  END;
               END;
            END
         ELSE    << IS NORMAL FORM >>
            BEGIN
            VREADFIELDS (COMAREA);
            CHECK'ERROR;

            if com'lastkey <> 0 then
              if com'term'type = 15 or       << HP3075 >>
                 com'term'type = 16 then     << Hp3076 >>
                if com'keyboard'type = 1 then   << Numeric keyboard >>
                  com'lastkey := com'lastkey - 16;

            IF NOT ERRORS THEN
               CASE COM'LASTKEY OF
                  BEGIN

                  << ENTERKEY: >>
                     BEGIN
                     VFIELDEDITS (COMAREA);
                     CHECK'EDIT'ERROR;

                     IF NOT ERRORS THEN
                        BEGIN
                        VFINISHFORM (COMAREA);
                        CHECK'EDIT'ERROR;

                        IF NOT ERRORS AND BATCH THEN
                           BEGIN
                           VWRITEBATCH (COMAREA);
                           CHECK'ERROR;

                           IF NOT ERRORS THEN
                  BEGIN
                  COM'RECNUM := COM'RECNUM + 1D;
                  IF (COM'RECNUM MOD DOUBLE(PARMVAL) = 0D) THEN
                     VPOSTBATCH (COMAREA);
                  END;

                           END;

                        END;

                     END;

                  << HEADKEY: >>
                     BEGIN
                     COM'REPEATOPT := NORM;
                     COM'NFOPT     := NORM;
```

```
   MOVE COM'NFNAME := "$HEAD          ";
   END;

<< DELETEKEY: >>
   ENTRY'ERROR (DELETE'NOT'DEFINED);

<< PRINTKEY: >>
   BEGIN
   VPRINTFORM (COMAREA, UNDERLINE, PAGE'EJECT);
   CHECK'ERROR;
   END;

<< REFRESHKEY: >>
   MOVE COM'NFNAME := "$REFRESH       ";

<< PREVKEY: >>
   ENTRY'ERROR (PREV'NOT'ALLOWED);

<< NEXTKEY: >>
   BEGIN
   IF COM'REPEATOPT = NORM THEN
      ENTRY'ERROR (NOT'REPEATING)
   ELSE
      COM'REPEATOPT := NORM;
   END;

<< BROWSEKEY: >>
   BEGIN
   IF NOT BATCH THEN
      ENTRY'ERROR (NO'BATCH)
   ELSE
      IF COM'NUMRECS = 0D THEN
         ENTRY'ERROR (NO'BATCH'RECS)
      ELSE
         BEGIN
         LAST'REC'NUM := COM'RECNUM;
         MOVE SAVED'FORM'NAME := COM'CFNAME,(NAMELEN);
         COM'MODE := BROWSE'MODE;
         COM'REPEATOPT := COM'NFOPT := NORM;

         BROWSE;

         COM'MODE := COLLECT'MODE;
         MOVE COM'NFNAME := SAVED'FORM'NAME,(NAMELEN);
         COM'RECNUM := LAST'REC'NUM;
         COM'REPEATOPT := COM'NFOPT := NORM;
         COM'DELETEFLAG := FALSE; << IF NO RECS >>

         IF COM'LASTKEY = EXITKEY THEN
            BEGIN
            MOVE COM'CFNAME :=
            SAVED'FORM'NAME,(NAMELEN);
            RETURN;
            END;
```

```
                           DO'COLLECT'LABELS;

                           END;

                        END;     << BROWSEKEY >>

                  << EXIT: >>
                     RETURN;

                  END;  << CASE COM'LASTKEY >>

            END;   << IS COM'DBUFLEN > 0? >>
         END
      UNTIL NOT ERRORS AND COM'LASTKEY <> PRINTKEY;

      END
   UNTIL COM'NFNAME = "$END           " AND
         COM'REPEATOPT = NORM;


   END;   << COLLECT >>
$PAGE "          ENTRY OUTER BLOCK"
<<**************************************************************>>
<<                                                              >>
<<                         OUTER BLOCK                          >>
<<                                                              >>
<<**************************************************************>>

INTRINSIC PRINT;    << FOR ID MESSAGE >>

<< FOR INTERNAL TESTING ONLY >>

INIT;

COLLECT;

EXIT;

END.
```

ENTRYBAS
```
   10 REM  ************************************************************
   15 REM  *                                                          *
   20 REM  *              ENTRY -- V/3000 Data Entry Program          *
   21 REM  *                                                          *
   22 REM  *                        BASIC Version                     *
   25 REM  *                                                          *
   30 REM  *                          9/1/79                          *
   35 REM  *                                                          *
   40 REM  ************************************************************
   45 REM
   50 REM
   55 REM
   60 REM  This program controls source data entry for any forms file.
   65 REM  It prompts the user for the name of a forms file and opens it
   70 REM  if possible.  It also prompts for the name of a BATCH file an
   75 REM  opens it if possible.  If all is OK, it displays the HEAD FOR
   80 REM  accepts input, edits the data, and if no errors, writes data
   85 REM  to the BATCH file.  This process continues until the form wit
   90 REM  name $END is encountered or the EXIT function key is pressed.
   95 REM
  100 REM  This program also provides BROWSE control and permits
  105 REM  modification of the collected data.
  110 REM
  115 REM  The function keys have the following meanings:
  120 REM
  125 REM          f1       f2        f3        f4
  130 REM         HEAD    DELETE     PRINT    REFRESH
  135 REM
  140 REM          f5       f6        f7        f8
  145 REM         PREV     NEXT     BROWSE/     EXIT
  150 REM                           COLLECT
  155 REM
  160 REM  The following variable assignments are used:
  165 REM
  170 REM  Strings:
  175 REM
  180 REM    B$[16]    - Name of current Form
  185 REM    BO$[16]   - Saves B$ during BROWSE
  190 REM    B1$[16]   - Name of next Form
  195 REM    E$[60]    - Entry error messages
  200 REM    M$[150]   - General messages
  205 REM    S$[150]   - Status line
  210 REM    U$[150]   - General user input and file names
  215 REM    V$[20]    - Version #
  220 REM
  225 REM Integers:
  230 REM
  235 REM    B1 - Batch : 0=false; 1=true
  240 REM    F1 - First time : 0=false; 1=true
  245 REM    W1 - Max window length (150)
  250 REM    R1 - Saves number of last accessed record in batch file
  255 REM    D1 - Browse direction: 0=forward; 1=backward
  260 REM    R2 - Local batch record number
```

```
265 REM
270 REM    E  - Errors Flag : 0=false; 1=true
275 REM
280 REM
285 REM Com area Array C(1:60)
290 REM
295 REM    C(1)   - Status  (0=OK; >0 ERROR)
300 REM    C(2)   - Language (1=BASIC)
305 REM    C(3)   - Com area length  (60)
310 REM    C(4)   - Com extention length  (2000)
315 REM    C(5)   - Mode
320 REM    C(6)   - Lastkey (# of last used function key)
325 REM    C(7)   - Numerrs
330 REM    C(8:10)  - not used
335 REM    C(11:18) - name of current form (packed)
340 REM    C(19:26) - name of next form (packed)
345 REM    C(27) - Repeat option
350 REM    C(28) - NF option
355 REM    C(29) - not used
360 REM    C(30) - Length of data buffer
365 REM    C(31) - not used
370 REM    C(33) - Delete flag
375 REM    C(34) - Show control
380 REM    C(35:42) - not used
385 REM    C(43:44) - Number of recs in batch file (double)*
390 REM    C(45:46) - Record # in batch file (double)*
395 REM    C(47,48) - not used
400 REM    C(49) - Terminal file #
405 REM    C(56) - Terminal options
410 REM
415 REM      *Only c(44) and c(46) are used in this program.
420 REM       Therefore, this program cannot handle BATCH files
425 REM       that have more than 32768 records.
430 REM
450 REM
451 REM  **************************************************************
452 REM
500 REM2 ** DECLARATIONS **
510 DIM B$[16],BO$[16],B1$[16],C$[2]
520 DIM E$[60],M$[150],S$[150],U$[150],V$[20],X$[2]
530 INTEGER B1,D1,F1,E,I,J,R1,R2,W1,X,Y,P,P1
540 INTEGER C[4560]
900 REM2 ** FUNCTIONS **
905 REM1 * UNPACK INTEGER INTO 2 CHARACTERS *
910 DEF FNU$(X)
915   Y=INT(X/256)
920   X=X-256*Y
925   RETURN CHR$(Y)+CHR$(X)
930 FNEND
950 REM1 * PACK 2 CHARACTERS INTO INTEGER *
955 DEF INTEGER FNP(X$)
960   RETURN NUM(X$[2;1])+NUM(X$[1;1])*256
965 FNEND
1000 REM2 ** ENTRY **
```

```
1010 REM *INITIALIZE*
1020 GOSUB 2000
1030 REM1 *COLLECT*
1040 GOSUB 3000
1050 REM1 *EXIT*
1060 GOSUB 5000
1070 END
2000 REM2 <INITIALIZE>
2010 MAT C=ZER
2020 C[2]=1
2030 C[3]=60
2040 C[4]=2000
2050 B1=1
2060 E=0
2070 W1=150
2080 V$="(BASIC)A.01.01"
2090 PRINT "ENTRY ";V$
2100 REM1 *OPEN FORMS FILE*
2110 C[1]=0
2120 PRINT " Enter FORMS file name and press RETURN: ";
2130 LINPUT U$
2140 IF NOT LEN(U$) THEN 9900
2150 U$=U$+" "
2160 CALL VOPENFORMF(C[*],U$)
2170 IF C[1] THEN DO
2180   GOSUB 9000
2190   GOTO 2100
2200 DOEND
2210 REM1 *OPEN BATCH FILE*
2220 PRINT " Enter BATCH file name and press RETURN: ";
2230 LINPUT U$
2240 IF NOT LEN(U$) THEN B1=0
2250 ELSE DO
2260   U$=U$+" "
2270   CALL VOPENBATCH(C[*],U$)
2280   IF C[1] THEN DO
2290     IF C[1]=70 OR C[1]=73 THEN DO
2300       IF C[1]=70 THEN PRINT &
    "WARNING: A different FORMS file was used to create this batch."
2310       IF C[1]=73 THEN PRINT &
    "WARNING: FORMS file was recompiled since this batch was created."
2320       PRINT "Enter "'34"Y"'34" to continue: ";
2330       LINPUT U$
2340       IF UPS$(U$)="Y" THEN C[1]=0
2350     DOEND
2360     ELSE GOSUB 9000
2370   DOEND
2380 DOEND
2390 REM2 ** OPEN TERMINAL **
2400 IF NOT B1 OR NOT C[1] THEN DO
2410   U$="A264X "
2420   CALL VOPENTERM(C[*],U$)
2430   IF C[1] THEN DO
2440     GOSUB 9000
```

```
2450      END
2460    DOEND
2470    C[56]=C[56]+8
2480    RETURN
2490 DOEND
2500 ELSE DO
2510    C[1]=0
2520    CALL VCLOSEBATCH(C[*])
2530    CALL VCLOSEFORMF(C[*])
2540 DOEND
2550 GOTO 2100
3000 REM2 <COLLECT>
3005 F1=1
3010 C[5]=C[33]=0
3015 IF NOT C[6] OR C[6]=6 THEN DO
3020    IF NOT C[27] AND NOT C[28] OR C[27]=2 THEN DO
3025      CALL VSHOWFORM(C[*])
3030      IF C[1] THEN GOSUB 9100
3035    DOEND
3040 DOEND
3045 REM1 *COLLECT LOOP*
3050 CALL VGETNEXTFORM(C[*])
3055 IF F1 AND C[1] THEN DO
3060    CALL VERRMSG(C[*],M$,B0,I)
3065    E=1
3070    RETURN
3075 DOEND
3080 IF C[1] THEN GOSUB 9100
3085 F1=0
3090 CALL VINITFORM(C[*])
3095 IF C[1] OR C[7] THEN GOSUB 9100
3100 IF NOT E THEN GOSUB 8000
3105 REM1 *SOFTKEY LOOP*
3110 E=0
3115 CALL VSHOWFORM(C[*])
3120 IF C[1] THEN GOSUB 9100
3125 C[34]=0
3130 IF C[30]<=0 AND C[27]=0 AND C[28]<>0 THEN DO
3132    IF NOT E AND B1 THEN DO
3135      CALL VWRITEBATCH(C[*])
3140      IF C[1] THEN GOSUB 9100
3145      IF NOT E THEN DO
3146        C[46]=C[46]+1
3147        P1=20
3148        P=C[46] MOD P1
3149        IF P=0 THEN CALL VPOSTBATCH(C[*])
3150      DOEND
3151    DOEND
3152 DOEND
3155 ELSE DO
3160    CALL VREADFIELDS(C[*])
3165    IF C[1] THEN GOSUB 9100
3170    IF NOT E AND C[6]=8 THEN RETURN
3175    IF NOT E THEN GOSUB C[6]+1 OF 3300,3350,3400,3450,3500,3550,&
```

```
    3600,3700
3180 DOEND
3185 IF C[6]=8 THEN RETURN
3190 IF E OR C[6]=3 THEN 3105
3195 GOSUB 8100
3200 IF B1$<>"$END" OR C[27] THEN 3045
3205 RETURN
3300 REM2 <ENTER KEY>
3305 CALL VFIELDEDITS(C[*])
3310 IF C[1] OR C[7] THEN GOSUB 9100
3315 IF NOT E THEN DO
3320   CALL VFINISHFORM(C[*])
3325   IF C[1] THEN GOSUB 9100
3330   IF NOT E AND B1 THEN DO
3335     CALL VWRITEBATCH(C[*])
3340     IF C[1] THEN GOSUB 9100
3342     IF NOT E THEN DO
3343       C[46]=C[46]+1
3344       P1=20
3345       P=C[46] MOD P1
3346       IF P=0 THEN CALL VPOSTBATCH(C[*])
3347     DOEND
3348   DOEND
3349 DOEND
3350 RETURN
3351 REM1 <HEAD KEY>
3355 C[27]=C[28]=0
3360 B1$="$HEAD"
3365 GOSUB 8200
3370 RETURN
3400 REM1 <DELETE KEY>
3410 E$="DELETE key defined only for BROWSE"
3420 GOSUB 9200
3430 RETURN
3450 REM1 <PRINT KEY>
3455 I=1
3460 J=49
3465 CALL VPRINTFORM(C[*],I,J)
3470 IF C[1] THEN GOSUB 9100
3475 RETURN
3500 REM1 <REFRESH KEY>
3505 B1$="$REFRESH"
3510 GOSUB 8200
3515 RETURN
3550 REM1 <PREV KEY>
3555 E$=" The PREV key is only defined for BROWSE mode."
3560 GOSUB 9200
3565 RETURN
3600 REM1 <NEXT KEY>
3610 IF NOT C[27] THEN DO
3620   E$=" The NEXT key is not defined for a non-repeating form."
3630   GOSUB 9200
3640 DOEND
3650 ELSE C[27]=0
```

```
3660 RETURN
3700 REM1 <BROWSE KEY>
3710 IF NOT B1 THEN DO
3720   E$=" No BATCH file was specified, so BROWSE is not allowed."
3730   GOSUB 9200
3740 DOEND
3750 ELSE DO
3760   IF NOT C[44] THEN DO
3770     E$=" There are no more batch records."
3780     GOSUB 9200
3790   DOEND
3800   ELSE DO
3810     R1=C[46]
3820     GOSUB 8100
3830     B0$=B$
3840     C[5]=1
3850     C[27]=C[28]=0
3860     GOSUB 4000
3870     C[5]=0
3880     B1$=B0$
3890     GOSUB 8200
3900     C[46]=R1
3910     C[27]=C[28]=C[33]=0
3920     IF C[6]=8 THEN DO
3921       B$=B0$
3922       RETURN
3923     DOEND
3930   DOEND
3940 DOEND
3950 RETURN
4000 REM2 <BROWSE>
4005 R2=C[46]
4010 C[46]=C[46]-1
4015 D1=1
4020 REM1 *BROWSE UNTIL EXIT OR COLLECT KEY*
4025 IF NOT C[44] THEN RETURN
4030 IF C[46]=R1 THEN DO
4035   E$=" There are no more batch records."
4040   GOSUB 9200
4045   C[46]=C[46]-1
4050   D1=1
4055 DOEND
4060 IF C[46]<0 THEN DO
4065   E$=" There are no previous batch records."
4070   GOSUB 9200
4075   D1=C[46]=0
4080 DOEND
4085 CALL VREADBATCH(C[*])
4090 IF C[1] THEN GOSUB 9100
4095 IF NOT C[33] THEN DO
4100   IF C[46]<>R2 OR C[6]=4 THEN DO
4105     IF D1 OR C[6]=4 THEN C[27]=C[28]=0
4110     ELSE DO
4115       GOSUB 8100
```

```
4120        IF B$<>B1$ THEN C[27]=0
4125      DOEND
4130      IF C[6]=4 THEN DO
4135        B1$="$REFRESH"
4140        GOSUB 8200
4145      DOEND
4150      CALL VGETNEXTFORM(C[*])
4155      IF C[1] THEN GOSUB 9100
4160      R2=C[46]
4165    DOEND
4170    IF NOT E THEN GOSUB 8000
4175    REM2 *SOFTKEY LOOP*
4180    E=0
4185    CALL VSHOWFORM(C[*])
4190    IF C[1] THEN GOSUB 9100
4195    C[34]=0
4200    CALL VREADFIELDS(C[*])
4205    IF C[1] THEN GOSUB 9100
4210    IF NOT E AND C[6]>6 THEN RETURN
4215    IF NOT E THEN GOSUB C[6]+1 OF 4300,4400,4450,4500,4550,4600,4650
4220    IF E OR C[6]=3 THEN 4175
4225  DOEND
4230  ELSE DO
4235    IF D1 THEN C[46]=C[46]-1
4240    ELSE C[46]=C[46]+1
4245  DOEND
4250  GOTO 4020
4255  RETURN
4300  REM2 <ENTER KEY>
4305  D1=0
4310  CALL VFIELDEDITS(C[*])
4315  IF C[1] OR C[7] THEN GOSUB 9100
4320  IF NOT E THEN DO
4325    CALL VFINISHFORM(C[*])
4330    IF C[1] THEN GOSUB 9100
4335    IF NOT C[27] AND C[28] OR C[27]=2 THEN DO
4340      CALL VSHOWFORM(C[*])
4345      IF C[1] THEN GOSUB 9100
4350    DOEND
4355    IF NOT E THEN DO
4360      CALL VWRITEBATCH(C[*])
4365      IF C[1] THEN GOSUB 9100
4370      IF NOT E THEN C[46]=C[46]+1
4375    DOEND
4380  DOEND
4385  RETURN
4400  REM1 <HEAD KEY>
4410  D1=C[27]=C[28]=C[46]=0
4420  RETURN
4450  REM1 <DELETE KEY>
4455  D1=0
4460  C[33]=1
4465  CALL VWRITEBATCH(C[*])
4470  IF C[1] THEN GOSUB 9100
```

```
4475 C[33]=0
4480 IF NOT E THEN C[46]=C[46]+1
4485 C[27]=C[28]=0
4490 RETURN
4500 REM1 <PRINT KEY>
4505 I=1
4510 J=49
4515 CALL VPRINTFORM(C[*],I,J)
4520 IF C[1] THEN GOSUB 9100
4525 RETURN
4550 REM1 <REFRESH KEY>
4560 RETURN
4600 REM1 <PREV KEY>
4610 D1=1
4620 C[46]=C[46]-1
4630 RETURN
4650 REM1 <NEXT KEY>
4655 D1=0
4660 C[46]=C[46]+1
4665 IF NOT C[27] AND C[28] OR C[27]=2 THEN DO
4670   CALL VSHOWFORM(C[*])
4675   IF C[1] THEN GOSUB 9100
4680 DOEND
4685 RETURN
5000 REM2 <EXIT>
5010 REM1 *CLOSE TERMINAL*
5020 CALL VCLOSETERM(C[*])
5030 IF C[1] THEN GOSUB 9000
5040 C[1]=0
5050 REM1 *CLOSE BATCH FILE*
5060 IF B1 THEN DO
5070   IF E THEN PRINT M1$
5080   ELSE DO
5090     CALL VCLOSEBATCH(C[*])
5100     IF C[1] THEN GOSUB 9100
5110     C[1]=0
5120   DOEND
5130 DOEND
5140 REM1 *CLOSE FORMS FILE*
5150 CALL VCLOSEFORMF(C[*])
5160 IF C[1] THEN GOSUB 9100
5170 C[1]=0
5180 RETURN
8000 REM2 <PRINT STATUS LINE>
8010 S$=" ENTRY "+V$+'27"&a31CBATCH RECORD # "
8020 CONVERT C[46]+1 TO S$[LEN(S$)+1]
8030 S$=S$+'27"&a65CMODE: "
8040 IF NOT C[5] THEN S$=S$+"COLLECT"
8050 ELSE S$=S$+'27"&dKBROWSE"
8060 I=LEN(S$)
8070 CALL VPUTWINDOW(C[*],S$,I)
8080 RETURN
8100 REM2 <GET FORM NAMES FROM COM ARRAY>
8110 B$=B1$=""
```

```
8120 FOR K=11 TO 18
8130   B$=B$+FNU$(C[K])
8140   B1$=B1$+FNU$(C[K+8])
8150 NEXT K
8160 B$=DEB$(B$[1,15])
8170 B1$=DEB$(B1$[1,15])
8180 RETURN
8200 REM2 <PUT NEXT FORM NAME IN COM ARRAY>
8210 B1$[LEN(B1$)+1,16]=""
8220 B1$[16,16]='0
8230 FOR K=1 TO 15 STEP 2
8240   C[INT(K/2)+19]=FNP(B1$[K;2])
8250 NEXT K
8260 RETURN
9000 REM2 <PRINT ERROR MESSAGE TO SCREEN>
9020 CALL VERRMSG(C[*],M$,W1,I)
9030 PRINT M$
9040 RETURN
9100 REM2 <PRINT VIEW ERROR MESSAGE TO DISPLAY WINDOW>
9105 IF E THEN RETURN
9110 E=1
9125 CALL VERRMSG(C[*],M$,W1,I)
9130 M$=" "+M$
9135 I=LEN(M$)
9140 C[1]=0
9145 CALL VPUTWINDOW(C[*],M$,I)
9150 RETURN
9200 REM2 <PRINT ENTRY ERROR MESSAGE TO DISPLAY WINDOW>
9210 IF E THEN RETURN
9220 E=1
9230 I=LEN(E$)
9240 CALL VPUTWINDOW(C[*],E$,I)
9250 RETURN
9900 REM2 ** TERMINATE **
9910 PRINT "END OF PROGRAM"
9920 END
```

V/3000 ERROR MESSAGES

APPENDIX B

Any error diagnosed during execution of V/3000 is listed in this appendix. The list consists of the error number and its associated message. If a message contains the word "FSERR," it indicates a file system error has occurred; the word "COM'STATUS" indicates an error executing one of the V/3000 procedures.

Exclamation points in the message will be replaced by particular numbers or characters before the message is displayed. For instance, if a message contains FSERR !!!!, the exclamation marks will be replaced by a file error number when the message is displayed.

If a message contains the word "Internal", this means that the problem is internal to the system and is not easily corrected by the user. When such messages are issued, most users should call the system manager for advice.

HP V/3000 ERROR MESSAGES

These messages are issued by:

- FORMSPEC
- REFSPEC
- Procedures called by user programs
- REFORMAT program

FORMSPEC ERROR MESSAGES

Access Method Errors

Error	Message	Cause/Action
086	Access Method: File code error.	
087	Access Method: Attempt to add a duplicate key.	
088	Access Method: Internal error: Key insertion location error.	
089	Access Method: Internal error: Block not present.	
090	Access Method: Attempt to open an old file as new.	
091	Access Method: Internal error: Key block buffer error.	
092	Access Method: Internal error: Invalid key block number.	
093	Access Method: Internal error: Block not verified.	
094	Access Method: Record not found.	
095	Access Method: Not enough space in extension for the directory.	Fast form files require 500 words of space in the COMAREA extension. If you are coding in BASIC, list the forms file to find how many words are needed for the extension, and increase the COMAREA field USERBUFLEN to this number.

096	Access Method: Internal error: Parent block not found.
097	Access Method: Internal error: Illegal entry number.
098	Access Method: The file is not a KSAMless forms/ref file.
099	Access Method: The file is at EOF.

Screen Definition Errors

Error	Message	Cause/Correction
101	Form is too big, overflowed screen buffer.	Maximum screen buffer size is 7984 bytes. /Reduce the size of your form.
102	Form ! is too big to be compiled.	Form code record has exceeded 8K bytes. Maximum number of fields is 127. Reduce the number of fields, size of screen, or amount of processing specifications.
104	An END FIELD must have a matching START FIELD.	Check your screen design to make sure that each "end field" delimiter has a matching "begin field" delimiter. If your delimiters are the unprinted kind ("esc[" and "esc]"), it may be more convenient to delete the line containing the delimiters and to replace them.
106	Internal error: All of the screen was not read correctly.	Possibly a missing record terminator. Press ENTER again. If the error still occurs, reduce the size of the form or of the processing specifications.
107	A name cannot be longer than 15 characters.	Limit your form, field, or save field names to 15 characters.

108	A name cannot have embedded " "´s or "."´s.	Blanks and periods not allowed in names.
109	A name must start with an alphabetic character.	Special characters and digits are not allowed.
110	A name cannot have embedded " "´s or "."´s.	See error 108.
111	A name can only contain alphas, digits, and "_"´s.	Letters (A-Z),(a-z),numbers (0-9),or an underline (_) are allowed in a name.
112	Internal error: Unexpected CR.	Unexpected carriage return.
113	Internal error: Screen terminator never found.	See message for error 106. Also make sure enough terminal buffers are configured to complete the rest of the screen. The number of buffers can be increased to 255.
114	Internal error: Expected LF after CR.	Line feed expected after carriage return.
115	A field must contain a name.	Each field on the screen must have a name within the field boundaries.
116	Internal error writing screen message. (COMS´STATUS!!!!)	
118	Terminal error initializing new screen. (COM´STATUS!!!!)	
119	Terminal error writing new screen. (COM´STATUS!!!!)	
120	Terminal error initializing old screen. (COM´STATUS!!!!)	
121	Terminal error writing old screen. (COM´STATUS!!!!)	
122	Terminal error reading screen. (COM´STATUS!!!!)	See error 106.

123	Internal error writing the screen source. (FSERR!!!!)	Could not write the screen source record to the form file. First try to correct the problem using the information given in the File System error message. If you are unsuccessful, try the following a) The file may be full. Build a larger file using the :BUILD command and FCOPY. b) A system crash or abort may have corrupted the form. Re-enter all menus defining the form. c) If the above procedures fail, delete the form and recreate it.
124	Internal error reading the screen source for form !. (FSERR!!!!)	Could not read the screen source record from the forms file. See error 123.
125	A name can only be in the first line of the field.	If a field is continued on a second line, the name must be in the first line.
126	A name must be in the first line of the field.	See error 125.
127	Internal error: Allocate of GLOB'SCREEN failed.	
128	Internal error writing the global screen record. (FSERR!!!!)	See error 123.
129	Internal error: Allocate of STBL failed.	Could not allocate stack space for symbol table.
130	Internal error writing STBL to the forms file. (FSERR!!!!)	See error 123.
131	**Internal error: Changing screen of family.**	

Field Symbol Table Errors

Error	Message	Cause/Action
301	Internal error: FST ALLOCATE for form ! failed.	FST is Field Symbol Table. See error 129.
302	Internal error: Field symbol table for form ! is missing.	See error 123. Could not find Field Symbol Table for old form in form file. Press ENTER on screen design to re-create FST.
303	Error writing FST to forms file. File full? (FSERR!!!!)	Could not write FST record in forms file. See error 123.
304	Internal error: FST does not have any more room.	The form contains more than the maximum of 128 fields. If it does not, record your form on cassette, delete it, and reenter it. Note that this will renumber all fields.
305	The name must be unique for this form.	The field name is the same as another field name in the same form.
306	Internal error: FST search range error.	Field Symbol Table search problem. See error 304.
307	Internal error: FST field length changed.	Field Symbol Table access problem. See error 304.
308	Internal error: FST start order.	Field Symbol Table structure problem. See error 304.
309	Internal error: FST end order.	Field Symbol Table structure problem. See error 304.
313	Internal error: FST unrefer-enced error.	Field Symbol Table structure problem. See error 304.
314	Internal error: In form !, field ! has an invalid screen order.	Field Symbol Table structure problem. See error 304.
315	The field could not be found.	Field name does not exist.

316	Internal error: FST update screen order.	See error 308.
317	The field number must be unique for this form.	See error 308.
319	In form !, screen order for field ! is out of bounds.	See error 304.
320	Internal error: FST wrap loop.	See error 308.
321	Internal error reading FST for form !. (FSERR!!!!)	See errors 123,303.
322	Internal error storing FST for form !. (FSERR!!!!)	See errors 123, 303.

Menu Processing Utility Errors

Error	Message	Cause/Action
601	Must be B,H,I,U in any combination, or NONE.	Enhancement code is incorrect; check and correct.
604	The name cannot have embedded blanks.	A field or form name has embedded blank; delete blank or use underline(_).
605	The form name must be unique for this forms file.	Form name duplicates another form name in file; make name unique.
606	Must be NUMn,DIG,IMPn,MDY, DMY,YMD, or CHAR.	Data type incorrect; use one of the listed codes - n indicates number of decimal positions.
608	Must be O(optional), R(required), P(process), or D (display only).	Field type incorrect; use one of the listed codes.
609	Internal error: Global allocate failed.	
610	Internal error writing globals buffer. (FSERR!!!!)	See error 123.
611	For a NUMn data type, n must be a digit.	Use digit (0-9) after NUM data type to indicate number of decimal positions.
612	For the IMPn data type, n is required.	You must specify the number of implied decimal places in an IMP type field.
613	The name cannot be a reserved word.	See table 3-1 for a list of the FORMSPEC reserved words.

Menu Initialization and Processing Errors

Error	Message	Cause/Action
701	The name cannot be blank.	The form name must be specified on FORM menu.
702	The next form name cannot be blank.	Specify a particular next form name, or $HEAD, $END, $RETURN, or $REFRESH.
703	Error trying to open the forms file. (FSERR!!!!)	Refer to the File System Error number for the cause of the open failure. Check form name. Do not use the key file if the form file is old.
704	The window line must be a number between 0 and 24, inclusive.	Correct window specification on GLOBALS menu.
705	Must be N(no repeat), R (repeat), or A(repeat and append).	Correct Repeat Option on FORMS menu.
706	Must be C(clear), A(append), or F(freeze,then append).	Correct Next Form option on FORMS menu.
707	Internal error writing top lines to terminal. (COM´STATUS!!!!)	Field Menu.
708	Internal error writing terminal initialization. (COM´STATUS!!!!)	
709	Internal error writing field specs to terminal. (COM´STATUS!!!!)	Field Menu.
710	Error in VREADFIELDS. (COM´STATUS!!!!)	
711	Internal error reading the terminal, unformatted. (COM´STATUS!!!!)	Screen read error or processing specification read error.
712	Internal error positioning the cursor. (COM´STATUS!!!!)	
713	Error in VSHOWFORM. (COM´STATUS!!!!)	

714	Internal error writing the window. (COM´STATUS!!!!)	
715	Internal error writing the Field Menu source record. (COM´STATUS!!!!)	
716	Internal error writing the error cursor pos´n. (COM´STATUS!!!)	
717	Internal error writing the field top lines. (COM´STATUS!!!!)	Field Menu.
718	Error in VGETNEXTFORM for Main Menu. (COM´STATUS!!!!)	
719	Error in VSHOWFORM for Main Menu. (COM´STATUS!!!!)	
720	Error in VREADFIELDS for Main Menu. (COM´STATUS!!!!)	
721	Selection cannot be blank.	Enter a value in the MAIN menu selection box.
722	The form could not be found.	Form specified on MAIN menu not found in forms file.
723	Selection must be one of the characters listed below:	Enter one of the specified characters in the MAIN menu selection box.
724	This selection requires a form name to be supplied.	Specify a form name in the appropriate box, or enter a different character in the selection box.
725	Internal error writing form int´m rec for form !. (FSERR!!!!)	See error 123. Field Symbol Table error.
726	Internal error initializing the form !.	
727	A forms file has not yet been specified, so selection is invalid.	To copy or compile from MAIN menu, forms file must have been specified on FORMS FILE menu.

734	There was an error trying to open this forms file. (FSERR!!!!)	
736	Internal error reading the forms file. (FSERR!!!!)	
740	There are no forms in this forms file.	Specified forms file has no defined forms.
742	Save field length cannot be left blank.	Enter number of characters in save field on SAVE FIELD menu.
743	Save field length must be an integer greater than 0.	Enter save field length as integer between 1 and 999.
744	This save field already exists.	Use a new save field name.
745	A forms file can only have 20 save fields.	
746	There are no save fields in this forms file.	
747	The save field name cannot be blank.	Enter a save field name on the SAVE FIELD menu.
748	WARNING: Forms file modified and not compiled. Press EXIT to exit.	You modified the forms file and then pressed EXIT. If you want to compile before terminating FORMSPEC, press MAIN, otherwise press EXIT again.
749	This file is a Fast Forms File and cannot be modified.	You are attempting to modify a fast forms file; use the forms file name you originally specified.
750	This file is a Reformat File; use REFSPEC to modify it.	Specify a forms file name, or RUN REFSPEC.PUB.SYS to modify the reformat file.
751	This file is not a V/3000 Forms File.	You specified an incorrect file name.
752	Error opening the Fast Forms File. (FSERR!!!!)	
753	Error reading the forms file. (FSERR!!!!)	

754	Error writing to the Fast Forms File. (FSERR!!!)	See error 123.
755	Error finding the correct record. (FSERR!!!!)	
756	Error reading the chained records. (FSERR!!!!)	
757	Error opening the old Fast Forms file. (FSERR!!!!)	
758	Error purging the old Fast Forms File. (FSERR!!!!)	
759	Error closing the Fast Forms File. (FSERR!!!!)	
760	The old file is not a V/3000 Fast Forms File.	The fast forms file name you specified is not an existing V/3000 fast forms file.
761	This forms file is not a V/3000 forms file.	See error 751.
762	Fast forms file name must be specified if key file is specified.	You are trying to compile to a fast forms file without naming the data file.
763	Copy/Rename/Delete I/O error. (FSERR !!!!)	See error 123.
764	Error when deleting an old record.	See error 123.
765	The specified family is not defined.	
766	Cannot change screen. The form is a family member.	User pressed ENTER on the screen of a son form or attempted to change the screen. Press REFRESH (f4) to repaint the original and press NEXT (f6) to go to the field menus.
767	Cannot change family of the form.	Do not attempt to change the name in the ''Reproduced from'' field of the form menu.

Menu Controller Errors

Error	Message	Cause/Action
801	Sorry, you cannot define more than 255 forms in a forms file	
802	The PREV key is not appropriate; there are no previous menus.	You are currently at the first menu; press NEXT to go to the next menu, NEXT FORM to display the next form menu, or use the MAIN menu to select a particular menu.
803	The function key just hit is not defined for this mode.	
804	Internal error reading source record from forms file. (FSERR!!!!)	See error 123.
805	Internal error getting next form !. (COM´STATUS!!!!)	
806	Error in VSHOWFORM. (COM´STATUS!!!!)	
807	Error in VREADFIELDS. (COM´STATUS!!!!)	
808	Error in VGETBUFFER. (COM´STATUS!!!!)	
809	Internal error writing source to forms file. (FSERR !!!!)	See error 123.
811	Internal error reading form record for form !. (FSERR!!!!)	See error 123.
812	Internal error reading globals record. (FSERR!!!!)	See error 123.
813	Internal error allocating GLOB´SCREEN for form !.	
814	Internal error allocating STBL for form !.	
815	NEXT is not meaningful until a forms file name is specified.	Enter a forms file name on the FORMS FILE menu. (Exit if necessary and run FORMSPEC again to display the menu.)

816	NEXT is not appropriate, since there are no more forms.	Use PREV or PREV FORM key, or go to the MAIN menu to request a particular form.
817	The save field could not be found.	
820	The PREV FORM key is not appropriate; there are no previous forms.	Use NEXT or NEXT FORM key, or go to the MAIN menu to request a particular form.
821	Internal error allocating the globals buffer	
822	This value is required and cannot be blank.	

Init and Compile Errors

Error	Message	Cause/Action
902	For form !, the next form ! has not been defined.	The next form name specified does not exist in the forms file. Change the next form name or create a form with that name.
903	The head form ! has not been defined.	The head form specified on the globals menu does not exist in the forms file. Create a form with the head form name or change the head form on the globals menu.
904	Internal error closing FORMSPEC forms file. (COMSTATUS!!!!)	
905	Internal error closing the terminal file. (COMSTATUS!!!!)	
909	Internal error writing the global code record. (FSERR!!!!)	See error 123.
911	Internal error writing form code record for form !. (FSERR!!!!)	See error 123.
912	Internal eror reading screen source for form !.	
913	Internal error closing your forms file. (FSERR!!!!)	
916	In form !, the field name ! is not unique.	Duplicate field name found in indicated form; use FIELD menu to change field name.
917	Internal error writing field names table for form !. (FSERR!!!!)	
918	Internal error writing save field table for form !. (FSERR!!!!)	See error 123.

919	The initial value is too long.	The initial value is longer than the length of the field as defined on the screen menu. Use the screen menu to lengthen the field or shorten the initial value on the field menu.
920	The initial value contains an invalid DIG value.	A DIG type field may contain only digits (0-9), no decimal point, commas, or sign.
921	The initial value contains an invalid NUM value.	A NUM type field may contain only digits, decimal point, commas, a sign.
922	The initial value contains an invalid numeric value.	This field may only have a numeric value; no letters or special characters.
923	Internal error: For form !, allocating local buffer for field !.	
924	Internal error: For form !, reading source for field !. (FSERR!!!!)	See error 123.
925	In form !, field ! has an invalid initial value.	The initial value specified does not match the field data type.
926	In form !, field name ! is a reserved word.	Check list of reserved words (table 3-1) and change field name.
927	The initial value contains an invalid MDY date value.	Make sure date is in order month day year, is a valid date, spelled correctly.
928	The initial value contains an invalid DMY date value	Make sure date is in order day month year, is a valid date, spelled correctly.
929	The initial value contains an invalid YMD date value.	Make sure date is in order year month day, is a valid date, spelled correctly.
932	The initial value contains an invalid IMP value.	An implied decimal field may contain only digits, decimal point, commas, sign.

950	EDIT statements are not allowed in the INIT phase.	Field edits cannot be performed in the INIT phase; move statement to FIELD phase.
951	EDIT statements are not allowed in the FINISH phase.	Field edits cannot be performed in the FINISH phase; move statement to FIELD phase.
952	Only a field, save field, or implied current field is allowed.	
953	Only one character is allowed in a FILL statement.	Enter single character surrounded by quotes.
954	Only 10 or 11 is allowed in a CDIGIT statement.	CDIGIT tests only modulo 10 or modulo 11 check digits.
994	In form !, field !, "!" is undefined; press NEXT to see error.	Indicated specification is not recognizable; NEXT will display FIELD menu so you can make correction and then press ENTER.
995	Internal error in form !, field !; press NEXT to see error.	NEXT displays FIELD menu and another message with the cursor positioned to the location of the error so you can correct the specification and then press ENTER.
996	Form !, field ! has a semantic error; press NEXT to see error.	NEXT displays FIELD menu so you can correct the error and then press ENTER. The syntax of your statement may be correct, but its meaning is in error.
997	Form !, field ! has a data type error; press NEXT to see error.	Data type inconsistent with specification; NEXT displays FIELD menu so you can correct data type or specification and press ENTER.
998	Form !, field ! has a syntactic error; press NEXT to see error.	NEXT displays FIELD menu so you can correct statement syntax and then press ENTER.
999	Field processing specification syntax error.	Correct syntax of statement and then press ENTER.

Scanner Errors

Error	Message	Cause/Action
1001	Too many levels of indenting, limit is !!!!.	Only 10 levels of nesting are allowed.
1002	Improper indenting, statement groups must be vertically aligned.	IF and ELSE parts at same nesting level must be indented the same amount.
1003	Improper indenting, must be within outer level of indenting.	An ELSE or subsequent IF starts to the left of first IF statement.
1010	In MATCH, more left braces than right braces were found.	Braces must match; add right brace, or delete extra left brace.
1011	In MATCH, more right braces than left braces were found.	Braces must match; add left brace or delete extra right brace.
1030	Token is too long, limit is !!!! characters.	A string exceeds 120 characters. Correct and press enter.
1031	A $ may only appear with a system defined name like $EMPTY.	A $ cannot be part of a name unless it is one of the system defined names (see table 3-1).
1032	Expected a statement beginning.	Syntax error. The cursor is positioned to the location of the error.
1033	String is too long, limit is !!!! characters.	
1034	String must end with a quote.	A literal string (character constant) must be enclosed within quotes.
1035	String continuation must begin with a quote.	When continuing a string to the next line, close the quotes, put an & at the end of first line, enclose string on second line within quotes.

1036	Only a comment may appear on the same line after an ampersand.	An ampersand (&) is used to continue a string; it cannot be followed by another statement, only by a comment starting with \.
1037	A date constant must end with an exclamation point.	A date used as a constant must be enclosed within exclamation points.
1038	Date constant too long, limit is !!!! characters.	
1039	Date constant must be a valid MDY date.	Date constants must be in the order: month day year, and must be valid dates.

Parser Errors

Error	Message	Cause/Action
1094	Invalid name: no field or save field with this name is defined.	A valid field name is expected.
1095	Internal error.	
1096	Semantic error	See error 1098.
1097	Data type error: Data types must be compatible.	Data type and value in processing specification do not correspond.
1098	Field processing specification syntax error.	Statement is incorrect. Cursor is positioned to the location of the error. Correct and press enter.

Apply Errors

Error	Message	Cause/Action
1101	Left operand must be type DIG, NUMn, IMPn, or an ex-	An arithmetic operand is expected as destination of SET or comparison statement. Cursor is positioned to the location of the error.
1102	Both operands must be type DIG, NUMn, IMPn, or an expression.	Arithmetic operands expected in SET or comparison statement. Cursor is positioned to the location of the error.
1103	Operand must be type DIG, NUMn, IMPn, or an expression.	Arithmetic operand expected in SET or comparison statement.
1104	Operand types must be compatible.	Operands of the same data type are expected in a SET or comparison statement.
1105	Operand types must agree unless source is a field or a save field.	Operands must be the same data type unless the source is a field or save field name.
1106	Right operand must be type CHAR.	A character type value is expected as source of SET or comparison statement.
1107	Left operand must be a field or save field.	Destination operand of SET statement may only be a field or save field name.
1108	NFORM name must be a FORM name in quotes, or a type character field.	In CHANGE NFORM statement, the next form is identified by its name in quotes, by a char type field containing the form name, or by a system constant such as $HEAD.
1109	MINLEN operand must be type DIG, NUMn, IMPn, or a number.	The minimum field length is specified as a positive integer, or a name of a numeric field.

1110	CDIGIT operand must be 10 or 11.	CDIGIT checks only modulo 10 or 11 check digits.
1120	Too many statements, code is too big.	At compilation, the form information generated does not fit into the maximum record size of 8192 bytes. Reduce the number of fields, processing specifications, or custom error messages.
1130	Improper indenting, statement groups must be vertically aligned.	Corresponding IF and ELSE parts of IF statement must start in same column.
1131	Syntax error: A name is not expected here.	A name occurs where a reserved word is expected. Check spelling, correct and press ENTER.
1132	Syntax error: The statement is incomplete.	A valid statement ends unexpectedly. Correct and press ENTER.
1133	Syntax error: Unexpected symbol.	A special character is not expected here.
1923	Internal error: Unexpected auto-indent.	(Internal error in IF-THEN-ELSE statement.
1924	Internal error: Error numbers are incorrect.	
1925	Internal error: MEM size is inconsistent with ARRAYSIZE.	
1926	Internal error: XEVAL recursion overflow.	
1927	Internal error: Invalid node in XEVAL.	
1928	Internal error: Invalid MATCH operand in XEVAL.	
1929	Internal error: Date conversion failed in XEVAL.	
1930	Internal error: XEVAL recursion overflow.	

Pattern Compile Errors

Error	Message	Cause/Action
2101	A "(",")","[", or "]" is not paired in pattern.	MATCH statement contains odd number of parentheses or brackets.
2102	Expected a character in pattern.	
2103	Tree too big in pattern.	
2104	Garbage characters found after pattern.	
2105	A range is transposed in pattern.	Range (a:b) in MATCH statement has first character greater than second character.
2202	Too many states required in pattern.	
2202 ?	Too many moves required in pattern.	
2203	FSM is too big in pattern.	Finite State Machine.
2204	Too many final states in pattern.	
2205	Too many final moves in pattern.	
2900	Internal error: COMPILE´ PATTERN failed.	

REFSPEC MESSAGES

Access Method Errors

Error	Message	Cause/Action
086	Access Method: File code error.	
087	Access Method: Attempt to add a duplicate key.	
088	Access Method: Internal error: Key insertion location error.	

089	Access Method: Internal error: Block not present.	
090	Access Method: Attempt to open an old file as new.	
091	Access Method: Internal error: Key block buffer error.	
092	Access Method: Internal error: Invalid key block number.	
093	Access Method: Internal error: Block not verified.	
094	Access Method: Record not found.	
095	Access Method: Not enough space in extension for the directory.	Fast form files require 500 words of space in the COMAREA extension. If you are coding in BASIC, list the forms file to find how many words are needed for the extension, and increase the COMAREA field USERBUFLEN to this number.
096	Access Method: Internal error: Parent block not found.	
097	Access Method: Internal error: Illegal entry number.	
098	Access Method: The file is not a KSAMless forms/ref file.	
099	Access Method: The file is at EOF.	

Output Sequence Definition Errors

These messages apply to the OUTPUT RECORD menu. Tab positions refer to the numbered positions on the menu illustrated in figure 5-17.

Error	Message	Cause/Action
101	Field name has more than 15 characters.	Limit name of input field to 15 characters.
102	Field name improperly aligned on screen.	Enter input field name at tab position 1.
103	Form name has more than 15 characters.	Limit name of input form to 15 characters.
104	Form name ! is not in input forms sequence.	A form name on the OUTPUT RECORD menu was not named on corresponding INPUT FORMS menu.
105	A digit is not a legal character in this field.	A field or form name is expected in this field.
107	Substring start column offset is improperly aligned on screen.	Enter input substring start column at tab position 2.
108	Substring starting column has more than 4 digits.	Substring start column must not exceed 4 digits.
111	Closing quote expected.	Character string constant must be enclosed within quotes; the closing quote is missing.
113	Input length field is improperly aligned on screen.	Enter input substring length at tab position 3.
114	Internal error: Expected LF after CR.	Line feed expected after carriage return.
115	Line must start with field name, literal or system constant.	First column of OUTPUT RECORD menu should contain first character of field name, a literal, or system constant.
116	Input length cannot have more than 4 digits.	Input substring length must not exceed 4 digits.
118	Field name cannot be blank.	A field name must be specified if other characteristics are entered on line.

119	Letters are not allowed in this field.	Only digits are expected in this field.
121	Form name is improperly aligned on screen.	Enter input form name at tab position 4.
123	Output field name has more than 15 characters.	A field name is limited to 15 characters.
124	Output field name is improperly aligned on screen.	Enter output field name at tab position 5.
126	Length of output field may only have up to 4 digits.	Output field length must not exceed 4 digits.
127	Output field length is improperly aligned on screen.	Enter output field length at tab position 7.
129	Output field starting offset can only have up to 5 digits.	Starting column of output field must not exceed 5 digits.
130	Output field starting offset is improperly aligned on screen.	Enter starting column of output field at tab position 6.
131	Start record indicator is not in its field.	Enter start-of-record marker at tab position 8.
132	Start record indicator should be a single character.	A start-of-record marker is limited to 1 character.
133	Column numbering starts with 1.	The starting column in the output record cannot be zero.
134	This is not a legal system constant.	Use one of the legal constants: $CR, $LF, $GS, $US, $RS, or $ followed by the numeric equivalent of an ASCII character. Other constants may not start with a $.
135	Output field length must be a positive integer greater than 0.	Output length cannot be zero or a negative number.
136	Output field starting column causes overlapping fields.	The starting column for this field falls within the bounds of a previous field. Enter a higher value.

137	A single literal cannot have more than 75 characters.	Literal must fit on a single line of OUTPUT RECORD menu and must not overwrite the start-of-record marker position.
138	Literal must not continue beyond column 75.	One line of OUTPUT RECORD menu is limited to columns 1-75.
139	This is not a field. No character may appear here.	Characters entered between "fields" on OUTPUT RECORD menu. Use tabs to go to start of each field, and enter only the allowed number of characters.
140	Only literal,ASCII constant or system constant expected here.	
141	Digit must follow sign.	
142	Value for ASCII character must be between 0 and 127.	The numeric equivalent of an ASCII character may be only 0-127.
143	Field name must start with char. No embedded blanks allowed.	First character of field name must be alphabetic; field name may not have embedded blanks.
144	Record size must have a value.	Record size on globals menu is a required field. User must give maximum size of output record.
145	"Y" or "N" expected in upshift field.	On GLOBALS menu, only a Y or N can be entered in response to "Upshift?"; default is N.
146	"Y" or "N" expected in answer to "Convert to EBCDIC?".	On GLOBALS menu, only a Y or N can be entered in response to "Convert to EBCDIC?"; default is N.
147	Illegal character.	Character in output record menu is neither alphabetic nor numeric.
148	This is not a field.	In output record menu, you may enter data only in the highlighted fields.

149	Form name must start with char. No embedded blanks allowed.	First character of form name must be alphabetic; name may not have embedded blanks.
150	Output field name must start with char. No embedded blanks allowed.	See error 143.
151	Internal error: Field alignment check failed.	
153	Fields cannot have embedded blanks.	Embedded blanks are not allowed in any field of the OUTPUT RECORD menu.
154	Input length field cannot embedded blanks.	Length of input field must be a positive integer with no embedded blanks.
155	Start Substring field cannot have embedded blanks.	Input starting column must not contain embedded blanks.
157	Output starting column field cannot contain blanks.	Embedded blanks are not allowed as part of the output starting specification.
158	Input length field cannot contain blanks.	No embedded blanks are allowed in the input length field.
159	Input length must be greater than 0.	Specify input length as positive integer greater than zero.
160	Substring starting column exceeds length of forms file field.	Start of input field must be within the forms file field; column specifed is greater than the number of characters in the form field.
161	Substring start column + input length > length of forms file field.	Specified starting column plus length of field add up to more characters than the forms file field contains.
162	Input length for reformat field exceeds length of forms file field.	Length specified for input field is greater than the number of characters in the field.

Validation Errors

These messages result from checking the validity of entries on the OUTPUT RECORD, INPUT FORMS, and GLOBALS menus.

Error	Message	Cause/Action
201	Field ! not found in form !.	Input field in OUTPUT RECORD menu not found in specified form.
202	Field ! not found in any form in input forms sequence.	Input field in OUTPUT RECORD menu not found in any form in the corresponding INPUT FORMS menu.
203	Reformat identifier ! is not a unique reformat identifier.	First form specified on INPUT FORMS menu is not unique to the reformat file. Use a form name that has not been specified as a reformat id-entifier on another menu.
204	Internal error: data type is bad.	See error 206.
205	Output record length is greater tha globally defined record length.	Change either the total output record length on the OUTPUT RECORD menu, or the record length on the GLOBALS menu, or start a new record using the start record marker.
206	Internal error: Bad entry in field table. Illegal field type.	Internal descriptions of fields are bad. Press ENTER again. If that fails, delete and recreate the offending format.
207	Error opening reformat file. (FSERR !!!!)	Refer to the file system error for the cause of the open failure.
208	A unique reformat identifier is required.	See error 203.
210	Internal error: OFST table is bad.	See error 206. OFST is Old Field Symbol Table.
211	Internal error: Field table is bad.	See error 206.

212	Form name must begin with an alphabetic character.	Form name cannot start with a special character or a digit.
213	Form name cannot have embedded " "´s or "."´s.	The only non-alphanumeric character allowed in a form name is the underline "_".
214	Form name may contain only alphas, digits and "_"´s.	Use the correct naming conventions: A name may contain letters, digits, or the underline, must start with a letter, and have no more than 15 characters.
215	Internal error in ADJUST´ALL´START´COL.	Internal error in recording the change in the default starting position of each field, a change resulting from altering the field or record separators in the globals menu.
216	Internal error reading reformat intermediate record for !.	Error reading an internally used record from the reformat file.
217	WARNING: Start column goes beyond global output record length.	Starting column for output record specified on OUTPUT RECORD menu is greater than the record length on the GLOBALS menu. To avoid truncation, change one of the two specifications.
218	Embedded blank form names not allowed in a sequence.	A form name consisting of a blank found in the INPUT FORMS menu. Remove the blank line from the sequence.
219	Internal error: Data type is bad.	See error 204.
220	WARNING: This field causes output record size to be too big.	A field specified on the OUTPUT RECORD menu is beyond the record size set on the GLOBALS menu. Change the GLOBALS specification or delete the field.

Field Table Errors

Error	Message	Cause/Action
301	Internal error: OFST ALLOCATE failed.	Space allocated for dynamic array is insufficient.
302	Internal error: OFST record missing. Write failed previously.	Attempt to read internal record failed because the record did not exist.
303	Internal error: OFST write failed. (FSERR !!!!)	Attempt to write an internal record failed.
304	Internal error: OFST does not have any more room.	
305	Output field name ! not unique.	Output field name on OUTPUT RECORD menu must be unique for the record. Use output field name to provide unique name.
306	Internal error: OFST search range error.	See error 301.
307	Internal error reading OFST record. (FSERR !!!!)	Failure to read an intermediate (internal) record.
308	Internal error: OFST start order.	See error 301.
309	Internal error: OFST end order.	See error 301.
313	Internal error: OFST unref-erenced error.	See error 301.
316	Internal error: OFST update screen order error.	See error 301.
320	Internal error: OFST wrap loop error.	See error 301.
400	Internal error: Field table allocation failed.	See error 301.
401	Internal error: Field table record missing. Prev write failed.	See error 302.
402	Internal error reading FST from forms file. (FSERR !!!!)	Field names in output record record menu cannot be checked against the forms file because of internal inability to read information from forms file.

403	Internal error writing FST to reformat file. (FSERR !!!!)	See error 303.
404	Internal error: Field table has no more room.	See error 301.
405	Internal error reading input seq from reformat file. (FSERR !!!!)	See error 402.
406	Internal error: Input seq table record missing. Prev write failed.	See error 302.
407	Internal error writing input seq table to refmt file. (FSERR !!!!)	
450	Form ! does not exist in the forms file.	Form name specified on INPUT FORMS menu is not in forms file. Check name and correct it.
451	Internal error reading glob forms buf from forms file. (FSERR !!!!)	Failure to read list of forms in the forms file.
452	Internal error: Global forms file rec missing. Prev write failed.	
453	Internal error reading field name table for form !. (FSERR !!!!)	Failure to read list of all fields for a particular form in the forms file.
460	Internal error in reading globals from forms file. (FSERR !!!!)	
461	Internal error: Allocation of form´glob´buf failed.	
462	Internal error: Allocation of form´flds´buf failed.	
463	Internal error: Allocation of literal table failed.	
464	Internal error: Literal table record missing. Prev write failed.	

465	Internal error: Error reading literal table record. (FSERR !!!!)	
466	Internal error writing literal table record. (FSERR !!!!)	
468	Internal error: Allocation of global buffer failed.	
469	Form ! does not exist in this forms file.	See error 450.

Menu Processing Errors

These errors apply to the MAIN menu.

Error	Meaning	Cause/Action
501	Selection field cannot be blank.	Enter one of the following codes in the MAIN menu selection box: A, X, G, F, L, or D.
502	Reformat identifier ! not found.	A reformat id entered on the MAIN menu is not in the reformat file. Check the INPUT FORMS menus or correct the identifier.
503	Output field name ! not found.	An output field name entered on the MAIN menu is not in the reformat file. Check the OUTPUT RECORD menus or correct the field name.
504	No reformat identifier was specified.	A selection on the MAIN menu requires a reformat identifier. Enter the identifier or change the selection.
505	There are no input forms sequences in the reformat file.	A reformat identifier was specified, but no INPUT FORMS menu has been entered.
506	No reformat to be deleted was specified.	"D" was entered in MAIN menu selection box; specify the reformat identifier to be deleted.
507	No such reformat identifier exists.	A reformat identifier on the MAIN menu is not on any INPUT FORMS menu.
508	Main selection field must be "A", "X", "G", "F", "L", or "D".	Enter one of the listed codes.
509	This field must be blank when deleting.	The reformat id for deletion should be in the righthand box; clear the lefthand box.
510	Listing is not done on field level.	Enter "L" to list, and a reformat identifier in the lefthand box; you cannot list a single field.

511	Form name must be blank for compile.	When you enter "X" to compile the reformat file, you cannot enter a reformat identifier.
512	Field name must be blank for compile option.	When you enter "X" to compile the reformat file, you cannot enter an output field name.
513	Form name is not specified when adding a form.	When you enter "A" to add a reformat, do not specify the reformat id; reformat id´s are entered on the INPUT FORMS menu.
514	Fields are not added from the MAIN menu.	Output fields are added on the OUTPUT RECORD menu,not using the "A" option of the MAIN menu.
515	Form name must be blank for this option.	A selection was made on the MAIN menu that cannot use a reformat id. Clear the reformat id box.
516	Field name must be blank for this option.	A selection was made on the MAIN menu that cannot use an output field name. Clear the output field box.

Menu Processing Utility Errors

These messages result from checking selections made on the OUTPUT FIELD and GLOBALS menus.

Error	Meaning	Cause/Action
600	NUMn can have digits only from "0" to "9".	A value other than a digit was entered for a NUMn type field; change the data type or the selection.
606	Must be NUMn, DIG, IMPn, MDY, DMY, YMD, or CHAR.	Enter one of the listed codes as the data type of this field; n can be a digit 0-9 for NUM, 1-9 for IMP.
608	Justify field must be "L", "R", or "C".	Enter one of the codes to justify field Left, Right, or Center; default is Right justify for numbers or dates, no justification for characters.
609	Checkdigit must be either "10" or "11".	If you want to insert a check digit, specify 10 for modulus 10, 11 for modulus 11.
610	Checkdigit can only be added to fields of type DIGIT or CHAR.	If you want to add a check-digit, change the data type on the OUTPUT FIELD menu.
611	Internal error: substring start column value could not be displayed.	Intrinsic ASCII failed.
612	Numeric fields cannot be converted to fields of date type.	When the input field is a number, you cannot specify a data type of MDY,DMY, or YMD on the OUTPUT FIELD menu. Change the data type.
613	Date fields cannot be converted to fields of numeric type.	When the input field is a date, you cannot specify DIG, IMPn, or NUMn as the data type on an OUTPUT FIELD menu. Change the data type to a date or CHAR.

614	Fields of type CHAR cannot be converted to other data types.	When the input field is type CHAR, you cannot change the data type on an OUTPUT FIELD menu. Leave the type as CHAR.
615	Internal error: Input length value could not be displayed.	Intrinsic ASCII failed.
617	Only fields of type NUM, NUMn, or IMPn can be signed.	If the input field is not one of the listed types, you cannot specify a sign on the OUTPUT FIELD menu.
618	Sign must be "L", "R", "F", "Z", "N", or " ".	Only the listed codes or a space can be entered on the SIGN box of the OUTPUT FIELD menu. Correct your entry.
620	Plus field must be "Y" or "N".	If you want the plus sign retained in the field, enter Y in the PLUS SIGN? box; if N or blank, only minus signs are inserted.
621	Expected "F", "V", or "U" in record format field.	Enter one of the listed codes in the Output Record Format box of the GLOBALS menu; default is F (fixed length).
622	Recordsize must be a numeric value less than 32767.	Enter a positive integer less than 32767 in the Record Length box of the GLOBALS menu, or leave the default length of 80 characters.
624	Expected literal, system constant, or ASCII equivalent in string.	Record terminator or field separator string can only be a quoted string, the numeric equivalent of an ASCII character preceded by $, or a system constant: $LF, $CR, $GS, $US, or $RS.
625	Output field must be expanded to allow for checkdigit.	If you enter 10 or 11 in the CDIGIT box of the OUTPUT FIELD menu, be sure to increase the size of the output field on the OUTPUT RECORD menu.

626	Internal error: Checkdigit option not valid for data type.	See error 610.
627	Strip fields must start and end with a quote.	Enclose any characters in a STRIP box of the OUTPUT FIELD menu between quotes.
628	Use "" or `´ to indicate " or ´ inside a literal.	Use double quotes for any quotation marks within a literal (literal is entered in the GLOBALS menu.)
629	Plus option cannot be "Y" if sign option is "N".	If you want a plus sign, you must also specify where you want any sign in the SIGN box of the OUTPUT FIELD menu.
630	Fill all has already filled leading blanks.	Enter a character in Fill All OR Fill Leading, not both.
631	Fill all has already filled trailing blanks.	Enter a character in Fill All OR Fill Trailing, not both.
632	IMPn can have digits only from "0" to "9".	Number of implied decimal places must be a digit.

Menu Initization and Processing Errors

Error	Message	Cause/Action
702	Internal error: Intrinsic ASCII failed.	
703	Largest legal integer value is 32767.	Error in Record Length field of GLOBALS menu.
704	Length of field cannot be negative.	Negative value specified as field length on OUTPUT RECORD menu.
705	Internal error initializing OUTPUT RECORD menu. (COMSTATUS !!!!)	Internal error in writing the header part of the output record menu.
706	Internal error writing old output recdef to screen. (COMSTATUS !!!!)	Internal error in writing user´s specifications--part of output menu.
707	Internal error reading output recdef from screen. (COMSTATUS !!!!)	Attempt to read the output record menu failed.
708	Internal error in writing output recdef to refmt file. (FSERR !!!!)	
709	Internal error reading source rec from reformat file. (FSERR !!!!)	
710	Internal error in writing source rec to reformat file. (FSERR !!!!)	
714	Internal error writing the window. (COMSTATUS !!!!)	
716	Internal error writing the error cursor pos´n. (COMSTATUS !!!!)	
742	Internal error: Start column could not be displayed.	Intrinsic ASCII failed.
743	Internal error: Length could not be displayed.	Intrinsic ASCII failed.
744	Forms file is not a valid forms file or forms file is not compiled.	The forms file specified on the FORMS FILE menu is not valid. Correct the name or go back to FORMSPEC and compile the forms file.

745	Internal error in T´READ´NO´ HOME. Bad key read. (COMSTATUS !!!!)	
746	Internal error in cursor positioning before read.	
747	Internal error writing the current reformat rec for !. (FSERR !!!!)	
748	Internal error initializing Input Form Menu for reformat id !.	
749	Internal error reading MAIN MENU form. (COMSTATUS !!!!)	
750	Internal error showing MAIN MENU form. (COMSTATUS !!!!)	
751	Internal error getting MAIN MENU form. (COMSTATUS !!!!)	
752	Internal error in initializ-ing output record menu. (COMSTATUS !!!!)	
753	Internal error: VSETERROR failed. (COMSTATUS !!!!)	
754	Internal error: VGETINFO failed. (COMSTATUS !!!!)	
755	File is not a forms file.	File name specified on the FORMS FILE menu does not name an existing forms file. Correct the name.
756	File is not a reformat file.	The file name specified on the REFORMAT file menu does not identify an existing reformat file.
757	Internal error: Could not get information from refor-mat file.	Intrinsic FGETINFO failed.
758	Internal error: Could not get information from forms file.	Intrinsic FGETINFO failed.

Menu Controller Errors

These messages result from pressing the wrong function keys.

Error	Message	Cause/Action
802	The PREV key is not allowed here.	There is no previous menu.
803	The function key just hit is not defined for this mode.	This key has no meaning on the current menu.
805	Error getting next form. (COMSTATUS !!!!)	
806	Internal error in VSHOWFORM. (COMSTATUS !!!!)	
807	Internal error in VREADFIELDS. (COMSTATUS !!!!)	
808	Internal error in VGETBUF. (COMSTATUS !!!!)	
810	Internal error in writing reformat rec for reformat !. (FSERR !!!!)	
811	Internal error in reading reformat rec for reformat !. (FSERR !!!!)	
812	Internal error in reading globals record. (FSERR !!!!)	
815	NEXT is not meaningful until a forms file name is specified.	Enter a forms file name on the FORMS FILE menu before using the NEXT key.
816	NEXT is not appropriate. There are no more reformats.	Use the MAIN menu to go to a particular reformat, or use PREV REFORMAT key to go to a prior reformat.
817	NEXT is not meaningful until a reformat name is specified.	Enter a reformat file name on the REFORMAT FILE menu before using the NEXT key.
818	You cannot specify more than 255 reformats.	You have already entered 255 INPUT FORMS menus; try to use fewer.
819	The NEXT REFORMAT key is inappropriate. There are no more reformats.	There are no more INPUT FORMS menus in the file.

820	There are no previous reformats.	The current reformat is the first one. Use the main menu to go to a particular reformat or use the NEXT REFORMAT key to go to the next input forms sequence.
821	Internal error in writing globals table record. (FSERR !!!!)	
822	WARNING: Changes in input forms menu may have invalidated this menu.	The OUTPUT FIELD menu may be affected by changes to the INPUT FORMS menu.
823	WARNING: Changes in input forms menu may have invalidated this menu.	The OUTPUT RECORD menu may be affected by changes to the INPUT FORMS menu.
824	Internal error: Missing record for reformat !. Prev write failed.	
825	NEXT´REFORMAT is not meaningful until forms file name is specified.	
826	NEXT´REFORMAT is not meaningful until reformat file name is specified,	
827	Cannot go to MAIN MENU until reformat file name has been specified.	
828	Cannot go to MAIN MENU until forms file name has been specified.	Do not press MAIN key until you have entered forms file name on FORMS FILE menu.
829	Reformat file was modified since last compile. Press EXIT to exit.	You pressed EXIT before compiling the reformat file; press MAIN to go to MAIN menu to compile, or press EXIT to exit without compiling.

Initization and Compile Errors

These messages all result from file system errors found while initializing or compiling the reformat file.

Error	Message	Cause/Action
906	Error writing reformat !. (FSERR !!!!)	
913	Internal error closing your forms file. (FSERR !!!!)	
914	Internal error in storing the field symbol table. (FSERR !!!!)	
917	Internal error in storing OFST. (FSERR !!!!)	
918	Error closing reformat file. (FSERR !!!!)	
919	Internal error in closing REFSPEC forms file. (FSERR !!!!)	
920	Internal error in closing terminal file. (FSERR !!!!)	
921	Internal error in writing globals record on exit. (FSERR !!!!)	
922	Internal error in storing literal table. (FSERR !!!!)	
923	Error in opening user's forms file. (FSERR !!!!)	Refer to the file system error to determine the cause of the open failure.

Compile Errors

Error	Message	Cause/Action
1000	Internal error on compile. Bad field type.	
1001	Internal error writing field intermediate code rec. (FSERR !!!!)	
1002	Internal error writing reformat code record. (FSERR !!!!)	
1003	Internal error: Allocation of compile buffer failed.	
1004	Internal error writing globals code record. (FSERR !!!!)	
1005	Internal error reading field intermediate code rec. (FSERR !!!!)	
1006	REFSPEC version has been changed. Reformat file invalid.	The specified reformat file was produced on an earlier version of REFSPEC.
1007	Reformat file is empty. There is nothing to compile.	Enter A to add a reformat before entering X to compile.
1008	Error in output record menu in reformat !.	Go to the OUTPUT RECORD menu for the specified reformat, and make correction.
1009	Error in input forms menu in reformat !.	Go to the specified reformat (INPUT FORMS menu) and make correction.
1010	Output field ! of reformat ! is in error.	Go to the specified OUTPUT FIELD menu and make correction.
1011	Internal error writing code rec for input forms seq. (FSERR !!!!)	

HP V/3000 PROCEDURE ERROR MESSAGES

These messages are issued as a result of errors executing HP V/3000 procedures called from user programs. The messages are organized according to the particular procedure called.

VOPENTERM and VCLOSETERM Messages

Error	Message	Cause/Action
001	Internal error: Terminal file initialization failed.	FCONTROL or FSETMODE intrinsic failed.
002	Your terminal does not seem to be an HP 264x terminal.	HP V/3000 only works on the following HP terminals: 2640B, 2641, 2644, 2645, 2647, and 2648.
003	Internal error: SETMSG command failed.	VOPENTERM could not execute the MPE command :SETMSG OFF or VCLOSETERM could not execute :SETMSG ON.
004	The HP264x terminal status failed.	VOPENTERM's request for terminal status failed. Check that strapping is correct. May also be data comm problem.
005	The terminal needs the D strap in (closed).	Message used only with the 2640A terminal, which is not supported by V/3000.
006	The terminal needs the H strap in (closed).	Refer to appendix G for instructions on how to strap your terminal for HP V/3000.
007	Internal error: HP2640 mode could not be correctly set.	Possible block mode or data comm problem.
008	Internal error: HP2645 mode could not be correctly set.	See error 007.
009	The terminal open failed. (FSERR !!!!)	FOPEN intrinsic failed.
010	Internal error: The terminal close failed. (FSERR !!!!)	FCLOSE intrinsic failed.
011	Sorry, the HP2640A terminal is not supported by V/3000.	The HP2640B is supported, as are the HP2641/44A,45/47/48.
012	The HP2640B needs straps A,C, H in (closed); D,F,G out.	You must strap this terminal (and the 2644A) yourself;

		other terminals are strapped automatically. (Refer to appendix G.)
013	The HP2644A needs straps A,C, H in (closed); D,F G out.	You must strap this terminal (and the 2640B) yourself; other terminals are strapped automatically. (Refer to appendix G.)
014	The HP2645A/41A/48A needs straps A,C,H,J in (closed); D,G out.	Note that these terminals should be configured automatically by HP V/3000.
015	Internal error: Type of HP246x terminal incorrectly determined.	Possible block mode or data comm problem. Check straps. Note: 2645 multipoint straps should never be touched!
016	Internal error: HP2645A multipoint terminal mode not correctly set.	See error 015.

VOPENFORMF and VCLOSEFORMF Messages

Error	Message	Cause/Action
040	Internal error: DLSIZE failed; there is not enough DL area space.	Stack size or MAXDATA too small. Re-PREP your application program with a larger MAXDATA parameter.
041	Internal error: DLSIZE failed; the stack is frozen.	Stack size or MAXDATA too small. Re-PREP your application program with a larger STACK= parameter.
045	TERMINAL or FORMSFILE has not yet been opened.	
050	Forms file open failed. (FSERR !!!!)	FOPEN intrinsic failed. Check forms file name.
051	The file is not a V/3000 forms file.	Check the formfile parameter to make sure the file name is correct.
052	Internal error: Forms file FGETINFO failed. (FSERR !!!!)	FGETINFO intrinsic failed.
053	Forms File probably hasn´t been compiled. (FSERR !!!!)	Run FORMSPEC and compile the forms file.
060	The program supplied COMAREA extension is too small.	If you are using BASIC, check word 4 of COMAREA (USRBUFLEN), and increase the length of the COMAREA extension.
068	Internal error: there is not enough DL area space.	For languages other than BASIC, V/3000 uses the DL area as the COMAREA extension. See error 040.
069	Internal error: the stack is frozen.	See error 041.

VOPENBATCH and VCLOSEBATCH Messages

Error	Message	Cause/Action
070	Form file was re-compiled since last using this batch.	You may want to specify a different batch file name since the forms file has been changed since the specfied batch file was last used with the forms file.
071	A forms file must be opened before opening a batch file.	Call VOPENFORMF before calling VOPENBATCH.
072	The file does not appear to be an existing batch file.	Check the batch file name you specified; if this is a new file, it will be created automatically and then opened.
073	**A different forms file created this batch.**	**You opened a forms file that is the not the forms file associated with the batch file specified. See 070.**
074	**Batch file not yet opened.**	
080	Batch file open failed. (FSERR !!!!)	Refer to the file system error to determine the cause of the open failure. Check file name.
081	Can´t read the batch file´s user label. (FSERR !!!!)	FREADLABEL intrinsic failed.
082	Internal error: Batch file FGETINFO failed. (FSERR !!!!)	FGETINFO intrinsic failed.
084	Can´t write user label to batch file. (FSERR !!!!)	FWRITELABEL intrinsic failed.
085	Internal error: Batch file close failed. (FSERR !!!!)	FCLOSEBATCH intrinsic failed. Refer to the file system error to determine the cause of the close failure.

Access Method Errors		Cause/Action
Error	Message	
086	Access Method: File code error.	
087	Access Method: Attempt to add a duplicate key.	
088	Access Method: Internal error: Key insertion location error.	
089	Access Method: Internal error: Block not present.	
090	Access Method: Attempt to open an old file as new.	
091	Access Method: Internal error: Key block buffer error.	
092	Access Method: Internal error: Invalid key block number.	
093	Access Method: Internal error: Block not verified.	
094	Access Method: Record not found.	
095	Access Method: Not enough space in extension for the directory.	Fast form files require 500 words of space in the COMAREA extension. If you are coding in BASIC, list the forms file to find how many words are needed for the extension, and increase the COMAREA field USERBUFLEN to this number.
096	Access Method: Internal error: Parent block not found.	
097	Access Method: Internal error: Illegal entry number.	
098	Access Method: The file is not a KSAMless forms/ref file.	
099	Access Method: The file is at EOF.	

VGETNEXTFORM Messages

Error	Message	Cause/Action
100	Can´t find the next form. (FSERR !)	Check form name in application program.
101	Form $HEAD does not apply, since no forms have been displayed.	NFNAME in COMAREA is $HEAD, but this specification only applies when forms have been displayed.
102	An invalid COM´REPEAT value was supplied programmatically.	REPEATAPP in COMAREA should be set to one of the codes: 0 - no repeat/append 1 - repeat current form 2 - repeat & append current form
103	An invalid COM´NFOPT value was supplied programmatically	The next form option in FREEZAPP of COMAREA can be: 0 - clear current form, display next form 1 - append next form to current form 2 - freeze current form, append next form

VSHOWFORM Messages

Error	Message	Cause/Action
130	Internal error: Terminal write failed. (FSERR !)	FWRITE intrinsic failed.

VREADFIELDS Messages

150	The supplied read buffer (dbuf) is too small.	The data buffer in memory is too small for the data on the form. Check application program DBUF array size and DBUFLEN.
151	Internal error: An expected DC2 from the terminal was missing.	Block mode or data comm problem. Press REFRESH.
152	Internal error: A read terminator (an RS or GS) was expected.	Press ENTER again. Also try REFRESH. If problem recurs, check DBUFLEN in application program. Possible block mode or data comm problem.
153	A status request was pending, so the terminal read was invalid.	Application program requested terminal status, then called VREADFIELDS.
154	An unrecognized escape sequence was read.	Possible softkey problem. Press REFRESH or perhaps RESET to correct.
155	Read length error.	Length of data actually read is less than expected for current form.
160	Internal error. Terminal read failed. (FSERR !)	FREAD intrinsic failed.

VPRINTFORM Messages

Error	Message	Cause/Action
190	Can´t open the print file. (FSERR !)	FOPEN intrinsic failed. Refer to file system error to determine the cause.
191	Can´t write to the print file. (FSERR !)	FWRITE intrinsic failed.

VINITFORM, VFIELDEDITS, and VFINISH Messages

These messages result from checking the data entered in the current field. Cursor is positioned to the field whose processing specifications were being executed when the error was detected.

Error	Message	Cause/Action
202	The required field is empty.	A required field in current form is blank.
203	Invalid field type.	Valid field types are: O(optional), R(required), D(display only, P(Process even if blank)
204	The DIG value can only contain digits.	The field is type DIG, but contains characters other than 0-9.
205	The NUM value can only contain a number.	A NUM type field has characters other than 0-9, a decimal point, comma, or sign.
206	The NUMn value can only contain a number with max n decimal places.	Reduce the number of digits after the decimal point to n.
207	The IMPn value can only contain a number, with max n decimal places.	The data in an IMPn field is not a number (0-9,decimal point,sign,comma) or has more than n digits after an explicit decimal point.
208	The DMY value must be a valid date, in DMY order.	The data in a DMY field is not a valid date, or is not in order: day, month, year.
209	The MDY value must be a valid date, in MDY order.	The data in an MDY field is not a valid date, or is not in order: month, day, year.
210	The YMD value must be a valid date, in YMD order.	The data in a YMD field is not a valid date, or is not in order: year, month, day.
211	Internal error: Invalid field data type.	Field table problem.
220	Internal error: Invalid destination.	Code execution problem.
222	An edit test failed.	The default field edit error

		message when no custom error message had been provided. Check the data you entered in the field against the processing specifications.
321	Attempted a SET into a numeric field that is too short.	The destination of a SET statement has fewer characters than the number being moved to it.
323	Internal error: Invalid op code.	Code execution problem.
325	Attempted a SET into a data field that is too short.	The destination of a SET statement has fewer characters than the value being moved to it.
326	Internal error in field.	
327	User-defined error for this field.	A FORMSPEC custom error message is associated with this field, but could not be correctly retrieved from forms file.
328	The field cannot contain a negative number (Dtype DIG).	A minus sign is in a field whose data type is DIG; only digits (0-9) allowed.
330	Internal error: invalid op in logical eval.	Code execution problem.
331	Range error: Low value is greater than high value.	The first value in an IN/NIN range is greater than the second value (for a:b, a>b)
332	Check digit requested on empty field.	CDIGIT is specified for a field that contains no data. There must be a value in order to test the check digit.
333	Check digit requested on field containing only + or - sign.	CDIGIT is specified for a field that has no value except a sign; there must be a value in order to test the check digit.
334	Check digit requested on field containing special characters.	CDIGIT only operates on numeric (0-9) or alphabetic (A-Z) values.
335	Check digit 10 is invalid for modulus 11 calculation.	Since value has remainder greater than 9, it has no possible modulus 10 check digit. Use CDIGIT 11.

336	Internal error: Check digit modulus must be 10 or 11.	
340	Division by zero was attempted.	Expression in field processing statement evaluated to division by zero.
341	Attempted a SET from a field containing an invalid number.	A numeric value is expected as the source of a SET statement, but the source is not a valid number.
344	Internal error: Invalid op in long eval.	
345	Overflow in add operation.	Value is outside 4-word REAL range. Range of 4-word REAL is from approximately 8.636 times 10^(-78) to 1.158 times 10^77.
346	Overflow in subtract operation	See error 345.
347	Overflow in multiply operation.	See error 345.
348	Overflow in divide operation.	See error 345.
349	Overflow in % operation.	See error 345.
350	Index expression out of range in an "index of" numeric statement.	In an index retrieve operand (index OF element,element...) the index evaluates to a digit greater than the number of elements in the list. (Numeric operands).
351	Overflow in negate operation.	See error 345.
360	Attempted a SET from a field containing an invalid date.	A date is expected as the source of a SET statement, but the date is not valid.
363	Internal error: Invalid op in double eval.	
364	Index expression out of range in an "index of" date statement.	See error 350. (Date operands)

370	Internal error: Invalid op in text eval.	
371	Index expression out of range in an "index of" text statement.	See error 350. (CHAR operands)
380	The index expression evaluates to "empty".	An arithmetic expression used as the index in an index OF element, element.. operand is blank or zero.

VREADBATCH Messages

Error	Message	Cause/Action
400	Can't read the batch file record. (FSERR !)	FREADDIR intrinsic failed.
401	Warning: Can't browse a batch file with variable length records.	Browse mode is illegal for batch records which are not fixed length.

VWRITEBATCH Messages

Error	Message	Cause/Action
450	Can't write the batch record. (FSERR !)	FWRITEDIR intrinsic failed.

VSETERROR, VGETFIELD, VPUTFIELD, VPUTtype, and VGETtype Messages

Error	Message	Cause/Action
500	A field with the field number supplied does not exist.	The fieldnum parameter contains a value that is not a field number in the current form.
501	The field number supplied is out of range.	The fieldnum parameter contains a value greater than any field number associated with the current form.
502	The field requested is in error.	The field specified by the fieldnum parameter contains invalid data.
503	The field requested is empty.	The field specified by the fieldnum parameter contains no data.
504	Error converting field to numeric type.	VGETtype tried to convert value to a numeric type, and failed. Check conversion rules. (For VGETINT, message indicates attempt to convert number > 32767; receiving field is unchanged.)
505	Error converting numeric type to ASCII.	VPUTtype tried to convert a numeric value to ASCII characters, and failed. Check conversion rules.
506	A DIGIT field cannot contain a negative number.	A value in a DIG type field has a minus sign; DIG values may only be digits (0-9).

Information intrinsic messages

601	Invalid input parameters to information intrinsic.
602	Form specified to FORMINFO does not exist.
603	Info intrinsic: Form field list read error (FSERR !!!!).
604	Info intrinsic: Form record read error (FSERR !!!!).
605	FF version is old. Must recompile to use FORMINFO intrinsic.
606	Invalid field specified to FIELDINFO intrinsic.
607	Invalid form specified to FORMINFO intrinsic.

REFORMAT MESSAGES

These error messages are issued by program REFORMAT. REFORMAT reformats data in a batch file according the specifications in a reformat file and writes the reformatted data to an output file.

Error	Message	Cause/Action
2	Error opening reformat file. (FSERR !!!!)	Check file system error number.
4	Batch file could not be opened. (FSERR !!!!)	Check file system error number.
5	No reformatting was done.	No sequence of forms in batch matched with any Reformat sequence.
6	Batch file: !	Not an error. Listing message.
8	Error closing batch file. (FSERR !!!!)	Check file system error number.
9	Error closing output file. (FSERR !!!!)	
10	Error closing reformat file. (FSERR !!!!)	
11	Error opening output file. (FSERR !!!!)	
13	Cannot append to file with variable length records.	If you specify an existing output file in the :FILE command, and expect to write/append to that file, it must have fixed-length records.
14	Reformat file not compiled or not a legal reformat file.	Check the reformat file name in your :FILE command; you may have to run REFSPEC to compile the reformat file.
15	Form file versions from batch and reformat file don't match.	A warning that forms file was recompiled since batch file was created.
16	Batch data must be from forms file !.	Reformat file expects batch file written from forms in specified forms file. Check that batch and reformat names are correct.

17	Not a legal batch file.	Check batch file name in :FILE command.
18	REFORMAT TERMINATED ABNORM-ALLY	FGETINFO failed. See error message given in your output above this message.
19	Internal error getting info from reformat file. (FSERR !!!!)	
20	Reformat file: !	Not an error.
22	Output file: !	Not an error.
23	Internal error getting in-formation from batch file. (FSERR !!!!)	FGETINFO failed.
24	File specified was not a reformat file.	Check reformat file name in :FILE command.

Allocation Errors

These errors result from attempts to allocate memory for the reformat operation.

Error	Message	Cause/Action
30	Internal error: Batch data space too small. Allocation failed.	
31	Internal error allocating batch information buffer.	
32	Internal error allocating batch data buffer.	
33	Internal error allocating batch temporary buffer.	
34	Internal error allocating reformat code buffer.	
35	Internal error allocating output buffer.	
36	Internal error allocating temporary output buffer.	
37	Internal error allocating data buffer.	

I/O Errors

These errors all result from problems accessing the batch, reformat, or output files. Check the file system error code issued with the message.

Error	Message	Cause/Action
50	Error reading batch record # !!!. (FSERR !!!!)	
51	Error reading from reformat file. (FSERR !!!!)	
52	Error reading from reformat file. (FSERR !!!!)	
54	End of file reached on output file. Record not written.	
55	Error writing to output file. (FSERR !!!!)	
56	Error reading from reformat file. (FSERR !!!!)	

Procedure Errors

Error	Message	Cause/Action
73	Internal error: Translation to EBCDIC failed.	Intrinsic failed.

Access Method Errors

Error	Message	Cause/Action
086	Access Method: File code error.	
087	Access Method: Attempt to add a duplicate key.	
088	Access Method: Internal error: Key insertion location error.	
089	Access Method: Internal error: Block not present.	

090	Access Method: Attempt to open an old file as new.	
091	Access Method: Internal error: Key block buffer error.	
092	Access Method: Internal error: Invalid key block number.	
093	Access Method: Internal error: Block not verified.	
094	Access Method: Record not found.	
095	Access Method: Not enough space in extension for the directory.	Fast form files require 500 words of space in the COMAREA extension. If you are coding in BASIC, list the forms file to find how many words are needed for the extension, and increase the COMAREA field USERBUFLEN to this number.
096	Access Method: Internal error: Parent block not found.	
097	Access Method: Internal error: Illegal entry number.	
098	Access Method: The file is not a KSAMless forms/ref file.	
099	Access Method: The file is at EOF.	

Code Interpretation Errors

Error	Message	Cause/Action
101	Record terminator causes output record overflow. Data truncated.	End-of-record marker makes record longer than output record length defined on GLOBALS menu for reformat file.
102	Field separator causes output record overflow. Data Truncated.	See error 101. See globals menu.
103	Literal causes output record overflow.	A literal constant in output record makes record longer than output record length defined for file in GLOBALS menu.
104	Internal error: Bad code for reformat sequence.	
105	Output length greater than largest possible output record.	Length of actual output record is longer than largest variable length record as defined on GLOBALS menu for file.
106	Field causes output record overflow. Data truncated.	An output field causes the output record to be longer than the record length defined on the GLOBALS menu.
107	Field causes output record overflow. Field data not written.	In this case, the entire field is omitted from the output record. (For error 106, the field is written with data truncated.)
108	Field causes overwrite on previous data. Field data not written.	The beginning of this field overlaps a previous field. Check OUTPUT RECORD specification for reformat file.
109	System constant $GS causes record overflow. Data truncated.	Group separator in output record makes record longer than maximum record size defined for output file.
110	System constant $RS causes record overflow. Data truncated.	Record separator in output record makes record longer than maximum record size defined for output file.

111	System constant $US causes record overflow. Data truncated.	Unit separator in output record makes record longer than maximum record size defined for output file.
112	System constant $LF causes record overflow. Data truncated.	Line feed specified in output record makes record longer than maximum record size defined for output file.
113	System constant $CR causes record overflow. Data truncated.	Carriage return specified in output record makes record longer than maximum record size defined for output file.
114	$ASCII character causes output record overflow. Data trunctated.	The numeric equivalent of an ASCII character in the output record makes record longer than maximum record size defined for output file.

Statistics Messages

Error	Message	Cause/Action
200	Reformatting completed successfully with no errors.	Hurrah!
201	!!!! reformats attempted.	Message reports how many times reformat identifiers were matched with records in the reformat file.
202	!!! records written to output file.	This message tells you the total number of reformatted records written by REFORMAT. Message is always issued.
203	!!! batch records processed.	This message tells you the total number of records in the batch file that were processed by REFORMAT.
204	!!!! errors.	The total number of errors detected by REFORMAT.
205	Some errors were found, see above. REFORMAT terminated normally.	This message is issued after message 204 when REFORMAT completes despite errors.

REFORMAT Errors

Error	Message	Cause/Action
300	Invalid numeric data.	Numeric data expected for output field, but data in batch field not numeric.
301	Numeric data conversion failed.	Conversion of numeric data from batch field to output field not successful. Check that OUTPUT FIELD is a numeric type, that batch field contains numeric data.
302	Numeric data conversion failed. Output field too small.	Output field defined in reformat file not large enough for numeric data in batch field.
303	Invalid data for digit data type.	Output field defined as type DIG, but data in batch field is not solely digits.
304	Numeric data conversion failed.	See error 301
305	Numeric data conversion failed. Output field too small.	See error 302.
306	Date conversion failed.	Conversion of a date from batch field to output field not successful. Check that OUTPUT FIELD is a date type, that batch field contains a date.
307	Date conversion failed. Output field too small.	The output field is not large enough to contain the date in the batch field. Check field size in OUTPUT RECORD menu, check batch field.
308	Invalid data in date input field.	Output field expects a date, but batch field does not contain a date.
309	No data movement. Truncation of significant data would occur.	The numeric data is not written to the output field because it would be truncated.

315	Non-numeric characters cannot be converted to zone decimals.	Output field defined as SIGN = Z, but batch field does not contain numeric data.
316	No sign inserted. Truncation of significant data would occur.	Output field defined as SIGN = F, L, P, or Z, but inserting sign would cause value in batch field to be truncated.
317	Data truncated for output record # !!!.	Data written to the specified output record has been truncated. Presumably, the data is not significant.
318	No reformatting done for the following:	The listed (fields/records) in the batch file have not written to the output file.
319	Insertion of checkdigit would cause loss of significant data.	The output field into which a check digit is to be inserted is not large enough to contain the extra digit. The check digit is not added.
320	Illegal character in field. Checkdigit not generated.	In performing the check digit calculations, a character other than a letter (A-Z) or a digit (0-9) was found.
321	Internal error: More than 9 decimal places expected.	Batch data. Involves NUM(n) field where n is valued internally to be greater than 9.
322	All blank numeric field cannot be reformatted.	A batch field containing blanks cannot be converted to a numeric output field.
323	Commas in numeric data field not correctly positioned.	Any commas in a numeric batch field must be correct; otherwise value cannot be converted to numeric output field.
324	Numeric data has more than the expected number of decimal places.	Numeric batch field has more decimal digits than defined for the output field. Output field is replaced with blanks.
325	Invalid character in numeric data field.	Batch field to be converted to numeric output field contains characters other than digits, sign, decimal point, or commas.

326	Invalid character in digit data field.	Batch field to be converted to digit type output field contains characters other than digits. Output field replaced with blanks.
327	Only a plus or minus sign was found in numeric field.	Batch field to be converted to numeric output field contains only a sign. REFORMAT replaces the output field with blanks.
328	Internal error: More than 9 decimal places expected.	See error 321
329	An all blank field is not a legal implied value.	A batch field to be converted to an IMPn type output field must contain values other than blanks.
330	Commas in implied data field are not correctly positioned.	In an IMPn type field, commas should be positioned counting back from the implied decimal point.
331	Implied data has more than the expected number of decimal places.	IMPn type batch field has more than n digits to the right of an actual decimal point. Output field replaced with blanks.
332	Illegal character found in implied field.	IMPn type batch field contains character other than a digit, decimal point, sign, or commas.
333	Only a plus or minus sign was found in implied field.	IMPn type batch field contains only a sign; cannot convert to numeric output field.
334	IMPn field must have at least !!!! digits.	IMPn type batch field has fewer than n digits.
335	Illegal implied field data.	IMPn type batch field has data other than digits, decimal point, sign, or commas.
336	Check digit requested on empty field.	Cannot generate check digit from blank batch field.
337	Check digit requested on field containing only + or - sign.	Cannot generate check digit from sign only.

338	Check digit requested on field containing special characters.	Can only generate check digit from digits (0-9) or letters (A-Z).
339	Check digit 10 is invalid for modulus 11 calculation.	This number or string is inappropriate for modulus 11 check digit calculation because that process yields a value of 10 when only single digit results can be used.
340	Check digit modulus must be 10 or 11.	Check OUTPUT FIELD in reformat file; only modulus 10 or 11 check digits can be generated.
341	Check digit generation failed.	See appendix D. Check digits cannot be calculated for some fields.

Message Info

Error	Message	Cause/Action
400	Reformat identifier: !	
401	Reformat field name: !	
402	Batch record form name: !	
403	Batch record #: !!!	
404	Output record #: !!!	
405	Output record col#: !!!!	

Header Messages

```
Error    Message

500      ***********************************************************
501      *                                                         *
502      *                  REFORMAT HP32209!                      *
503      *                                                         *
504      Page !!!!
505      !
506
507      REFORMAT/3000 HP32209!
```

TESTLIST Errors

These errors result from attempting to use the TESTLIST file.

Error	Message	Cause/Action
640	Error writing to testlist file. (FSERR !!!)	
641	EOF reached on testlist file.	
642	Error opening testlist file. (FSERR !!!)	
643	Error allocating testlist buffer.	

ASCII CHARACTER SET IN COLLATING SEQUENCE

APPENDIX C

HOW TO USE THIS TABLE

- The table is sorted by character code, each code being represented by its decimal, octal, and hexadecimal equivalent.
- Each row of the table gives the ASCII and EBCDIC meaning of the character code, the ASCII↔EBCDIC conversion code, and the Hollerith representation (punched card code) for the ASCII character.

The following examples describe several ways of using the table:

Example 1: Suppose you want to determine the ASCII code for the $ character. Scan down the ASCII graphic column until you locate $, then look left on that row to find the character code – 36 (dec), 044 (oct), and 24 (hex). This is the code used by an ASCII device (terminal, printer, computer, etc.) to represent the $ character. Its Hollerith punched card code is 11-3-8.

Example 2: The character code 5B (hex) is the EBCDIC code for what character? Also, when 5B is converted to ASCII (for example, by FCOPY with the EBCDICIN option), what is the octal character code? First, locate 5B in the hex character code column and move right on that row to the EBCDIC graphic which is $. The next column to the right gives the conversion to ASCII, 044. As a check, find 044 (oct) in the character code column, look right to the ASCII graphic column and note that $ converted to EBCDIC is 133 (oct) which equals 5B (hex).

CHAR CODE			ASCII			EBCDIC	
Dec	Oct	Hex	Cntl/ Gph	to EBCDIC (Oct)	Hollerith	Cntl/ Gph	to ASCII (Oct)
0	000	00	NUL	000	12-0-1-8-9	NUL	000
1	001	01	SOH	001	12-1-9	SOH	001
2	002	02	STX	002	12-2-9	STX	002
3	003	03	ETX	003	12-3-9	ETX	003
4	004	04	EOT	067	7-9	PF	234
5	005	05	ENQ	055	0-5-8-9	HT	011
6	006	06	ACK	056	0-6-8-9	LC	206
7	007	07	BEL	057	0-7-8-9	DEL	177
8	010	08	BS	026	11-6-9		227
9	011	09	HT	005	12-5-9		215
10	012	0A	LF	045	0-5-9	SMM	216
11	013	0B	VT	013	12-3-8-9	VT	013
12	014	0C	FF	014	12-4-8-9	FF	014
13	015	0D	CR	015	12-5-8-9	CR	015
14	016	0E	SO	016	12-6-8-9	SO	016
15	017	0F	SI	017	12-7-8-9	SI	017
16	020	10	DLE	020	12-11-1-8-9	DLE	020
17	021	11	DC1	021	11-1-9	DC1	021
18	022	12	DC2	022	11-2-9	DC2	022
19	023	13	DC3	023	11-3-9	TM	023
20	024	14	DC4	074	4-8-9	RES	235
21	025	15	NAK	075	5-8-9	NL	205
22	026	16	SYN	062	2-9	BS	010
23	027	17	ETB	046	0-6-9	IL	207
24	030	18	CAN	030	11-8-9	CAN	030
25	031	19	EM	031	11-1-8-9	EM	031
26	032	1A	SUB	077	7-8-9	CC	222
27	033	1B	ESC	047	0-7-9	CU1	217
28	034	1C	FS	034	11-4-8-9	IFS	034
29	035	1D	GS	035	11-5-8-9	IGS	035
30	036	1E	RS	036	11-6-8-9	IRS	036
31	037	1F	US	037	11-7-8-9	IUS	037
32	040	20	SP	100	Blank	DS	200
33	041	21	!	117	12-7-8	SOS	201
34	042	22	"	177	7-8	FS	202
35	043	23	#	173	3-8		203
36	044	24	$	133	11-3-8	BYP	204
37	045	25	%	154	0-4-8	LF	012
38	046	26	&	120	12	ETB	027
39	047	27	'	175	5-8	ESC	033
40	050	28	(	115	12-5-8		210
41	051	29	)	135	11-5-8		211
42	052	2A	*	134	11-4-8	SM	212
43	053	2B	+	116	12-6-8	CU2	213
44	054	2C	,	153	0-3-8		214
45	055	2D	-	140	11	ENQ	005
46	056	2E	.	113	12-3-8	ACK	006
47	057	2F	/	141	0-1	BEL	007
48	060	30	0	360	0		220
49	061	31	1	361	1		221
50	062	32	2	362	2	SYN	026
51	063	33	3	363	3		223
52	064	34	4	364	4	PN	224
53	065	35	5	365	5	RS	225
54	066	36	6	366	6	UC	226
55	067	37	7	367	7	EOT	004
56	070	38	8	370	8		230
57	071	39	9	371	9		231
58	072	3A	:	172	2-8		232
59	073	3B	;	136	11-6-8	CU3	233
60	074	3C	<	114	12-4-8	DC4	024
61	075	3D	=	176	6-8	NAK	025
62	076	3E	>	156	0-6-8		236
63	077	3F	?	157	0-7-8	SUB	032
64	100	40	@	174	4-8	SP	040
65	101	41	A	301	12-1		240
66	102	42	B	302	12-2		241
67	103	43	C	303	12-3		242
68	104	44	D	304	12-4		243
69	105	45	E	305	12-5		244
70	106	46	F	306	12-6		245
71	107	47	G	307	12-7		246
72	110	48	H	310	12-8		247
73	111	49	I	311	12-9		250
74	112	4A	J	321	11-1	¢	133
75	113	4B	K	322	11-2	.	056
76	114	4C	L	323	11-3	<	074
77	115	4D	M	324	11-4	(	050
78	116	4E	N	325	11-5	+	053
79	117	4F	O	326	11-6	|	041
80	120	50	P	327	11-7	&	046
81	121	51	Q	330	11-8		251
82	122	52	R	331	11-9		252
83	123	53	S	342	0-2		253
84	124	54	T	343	0-3		254
85	125	55	U	344	0-4		255
86	126	56	V	345	0-5		256
87	127	57	W	346	0-6		257
88	130	58	X	347	0-7		260
89	131	59	Y	350	0-8		261
90	132	5A	Z	351	0-9	!	135
91	133	5B	[	112	12-2-8	$	044
92	134	5C	\	340	0-2-8	*	052
93	135	5D	]	132	11-2-8	)	051
94	136	5E	^	137	11-7-8	;	073
95	137	5F	_	155	0-5-8	¬	136

CHAR CODE			ASCII		EBCDIC		
Dec	Oct	Hex	Cntl/ Gph	to EBCDIC (Oct)	Hollerith	Cntl/ Gph	to ASCII (Oct)
96	140	60	`	171	1-8	-	055
97	141	61	a	201	12-0-1	/	057
98	142	62	b	202	12-0-2		262
99	143	63	c	203	12-0-3		263
100	144	64	d	204	12-0-4		264
101	145	65	e	205	12-0-5		265
102	146	66	f	206	12-0-6		266
103	147	67	g	207	12-0-7		267
104	150	68	h	210	12-0-8		270
105	151	69	i	211	12-0-9		271
106	152	6A	j	221	12-11-1	¦	174
107	153	6B	k	222	12-11-2	,	054
108	154	6C	l	223	12-11-3	%	045
109	155	6D	m	224	12-11-4	_	137
110	156	6E	n	225	12-11-5	>	076
111	157	6F	o	226	12-11-6	?	077
112	160	70	p	227	12-11-7		272
113	161	71	q	230	12-11-8		273
114	162	72	r	231	12-11-9		274
115	163	73	s	242	11-0-2		275
116	164	74	t	243	11-0-3		276
117	165	75	u	244	11-0-4		277
118	166	76	v	245	11-0-5		300
119	167	77	w	246	11-0-6		301
120	170	78	x	247	11-0-7		302
121	171	79	y	250	11-0-8	`	140
122	172	7A	z	251	11-0-9	:	072
123	173	7B	{	300	12-0	#	043
124	174	7C	¦	152	12-11	@	100
125	175	7D	}	320	11-0	'	047
126	176	7E	~	241	11-0-1	=	075
127	177	7F	DEL	007	12-7-9	"	042
128	200	80		040	11-0-1-8-9		303
129	201	81		041	0-1-9	a	141
130	202	82		042	0-2-9	b	142
131	203	83		043	0-3-9	c	143
132	204	84		044	0-4-9	d	144
133	205	85		025	11-5-9	e	145
134	206	86		006	12-6-9	f	146
135	207	87		027	11-7-9	g	147
136	210	88		050	0-8-9	h	150
137	211	89		051	0-1-8-9	i	151
138	212	8A		052	0-2-8-9		304
139	213	8B		053	0-3-8-9		305
140	214	8C		054	0-4-8-9		306
141	215	8D		011	12-1-8-9		307
142	216	8E		012	12-2-8-9		310
143	217	8F		033	11-3-8-9		311
144	220	90		060	12-11-0-1-8-9		312
145	221	91		061	1-9	j	152
146	222	92		032	11-2-8-9	k	153
147	223	93		063	3-9	l	154
148	224	94		064	4-9	m	155
149	225	95		065	5-9	n	156
150	226	96		066	6-9	o	157
151	227	97		010	12-8-9	p	160
152	230	98		070	8-9	q	161
153	231	99		071	1-8-9	r	162
154	232	9A		072	2-8-9		313
155	233	98		073	3-8-9		314
156	234	9C		004	12-4-9		315
157	235	9D		024	11-4-9		316
158	236	9E		076	6-8-9		317
159	237	9F		341	11-0-1-9		320
160	240	A0		101	12-0-1-9		321
161	241	A1		102	12-0-2-9	~	176
162	242	A2		103	12-0-3-9	s	163
163	243	A3		104	12-0-4-9	t	164
164	244	A4		105	12-0-5-9	u	165
165	245	A5		106	12-0-6-9	v	166
166	246	A6		107	12-0-7-9	w	167
167	247	A7		110	12-0-8-9	x	170
168	250	A8		111	12-1-8	y	171
169	251	A9		121	12-11-1-9	z	172
170	252	AA		122	12-11-2-9		322
171	253	AB		123	12-11-3-9		323
172	254	AC		124	12-11-4-9		324
173	255	AD		125	12-11-5-9		325
174	256	AE		126	12-11-6-9		326
175	257	AF		127	12-11-7-9		327

CHAR CODE			ASCII		EBCDIC		
Dec	Oct	Hex	Cntl/ Gph	to EBCDIC (Oct)	Hollerith	Cntl/ Gph	to ASCII (Oct)
176	260	B0		130	12-11-8-9		330
177	261	B1		131	11-1-8		331
178	262	B2		142	11-0-2-9		332
179	263	B3		143	11-0-3-9		333
180	264	B4		144	11-0-4-9		334
181	265	B5		145	11-0-5-9		335
182	266	B6		146	11-0-6-9		336
183	267	B7		147	11-0-7-9		337
184	270	B8		150	11-0-8-9		340
185	271	B9		151	0-1-8		341
186	272	BA		160	12-11-0		342
187	273	BB		161	12-11-0-1-9		343
188	274	BC		162	12-11-0-2-9		344
1B9	275	BD		163	12-11-0-3-9		345
190	276	BE		164	12-11-0-4-9		346
191	277	BF		165	12-11-0-5-9		347
192	300	C0		166	12-11-0-6-9	{	173
193	301	C1		167	12-11-0-7-9	A	101
194	302	C2		170	12-11-0-8-9	B	102
195	303	C3		200	12-0-1-8	C	103
196	304	C4		212	12-0-2-8	D	104
197	305	C5		213	12-0-3-8	E	105
198	306	C6		214	12-0-4-8	F	106
199	307	C7		215	12-0-5-8	G	107
200	310	C8		216	12-0-6-8	H	110
201	311	C9		217	12-0-7-8	I	111
202	312	CA		220	12-11-1-8		350
203	313	CB		232	12-11-2-8		351
204	314	CC		233	12-11-3-8	ʃ	352
205	315	CD		234	12-11-4-8		353
206	316	CE		235	12-11-5-8	⑂	354
207	317	CF		236	12-11-6-8		355
208	320	D0		237	12-11-7-8	}	175
209	321	D1		240	11-0-1-8	J	112
210	322	D2		252	11-0-2-8	K	113
211	323	D3		253	11-0-3-8	L	114
212	324	D4		254	11-0-4-8	M	115
213	325	D5		255	11-0-5-8	N	116
214	326	D6		256	11-0-6-8	O	117
215	327	D7		257	11-0-7-8	P	120
216	330	D8		260	12-11-0-1-8	Q	121
217	331	D9		261	12-11-0-1	R	122
218	332	DA		262	12-11-0-2		356
219	333	DB		263	12-11-0-3		357
220	334	DC		264	12-11-0-4		360
221	335	DD		265	12-11-0-5		361
222	336	DE		266	12-11-0-6		362
223	337	DF		267	12-11-0-7		363
224	340	E0		270	12-11-0-8	\	134
225	341	E1		271	12-11-0-9		237
226	342	E2		272	12-11-0-2-8	S	123
227	343	E3		273	12-11-0-3-8	T	124
228	344	E4		274	12-11-0-4-8	U	125
229	345	E5		275	12-11-0-5-8	V	126
230	346	E6		276	12-11-0-6-8	W	127
231	347	E7		277	12-11-0-7-8	X	130
232	350	E8		312	12-0-2-8-9	Y	131
233	351	E9		313	12-0-3-8-9	Z	132
234	352	EA		314	12-0-4-8-9		364
235	353	EB		315	12-0-5-8-9		365
236	354	EC		316	12-0-6-8-9	⑁	366
237	355	ED		317	12-0-7-8-9		367
238	356	EE		332	12-11-2-8-9		370
239	357	EF		333	12-11-3-8-9		371
240	360	F0		334	12-11-4-8-9	0	060
241	361	F1		335	12-11-5-8-9	1	061
242	362	F2		336	12-11-6-8-9	2	062
243	363	F3		337	12-11-7-8-9	3	063
244	364	F4		352	11-0-2-8-9	4	064
245	365	F5		353	11-0-3-8-9	5	065
246	366	F6		354	11-0-4-8-9	6	066
247	367	F7		355	11-0-5-8-9	7	067
248	370	F8		356	11-0-6-8-9	8	070
249	371	F9		357	11-0-7-8-9	9	071
250	372	FA		372	12-11-0-2-8-9	|	372
251	373	FB		373	12-11-0-3-8-9		373
252	374	FC		374	12-11-0-4-8-9		374
253	375	FD		375	12-11-0-5-8-9		375
254	376	FE		376	12-11-0-6-8-9		376
255	377	FF		377	12-11-0-7-8-9	EO	377

CHECK DIGIT CALCULATION

APPENDIX D

Check digits are digits added to the end of a value that contains only numbers (0-9) or letters of the alphabet (A-Z). A FORMSPEC option can be selected that causes a modulus check to be made on a value entered at the terminal.

Whenever a value is entered for which check-digit verification has been specified, a modulus calculation is performed. The result of a modulus calculation is a single digit that must match the last digit (the "check digit") of the entered value. If the match is not successful, an error is diagnosed. Note that for modulus checks to be meaningful, the last digit verified by such a check must have been calculated by the same method used for the check.

REFSPEC has an option that lets you add a check digit to an entered value. The check digit is added when program REFORMAT is run. Thus, the check digit is added after the value is entered at the terminal but before it is input to the application program. In this case, the modulus calculation is made programmatically.

The modulus calculations used by V/3000 are either modulus 10 or modulus 11. A value is checked by one or the other of these calculations, depending on which was used to add the check digit. Thus, a value whose check digit was added using modulus 10 can be checked only by modulus 10; and a value with a modulus 11 check digit can be checked only with modulus 11 calculations.

MODULUS 10

Modulus 10 calculations detect single transpositions and incorrect keying of a single digit. The calculation is performed as follows:

- The units position and every alternate position of the basic number is multiplied by 2.
- The digits resulting from the multiplication are added to those that were not multiplied.
- The total is subtracted from the next higher number ending with zero. The result is the check digit.

For example:

Assume a basic number	9 6 4 3 8	
Unit and every alternate	9 4 8	
Multiplied by 2	18 8 16	
Digits not multiplied	6 3	
Add them together	1+8+6+8+3+1+6 =	33
Next higher number ending in zero		40
Subtract sum of digits		–33
Check digit=		7

The base number with check digit = 964387

MODULUS 11

Modulus 11 detects single digit errors, single transpositions, and double transpositions. Unlike other check-digit systems, it is based on a weighted checking factor for each digit in the basic number. The modulus 11 check digit is obtained as follows:

- Each digit position of the basic number is assigned a weighted checking factor. The following factors are assigned, starting with the units digit and progressing toward the high-order digit:

  2 3 4 5 6 7 2 3 4 5 6 7 2 3 4 . . .

- Each digit in the basic number is multiplied by its checking factor.

- The products are summed and then divided by 11. The remainder is subtracted from 11. The result is the check digit.

For example:

Assume a basic number	5 1 6 1 9 2 8 7 2
Checking factors	4 3 2 7 6 5 4 3 2
Add the products	20+3+12+7+54+10+32+21+4 = 163
Divide by 11	163/11 = 14 with remainder = 9
Subtract remainder from 11	11 – 9 = 2
Check digit =	2
Self-checking number	516192872$\underline{2}$

ALPHABETIC CHECK DIGITS

Letters of the alphabet are treated like numbers in either modulus 10 or modulus 11 check digit calculations. This is done by assigning a digit to each letter of the alphabet, as follows:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	△	(△ = space)
1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9	2	3	4	5	6	7	8	9	0	

A value to be checked may be preceded by a plus or minus sign, however, the sign is ignored in the check-digit calculations.

To illustrate, perform a modulus 10 check digit calculation on a value that contains letters as well as numbers:

Assume a basic value	3 4 G 1 2 H
Change the letters to digits	3 4 7 1 2 8
Multiply alternate digits by 2	8 2 16
Digits not multiplied	3 7 2
Sum with digits that were not multiplied	3+8+7+2+2+1+6 = 29
Subtract sum from next higher number ending with zero	30 – 29 = 1
Check digit =	1
Self-checking number	34G12H1

NOTE

If a check digit is generated using modulus 11 calculations and the result is 10, the check digit cannot be used and an error is returned. Modulus 11 check digits are the remainder from dividing the product of the calculations by 11 (see example above). Thus, if check digits are being generated for a continuous series of numbers, every eleventh number must be skipped to avoid this error.

If the product generated through the modulus 11 calculations is evenly divisible by 11 (no remainder), the resulting check digit is 11. In this case, the digit 0 is appended to the basic number.

To summarize, if the calculated check digit is 10, an error is returned; if the calculated check digit is 11, a zero is appended to the basic number.

When you attempt to add a modulus 11 check digit that evaluates to 10, the reformatter issues the message: "Check digit is invalid for modulus 11 calculation".

If ENTRY checks a field according to the FORMSPEC statement CDIGIT 11 and that field contains a value with a check digit that evaluates to 10, the same message is issued.

APPLICATION HINTS

APPENDIX E

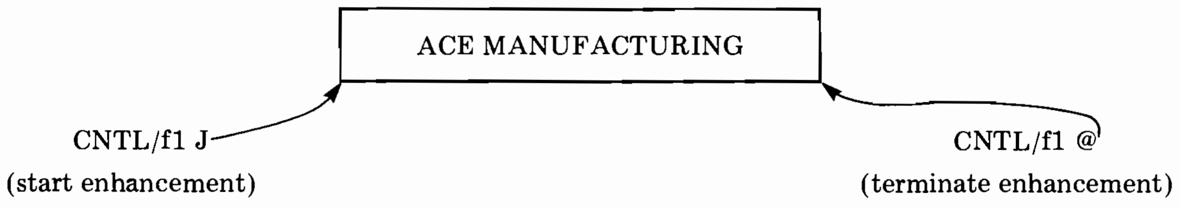
The following hints may help you design your forms file with FORMSPEC:

1. If the status/window line is the last line on the screen, it should be no longer than 79 characters. This is because using the 80th character on the bottom line of the screen causes the screen to roll up one line temporarily. If this roll-up is not a problem, then the last line can use 80 characters.

2. When a form is to be appended, column 80 of any line of the form with an unprotected field should not be used. When the next form is displayed, column 80 of the current form disappears.

3. If you want to enhance text (the protected areas of the screen), you must enter the enhancements with the terminal enhancement keys. On most terminals CNTL/f1 allows you to enter the particular enhancement you want. For example, to initiate an inverse video, half bright enhancement, press CNTL/f1 followed by the letter J; to terminate the enhancement, press CNTL/f1 followed by an "at" sign (@). (Enhancement codes are listed under WINDOWENH in the description of COMAREA in section VI, or in your terminal user's manual.)

   Any enhancement is terminated automatically after the last non-blank character of a line. If you want to extend the enhancement past the last non-blank character, you must use the enhancement terminator. For example, if you want to enhance a heading including blanks on either side, specify the enhancements when you design your screen as follows:



4. If you want blank lines at the bottom of a form, you must assign them a null enhancement with the terminal control keys. Otherwise, FORMSPEC deletes trailing blank lines from a form. A null enhancement is specified by pressing the following keys: CNTL/f1 followed by @.

5. If you use save fields to accumulate totals and you expect the user will browse and correct the fields used for summation, special care must be taken to insure that the totals are accurate. If an entered value is summed into a save field, and then this value is changed in browse/modify mode, the new value is also summed into the save field unless you specify the SET statement to account for this possibility.

   For example, assume you are accumulating values entered in a field "F1" into a save field used for batch totals "BT". In order to allow only the correct values to be accumulated, construct a display-only field "OLDF1" identical in its characteristics to "F1". OLDF1 is initialized to zero in collect mode only, but not in browse mode. In browse mode, OLDF1 contains the previous value of F1. This previous value is then subtracted from the sum in BT. The following specifications are entered in the F1 field menu to insure that only the correct values are summed:

```
INIT

  SET OLDF1 TO 0  ◄──────────────────────  executed only in collect mode

FIELD

  <any edit statements>      }
  SET BT TO BT + F1 - OLDF1  } ◄──────────  executed in browse or collect mode
  SET OLDF1                  }
```

6. If you want to include a logical AND function in your processing specifications, use nested IF statements. For example, to get the effect of IF A=B *AND* C=D *AND* E=F use the following statements:

```
IF A=B THEN
  IF C=D THEN
    IF E=F THEN
```

To negate an IF condition, simply use the ELSE part with a null THEN part.

A logical OR or NOR cannot be similarly specified, but for comparisons on a single field, use IN or NIN with a list. For example, to get the effect of IF A=B OR A=D OR A=F, use the edit statement:

```
A IN B, D, F
```

7. Sometimes in data entry applications, the operator would like to knowingly enter a value that normally fails the designer's edits. This is called "Edit Override", and can be implemented through FORMSPEC in a number of ways.

a) Adopt an operator convention to require some special character(s) in the field along with the desired data. (Note that the field must be made long enough to accommodate the extra character(s).) For example:

```
IF MATCH ?*!! THEN          \ Skip the edits.

ELSE <edit statements>      \ No trailing "!"; apply normal edits
```

Note that the special characters must satisfy the default data type edit. The above statement, for instance, would not work with a numeric field. Some possibilities of special characters for numeric fields are leading zeros (MATCH 0?*), a leading plus sign (MATCH !+?*), or a comma (MATCH 000,?*). If signed numbers are expected, prefix your pattern with !+,–.

b) Include an auxiliary EDIT OVERRIDE field that must be marked in order to bypass edits. For example, assume a field "EO":

```
IF EO MINLEN 1 THEN

ELSE <edit statements>      \EO not marked; apply edits
```

8. The contributed utility program RESTORE will not correctly copy records larger than 2000 bytes. If you wish to copy a KSAM forms file or reformat file from a store tape to a disc file, use the MPE :RESTORE command and the FCOPY subsystem.

9. It is possible to create fast form files with considerably more than the normal limit of 255 forms per file. Use the contributed utility program CONCAT to concatenate two non-KSAM fast form files.

10. When using a remote terminal via the DS facility, some forms over 255 characters long are not correctly displayed. This occurs when the LINEBUF parameter is not used in the DSLINE command. When forms larger than 255 characters are being used, include LINEBUF=n in the DSLINE command, where "n" is the number of words required by the screen image of the largest form in the forms file.

STATE CODES

The special system constant $STATE consists of a table of all the state codes for the 50 states of the United States plus five territories. When an entered state code is matched against the table, the entered code can be in any combination of upper or lower case letters. For instance, the code for California is CA; any of the following codes can be successfully matched against the California code: "CA", "ca", "Ca", or even "cA".

The state codes listed below are shown only in upper case for convenience

Alabama	AL	Montana	MT
Alaska	AK	Nebraska	NE
Arizona	AZ	Nevada	NV
Arkansas	AR	New Hampshire	NH
California	CA	New Jersey	NJ
Canal Zone	CZ	New Mexico	NM
Colorado	CO	New York	NY
Connecticut	CT	North Carolina	NC
Delaware	DE	North Dakota	ND
District of Columbia	DC	Ohio	OH
Florida	FL	Oklahoma	OK
Georgia	GA	Oregon	OR
Guam	GU	Pennsylvania	PA
Hawaii	HI	Puerto Rico	PR
Idaho	ID	Rhode Island	RI
Illinois	IL	South Carolina	SC
Indiana	IN	South Dakota	SD
Iowa	IA	Tennessee	TN
Kansas	KS	Texas	TX
Kentucky	KY	Utah	UT
Louisiana	LA	Vermont	VT
Maine	ME	Virginia	VA
Maryland	MD	Virgin Islands	VI
Massachusetts	MA	Washington	WA
Michigan	MI	West Virginia	WV
Minnesota	MN	Wisconsin	WI
Mississippi	MS	Wyoming	WY
Missouri	MO		

TERMINAL INFORMATION

STRAPPING THE TERMINAL

If you are using a 2640B terminal, you can use the function keys without the CNTL key by strapping your terminal for this purpose. You do this by opening the E switch on the KEYBD I/F PCA panel inside the terminal. (Note that the D, F, and G switches must also be open for BLOCK MODE/PAGE operation.) The A, C, and H straps must be closed.

The HP 2645, 2641, 2647, and 2648 terminals are automatically configured by HP V/3000. The following table summarizes the strapping for the 2640B and for the 2644A terminals:

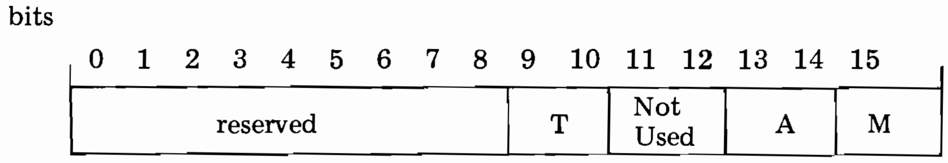
Switch	2640B	2644A
A	closed	closed
B	—	—
C	closed	closed
D	open	open
E	open	—
F	open	—
G	open	open
H	closed	closed

TERMINAL COMMUNICATIONS AREA

Words 49 through 58 (relative to 1) of the communications area contain information about the terminal. This information includes the terminal file number, terminal type, terminal allocation and error logging, and terminal options. Most users will never use this area. However, in case you want to know the MPE file number of your terminal, how to suppress messages, read data automatically, or specify a timeout value for reading keys, the terminal communications area words that perform these functions are outlined below.

Word 49 — TERM'FILEN — contains the MPE file number of your terminal.

Word 56 — TERM'OPTIONS — allows you to suppress mode messages in VOPENTERM and VCLOSE-TERM, enable an AUTOREAD function in VREADFIELDS, suppress the TERMINAL RESET key in VOPENTERM, and enable ENTER and function key timeouts in VREADFIELDS. The bit layout of word 56 is:



T: 01 = Enable ENTER key or function key timeout in VREADFIELDS. Wait the number of seconds specified in word 58 (see below), for an ENTER key or function key to be read by VREADFIELDS. If a key is not read within the specified time, VREADFIELDS sets the status word (word 1 of the communications area to the value 160.

00 = Disable ENTER key timeout in VREADFIELDS (default). No time limit for reading the ENTER or function keys.

A: 01 = Enable AUTOREAD in VREADFIELDS. The AUTOREAD feature causes VREADFIELDS to send an "ESC d" to the terminal instead of waiting for the operator to press the ENTER key. This automatic ENTER is useful for measuring performance.

00 = Disable AUTOREAD in VREADFIELDS (default). The ENTER key must be pressed to read data with VREADFIELDS.

M: 1 = Suppress mode messages in VOPENTERM and VCLOSETERM. Do not issue block mode warning messages when the terminal is opened or closed.

0 = Display mode message in VOPENTERM (default). The mode message for VOPENTERM is "BLOCK MODE/PAGE IS SET"; for VCLOSETERM, the mode message is "REMEMBER TO UNLATCH THE BLOCK MODE KEY".

Set the entire word 56 to zero to specify the default values. (The first 8 bits should always be zero.)

Word 58 — TERM'TIMEOUT — If the "T" bits in word 56 are set to 01 to enable an ENTER or function key timeout, the value in word 58 indicates the number of seconds to wait for the key to be pressed. The maximum timeout value is 5 minutes (300 seconds).

Word 59 — IDENTIFIER — type of terminal.

1	2640B
2	2644A
3	2645
4	2641A
5	2648A
6	2647A
8	2626A
9	2624A
10	2642A
11	2622A, 2623A
15	3075A
16	3076A

TERMINAL BUFFER CONFIGURATION

When using FORMSPEC, REFSPEC, ENTRY, or the intrinsics VREADFIELDS or VSHOWFORM, there must be sufficient terminal buffers available for all concurrently executing terminal I/O operations. It is recommended that the number of terminal buffers be at least 150. Creating, modifying, or displaying a form of 4000 characters requires 134 terminal buffers. Terminal buffers may be set to a maximum of 255, shared by all processes. (See configuration dialogue in System Manager manual part number 30000-90014.)

EXAMPLE SET OF TERMINAL EDITS

2624 Terminal Edits:

ALPHABETIC	Upper and lower case characters, blanks, period, dash.
ALPHA_NUMERIC	Alpha characters or digits, spaces and certain special characters.
CONSTANT	No characters may be entered from the keyboard.
DEC_DIGITS n	Set the number of digits to the right of the decimal point in numbers and implieds to n digits.
DEC_TYPE_EUR	European number format.
DEC_TYPE_US	US number format.
IMP_DEC	Implied decimal. (The number of digits after the decimal point is governed by the last DEC_DIGITS command in the form.)
IMP_DEC_FILL	Implied decimal, right justified and filled with zeros.
INTEGER	Digits and blanks.
INTEGER_FILL	Integer edit right justified and filled with zeros.
JUSTIFY	For numeric types, justify right, for character types, justify left.
MUST_FILL	If anything is entered in the field it must be filled.
REQUIRED	This field must have an entry at the time the "enter" key is pressed.
SIGN_DEC	Signed decimal. (Number of digits to the right of the decimal place is governed by the last DEC_DIGITS command.)
SIGN_DEC_FILL	Signed decimal, right justified, filled to the left with zeros.
TRANSMIT_ONLY	This field will not have a tab stop. (This edit may be used in combination with any other.)
UNRESTRICTED	Any character.
UPSHIFT	Alphabetic characters are upshifted as they are entered.

Note that this description is of the 2624 implementation of the edits. The actual implementation is terminal dependent and may be done in a different manner on a different terminal.

INDEX

INDEX (continued)

READER COMMENT SHEET

HP 3000 Computer System
V/3000 Reference Manual

32209-90001 **Feb 1981**

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate?	Yes ☐	No ☐	(If no, explain under Comments, below.)
Are the concepts and wording easy to understand?	Yes ☐	No ☐	(If no, explain under Comments, below.)
Is the format of this manual convenient in size, arrangement, and readability?	Yes ☐	No ☐	(If no, explain or suggest improvements under Comments, below.)

Comments:

FROM: Date ____________________

Name ____________________

Company ____________________

Address ____________________

____

____

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 492 CUPERTINO, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

Customer Information Manager
Hewlett-Packard Company
Information Systems Division
19420 Homestead Road
Cupertino, California 95014

FOLD FOLD