

Computer Systems Division

HP Data Entry and Forms Management System (VPLUS/V)

Reference Manual



19447 Pruneridge Ave., Cupertino, CA 95014

Part No. 32209-90001
U1187

Printed in U.S.A. 7/86
Updated 11/87

NOTICE

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

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

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company.

Copyright © 1978,1980,1981,1982,1986,1987 by HEWLETT-PACKARD COMPANY

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 dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the date indicates the version level of the software product at the time the manual 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 #1	Apr 1980	32209A.01
Update #1 Incorporated	Aug 1980	32209A.01
Third Edition	Feb 1981	32209B
Fourth Edition	Nov 1982	32209B.02
Update #1	May 1983	32209B.03
Update #2	Sep 1984	32209B.04
Update #2 Incorporated	Apr 1985	32209B.04
Fifth Edition	Jul 1986	32209B.04.17
Update #1	Nov 1987	32209B.05.00

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.



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.

Effective Pages	Date	Effective Pages	Date
v to vii	Nov 1987	3-53 to 3-54	Nov 1987
xi to xxiv.	Nov 1987	4-47 to 4-48.	Nov 1987
1-1 to 1-4	Nov 1987	6-1 to 6-4	Nov 1987
1-13 to 1-14a	Nov 1987	6-9 to 6-12b.	Nov 1987
3-1 to 3-4	Nov 1987	6-17 to 6-20.	Nov 1987
3-7 to 3-8	Nov 1987	6-43 to 6-44.	Nov 1987
3-15 to 3-18.	Nov 1987	6-49 to 6-50.	Nov 1987
3-33 to 3-34.	Nov 1987	6-63 to 6-66.	Nov 1987
3-37 to 3-40.	Nov 1987	6-85 to 6-92.	Nov 1987
3-45 to 3-50.	Nov 1987	7-1 to 7-4	Nov 1987

LIST OF EFFECTIVE PAGES (continued)

Effective Pages	Date	Effective Pages	Date
7-7 to 7-12.	Nov 1987	H-1 to H-2.	Nov 1987
7-15 to 7-20.	Nov 1987	H-5 to H-6.	Nov 1987
B-15 to B-16b.	Nov 1987	Appendix J	Nov 1987
B-23 to B-24j.	Nov 1987	Index	Nov 1987
B-45 to B-46b.	Nov 1987		
B-55 to B-58.	Nov 1987		
B-69 to B-70.	Nov 1987		
E-5 to E-6	Nov 1987		
G-1 to G-2b	Nov 1987		
G-15 to G-20	Nov 1987		

PREFACE

This publication is the reference manual for VPLUS/V, a comprehensive software system that operates on the HP 3000 computer system to provide terminal data entry and control for most HP terminals and data capture devices. VPLUS/V 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.

VPLUS/V 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 any of the supported programming languages (listed in Appendix A.)

VPLUS/V users may be terminal users (such as data entry clerks), forms designers with or without programming experience, or programmers in any of the supported languages. 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 intrinsics used for programmatic interface with forms and terminals.

In order to use the VPLUS/V 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. The manuals in the following list contain all the information you might need as a supplement to this manual:

Using VPLUS/V (32209-90004)
HP 3000 Guide for the New User (32033-90009)
MPE Commands Reference Manual (30000-90009)
MPE Intrinsics Reference Manual (30000-90010)
MPE Software Pocket Guide (30000-90049)
Using Files (30000-90102)
Native Language Support Reference Manual (32414-90001)

Also, refer to the reference manual for the programming language to be used with VPLUS/V (listed in Appendix A), and the reference manual for your terminal.



CONVENTIONS USED IN THIS MANUAL

NOTATION

DESCRIPTION

nonitalics

Words in syntax statements which are not in italics must be entered exactly as shown. Punctuation characters other than brackets, braces and ellipses must also be entered exactly as shown. For example:

EXIT;

italics

Words in syntax statements which are in italics denote a parameter which must be replaced by a user-supplied variable. For example:

CLOSE *filename*

[]

An element inside brackets in a syntax statement is optional. Several elements stacked inside brackets means the user may select any one or none of these elements. For example:

$\left[\begin{array}{l} A \\ B \end{array} \right]$ User *may* select A or B or neither.

{ }

When several elements are stacked within braces in a syntax statement, the user must select one of those elements. For example:

$\left\{ \begin{array}{l} A \\ B \\ C \end{array} \right\}$ User *must* select A or B or C.

...

A horizontal ellipsis in a syntax statement indicates that a previous element may be repeated. For example:

[,*itemname*]...;

In addition, vertical and horizontal ellipses may be used in examples to indicate that portions of the example have been omitted.

■

A shaded delimiter preceding a parameter in a syntax statement indicates that the delimiter *must* be supplied whenever (a) that parameter is included or (b) that parameter is omitted and any *other* parameter which follows is included. For example:

itema[■*itemb*][■*itemc*]

means that the following are allowed:

itema
itema,itemb
itema,itemb,itemc
itema,,itemc

CONVENTIONS (continued)

Δ When necessary for clarity, the symbol Δ may be used in a syntax statement to indicate a required blank or an exact number of blanks. For example:

```
SET[(modifier)] $\Delta$ (variable);
```

underlining When necessary for clarity in an example, user input may be underlined. For example:

```
NEW NAME? ALPHA
```

Brackets, braces or ellipses appearing in syntax or format statements which must be entered as shown will be underlined. For example:

```
LET var[[subscript]] = value
```

Output and input/output parameters are underlined. A notation in the description of each parameter distinguishes input/output from output parameters. For example:

```
CREATE (parm1,parm2,flags,error)
```

shading

Shading represents inverse video on the terminal's screen. In addition, it is used to emphasize key portions of an example.

The symbol may be used to indicate a key on the terminal's keyboard. For example, RETURN indicates the carriage return key.

CONTROL *char*

Control characters are indicated by CONTROL followed by the character. For example, CONTROLY means the user presses the control key and the character Y simultaneously.

Section 1

INTRODUCING VPLUS/V

Source Data Entry	1-1
Transaction Processing	1-1
Features	1-2
Using The Manual	1-4
Forms Designer	1-4
Applications Programmer	1-4
Application User	1-5
Designing Forms	1-6
Levels of Form Design	1-6
Simple Collection	1-6
Simple Editing	1-8
Field Type	1-8
Data Type	1-9
Field Name	1-9
Field Enhancement	1-9
Initial Value	1-10
Full Field Edits	1-10
Advanced Form Processing	1-11
Advanced Processing Statements	1-12
Save Fields & Other Global Specifications	1-13
Application-ready Buffer	1-13
Benefits of the ARB	1-13a
Program Interface	1-14
Data Entry Program	1-16
Data Collection	1-16
Browse/Modify	1-17
Reformatting Data	1-18

Section 2

ENTERING DATA WITH ENTRY

Protected and Unprotected Fields	2-1
Detecting Errors	2-1
Modifying Data	2-1
Reformatting Data	2-1
Using the Terminal	2-2
Function Keys	2-4
Display Enhancements	2-4
Printing Forms and Data	2-4
Running Entry	2-5
Specifying ENTRY Files	2-5
Forms File	2-5
Batch File	2-5
Removing Deleted Records From a Batch	2-7
Expanding the Batch File	2-7
Block Mode	2-8

CONTENTS (continued)

Local Form Storage	2-8
Forms Sequence	2-9
Repeating Forms	2-9
Appended forms	2-9
Frozen Forms	2-9
Entering Data	2-11
Optional and Required Fields	2-12
Program Generated Data	2-13
Correcting Errors	2-14
Before ENTER	2-14
After ENTER	2-14
System and Logic Errors	2-14
Interrupting Data Entry	2-15
Terminating a Session	2-15
Unexpected Program Interruption	2-15
Modifying Data	2-16
Entering Browse/Modify Mode	2-16
Resuming Data Collection	2-16
Viewing the Data	2-17
Modifying the Data	2-17
Deleting Data	2-17

Section 3

INTRODUCTION TO FORMS DESIGN

Getting Started with FORMSPEC	3-2
Specify a Forms File	3-2
Choose an Option	3-3
FORMSPEC Function Keys	3-4
Menu Sequence	3-5
Creating an Application-ready Buffer (ARB)	3-7
Terminating FORMSPEC	3-7
Unexpected Program Interruption	3-7
Ease of Forms Design	3-8
Form Layout	3-8
Field Delimiters	3-9
Field Tag	3-11
Defining the Fields	3-11
Field Number	3-12
Field Length	3-12
Field Name	3-13
Display Enhancement	3-14
Field Type	3-14
Data Type	3-15
Initial Value	3-17b
Understanding Form Sequencing	3-18
The Repeat Option	3-18
The Next Form Option	3-19
Sample of Forms Design	3-22

CONTENTS (continued)

Forms File	3-25
Forms Modification	3-25
Forms File Size	3-25
Expand Forms File	3-27
Compiling the Forms File	3-27
Listing Forms	3-28
Form Families	3-30
Creating Child Forms	3-30
Form Family Example	3-31
Using Terminal Features	3-33
The Security Enhancement	3-33
Color Enhancement	3-33
Terminal Color Pairs	3-33
Color Field Enhancements	3-33
Color Error Enhancements	3-34
Color Window Enhancements	3-34
Color Locks	3-34
Forms File Menu	3-35
Main Menu	3-37
Form Menu	3-40
Form Function Key Labels Menu	3-43
Form Layout	3-44
Field Menu	3-45
Globals Menu	3-48
Global Function Key Labels Menu	3-51
Save Field Menu	3-52
Using FORMSPEC to Create an ARB	3-53a
Setting Up the Data Type Conversion Record	3-53c
Generating the ARB	3-53d
Types of Field	3-53e
Using and Defining ARBs	3-53j
ARB Trace Facility	3-53k
Terminal/Language Selection Menu	3-54
Data Capture Device Specifications Menu	3-56

Section 4

ADVANCED FORMS DESIGN

Levels of Advanced Design	4-2
Entering Processing Specifications	4-3
Special Cases	4-4
Correcting Existing Specifications	4-4
Statement Syntax	4-5
Comments	4-5
Continuing Lines	4-5
Custom Error Messages	4-5
Checking Data Entered	4-6
Operands	4-9

CONTENTS (continued)

Field	4-10
Constant	4-10
Arithmetic Expression	4-11
Index Retrieve Operand	4-12
Processing Specifications for Configuration	4-13
DEVICE	4-14
LIGHT	4-15
LOCALEDITS	4-16
Processing Specifications for Field Edits	4-20
Edit Statements	4-20
LENGTH CHECK	4-21
SINGLE VALUE COMPARISONS	4-23
Character Comparisons	4-23
Numeric Comparisons	4-24
Date Comparisons	4-24
Native Language Support	4-24
TABLE CHECKS	4-25
Native Language Support	4-26
RANGE CHECKS	4-27
Native Language Support	4-28
CHECK DIGIT	4-29
PATTERN MATCH	4-30
Special Pattern Characters	4-30
Transparency	4-31
Choice	4-31
Range	4-32
Grouping and Optional	4-32
Repetition	4-33
Operator Hierarchy	4-34
Processing Specifications for Advanced Processing	4-36
SET	4-37
Assigning a Value to the Current Field	4-38
Moving Data Between Fields	4-40
Automatic Formatting	4-41
Character Type	4-41
Numeric Type	4-41
Date Type	4-42
Default Formatting	4-44
STRIP	4-46
JUSTIFY	4-47
FILL	4-48
UPSHIFT	4-49
CHANGE	4-50
IF	4-52
FAIL	4-55
STOP	4-56
PHASES	4-57
Configuration Phase	4-58
Initialization Phase	4-58

CONTENTS (continued)

Field Edit Phase	4-60
Finish Phase	4-60

Section 5 REFORMATTING SPECIFICATIONS

Files	5-3
Reformat File	5-3
Relation of Forms to Output Records	5-5
Input Forms Sequence	5-5
Output Record Definition	5-6
Field Specifications	5-6
Combining Data from Several Forms	5-7
Separating Data into Several Records	5-9
Reformatting Repeating Forms	5-10
Separating Data from One Batch File into Several Output Files	5-12
Using REFSPEC	5-13
Terminating REFSPEC	5-13
Unexpected Program Interruption	5-13
REFSPEC Function Keys	5-14
REFSPEC Menus	5-15
Reformat File Menu	5-17
Forms File Menu	5-18
Main Menu	5-19
Adding a Reformat	5-19
Compiling the Reformat File	5-19
Adding Global Specifications	5-20
Changing the Forms File Name	5-20
Modifying Reformat Specifications	5-20
Changing an Input Forms Sequence	5-20
Changing an Output Record Definition	5-21
Deleting a Reformat	5-21
Listing a Reformat	5-21
Resuming Design from Main Menu	5-21
Globals Menu	5-24
Input Forms Menu	5-27
Output Record Menu	5-29
Output Field Menu	5-34
Data Type	5-35
Field Formatting	5-35
Strip	5-38
Justify	5-39
Sign	5-40
Fill	5-42
Check Digit	5-43

CONTENTS (continued)

Running Reformat	5-44
Concatenating Batch Files	5-45
Using a User-Defined Command	5-45
Streaming REFORMAT	5-46

Section 6 USING VPLUS/V INTRINSICS

Multipurpose	6-1
Multilanguage	6-1
Error Detection	6-1
How Intrinsic are Used	6-4
Form Definition Area	6-4
Data Buffer Area	6-4
Window Area	6-6
Key Label Area	6-6
Calling VPLUS/V Intrinsic	6-7
Parameter Types	6-7
Communication Area	6-9
Error Handling	6-22
Intrinsic Call or File Errors	6-22
Editing Errors	6-22
Error Messages	6-22
Determining Fields in Error	6-23
Using Terminal Features	6-24
The Touch Feature	6-24
Local Form Storage	6-24
The HP 2626A vs. the HP 2624B/HP 2394A	6-24
Form Storage Directory	6-25
Loading Forms	6-25
Form Families and Local Storage	6-26
Appending Forms and Local Storage	6-26
Purging Forms From Local Storage	6-27
Intrinsic Descriptions	6-28
Dependency Between Intrinsic	6-31
VCHANGEFIELD	6-32
VCLOSEBATCH	6-34
VCLOSEFORMF	6-35
VCLOSETERM	6-36
VERRMSG	6-37
VFIELDDEITS	6-39
VFINISHFORM	6-41
VGETBUFFER	6-43
VGETFIELD	6-45

CONTENTS (continued)

VGETFIELDINFO	6-48
Passing INFOBUF Without Entering Keys	6-49
Table Wraparound	6-50
VGETFILEINFO	6-51
VGETFORMINFO	6-54
Passing INFOBUF Without Entering Keys	6-55
Table Wraparound	6-56
Error Determination	6-56
VGETKEYLABELS	6-58
VGETLANG	6-60
VGETNEXTFORM	6-62
VGETtype	6-64
VINITFORM	6-68
VLOADFORMS	6-70
VOPENBATCH	6-72
Existing File	6-73
New File	6-73
Record Format	6-73
Labels	6-74
Creating Your Own Batch File	6-75
VOPENFORMF	6-76
VOPENTERM	6-78
VPLACECURSOR	6-80
VPOSTBATCH	6-82
VPRINTFORM	6-84
VPRINTSCREEN	6-85a
Modes of Operation	6-85b
Normal Mode	6-85b
Documentation Mode	6-85c
Limitations	6-85d
VPUTBUFFER	6-86
VPUTFIELD	6-88
VPUTtype	6-90
VPUTWINDOW	6-92
VREADBATCH	6-94
VREADFIELDS	6-96
VSETERROR	6-99
VSETKEYLABEL	6-102
VSETKEYLABELS	6-104
VSETLANG	6-106
VSHOWFORM	6-108
Performance Optimization	6-109
Display and Local Form Storage	6-109
Controlling the Keyboard	6-110
Controlling Preload of Forms	6-110



CONTENTS (continued)

Controlling the Touch Feature	6-111
Controlling the Bell	6-111
VUNLOADFORM.	6-112
VWRITEBATCH.	6-114
Data Collection Mode	6-115
Modify Mode	6-115

Section 7

USING FORMSPEC IN BATCH MODE

Compiling Forms File in Batch Mode	7-1
Updating Forms in Batch Mode.	7-1
Listing Forms Files in Batch Mode.	7-1
Relating Forms in Batch Mode	7-1
Invoking FORMSPEC in Batch Mode.	7-2
Batch Mode Commands	7-3
COMPILE	7-5
COPY.	7-6
DELETE.	7-7
EXIT	7-8
FIELD	7-9
FILE	7-10
FKLABELS	7-11
FORMS	7-12
LIST	7-15
RELATE.	7-16
RENUMBER	7-17
SELECTLANG.	7-18
SELECTTERM.	7-19
ARB Batch Mode Commands	7-19a
ADDARBFIELD.	7-19b
ARBTOSCREEN.	7-19c
DELARB	7-19d
DELARBFIELD	7-19e
GENARB	7-19f
MODARBFIELD.	7-19g
MOVEARBFIELD.	7-19h
RENAMEARBFIELD.	7-19i
SCREENTOARB.	7-19j
Samples of FORMSPEC in Batch Mode.	7-20
Sample Command File	7-20
Sample Job Stream.	7-20

CONTENTS (continued)

Section 8 NATIVE LANGUAGE SUPPORT

Language Attribute	8-1
Unlocalized (NATIVE-3000)	8-1
Language-Dependent	8-1
International	8-2
NLS CAPABILITIES	8-2
Setting the Native Language ID Numbers	8-2
Field Edits	8-3
Date Handling	8-3
Numeric Data	8-3
Native Language Characters	8-3
Entry and Language ID Number	8-4

Appendix A SAMPLE PROGRAMS

SPL	A-2
COBOL	A-26
FORTRAN 77	A-38
BASIC	A-57
TRANSACT	A-66
PASCAL	A-75

Appendix B VPLUS/V ERROR MESSAGES

Error Messages	B-2
--------------------------	-----

Appendix C USASCII CHARACTER SET IN COLLATING SEQUENCE

Appendix D CHECK DIGIT CALCULATION

Modulus 10	D-2
Modulus 11	D-3
Alphabetic Check Digits	D-4

Appendix E APPLICATION HINTS

Designing your Forms File	E-2
Rules for the DL Area	E-6
Coding the Touch Feature	E-7

CONTENTS (continued)

Appendix F STATE CODES

Appendix G TERMINAL INFORMATION

Supported Terminals and Features	G-2
Termtypes	G-2
Modified Data Tag	G-2
Extended Local Edits	G-2
Relabeling Function Keys	G-2
Security Display Enhancement	G-2
Local Form Storage Capabilities	G-3
X.25 Capability	G-4
Color Enhancement	G-4
Data Capture Device Special Features	G-4
The HP 264X Terminals	G-5
HP 2640B Terminal	G-5
HP 2644A Terminal	G-6
The HP 262X Terminals	G-7
HP 2624A Terminal	G-7
HP 2624B Terminal	G-7
HP 2625A, HP 2628A Terminals	G-7
HP 2626A Terminal	G-8
HP 2627A Terminal	G-8
The HP 239X and HP 150 Terminals	G-9
HP 2392A Terminal	G-9
HP 2393A Terminal	G-9
HP 2394A Terminal	G-9
HP 2397A Terminal	G-9
HP 150 Terminal	G-9
The Data Capture Devices	G-10
Terminal Communication Area	G-11
Terminal Buffer Configuration	G-13
Recovering from Unexpected Program Interruption	G-14
User Environment Control File	G-15
Advanced Terminal I/O Procedures	G-16
VTURNON/VTURNOFF	G-16
VTURNOFF	G-17
VTURNON	G-19

CONTENTS (continued)

Appendix H VERSION CONTROL

KSAM File Management	H-2
Purge Forms File	H-2
Rename Forms File	H-2
Convert KSAM File	H-3
Adjust Menu	H-4
HP32209	H-5
The HP32209B Utility	H-6

Appendix I BIT MAP CONVERSION

Appendix J HP PRECISION ARCHITECTURE

HPPA Intrinsics	J-2
Introduction	J-2
VGETIEEEREAL	J-3
VGETIEEELONG	J-5
VPUTIEEEREAL	J-7
VPUTIEEELONG	J-9
The HP32209S Utility	J-11

FIGURES AND TABLES

FIGURES

Figure 1-1.	Overview of VPLUS/V	1-3
Figure 1-2.	FORMSPEC Form Menu	1-7
Figure 1-3.	FORMSPEC Screen Design	1-8
Figure 1-4.	Sample Field Menu	1-9
Figure 1-5.	Advanced Processing Specifications	1-11
Figure 1-6.	Transfers Controlled by VPLUS/V Intrinsic	1-15
Figure 2-1.	HP Terminal Keyboard	2-2
Figure 2-2.	Example of Forms Sequence	2-10
Figure 2-3.	Bracketed Fields	2-11
Figure 2-4.	Fields Without Brackets	2-12
Figure 2-5.	Example of Data Entry	2-13
Figure 2-6.	Example of Data Modification	2-18
Figure 3-1.	Forms File Menu	3-2
Figure 3-2.	Main Menu	3-3
Figure 3-3.	Relation Between Menus & Function Keys	3-6
Figure 3-4.	Sequence of Menus for Form Design	3-8
Figure 3-5.	Example of Form Layout	3-9
Figure 3-6.	Example of Form Layouts	3-10
Figure 3-7.	Example of a Field Menu	3-11
Figure 3-8.	The Field Attributes	3-12
Figure 3-9.	Example of a Field Extending over Several Lines	3-13
Figure 3-10.	Form Menu	3-18
Figure 3-11.	Sample Forms File Layout	3-23
Figure 3-12.	Forms File Prototype	3-26
Figure 3-13.	Forms File Listing	3-28
Figure 3-14.	Form Family Relationship	3-30
Figure 3-15.	Parent Form	3-32
Figure 3-16.	Child Form	3-32
Figure 3-17.	Forms File Menu	3-35
Figure 3-18.	Main Menu	3-37
Figure 3-19.	Form Menu	3-40
Figure 3-20.	Form Function Key Labels Menu	3-43
Figure 3-21.	Form Layout Menu	3-44
Figure 3-22.	Field Menu	3-45
Figure 3-23.	Globals Menu	3-48
Figure 3-24.	Global Function Key Labels Menu	3-51
Figure 3-25.	Save Field Menu	3-52
Figure 3-25a.	Menu Sequence for ARB Feature	3-53b
Figure 3-25b.	Globals Menu	3-53c
Figure 3-25c.	Data Type Conversions Menu	3-53d
Figure 3-25d.	ARB Menu	3-53e
Figure 3-25e.	Restructure ARB Menu	3-53h
Figure 3-25f.	ARB Field Menu	3-53j
Figure 3-26.	Terminal/Language Selection Menu	3-54
Figure 3-27.	Data Capture Device Specification Menu	3-56
Figure 3-28.	Form Layout in Formspec	3-59

FIGURES AND TABLES (continued)

Figure 3-29.	Displayed on Standard Character Set Mini-CRT Screen	3-59
Figure 3-30.	Displayed on 32-Character One Line Alpha Display Screen	3-59
Figure 3-31.	Displayed on Large Character Mini-CRT Screen	3-60
Figure 4-1.	Field Menu with Processing Specifications	4-3
Figure 4-2.	Field Menu With Custom Error Messages	4-6
Figure 4-3.	Field Menu with a Set Statement	4-44
Figure 4-4.	Flow of Phase Execution for Entry	4-59
Figure 5-1.	Relation Among Files Used for Reformatting	5-2
Figure 5-2.	Prototype of Reformat File	5-4
Figure 5-3.	Combining Data From 3 Forms into 1 Output Record	5-7
Figure 5-4.	Reformat Data From 3 Forms into 2 Output Records	5-8
Figure 5-5.	Separating Data From 1 form into Several Records	5-9
Figure 5-6.	Reformatting Data From Repeating Forms	5-10
Figure 5-7.	Reformatting Data Based on Form Sequence	5-11
Figure 5-8.	Generating 2 Output Files From 1 Batch File	5-12
Figure 5-9.	Relation of REFSPEC Menus to Function Keys	5-16
Figure 5-10.	Reformat File Menu	5-17
Figure 5-11.	Forms File Menu	5-18
Figure 5-12.	Main Menu	5-19
Figure 5-13.	Reformat Listing	5-22
Figure 5-14.	Globals Menu	5-24
Figure 5-15.	Input Forms Menu	5-27
Figure 5-16.	Output Record Menu	5-29
Figure 5-17.	Sample Output Record Menu	5-30
Figure 5-18.	Output Field Menu	5-34
Figure 6-1.	Operation of VPLUS/V Intrinsic	6-5
Figure 6-2.	Intrinsic Flow	6-29
Figure 6-3.	Intrinsic Dependencies	6-31
Figure H-1.	The Adjust Menu	H-4

TABLES

Table 2-1.	Function Keys for Entry	2-3
Table 3-1.	FORMSPEC Key Labels	3-4
Table 3-2.	FORMSPEC Reserved Word List	3-13
Table 3-3.	Display Enhancement	3-14
Table 3-4.	How Each Numeric Data Type Interprets entered Values	3-16
Table 3-4a.	Recommended Data Type Conversions	3-17a
Table 3-4b.	Valid Screen Data Types	3-17b
Table 3-4c.	Valid Application Data Types	3-17c
Table 3-5.	Valid Dates	3-17
Table 3-6.	Invalid Dates	3-17
Table 3-7.	Default Color Pairs	3-34
Table 3-8.	Forms File Menu	3-36
Table 3-9.	Form/ARB Relationships	3-36

FIGURES AND TABLES (continued)

Table 4-1.	Summary of Processing Statements	4-7
Table 4-2.	Terminal Local Edits	4-17
Table 4-3.	Local Edits Corresponding to Field Type Selections in the Terminal Edit Checks Menu	4-18
Table 4-4.	Local Edits Corresponding to Attribute Selection in the Terminal Edit Checks Menu	4-18
Table 4-5.	Local Edits Corresponding to Format Mode Selection in the Terminal Configuration Menu	4-19
Table 4-6.	Pattern Match Operators	4-34
Table 4-7.	Automatic Formatting for Character Data	4-41
Table 4-8.	Automatic Formatting for Numeric Data	4-42
Table 4-9.	Automatic Formatting for Dates	4-43
Table 4-10.	Conversion During Data Movement	4-43
Table 4-11.	Variations on the IF Statement	4-54
Table 5-1.	Key Definitions	5-14
Table 5-2.	Standard Formatting by Data Type	5-36
Table 5-3.	Examples Using STRIP	5-39
Table 5-4.	Examples of Three Types of Justification	5-40
Table 5-5.	Correspondence Between Signed Digits and Characters	5-41
Table 5-6.	Example Using the SIGN Option	5-41
Table 5-7.	Example Using the FILL Option	5-42
Table 5-8.	Example Using the FILL LEADING and FILL ALL Option	5-42
Table 6-1.	Summary of VPLUS/V Intrinsic	6-2
Table 6-2.	Examples of Intrinsic Call Formats for each Language	6-7
Table 6-3.	Data Types Used for Various Languages	6-8
Table 6-4.	Outline of Communications Area Contents	6-9
Table 6-5.	Communication Area Contents for Data Capture Devices	6-11
Table 6-6.	Codes for the Window Line Enhancement	6-13
Table 6-7.	Intrinsics by Function Group	6-28
Table 6-8.	Specifications Buffer	6-33
Table 6-9.	Actions Used by Intrinsics	6-40
Table 6-10.	Field Information Buffer	6-49
Table 6-11.	File Information Buffer	6-52
Table 6-12.	Form Information Buffer	6-55
Table 6-13.	Numeric Type Conversion	6-65
Table 6-14.	Correlation of VGETtype with the Data Types for each Language	6-66
Table 7-1.	FORMSPEC in Batch Mode Commands	7-3
Table A-1.	Program Languages and References	A-1
Table G-1.	Terminals Supported by VPLUS/V	G-3

VPLUS/V is a comprehensive software system that implements and controls source data entry. It also provides an interface between a terminal and any transaction processing program. Refer to Figure 1-1 for an overview of the entire VPLUS/V system.

Source Data Entry

As a source data entry system, VPLUS/V provides easy forms design with data editing and validation built into the forms. It also provides a ready-to-use data entry program called ENTRY that you can use to enter data without any programming effort. ENTRY allows you to "browse" the entered data and modify it as it is entered.

Thus, source data entry through VPLUS/V can be done without programming by simply using ENTRY. If, however, you need additional or different capabilities, you can write your own application incorporating VPLUS/V intrinsics. You may want to refer to the sample applications provided in Appendix A for guidance.

Transaction Processing

As an interface to transaction processing applications, VPLUS/V provides a set of intrinsics that allows you to control forms and data on a terminal from an application program. These intrinsics are available to programs written in any of the supported programming languages listed in Appendix A.

Additionally, some languages (also listed in Appendix A) provide the programmer with a special interface with terminals and forms, as described in their respective reference manuals. With these languages, the programmer does not call the VPLUS/V intrinsics directly. Instead, the programmer specifies the statements appropriate to the special interface.

VPLUS/V also provides a batch 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 VPLUS/V intrinsics and the reformatting capability, either singly or in combination, provide a "front end" to existing transaction processing applications. Thus, VPLUS/V 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 VPLUS/V system:

- A forms design program (FORMSPEC) that allows quick and easy forms design at a terminal using "menus".
- Batch mode management of forms files (through FORMSPEC) that allows a forms file to be updated, compiled, and listed without tying up a terminal.
- Powerful runtime data transformation capability (Application-ready Buffer, or ARB) that allows data to be entered on the screen in an order and format that make sense to the user, while the application receives the data converted and reordered to match the format of its storage record.
- Advanced forms design (through FORMSPEC) that provides for editing, formatting, movement, and computation of data when the form is executed, using the user's native language for alphabetic information and the local customs for numeric and date information.
- 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 intrinsics that provides a powerful programming language interface to terminals, using FORMSPEC definitions, from applications written in any of the supported programming languages.

Figure 1-1 shows data entered at the terminal. The data is then read and manipulated by VPLUS/V intrinsics in an application such as ENTRY. This manipulation is done 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 then available for use by an application. If desired, it can be reformatted by REFORMAT and written to an "output file".

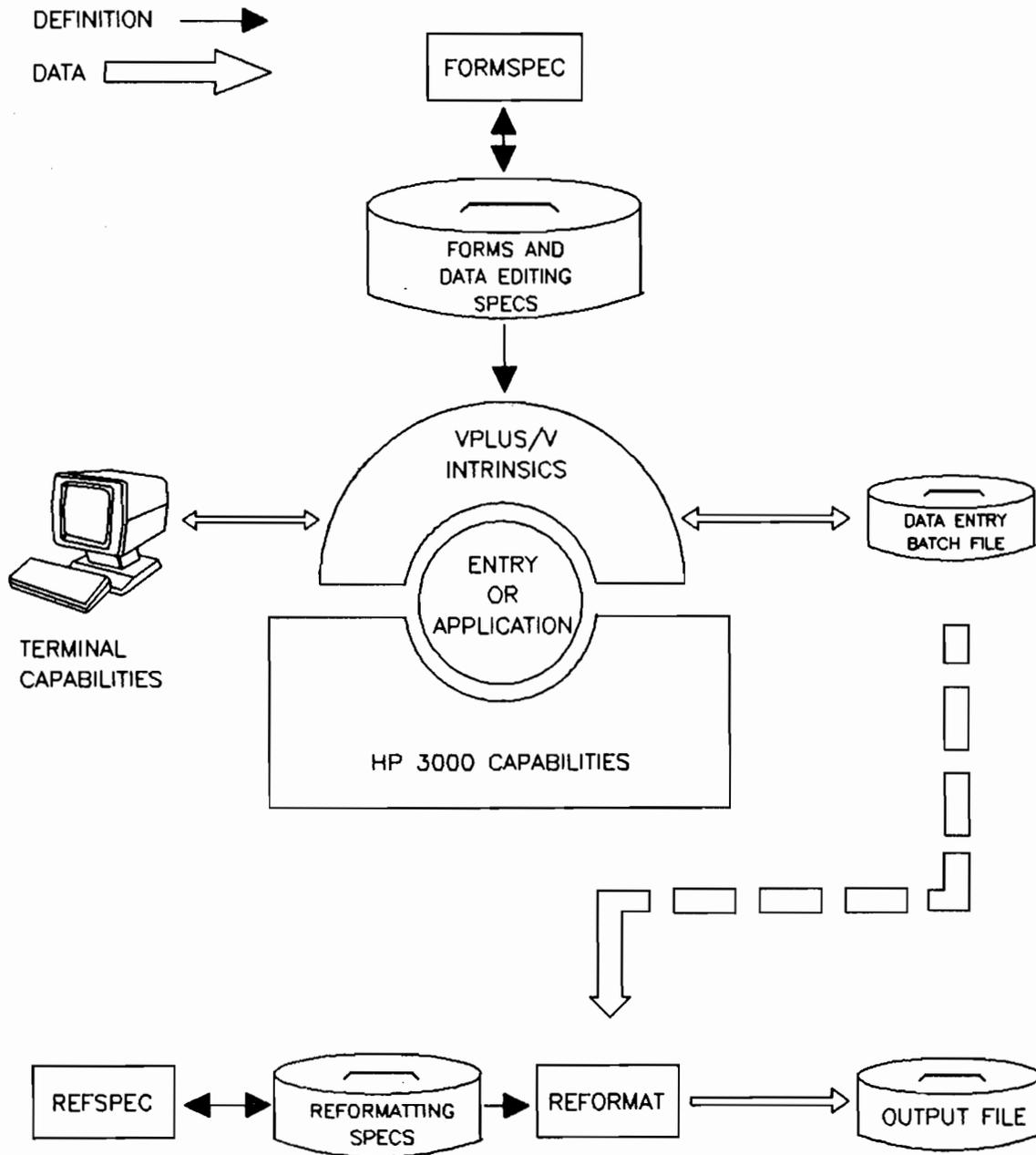


Figure 1-1. Overview of VPLUS/V

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 displayed, 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.
- **Application User** who enters data on the forms defined with FORMSPEC for processing by an application or by ENTRY.

Forms Designer

The forms designer who uses FORMSPEC to design forms should read Sections 3 and 4. These sections describe simple and advanced forms design with FORMSPEC. If designing forms in a native language other than the default (NATIVE-3000), the forms designer should refer to Section 8 for information on Native Language Support. The forms designer may also have the responsibility for reformatting the data entered through the forms. If so, the forms designer should also read Section 5, which describes how to specify reformatting with REFSPEC. Unless the forms designer is also the applications programmer, the forms designer can skip Sections 2, 6, and 7.

Applications Programmer

The first thing the applications programmer should determine is whether ENTRY, the data entry program provided with VPLUS/V, fulfills the user's application needs, either as is or with a few modifications. Using ENTRY from the user's point of view is described in Section 2.

If ENTRY is not suitable, the applications programmer should read Section 6 that describes the VPLUS/V intrinsics callable from any of the supported programming languages listed in Appendix A. If using one of the special interface programming languages, the applications programmer should refer to the reference manual for that language.

VPLUS/V Native Language Support enables an application programmer to create interactive applications which reflect native language numeric and date conventions in more than a dozen supported languages. For information on using VPLUS/V with Native Language Support, the applications programmer should refer to Section 8.

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. Interactive FORMSPEC is described in Sections 3 and 4; batch mode FORMSPEC is described in Section 7.

The applications programmer should also read Section 5 that describes the specifications for reformatting data entered through an application or ENTRY. Reformatting may provide a way to adapt entered data to an existing application.

Application User

The application user, or user, is the person who actually enters data into the forms designed with FORMSPEC and displayed by the application.

If the applications programmer has chosen to use ENTRY, then Section 2, which describes how to enter data to be processed by ENTRY, can be used as an instruction guide for users (or for their trainer).

If ENTRY has been modified, the applications programmer should modify Section 2 to reflect the changes.

If ENTRY is not used, Section 2 should not be used. Instead, the applications programmer should provide a similar guide that explains how to:

- Log on and off the HP 3000.
- Use the terminal keys, especially the function keys.
- Enter and modify data.
- Run REFORMAT (if applicable).

DESIGNING FORMS

FORMSPEC 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.

Forms are kept in a forms file, which must be compiled before the forms can be used. There may be one or many forms in a forms file. In fact, there may be multiple forms files per application or multiple applications per forms file, depending on the complexity of the task to be performed. 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 displayed 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:

1. Simple Collection Layout the form and allow any character set codes entered by the user. This level keeps the default field attributes and does not require any other FORMSPEC editing or formatting capabilities.
2. Simple Editing Layout the form and specify edits for the field attributes, such as on field type (optional, required, display only) or data type (character, numeric, or date). No processing specifications are required for the field attributes.
3. Full Field Edits Specify field edits using those processing specifications that apply to individual fields in a form. These include: minimum length, range checks, pattern checks, and so forth. A subset of the available processing specifications are used for these edits.
4. 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 processing specifications. This level uses the full range of the processing specifications.

Simple Collection

When you run FORMSPEC, the first menu is the Forms File Menu where you specify the name of the forms file. Then the Main Menu is displayed where you can select what you want to do: add a form or a save field, modify or delete a form, copy a form, select a terminal type or a native language, compile the forms file, or go to the global characteristics or to a particular form or field.

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 v.uu.ff 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]		
PREV FORM	NEXT FORM	REFRESH	PREV NEXT MAIN/RESUME EXIT

Figure 1-2. FORMSPEC Form Menu

Note that the sequence in which forms are displayed is very flexible. In Sections 3 and 4 you will learn how to 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 layout 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. Refer to Figure 1-3 for a sample form layout. The special terminal capabilities that provide field enhancements such as "half bright", "color", "inverse video", or "blinking" are entered through FORMSPEC, as described in Sections 3 and 4. You do not need to use complicated escape sequences to request them.

The position of a field is defined by field delimiters. You identify each field 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, as shown in Figure 1-3.

```

*****ABC MANUFACTURING*****

Order Number: [ordnum]                               Date: [ordate   ]

SHIP TO:
  Name [name           ]
  Address [address      ]
  City [city           ] State [st]
  Zip [zip             ] Telephone [phone          ]

QTY   PART NO.   DESCRIPTION                               UNIT   TOTAL
qty.. partnum..  description.....                               PRICE  PRICE
uprice. tprice.

PREV  NEXT      REFRESH      PREV  NEXT  MAIN/  EXIT
FORM  FORM

```

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 displayed by FORMSPEC for each field in a form. Refer to Figure 1-4 for an illustration of a typical Field Menu, which shows the various field attributes in the highlighted fields. The field name and length determined by the form design are displayed on this menu. The menu also displays the data type, field enhancements, and the 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 attribute 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 user 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".

FORMSPEC v.uu.ft Field Menu						FORM NAME: SHIPTO	
QTY	PART NO.	DESCRIPTION			UNIT	TOTAL	
qty..	partnum..	description.....			uprice.	tprice.	
Num [9]	Len [7]	Name [UPRICE	Enh [HI]	FType [R]	DType [NUM2]		
Initial Value [
*** Processing Specifications ***							
PREV FORM	NEXT FORM	FIELD TOGGLE	REFRESH	PREV	NEXT	MAIN/ RESUME	EXIT

Figure 1-4. Sample Field Menu

Data Type

This attribute 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 character set code) 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 form 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 the Globals Menu), each field is displayed in half bright, inverse video (code=HI). Other enhancements, underline, blinking, color, and security (color and security are only supported on some terminals - refer to "Supported Terminals Features" in Appendix G), can be combined with or replace the default enhancements, or you can eliminate field enhancements altogether.

Initial Value

You can specify in the Field Menu a particular value to be displayed in that field when the form is first displayed at the terminal.

The field type of the item PRICE in Figure 1-4 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 in the lower portion of the Field Menu, the area labeled *Processing Specifications*. 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 field F7 plus 5.
- Pattern Check** 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. Figure 1-5 illustrates a Field Menu that includes advanced processing specifications.

FORMSPEC v.uu.ff Field Menu				FORM NAME: SHIPTO	
QTY	PART NO.	DESCRIPTION	UNIT	TOTAL	
qty..	partnum..	description.....	uprice.	tprice.	^
Num [9]	Len [7]	Name [UPRICE]	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"					
PREV FORM	NEXT FORM	FIELD TOGGLE	REFRESH	PREV	NEXT
			MAIN/ RESUME	EXIT	

Figure 1-5. Advanced Processing Specifications

In this example, the UPRICE 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 user. Further, if a value greater than 10,000 is entered, a special next form, FORM2, will be displayed when the user presses **ENTER**.

Advanced Processing Statements

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 17,1986!

moves the date constant (delimited by exclamation points) to the current field.

Another example: 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 head form is the first form displayed.

Another example: 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: configuration CONFIG, initialization INIT, field editing FIELD, and finish FINISH.

Save Fields & 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 Field Menu.

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

Application-ready Buffer

An application-ready buffer (ARB) is a buffer passed between the application and the run-time intrinsics for data collection and display. It has these features:

- It stores data as the application will use it, which may differ from the way it appears on the screen.
- It is synchronized with other application sources, such as the program code and databases.
- It facilitates maintenance in that the user can rearrange fields on the screen without needing to modify the application program. Conversely, changes to the database affect the ARB, but not necessarily the screen.
- It supports an extended range of data types, including floating-point decimal and COBOL packed and zoned decimal.
- It simplifies coding because calls to VGETtype and VPUTtype are no longer needed: one call to VGET/PUTBUFFER is sufficient.

The application-ready buffer presents data as the application expects to use it, for example, to update a database, whereas the screen receives and displays data in the format that makes best sense to the user. The distinction between screen data and application-ready data facilitates maintenance and provides a mechanism for integrating FORMSPEC with a data dictionary if desired. If the definition of a database element changes, the ARB must be changed accordingly, but no change need necessarily be made to the screen. The designer can also change the screen layout, by altering the order of fields, for example, without having to modify the application source code.

The application receives the ARB from a VPLUS/V VGETBUFFER or sends it to the screen via VPUTBUFFER. The ARB returned to the application contains the data that the user entered, now transformed according to the designer's specifications. These may include adding, deleting and rearranging fields, as well as transforming and replacing data. A *mode* flag in the VPLUS *comarea* indicates whether the data is to be transformed or left "raw". As input to VPUTBUFFER, the ARB contains the information the application wishes to display on the user's screen.

Benefits of the ARB

The Application-Ready Buffer greatly simplifies screen/application data conversions. Previously, application programmers had to call VGET/PUTtype for each field that needed to be converted in a given transaction (or had to write code to perform conversions). Now they need only specify a default conversion for each screen and ARB data type, request automatic generation of the ARB from the form, and make any modifications that may be necessary for a particular field. This should considerably simplify the code, making it easier to read and maintain, and transaction response time should be unaffected.

The creation of an ARB is described in detail in Section 3.

PROGRAM INTERFACE

A library of high-level intrinsics is available for use by all terminal-oriented applications. These intrinsics can be called by any of the supported programming languages listed in Appendix A. Appendix A also contains listings of sample applications that use the intrinsics.

The program interface provides:

- **Terminal Interface** Intrinsics to open and close a terminal file, to load and unload forms, to display a form to the terminal, and read data entered in fields of the displayed form.
- **Forms File Interface** Intrinsics 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** Intrinsics 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.
- **Access to Data** Intrinsics 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 Transformation** Intrinsics to gather data entered by the user at the screen and transform it into the format most useful to the application, or vice versa. Data passed to or from fields can be converted to or from a variety of data types.
- **Status/Error Control** Intrinsics to set or capture error flags and display messages.
- **Batch Data Collection** Intrinsics that open and close the batch file, write data to the batch file, or read data from the batch file.

The VPLUS/V intrinsics 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 VPLUS/V intrinsics. The VPLUS/V intrinsics, as illustrated in Figure 1-6, 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 form to the Data Buffer area of memory.
- Transfer data between the Form Definition and the Data Buffer areas of memory according to processing specifications.
- Transfer data between the Data Buffer in memory and either an application program buffer area or the batch data file used for data entry.

Thus, Figure 1-6 shows an overview of the various transfers between elements controlled by the VPLUS/V intrinsics. Note that many applications use their own files rather than the VPLUS/V batch file for the collected data.

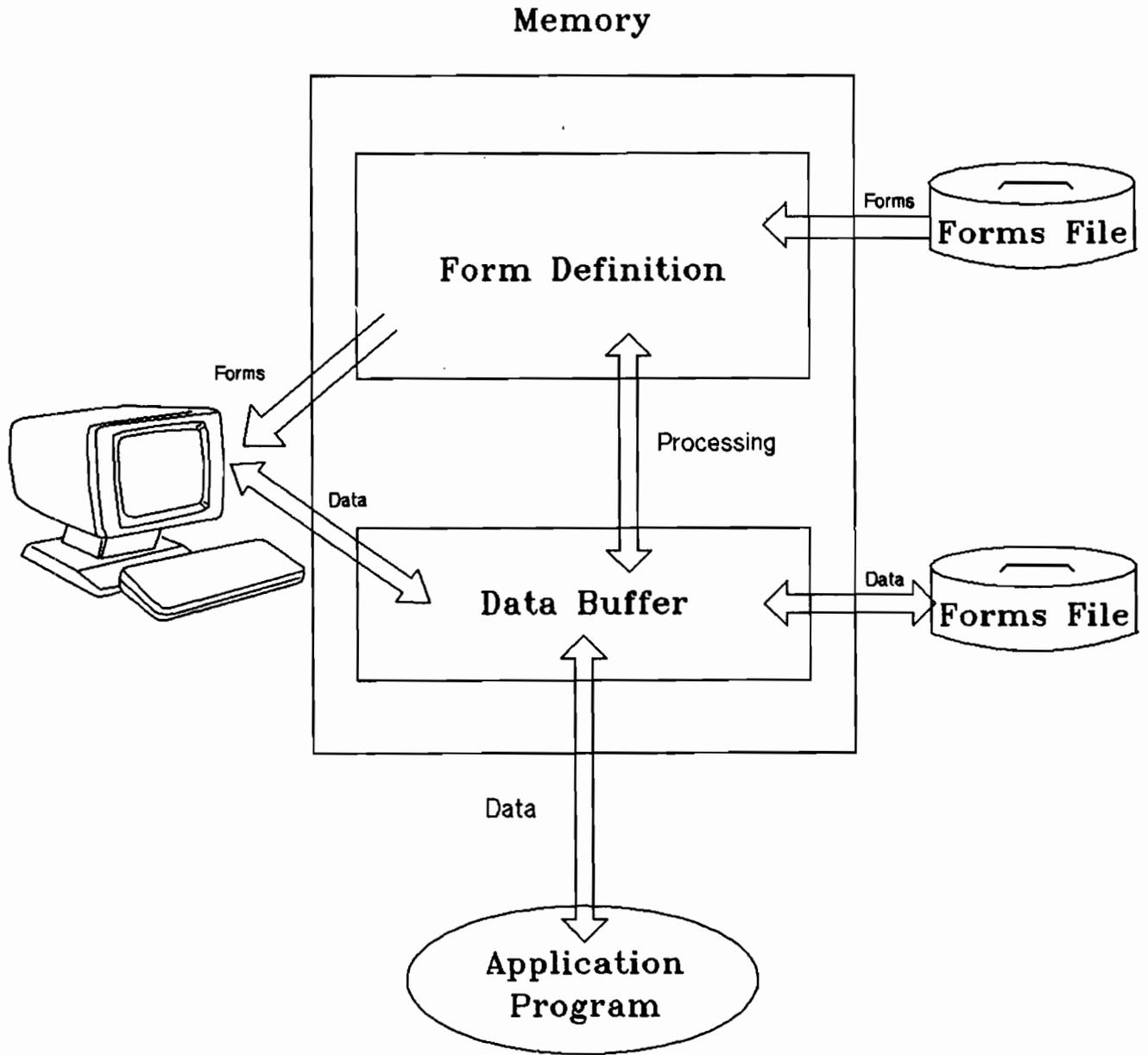


Figure 1-6. Transfers Controlled by VPLUS/V Intrinsic

DATA ENTRY PROGRAM

A stand-alone data entry program, ENTRY, is provided by VPLUS/V. This program, run by the user, 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 user 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 user types data into the "unprotected" fields on the form. (These are the fields that permit data entry; they include all fields defined on a form except "display only" fields.)

After typing in the data, the user presses **ENTER**. ENTRY 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. The default error enhancement is inverse video underline.) ENTRY also displays a message that describes the first field with an error. The user 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 user using ENTRY has some control over which forms are displayed. This control is provided by a set of terminal function keys that allow the user 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 ENTRY.

Browse/Modify

The user 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 user 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 user 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 user wants to look over the entire file, the user 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 **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 user can progress through all the records in the batch file.

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

REFORMATTING DATA

Data entered through ENTRY is written to a batch file. This file can then be used as input to an application. Sometimes it is necessary to reformat the data in the batch file so that it meets the input requirements of the application. VPLUS/V 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.

REFSPEC, described in Section 5, is used to specify how you want the batch file reformatted. This program operates very much like FORMSPEC in that it displays 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 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.

VPLUS/V provides a Data Entry Program (ENTRY) to control all data entered on forms contained in a VPLUS/V forms file. This section describes how a user enters data under control of ENTRY. (Refer to Sections 3 and 4 for a description of forms design; and to Section 6 for a description of the intrinsics with which ENTRY is implemented and can be modified).

In order to enter data, a user runs ENTRY. Then, with the terminal in block mode, ENTRY displays forms and the user can enter data into the unprotected fields displayed on the forms.

Protected and Unprotected Fields

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

Detecting Errors

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



Modifying Data

After data has been entered, the user 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 user runs 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 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 ENTRY were used to control data entry.

USING THE TERMINAL

In order to enter data through VPLUS/V, any of the terminals listed in Appendix G as supported terminals may be used. The HP 262X terminal keyboards generally conform to the one illustrated in Part A of Figure 2-1, and the HP 239X keyboards look like the one shown in Part B of that figure. Refer to your terminal reference manual for complete instructions on terminal use.



Figure 2-1. HP Terminal Keyboards

Table 2-1. Function Keys for 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 F1	Display first record in batch file on form used to enter data.
	Not used; beeps if pressed accidentally.	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 (all values typed in fields and recorded by pressing ENTER).	PRINT F3	Same as in Data Collection.
REFRESH F4	Clear screen; initialize terminal; and redisplay with initial values. On terminals with local form storage, clear form storage memory and reinitialize form storage directory. Also used to recover from "Unexpected Program Interruption", as described later in this section.	REFRESH F4	Same as in Data Collection except that previously entered data is displayed.
	Not used; beeps if pressed accidentally.	PREV REC 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 F6	Display next record in batch file on form used to enter data.
BROWSE F7	Enter browse/modify mode. The previous form is displayed with entered data; the window line shows BROWSE.	COLLECT F7	Return to Data Collection mode; the window line shows COLLECT.
EXIT F8	Exit from ENTRY; return to MPE control.	EXIT F8	Same as in Data Collection.



Function Keys

The terminal function keys are assigned special functions during data entry and subsequent data modification. Depending on whether you are entering new data into the system (data collection) or are reviewing or modifying existing data (browse/modify), these keys have slightly different meanings. The keys are illustrated in Table 2-1 followed by their function for each mode.

NOTE

The function keys do not transfer data; **ENTER** is the only one that transfers data to the data buffer.

Global function key labels for a form can be changed with FORMSPEC (as described in Section 3). The labels you define with FORMSPEC are displayed; 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 are detected. The data capture devices 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 **PRINT**. 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, the data you see on the screen is what is printed. However, if you key in data on the form and press **PRINT** before you press **ENTER**, the new data is not printed. In collect mode, only initial data displayed with the form is printed. In browse/modify mode, any data already recorded on a batch record is printed.

RUNNING ENTRY

In order to display the forms for data entry, you run 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* or *HP 3000: A Guided Tour* 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 lowercase 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 groupname are also needed, the system manager can explain how to enter them. Suppose your user name is JOAN and your account name is INVENTORY, you log on with the following command:

```
:HELLO JOAN.INVENTORY
```

Then press **RETURN** 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 following command to run 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. If you do not know the names of these files, ask your system manager.

Forms File

The forms file is a standard MPE file identified by its data file name. 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.

Entering Data with ENTRY

ENTRY 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**. If a password is associated with a forms file, ENTRY will prompt the user for it twice. This is because the VPLUS/V intrinsic must open the formsfile twice, once to determine the forms file type (KSAM or non-KSAM) and again to open it in the correct mode. If you wish to avoid being prompted twice for the forms file password, enter it along with the forms file name in the format: *formsfile/lockword*. 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 entered 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 re-issued so you can enter a different forms file name 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.

If the forms file has been designed to accept data using a native language other than NATIVE-3000 and ENTRY does not know the exact language which should be used you may receive the message:

ENTER LANGUAGE ID NUMBER AND PRESS RETURN:

If you receive this message and do not know the appropriate number, contact your system manager. For more information on Native Language Support, see Section 8.

Removing Deleted Records From a Batch

When records are deleted from a batch file, they are only "tagged" for deletion. If you wish to physically remove deleted batch records, you use the HP file copier, FCOPY. The following FCOPY operation will work:

1. Determine the record size of the batch file (in bytes) by using the MPE :LISTF command. Let this value be N.
2. The delete flag is located 19 bytes from the end of a batch record, or at location N-19. Let this value (N-19) be M.
3.

```
:RUN FCOPY.PUB.SYS
>FROM=oldbatch;TO=newbatch;SUBSET=#0,0#,M;NEW
>EXIT
```
4. If desired,

```
:PURGE oldbatch
:RENAME newbatch,oldbatch
```

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 the MPE :FILE command before running ENTRY:

```
: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 can also use FCOPY to enlarge an existing batch file. Once again, first issue the MPE :FILE command:

```
:FILE filename;DISC=numrec
```

Then run FCOPY 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 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.

Entering Data with ENTRY

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 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, ENTRY operates in "block mode". During log on, when you request ENTRY, and when you specify the forms and batch file names, your commands are entered into the system by pressing **RETURN** at the end of each line. When ENTRY operates in block mode, your data is entered only by pressing **ENTER**. This allows you to move around on the screen, pressing **RETURN** if you wish, and to type in or correct data. ENTRY 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".

Local Form Storage

Some terminals have a feature that allows forms to be stored locally in terminal memory. Forms stored in the terminal can be displayed on the screen directly rather than having to be retrieved from the memory area of the computer. This feature, with look ahead mode, is automatically activated when you run ENTRY. Refer to Appendix G for a list of supported terminals; refer to Section 6 for more information on Local Form Storage.

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**. 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 sequence have been displayed, or you press **EXIT**.

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 **NEXT FORM**. 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 **HEAD FORM**.

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.

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 rolled back down for review. Data in the frozen form can be changed only by entering the browse/modify mode with **BROWSE**.

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

Entering Data with ENTRY

HEAD FORM (frozen), press **ENTER** to append: NEXT FORM

NEXT FORM is then repeated/appended each time **ENTER** is pressed.

•
•
•

```

ENTRY v.uu.ff. Batch Record #5 Mode: Collect
Order Number: [ ] Date: [ ]
SHIP TO [ ]
QTY PART NO. DESCRIPTION UNIT PRICE NET PRICE
HEAD FORM PRINT REFRESH NEXT FORM BROWSE EXIT
    
```

Press **NEXT FORM** to stop repeat, clear screen, and display: NEXT FORM

•
•
•

```

ENTRY v.uu.ff. Batch Record #6 Mode: Collect
Order Number: [ ] Date: [ ]
TAX RATE [ ] TOTAL NET PRICE [ ]
SALES TAX [ ] SHIPPING [0.00]
TOTAL PRICE [ ]
CUSTOMER NAME [ ]
HEAD FORM PRINT REFRESH NEXT FORM BROWSE EXIT
    
```

Press **ENTER** to return to HEAD FORM.

```

ENTRY v.uu.ff. Batch Record #7 Mode: Collect
Order Number: [ ] Date: [ ]
SHIP TO: [ ]
QTY PART NO. DESCRIPTION UNIT PRICE NET PRICE
HEAD FORM PRINT REFRESH NEXT FORM BROWSE EXIT
    
```

Figure 2-2. Example of Forms Sequence.

Entering Data with ENTRY

ENTRY v.uu.ff		Batch Record #4		Mode: COLLECT	
Order Number: 100001			Date: 04/02/86		
SHIP TO:					
Name					
Address					
City		State			
Zip		Telephone			
QTY	PART NO.	DESCRIPTION	UNIT PRICE	TOTAL PRICE	
HEAD FORM		PRINT REFRESH		NEXT FORM BROWSE EXIT	

Figure 2-4. Fields Without Brackets

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, the unit price, and the part number of an 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 2-5 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, 1986, may be displayed in the same (or a different) field as 9/15/86.

1. Key in values for QTY, PART NO., DESCRIPTION and UNIT PRICE.

2. Press **ENTER**.

•
•
•

3. ENTRY calculates and displays NET PRICE.

```

ENTRY v.uu.ff          Batch Record #87          Mode: Collect
Order Number: [100081]          Date: [09/15/86]
SHIP TO [ ]
QTY  PART NO.  DESCRIPTION          UNIT PRICE  NET PRICE
[2]  [PK982-CCC] [4-INCH HINGE]  [15.00]    [ ]
HEAD FORM          PRINT  REFRESH          NEXT FORM  BROWSE  EXIT
    
```

```

ENTRY v.uu.ff          Batch Record #87          Mode: Collect
Order Number: [100081]          Date: [09/15/86]
QTY  PART NO.  DESCRIPTION          UNIT PRICE  NET PRICE
[2]  [PK982-CCC] [4-INCH HINGE]  [15.00]    [30.00]
HEAD FORM          PRINT  REFRESH          NEXT FORM  BROWSE  EXIT
    
```

Figure 2-5. Example of Data Entry

Correcting Errors

You can correct errors either before or after pressing **ENTER**. If an error is detected by ENTRY 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 form for errors before pressing **ENTER**. If you notice an error anywhere on the form, you can correct it and then press **ENTER**. Refer to your terminal reference manual for information on how to use the cursor positioning and editing keys.

All unprotected fields on the screen can be cleared to spaces or the original default values by pressing **REFRESH**.

After ENTER

Once you have pressed **ENTER**, ENTRY 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). Also, a message describing the first error is displayed in the "window" line. This line is dedicated to error and status messages, and it appears on the form in the position specified in the forms file with FORMSPEC.

You should then correct the field in 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 user 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 **ENTER** to record the data on the current form, and then press **EXIT**. The next time you run 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 **EXIT** 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 **EXIT** 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 and pressing **RETURN**, as follows:

```
:BYE
```

Unexpected Program Interruption

The program may interrupt unexpectedly because of a power failure or because you accidentally pressed **BREAK**. Refer to Appendix G for instructions on how to recover from this situation. Once you have recovered, the form displayed at the time of the failure is redisplayed, and you can continue 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 **BROWSE** to enter browse/modify mode. If you have exited from 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. ENTRY displays the next form on which you can enter data. To enter browse/modify mode in order to view data already entered, press **BROWSE**.

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

You may press **FIRST REC** 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 to be displayed:

There are no more batch records

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 records in the batch file rather than to forms in the forms file.

Resuming Data Collection

To return to data collection mode, press **COLLECT**. 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** or **NEXT REC**, 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 data generated by ENTRY 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 with you press **NEXT REC**. However, when you move backward with **PREV REC**, 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, ENTRY 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, ENTRY calculates net price from the quantity and a unit price. Refer to Figure 2-6 for an example.

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 **REFRESH** to reset the data on a form during modification. However, in browse/modify mode, ENTRY redisplay the data previously collected to the batch file, replacing 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 **DELETE REC**. Thereafter, ENTRY 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, modify the field to show all blanks and press **ENTER**.

Entering Data with ENTRY

Press **BROWSE** to enter Browse Mode and display previous batch record.

•
•
•

```

ENTRY v.uu.ff          Batch Record # 5          Mode: Browse
[ B ] [ PA123465 ] [ 4-INCH HINGE ] [ 8.50 ] [ 57.00 ]

FIRST  DELETE  PRINT  REFRESH  PREV  NEXT  COLLECT  EXIT
REC   REC     REC     REC       REC   REC
    
```

Press **PREV REC** until particular record to be modified, Batch Record #1, is displayed. Modify **TAX RATE** to 0.04. **ENTRY** recalculates **SALES TAX** and **TOTAL PRICE** and displays the next form.

•
•
•

```

ENTRY v.uu.ff          Batch Record #3          Mode: Browse
Order Number: [100008]          Date: [09/28/88]
                                TOTAL NET PRICE [ 57.00 ]
                                TAX RATE [ 0.04 ]    SALES TAX [ 2.28 ]
                                SHIPPING [ 0.90 ]    TOTAL PRICE [ 60.28 ]
CUSTOMER NAME [ ACME MANUFACTURING ]

FIRST  DELETE  PRINT  REFRESH  PREVIOUS  NEXT  COLLECT  EXIT
REC   REC     REC     REC       REC       REC
    
```

Press **PREV REC** again to see the recalculated values. Press **COLLECT** to continue data collection.

```

ENTRY v.uu.ff          Batch Record #3          Mode: Browse
Order Number: [100008]          Date: [09/28/88]
                                TOTAL NET PRICE [ 67.00 ]
                                TAX RATE [ 0.06 ]    SALES TAX [ 3.42 ]
                                SHIPPING [ 0.00 ]    TOTAL PRICE [ 60.42 ]
CUSTOMER NAME [ ACME MANUFACTURING ]

FIRST  DELETE  PRINT  REFRESH  PREVIOUS  NEXT  COLLECT  EXIT
REC   REC     REC     REC       REC       REC
    
```

Figure 2-6. Example of Data Modification.

INTRODUCTION TO FORMS DESIGN

SECTION

3

The forms displayed to a user are designed at a terminal with the interactive program, FORMSPEC. Using FORMSPEC, the designer designs the forms and defines parameters for editing, manipulating and displaying data. Forms defined in this manner are saved in a **forms file**. At any time during forms design, FORMSPEC allows the designer to modify the forms in the forms file. Once a form has been defined, the designer can create an application-ready buffer (ARB) for it. The ARB stores screen data in a format compatible with the requirements of the application program: this format may differ from the screen layout.

GETTING STARTED WITH FORMSPEC

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

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

FORMSPEC runs entirely in block mode. HP block mode terminals are placed in block mode automatically by VPLUS/V; refer to Appendix G for a list of supported terminals.

When you run FORMSPEC, it displays 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. Refer to the menu descriptions later in this section for more information on all the FORMSPEC menus.

With FORMSPEC, you press **ENTER** to record data; you use function keys to move between FORMSPEC menus and for other tasks, as described below. You can use your terminal editing keys to correct data typed into fields on the FORMSPEC menus; refer to your terminal manual for more information.

Most of the capabilities of interactive FORMSPEC are also available with batch mode FORMSPEC. (See "Using FORMSPEC in Batch Mode", Section 7 for more information.)

Specify a Forms File

The first menu displayed whenever you run FORMSPEC is the Forms File Menu, as shown in Figure 3-1. The letters v.uu.ff used for Figure 3-1, and for all the examples shown in this manual, will be replaced by the version number of VPLUS/V used on your system.

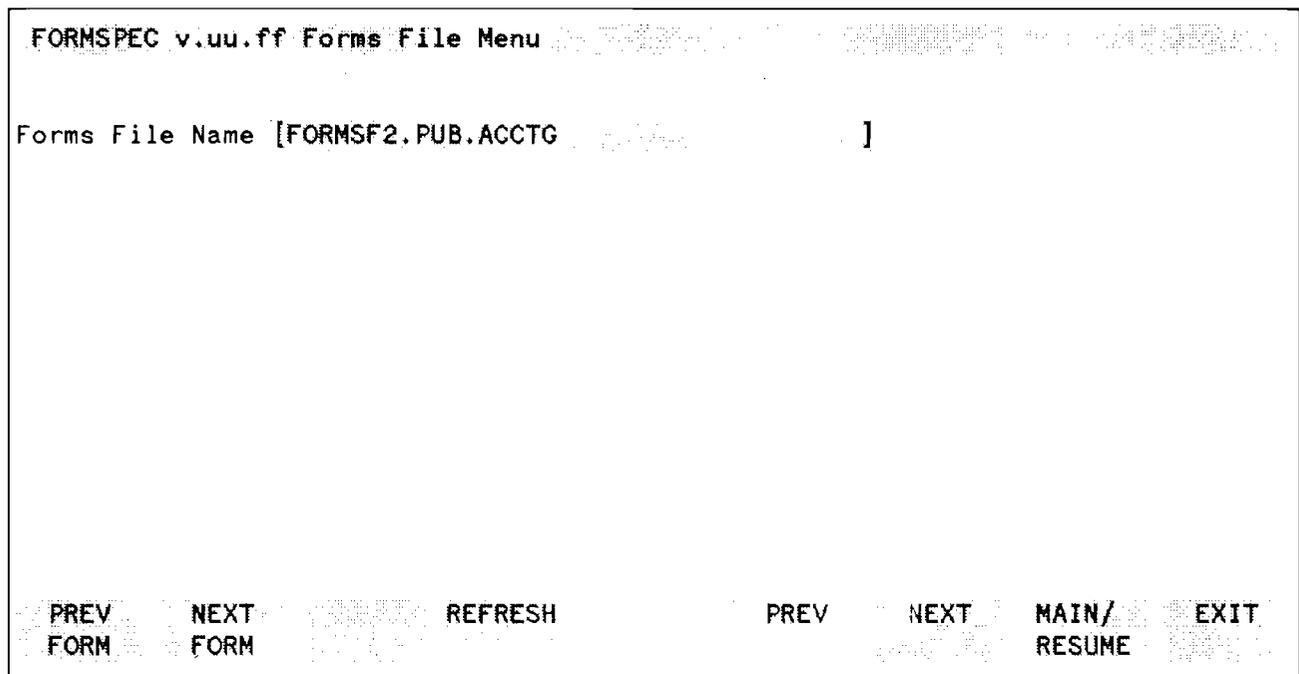


Figure 3-1. Forms File Menu

At the Forms File Menu, you specify the name of the forms file to be created or modified, such as the example FORMSF2.PUB.ACCTG shown in Figure 3-1; press **ENTER**. If the forms file does not exist, FORMSPEC prompts you to press **ENTER** again to confirm that you want to create a forms file.

Choose an Option

Once you have specified the name of the forms file and pressed **ENTER**, the next menu is the Main Menu, as shown in Figure 3-2. Notice that the top line, called the **window line**, contains useful information, which includes the version number (v.uu.ff), the title of the menu (Main Menu), and the name of the forms file (FORMSF2.PUB.ACCTG).

```

FORMSPEC v.uu.ff Main Menu                FORMS FILE: FORMF2.PUB.ACCTG

[A ] Enter Selection
                                     Forms file is 1% full
A--Add a form
S--Add a Save field
T--Terminal/Language 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 [ ]
R--Relate child form [ ] to parent form [ ]
B--Go to ARB Menu..... [ ]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM  FORM      FORM  FORM  RESUME  FORM

```

Figure 3-2. Main Menu

From the Main Menu, you have access to the menus of FORMSPEC with all of their forms design features. You simply type the option desired in the selection box and press **ENTER**. In Figure 3-2, option **A** is specified, which selects "Add a form" and leads to a sequence of menus that allow you to design a form simply and easily. This process is described later under "Ease of Forms Design".

Many of the options, including **L**, **D**, **C**, and **X**, only require information in the fields on the Main Menu in order to be executed immediately. Option **B** lets you create and modify an application-ready buffer for a particular form.

Some options control the more advanced features of FORMSPEC, such as "Form Families" and "Using Terminal Features"; discussed later in this section. All options are described under "Main Menu" in this section.

FORMSPEC Function Keys

In addition to the control provided by the Main Menu, the defined function keys, shown in Table 3-1, allow you to select menus for initial definition of an existing forms file, or for modifying it. The Main Menu and function keys together give you the ability to change any form, field, or global specification as you define the forms file.

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.
FIELD TOGGLE F3	Only used in Field Menus to switch between field attributes and the optional processing specifications.
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 unexpected program interruption (refer to Appendix G).
PREV F5	Displays previous menu in sequence of menus (refer to "Menu Sequence").
NEXT F6	Displays next menu in sequence of menus (refer to "Menu Sequence").
MAIN/ RESUME F7	Request Main Menu or, if Main Menu displayed, return to menu displayed when MAIN/RESUME 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 defined by an application or by ENTRY for use while the application is executing, as discussed under "Advanced Forms Design" later in this section. Although the two groups of keys are physically the same, their functions differ in most cases.

Menu Sequence

FORMSPEC displays its menus in a predetermined sequence. As each menu is displayed, 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 function keys (**NEXT FORM**, **PREV FORM**, **NEXT**, and **PREV**). These keys allow you to select menus relative to the current menu in order to make changes or additions. Figure 3-3 illustrates the relation between the menus provided by FORMSPEC and the function keys that control menu sequence.

You can move forwards or backwards through the sequence of menus to locate a particular menu by pressing **NEXT** or **PREV**, respectively. If you want to skip the field menus associated with each form, you can use the **PREV FORM** or **NEXT FORM**. An alternate method is to request the Main Menu with **MAIN/RESUME**, and then select an existing menu.

Although it is not technically a menu, the form layout associated with each form is treated as the menu following the Forms Menu when **NEXT** is pressed, and as the menu preceding the first Field Menu when **PREV** is pressed.

The sequence control keys can also be used from the Main Menu. You can go to the Main Menu at any time from any other menu by pressing **MAIN/RESUME**. You can return to the menu from which you requested the Main Menu by pressing **MAIN/RESUME**. In this case, the menu on the screen when **MAIN/RESUME** was requested is treated as the current menu. If you have just entered FORMSPEC and the Main Menu is displayed, pressing **MAIN/RESUME** results in a display of the Globals Menu. From the Globals Menu, **NEXT FORM** causes the first Form Menu to be displayed; **NEXT** causes a Save Field Menu to be displayed.

NOTE

The number of Field Menus displayed depends on the number of fields you defined on the screen for the form. The system automatically displays 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 displayed, the Form Menu is redisplayed. If you have no more forms to design, press **MAIN/RESUME** for the Main Menu through which you compile the forms file. Otherwise, continue defining forms by specifying the next form on the Form Menu.



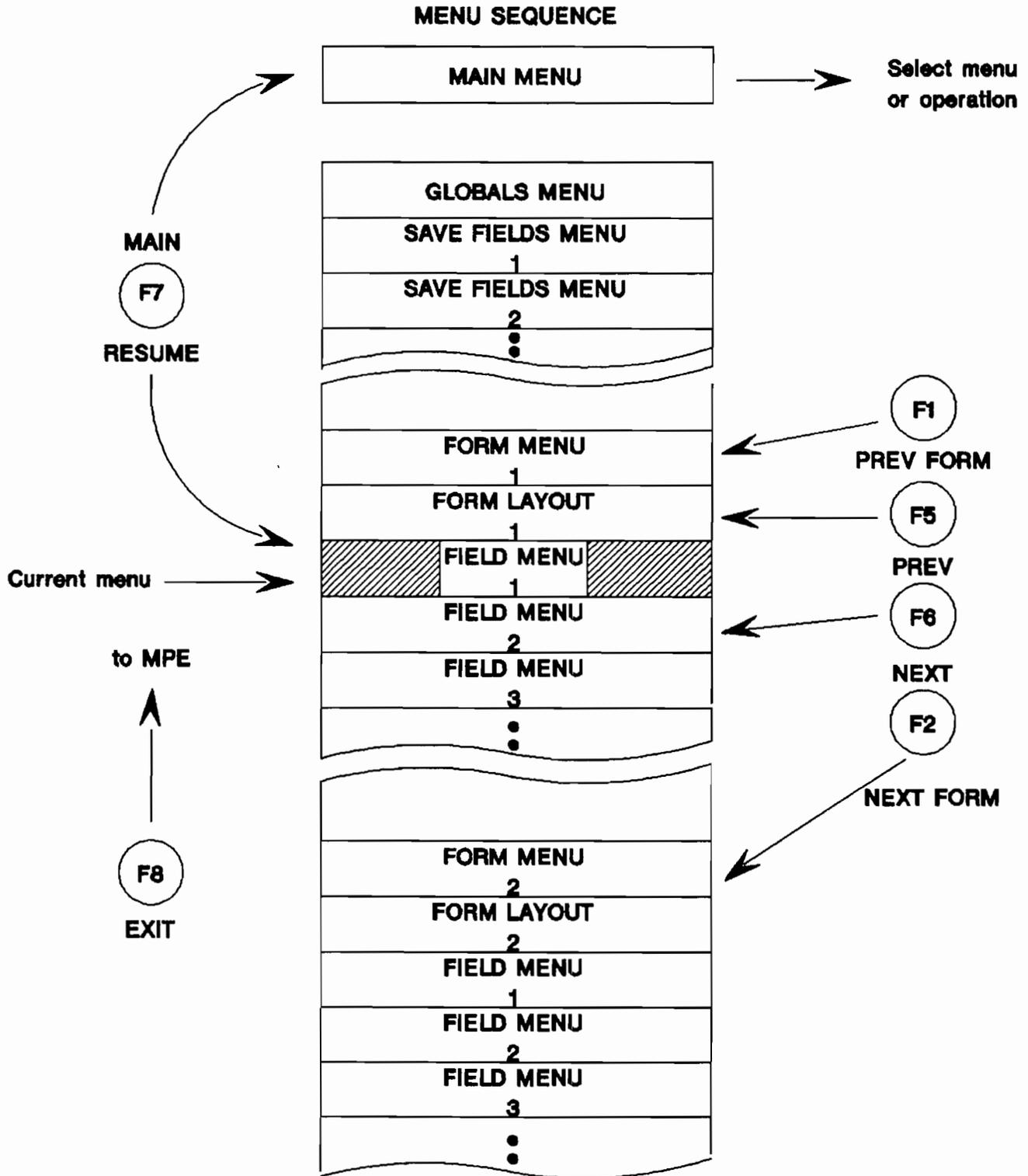


Figure 3-3. Relation Between Menus & Function Keys

Creating an Application-ready Buffer (ARB)

The purpose of the ARB is to allow you to transform screen data into application data and back again by converting data types between screen and application, and by reordering the fields.

Once you have created a form and saved it in the forms file, you can create an application-ready buffer (ARB) for it if you choose. Not every form needs to have an ARB; you can create ARBs for selected forms only. You can also create an ARB for a form at any time after you have created the form itself, and you can delete an existing ARB and generate a new one if you have made extensive changes to the associated form. Fields on the form are mapped one-to-one to the ARB when you select **GEN** from the ARB Menu. You can rearrange fields on the form using the Screen Design Menu without affecting the ARB, and you can add and delete ARB fields without affecting the form by selecting **RES** (Restructure) from the ARB Menu.

Before you can generate an ARB, you must define the Data Type Conversions you want to take place between the form on the screen and the ARB. From the Globals Menu, you access the Data Type Conversions menu and define the data conversion defaults. Screen and application data types are listed under "Data Type" further on in this section. Creation of an ARB is described in detail under "Using FORMSPEC to Create an ARB" in this section.

At runtime, the ARB transforms the data as it appears on the screen into a format that the application can use without further manipulation, and converts application data back into a format suitable for display on the screen.

Terminating FORMSPEC

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

However, if you have made any modifications and have not compiled the forms file, FORMSPEC issues a warning. By pressing **EXIT** again you terminate operation of FORMSPEC without compiling the file. Refer to "Forms File" for more information on compiling the forms file.

Unexpected Program Interruption

In case of unexpected interruption due to hitting **BREAK** or a terminal power failure, control returns to MPE. Refer to Appendix G for the steps to recovery from such a situation. Once you have recovered, the menu will be cleared to initial or previously entered values. To ensure 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.

EASE OF FORMS DESIGN

Once you have specified a forms file and the Main Menu is displayed, you are ready to use FORMSPEC to create forms. Whenever you create a new forms file, your first task is always to add a form; if you are modifying an existing file, you may want to add a form. In either case, you follow the sequence of menus shown in Figure 3-4. Refer to the menu descriptions later in this section for more information on all the FORMSPEC menus.

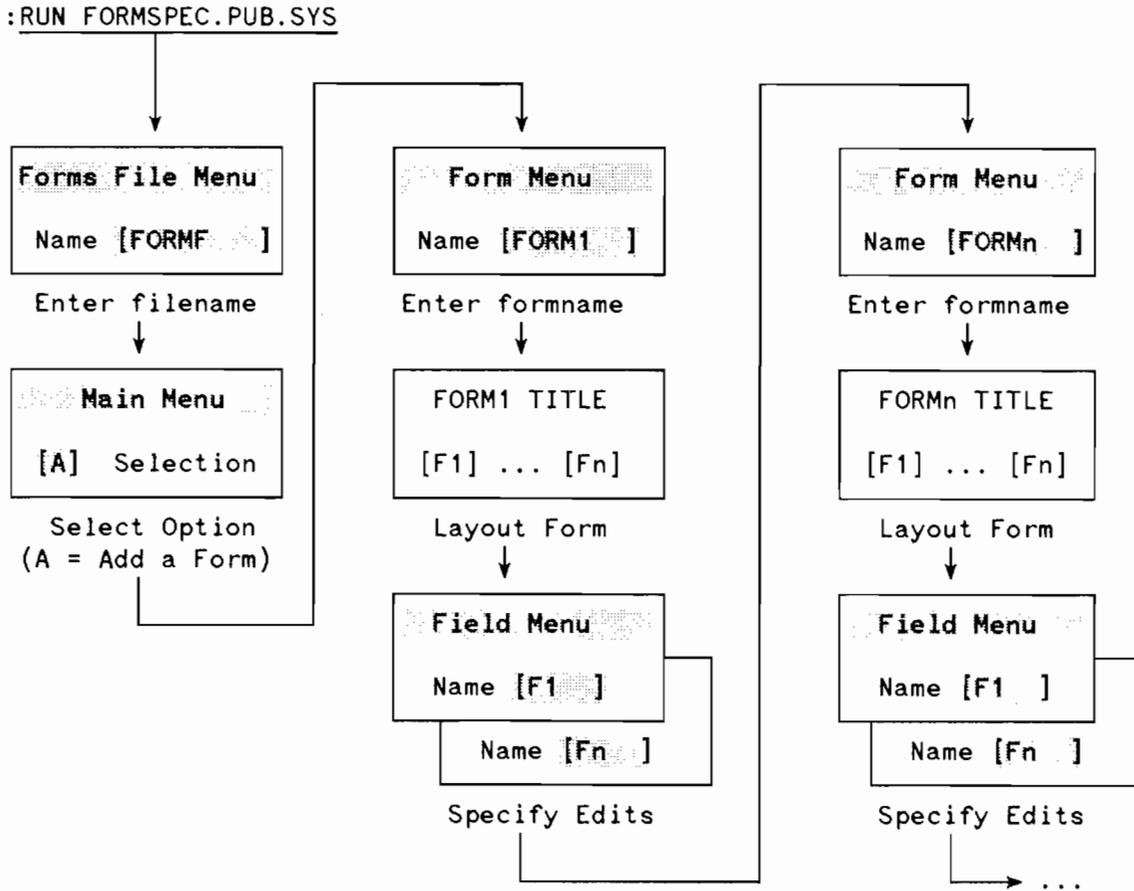


Figure 3-4. Sequence of Menus for Form Design

A Form Menu, shown later in Figure 3-10, allows you to define the characteristics of each form, such as the form name and what form sequencing options to use. Refer to "Understanding Form Sequencing" below.

Form Layout

The Form Menu is followed by a blank screen on which you design the layout of the form as it will appear on the screen. An example of a form layout is shown in Figure 3-5. This layout is easy to draw on the screen -- and easy to change at this stage or later. You use your terminal editing keys to layout the fields and text of your form on the screen; refer to your terminal manual for more information.

```

*****ABC MANUFACTURING*****
                                     Date: [ordate      ]

SHIP TO:
  Name [name                          ]
Address [address                        ]
  City [city                          ] State [st]
  Zip  [zip        ] Telephone [phone   ]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM  FORM     FORM  FORM  RESUME  FORM

```

Figure 3-5. Example of a Form Layout

Each layout consists of two kinds of information:

- Fields** Areas, delimited on the form, into which data will be entered by or displayed to the user. The maximum number of fields allowed on any one form is 128.
- Text** The headings and other displayed information that appears on the form but is never altered during execution.

You must distinguish between these two kinds of information within the form layout using field delimiters and field tags to indicate the fields.

Field Delimiters

The data fields are delimited by brackets (**[]**), by **ESCAPE** followed by brackets, or by a combination of these. Pressing **ESCAPE** then bracket prevents the brackets from being displayed on the screen.

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 form. If you must concatenate two data fields, you should use nonprinting delimiters to delimit the fields.

Nonprinting Delimiters. You can also delimit fields by pressing **ESCAPE** followed by:

the left bracket (**[**) OR the right bracket (**]**).

The advantage of these delimiters is that they take up no space on the form 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 form design.

Mixing Printing and Nonprinting Delimiters. To use one printing and one nonprinting delimiter to fix the boundaries of a field, use the delimiters as follows:

If the printing delimiter is to come first, use

```
[ (ESCAPE) [fieldtag. . .(ESCAPE) ]
```

If the printing delimiter is to come last, use

```
(ESCAPE) [fieldtag. . .(ESCAPE) ]
```

In the case where a field begins with (ESCAPE) [and ends with], the terminal will insert (ESCAPE)] if a following field also starts with (ESCAPE) [. As a result VPLUS/V will reject the form layout. This terminal feature can be avoided by properly using the above rules for mixing delimiters. Refer to Figure 3-6 for examples of different ways to layout a form.

A. Using Brackets as Field Delimiters:

```
SHIP TO:  
Name [name.....|  
Address [address .....|  
City [city.....| State [st|  
Zip [zip.....| Telephone [phone .....
```

B. Using (ESCAPE) [and (ESCAPE)] as Field Delimiters:

```
SHIP TO: (ESCAPE) [  
Name name.....|  
Address address .....| (ESCAPE) [  
City city.....| State st. |  
Zip zip.....| Telephone phone .....
```

C. Combining Brackets with (ESCAPE) [and (ESCAPE)]

```
SHIP TO: (ESCAPE) [  
Name [name.....|  
Address [address .....| (ESCAPE) [  
City [city.....| State [st. |  
Zip [zip.....| Telephone [phone .....
```

Figure 3-6. Examples of Form Layouts

Each Field Menu displays the portion of the form containing the current field. Compare Figures 3-5 and 3-7; notice the portion of the form layout of Figure 3-5 that is displayed in the Field Menu in Figure 3-7. The current field is indicated by a caret (^) under the field tag. In the example in Figure 3-7, the current field is labeled Date: with the field tag of ordate.

The information you enter on the Field Menu is divided into two categories: field attributes and processing specifications. The field attributes are the two lines of fields below the portion containing the current field. For example:

```
Num [7] Len [18] Name [ORDATE] Enh [HI] FType [0] DType [CHAR]
Initial Value [ ]
```

Figure 3-8. The Field Attributes

Only the field attributes are discussed here. The processing specifications are described in Section 4. If you can use the field attributes without change, the entire form design is complete at this point. If not, you can specify any simple edits, as described below, or, if you want to use the FORMSPEC processing specifications, described in Section 4, you can enter appropriate processing specifications on the Field Menu. As shown in Figure 3-4, the form design sequence can be repeated until all forms in the file are defined.

Field Number

Although your form may have a maximum of 128 fields, a number between one and 256 is assigned by FORMSPEC to each new field. This number is 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 field numbers of a form can be renumbered in screen order with the FORMSPEC batch mode command RENUMBER, as described in Section 7.

Field Length

This is the length of the field as determined by the number of characters entered between the field delimiters during form layout. The length of a field cannot be changed on the Field Menu. If you want to change field length, you must display the form layout and actually change the field on the form. If you change the field length on the form, the new length is automatically reflected in the Field Menus.

If you want to design a field that is longer than one line, you start the field with a bracket (or `ESCAPE` `[`) and terminate it with a closing bracket (or `ESCAPE` `]`). At the beginning of each intermediate line of the field, you enter an `ESCAPE` `[`.

For example, as shown in Figure 3-9, consider a field that extends across three lines.

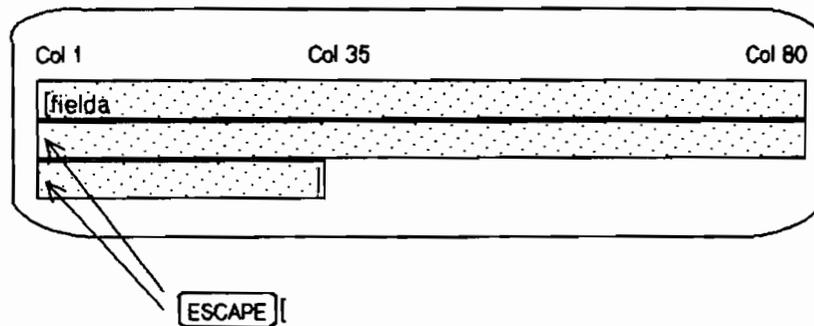


Figure 3-9. Example of a Field Extending Over Several Lines

Assuming the first line of the field contains 79 characters, the second line contains 80 characters, and the third line contains 34 characters, the entire field is 193 characters long. This count excludes the printing brackets which delimit the beginning and the end of the field.

Field Name

The field tag assigned during form design is shifted up to all uppercase letters and becomes the field name. 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. It must not be one of the reserved words listed in Table 3-2.

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	IF	MINLEN	UPC
CHANGE	ILV	NE	UPSHIFT
CLEAR	IN	NFORM	\$EMPTY
COD	INIT	NIN	\$END
CONFIG	I25	NOCUT	\$HEAD
CUT	I39	NONE	\$LENGTH
DEVICE	JUSTIFY	NOREPEAT	\$REFRESH
DISPLAY	KEYBOARD	OF	\$RETURN
EAN	LARGECHAR	PRINTER	\$STATE
ELSE	LE	RELAY	\$TODAY
EQ	LEADING	REPEAT	
FAIL	LEFT	RIGHT	
FIELD	LIGHT	SET	
	LOCALEDITS	STDCHAR	

You can enter any name up to 15 characters long. Like other FORMSPEC names, it must be USASCII and start with an alphabetic character. It may be followed by uppercase or lowercase letters (A-Z), numbers (0-9), or an underline (_).

Default The uppercase field tag.

Display Enhancement

You can change the enhancement for the particular field with any combination of the codes shown in Table 3-3. Up to four of the available enhancements may be used at one time. The data capture devices do not support display enhancements.

Table 3-3. Display Enhancement

Enhancement	Code
Half Bright	H
Inverse video	I
Underline	U
Blinking	B
Security	S *
Color	1-8 *
None	NONE

* Security and color enhancements are only available on terminals with the security or color feature, as listed in Appendix G. Refer to "Using Terminal Features" for more information on security and color.

Field Type

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

Display(D) Field cannot be modified by the user. Intended for display only, the field can receive data as specified by processing specifications (see Section 4). It is a protected field on the form, but data is written to display only fields exactly as if it had been entered by the user.

Required(R) Field cannot be left blank. User must enter nonblank values in field.

Optional(O) Field may be left blank by user. Edit checks and field phase processing specifications for the field are skipped if the field is left blank.

Process(P) 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 treated as "input" fields. An input field is one in which the user can enter or change data, as opposed to a display only field which is protected from user input. Note that initial values may be displayed in any field, but such values can be, and usually will be, changed by the user.

initial values may be displayed in any field, but such values can be, and usually will be, changed by the user.

Data Type

There are two sets of data types, the **screen** data types and the **ARB** data types. If you are creating an ARB for a form, the application needs to know the ARB data types. Each field on the form and the ARB 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.

Screen data types impose format and edit rules for data entered on the screen. ARB data types determine how that data will be interpreted by the application. Conversions from screen data type to ARB data type and vice versa occur automatically at runtime (see VGET/PUTBUFFER in section 7).

Screen Data Type. Valid screen data types are: CHAR, NUM[*n*], DIG, IMP n , MDY, DMY, and YMD. Table 3-4 illustrates how each numeric data type interprets an entered value on the screen. The data types are described in the following paragraphs.

Default CHAR

Character Type.

CHAR Data entered in the field is assumed to be a string of any characters. No validity checking is performed on data at time of entry. No arithmetic operations can be performed on data of this type. The user can enter any characters in a CHAR type field. (As noted above, CHAR is the default data type.)

For example, \$12.59, A-15-75, and **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 leading 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. When Native Language Support is used, the symbols used to indicate thousands and decimals are language-dependent. For more information on Native Language Support, see Section 8.

The user must enter numeric data in this field. If *n* specifies a maximum number of decimal positions, the user 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 user can enter a value with no decimal places (such as 500) or 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 and 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 an optional leading sign (+ or -) 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.

With Native Language Support, the symbols accepted for thousands and accepted or required for decimal indication are language dependent. For more information on Native Language Support, see Section 8.

Table 3-4. How Each Numeric Data Type Interprets Entered

Values

Entered Value	Interpreted Value (based on data type)				
	NUM2	IMP2	NUM	NUM0	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, 1986 or 02/06/86 or FEBRUARY 6 86 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 1986 or 02-03-86 or 2/3/86 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 1986, APRIL 12 or 86/4/12 or 86-04-1 or 860402.

The entered data is checked by **VPLUS/V** for correct format and that it is a valid date. Arithmetic operations are not allowed on date fields.

Native Language Support does not affect the order of the date field. It does, however, accept the names of months and their abbreviations for each native language at run-time. For more information on Native Language Support, see Section 8.

To illustrate how the three date type specifications interpret entered dates for **NATIVE-3000**, Table 3-5 shows legal dates followed by Table 3-6 with illegal dates.

Table 3-5. Valid Dates

MDY	DMY	YMD
February 7, 1986	7 February 1986	1986, February 7
FEB 7 1986	7 FEB 1986	1986 FEB 7
02/07/86	07/02/86	86/02/07
2/7/86	7/2/86	86/2/7
02-7-86	07-2-86	86-2-07
2 7 86	7 2 86	86 2 7
020786	070286	860207

Table 3-6. Invalid Dates

MDY	DMY	YMD	Reason
February 7, 1986	7 Febrary 1986	1986, February 7	Misspelled
FEBR 7 1986	7 FEBR 1986	1986 FEBR 7	Four letter abbreviation
2786	7286	8627	Must be a two-digit month, day when no separator is included.

ARB Data Types. The valid choices for ARB data types are: CHAR, INT, DINT, REAL, LONG, SPACK n , PACK n , SZONEn, ZONEn, and YYMMDD. Data type and length may differ from screen to ARB. The ARB data types include some that are language-specific. For example, SPACK n and PACK n correspond to COBOL COMP-3; SZONEn and ZONEn correspond to COBOL unsigned display numeric and signed display numeric respectively; and INT and DINT correspond to COBOL COMP. For more information on COBOL data types, see the *COBOLII/3000 Reference Manual*. The default data type and length for an ARB field are derived from the data type and length of the corresponding field on the associated form and the Data Type Conversion record.

Table 3-4a sets out recommended guidelines for data type conversions from screen to ARB and back.

Table 3-4a. Recommended Data Type Conversions

Screen Data Type	--TO-->	ARB Data Type
CHAR		CHAR only
YMD, DMY, MDY		YYMMDD, CHAR
DIG		any except YYMMDD, CHAR
NUM[n], IMPn		any except YYMMDD, CHAR; "n" truncated if INT, DINT
ARB Data Type	--TO-->	Screen Data Type
CHAR		CHAR
YYMMDD		MDY, YMD, DMY, CHAR, DIG
INT, DINT		DIG, NUM[n], IMPn; positive only if DIG
REAL, LONG, PACK, ZONE		DIG, NUM[n], IMPn; if DIG - "n" truncated, positive only

FORMSPEC will ensure the following length specifications for ARB data types:

- YYMMDD length = 6
- INT length = 2
- DINT length = 4
- REAL length = 4
- LONG length = 8

YYMMDD is defined as a 6-byte ASCII field containing numeric data with no separators, in YMD order; for example, 860419.

The designer is responsible for making legitimate data conversions. The three critical factors are **data type, value, and length**. The following examples illustrate their importance:

- **Data Type:** Runtime errors may arise from specifying a CHAR/DATE source conversion to a numeric destination, or a CHAR source to a DATE destination.
- **Value:** The ARB data type INT may have a screen type of CHAR. This works if the field is for display only, but if it is an entry field, the user could input ABC unless the designer has taken steps to prevent it.
- **Length:** Data may be truncated if the screen data type DIG is specified for a field of length 10 and the ARB type is INT (length 2 bytes, equal to 1 HP 3000 word), since the largest number COBOL can store in one word is 4 digits and Pascal, 5. If the user enters 9 digits, a runtime error will result.

NOTE

These are recommendations only, and are not enforced by FORMSPEC edits. The programmer may set up the application code to handle non-standard conversions.

Table 3-4b shows the valid screen data types and their corresponding values for COBOL, Pascal and FORTRAN.

Table 3-4b. Valid Screen Data Types

ARB Data Type	COBOL	Pascal	FORTRAN
CHAR	PIC X (_)	Packed array CHAR [. . _]	CHAR
YMD	as above	as above	as above
DMY	as above	as above	as above
DIG	as above	as above	as above
NUM[n]	as above	as above	as above
IMP[n]	as above	as above	as above

Note: [n] represents the number of decimal digits

Table 3-4c shows valid application data types and their values in COBOL, Pascal and FORTRAN.

Table 3-4c. Valid Application Data Types

FORMSPEC	COBOL	Pascal	FORTRAN
CHAR	X	Packed array CHAR [.._]	CHAR * <u> </u>
YYMMDD	X(6)	as above	CHAR *6xx
ZONE _n	9(<u> </u>). _n	---	---
PACK _n	9(<u> </u>)V _n COMP-3	---	---
SPACK _n	S9(-)V _n COMP-3	---	---
REAL	---	REAL	REAL
LONG	---	LONG	Double Precision
INT	S9(4) COMP	Subrange -32768..32767	Integer **2
DINT	S9(9) COMP	Integer	Integer **4

Note: [n] represents the number of decimal digits

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 **REFRESH** is pressed during data collection in **ENTRY** and the form is cleared to its initial values.

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

(Refer to Section 4 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.)

If Native Language Support is used, you must specify the initial value in NATIVE-3000. The value will appear with the conventions of the native language at run-time. For more information on Native Language Support, see Section 8.

Understanding Form Sequencing

As shown in Figure 3-4, the Form Menu is the first menu in the sequence of menus used to define a form. On the Form Menu, as shown in Figure 3-10, you specify a name for the form and the form sequencing options to use (Repeat Option and Next Form). The form sequencing options are described below. The other fields pertain to advanced features such as "Form Families" discussed later in this section.

```

FORMSPEC v.uu.ff Form Menu                                FORMS FILE: FORMSF2.PUB.ACCTG

Form Name [FORM1 ]

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

Next Form [C] Name [$HEAD ]
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 [ ]

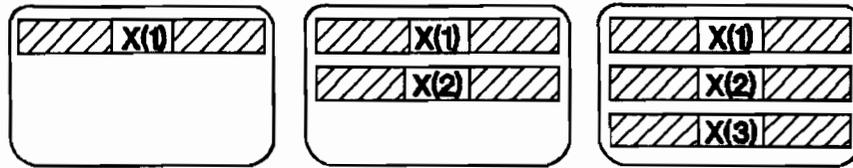
PREV NEXT REFRESH PREV NEXT MAIN/ EXIT
FORM FORM RESUME
    
```

Figure 3-10. Example of a Form Menu

The Repeat Option

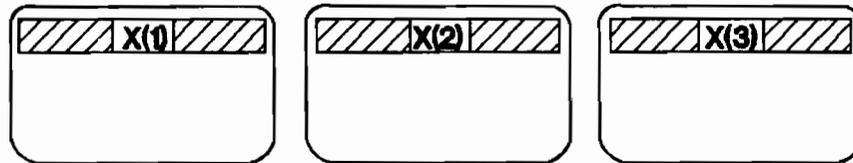
When the forms in a forms file are displayed, a form may be repeated and appended to itself (A); it may be repeated overlaying the previous display of itself (R); or it can be a non-repeating form that is displayed once (N).

To illustrate how the three choices of the repeat option work, assume that form X is a repeat/append form (A). This form is displayed, the user 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 user 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.

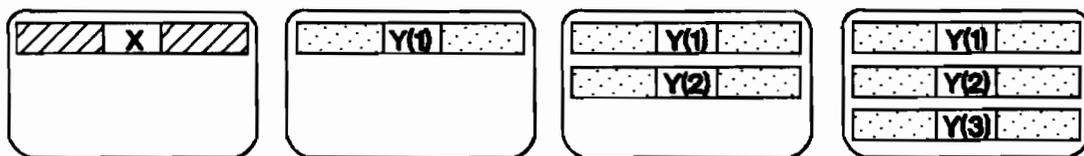
The Next Form Option

You specify the name of the next form to be displayed after the current form or keep the default of \$HEAD. 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 frozen on the screen when the screen fills up (F). If the screen is to be cleared before the next form is displayed, keep the default of C.

The following examples illustrate how freeze (F) and append (A) interact 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).

- Current Form X -- Repeat Option = N
Next Form Option = C

Next Form Y -- Repeat Option = A



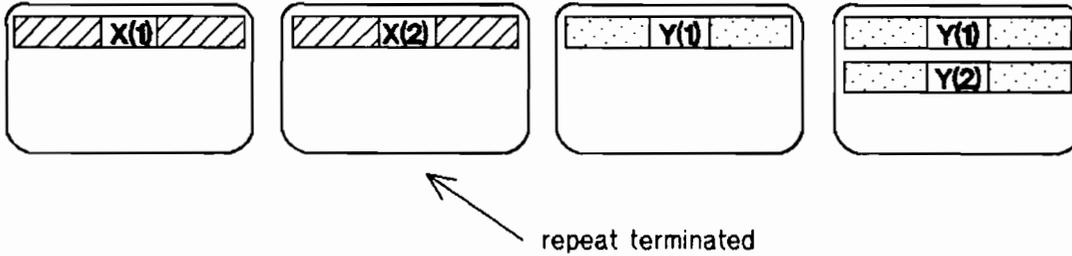
← screen cleared before next form displayed

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 user presses **NEXT** or the application changes the repeat option.

Introduction To Forms Design

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

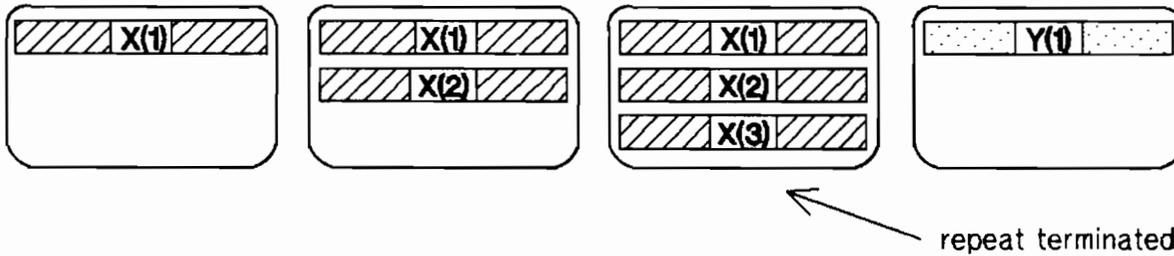
Next Form Y -- Repeat Option = A



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

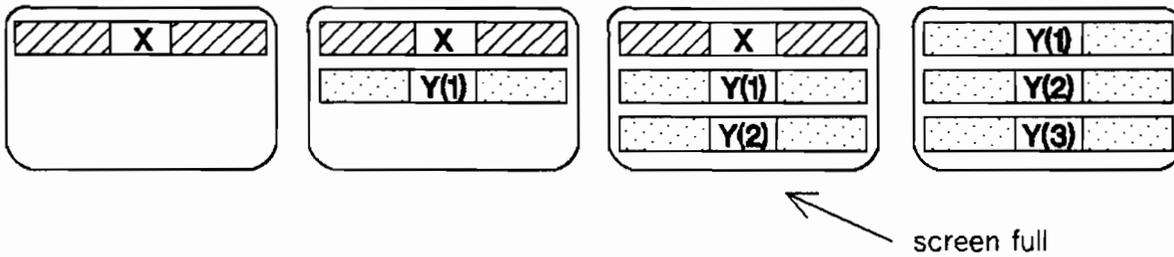
Next Form Y -- Repeat Option = A



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

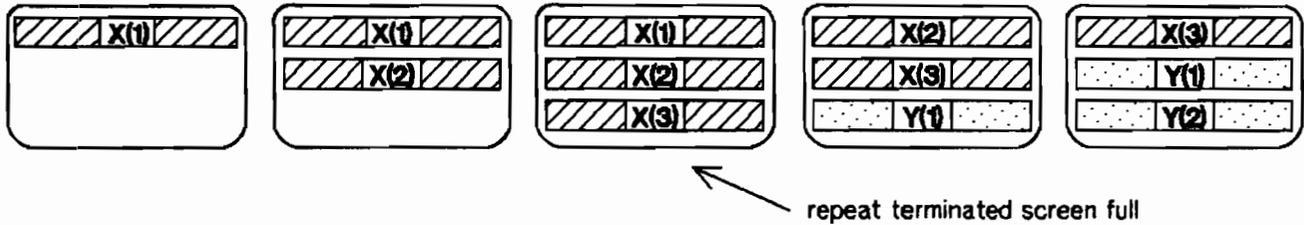
Next Form Y -- Repeat Option = A



X remains on the screen while Y forms (in this example, two copies of Y) are appended to it until no room is left on the screen. When the third copy of Y is appended, form X (or part of it) is rolled off the screen and deleted to make room. However, 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

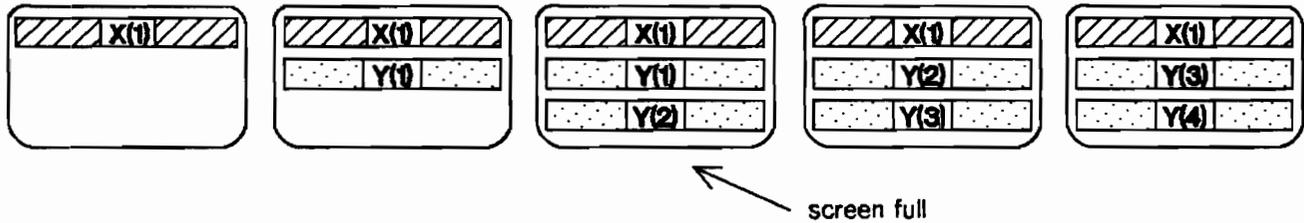
Next Form Y -- Repeat Option = A



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

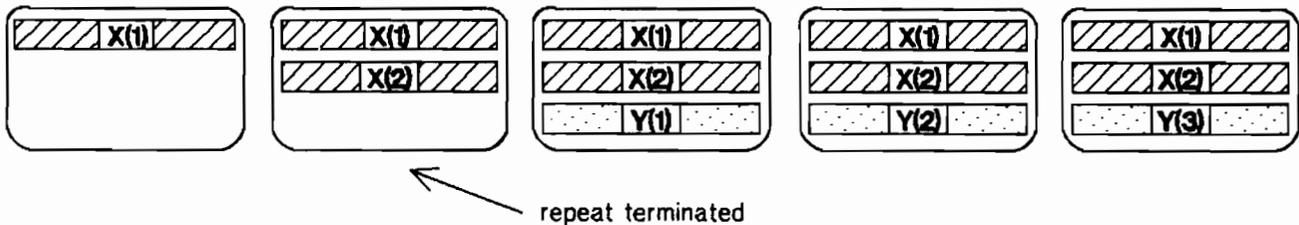


Form X remains frozen on the screen as form Y is appended to it. When the screen is filled, X remains frozen on the screen while the oldest Y form, Y(1) is rolled off to make room for Y(3). Note that if Y(1) fills the screen below X, it will be rolled off the screen and deleted when Y(2) is displayed.

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 one 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.

Sample of Forms Design

Before running FORMSPEC to define your forms file, it is very helpful to decide on some of the basics of forms design, such as:

- Consider what kind of data the user will be entering -- character, numeric, or date.
- Determine the position of the fields -- what order, how many on a line, which need to be on the same line, the same form.
- Decide on which enhancements to use for each field.
- Take into account what type of terminal the users will have. Are there any special features? Will they be used?
- What native language will be used?
- Will the forms file be used with ENTRY or with an application? How will form sequencing be handled? Do function keys need to be defined? If so, which ones?

Not all of these questions need be answered prior to forms design, since FORMSPEC provides defaults for most form and field characteristics. Still, you may find it helpful to roughly sketch each form layout on paper, something along the lines of the example in Figure 3-11. The field name and length should be noted, and if a field has special (nondefault) characteristics, these too may be noted on your preliminary sketch, as was done in Figure 3-11. Preparing your forms layout in this way allows you to then sit down at the terminal and actually specify the complete forms file in a matter of minutes. You may find that completing the form layout and accompanying Field Menus is sufficient to define the characteristics of all fields on the form. In many cases, the default values supplied by FORMSPEC can be used, thereby reducing your actual input to a minimum. However, as you become familiar with the capabilities of VPLUS/V, including the processing specifications of FORMSPEC, described in Section 4, and the intrinsics

available to applications, described in Section 6, you may find yourself taking advantage of the additional options as you finalize the design of your forms file.

The example in Figure 3-11 answers many of the questions listed above, both graphically and in accompanying notes. This forms design, when entered through the FORMSPEC menus, will generate a set of forms similar to those used as a data entry example with ENTRY in Section 2. (What data will your application need? How should it be laid out?) For forms sequencing, note that the first form is "frozen" on the screen, the second form is appended to the first and is repeated until the user presses **NEXT** to display the next and last form, TOTALS. (In this example, since ENTRY is used, the ENTRY function keys are used as well as the forms sequencing options of the Form Menu. Is this true for your design?) 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. (How will you handle multiple forms?) These are some of the questions you should consider when designing your own forms file.

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 Form = ORDER

Layout:

					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 3-11. Sample Forms File Layout

FORM #2

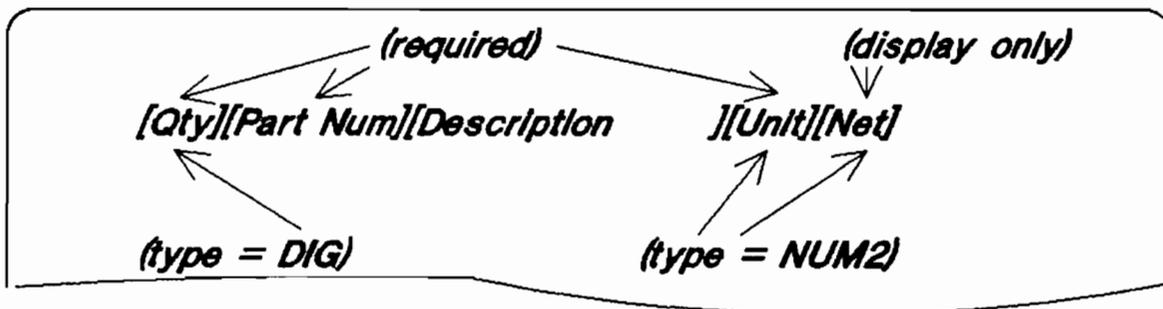
Name = ORDER

Repeat = A ← form is repeated and appended

freeze/append = C ← When next form is requested by user or program, current form is cleared.

Next Form = TOTALS

Layout:



FORM #3

Name = TOTALS

Repeat = N

freeze/append = C ← current form cleared after ENTER

Next Form = \$HEAD ← returns to first form

Layout:

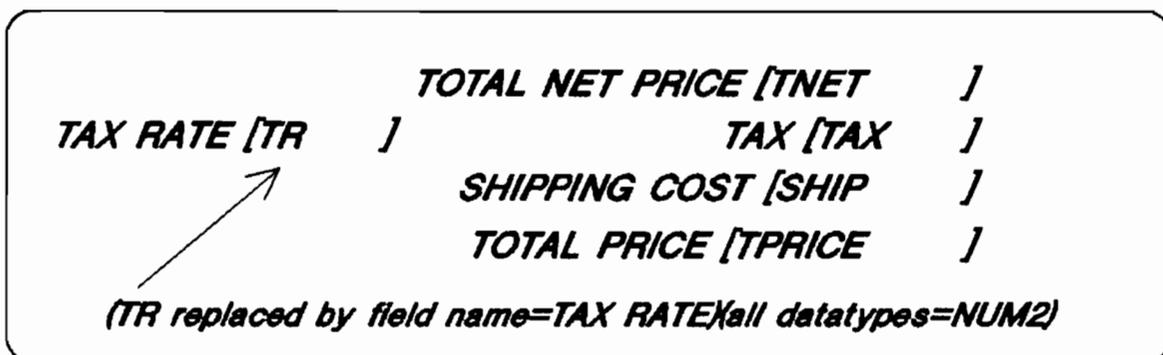


Figure 3-11. Sample Forms File Layout (continued)

FORMS FILE

The forms file consists of "global" specifications that apply to all forms in the file, followed by the individual forms specification. Within each form specification, the form is identified, and its form layout is defined. The form layout defines each field into which data can be entered. Each of these different types of specification is entered on a menu displayed by FORMSPEC. (Refer to Figure 3-12 for an illustration of a prototype forms file.)

Forms Modification

At any time during forms design, you can change any form or field currently specified in the forms file. The 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. Refer to "Ease of Forms Design" earlier in this section.

Another method of modifying a form file is to use the delete option on Main Menu. This option allows you to delete an entire form or to delete fields global to all forms (save fields). 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. Note that when you delete a parent form, all child forms of that form family will be deleted also. Refer to the "Main Menu" menu description and the "Form Families" discussion for more information.

The copy option on the Main Menu provides yet another method of modifying a form file. This useful option allows you to copy an entire form from the current forms file or from another forms file. The copied form is created by FORMSPEC as a new form that is an exact replica of the existing form. The form being copied must be given a name unique to the current forms file. The copied form can then be displayed and modified to suit your design.

The copy option does 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. Also, form family relationships are not maintained when forms are copied from one file to another. You may re-establish form family relationships using the relate option on the Main Menu. Refer to the "Main Menu" description for more information.

Most of the modifications can also be made using FORMSPEC in batch mode. (See Section 7 for more information.)

Forms File Size

FORMSPEC indicates what percentage of the forms file is full whenever the Main Menu is displayed from the Forms File Menu. The percentage is determined by the formula:

$$\text{file EOF} / \text{file limit}$$

This message is designed to give the user an idea of the space utilized in the forms file. Since a single forms file can contain an essentially unlimited number of forms (subject to system constraints), the forms file may reach its end-of-file (default=4000 records). When the percentage indicated on the Main Menu grows large, you should increase the size of your forms file, as described below.

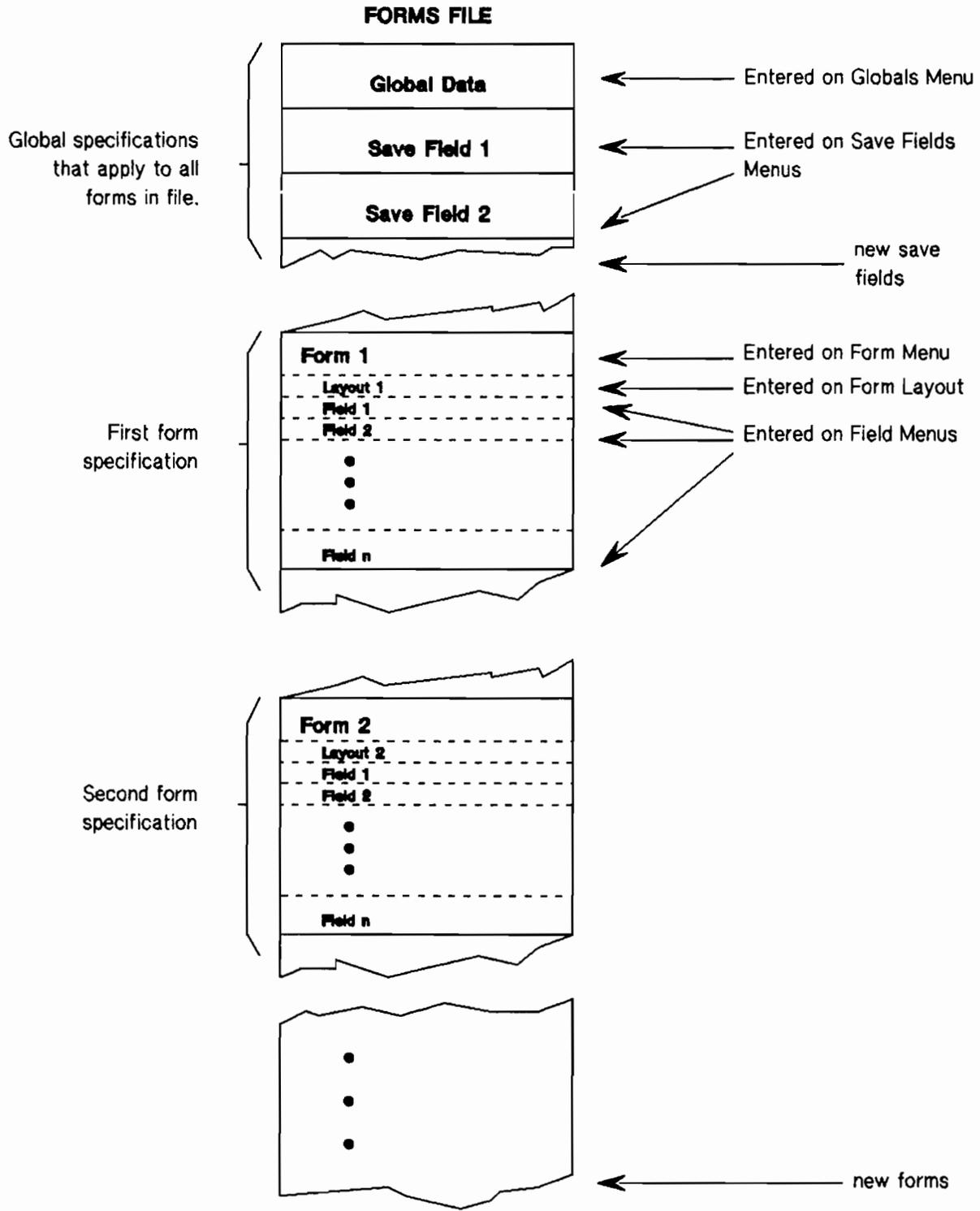


Figure 3-12. Forms File Prototype

Expand Forms File

When you require the forms file to be larger than the default limit of 4000 records, you may override the FORMSPEC default size to create a new forms file using the MPE :FILE command. The :FILE command must be issued prior to the :RUN command to initiate FORMSPEC. Another method is to use the MPE :BUILD command to create a larger file and then use FCOPY to copy an existing forms file into the larger file.

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

```
:FILE ORDFORM;DISC=6000
```

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 6000 records, into which you may add or copy forms using the FORMSPEC Main Menu options. Note that the FORMSPEC copy Main Menu option will not copy save fields from other forms files or maintain form family relationships across forms files. You may re-establish form family relationships using the FORMSPEC relate Main Menu option. Refer to the "Main Menu" description for more information.

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

```
:BUILD ORDFORM2;DISC=6000;REC=128,1,F,ASCII;CODE=VFORM
```

desired disc space

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

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

EOF FOUND IN FROMFILE AFTER RECORD 1015 ← *FCOPY response*

1016 RECORDS PROCESSED *** 0 ERRORS

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 global attribute of the forms file is modified, only those forms which were changed are actually recompiled. The global attributes include any save fields or the fields on the Globals Menu or the Terminal/Language Selection Menu.

Note that the source version is the sequence of specification records entered on the menus. The compiled version is the sequence of forms displayed to the user.

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

largest form in the file. Also, it contains only the information that VPLUS/V needs at run-time, so it has fewer records. Because the record size and the number of records are 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.

The forms file can also be compiled using FORMSPEC in batch mode. (See Section 7 for more information.)

Listing Forms

Figure 3-13 illustrates the listing for a forms file. Note that the listing includes the current status of the forms file, including when it was last modified and compiled, and the number of two-byte words of stack space needed by VPLUS/V when accessing this forms file at execution time.

Forms can also be printed using FORMSPEC in batch mode. (See "Using FORMSPEC in Batch Mode", Section 7 for more information.) Consult Appendix E, for information regarding the file equations which VPLUS/V uses when listing forms files.

```

*****
* FORMSPEC Version v.00.00ff *
* WED, APR 16, 1986, 3:34 PM *
* ORDFORM1.PUB.ACCTG *
*****
Forms File Status
Modified: WED, APR 16, 1986, 3:25 PM
Compiled: WED, APR 16, 1986, 3:25 PM
Requires 5605 bytes of data space in addition to the application
defined Comarea. (Subtract 1600 bytes if only using fast forms files.)
(Local forms storage requires 32 bytes per form storage directory entry
plus 6 bytes of overhead in addition to the above data space requirement.)

Head Form: ORDER_HEAD
Default Display Enhancement: HI
Error Enhancement: IU
Window Display Line: 1
Window Enhancement: HI

THERE ARE 4 SAVE FIELDS IN THIS FORMS FILE:
Save Field: SORDNUM Length: 6 Data Type: DIG
Init Value: 100001

Save Field: SNAME Length: 42 Data Type: CHAR
Init Value:

Save Field: STOTNET Length: 10 Data Type: NUM2
Init Value: 0

Save Field: LINE_COUNT Length: 2 Data Type: DIG
Init Value: 0

There are 3 forms in this forms file:

Form Name Field Counts Largest Number of
Total/Display Field Lines in
Only Number Screen Next Form
ORDER HEAD 8 / 2 8 16 ORDER LINE
ORDER LINE 5 / 1 13 1 ORDER TOTALS
ORDER TOTALS 11 / 7 11 19 $HEAD
    
```

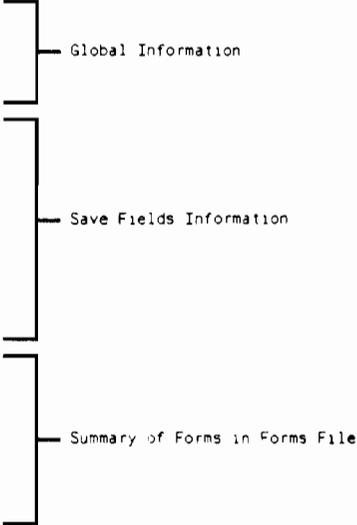


Figure 3-13. Forms File Listing

FORM FAMILIES

A family of forms is a collection of forms that share a common form layout but may have different field attributes, processing specifications, or form sequencing options. Use of form families may improve VPLUS/V performance in some transaction processing applications; since only the internal form characteristics are different among members of a family, you do not have to wait for VPLUS/V to repaint the screen when you change to another form in the same family. Figure 3-14 illustrates some form family concepts.

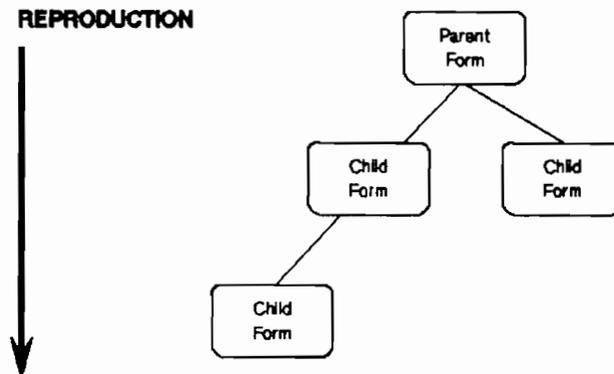


Figure 3-14. Form Family Relationships

Parent Form	Also called father or root form. A parent form is a standard VPLUS/V form created in FORMSPEC either by designing it or by copying it from another form. The form layout of the parent form determines the layout of all child forms that are reproduced from it. Changing the layout of the parent form causes the layouts of the child forms to change.
Child Form	Reproduced from the parent form, the form layout of the child form may not be modified. You may change form sequencing options, field attributes and processing specifications. As Figure 3-14 indicates, you may reproduce a child form from another child form, although VPLUS/V still considers the original root form to be the parent of both forms.
Reproduce	Generate a related form by entering the name of an existing form in the "Reproduced from" field of the Form Menu.

Creating Child Forms

You create child forms by following these steps:

1. Select **(A)** for "Add a form" at the Main Menu. Press **(ENTER)**.
2. When the Form Menu comes onto the screen, specify the name of the child form you wish to create, and, in the "Reproduced from" field, specify the name of the form from which you wish to generate the child form. You may change any of the other Form Menu fields. Press **(ENTER)**.

3. The layout for the form you named in step 2 is displayed. Then press **NEXT**. Since the form layout must be the same for parent and children, pressing **(ENTER)** causes the error message "Cannot change screen, the form is a family member." to appear in the window.
4. You may change any of the attributes or processing specifications on the Field Menus (except length and field number) to adjust the internal characteristics of the child form to your needs. A child form continues to have all the characteristics of the parent if you do not modify the child's Field Menus.

To modify the form layout of a form family, you need to change only the layout of the parent form, and recompile the forms file. This changes the layouts of the child forms automatically. Deleting the parent form deletes the entire form family.

Careful naming of forms and the use of the optional comment field of the Form Menu makes 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 applications, 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. Refer to the SET command in Section 4.

Although family relationships are not maintained after a copy operation, they can be re-established using the **(R)** option for "Relate child form to parent form" at the Main Menu. In order to use this method of creating family relationships, certain criteria must be met. The forms must already exist and their form layouts must be identical. The forms must have identical field number sequences. The forms must be distinct forms; that is, one form cannot be both the parent form and the child form. Furthermore, the child form cannot already be a child form or a parent form. Thus, it is also possible to relate forms which were never previously related.

Users should note that if a new form is to be reproduced from a form which is already a child, then its parent will be designated as the actual parent. For example, suppose that an existing forms file contains three forms named PARENT, CHILD and NEWCHILD, where PARENT and CHILD already maintain a relationship. At the Main Menu, it is indicated that NEWCHILD should become a child form of CHILD. If the forms meet the above criteria, FORMSPEC actually creates the relationship with PARENT as the parent form and NEWCHILD as its child. The form named CHILD continues to be a child form of PARENT.

Forms can also be related using the RELATE command from FORMSPEC in batch mode. (For further information see "Using FORMSPEC in Batch Mode", Section 7.)

Form Family Example

Suppose you have a VPLUS/V application which, after accepting a customer's identification number, retrieves items of pertinent customer information from a data base and displays 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 user views or alters the customer data. If you don't use the form family feature, the user who is alternating between entering customer ID's and inspecting or modifying data constantly has to wait for the form to be repainted each time the user needs information on another customer.

The solution to this problem of long user waiting time is to create two forms with the same form layout but different field attributes.

Suppose the root or parent form of this example is named CUSTPAR, and that its form layout is as shown in Figure 3-15 below. All of the fields in this form are display only, except for the Customer ID Number field, which is required.

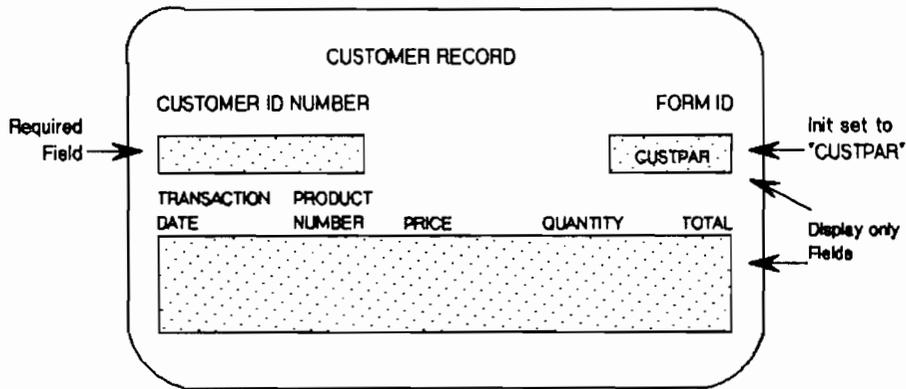


Figure 3-15. Parent Form

Create a child form, CUSTCHILD, by specifying CUSTPAR in the "Reproduced from" field of the Form Menu. Press **NEXT** until the Field Menu for CUSTID appears and make it a display only field. Make all other data fields in the form optional. Compile the forms file.

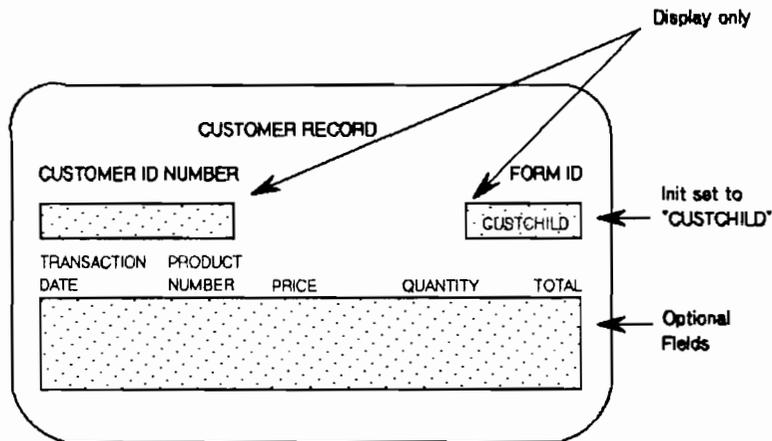


Figure 3-16. Child Form

Now when the user types the customer ID and presses **ENTER**, the application retrieves the related information from the data base and displays it in the form immediately. Once the form is on the screen, it appears to remain there as long as the user uses the application, rather than being repainted each time for every customer about whom the user calls up data.

USING TERMINAL FEATURES

VPLUS/V takes advantage of special terminal features using processing specifications (refer to Section 4) or VPLUS/V intrinsics (refer to Section 6), with selections on the FORMSPEC menus. Two of these features, described below, are security and color. Refer to Appendix G for a list of terminals that support these features.

The Security Enhancement

To use the security enhancement, you should:

- choose the S type display enhancement on the Field Menu,
- select the appropriate terminal family on the Terminal/Language Selection Menu,
- use visible brackets only.

Refer to Appendix G for a list of terminals that support this enhancement.

Color Enhancement

VPLUS/V offers a simple way to provide color field, error and window enhancements for use with terminals that have the color feature. The addition of color to an application requires no programming changes; all color definitions are contained in the forms file.

A well-designed screen layout employing color improves the attractiveness and effectiveness of data presentation. The added dimension may simplify the expression of complex ideas and allow the representation of elusive concepts.

Terminal Color Pairs

There is a set of 8 default color pairs (see Table 3-7). VPLUS/V uses the terminal color pairs to implement color enhancements, with the terminal color pair 0 mapping to VPLUS/V color pair 8 and terminal color pairs 1-7 mapping to VPLUS/V color pairs 1-7. VPLUS/V assumes that an application has set its own values if the default values are not acceptable. It does not reinitialize the terminal's color pairs when the terminal is opened.

Color Field Enhancements

The Field Menu in FORMSPEC allows you to define color field enhancements by entering the digits 1-8 in the enhancement (ENH) field. Each field is enhanced at run-time in the color pairs initialized programmatically or by a terminal hard reset/power on.

VPLUS/V also allows the global definition of default field enhancements. The digits 1-8 are used at the Globals Menu to define a default color pair for all fields in a forms file.

Table 3-7. Default Color Pairs

VPLUS/V Color Pair	Foreground	Background
8	white	black
1	red	black
2	green	black
3	yellow	black
4	blue	black
5	magenta	black
6	cyan	black
7	black	yellow

The ENH fields in the Field and Globals Menu are four characters long, permitting a up to four different enhancements from the set of H, I, B, U, S or NONE. Thus, the combinations of color and display enhancements are limited. For example, a field defined with a combination of half bright, inverse video, underline and blinking (HIBU) cannot accommodate an extra character for color definition (HIBU2). If a color pair is specified with no other enhancements, NONE is assumed.

Color Error Enhancements

VPLUS/V supports global error enhancements by allowing the forms designer to define the desired enhancement on the Globals Menu at design time. This enhancement applies to all fields that appear in error at run time. Like the field enhancement, the error enhancement can be defined as up to four characters. One may be a digit from 1 to 8, indicating a color pair.

Color Window Enhancements

The window enhancement field on the Globals Menu in FORMSPEC allows you to define a four-character combination for enhancing the window. Like the field and error enhancements, a color pair, indicated by a digit, can be included.

Color Locks

FORMSPEC allows the forms designer to add color escape sequences to enhance the text of a form when it is being created. If the form is displayed on a HP 262X compatible terminal which does not support color, the color escape sequences are stripped. Therefore, the form displayed in FORMSPEC will no longer contain the color escape sequences when **ENTER** is pressed. To avoid the loss of valuable work, FORMSPEC will disallow color form modification on non-color terminals if an X is entered in the HP 2627A field on the Terminal/Language Selection Menu.

FORMS FILE MENU

The first menu displayed whenever you run FORMSPEC is the Forms File Menu, as shown in Figure 3-17, on which you enter the name of the forms file to be created or modified.

The next menu displayed after you specify your forms file on the Forms File Menu and press **ENTER** is the Main Menu. You may return to the Forms File Menu by pressing **PREV** from the Main Menu.

```

FORMSPEC v.uu.ff Forms File Menu

Forms File Name [          ]

PREV FORM  NEXT FORM  REFRESH  PREV  NEXT  MAIN/RESUME  EXIT
  
```

Figure 3-17. Forms File Menu

Fields

Forms File Name

Enter the name of the forms file in this field. It may consist of up to 36 characters, as shown in Table 3-8.

NOTE

In order to maintain (and browse or list) a forms file, the user must have exclusive write access to the file.

FORMSPEC accepts only forms files that are MPE files with the file code of VFORM. If you have a forms file that is a KSAM file, created by versions of VPLUS/V prior to A.01.01, refer to Appendix H for instructions on converting KSAM files to MPE files.

Table 3-8. Forms File Menu - Element of Filename.

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

NOTE

If your file has a lockword, you *must* enter the lockword along with the file name, as *filename/lockword*. If you do not enter a lockword with the file name and one is required, your terminal hangs. This is because the MPE prompt requesting the lockword is in character mode and cannot be received while your terminal is in block mode. You can recover from this error by doing a hard reset followed by pressing **RETURN** at least four times, then pressing **EXIT**. How you perform a hard reset depends on what type of terminal you are using; consult your terminal manual for instructions.

MAIN MENU

The Main Menu, as shown in Figure 3-18, is the main control menu for all FORMSPEC operations. If the forms file is new, you usually select **(A)** for Add a form. FORMSPEC then displays the menus that allow you to define your forms file, as described in "Ease of Forms Design". If the forms file already exists, you may enter any selection depending on what you want to do, such as add new forms to the file, select a particular menu in order to change it, delete a form, and so forth. Most of the options available on the Main Menu are also available using FORMSPEC in batch mode. (See Section 7 for more information.)

After modifying the forms file, you must compile it before it can be executed with the modifications. Only those forms of the existing file which have been changed are actually recompiled, unless a global attribute of the forms file, such as a save field, has been modified.

```

FORMSPEC v.uu.ff Main Menu                                FORMS FILE: filename

[ ] Enter Selection

Forms file is 1% full

A--Add a form
S--Add a Save field
T--Terminal/Language 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 [ ]

R--Relate child form [ ] to parent form [ ]

B--Go to ARB Menu.....[ ]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM  FORM      FORM  FORM  RESUME  FORM

```

Figure 3-18. Main Menu

Fields

A--Add a Form

To add a form, simply type **(A)** in the selection box and press **(ENTER)**. In response, FORMSPEC displays a Form Menu, as shown in Figure 3-19, so you can define the form. For details of form definition using this menu, see the description below under "Form Menu".

S--Add a Save field Specify **S** in the selection box and press **ENTER**. A blank Save Field Menu is displayed. This menu (see Figure 3-25) allows you to specify a save field description. To modify a save field, use **PREV** or **NEXT** to locate the particular Save Field Menu specification. For more information, refer to the "Save Fields Menu" description.

T--Terminal/Language Selection Menu Specify **T** in the selection box and press **ENTER**. The Terminal/Language Selection Menu is displayed. This menu (see Figure 3-26) allows you to specify the terminal or set of terminals as well as the native language (if utilizing Native Language Support) with which the forms file will be used. Refer to "Terminal/Language Selection Menu" below and Section 8 for detailed information.

G--Go to Globals Menu Type **G** in the selection box, but do not specify a form or field name. Press **ENTER**. The Globals Menu (see Figure 3-23) is displayed so you can make changes to the global characteristics of the forms file. It provides access to the Data Type Conversions Menu. (Refer to "Globals Menu" below for detailed information.)

G--OR Go to form To display the Form Menu for a specific form, type **G** in the selection box, specify the form name, and press **ENTER**. (You can also locate the Forms Menu with **PREV FORM** or **NEXT FORM**.) 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.)

G--OR Go to field The Field Menu for a specific field can be located by typing **G** in the selection box then specifying both the form and field name, and pressing **ENTER**.

You can display or change field descriptions either on the Field Menu or directly during the form layout. The field attributes (except for length and number) and processing specifications can be entered and changed directly on the Field Menu. New fields are added, existing fields deleted, and the number and/or length of fields are changed by changing the form layout. (For details, refer to the "Ease of Forms Design" discussion earlier.)

L--List Forms File OR List form Type **L** in the selection box, specify the form name and press **ENTER** to print a description of any form in your forms file. If you want to list all the forms in the file with a description of the file in general, simply type **L** but do not specify a form name. Press **ENTER**. Refer to the example listing in Figure 3-13. The listings are printed on the standard list device (;DEV=LP, usually, the line printer) if there is no overriding MPE :FILE command. Consult Appendix E for information on the file equations and the JCW that VPLUS/V uses when listing forms files.

D--Delete Save Field To delete a save field, specify **D** in the Main Menu selection box, specify the save field name and press **ENTER**. The field will be deleted. **You should be sure there are no references to the deleted field in the processing specifications of any other field** (refer to Section 4).

D--Delete Form You can delete an entire form from the forms file by typing **D** in the selection box on the Main Menu, specifying the form name, and pressing **ENTER**. If the form has an ARB associated with it, the ARB is deleted at the same time.

**C--Copy new form
name from form
from Forms file (opt)**

To copy a form, type **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. Press **ENTER**. If the form has an ARB associated with it, the ARB will also be copied.

X--Compile Forms File

You can compile the current forms file by typing **X** in the selection box on the Main Menu and pressing **ENTER**.

Once the forms file has been compiled, only new and modified information is recompiled, unless a global attribute such as a save field, an item on the Terminal/Language Selection Menu, or an item on the Globals Menu has been modified. The newly 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 compiled version is lost.

**X--Compile Optional:
Fast Forms File**

To compile to a fast forms file, type **X** in the Main Menu selection box exactly as you would for any compilation (or recompilation), but also supply the fast forms file name and press **ENTER**.

**R--Relate child form
to parent form**

Form family relationships can be created by typing an **R** in the Main Menu selection box, and the names of the two forms which are to be "related" in the Relate fields and pressing **ENTER**. The forms must already exist and their form layouts must be identical. The forms must be distinct forms; that is, one form cannot be both the parent form and the child form. Furthermore, the child form cannot already be a child form or a parent form. Thus, it is also possible to relate forms which were never previously related. Family forms and their characteristics are fully discussed under "Form Families" earlier in this section.

B--Go to ARB Menu

To create, modify or delete an application-ready buffer (ARB), type in the Main Menu selection box, and the name of the associated form in the ARB Menu field, and press **ENTER**. The form must already exist in the forms file and the data type conversion record must have been set up using the Globals menu. Construction of an ARB is discussed in detail later in this section.

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 displays a blank screen on which you design the layout of the form. This form layout phase establishes the field names and their characteristics. Based on the form layout, FORMSPEC then displays a Field Menu for each field established on the form.

The Form Menu is displayed once for each form you define. When you have no more forms to define, press **MAIN/RESUME** to display the Main Menu in order to compile the forms file. The Form Menu is illustrated in Figure 3-19.

FORMSPEC stores forms in alphabetic order. When new forms are added to the forms file, they are inserted in the USASCII collating sequence. The Globals Menu (see description later in this section) allows you to specify which form is to be executed first. This form can be referenced by the system-defined value \$HEAD. If you do not specify the head form name, \$HEAD defaults to the first form in the file. Adding a new form may change the value of this default.

```

FORMSPEC v.uu.ff Form Menu                                FORMS FILE: filename

Form Name [ ]

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

Next Form [C]                                           Name [$HEAD]
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 [ ]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM

```

Figure 3-19. 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 form layout.

Fields

Form Name

Enter a name to identify the form. The name can be from 1 to 15 characters, either letters of the USASCII alphabet (A-Z), digits (0-9), or the underline (). The first character must be alphabetic. The name can be entered using uppercase or lowercase letters, but all lowercase letters are shifted to uppercase 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 earlier in Table 3-2. Note that the form name can be changed by replacing the name in the Form Name field. If you change a form name, you should make sure that references to this form as "Next Form" in other Form Menus are also changed.

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 changes the repeat option, or, in ENTRY, the user presses **NEXT** to request the next different form.

Default No repeat (N).



NOTE

Both Repeat Option and Next Form can be changed dynamically within the application (refer to Section 6) as well as through processing specifications (refer to Section 4).

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.

Default Clear (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 processing specification described in Section 4.

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 described in Section 4.

Default Clear screen before displaying next form (C).

Function Key Labels To define or modify form level function key labels, enter a Y in this box to obtain the Form Function Key Labels Menu. After function key labels have been defined, FORMSPEC will place an X in the Function Key Label field. The default function key labels can be reset by entering an N. If you specify a Y in this field, the Form Function Key Labels Menu is displayed, as shown in Figure 3-20. The data capture devices do not support function key labeling.

Reproduced from (opt) This optional field is used in the creation of form families, a feature discussed earlier in this section.

Comments You can enter a comment to help 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.

FORM FUNCTION KEY LABELS MENU

This menu is used to specify function key labels that will be displayed along with the form. Labels you specify replace original labels provided by VPLUS/V. 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. The color pairs fields allow you to specify a number, (1 - 8) to indicate a color for each function key; refer to Table 3-7 for the correct color to number mapping.

For example, to change the label for Key 1 to read:

change the field for Key 1 as shown:

ORDER
NUMBER

Function Key 1 [ORDER] [NUMBER]

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. The function key labels can be updated using FORMSPEC in batch mode. (See Section 7 for more information.)

```

FORMSPEC v.uu.ff Form Function Key Labels Menu                                FORM: formname

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 ] [          ]

Color pairs:
f1:[ ] f2:[ ] f3:[ ] f4:[ ]      f5:[ ] f6:[ ] f7:[ ] f8:[ ]

PREV FORM  NEXT FORM  REFRESH  PREV  NEXT  MAIN/RESUME  EXIT

```

Figure 3-20. Form Function Key Labels Menu

FORM LAYOUT

Each form is associated with a picture that is displayed when the form is requested during execution. This picture is called the "form". You design the form on a blank display screen, as shown in Figure 3-21, using all the terminal capabilities associated with block mode.

Once you have entered your headings and other text, and delimited the data fields, the form is basically designed. Each field has default field attributes assigned to it by FORMSPEC, so that unless you want to specify different attributes or processing specifications for the field, form design is complete.

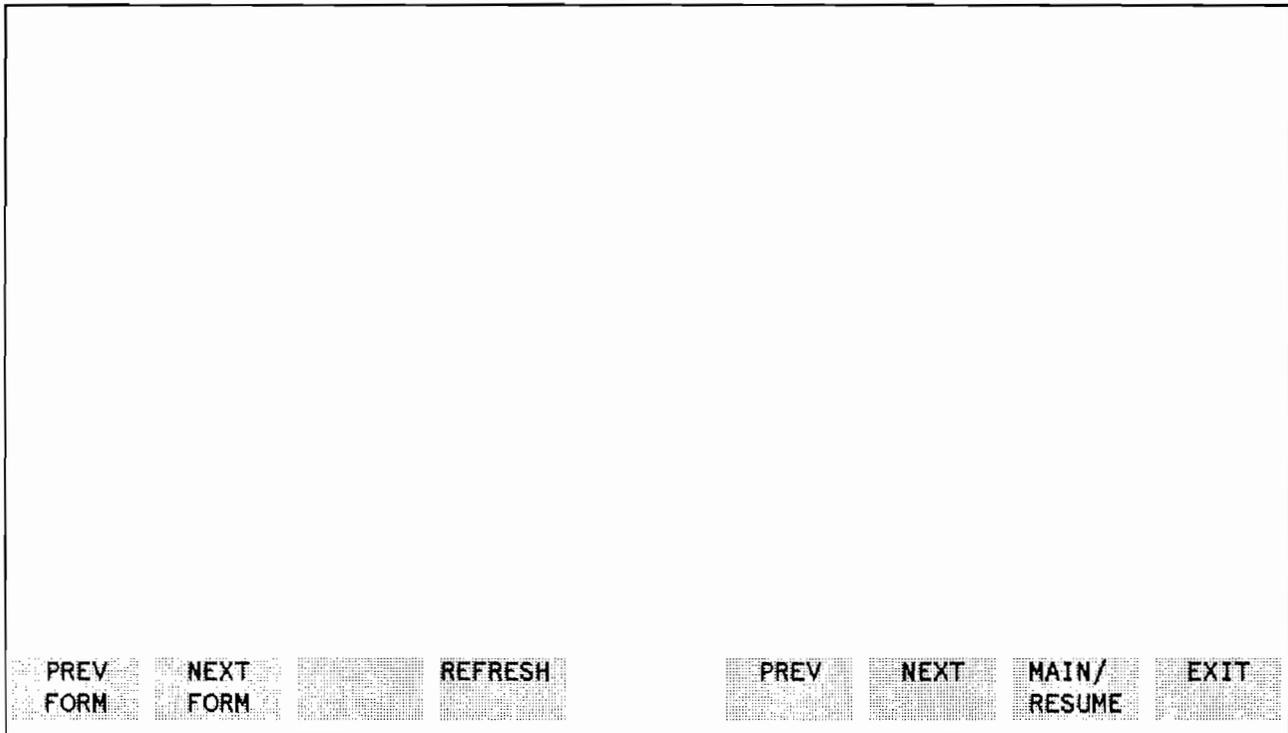


Figure 3-21. Form Layout Menu

Note that you can use the terminal escape sequences or function keys to enhance the text. If you display another set of function keys (such as with `AIDS` or `MODES`) while designing the screen, you must restore the function keys to their FORMSPEC-defined values (such as with `USERKEYS`) before continuing forms design on the next menu.

Note that if you are designing a form to run on the HP 264X terminals, you must not use column 79 if the form is to be part of a form family nor use column 80 if the form is to be either appended or part of a form family. A multiline field must go all the way through column 80 in order to continue on the next line.

FIELD MENU

If you want to assign special field characteristics, you can use a Field Menu (see Figure 3-22) for each field delimited during form design. Each menu displays up to three lines of the form, marks the field to which the menu applies, and displays the default attributes of that field. The Field Menus are presented in the order the fields appear on the form layout, from left to right and top to bottom.

You can change the default field attributes by typing new values and pressing **ENTER**. If you do not want to change the defaults, simply press **NEXT** to display the next Field Menu. If you are designing simple forms, in which all data is accepted and no special edits are performed, you can skip all the Field Menus by pressing **NEXT FORM**. If you change the field length or type, the screen length or type of the associated ARB (if any) is updated at the same time, but the ARB length and type are not changed.

The field attributes, all except length, can also be changed using FORMSPEC in batch mode. See Section 7 for more information. Processing specifications can be specified on the Field Menu. See Section 4 for a description.

FORMSPEC v.uu.ff Field Menu				FORMS FILE: filename			
Num [n]	Len [n]	Name [fieldtag]	Enh [HI]	FType [O]	DType [CHAR]		
Initial Value []							
*** Processing Specifications ***							
PREV FORM	NEXT FORM	FIELD TOGGLE	REFRESH	PREV	NEXT	MAIN/ RESUME	EXIT

Figure 3-22. Field Menu

Fields

Num (Display only) A number between 1 and 256 assigned by FORMSPEC to the field at form layout. The number is permanently assigned to the field and is not changed even if other field attributes or the position of the field on the form layout changes. However, if the field tag is changed, a new field number is assigned to the field. The field numbers of a form can be renumbered in screen order with the FORMSPEC batch mode command RENUMBER, as described in Section 7.

Len (Display only) The length of the field as determined during form layout.

Name Specify any USASCII name of up to 15 characters long; it must start with an alphabetic character and may be followed by any alphanumeric (A-Z, a-z, 0-9) or the underline (_). It cannot be one of the reserved words listed earlier in Table 3-2.

Enh Specify any combination of enhancements using up to four characters:

H	Half Bright
I	Inverse video
U	Underline
B	Blinking
S	Security
1-8	Color (refer to Table 3-7)
NONE	No enhancement

Default HI

FType Specify the field type:

D	Display only -- cannot be modified by user.
R	Required -- cannot be left blank.
O	Optional -- may be left blank; edit checks and field phase processing specifications are skipped if the field is blank.
P	Process -- may be left blank, but edit checks are performed and field phase processing specifications are executed even if the field is blank.

Default O

DType

Specify the data type:

CHAR	Specifies data to be a string of any characters.
NUM[<i>n</i>]	Specifies data to be numeric. The maximum number of decimal places is indicated by the optional digit (<i>n</i>), where <i>n</i> is a value between 0 and 9. An optional leading sign (+ or -) is allowed; as are commas and decimal points if correctly placed.
DIG	Specifies data to be a positive integer. No sign, no comma, no decimal point allowed.
IMP <i>n</i>	Specifies data to be numeric with an assumed decimal point. The assumed number of decimal places is indicated by the required digit (<i>n</i>), where <i>n</i> is a value between 0 and 9. A sign and commas are allowed. A decimal point is allowed but should not be entered.
MDY	Specifies data to be a date in the order month, day, year.
DMY	Specifies data to be a date in the order day, month, year.
YMD	Specifies data to be a date in the order year, month, day.

When Native Language Support is used, the symbols used to indicate thousands and decimals are language-dependent. The names of months and their abbreviations are accepted for each native language at run-time. For more information on Native Language Support, see Section 8.

Default CHAR**Initial Value**

Specify data to be displayed in the field when the form is initially displayed.

GLOBALS MENU

Certain characteristics of the entire forms file are defined as global specifications and specified on the Globals Menu, illustrated in Figure 3-23. The global specifications include the default screen enhancements for fields and errors, the first form to be executed, where the message "window" line appears on the form, and provides for user-defined labels for the function keys. These specifications apply to the entire forms file rather than to individual forms or to fields within the forms. If you are planning to use the ARB feature, you must also supply global defaults for screen/ARB data conversions.

FORMSPEC supplies default values for these characteristics and, unless you want to change the defaults, you need never be concerned with global specifications. 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 data capture devices on the Terminal/Language Selection Menu, the following message will appear at the bottom of the Globals Menu:

Press NEXT to select HP 3075/6 device specifications

(The Data Capture Device Specifications Menu is described later in this section).

```

FORMSPEC v.uu.ff Globals Menu                FORMS FILE: filename

      Head Form Name  [  ]

Default Display Enhancement  [HI  ]

      Error Enhancement  [IU  ]

      Window Display Line  [24]

      Window Enhancement  [HI  ]

Define Function Key Labels  [ ] ("Y")

Define Data Conversions     [ ] ("Y")

PREV FORM    NEXT FORM    REFRESH    PREV    NEXT    MAIN/RESUME    EXIT

```

Figure 3-23. Globals Menu

When you press **ENTER**, FORMSPEC checks and records the global values (default or specified) and then displays the next menu. The next menu is the Save Field Menu. To get to the first Form Menu, press **NEXT FORM**.

Fields

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 in the forms file will be displayed. Adding forms to the forms file may change the first form. When the forms file is compiled, this name must identify an existing form. If you enter the name using lowercase letters, FORMSPEC shifts the letters to uppercase and then looks for a matching form name.

Default First form in USASCII collating sequence in the forms file. (\$HEAD)

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 them on the Field Menu, FORMSPEC assigns default field enhancements. If you do not want to use these defaults, you can specify your own default enhancements. Enter one or more of the display enhancement codes (B, H, I, S, U, and 1 - 8) or enter NONE if you do not want the fields to be enhanced at all. 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. If you want to remove all display enhancements, enter NONE. The data capture devices do not support display enhancements.

Default Half bright, Inverse video (HI).

Error Enhancement

When a user entering data on a form makes an error, the field with the error is highlighted on the form by special display enhancements. You can change the default error enhancement by entering one or more of the display enhancement codes (B, H, I, S, U, and 1 - 8) 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.

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

Window Display Line

You can specify the line of the terminal 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. In this case, no error or status messages are displayed on the form 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. If function key labeling on HP 264X terminals is in effect and a window line of 23 or 24 is chosen, line 22 is actually used.

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, S, and 1 - 8), or you can specify no enhancements by entering NONE.

Default Inverse video, Half bright (HI).

Define Function Key Labels

To define function key labels, enter Y to obtain the Global Function Key Labels Menu. If you wish to retain the default global labels, leave this box blank. When function key labels have been defined, FORMSPEC will place an X in the Define Function Key Labels field. Once function key labels have been defined, the default labels can be reset by entering an N in this field. If you specify a Y in this field, the Global Function Key Labels Menu is displayed, as shown in Figure 3-24. The data capture devices do not support function key labeling.

Define Data Conversions

To use the ARB feature, you must specify default conversions for screen data to application data and vice versa. Type Y in this field to reach the Data Type Conversions Menu, shown in Figure 3-23a. The default data type conversions must be set up before you can create an ARB for a form.

GLOBAL FUNCTION KEY LABELS MENU

This menu is used to specify new function key labels which will appear when an application is executed. Labels you specify replace original labels provided by VPLUS/V.

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. The color pairs fields allow you to specify a number, (1 - 8) to indicate a color for each function key; refer to Table 3-7 for the correct color to number mapping.

For example, to change the label for Key 1 to read:

```
ORDER
NUMBER
```

change the field for Key 1 as shown:

```
Function Key 1 [ ORDER ] [ NUMBER ]
```

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 v.uu.ff Global Function Key Labels Menu          FORM FILE: filename

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 ] [          ]

Color pairs:
f1:[ ] f2:[ ] f3:[ ] f4:[ ]    f5:[ ] f6:[ ] f7:[ ] f8:[ ]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM  [          ]  [          ]  [          ]  RESUME  [          ]
```

Figure 3-24. Global Function Key Labels Menu

SAVE FIELD MENU

A save field is a global field whose value can be used anywhere in the forms file where field references are allowed. Save fields are not part of a particular form like other fields, but are "global" to all forms. Like other fields, they always contain data in external, character form. The save fields can be used by most processing statements, but are particularly useful for passing data between fields on different forms. Refer to the SET command in Section 4. A maximum of 20 save fields can be defined for any forms file.

Save fields are specified on the Save Field Menu, which is displayed 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 "Forms File" earlier in this section for the order of specifications in the forms file.) The Save Field Menu is illustrated in Figure 3-25.

FORMSPEC v.uu.ff Save Field Menu FORMS FILE: filename

Save Field Name [] Length [] Data Type []

Initial Value []

PREV FORM NEXT FORM REFRESH PREV NEXT MAIN/RESUME EXIT

Figure 3-25. 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 letter 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/RESUME** to request the Main Menu and then type S in the selection box and press **ENTER**. A blank Save Field Menu is then displayed. You can also display a blank Save Field Menu by pressing **PREV** 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 save field to be deleted in the field name box.

Fields

Save Field Name A standard field identifier composed of up to 15 USASCII characters, beginning with an upper or lowercase letter. The remaining characters can be upper or lowercase 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 earlier in Table 3-2.)

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.

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 "Ease of Forms Design" for a complete discussion of these data types.)

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 user; 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, 86	type is MDY
John	type is CHAR

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

USING FORMSPEC TO CREATE AN ARB

Once you have created a form and saved it in a forms file, you can create an associated application-ready buffer (ARB) for that form. There are two steps to the process; first, you must set up a data type conversion (DTC) record, then you can generate the ARB. The DTC record need only be set up once for the entire forms file.

You may want to transform data between the screen and application for several reasons. First, the data the application will store may differ from what appears on the screen: menu-selection and next-screen fields, for example, would normally be excluded from the ARB. Conversely, you may want to store data from a source other than the screen, such as key fields for an IMAGE dataset, along with the screen data. This can be done by using "filler" fields, fields that exist on the ARB but not on the corresponding form (see **RES** below).

Second, the order in which the application stores data may differ from the order in which it is entered on the screen. An arrangement of fields that makes logical sense to the user may not be suitable for a database, for example. Fields can be rearranged on the screen without affecting their order on the ARB, and vice versa.

Third, data type and length may differ from screen to ARB, and the designer uses the ARB screens, in conjunction with the Data Type Conversion Menu, to specify conversions (see discussion under "ARB Data Types").

Figure 3-25a shows the sequence of menus used to create, modify and delete an ARB and the fields on it.

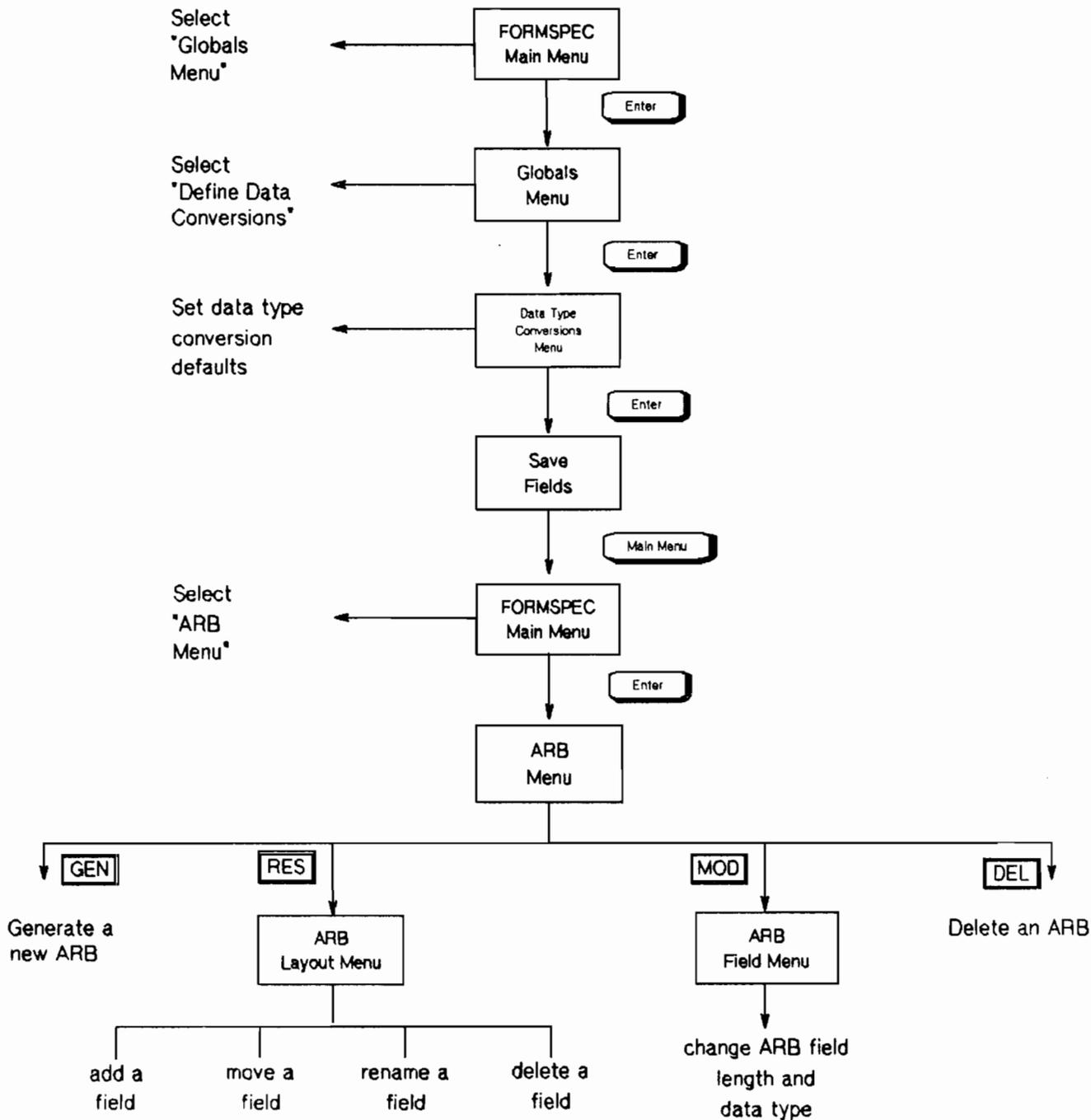


Figure 3-25a. Menu Sequence for ARB Feature

Setting Up the Data Type Conversion Record

The data type conversion record allows the forms designer to define the default values for screen and ARB data types. You **must** define these values, even if you only press **ENTER** to accept FORMSPEC's preset defaults, shown in Figure 3-25c. You will probably want to replace some, if not all, of these defaults with your own.

Set up the data type conversion record by following these steps:

1. Make sure you are at the FORMSPEC Main Menu and select **G** for "Go to Globals Menu". When you press **ENTER**, the Globals Menu is displayed.

```

FORMSPEC v.uu.ff Globals Menu                                FORMS FILE: filename

      Head Form Name [FORM 1 ]

Default Display Enhancement [HI ]

      Error Enhancement [IU ]

      Window Display Line [1 ]

      Window Enhancement [NONE]

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

Define Data Conversions   [Y ] ("Y")

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM  FORM      FORM  FORM  RESUME
    
```

Figure 3-25b. Globals Menu

2. Enter **Y** for "Define Data Conversions" in the last box. The Data Type Conversions menu is displayed. This screen allows you to specify default conversions to and from every available screen and application data type.

```

FORMSPEC v.uu.ff Data Type Conversions Menu          FORMS FILE: FILENAME

Default Data Type Conversions:
-----

                From Screen Type to Application Type

    CHAR      date      DIG      NUMn      IMPn
  [CHAR ]   [CHAR ]   [CHAR ]   [CHAR ]   [CHAR ]

-----

                From Application Type to Screen Type

    CHAR      INT/      REAL/      SPACKn/      ZONEn      YMMDD
  [CHAR ]   [CHAR ]   [CHAR ]   [CHAR ]   [CHAR ]   [CHAR ]

-----

  PREV      NEXT      REFRESH      PREV      NEXT      MAIN/      EXIT
  FORM      FORM      FORM          FORM      FORM      RESUME      FORM

```

Figure 3-25c. Data Type Conversions Menu

The *n* in the numeric types stands for a number of decimal places. When specifying a conversion to one of these types, you must replace *n* with a digit or *a*. When you set the number of decimals to *a*, you are instructing FORMSPEC to determine the decimals algorithmically on the basis of the source type. Note that you can only select the *a* option on this menu, and not on the field menus. For recommendations on data type conversions, see "Data Types" earlier in this section.

3. When you have set your defaults, press **ENTER** and then press **MAIN/RESUME** at the SAVE FIELD Menu to return to the Main Menu.

Generating the ARB

Follow these steps to generate an ARB for a form.

1. At the Main Menu, enter **B** for "Go to ARB Menu" in the Main Menu Selection box.
2. In the "Go to ARB Menu" field, enter the name of the form for which you wish to generate an ARB and press **ENTER**. The ARB Menu is displayed.

```
FORMSPEC v.uu.ff ARB Menu
Form Name [FORM1] NO field(s) in ARB
[GEN] Enter selection
GEN--Generate ARB
RES--Restructure ARB (add, move and delete fields)
MOD--Modify ARB field [ ]
DEL--Delete entire ARB

REFRESH MAIN/RESUME EXIT
```

Figure 3-25d. ARB Menu

The form name is displayed in the first field. This is a display-only field: you cannot select a new form from this menu. If you want to create an ARB for a different form, you must go back to the Main Menu.

The next field shows the number of fields in the ARB: in this case there are none, since there is no ARB yet.

Types of Field

An ARB may contain two kinds of field. They are live fields and filler fields.

- Live fields exist on both the screen and the ARB.
- Filler fields exist on the ARB only; for example, alignment and accumulator fields.

The designer can select one of four options on the ARB Menu.

Options

GEN (Generate) Create an ARB from scratch, containing a field for each field on the form. The sequence of the fields accords with their sequence on the form, and can be altered using the Restructure ARB Menu (Figure 3-2c). The type and length

of the fields are determined by comparing the screen attributes with the data conversion record. They can be changed using the ARB Field Menu.

This option is only valid if no ARB exists for the form.

RES (Restructure)

Display the current ARB, showing the sequence of fields. Allows the designer to add, delete, move and rename fields using the Restructure ARB Menu.

MOD (Modify)

Modify the length and data type of the designated field in the ARB, using the Arb Field Menu.

DEL (Delete)

Delete the entire ARB. If you want to make a lot of changes to the fields on a form, it is usually better to delete the existing ARB and generate a new one once the form has been altered. If you copy or delete a form, the corresponding ARB will be copied or deleted as well.

Table 3-9 shows how an alteration made to the form will affect the ARB, if at all.

Table 3-9. Form/ARB Relationships

Form	ARB
Adding a form...	Does not add an associated ARB.
Copying a form...	Copies the associated ARB.
Renaming a form...	Renames the associated ARB.
Deleting a form...	Deletes the associated ARB.
Changing a form: - adding a field... - changing a field... - deleting a field...	Does not add it to the ARB once the ARB has been created, unless you create a new ARB. Changes the screen length and screen data type on the ARB, but does not affect the ARB field length and data type. Changes the corresponding ARB field to a "filler" field.
No effect on form...	Add, change, delete ARB or the fields in it.
Form can exist without an ARB.	ARB cannot exist without a form.

The Effect of Renaming a Field on a Form:

There are three possibilities, which are described below.

- **The old name is in the ARB but the new name is not:** The ARB field is renamed to the new name, retaining its "live" field type, and the screen length and screen type are altered if necessary.
- **The old name is not in the ARB, but the new name is, as a "filler" field:** The ARB field changes from "filler" to "live" field type, and the screen length and screen data type are updated if necessary.
- **Both old and new names exist in the ARB:** The *old* name changes from "live" to "filler" type, and the screen length and screen data type change to zero and blanks respectively. The *new* name changes from "filler" to "live" field type, and the screen length and screen data type are updated.

NOTE

FORMSPEC allows the designer to delete a field from a form after a corresponding field has been added to the ARB. The field need not be deleted from the ARB as well; at compile-time, the ARB field is converted to a "filler" field, with screen length set to zero and screen type blank. The designer can, however, choose to delete or rename the ARB field.

It is possible, though usually not advisable, to create an ARB without using the **GENERATE** feature. To do this, type **RES** and use the Restructure ARB Menu to add fields one at a time.

If your default conversion for any field(s) could cause run-time errors, you will get a message to that effect (see Appendix B, *VPLUS/V ERROR MESSAGES*). Go to the ARB Field Menu to check that the conversion does make sense in this case, or to change the target datatype if it doesn't.

3. Enter **GEN** in the Selection box and press **(ENTER)** to generate an ARB for this form. The "NO field(s) in ARB" changes to show a number equivalent to the number of fields on the form.
4. You can now change the position of a field in the ARB, add or delete fields, or modify a field by changing its length and/or data type.
5. To **move, rename, add or delete** fields, type **RES** in the Selection box and press **(ENTER)**. The Restructure ARB menu is displayed.

```

FORMSPEC v.uu.ff Restructure ARB Menu

Form Name [FORM 1 ] 48 field(s) in ARB

Enter selection: [MOVE ] (Add, Move, Rename or DELETE)
Field or Range: [FIELDA/X1 ]
(Before, After): [AFTER ] Destination: [ADDRESS2 ]

001 FIELDA .002 FIELDDD 003 FIELDDC 004+XNOTFIELD
005 FIELDX 006 NAME 005+X1 006 ADDRESS
007+X2 008 CITY 009+HUNDREDS 010 THOUSANDS
011 ADDRESS2

REFRESH PREV NEXT MAIN/ ARB MENU
PAGE PAGE RESUME

```

Figure 3-25e. Restructre ARB Menu

This menu displays all the fields in the ARB by name and in sequence. The form name is displayed in the first field (display only). The lower part of the screen shows the ARB fields. Use **PREV PAGE** and **NEXT PAGE** to view them all if necessary.

- Use the fields at the top of the screen to change the order by moving fields or ranges. Press **ENTER** to execute all changes.

Options

Command Must state MOVE, ADD, RENAME or DELETE. DELETE must be spelled out; the other commands may be abbreviated.

Field or Range Name(s) or numbers(s) of the field(s). A slash (/) indicates a range: valid for MOVE or DELETE only. Any field may be deleted.

You can ADD only a single field at a time; it may not already exist on the ARB and it must conform to FORMSPEC naming rules. You may add a field that does not exist on the associated form. This "filler" field will have an ARB default length of 1 byte, and ARB type of CHAR; screen length is zero and screen type is blank. Fillers are marked with a "+" on the Restructure ARB Menu.

To **RENAME** a field, enter the name of an existing field, and enter the new name of the field in **DESTINATION**. "Live" fields exist on both form and **ARB**; "filler" fields exist on the **ARB** only. The following defaults apply to renamed **ARB** fields:

- **Live name changed to live name** (new **ARB** field has corresponding field on the form): new **ARB** field length and type derived algorithmically from the length and type of the corresponding field on the form.
 - **Filler name changed to live name** (new **ARB** field has corresponding field on the form): new **ARB** field length and type derived algorithmically from the length and type of the corresponding field on the form.
 - **Live name changed to filler name** (new **ARB** field has no corresponding field on the form): new field retains length and type of old **ARB**, and screen length becomes zero and type blank.
 - **Filler name changed to filler name** (new **ARB** field has no corresponding field on the form): new field retains length and type of old **ARB**, and screen length remains zero and type blank.
7. When you have finished modifying the field order on the **ARB**, press **ARB MENU** to return to the **ARB** Menu, or **MAIN/RESUME** to go back to the Main Menu.
8. To change the length or data type of an **ARB** field, type **MOD** on the **ARB** Menu. The **ARB** Field menu is displayed.

FORMSPEC v.uu.ff ARB Field Menu

Form Name [FORM1]

		---Screen---		-----ARB-----
ARB Sequence #	Field	Len	Dtype	Byte Len Dtype
[1]	[A2]	[3]	[CHAR]	[3] [CHAR]

REFRESH PREV FIELD NEXT FIELD MAIN/RESUME ARB MENU

Figure 3-25f. ARB Field Menu

This screen allows you to scroll through all the fields in the ARB and change their length and data type if required.

NOTE

You can change only the **ARB** field type and length. To change the characteristics of the field on the **form**, you must use the FORMSPEC Field Menu (see Figure 3-22). If you use this menu to change the **length** or **type** of a field on the form, the corresponding screen length and screen type of the field in the ARB will be changed accordingly, but the ARB length and ARB type will remain the same.

Using and Defining ARBs

The programmer is responsible for:

- Using the correct data types for each programming language; for example, REAL is invalid for COBOL applications.

- Aligning and/or padding the ARB/source code. If an odd number of bytes is followed by an integer, some languages, including Pascal and FORTRAN, automatically pad the record definition, forcing the integer to be word-aligned. No such padding occurs in COBOL unless the SYNCHRONIZED clause is used (see the *COBOL 11/3000 Reference Manual*).

If you use a language whose compiler pads to ensure word alignment for integers, you must pad your ARB correspondingly. For example, suppose you are coding in Pascal and you declare a record that looks like this:

```
ODD_BYTE_EXAMPLE = RECORD
  THREE_BYTES   : PACKED ARRAY [1..3] OF CHAR;
  GOTCHA        : INTEGER
  .
  .
  .
END;
```

The Pascal compiler will insert an additional byte after THREE_BYTES to align the integer on a word boundary. You must do the same with the ARB record; use the ARB LAYOUT screen to add a filler field after the three-byte field so that the ARB looks like this:

<i>Field Name</i>	<i>ARB Type</i>	<i>ARB Length</i>
DEPT	CHAR	3
FILLER1	CHAR	1
TOTAL PURCHASES	DINT	4

- Ensuring the application specifications match the ARB specifications; for example, if the ARB type is PACK, the COBOL specification should be COMP-3, not COMP.
- Avoiding run-time errors, which may occur when:
 - Converting a CHAR or date source to a numeric destination
 - Converting a numeric or CHAR source to a date destination
 - Invalid length specifications are encountered
 - There are alignment problems; FORMSPEC does not detect these

There is also the possibility that data will be truncated; for example, if DIG ---> INT, but *screenlen* is greater than 5.

The ARB Trace Facility

The ARB trace facility may be enabled by setting the JCW **VPLUSARBTRACE** to 1. This will print trace messages to the stdlist. You may direct VPLUS/V screens to a different device by using a **FILE** command to set the device of the filename used by **VOPENTERM** to a device other than the stdlist device.

This is an example of a trace from a form that has seven fields:

```
:SETJCW VPLUSARBTRACE,1
:FILE VTERM;DEV=99
:RUN ARBPROG
```

```
Field #1   : Buffer offset from 0 = 1
              Length = 2
              Type = INT
              Value = 1234

Field #2   : Buffer offset from 0 = 3
              Length = 2
              Type = INT
              Value = -1234

Field #3   : Buffer offset from 0 = 6
              Length = 2
              Type = INT
              Value = 1234

Field #4   : Buffer offset from 0 = 9
              Length = 4
              Type = DINT
              Value = 12345678

Field #5   : Buffer offset from 0 = 14
              Length = 4
              Type = DINT
              Value = -1234567

Field #6   : Buffer offset from 0 = 19
              Length = 4
              Type = DINT
              Value = 12345678

Field #7   : Buffer offset from 0 = 23
              Length = 8
              Type = CHAR
              Value = ABCDEFGH
```

```
END OF PROGRAM
:
```

The trace provides the following information. It shows:

- The location of each field in the ARB (Buffer offset);
- The length of each field in the ARB (Length);
- The type of transformation that occurs (Type);
- The value that is transformed in the ARB field (Value).

NOTE

This ARB is tied to seven form fields, but there are gaps in the buffer before field #1, and between each field from #2 to #6. These gaps are one-byte filler fields.



TERMINAL/LANGUAGE SELECTION MENU

The Terminal/Language 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 four families (HP 264X, HP 262X, HP 239X (includes HP 150) and HP 307X). The HP 264X family is the terminal default value. When the selection is defaulted, the forms file will run on the HP 264X family, the HP 262X/HP 239X family and the HP 150, but the following features of selected terminals will not be supported: color, local edits, function key labeling on HP 264X terminals, and security display enhancement. Consult Appendix G to see which HP terminals support which features. The family of terminals need not be specified unless the forms file will be making use of these terminal features or will be used with the data capture devices. Refer to the field descriptions below for more information.

The Terminal/Language Selection Menu also allows you to specify the native language with which the forms file will be used. See "Setting the Language ID Numbers", Section 8, for more information.

The options available on the Terminal/Language Selection Menu are also available using FORMSPEC in batch mode. (See Section 7, for more information.)

```

FORMSPEC v.uu.ff Terminal/Language Selection Menu      FORMS FILE: filename

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

      [X] HP264X Family      [ ] HP3075A, HP3076A, HP3081A

      [ ] HP262X, HP239X    [ ] HP2627A, HP2397A
      Families

      [ ] Reserved         [ ] Reserved

-----
Enter Language ID:

FORMSPEC Language      ID      Name
      [ 0 ]      NATIVE-3000

Forms File Language    [ 0 ]      NATIVE-3000

PREV      NEXT      REFRESH      PREV      NEXT      MAIN/      EXIT
FORM      FORM
  
```

Figure 3-26. Terminal/Language Selection Menu

Fields

Select (X) the terminals

The choices are:

HP264X Family Specify an X for HP 264X terminals; specify a Y for HP 264X terminals with function key labeling.

HP262X, HP239X Families Specify an X for HP 262X and HP 239X terminals; includes HP 150 terminals.

HP3075A,HP3076A,HP3081A Specify an X in the HP 3075A, HP 3076A, HP 3081A box in order for the forms file to run on the data capture devices; specify an 8 to run on the data capture devices and take full advantage of the HP 3081A.

HP2627A,HP2397A Specify an X for terminals with the color feature; you must also specify an X in the HP262X, HP239X Families field.

If you want the forms file to run on the HP 264X and HP 262X/HP 239X families, you must specify *both* on the Terminal/Language Selection Menu.

Enter Language ID:

Specify the following:

FORMSPEC Language Only 0 for NATIVE-3000 is supported.

Forms File Language Specify one of the following:
 0 for NATIVE-3000
 the native language ID number for a language-dependent forms file or
 -1 for an international forms file.
 Refer to Section 8 for more information.

Default 0 for NATIVE-3000

DATA CAPTURE DEVICE SPECIFICATIONS MENU

If you select the data capture devices on the Terminal/Language Selection Menu, the following message will appear at the bottom of the Globals Menu:

Press NEXT to select HP 3075/6 device specifications

Press NEXT to display the Data Capture (3075/6) Device Specifications Menu, as shown in Figure 3-27. On this menu you can specify how you want messages longer than 24 (or 32 for the HP 3081A) characters to be presented on singleline display. In addition, you can specify which keyboard 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.

```

FORMSPEC v.uu.ff 3075/6 Device Specifications          FORMS FILE: filename

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]
Clock On/After/NONE [NONE]

Barcode Reader:
Format [UPC] Disable Check Digit [N]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM

```

Figure 3-27. Data Capture Device Specifications Menu

Fields

Split Message Pause (seconds)

You may specify the number of seconds between screen presentations for messages longer than 24 (or 32) characters. Enter a value between 1 and 99 in this box and a NO in the Wait for ENTER box. User and VPLUS/V's messages will be broken so as not to display part of a word. This field does not apply to Mini-CRT display.

Default 3 (seconds)

OR Wait for user to press ENTER

You may also specify that screen presentations for messages longer than 24 (or 32) characters be displayed after **ENTER** is pressed. Enter a YES in this box and a zero (0) in the Split Message Pause box. This field does not apply to Mini-CRT display.

Default NO

Error Light

You may specify which keyboard light is to be lit if an error is detected on the form by the application or by VPLUS/V. 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

The card or badge will have holes or will be mark sensed.

Default HOLES

Corner Cut Required

The card or badge must be presented with the corner in accordance with the diagram on the terminal.

Default YES

Clock On/After /NONE

The 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

If you select the Bar Code Reader as an input device, you can specify how it should be configured. The following options are available:

Format

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

UPC -- Universal Product Code

I25 -- Industrial 2 out of 5

MAT -- INTERMEC 2 out of 5 MATRIX code

I39 -- INTERMEC code 39 alphanumeric bar code*

EAN -- European Article Numbering

ILV -- Interleaved 2 out of 5

Default UPC

*code I39 is a registered trademark of Interface Mechanism, Inc.

**Disable Check
Digit**

When you specify the bar code reader as an input device, VPLUS/V configures the device to require check digits. You can change this by specifying yes (Y) in this field.

Default N

Example of how a typical form layout will look displayed on the data capture devices:

```
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 3-28. Form Layout in FORMSPEC

```
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 3-29. Displayed on 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 3-30. Displayed on 32 Character One 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 3-31. Displayed on Large Character Mini-CRT Screen

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 fields, 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.
- Perform processing specifications in different phases of form execution.

LEVELS OF ADVANCED DESIGN

Advanced forms design can be separated into two levels of processing specifications:

- **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 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 sequencing options 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 any programming language, is affected by the order in which the statements are specified. The order of statement execution can be defined explicitly through phase statements.

Advanced forms design uses processing specifications that are specified 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 four phases: configuration, initialization, field edits, and finish. If used, phases allow the selection of:

- terminal configuration for local edit terminals and data capture devices,
- initialization of fields in the form before data is entered in the fields,
- editing of data in each field after the data is entered, and,
- after all fields are edited, any finish processing of the form.

The application requests execution of each phase with appropriate VPLUS/V intrinsics, except for configuration, which is a passive phase built into the presentation of the form. Refer to "Phases" later in this section for more information.

If you have collected data in a batch file, you may also use the Reformatting Capability (described in Section 5) to reformat the data in the batch file for subsequent use by an application.

ENTERING PROCESSING SPECIFICATIONS

After defining the field attributes on the Field Menu (as described in Section 3), if you want to define additional processing, press **FIELD TOGGLE**. You can then enter the 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 field attributes.
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 (refer to your terminal manual for details.)
3. Positions the cursor at the beginning of the first line on which you can enter the processing specifications.

FORMSPEC v.uu.ff Field Menu			FORM NAME: filename	
QTY	PART NO.	DESCRIPTION	UNIT	TOTAL
			PRICE	PRICE
qty..	partnum..	description.....	uprice.	tprice. ^
Num [9]	Len [7]	Name [TPRICE]	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.		
				
PREV FORM	NEXT FORM	FIELD TOGGLE	REFRESH	PREV NEXT MAIN/ RESUME EXIT

Figure 4-1. Field Menu With Processing Specifications

You can then enter any of the processing statements described in this section. They can be typed on any line in the processing specifications area, and must conform to the specified syntax rules. When you have typed all the specifications for this field, press **ENTER**. When **ENTER** is pressed, the field attributes are checked first. If there are no errors, the 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.

Special Cases

If a processing specification statement line extends to the 80th column, the line is expanded to two lines when you return to the Field Menu. This may in turn result in processing specification truncation due to either terminal memory overflow or FORMSPEC internal buffer overflow.

In nonformatted mode, it is possible to accidentally clear or delete any field attributes with a key such as CLEAR DISPLAY or DELETE LINE. Doing this is only a problem if the values for the field attributes have not yet been recorded by pressing **ENTER**. To recover, press **REFRESH** and then reenter any changed values for the field attributes.

While in the processing specification area of the Field Menu, recovering from accidentally pressing **BREAK** or from system problems requires a special recovery procedure. Refer to Appendix G.

Correcting Existing Specifications

Whenever you return to a Field Menu, the terminal is in formatted mode with the cursor positioned to the first unprotected field in the upper part of the screen. To change the field attributes, **TAB** to the field you want to change and type in the new value. To change the processing specifications in the lower part of the screen, you must press **FIELD TOGGLE** to put the terminal in nonformatted mode. The cursor is then positioned to the beginning of the processing specification area.

If you want to change a field attribute in the upper part of the Field Menu when you are in the lower part entering processing specifications, you must press **FIELD TOGGLE** to return to formatted mode. The cursor is then positioned at the first unprotected field in the upper part of the menu, and you can then change the field attributes.

STATEMENT SYNTAX

The processing specifications consist of statement names followed by parameters. The description of each statement uses the conventions defined at the beginning of the manual. 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 backslash (\). Anything typed between the backslash and the end of the line is ignored. For example:

<pre>GT 12</pre> <p>statement</p>	<pre>\This field must have a value greater than 12.</pre> <p>comment</p>
-----------------------------------	--

Continuing Lines

Statements that are not completed before the first backslash 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 the string of
"UVWXYZ"             \the uppercase letters ABCDEFUVWXYZ.
```

Custom Error Messages

Whenever a field edit statement detects an error at run-time, the application can call the appropriate VPLUS/V intrinsics to have an error message (provided by VPLUS/V) issued and displayed in the window line, such as is done with ENTRY. You may choose to write a custom message to be issued when a field fails a particular edit specification. To do this, you specify the custom message in quotes immediately following the statement to which it applies. Figure 4-2 illustrates a Field Menu in which the processing specifications contain custom error messages. When and if the statement causes a field to fail, the custom message is displayed instead of a VPLUS/V error message. For example:

```
MATCH uddd "Field has wrong format for Product Number"
```

FORMSPEC v.uu.ff Field Menu				FORM: formname			
Address [address]							
City [city]			State [st]				
Zip [zip]		Telephone [phone]					
^							
Num [6]	Len [10]	Name [ZIP]	Enh [HI]	FType [R]	DType [CHAR]		
Initial Value []							
*** Processing Specifications ***							
MATCH ddddd[-dddd] "Enter ZIP code, must be either 5 " & "digits or 9 digits with hyphen"							
PREV FORM		NEXT FORM		FIELD TOGGLE		REFRESH	
PREV		NEXT		MAIN/RESUME		EXIT	

Figure 4-2. Field Menu With Custom Error Messages

Checking Data Entered

The field attributes for the field determine the first checks made on the data in the field. For example, if the field is required (FType=R), it is checked to be sure that the user 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 field is checked to be sure that only the digits 0-9 were entered in the field. Next, the processing specifications are tested in the order they are specified. As an example, consider the specifications in Figure 4-2. After making the checks based on the field attributes, data entered in the field is tested to be sure that at least 5 digits were entered (MATCH ddddd), and optionally the value is checked for a hyphen and 4 more digits ([-dddd]). In all cases, the checks are performed by VPLUS/V when the application calls the appropriate VPLUS/V intrinsics, as described in Section 6.

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 ALPHANUMERIC CONSTANT IMP_DEC IMP_DEC_FILL INTEGER INTEGER_FILL SIGN_DEC SIGN_DEC_FILL UNRESTRICTED UPSHIFT	[JUSTIFY MUST_FILL REQUIRED] ... [DEC_DIGITS n] [DEC_TYPE-EUR DEC_TYPE_US] [TRANSMIT_ONLY]
------------	--	---

LIGHT	@ A ... O P
-------	-------------------------	---------

DEVICE	PRINTER KEYBOARD MAGSTRIPE TYPEV MULTIFUNCTION READER MFR [CAD MARKS NONE HOLES [CUT CAD HOLES NOCUT]] DISPLAY [LARGECHAR STDCHAR] BARCODE [READER] [UPC I25 I39 MAT EAN IVL]
--------	--	---------

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

Edit Statements

MINLEN <i>value</i> ["msg"]	$\left\{ \begin{array}{c} \text{GT} \\ \text{LT} \\ \text{GE} \\ \text{LE} \\ \text{EQ} \\ \text{NE} \end{array} \right\}$	<i>value</i> ["msg"]
$\left\{ \begin{array}{c} \text{IN} \\ \text{NIN} \end{array} \right\}$	$\left\{ \begin{array}{l} \textit{value}[, \textit{value}] \\ \textit{lowvalue}:\textit{highvalue}[, \textit{lowvalue}:\textit{highvalue}]... \\ \textit{value}[, \textit{lowvalue}:\textit{highvalue}]... \end{array} \right\}$	["msg"]
MATCH <i>pattern</i> ["msg"]		$\text{CDIGIT} \left\{ \begin{array}{c} 10 \\ 11 \end{array} \right\} \text{ ["msg"]}$

Formatting Statements

STRIP	$\left\{ \begin{array}{c} \text{TRAILING} \\ \text{LEADING} \\ \text{ALL} \end{array} \right\}$	"character"	JUSTIFY	$\left\{ \begin{array}{c} \text{LEFT} \\ \text{RIGHT} \\ \text{CENTER} \end{array} \right\}$
FILL	$\left\{ \begin{array}{c} \text{LEADING} \\ \text{TRAILING} \end{array} \right\}$	"character"	UPSHIFT	

SET Statements

SET	$\left\{ \begin{array}{l} \text{TO} \left\{ \begin{array}{l} \textit{value} \\ \textit{index OF } \textit{value}[, \textit{value}]... \end{array} \right\} \\ \left[\left\{ \begin{array}{l} \textit{field} \\ \textit{save field} \end{array} \right\} \left[\text{TO} \left\{ \begin{array}{l} \textit{value} \\ \textit{index OF } \textit{value}[, \textit{value}]... \end{array} \right\} \right] \right] \end{array} \right\}$
-----	--

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

CHANGE Statement

CHANGE	{ CFORM TO { NOREPEAT REPEAT REPEAT APPEND } NFORM TO { [CLEAR APPEND FREEZE APPEND] }	["formname" fieldname index OF value[,value]... \$RETURN \$HEAD \$END \$REFRESH]]]

Conditional Statement

<pre> IF [value] editstatement THEN [statement]... [statement]... [ELSE [statement]...] [statement]... </pre>
--

Control Statements

<pre> FAIL ["msg"] STOP </pre>

Operands

Whenever *value* is specified as an operand in a field processing statement, a field name, constant, expression, 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*, 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 (<i>any data type</i>).
\$LENGTH	equivalent to the length of the field the last time the form was compiled (<i>numeric data types</i>).
\$STATE	equivalent to a table of constants each of which is a two-character upper or lowercase United States state or territory code; refer to Appendix F for the codes (<i>character type; used only in table checks</i>).
\$TODAY	equivalent to today's date in the form <i>dd/dd/dd</i> where the order of month day year is determined by the field type (<i>date type only</i>).

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  
$STATE
```

Numeric. Numeric constants are digit strings with an optional leading sign and optional decimal point. Commas are not allowed in numeric constants. 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 length of the current field. The following examples illustrate numeric constants:

```
.23  
-123  
5000  
-3729  
$EMPTY  
$LENGTH
```

Date. A date constant can be any legal date format, but it must be in the order MDY. With Native Language Support, the date constant order (e.g. *mmdyy*) will not be language dependent. For more information on Native Language Support, see Section 8. 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, 1986!
!12/24/86!
!NOVEMBER 7, 86!
$TODAY
$EMPTY
```

Arithmetic Expression

Any numeric constant, field, or save field can be combined into an arithmetic expression. The expression is evaluated to generate a constant that 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 % 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/1000	Divide value in TOTALCENTS by 1,000.
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 the VPLUS/V intrinsics execute an arithmetic expression, they expect 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 user as if the expression had not been executed and no message is issued.

Index Retrieve Operand

An index retrieve operand (enclosed within parentheses) can be used in any edit statement where a constant, 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]....
```

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.

PROCESSING SPECIFICATIONS FOR CONFIGURATION

You can use the processing specifications for configuration to specify local editing to be done on the HP 2394A and HP 2624A/B terminal, and to specify the input device and prompting lights for use on the data capture devices.

These statements may only be used in the configuration phase of field processing, and must have the CONFIG heading in the processing specification area of the Field Menu.

NOTE

VPLUS/V edit processing specifications and terminal edit processing statements are separate and are not checked for compatibility. There will be no check to see that the designer has specified a terminal local edit (DEC_TYPE_EUR, DEC_TYPE_US) in the configuration phase which is consistent with the language-dependent decimal indicator for the native language specified in the forms file. For more information on Native Language Support see Section 8.

DEVICE

Enables the input media device on a data capture device.

Syntax

```
DEVICE {device1,device2,...devicen}
```

Parameters

device

Any of the following may be specified:

PRINTER
KEYBOARD
MAGSTRIPE
TYPEV
MULTIFUNCTION READER
DISPLAY
BARCODE READER

There are no default devices.

Discussion

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.

NOTE

The Interleaved 2 out of 5 bar code is only supported from the Globals Menu. If you wish to use this bar code format, it must be defined globally from that menu.

Example

Processing Specifications

```
CONFIG  
DEVICE PRINTER, KEYBOARD, DISPLAY, BARCODE READER
```

Specifies which light will be lit during field presentation on a data capture device.

Syntax

```
LIGHT {light1,light2,...lightn}
```

Parameters

light

Specifying @ turns 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.

Discussion

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.

Example

Processing Specifications

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

LOCALEDITS

Used to stipulate edits to be done within the terminal.

Syntax

```
LOCALEDITS {edit1,edit2,...editn}
```

Parameters

edit

Table 4-2 lists available local edits along with their meanings. Refer back to Table 4-1 for valid local edit combinations. For example, you can specify alphanumeric, required, decimal type, and number of digits to the right of the decimal together. You cannot specify alphanumeric and upshift together.

Discussion

This command allows you to specify local edits that are performed as keys are pressed by the user entering data into the form. At run-time, VPLUS/V checks the terminal type and, if the terminal supports local edits, loads the specified local edits to the terminal. If the terminal does not support local edits, VPLUS/V ignores them.

The last three decimal edits in Table 4-2 alter the mode of the terminal; therefore, the last field for which a decimal edit is specified that is displayed on the terminal alters the way the terminal processes *all* fields until the terminal mode is changed by another screen containing a decimal edit. All other local edit specifications affect only the field for which they are specified.

When an invalid key is pressed, the terminal beeps, and an error message issued by the terminal (not by VPLUS/V) is displayed. Press **RETURN** to clear, then reenter data into the field.

If you specify local edits with a form family, the local edits associated with the first displayed form in the family are retained as subsequent family members are displayed (in other words, it is not possible to change local edit specifications among family members at run-time).

VPLUS/V "edit" processing specifications and "terminal" edit processing statements are separate and are not checked for compatibility. There will be no check to see that the designer has specified a terminal local edit (DEC_TYPE_EUR, DEC_TYPE_US) in the configuration phase which is consistent with the language-dependent decimal indicator for the native language specified in the forms file. For more information on Native Language Support, see Section 8.

Table 4-2. Terminal Local Edits

Local Edit	Description
ALPHABETIC	Upper and lowercase alphabetic, space, period, hyphen.
ALPHANUMERIC	Upper and lowercase alphabetic, digits, space, period, minus sign, plus sign, comma.
CONSTANT	No characters may be entered from the keyboard.
IMP_DEC	Implied decimal (the number of digits to the right of the decimal is governed by the last DEC_DIGITS command in the form).
IMP_DEC_FILL	Implied decimal, right justified and zero filled (the number of digits to the right of the decimal is governed by the last DEC_DIGITS command).
INTEGER	Digits and spaces.
INTEGER_FILL	Digits and spaces, right justified and zero filled.
SIGN_DEC	Signed decimal.
SIGN_DEC_FILL	Signed decimal, right justified and zero filled.
TRANSMIT_ONLY	No data may be entered unless the cursor control keys are used to access the field.
UNRESTRICTED	Any character.
UPSHIFT	Lowercase alphabetic changed to uppercase when entered.
JUSTIFY	Right justify numeric types; left justify character types.
MUST_FILL	The field must be blank or filled entirely.
REQUIRED	The field must have an entry when ENTER is pressed.
DEC_TYPE_EUR	European number format.
DEC_TYPE_US	United States number format.
DEC_DIGITS <i>n</i>	Specifies number of digits to right of decimal point.

LOCALEDITS

Here is a table showing the local edit terminal menu selection and associated VPLUS/V local edits.

Table 4-3. Local Edits Corresponding to Field Type Selections in the Terminal Edit Checks Menu

Local Edit	Terminal Menu Selection	Justify is used
UNRESTRICTED	0. ALL CHARACTERS	Left Justify
ALPHABETIC	1. ALPHABETIC	Left Justify
UPSHIFT	2. AUTO UPSHIFT	Left Justify
ALPHANUMERIC	3. ALPHANUMERIC	Left Justify
INTEGER	4. INTEGER	Right Justify
SIGN_DEC	5. SIGNED DECIMAL	Right Justify
IMP_DEC	6. IMPLIED DECIMAL	Ignore
CONSTANT	7. CONSTANT	Ignore
INTEGER_FILL	8. INTEGER/FILL	Ignore
SIGN_DEC_FILL	9. SIGNED DECIMAL/FILL	Ignore
IMP_DEC_FILL	10. IMPLIED DECIMAL/FILL	Ignore

Table 4-4. Local Edits Corresponding to Attribute Selection in the Terminal Edit Checks Menu

Local Edit	Terminal Menu Selection	Default
REQUIRED	REQUIRED	OPTIONAL
JUSTIFY	JUSTIFY	NO JUSTIFY
MUST_FILL	TOTAL FILL	NO TOTAL FILL

For each field, the user may select one edit from Table 4-3 and any combination of edits from Table 4-4, according to the following guidelines:

1. If no selection is made from Table 4-3, the default edit of UNRESTRICTED is used.
2. The CONSTANT and REQUIRED edits are incompatible, since the required data cannot be entered in the CONSTANT field.
3. The terminal executes all edits implied by field type before it checks for TOTAL_FILL. The MUST_FILL edit is always met by INTEGER_FILL, SIGN_DEC_FILL, and IMP_DEC_FILL.
4. The effect of the JUSTIFY attribute on each of the field types is shown in Table 4-3.

Table 4-5. Local Edits Corresponding to Format Mode Selection in the Terminal Configuration Menu

Local Edit	Terminal Menu Item	Menu Selection
DEC_DIGITS <i>n</i>	Implied Dec digits	0 - 9
DEC_TYPE_EUR	Decimal Type	EUR
DEC_TYPE_US	Decimal Type	US

Along with the selections from Tables 4-3 and 4-4, you may specify the number of decimal digits (DEC_DIGITS *n*) and select a decimal format (DEC_TYPE_US or DEC_TYPE_EUR). (Note that the DEC_DIGITS *n* setting may only be used with IMP_DEC or IMP_DEC_FILL edits.) Unlike other edits, DEC_DIGITS and DEC_TYPE edits are global terminal configurations that remain in effect until they are reset by another DEC_TYPE or DEC_DIGIT edit. Please note that VOPENTERM and VCLOSETERM do not reconfigure DEC_DIGIT and DEC_TYPE; these settings may carry over from one application to the next.

Example



Processing Specifications

CONFIG
LOCALEDITS CONSTANT

\No characters may be entered
\from the keyboard.

CONFIG
LOCALEDITS DEC_TYPE_US,REQUIRED

\Entered data must be in United States
\number format; the field must contain
\data when ENTER is pressed.

PROCESSING SPECIFICATIONS FOR FIELD EDITS

If only field editing is needed, the edit statements listed below provide full field edits. Edit statements apply to the current field only. During table and range checks leading and trailing blanks are stripped. (The current field is the field defined in the Field Menu in which the edit statements are specified.)

Edit Statements

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

- is at least a specified minimum length; refer to "Length Check" for the MINLEN command.
- is greater than, equal to, or less than a specified value; refer to "Single Value Comparisons" for the GT, GE, LT, LE, EQ, and NE commands.
- is one of a list (table) of values, or is not in the list; refer to "Table Checks" for the IN and NIN commands.
- is within a range of specified values; refer to "Range Checks" for the IN and NIN commands.
- has a valid check digit; refer to "Check Digit" for the CDIGIT command.
- matches a general pattern of characters; refer to "Pattern Match" for the MATCH command.

Edit statements use the set of statements listed in Table 4-1. 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.

LENGTH CHECK

Used to specify the minimum length of a field value.

Syntax

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

Parameters

- value* The value can be a field, constant, save field, or a numeric expression, or it may be an index retrieve operand within parentheses. Refer to "Statement Syntax" earlier in this section for details.
- message* Specifies a custom error message.

Discussion

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 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. The \$LENGTH constant is equal to the length of the field when the forms file is compiled.

Example

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

```
MINLEN 2
```

If the user 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 filled, leaving no leading or trailing blanks:

```
MINLEN $LENGTH
```

LENGTH CHECK

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

Used to compare the field value against a single value.

Syntax



Parameters

value

The value can be a field, constant, save field, or a numeric expression, or it may be an index retrieve operand within parentheses. Refer to "Statement Syntax" earlier in this section for details.

message

Specifies a custom error message.

Discussion

You may use these statements to compare the contents of the current field with a specified value. 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. Collating sequence order is used to determine the numeric value of a character. (Refer to Appendix C for the USASCII collating sequence; refer to the *Native Language Support Reference Manual* for the collating sequences for all supported native languages.) For example, for NATIVE-3000:

a is greater than A
A is greater than Δ
AA is greater than AΔ
AΔ is greater than Δa

where Δ represents a blank.

SINGLE VALUE COMPARISONS

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 IMP_n can be compared with NUM_n, and NUM_n or IMP_n 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, 1986, specified in the current field as 9 FEB 1986 (DMY) can be successfully matched against the constant !02/09/86! whose order is MDY. Remember: a date constant must always be in the order MDY. For example:

2/3/86 (MDY) *is equal to* Feb 3, 1986 (MDY)
2/3/86 (MDY) *is equal to* 86/2/3 (YMD)

Example

- | | |
|-------------|--|
| 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. |
| GT "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 in NATIVE-3000 all uppercase characters have a lower value than any lowercase character. |
| GE !5/7/86! | Any date including and after May 7, 1986 may be in current field. The date may be in any of three formats: MDY, DMY, or YMD. |

Native Language Support

NATIVE-3000 must be used to define values for date and numeric fields within FORMSPEC. VPLUS/V will convert the value when the forms file is executed to be consistent with the native language selected. Single value comparisons (GT, GE, LT, LE, EQ, NE), specified within FORMSPEC may contain any character in the 8-bit extended character set consistent with the selected language ID. When the form is executed, the collating table for the native language specified is used. For more information on Native Language Support, see Section 8.

TABLE CHECKS

Used to verify that the field value is either in a list (table) of values, or is not in that list.

Syntax

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

Parameters

value The values specified in the list may be any mixture of field names, constants, save fields, numeric expressions, or it may be an index retrieve operand within parentheses. Refer to "Statement Syntax" earlier in this section for details.

message Specifies a custom error message.

Discussion

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 *NOR* value2 *NOR*...

One system defined table, \$STATE, is provided that consists of a list of all two-character United States state and territory 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. The \$STATE constant 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.

Example

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 legitimate state code or be MX. Note that the system constant \$STATE can be part of a list including other values.

TABLE CHECKS

Native Language Support

NATIVE-3000 must be used to define values for date and numeric fields within FORMSPEC. VPLUS/V will convert the value when the forms file is executed to be consistent with the native language selected. Table checks specified within FORMSPEC may contain any character in the 8-bit extended character set consistent with the selected language ID. For more information on Native Language Support, see Section 8.

RANGE CHECKS

Used to check whether the field value is within, or is not within, a specified range of values.

Syntax

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

Parameters

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

message Specifies a custom error message.

Discussion

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 NOR range2 NOR...

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).

Example

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:!
12/12/99! Any date between today's date and December 12, 1999 may be entered.

RANGE CHECKS

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. In NATIVE-3000, all lowercase letters are greater than any uppercase letters.
- IN "20":"30" Allows any character values between 20 and 30 in the USASCII collating sequence. Thus 2A and 3& are valid for NATIVE-3000. (Other native languages use other collating sequences.)

Native Language Support

NATIVE-3000 must be used to define values for date and numeric fields within FORMSPEC. VPLUS/V will convert the value when the forms file is executed to be consistent with the native language selected. Range checks specified within FORMSPEC may contain any character in the 8-bit extended character set consistent with the selected language ID. When the form is executed the collating table for the native language specified is used to check whether the field is within range. For more information on Native Language Support, see Section 8.

CHECK DIGIT

Used to test the check digit in a numeric or alphanumeric field (modulus 10 or 11).

Syntax

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

Parameters

message Specifies a custom error message.

Discussion

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 REFSPEC, as described in Section 5, to add a check digit to an existing field value in a batch file.)

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 and the CDIGIT edit test is selected, then the last digit is checked against the same calculations. If the number was keyed incorrectly, 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

Used to test a field value against a pattern of characters.

Syntax

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

Parameters

- pattern* A pattern of characters which indicate which type of character in which position is acceptable input for this field.
- message* Specifies a custom error message.

Discussion

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.

NOTE

Native Language Support does not support pattern matching with native language characters. MATCH uses USASCII (NATIVE-3000) rules. For more information on Native Language Support, see Section 8.

Special Pattern Characters

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

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

PATTERN MATCH

The beginning of the pattern is defined by the first nonblank character after `MATCH` and, in the simplest case, is terminated by the first blank encountered. A pattern beginning or ending with the special character `b` will interpret the `b` as the alphabetic character and not the special character indicating a blank. Leading and trailing blanks should be indicated as spaces within a grouping as described on the next page.

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 user) is also used to allow inclusion of any of the pattern operators listed below, and described in Table 4-3.

- ! (transparency)
- ,
- :
- { } (grouping)
- [] (optional)
- + (repetition--1 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.
```

PATTERN MATCH

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 USASCII 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 the letter 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 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. For example; 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]
dd
[% ,d]} Identical to the example above, except that each pattern component is listed on a separate line.

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. Leaving the field blank is not acceptable unless the field type is O for optional, in which case a blank is accepted because processing specifications are ignored.

For example, 2 or 7654321 or 55 are acceptable.

MATCH Xd+ This pattern accepts the letter X followed by one or more digits.

For example: 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 M12333CAA9.

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.

For example, 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 uppercase 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.

PATTERN MATCH

Operator Hierarchy

Table 4-6 summarizes the operators allowed in a MATCH pattern. The pattern operators are evaluated in the following order, where x and y are any patterns:

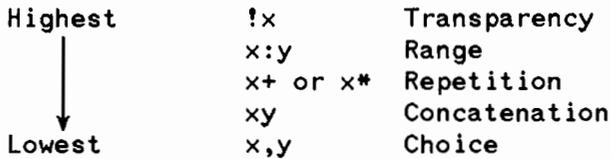


Table 4-6. Pattern Match Operators

Operator	Function	Example
! Exclamation point	Transparency operator that allows use of any special MATCH characters as an element in the pattern.	MATCH !u,!d,!,,!! accepts any of the values u, d,, or !.
, Comma	Choice of subpatterns, any one of which satisfies the match.	MATCH A,B,dd accepts values such as A,B, and 22.
: Colon	Range of single characters in ascending collating order, any one of which satisfies the match.	MATCH 2:6 accepts only the values 2, 3,4,5, or 6.
{ } Braces	Required grouping that specifies at least one occurrence of any pattern within braces.	MATCH {A,B}dd{%,d} accepts A223, B34%,A795, and so forth.
[] Brackets	Optional grouping that allows zero or one occurrence of any item in pattern within brackets.	MATCH [A,B]dd[%,d] accepts 24,A99, 10%,123, and so forth.
+ Plus	Required repetition that specifies at least one or more occurrences 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, or 33ABC9.
* Asterisk	Optional repetition that 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.

Example

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]]	Accepts 12-hour clock time, such as 12:15 PM or 01:30 AM.
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 United States Social Security number, such as 044-24-0474.
MATCH [(ddd)]b*ddd-dddd	Accepts a United States phone number with an optional area code.



PROCESSING SPECIFICATIONS FOR ADVANCED PROCESSING

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

- **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.
- **Data formatting.** When data is moved between fields with the SET statement, 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 use the FORMSPEC formatting statements to reformat the current data field during data collection. These statements are: STRIP, JUSTIFY, FILL, UPSHIFT.
- **Altering forms sequence.** The CHANGE statement allows you to alter the sequence in which forms are displayed.
- **Conditional processing.** The IF statement allows you to specify under which conditions the processing statements are to be executed.
- **Control Statements.** Two statements provide control over forms processing. The FAIL statement forces a failure of the current field edit, the STOP statement stops all processing of the current phase of the current form.
- **Processing phases.** Processing statements may be associated with one of four phases 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.

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 refer to "Phases" at the end of the section before reading the statement descriptions. In any case, you should be aware that all field edit and advanced processing statements can be executed in all phases, except configuration. Only configuration processing statements are executed in the configuration phase. Phases allow terminal configuration preceding initial processing for the form, which in turn precedes field edits and finish processing.

SET

Used to move data either by setting the current field to a value or by moving data between fields. Also performs automatic formatting.

Syntax

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

Parameters

source

Can be a field name, save field name, constant, arithmetic expression, or an index retrieve operand within parenthesis. Refer to "Statement Syntax" earlier in this section for details.

destination

Can be a field name or save field name.

Discussion

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 4-10 shows the conversion that is performed when data is moved between fields.

SET

Assigning a Value to the 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:

Syntax

```
SET [fieldname] TO source
```

Parameters

<i>fieldname</i>	The current field name (default).
<i>source</i>	Can be a field name, save field name, constant, arithmetic expression, or an index retrieve operand within parenthesis. Refer to "Statement Syntax" earlier in this section for details.

Discussion

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 4-8).

By default, all fields are initialized to blanks in the initialization phase (refer to "Phases" at end of this section for more information on phases.) You can specify a particular initial value for any field by including an initial value in the Field Menu field attributes. 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 user input at a terminal. (Remember that input in FORMSPEC is NATIVE-3000; if a native language is specified, data is converted at run-time. Refer to Section 8.) When you assign a value through the SET statement, you have more flexibility. 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.

If a source field is of date type (MDY, YMD, DMY), the current field must be defined to be 8 bytes long to allow for internal delimiters. Some examples using the SET TO *source* statement follow.

Example

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 10,1986! \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, the save field 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.

SET

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:

Syntax

SET	{ <i>destination</i> }
	{ <i>destination TO source</i> }

Parameters

source Can be a field name, save field name, constant, arithmetic expression, or an index retrieve operand within parenthesis. Refer to "Statement Syntax" earlier in this section for details.

destination Can be a field name or save field name.

Discussion

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.)

Example

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.

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 4-10 for a summary of the conversion performed during data movement.

Character Type

If the destination is a character field, data moved to it does not change its relative position. 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 on the right.

Table 4-7. Automatic Formatting for Character Data

Source	Destination
"ARMSTRONG" (9 characters)	"ARMSTRONGΔ" (10 characters)
"ΔΔARMSTRONG" (11 characters)	"ΔΔARMSTRON" (10 characters)
"ARMSTRONGΔΔΔΔΔ" (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 after being stripped of trailing zeros following 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.

SET

Leading Zeros Leading zeros are stripped in all cases.

The result is then placed, right justified, as shown in Table 4-8, in the destination field.

With Native Language Support, decimal and thousands indicators are language-dependent in the NUM[n] and IMP[n] fields. When data is moved between fields and automatic formatting occurs for data entered in any field, recognition, removal or insertion of these decimal and thousands indicators also depends upon the local custom for the native language specified for the forms file. The optional decimal symbols in constants will be local custom-dependent. For more information on Native Language Support, see Section 8.

Table 4-8. Automatic Formatting for Numeric Data

Source		Destination	
123	(DIG)	123	(DIGIT, length is 5)
123	(IMP2)	1.23	(NUM2, length is 4)
12.3	(NUM1)	12.3	(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, as shown in Table 4-9. Thus, if a date is going to be placed into a field using a SET command, the field must be at least eight characters long. The order depends on whether the destination is specified as MDY, DMY, or YMD. With Native Language Support, conversion from alphabetic months to the numeric destination month is language dependent. For more information on Native Language Support, see Section 8.

Table 4-9. Automatic Formatting for Dates

Source	Destination
FEB 5, 1986	02/05/86 (defined as MDY)
2/5/86	86/02/05 (defined as YMD)
2,5,86	02/05/86 (defined as MDY)
February 5, 1986	05/02/86 (defined as DMY)
September 16, 1986	09/16/86 (defined as MDY)
Oct. 23, 1986	86/10/23 (defined as YMD)
October 23, 1986	23/10/86 (defined as DMY)

Table 4-10. Conversion During Data Movement

From	To					
	CHAR	NUM	NUMn	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,insert -ing 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
NUMn						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

SET

Default Formatting

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

```
SET TO thisfield
```

Parameters

thisfield is the name of the field in which SET appears. Essentially, you set the field to itself.

For example, you may want to ensure 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 Field Menu in Figure 4-3 illustrates this use of SET.

FORMSPEC v.uu.ff Field Menu				FORM NAME: filename							
QTY	PART NO.	DESCRIPTION		UNIT	TOTAL						
				PRICE	PRICE						
qty..	partnum..	description.....		uprice	tprice						
				^							
Num	[10]	Len	[6]	Name	[UPRICE]	Enh	[HI]	FType	[R]	DType	[NUM2]
Initial Value											
*** Processing Specifications ***											
SET TO UPRICE \formats UPRICE value when entered											
PREV	NEXT	FIELD	REFRESH	PREV	NEXT	MAIN/	EXIT				
FORM	FORM	TOGGLE				RESUME					

Figure 4-3. Field Menu with a SET Statement

Example

1. Suppose the user 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 user enters a date in the form:

[FEB 12, 86]

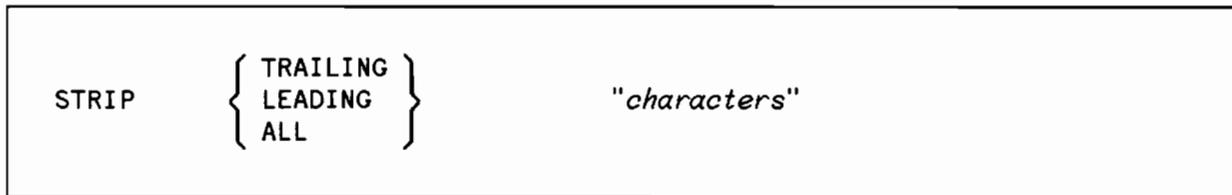
A SET statement of the form SET TO DATE1 in the Field Menu describing DATE1 formats the user entered data as:

[02/12/86ΔΔ]

STRIP

Used to delete any specified character in the field.

Syntax



Parameters

TRAILING If TRAILING is specified, all occurrences of each character at the end of the field are replaced with blanks.

LEADING STRIP LEADING replaces with blanks all occurrences of each character at the beginning of the field.

ALL STRIP ALL deletes all occurrences of each character, compressing the data to the left.

characters Any string of one or more characters can be specified. Since the statement does not apply to leading or trailing blanks, the statements STRIP LEADING " " or STRIP TRAILING " " are not useful.

Discussion

The STRIP statement can be used to format data entered into or moved to any field. (For automatic formatting, refer to the SET command earlier in this section.) As is true of all formatting statements, STRIP has an immediate effect on the data in the memory buffer; the formatted data is then available for display or copying into the application. When the user enters data into a field for which formatting is specified, the data is formatted as soon as **ENTER** or an appropriate function key is pressed. Thus, FORMSPEC formatting is done before data is written to the data or batch file. (In contrast, reformatting through REFORMAT, described in Section 5, is done after data is collected, edited, and written to the batch file.)

Note that only the characters between the leftmost nonblank and the rightmost nonblank characters are affected. That is, leading and trailing blanks are not included when the data is formatted. No formatting occurs when the field is blank since there are no characters to strip.

Example

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

Used to move data within a field to the left or right boundary of the field, or center it within the field.

Syntax

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

Parameters

- LEFT The data in the current field is shifted to the left.
- RIGHT The data in the current field is shifted to the right.
- CENTER The data in the current field is centered in the field.

Discussion

The JUSTIFY statement can be used to format data entered into or moved to any field. (For automatic formatting, refer to the SET command earlier in this section.) As is true of all formatting statements, JUSTIFY has an immediate effect on the data in the memory buffer; the formatted data is then available for display or copying into the application. When the user enters data into a field for which formatting is specified, the data is formatted as soon as **ENTER** or an appropriate function key is pressed. Thus, FORMSPEC formatting is done before data is written to the data or batch file. (In contrast, reformatting through REFORMAT, described in Section 5, is done after data is collected, edited, and written to the batch file.)

Example

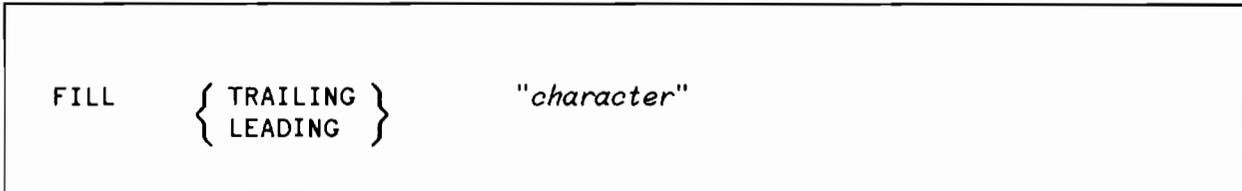
Statement	Data Entered	After Formatting
JUSTIFY LEFT	[ΔΔSMITHΔΔΔΔ]	[SMITHΔΔΔΔΔΔ]
JUSTIFY RIGHT	[ΔΔSMITHΔΔΔΔ]	[ΔΔΔΔΔΔSMITH]
JUSTIFY CENTER	[ΔΔSMITHΔΔΔΔ]	[ΔΔΔSMITHΔΔΔ]

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

FILL

Used to replace all the blanks between the field boundaries and the first or last nonblank data character with a designated character.

Syntax



Parameters

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

LEADING FILL LEADING replaces the blanks preceding the data with the specified character.

character The specified *character* is any single character.

Discussion

The FILL statement can be used to format data entered into or moved to any field. (For automatic formatting, refer to the SET command earlier in this section.) As is true of all formatting statements, FILL has an immediate effect on the data in the memory buffer; the formatted data is then available for display or copying into the application. When the user enters data into a field for which formatting is specified, the data is formatted as soon as **ENTER** or an appropriate function key is pressed. Thus, FORMSPEC formatting is done before data is written to the data or batch file. (In contrast, reformatting through REFORMAT, described in Section 5, is done after data is collected, edited, and written to the batch file.)

Note that no formatting occurs when the field is blank.

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.

Used to shift every lowercase character in a field to its uppercase equivalent.

Syntax

```
UPSHIFT
```

Discussion

The UPSHIFT statement can be used to format data entered into or moved to any field. (For automatic formatting, refer to the SET command earlier in this section.) 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. With Native Language Support, if a native language ID has been specified in the forms file, the UPSHIFT formatting statement will use native language upshift tables. For more information on Native Language Support, see Section 8.

As is true of all formatting statements, UPSHIFT has an immediate effect on the data in the memory buffer; the formatted data is then available for display or copying into the application. When the user enters data into a field for which formatting is specified, the data is formatted as soon as **ENTER** or an appropriate function key is pressed. Thus, FORMSPEC formatting is done before data is written to the data or batch file. (In contrast, reformatting through REFORMAT, described in Section 5, is done after data is collected, edited, and written to the batch file.)

Example

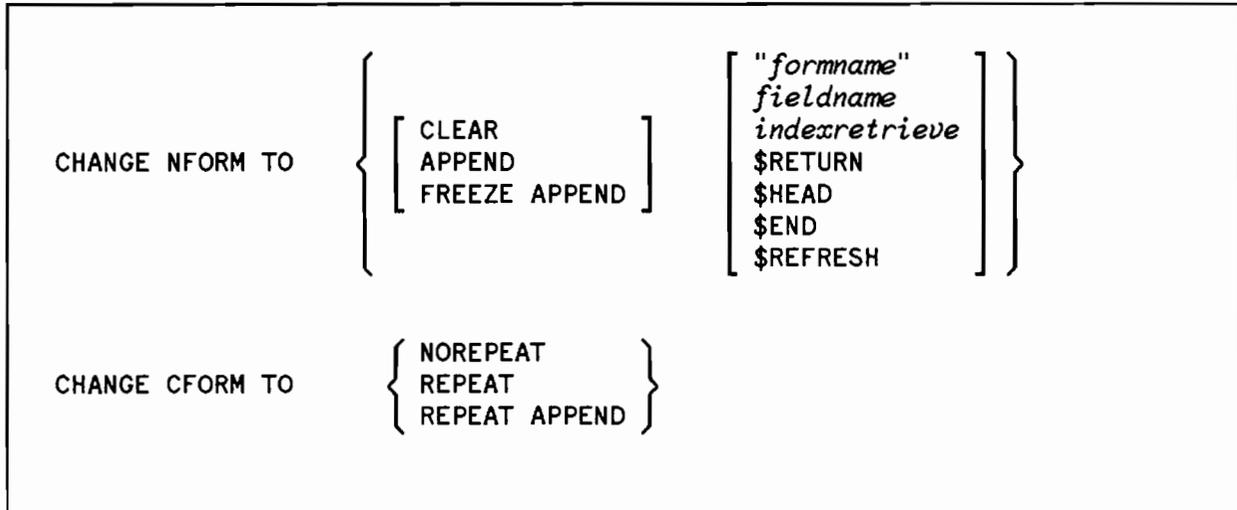
Suppose the field contains an United States state code that is to be checked against a list of uppercase state codes. The following statements ensure that codes entered in lowercase pass the edit:

```
UPSHIFT  
IN "NY", "NJ", "PA"
```

CHANGE

Used to change the specification of either the current form or the next form.

Syntax



Parameters

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.
<i>formname</i>	is the name of any existing form in the forms file.
<i>fieldname</i>	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 "*name*" [,"*name*"]...

where *index* is an integer, and *name* is a form name or any field or save field whose value is a form name. The *name* 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 to ENTRY or an application 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.

Discussion

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 finished 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 3), 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.

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 APEND "FB"
```

The next form, FB, is to be appended to the current form.

IF

Used to execute any of the processing statements only under certain conditions.

Syntax

```
IF condition THEN [statement]  
    [statement]  
    .  
    .  
    [statement]  
[ ELSE [statement]  
    [statement]  
    .  
    .  
    [statement] ]
```

where *condition* is:

```
[ constant  
  field  
  save field  
  expression  
  (indexretrieve) ] editstatement
```

Parameters

condition

A condition specified in an IF statement can be any edit statement described below. 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. Refer to "Statement Syntax" earlier in this section for details. If an operand precedes the edit statement, that value is tested; otherwise, the value of the current field is tested.

If editing statement passes, then the condition is true. If it does not, then the condition is false. Note that this differs from interpretation of a standalone 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.

editstatement

A condition specified in an IF statement can be any edit statement MATCH and CDIGIT

statement

Any of the processing statements can be executed conditionally depending on the run-time interpretation of the specified condition. Refer to the syntax rules below.

Discussion

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. Table 4-11 illustrates some variations on the IF statement according to the syntax rules, which are:

- 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 "Phases" later in this section.)

Example

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.

IF

Table 4-11. Variations on the IF Statement

Variation	Description
IF <i>condition</i> THEN <i>statement</i>	Simple IF statement. If <i>condition</i> is true, then <i>statement</i> is executed; if false, then <i>statement</i> is not executed.
IF <i>condition</i> THEN <i>statement</i> <i>statement</i> <i>statement</i>	If <i>condition</i> is true, all three <i>statements</i> are executed in the order specified. If <i>condition</i> is false, none of the three <i>statements</i> is executed.
IF <i>condition</i> THEN <i>statementA</i> ELSE <i>statementB</i>	If <i>condition</i> is true, <i>statementA</i> is executed; otherwise, <i>statementB</i> is executed.
IF <i>condition</i> THEN <i>statementA</i> <i>statementB</i> <i>statementC</i> ELSE <i>statementD</i>	The <i>statementA</i> , <i>statementB</i> and <i>statementC</i> are executed if the <i>condition</i> is true; <i>statementD</i> is executed if the <i>condition</i> is false,
IF <i>condition</i> THEN <i>statementA</i> <i>statementB</i> <i>statementC</i> ELSE <i>statementD</i>	If <i>condition</i> is true, <i>statementA</i> , <i>statementB</i> , and <i>statementC</i> are executed; otherwise <i>statementD</i> is executed.
IF <i>condition1</i> THEN IF <i>condition2</i> THEN <i>statementA</i> <i>statementB</i> ELSE IF <i>condition3</i> <i>statementC</i> <i>statementD</i> ELSE <i>statementE</i>	<p>Only if <i>condition1</i>, <i>condition2</i> are both true, <i>statementA</i>, <i>statementB</i> are executed.</p> <p>Only if <i>condition1</i>, <i>condition3</i> are true, but <i>condition2</i> is false, is <i>statementC</i> executed.</p> <p>Only if <i>condition1</i> is true, regardless of whether <i>condition2</i>, <i>condition3</i> are true, is <i>statementD</i> executed.</p> <p>Only if <i>condition1</i> is false, is <i>statementE</i> executed.</p>

Used to force failure of a field edit.

Syntax

```
FAIL ["message"]
```

Parameters

message Specifies a custom error message.

Discussion

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.

Example

For example, suppose you want to ensure that an entered value is in a table of values. You can use the FAIL statement to ensure 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 in the table, the FAIL statement is not performed nor is the field cleared to blank.

STOP

Used to stop processing the current phase of the current form.

Syntax

```
STOP
```

Discussion

When STOP is executed, no further processing is performed on the current field, or on any subsequent fields in the form in this phase.

Example

For example, if using ENTRY, processing of the entire forms file could be terminated when "END" is entered in the current field with the following specification:

```
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.

The STOP command ends processing of this form; using the \$END constant as the next form instructs ENTRY to end processing of all forms.

Used to determine when specified processing specifications are executed.

Syntax

```
phaseheader  
[statement]  
.  
.  
[statement]
```



Parameters

phaseheader

One of the following headings. The phase headings *must* be specified in the order shown:

CONFIG	Perform the following configuration during form presentation.
INIT	Perform the following statements during initialization (VINITFORM).
FIELD	Perform the following statements during the field edit phase (VFIELDDEDITS).
FINISH	Perform the following statements during the finish phase (VFINISHFORM).

Configuration processing specifications must always be preceded by the phase header CONFIG. The INIT header can be omitted if the initialization processing specifications are followed by the FIELD phase header and field edit processing specifications. The FIELD phase header can be omitted if there are no configuration or initialization processing specifications. The FINISH header must always be specified to indicate that a statement is to be executed in that phase.

statement

A processing statement to be executed in the specified phase.

Discussion

During the configuration phase, local edits are loaded to terminals that support the local edit feature (see Appendix G), and the output device and error lights to be lit are identified for data capture devices. During the initialization phase, all fields in the form are initialized; during the field edit phase, all fields (in screen order left to right, top to bottom) are validated and edited. An application usually requests execution of the finish phase only when all fields have passed the edit tests of the field phase. Finish

PHASES

statements apply to the entire form. (Refer to Figure 4-4 for an illustration of the flow of form execution under ENTRY control.)

You can specify that a statement be executed during a particular phase with the phase specification statements. It is important to note that while specifications are entered field by field, execution of the phases applies to the entire form. If you omit these phase headings altogether, all statements are executed in the field edit phase.

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 TO SAVE_FIELD_1
```

If you want statements to be executed in a particular phase, you must precede the statements by the appropriate phase headings. Phase headings must be specified in the order shown above.

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. The commands that can be used in the configuration phase are: DEVICE, LIGHT and LOCALEDITS. (Refer to the "Processing Specifications for Configuration" earlier in this section.)

Initialization Phase

The initialization phase occurs when VINITFORM is called. 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 are included in the field processing specifications, these are executed next. This process continues until all fields in the form are initialized.

If the form being initialized is a child or sibling to the previous form, data from the previous form is transferred to this form (with conversion if necessary) before initializations occur. Refer to "Form Families" in Section 3 for more information.

A SET TO source statement may be used to initialize field values.

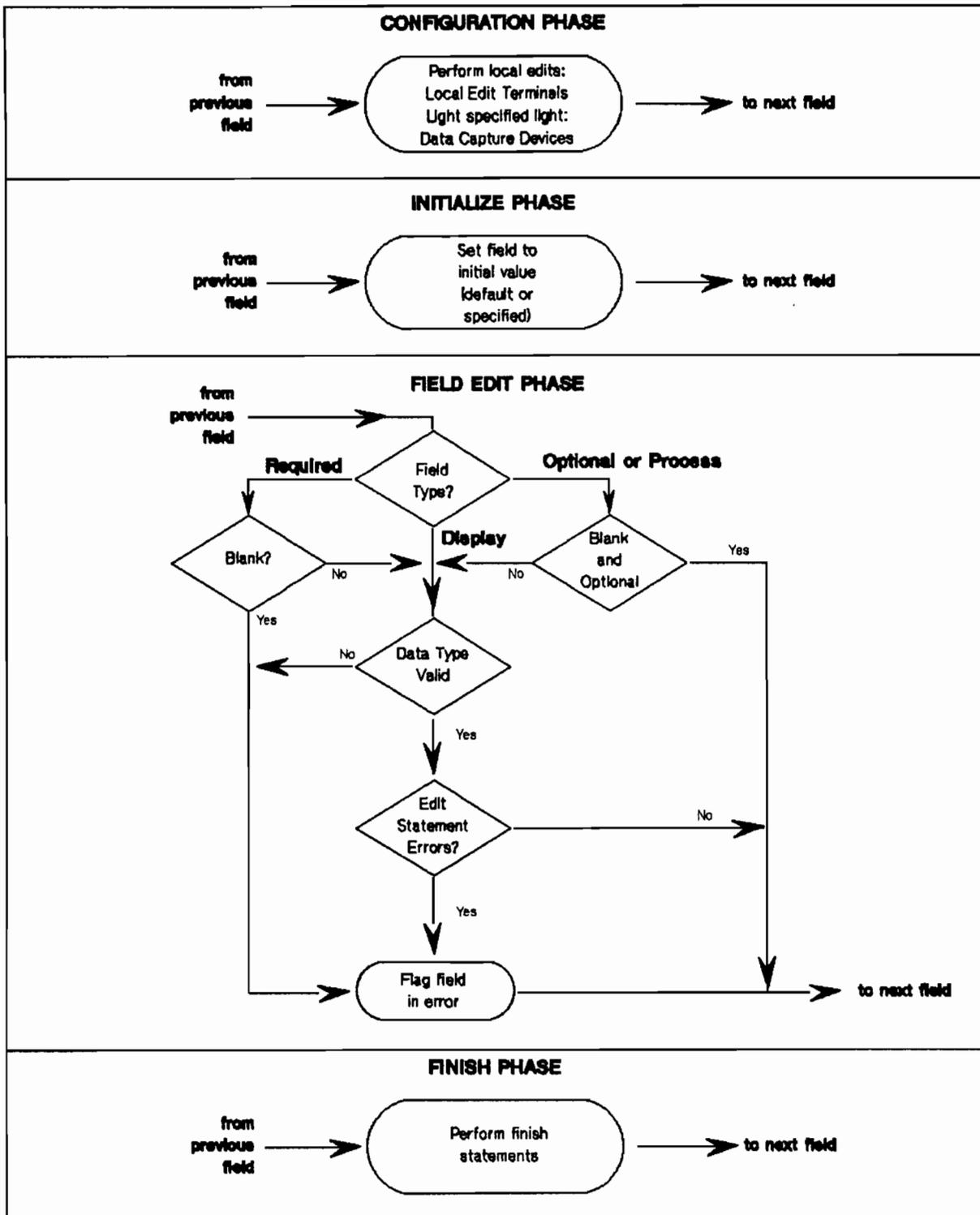


Figure 4-4. Flow of Phase Execution for ENTRY

PHASES

Field Edit Phase

The field edit phase occurs when VFIELDEDITS is called. The field edit phase is usually requested by the application after the form is displayed on the screen, and the user has input data. During this phase, the entered data is checked according to the field attributes and any processing specifications entered by the designer. The Edit statements can be executed only in the field edit phase.

Each field in turn is examined. The first action is to check the field type. If it is optional and the field is blank, the rest of the field edit phase for this field is skipped. If the field is required, it must contain a nonblank 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 user can be informed. All further processing on this field is stopped when an error is detected, and control passes to the next field.

ENTRY loops through all processing statements in the field edit phase until all errors have been corrected by the user.

Finish Phase

The finish phase occurs when VFINISHFORM is called. The finish phase usually is requested by the application after all data has been entered and validated for the entire form. The ENTRY program executes statements in this phase only when no errors are detected after **ENTER** is 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 user presses **REFRESH**, the initial values replace any that were entered. In the finish phase, such changes can no longer be made.

Under ENTRY control, the batch file is written as soon as the finish phase is complete. The assumption is that all fields have been tested and corrected; fields with errors are not enhanced in the finish phase.

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 4) 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 REFORMAT. 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 conjunction with an application programmer to specify how data is to be reformatted. Use of REFORMAT is described at the end of this section.

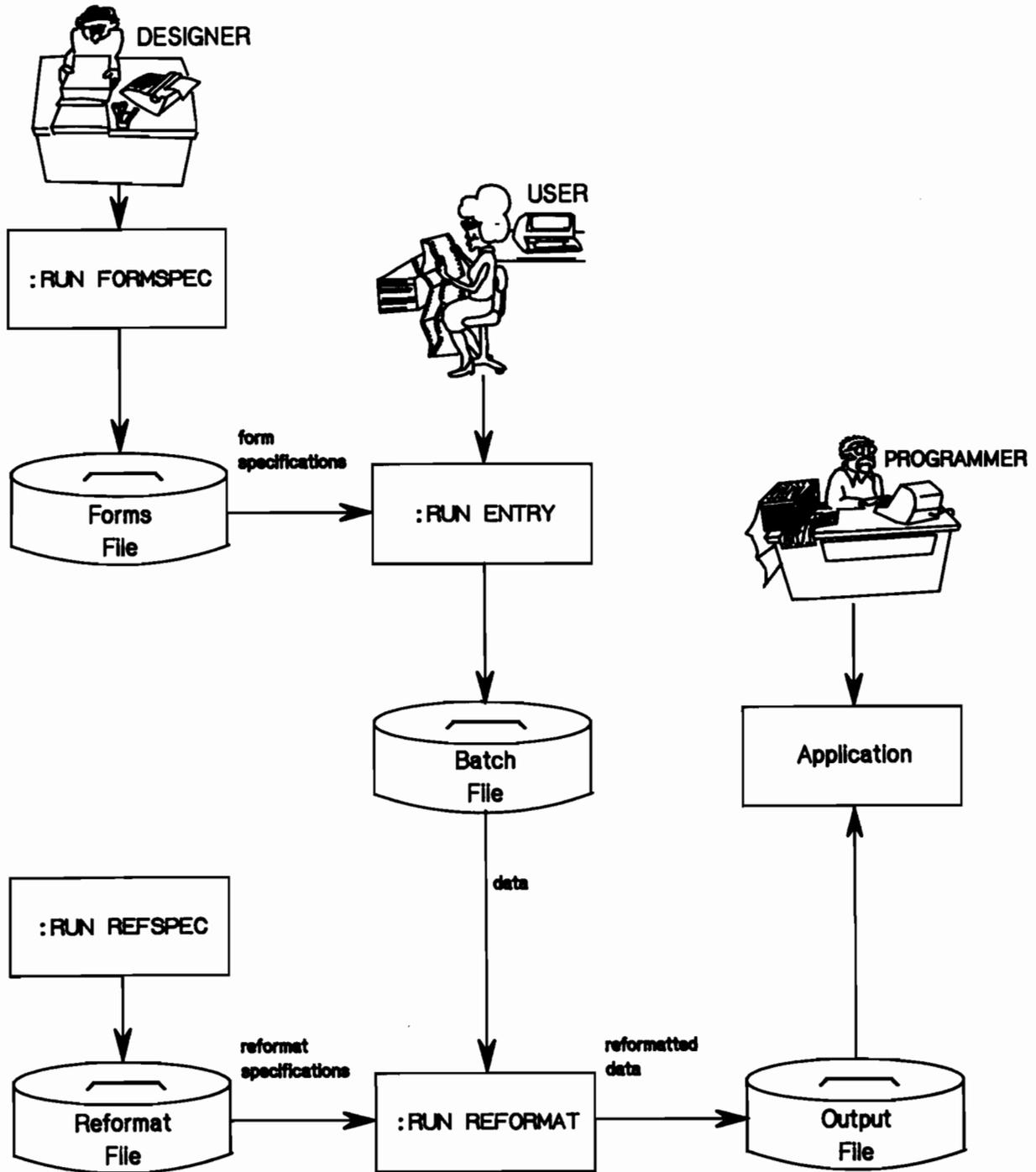


Figure 5-1. Relation Among Files Used for Reformatting

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 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 an MPE :FILE command equation. (See "Concatenating Batch Files", this section, for an example of this.) 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. REFSPEC accepts either KSAM or standard MPE reformat files. The only forms files created by REFSPEC, however, are MPE files. In the MPE reformat files, the key and form records are interspersed throughout the file. MPE reformat files do not require extra data segments and are automatically recovered after a power failure or system crash. Refer to Appendix H for information on converting KSAM files to MPE files.

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 is 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.

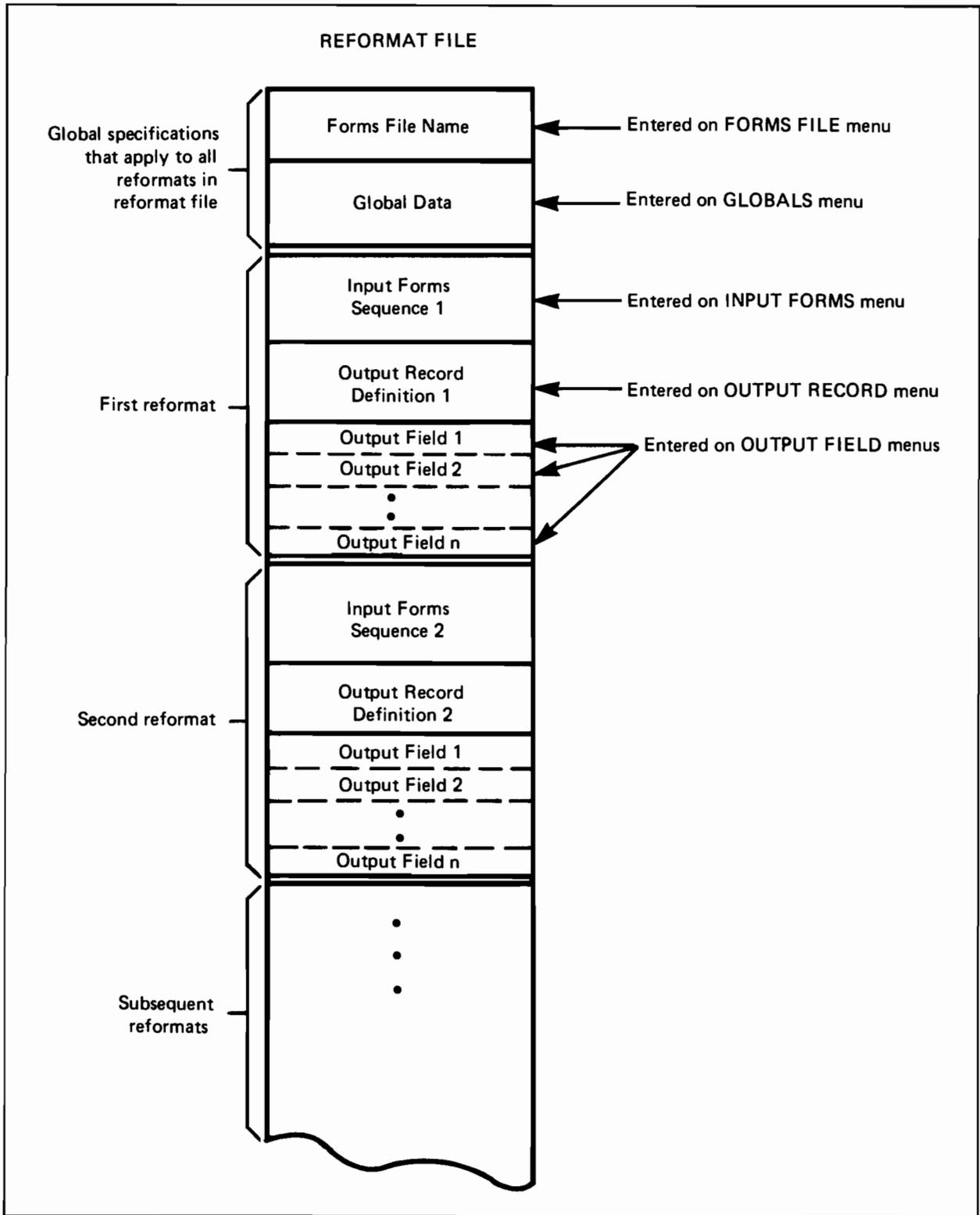


Figure 5-2. Prototype of REFORMAT File

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 sequence 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, REFORMAT reads each batch record in turn. Associated with each record is the form on which the data was entered. For the first batch record, REFORMAT 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 USASCII 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. REFSPEC 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 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 in 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.

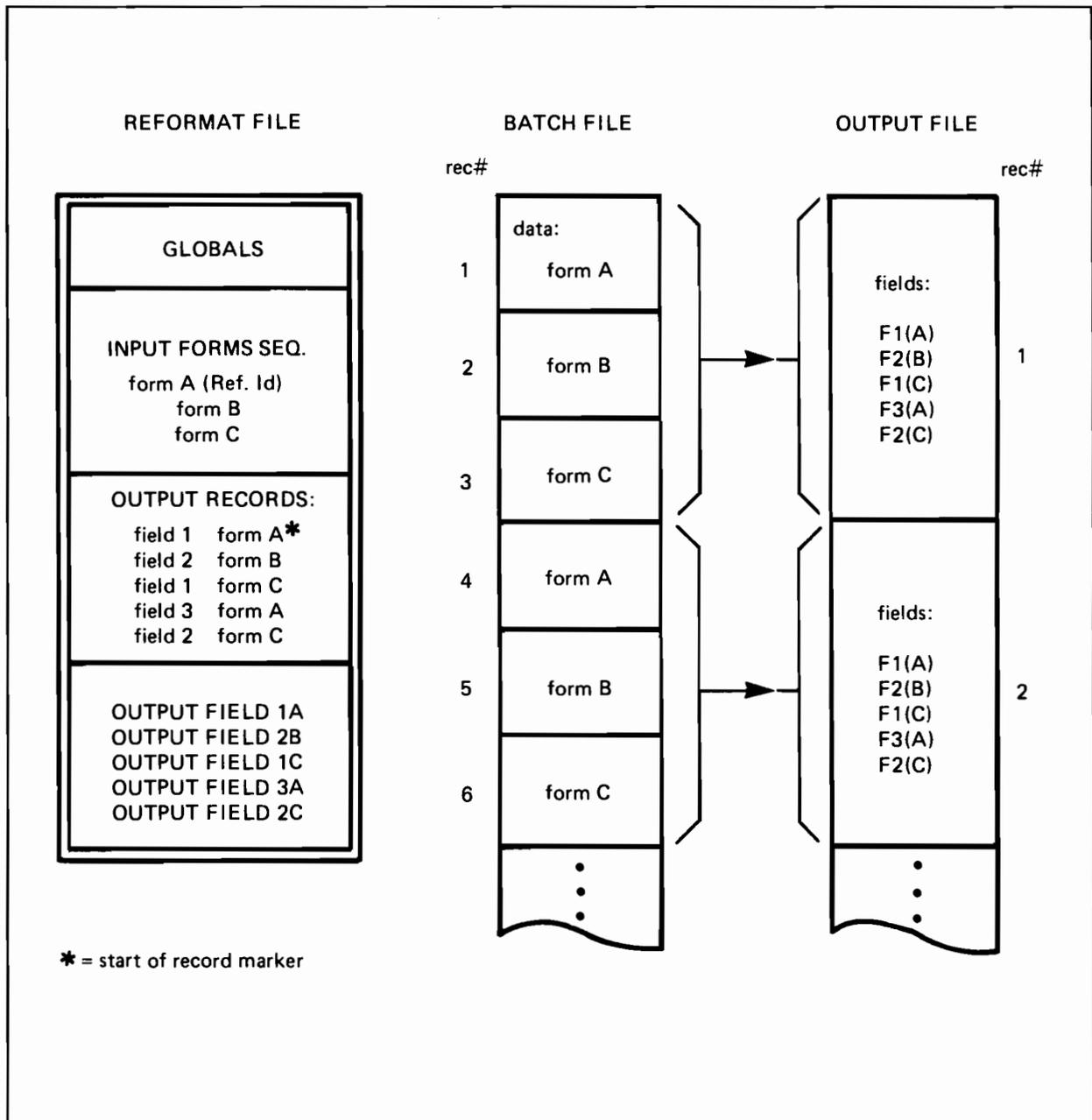


Figure 5-3. Combining Data From 3 Forms into 1 Output Record

Reformatting Specifications

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 field name for each occurrence (this is not specifically shown in Figure 5-4).

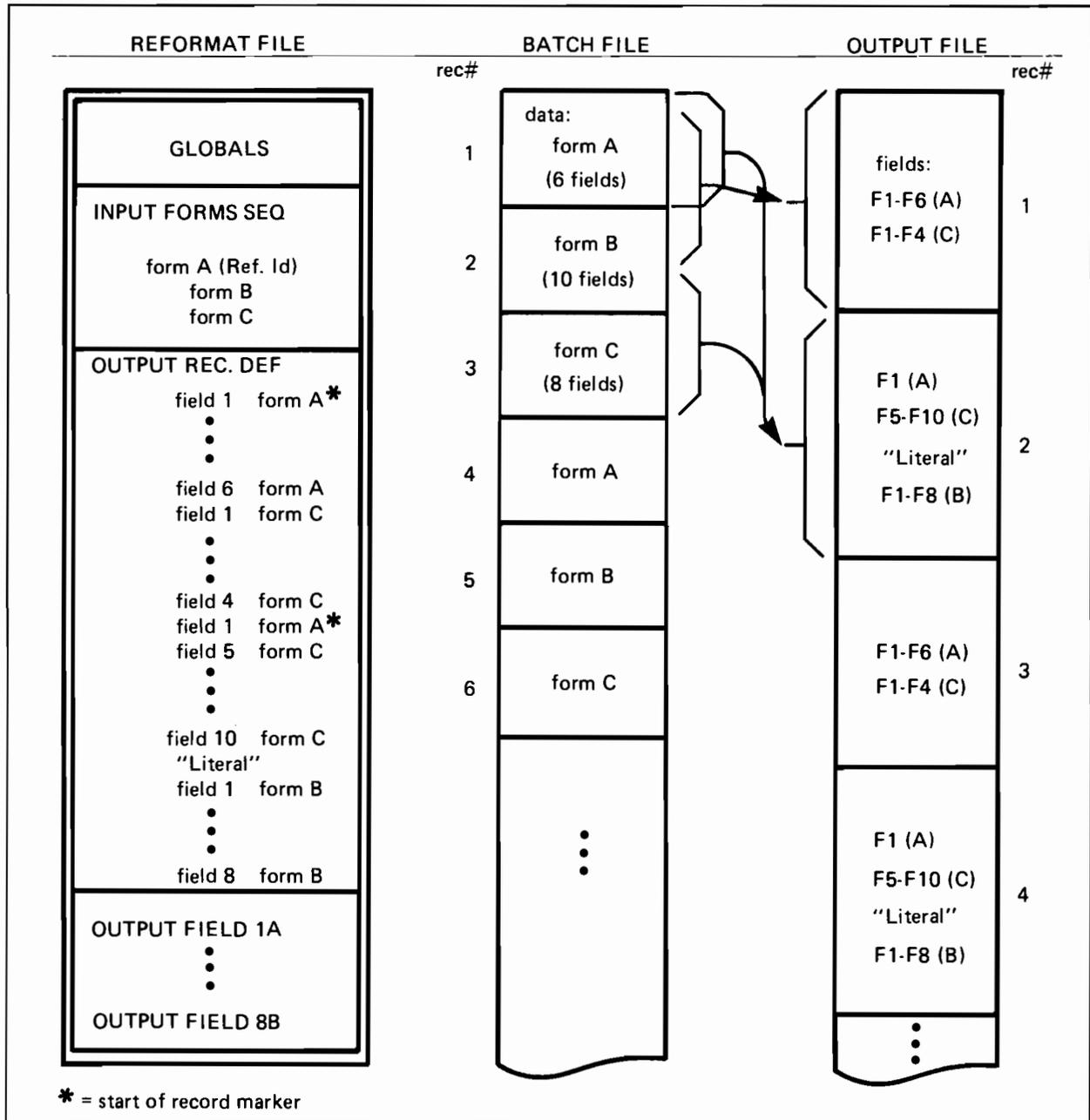


Figure 5-4. Reformat Data From 3 Forms into 2 Output Records

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 bytes. Data from a subsequent form is written to a single 40-byte record. If any group of data is less than the 40-byte record length, the record is padded with blanks. The data cannot be formatted to exceed the record length.

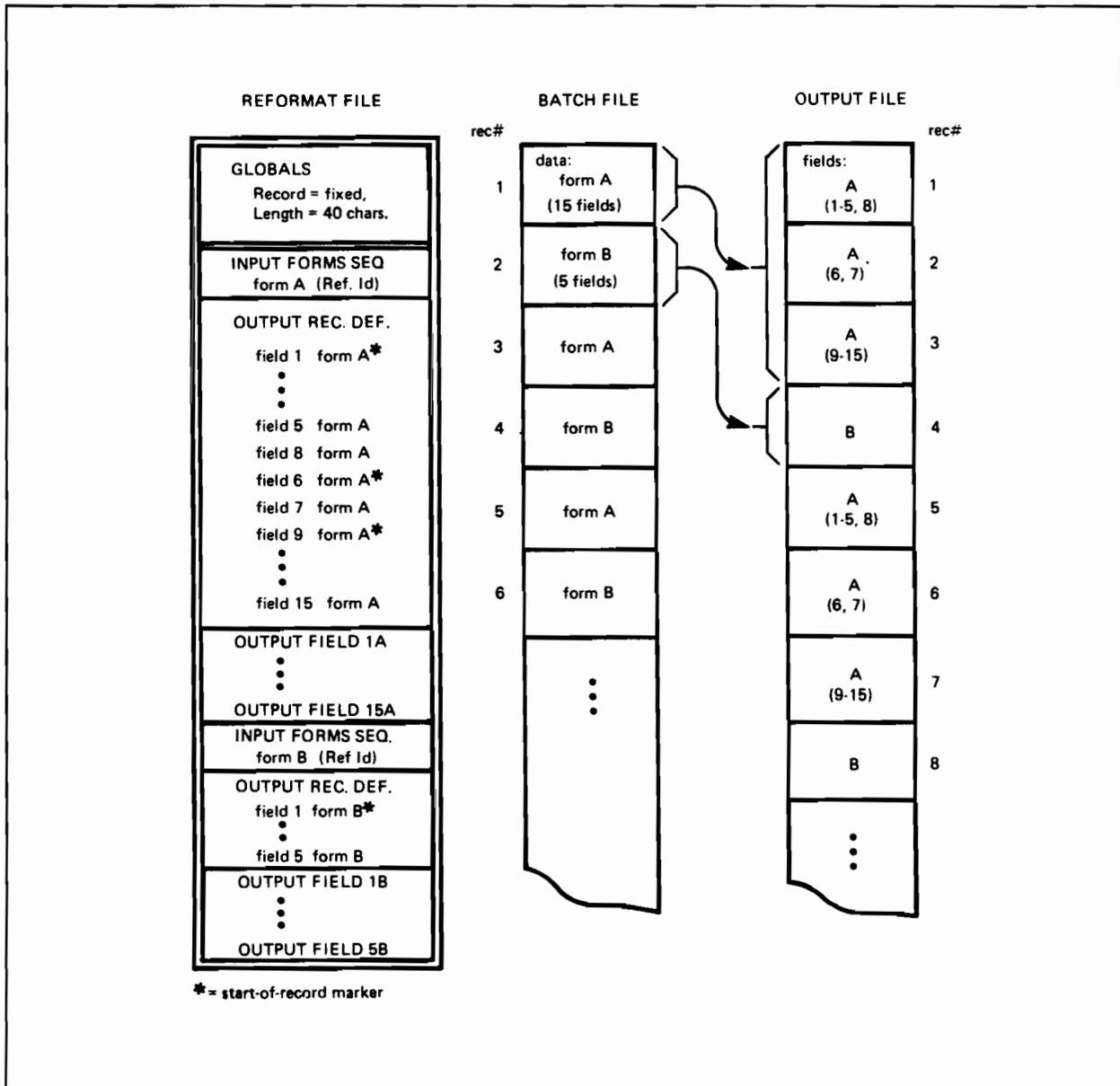


Figure 5-5. Separating Data From 1 Form into Several Records

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 REFORMAT is run.

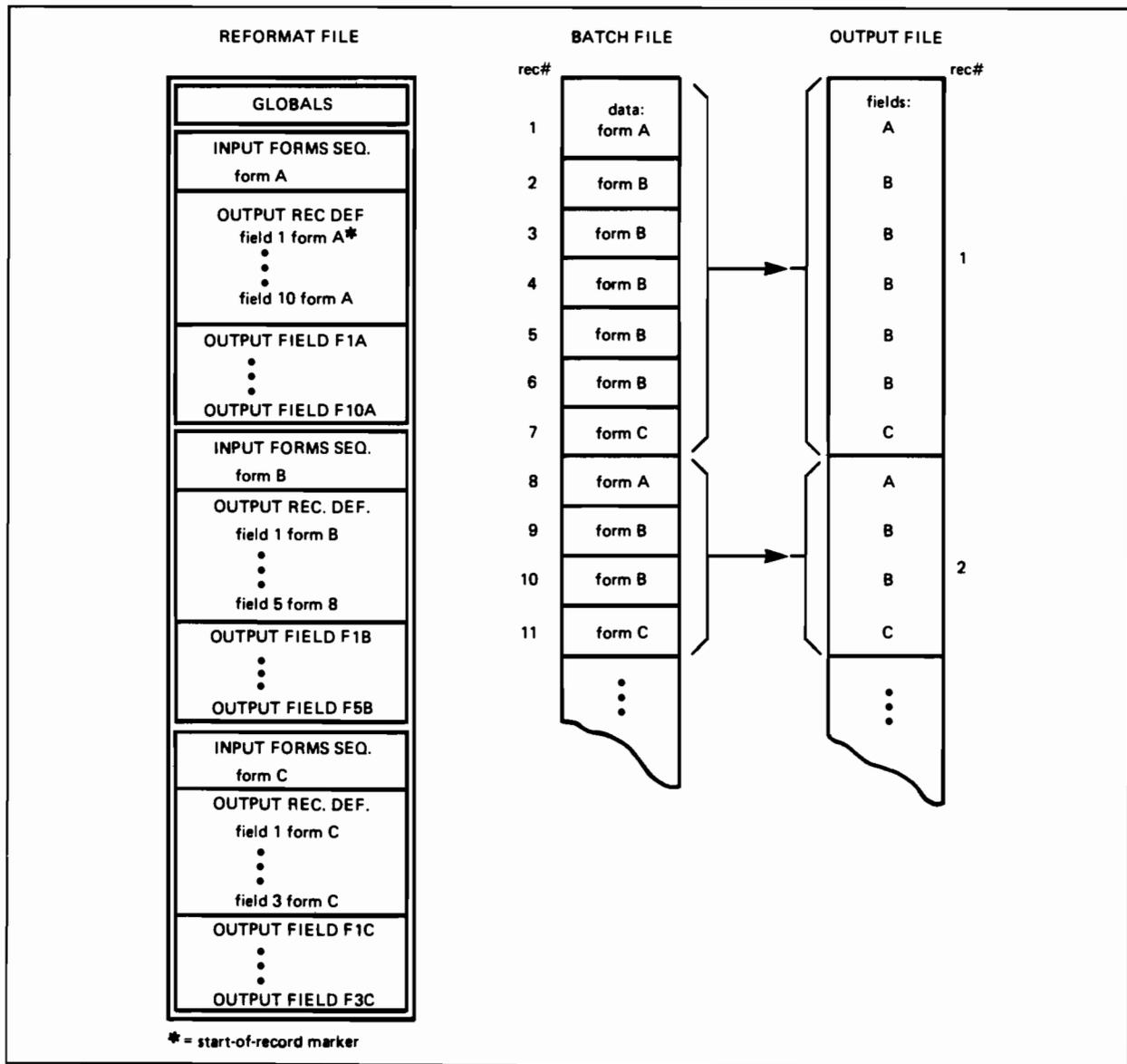


Figure 5-6. Reformatting Data From Repeating Forms

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.

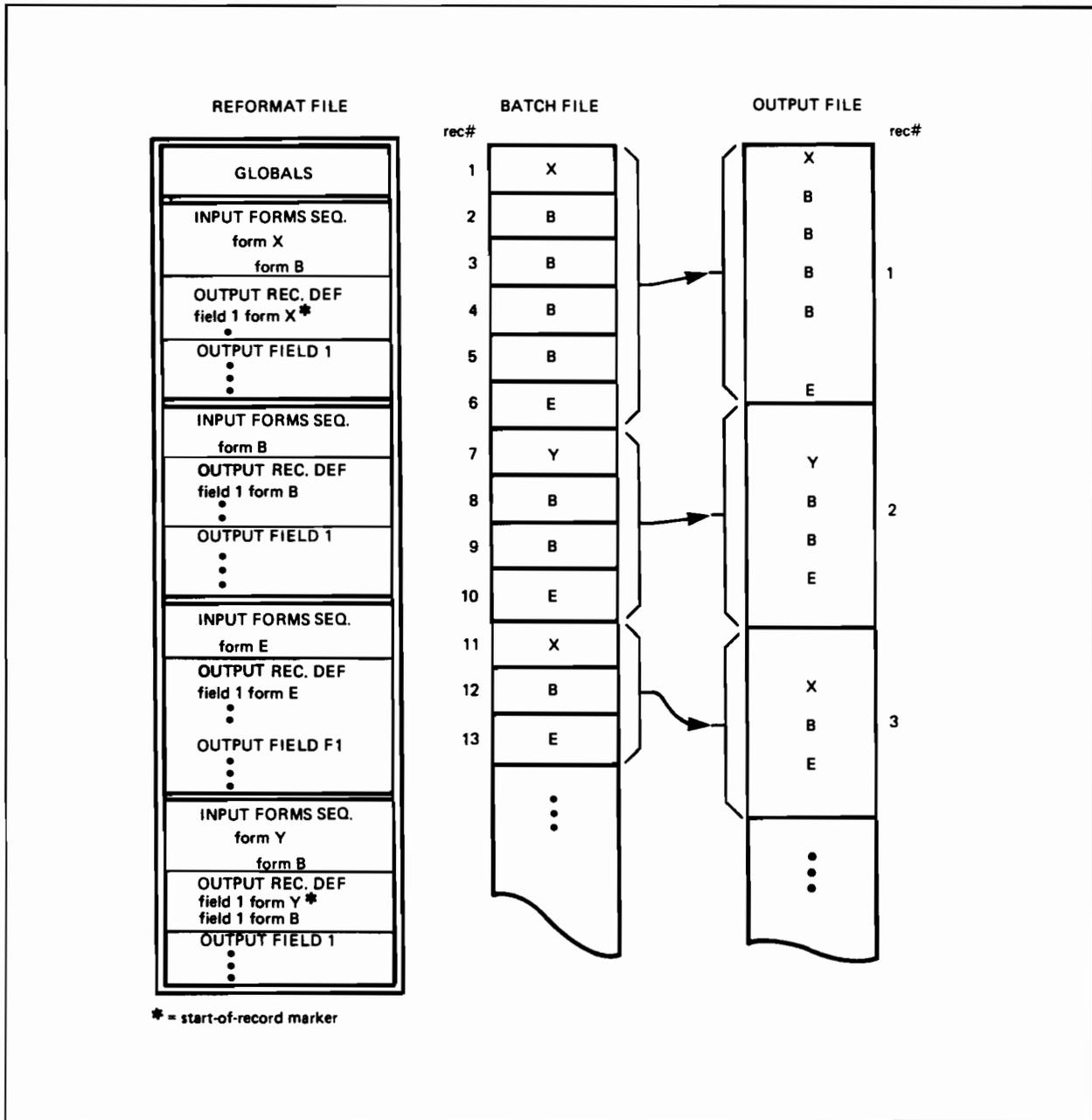


Figure 5-7. Reformatting Data Based on Form Sequence

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.

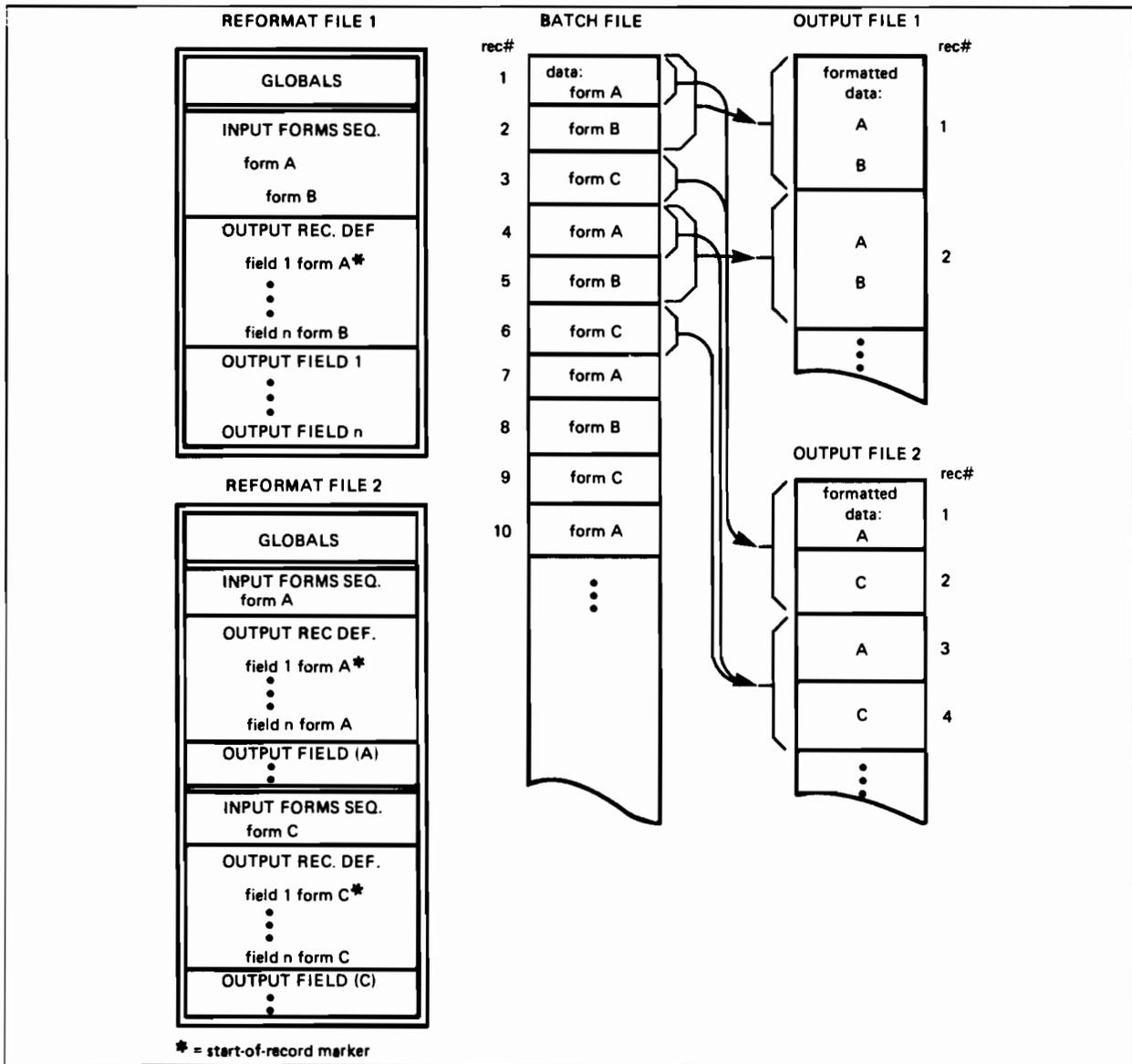


Figure 5-8. Generating 2 Output Files From 1 Batch file

USING REFSPEC

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

```
:RUN REFSPEC.PUB.SYS
```

Since REFSPEC runs entirely in block mode, your terminal is 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.

NOTE

If your file has a lockword, you *must* enter the lockword along with the file name, as *filename/lockword*. If you do not enter a lockword with the file name and one is required, your terminal hangs. This is because the MPE prompt requesting the lockword is in character mode and cannot be received while your terminal is in block mode. You can recover from this error by doing a hard reset followed by pressing **RETURN** at least four times, then pressing **FB** to exit. How you perform a hard reset depends on what type of terminal you are using; consult your terminal manual for instructions.

Terminating REFSPEC

You can terminate operation of REFSPEC at any time by pressing **EXIT** to return to MPE control, indicated by 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 **MAIN/RESUME** 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 user pressing **BREAK**, control returns to MPE. Refer to Appendix G for the steps to recover from such a situation. After you have recovered, you then press **REFRESH** to return to the REFSPEC Menu where you were interrupted. The menu is cleared to initial or previously entered values. To ensure 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 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 described in Table 5-1.

Table 5-1. Key Definitions

Key	Key Action
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).
f3	Not defined.
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 defined for ENTRY (see Figure 2-2). Although the two groups of keys are physically the same programmable keys, their functions differ significantly.

REFSPEC MENUS

REFSPEC displays menu screens in a predetermined sequence. As each menu is displayed, 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** to display the next menu.

When the reformatting is completely defined, request the Main Menu by pressing **MAIN/RESUME**. You can then compile the reformat file so that it can be executed.

The REFSPEC menu screens are shown in Figure 5-9 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 displayed.

You can move forwards or backwards through the sequence of menus to locate a particular menu by pressing **NEXT** or **PREV**, respectively. You can skip an entire reformat specification by pressing **NEXT REFORMAT** or **PREV REFORMAT**. (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 **MAIN/RESUME** and then selecting a particular menu.

Press **EXIT** to return to MPE control from REFSPEC.

Figure 5-9 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.

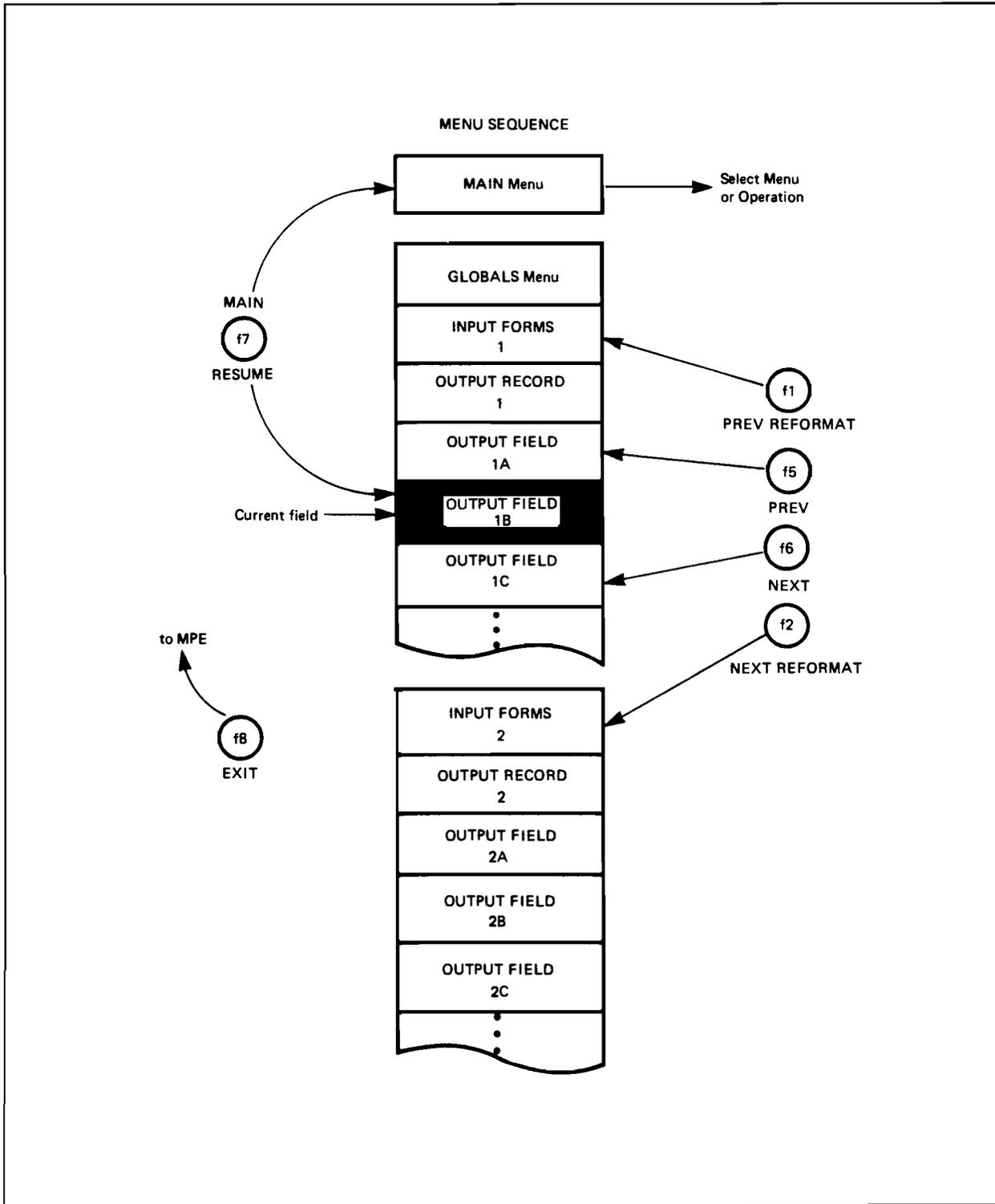


Figure 5-9. Relation of REFSPEC Menus to Function Keys

REFORMAT FILE MENU

The first menu displayed after you run REFSPEC is the Reformat File Menu shown in Figure 5-10. The only information entered on this menu is the name of the reformat file.

```
REFSPEC v.uu.ff Reformat File Menu

Reformat File Name [ ]

PREV REFORMAT  NEXT REFORMAT  REFRESH  PREV  NEXT  MAIN/RESUME  EXIT
```

Figure 5-10. Reformat File Menu

If the name identifies a new 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 5-12).

FORMS FILE MENU

The Forms File Menu, shown in Figure 5-11, 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. The name of the data file may be a fully qualified MPE file name.

```
REFSPEC v.uu.ff Forms File Menu                REFORMAT FILE: filename
Forms File Name [                               ]
PREV REFORMAT  NEXT REFORMAT  REFRESH          PREV  NEXT  MAIN/RESUME  EXIT
```

Figure 5-11. Forms File Menu

After you have identified the forms file, the Main Menu is issued.

NOTE

In order to maintain (and browse or list) a forms file, the user must have exclusive write access to the file.

MAIN MENU

The Main Menu, shown in Figure 5-12, 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.

```

REFSPEC v.uu.ff Main Menu                                REFORMAT FILE: filename

Enter Selection [ ]

      A--Add a reformat
      X--Compile Reformat File
      G--Go to GLOBALS Menu
      F--Go to FORMS FILE Menu

      G--Go to   reformat id           output field
      L--List   reformat id
      D--Delete                reformat id

                               [ ] [ ]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
REFORMAT REFORMAT  REFORMAT  REFORMAT  RESUME  REFORMAT

```

Figure 5-12. 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. New reformats are added to the end of the reformat file. To add a reformat, simply enter (A) in the Main Menu selection box. In response, REFSPEC displays the Input Forms Menu, shown in Figure 5-15.

Compiling the Reformat File

In order for REFORMAT 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 nondefault global characteristics, enter **G** in the Main Menu selection box, but do not specify a reformat identifier or output field name. The Globals Menu, shown in Figure 5-14, is 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 :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 **PREV REFORMAT** or **NEXT REFORMAT**. 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 changed 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 **ENTER** 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. The 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 **NEXT** or **PREV**. 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 **REFORMAT**. 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 5-13 illustrates the listing of a reformat.

Resuming Design from Main Menu

If you requested the Main Menu by pressing **MAIN/RESUME** during reformat design, you can return to the menu you were in by pressing **MAIN/RESUME** again. Note that this is the same key in each case, 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** or **NEXT** 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** or **NEXT REFORMAT**. The previous or next reformat is relative to the reformat you were designing when **MAIN/RESUME** was requested.

GLOBALS MENU

The Globals Menu, shown in Figure 5-14, requests information that applies to the total reformat process. All information entered on the Globals Menu relates to the output file produced when REFORMAT is run. Since only one output file is generated for each REFORMAT execution, this information appears only once in the reformat file.

```

REFSPEC v.uu.ff Globals Menu                                REFORMAT FILE: filename

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  _____

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
REFORMAT REFORMAT  RESUME
    
```

Figure 5-14. Globals Menu

When the Globals Menu is displayed, default values are shown for each option. The possible values and their defaults are listed below. You can change any of the values or you can keep the default values provided by REFSPEC. Press **ENTER** to record the values and request the next menu, the Input Forms Menu.

Fields

Output 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 user who runs 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 the user.

Default = N (do not upshift).

If nothing is specified, an error is returned.

Convert to EBCDIC?

Indicate whether the output file is to be written in EBCDIC code rather than USASCII.

Y - write output file in EBCDIC

N - leave output file in USASCII

Default = N (do not convert).

Record Terminator String

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 USASCII characters, including blanks, enclosed within single or double quotes. For example: "eof" or " " or '***'.
- USASCII Code--Numeric equivalent to an USASCII character preceded by a dollar sign (\$). Code may be any decimal number in range 0-127 (refer to Appendix C for decimal equivalent to USASCII code). For example: \$34 is the numeric equivalent of quotation mark; \$65 is the equivalent of the letter 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 String

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 for this example is shown below.

The separator may be a quoted string, a USASCII code, or a system constant, as described above for the Record Terminator.

Unless a separator is specified, fields in an output record are not separated, but are written as one continuous string of characters.

Default = No separator.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
				A	B	C	D	*	*					X	Y	Z

INPUT FORMS MENU

The Input Forms Menu, shown in Figure 5-15, 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.

REFSPEC v.uu.ff Input Forms Menu REFORMAT FILE: filename

Forms in Input Sequence: [] (Reformat identifier)

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

PREV REFORMAT NEXT REFORMAT REFRESH PREV NEXT MAIN/RESUME EXIT

Figure 5-15. Input Forms Menu

Fields

Forms in Input Sequence

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 5-11.)

Reformatting Specifications

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 REFORMAT to match batch file records with input forms sequences in the reformat file.

REFORMAT:

- 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.

REFORMAT 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, shown in Figure 5-16, 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 freeform 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.

REFSPEC v.uu.ff Output Record Menu				REFORMAT IDENTIFIER: A			
INPUT				OUTPUT			
Field Name	Substring Strt Len	Form Name	Field Name	Strt Col	Len	Strt Rec	
^	^ ^	^	^	^	^	^	
1	2 3	4	5	6	7	8	

PREV REFORMAT	NEXT REFORMAT	REFRESH	PREV	NEXT	MAIN/ RESUME	EXIT
------------------	------------------	---------	------	------	-----------------	------

Figure 5-16. An Example of an Output Record Menu

When the Output Record 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 **TAB**, the cursor is moved to each subsequent position. Figure 5-17 illustrates an example of a completed Output Record Menu specification.

REFSPEC v.uu.ff 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				
ZIP			FORMA				
\$RS							
"Account Number:"							*
ACCTNO			FORMA	ACCTB			
\$RS							
-							

PREV REFORMAT
NEXT REFORMAT
REFRESH
PREV
NEXT
MAIN/RESUME
EXIT

Figure 5-17. Sample Output Record Menu

When the output record definition has been defined, press **ENTER**. A Field Menu is issued for each field name associated with an input field name in the output record definition.

Note that the only required entry is the field name; other specifications depend on the particular definition. You can specify a constant (literal string, system constant, or numeric equivalent of an USASCII character) instead of a field name. A constant may start anywhere on the line, but cannot extend past column 75.

NOTE

Since the output record specifications are entered on a freeform 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.

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 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.)

Fields

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.

Substring 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).

Substring 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.

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 ensure 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 a USASCII 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. When multiple constants are 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.

OUTPUT FIELD MENU

A separate Output Field Menu shown in Figure 5-18, is displayed for every input field named in the preceding Output Record Menu. The 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.

```

REFSPEC v.uu.ff 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 [ ]

PREV NEXT REFRESH PREV NEXT MAIN/ EXIT
REFORMAT REFORMAT RESUME RESUME
    
```

Figure 5-18. 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 5-2 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.

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, 1986. If the output data type is left as CHAR, the date is written to the output field as MAR 6, 1986. If, on the other hand, the output data type is changed to MDY, the date is written as 03/06/86. 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 5-2. 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.

Table 5-2. Standard Formatting by Data Type.

Destination Data Type	Standard Formatting
CHAR	<p>Conversion: none.</p> <p>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.</p> <p>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</p>
DATE	<p>Conversion and Movement: Convert entered date to <i>dd/dd/dd</i>, where the order of digits depends on the date type. A date is written to MDY type field as <i>mm/dd/yy</i>, to DMY type field as <i>dd/mm/yy</i>, to YMD field as <i>yy/mm/dd</i>. 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.</p> <p>Assume an MDY date, length=12 JAN 31, 1986 moved to MDY field, length=12 01/31/86 ΔΔΔΔ to DMY field, length=8 31/01/86 to YMD field, length=8 86/01/31 to any date field, length=7 ERROR</p>
DIGIT	<p>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.</p> <p>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</p>

Table 5-2. Standard Formatting by Data Type (continued)

Destination Data Type	Standard Formatting
NUMn or NUM	<p>Conversion and Movement: Strip any commas, sign, or leading zeros. Float any minus sign to position preceding the first nonblank character.</p> <p>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.</p> <p>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</p> <p>Assume signed value, length=10 +1234.510Δ moved to NUM2 field, length=10 ΔΔΔ1234.51 to NUM2 field, length=3 ERROR</p> <p>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</p>
IMPn	<p>Conversion and Movement: Strip any decimal point, commas, sign, or leading zeros. Float any minus sign to the position preceding the first nonblank 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.</p> <p>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</p> <p>Assume a value, length=7 .0120ΔΔ to IMP4 field, length=3 120 to IMP4 field, length=2 ERROR</p>

Reformatting Specifications

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 is 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 <i>characters</i>	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 <i>characters</i>	Strips each occurrence of each specified character only if it appears before any other nonblank 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 <i>characters</i>	Strips each occurrence of each specified character only if it appears after all other nonblank characters. Stripped characters are replaced by blanks.

In Table 5-3 are examples using STRIP where 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.

Table 5-3. Examples using STRIP

Data Type	Input Value	Specification	Output Value
CHAR	ΔACBAXCBYZBB	none STRIP ALL "ABC" STRIP LEADING "ABC" STRIP TRAILING "ABC"	ΔACBAXCBYZBB ΔXYZΔΔΔΔΔΔΔ ΔΔΔΔXCBYZBB ΔACBAXCBYZΔΔ
DIGIT	Δ234567ΔΔΔΔ	none STRIP ALL "35" STRIP LEADING "0" STRIP TRAILING "7"	ΔΔΔΔ1234567 ΔΔΔΔ12467ΔΔ ΔΔΔΔ1234567 ΔΔΔΔ3456Δ
NUM2	Δ12,345.67ΔΔ	none STRIP ALL "35" STRIP LEADING "01" STRIP TRAILING "7"	ΔΔΔ-12345.67 ΔΔΔ-124.67ΔΔ ΔΔΔΔ-2345.67 ΔΔΔ-12345.6Δ
IMP2	Δ12,34567ΔΔΔ	none STRIP ALL "35" STRIP LEADING "01" STRIP TRAILING "7"	ΔΔΔΔ-1234567 ΔΔΔΔ-12467ΔΔ ΔΔΔΔΔ-234567 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 nonblank, 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 nonblank, 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. In Table 5-4, the examples illustrate the three types of justification. It is assumed the input and output data types are the same.

Table 5-4. Examples of Three Types of Justification.

Data Type	Input Value	Specification	Output Value
CHAR	ΔΔABCDEFΔΔ	JUSTIFY R JUSTIFY L JUSTIFY C	ΔΔΔΔABCDEF ABCDEFΔΔΔΔ ΔΔABCDEF
DATE	1/30/86ΔΔΔ	none JUSTIFY R JUSTIFY C	1/30/86ΔΔ ΔΔ01/30/86 Δ01/30/86Δ
DIGIT NUM or IMP	ΔΔ123456ΔΔ	none JUSTIFY L JUSTIFY C	ΔΔΔΔ123456 123456ΔΔΔΔ ΔΔ123456ΔΔ

Sign

With 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 nonblank 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 5-5. Correspondence Between Signed Digits and Characters

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

In Table 5-6 are examples that illustrate the use of the SIGN option. The input and output fields are assumed to be the same data type.

Table 5-6. Example using the SIGN Option.

Data Type	Input Value	Specification	Output Value
NUM0	-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 nonprinting control character, nor may it be more than one character. If FILL is not specified, no default replacement is made.

FILL ALL Replaces all blanks in the data with the specified character.
character

FILL LEADING Replaces all leading blanks with the specified character.
character

FILL TRAILING Replaces all trailing blanks with the specified character.
character
Default = 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 examples in Table 5-7 illustrate use of the FILL option. The input and output data types are the same.

Table 5-7. Example using the FILL Option.

Data Type	Input Value	Specification	Output Value
CHAR	ΔΔABCΔDEFΔΔ	FILL ALL* FILL LEADING* FILL TRAILING*	**ABC*DEF** **ABCΔDEFΔΔ ΔΔABCΔDEF**
NUM2	ΔΔΔΔ123.75	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 in Table 5-8 produce a meaningless result because the sign is floated before the field is filled with zeros; the second set of specifications produces the desired result.

Table 5-8. Example using FILL LEADING and FILL ALL Option.

Input Value	Specification	Output Value
-ΔΔ999	FLOAT SIGN FILL LEADING 0	000-999
-ΔΔΔ999	SIGN LEFT FILL ALL 0	-ΔΔΔ999 -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 4).

If the data is right justified, the nonblank 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 nonblank digit.

CHECKDIGIT 11 Calculate check digit according to modulo 11 formula and add it following rightmost nonblank 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 modulus 11 calculation."

If the product generated by a CHECKDIGIT 11 calculation evaluates to 11, a zero is appended to the basic number.

RUNNING REFORMAT

Once the reformat specifications have been defined and are stored in a compiled reformat file, you can run REFORMAT to actually reformat the data in the batch file and write it to the output file.

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.

REFORMAT needs the names of three files: the "reformat" file containing 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=outputfile
```

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 the file equation that sends the listing to the line printer:

```
FILE TESTLIST;DEV=LP
```

Another important file, particularly when you are running 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
```

Concatenating Batch Files

Data from a batch file can be appended to data from another batch file in an existing output file by issuing the following commands:

```
:FILE REFFILE=reformatfilename
:FILE OUTFILE=outputfilename
:FILE BATCH=firstbatchfilename
:RUN REFORMAT.PUB.SYS
```

These commands will run REFORMAT using the three files specified in the :FILE commands. Next, issue the following commands:

```
:FILE OUTFILE=outputfilename;ACC=APPEND
:FILE BATCH=secondbatchfilename
:RUN REFORMAT.PUB.SYS
```

The outputfile now contains the reformatted data from the first and second batch files.

Using a User-Defined Command

You may want to combine the three :FILE commands required to run REFORMAT into one user-defined command (UDC). You enter a UDC through a text editor, as follows:

```
REFORMAT REFSPECS, DATA, OUTPUT=$STDLIST
FILE REFFILE=!REFSPECS
FILE BATCH=!DATA
FILE OUTFILE=!OUTPUT
FILE REFLIST;DEV=LP
FILE TESTLIST;DEV=LP
RUN REFORMAT.PUB.SYS
```

Use the SETCATALOG command to set the UDC.

The UDC definition is recorded by the MPE :SETCATALOG command. Now, all you need to do in order to run REFORMAT, is to enter the following command:

```
:REFORMAT reformatfilename, batchfilename, outputfilename
```

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 REFORMAT

You may also want to stream REFORMAT. You must first create a text file of the commands to run the job through a text editor, and then issue the MPE :STREAM command to actually run the program. For example:

```
!JOB USER.ACCOUNT
!FILE REFFILE=REF1
!FILE BATCH=BAT1
!FILE OUTFILE=OUT1
!FILE TESTLIST;DEV=LP
!FILE REFLIST;DEV=LP
!RUN REFORMAT.PUB.SYS
!EOJ
```

To stream this job, assuming you named the text file REFSTREM, use the following command:

```
:STREAM REFSTREM
```

In many cases, a streamed job is part of a group of streamed jobs. When REFORMAT is part of a series of program executions, you should precede the RUN REFORMAT command by a MPE :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
!RUN REFORMAT.PUB.SYS
!SPLGO MYPROG
!EOJ
.
.
.
```

A set of callable intrinsics is provided by VPLUS/V. Table 6-1 lists the VPLUS/V intrinsics in alphabetic order and summarizes their functions. A full description of each intrinsic appears later in this section. They can be used by any application, either for data entry or for other terminal related applications. The VPLUS/V intrinsics manage the interface between an application, the terminal, a forms file, the entered data, and, optionally, the batch file to which entered data may be written. These intrinsics are used by the VPLUS/V Data Entry Program (ENTRY) to control data entry.

Appendix A contains listings in different programming languages of sample applications. These listings provide useful examples of how to use the VPLUS/V intrinsics.

Multipurpose

The VPLUS/V intrinsics can be called by any application that displays forms on a terminal supported by VPLUS/V. With the exception of the batch file management intrinsics used specifically for data entry, the VPLUS/V intrinsics can be used in conjunction with FORMSPEC to allow for data base inquiries and updates via the application, as well as any other operation that results in the display or collection of data through a terminal form. Such applications can also make use of the capabilities that VPLUS/V provides for validating input data.

Multilanguage

The VPLUS/V intrinsics can be called from programs written in any of the supported programming languages listed in Appendix A. Additionally, some languages (also listed in Appendix A) provide the programmer with a special interface with terminals and forms, as described in their respective reference manuals. With these languages, the programmer does not call the VPLUS/V intrinsics directly. Instead, the programmer specifies the statements appropriate to the special interface. Each of the supported programming languages calls the VPLUS/V intrinsics with the same parameters, and these parameters are essentially the same type and size. So that the intrinsics can adjust to any peculiarities of the calling programming language, one input sub-parameter specifies the language of the calling program.

Error Detection

If a system or program error causes a VPLUS/V intrinsic to fail, an error code is returned to the calling program. Once an error has been detected, subsequent intrinsic 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 argument. This method of error handling means that intrinsic errors do not cause unexpected program termination.

Table 6-1. Summary of VPLUS/V Intrinsic

INTRINSIC	FUNCTION
VCHANGEFIELD	Changes field attributes for specified fields at run-time.
VCLOSEBATCH	Closes batch file.
VCLOSEFORMF	Closes forms file.
VCLOSETERM	Closes terminal file.
VERRMSG	Returns message associated with error code.
VFIELDDEDITS	Performs field phase processing specifications.
VFINISHFORM	Performs final phase processing specified for form.
VGETBUFFER	Copies contents of data buffer into application.
VGETFIELD	Copies field contents from data buffer into application.
VGETFIELDINFO	Returns field information.
VGETFILEINFO	Returns forms file information.
VGETFORMINFO	Returns form information.
VGETKEYLABELS	Returns global or form function key labels.
VGETLANG	Returns the native language ID of the forms file being executed.
VGETNEXTFORM	Reads next form into form definition area of memory; window and data buffer are not affected.
VGETtype	Copies field contents from data buffer to application, converting data to specified type.
VINITFORM	Sets data buffer to initial values for form.
VLOADFORMS	Loads the specified forms into terminal local form storage memory.
VOPENBATCH	Opens batch file for processing.
VOPENFORMF	Opens forms file for processing.
VOPENTERM	Opens terminal file for processing.
VPLACECURSOR	Positions the cursor at a specified field after a form is displayed.

Table 6-1. Summary of VPLUS/V Intrinsic, continued

INTRINSIC	FUNCTION
VPOSTBATCH	Updates end of file mark in batch file after last record referenced.
VPRINTFORM	Prints current form and data buffer on offline list device.
VPRINTSCREEN	Prints entire contents of screen on offline list device.
VPUTBUFFER	Copies data from application to data buffer.
VPUTFIELD	Copies data from application to field in data buffer.
VPUTtype	Copies data of specified type from application to data buffer, converting data to external format.
VPUTWINDOW	Copies message from application 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 copies error message to window area.
VSETKEYLABEL	Temporarily sets a new label for a function key.
VSETKEYLABELS	Temporarily sets new labels for function keys.
VSETLANG	Specifies the native language ID to be used with an international forms file.
VSHOWFORM	Updates terminal screen, merging the current form, any data in buffer, any key labels, and any message in window.
VUNLOADFORM	Unloads a specified form from terminal local form storage memory.
VWRITEBATCH	Writes data from data buffer to batch file.

HOW INTRINSICS ARE USED

The VPLUS/V intrinsic control the interface between forms stored in a forms file, a terminal screen, data entered on a form, and an application. Figure 6-1 illustrates the relation between the VPLUS/V intrinsic, terminal, forms file, application, data file or optional batch file, and the buffer areas in memory used by the intrinsic.

The intrinsic 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. user-defined function key labels also reside in a buffer area while they are active. There is a buffer area for global labels and another for form-specific labels. These buffer areas are managed by VPLUS/V and are in addition to any buffers the application may define.

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 a user.

Any form written to the terminal screen by VSHOWFORM already resides as a form image in memory. It may also be stored in terminal local form storage. The 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. The form image can be loaded from the forms file into local form storage by VLOADFORMS. Or the form image is read from the forms file into memory by a call to VGETNEXTFORM. Then it is written to the terminal screen or loaded into local form storage depending on options that can be set in VREADFIELDS or VSHOWFORM.

Associated with each form is a set of data specifications -- field attributes and processing specifications defined in FORMSPEC. These specifications are read from the forms file by VGETNEXTFORM and reside in memory during execution of the form. The field attributes can be dynamically altered using VCHANGEFIELD.

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 **ENTER** is pressed at the terminal, VREADFIELDS transfers the user-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 user-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.

An application can send data to the buffer with VPUTBUFFER or to a single field in the buffer with VPUTFIELD or VPUTtype. Conversely, the data buffer can be copied to an application with VGETBUFFER, or a single field can be copied with VGETFIELD or VGETtype.

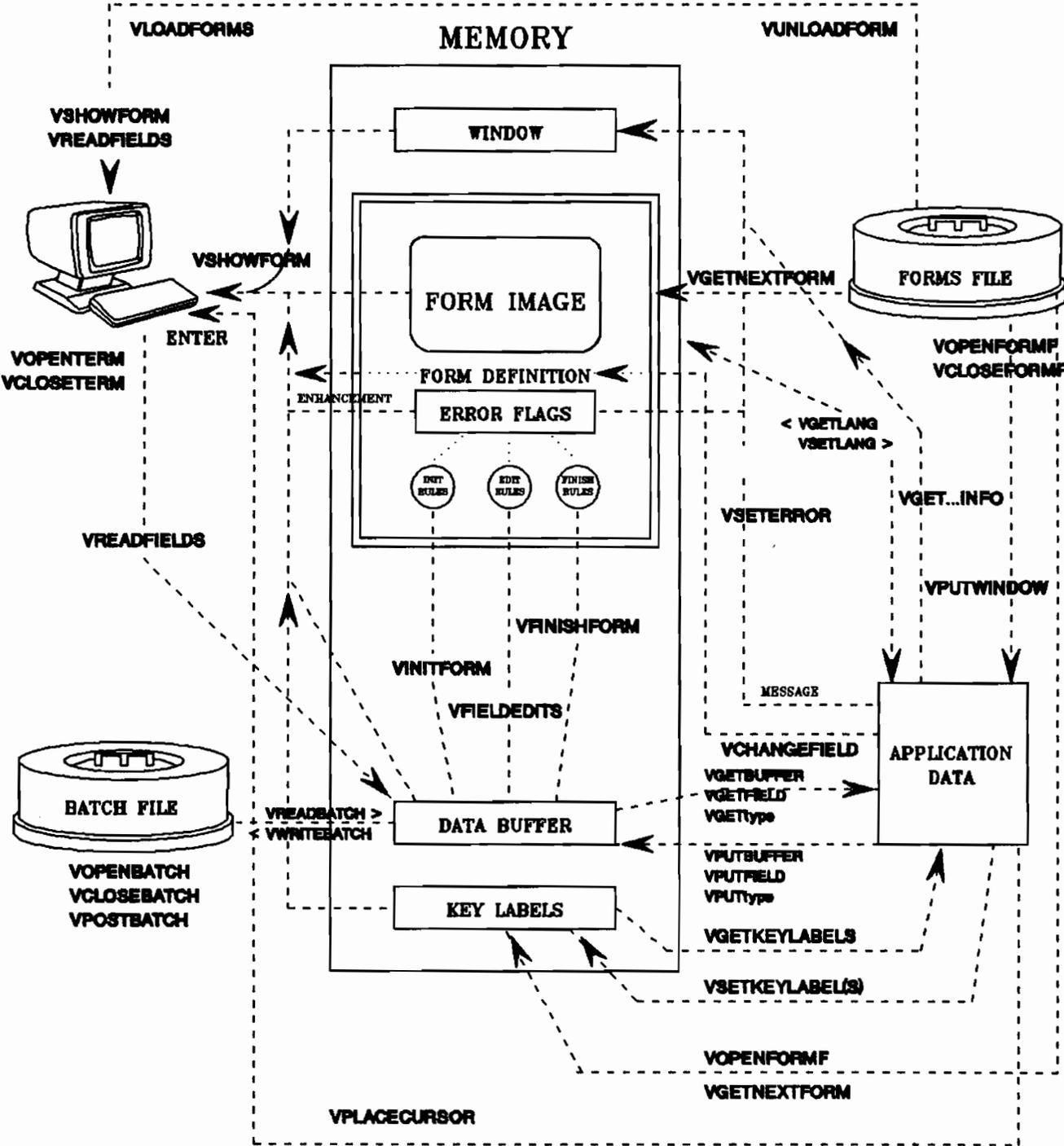


Figure 6-1. Operation of VPLUS/V Intrinsics

As is done in ENTRY, during data collection, data in the data buffer can be 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.

Window Area

If the field editing specifications detect an error in data entry or data movement, an error flag is set for the field. When called, VERRMSG returns the message associated with the error, and VPUTWINDOW puts the message into the window. If an error is detected by an application, VSETERROR may be used to set the error flag, and also to put any message associated with the error into the window. Any nonerror messages generated by an application are put in the window by VPUTWINDOW. A subsequent call to VSHOWFORM displays the contents of the window at the terminal.

Key Label Area

If there are user-defined function key labels, these labels are transferred to the key label area of memory. There is a key label area for global function key labels and another area for current form labels (global labels are associated with a forms file; form labels are associated only with the current form).

Labels may be defined with FORMSPEC. Alternately, VSETKEYLABEL or VSETKEYLABELS may be used to override existing labels with labels from the application. Global labels defined with FORMSPEC are transferred to the global key label area when the forms file is opened by VOPENFORMF. Form labels defined by FORMSPEC are transferred to the form key label area by VGETNEXTFORM. A global or current form label specified by VSETKEYLABEL is copied into the appropriate key label area from an application. VSETKEYLABELS functions in the same manner for multiple label definitions. Labels are displayed on HP terminals by a call to VSHOWFORM.

CALLING VPLUS/V INTRINSICS

The examples in table 6-2 show the format for calls to the VPLUS/V intrinsics from each language, where:

<i>intrinsicname</i>	Identifies the intrinsic being called.
<i>parameter</i>	At least one parameter is required for each intrinsic; the particular parameters are listed in the formats for the individual intrinsic descriptions.

Table 6-2. Examples of Intrinsic Call Formats for Each Language

Language	Intrinsic Call Format
COBOL	CALL " <i>intrinsicname</i> " USING <i>parameter1</i> [, <i>parameter2</i>]...
FORTRAN	CALL <i>intrinsicname</i> (<i>parameter1</i> [, <i>parameter2</i>]...)
BASIC	<i>label</i> CALL <i>intrinsicname</i> (<i>parameter1</i> [, <i>parameter2</i>]...)
PASCAL	<i>intrinsicname</i> (<i>parameter1</i> [, <i>parameter2</i>]...);
SPL	<i>intrinsicname</i> (<i>parameter1</i> [, <i>parameter2</i>]...);

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. The exception is VSETLANG, which has one parameter that is passed by value.
- 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 intrinsic call.
- Return type intrinsics are not allowed; any values returned by the intrinsic are sent to the *comarea* or to a passed parameter.

Parameter Types

The data types that are used in VPLUS/V intrinsics are shown in table 6-3. Note that not all types are allowed for all languages.

Table 6-3. Data Types Used for Various Languages

Data Type	COBOL	FORTRAN	BASIC	PASCAL	SPL
Character	DISPLAY PIC X(n)	Character	String	Packed Array of Char	Byte Array
Two-byte Integer	COMP PIC S9 thru PIC S9(4)	Integer*2	Integer	Subrange -32768..32767	Integer
Unsigned Two-byte Integer	COMP PIC 9 thru PIC 9(4)	Logical	Integer (with value <32767)	Subrange -32768..32767	Logical
Four-byte Integer	COMP PIC S9(5) thru PIC S9(9)	Integer*4	Integer Integer ¹	Integer	Double
Real Four-byte	--	Real	Real	Real	Real
Long 8-byte	--	Double Precision	Long	Longreal	Long

¹ 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 VPLUS/V parameters use only data types that are available in all programming languages: character, integer, logical, and double integer. The only exceptions are the transfer and conversion intrinsic, VPUTtype and VGETtype, which use real and long.

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 one to four digits.

COMMUNICATION AREA

Every application that calls VPLUS/V intrinsic must allocate a data area in the application for communication with the intrinsic. This area (called the *comarea*) is the first, and often the only, parameter in every call to a VPLUS/V intrinsic. Table 6-4 briefly outlines the contents of this communication area; Table 6-5 outlines the items used for data capture devices. It is essential to successful operation of your application that the *comarea* be defined exactly as shown in these tables; items that are "Reserved for system use" should not be set or used after they are initialized. The *comarea* items listed in table 6-4 and 6-5 are defined below.

Table 6-4. Outline of Communication Area Contents

Data Type	Position	Offset	Name	Function
Integer	1	0	<i>cstatus</i>	status, error returns
(Two-byte)	2	1	<i>language</i>	programming language of calling program
	3	2	<i>comarealen</i>	length of <i>comarea</i> in two-byte words
	4	3	<i>usrbuflen</i>	<i>comarea</i> extension length (BASIC)
	5	4	<i>cmode</i>	current mode (collect or browse)
	6	5	<i>lastkey</i>	code of last key pressed
	7	6	<i>numerrs</i>	number of errors in current form
	8	7	<i>windowenh</i>	code for window enhancement
	9	8	<i>multiusage</i>	Child of previous form
	10	9	<i>labeloption</i>	function key label indicator
Character Array	11-18	10-17	<i>cfname</i>	current form name (15 characters)
(16-byte)	19-26	18-25	<i>nfname</i>	next form name (15 characters)
Integer	27	26	<i>repeatapp</i>	current form flag (repeat/append)
(Two-byte)	28	27	<i>freezapp</i>	next form flag (freeze/append)
	29	28	<i>cfnumlines</i>	number of lines in current form
	30	29	<i>dbuflen</i>	data buffer length (in bytes)
Integer (Two-byte)	31	30	Reserved for system use.	--

Table 6-4. Outline of Communication Area Contents, continued

Data Type	Position	Offset	Name	Function
Logical (Two-byte)	32	31	<i>lookahead</i>	flag for background form load in VREADFIELDS
	33	32	<i>deleteflag</i>	delete current batch record
	34	33	<i>showcontrol</i>	control flags for VSHOWFORM
Integer (Two-byte)	35	34	Reserved for system use.	--
Integer (Two-byte)	36	35	<i>printfilnum</i>	MPE file number of forms file print file
	37	36	<i>filerrnum</i>	MPE file error number from FCHECK
	38	37	<i>errfilenum</i>	MPE file number, error message file
	39	38	<i>formstoresize</i>	number of forms in buffer
Integer (Two-byte)	40-42	39-41	Reserved for system use.	--
Integer (4-byte)	43-44	42-43	<i>numrecs</i>	number of records in batch file
	45-46	44-45	<i>recnum</i>	record # of current batch record
Integer (Two-byte)	47-48	46-47	Reserved for system use	--
Logical (Two-byte)	49	48	<i>filen</i>	MPE file number of terminal
Integer (Two-byte)	50-54	49-53	Reserved for system use	--
Logical (Two-byte)	55	54	<i>retries*</i>	max number of retries
	56	55	<i>termoptions*</i>	suppress msgs. and autoread
	57	56	<i>environ</i>	term environment: Sys LDEV
	58	57	<i>usertime*</i>	user defined time out period in seconds.

* Not supported on data capture devices.

Table 6-4. Outline of Communication Area Contents, continued

Data Type	Position	Offset	Name	Function
Logical (Two-byte)	59	58	<i>identifier</i>	type of terminal
	60	59	<i>labinfo</i>	number of function keys; length
Integer (Two-byte)	61-64	60-63	Reserved for system use	--
	65	64	<i>buffercontrol</i>	control ARB processing
	66	65	<i>bufferstatus</i>	indicate successful data conversion
	68-70	67-69	Reserved for system use	--
Refer to Table 6-5.	71-85	70-84	--	These <i>comarea</i> items are only referenced when using data capture devices.

Table 6-5. Communication Area Contents for Data Capture Devices

Data Type	Position	Offset	Name	Function
Integer (Two-byte)	71	70	<i>numflds</i>	number of fields on current form
	72	71	<i>splitpause</i>	length of pause (in seconds)
	73	72	<i>leftmodule</i>	type of terminal options
	74	73	<i>rightmodule</i>	type of terminal options
	75	74	<i>keyboard</i>	type of keyboard
	76	75	<i>display</i>	type of display
	77	76	<i>keyboardover</i>	whether to override or not
Character (Two-byte)	78	77	<i>errorlight</i>	error detection light
Logical (Two-byte)	79-80	78-79	<i>userlightson</i>	Lights on/off indicator
Integer (Two-byte)	81-85	80-84	Reserved for system use	--

NOTE

The *comarea* size required for each feature can be summarized as follows:

FEATURE	COMAREALEN
Data Capture Terminal	85
ARB Feature	70
All Others	60

The position of each item in *comarea* is given as a two-byte word offset in the table; for SPL programs the word offset starts with zero, for all other programs it starts with one.

The *comarea* must be at least 60 two-byte words long (120 bytes) unless you are working with an ARB. To take advantage of the ARB feature, the *comarea* must be at least 70 words long. For BASIC programs, the 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 VPLUS/V. Refer to Appendix E for more information on the DL area.

Before calling the first VPLUS/V intrinsic, you should initialize the entire *comarea* to binary zeroes. Do not change *comarea* values between calls except under documented conditions. Then, if you are coding in a programming language other than COBOL, you must set *language* to the code for the programming language you are using. If you are coding in BASIC, you must set *usrbuflen* to the number of two-byte words needed for the *comarea* extension. This value is displayed on the forms file listing generated by FORMSPEC. Finally, in order to provide for possible future extensions to *comarea*, set *comarealen* to its current total length.

Parameters

cstatus Two-byte integer to which the intrinsic status is returned. Set to zero if the call is successful; to a nonzero 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 Two-byte integer that indicates the programming language of the calling program:

- 0 = COBOL II/III
- 1 = BASIC
- 2 = FORTRAN 66
- 3 = SPL
- 5 = PASCAL/HP FORTRAN 77/HP BUSINESS BASIC

The *language* must be set by the calling program before any intrinsic is called.

comarealen

Two-byte integer that indicates the total length of *comarea* (use 60 for terminals; 85 for data capture devices). The *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 two-byte words to be appended to *comarea* for the form and data buffers and internal tables. (Note that the number of bytes required for this extension is printed when a form is listed through FORMSPEC.) The *usrbuflen* need not be set by other languages since VPLUS/V automatically uses the DL area for buffers and internal tables. The *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

If you want to save the form sequence for \$RETURN, you must set *cmode* to zero before calling VGETNEXTFORM.

lastkey

Two-byte integer set to a number between -256 and 26 (or -999) by VREADFIELDS to indicate user response, which can be **ENTER** or the last function key pressed at a standard terminal, the last field touched for a touch terminal, or the last key pressed for data capture devices. The value returned can be interpreted as follows:

For standard terminals, 0-8 is returned:

0 = **ENTER**
 1-8 = **F1** - **F8**

For touch, in addition to 0-8, -256 to -1 (or -999) is returned:

-1 to -256 = field number of touched field
 -999 = no field in area touched by user

For data capture devices, -1 to 26 is returned:

-1 = Attention key
 0 = enter
 1-26 = A-Z

numerrs

Two-byte integer set to the number of errors found when a form is edited according to FORMSPEC edit specifications.

windowenh

Two-byte integer in which the right byte contains a USASCII code for the window line enhancement. The code specifies any combination of enhancements according to table 6-6; zero indicates no enhancement. For example, to indicate Half-Bright, Inverse Video, *windowenh* is set to the character J.

Specifying an enhancement code with *windowenh* is independent, and in addition to, any color enhancement specified with FORMSPEC.

On data capture devices any enhancement except blinking is ignored. The entire display blinks, not just the field in error.

Table 6-6. Codes for the Window Line Enhancement.

	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Half-Bright									✓	✓	✓	✓	✓	✓	✓	✓
Underline					✓	✓	✓	✓					✓	✓	✓	✓
Inverse Video			✓	✓			✓	✓			✓	✓			✓	✓
Blinking		✓		✓		✓		✓		✓		✓		✓		✓
Stop enhancement	✓															

multiusage Two-byte integer that indicates whether the current form is in the same family as the previous form but is not the parent form.

- 0 = Not in the current family
- 1 = Child or sibling to previous form

labeloption Two-byte integer that indicates whether function key labels are to be used.

- 0 = Default function key labels are used (F1 - F8).
- 1 = User-defined labels are to be used

Space to store the labels defined either in FORMSPEC or specified with VSETKEYLABELS is allocated in the user stack by VOPENFORMF.

The *labeloption* must be set prior to a call to VOPENFORMF so that it can be determined whether or not to allocate space for user-defined function key labels.

cfname A 16-byte character array containing the name of the current form (15 characters). Updated by VGETNEXTFORM.

NOTE

All *comarea* entries start on two- or four-byte word boundaries. Thus, the 16-byte character array items *cfname* and *nfname* are each followed by one filler byte that is not part of the name.

nfname A 16-byte character array containing the name of the next form (15 characters). Required by VGETNEXTFORM and updated when necessary by VGETNEXTFORM. The *nfname* may be set by an application. The *nfname* may contain one of the following values:

- formname* Identifies the form to be read from the forms file.
- \$REFRESH Clear and reset the terminal. Set the internal flags to indicate to VSHOWFORM to redisplay the current form, window and data buffer. For terminals with local form storage, clear form storage memory and reinitialize the form storage directory.
- \$RETURN Retrieve previous different form; if current form is the head form, the current form is retrieved. There is a maximum of eight forms; after eight, each additional form replaces the most recent form.
- \$HEAD Retrieve first form displayed when the forms file is executed (the "head" form).

\$END For ENTRY only, terminate execution of VGETNEXTFORM; return to calling program without resetting any *comarea* items.

repeatapp

Two-byte 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

freezeapp

Two-byte 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. The *freezeapp* 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

Two-byte integer that specifies the number of lines in the current form. The *cfnumlines* is used by VPRINTF to print the form offline.

dbuflen

Two-byte integer that specifies the number of bytes 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.

lookahead

Two-byte integer that indicates whether or not look-ahead form loading is to occur when VREADFIELDS is called.

- 0 - ON The next form, as defined in FORMSPEC, is loaded before the current form is read if point-to-point data communications is being used. If multipoint data communications is being used, the next form is loaded after the current form is read. The least recently used form in the terminal's local form storage memory could be purged to make room for the new form if there is no room in the terminal or there are no available entries in the form storage directory.
- 1 - OFF No look-ahead form loading is performed, and no forms in the terminal's local form storage memory are purged (however, forms could still be purged if VSHOWFORM is called to preload forms; see the discussion of VSHOWFORM later in this section).

If *lookahead* is zero and a family member of the form specified as the next form is already in the terminal's local form storage memory, the new form is not loaded. Instead, the family member is changed into the required form when it is displayed. The *lookahead* feature is ignored if the terminal does not have form storage capability.

deleteflag

Two-byte logical flag that indicates whether the current batch record has been or is to be deleted. The *deleteflag* is returned by VREADBATCH and used by VWRITEBATCH:

FALSE (all zeros) = Current batch record not deleted
 TRUE (all ones) = Delete current batch record

showcontrol

Control options for VSHOWFORM or VREADFIELDS. These bits remain set until an application resets them. The following bits can be set:

- bit 15 = 1 Force form to be written to the terminal screen; overrides VPLUS/V optimization.
- 14 = 1 Force data and field enhancements to be written to the terminal screen; overrides VPLUS/V optimization.
- 13 = 1 Force window line to be written to the terminal screen; overrides VPLUS/V optimization.
- 12 = 0 Stop after displaying a form without fields (default); data capture devices only.
 - 1 Do not stop after displaying a form without fields; data capture devices only.
- 11 = 0 Do not put a right closing bracket (]) on all input fields (default); data capture devices only.
 - 1 Put a right closing bracket (]) on all input fields; data capture devices only.
- 10 = 0 Enable the keyboard for input (default).
 - 1 Do not enable the keyboard.
- 9 = 0 Do not preload form into the terminal's local form storage memory but write it directly to the screen (however, VSHOWFORM always checks to see if the form is in local storage already) (default).
 - 1 Preload form into the terminal's local form storage memory if it is not already there. Forms could be purged from local form storage if there is not room for the form (this could occur even when *lookahead* is 1).
- 8 = 0 Do not enable `AIDS`, `MODES`, or `USERKEYS` (default).
 - 1 Enable `AIDS`, `MODES`, and `USERKEYS`. These keys are enabled after a form is displayed on the screen and remain enabled until bit 8 is reset to 0.
- 7-1 = 0 The undefined bits (7-1) must be initialized to zero.

- 0 = 0 Do not enable the touch feature (default).
 1 Enable the touch feature.

Bits 15, 14, and 13 are used to override VSHOWFORM optimization (where only information that has changed since the last call to VSHOWFORM is written to the terminal screen). Bits 12 and 11 are used with VREADFIELDS on data capture devices. Bit 10 is used when consecutive calls are made to VSHOWFORM (no call to VREADFIELDS in between). Bit 9 is used with terminals having local form storage capability.

Bit 8 is used to allow access to the function control keys on the HP 239X and HP 262X terminals. For example, device control and device mode keys can be used for local printing to an integral printer. Enabling the function control keys using bit 8 is recommended for experienced users only. Misuse or accidental use of certain function control keys could yield undesirable results.

Bit 0 is used to activate touch available with terminals with the touch feature. VREADFIELDS returns *lastkey* with the field number of the touched field (as a negative number).

printfilnum

MPE file number to which a form is printed by VPRINTFORM. The formal designator is FORMLIST.

filerrnum

MPE file error number (FCHECK number) returned by VPLUS/V intrinsic 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.

formstoresize

Two-byte integer that indicates the number of entries allowed in the form storage directory.

- 1 No local form storage is to be performed. For the HP 2626A terminal the workspace configuration is under user control. VPLUS/V does not modify terminal workspace, window, or datacomm configuration, and either terminal port can be used.
- 0 No local form storage is to be performed. For the HP 2626A terminal VPLUS/V reconfigures the workspaces and windows to the terminal's default values, but only the first terminal port can be used.
- 1..255 One through 255 forms can be stored locally. For the HP 2626A terminal any value greater than four defaults to four (allowing an application to run on either the HP 2626A, HP 2394A, or HP 2624B terminals), where four indicates that one form is the displayed form and three forms are stored in the HP 2626A local forms storage. VPLUS/V reconfigures the HP 2626A terminal workspaces and windows to support local form storage, but only the first terminal port can be used.

The *formstoresize* allows the user to control the number of entries allowed in the form storage directory on the user stack and must be set prior to opening the terminal and the forms file. The space required is listed on the forms file listing generated by FORMSPEC.

When designing forms for local form storage terminals, keep in mind that large forms need a substantial amount of terminal display memory. For forms that are greater than 24 lines, so much terminal display memory may be used that insufficient terminal memory is available for form storage. If so, VPLUS/V resets *formstoresize* to zero and no local form storage is performed.

numrecs Four-byte integer that contains the number of nondeleted records in the current batch file. It is used by the V..*BATCH* intrinsic.

recnum Four-byte integer set to the current record number in the batch file. (Note that record numbers start with zero.) The *recnum* must be set by the program before writing to or reading from the batch file. It is used by the *VREADBATCH* and *VWRITEBATCH* intrinsic.

filen MPE file number used to identify the terminal.

retries Maximum number of retries.

- value = 0 Use default value (four retries)
- value > 0 Use this value as maximum
- value < 0 do not perform any retries

termoptions Terminal control options:

- | | |
|-----------|---|
| bits: 0-8 | reserved for system use |
| 9-10 | 01 = enables ENTER /function key timeout in <i>VREADFIELDS</i> .
11 or 00 = disables ENTER /function key timeout in <i>VREADFIELDS</i> . (default) |
| 11-12 | 10 = do not clear the screen at terminal open or close. |
| 13-14 | 01 = enables <i>AUTOREAD</i> in <i>VREADFIELDS</i> .
11 or 00 = disables <i>AUTOREAD</i> . (default) |
| | (<i>AUTOREAD</i> causes the terminal to do a programmatic enter instead of waiting for the user to press ENTER .) |
| 15 | 0 = Sound bell as usual (default).
1 = Suppress bell. |

NOTE

In programming languages which do not have the capability of setting single bits, programmers should add or subtract appropriate amounts to change single bits. For example, if *termoptions* was initialized entirely to zeros and the programmer wishes to enable the AUTOREAD, the following statement should be issued in COBOL:

ADD 2 TO TERM-OPTIONS.

Conversely, if disabling the AUTOREAD feature, the programmer should subtract 2 from *termoptions*. For more information on the AUTOREAD feature, please consult Appendix G.

<i>environ</i>	First byte is the logical device number of the terminal. The remaining byte is reserved for system use.
<i>usertime</i>	If enabled, the value in this position is used as the number of seconds to wait for either ENTER or a function key to be pressed. Consult Appendix G for more information regarding enabling user timeouts.
<i>identifier</i>	VPLUS/V identifier for the terminal model being used. See Appendix G for the interpretation of the contents of this word.
<i>labinfo</i>	First byte is the length of key labels (in bytes); Second byte is the number of key labels that VPLUS/V uses on the terminal.
<i>buffercontrol</i>	Controls the ARB options. The bit settings are as follows: Bit 15 (<i>ARB option</i>) 0 = return data "raw": no reordering or conversion. 1 = reorder and convert data according to ARB specifications. The remaining bits must be set to zero. Note that VGETBUFFER will not change these settings (see Section 6, VGETBUFFER, for special considerations).
<i>bufferstatus</i>	Return parameter indicating that VGET/PUTBUFFER performed the data conversion successfully (bit 15 in offset 65 will be set to 1).

NOTE

STATUS in the *COMAREA* will be non-zero if the conversion is not successful.

In addition to the above *comarea* descriptions, the data capture devices use the following items:

Parameters

splitpause

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 press key
- 0 - do not pause
- >0 - pause specified number of seconds



leftmodule

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

rightmodule

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

VPLUS/V does not communicate with the RS-232 interface (3) or the HP-IB interface (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

VPLUS/V does not support the terminal that has numeric display.

keyboardover

Two-byte 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).

errorlight

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.

lightsOn

Two two-byte 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:

TWO-BYTE WORD 1, a "1" in:

bit position 0	→	turns light of key "@"	ON
1		"A"	ON
2		"B"	ON
...		...	
14		"N"	ON
15		"O"	ON

TWO-BYTE WORD 2, a "1" in:

bit position 0	→	turns light of key "P"	ON
----------------	---	------------------------	----

The remaining bits are reserved for system use.

ERROR HANDLING

There are basically two types of errors that can occur as a result of calling VPLUS/V intrinsics. The first type consists of errors in the intrinsic call itself or in an attempt to access a file used by the called intrinsic. The second type are errors detected by editing data entered into FORMSPEC forms. These two error types are handled differently by VPLUS/V.

Intrinsic Call or File Errors

If a call to a VPLUS/V intrinsic causes an error so that the intrinsic cannot be executed correctly, or if an MPE file error occurs as a result of an attempt to access a file with a VPLUS/V intrinsic, the *comarea* word *cstatus* is set to a nonzero 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 VPLUS/V intrinsics called by the application return to the application without executing. As part of good programming practice, check *cstatus* after each call, report the error, and then reset *cstatus*. There is one exception -- VERRMSG uses *cstatus* to determine the error number. If *cstatus* is set to zero (indicating no error) prior to the call to VERRMSG, then no message is returned.

Editing Errors

Field processing may be specified in the forms description with FORMSPEC and checked by VFIELDEDITS, VINITFORM, or VFINISHFORM; or editing may be provided by user routines in an application.

Each field in a FORMSPEC form has an error flag associated with it. When one of the VPLUS/V intrinsics 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 VPLUS/V intrinsic VSETERROR in order to set the field error flag and increment *numerrs*. The *cstatus* item is not set when an editing error is detected and subsequent intrinsics may be executed without resetting this *comarea* item.

If new data is 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 intrinsics that can correct field errors and decrement *numerrs*. The intrinsic VREADFIELDS resets *numerrs* to zero when it reads new data into the buffer from the terminal. If *numerrs* has been set to a nonzero value by one of the VPLUS/V edit intrinsics, then VREADFIELDS with *reset* it to zero. Also, when VGETNEXTFORM is called, *numerrs* is reset to zero.

Error Messages

Messages associated with all VPLUS/V-detected errors can be retrieved by a call to VERRMSG. This intrinsic 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. Error messages may be general VPLUS/V messages (see Appendix B), or custom messages specified during forms design with 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 intrinsic 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.

Determining Fields in Error

VGETFORMINFO provides a method of determining which fields are flagged in error by the editing routines. VFIELDEDITS sets an error flag for each field that failed edit checks. Only the error number of the first field in error and the number of errors have been returned to an application by VFIELDEDITS via the *comarea*. All fields in error are enhanced with the error enhancement if VSHOWFORM is called after VFIELDEDITS. To enable applications to determine which fields are in error for a given form at run-time, VGETFORMINFO optionally returns data about these error flags to show which fields failed the edit checks.

USING TERMINAL FEATURES

VPLUS/V provides a way to take advantage of special terminal features, using processing specifications (refer to Section 4), VPLUS/V intrinsic, and selections on the FORMSPEC menus (refer to Section 3). Two of the special terminal features that can be controlled with VPLUS/V intrinsic are touch and local form storage, as described below. Refer to Appendix G for a list of terminals that support these features.

The Touch Feature

VPLUS/V supports the touch feature available with specified terminals, as listed in Appendix G. The touch feature is aimed to improve programmer productivity for the design of intuitive, friendly user interfaces.

To activate the touch feature, an application must set bit 0 of the *showcontrol* word in the *comarea* to 1 before calling VSHOWFORM. This feature will remain activated until the *showcontrol* bit 0 is set to 0, or until VCLOSETERM is called.

With the touch enhancement enabled, VREADFIELDS returns a number when a field on the form is touched. This number is the negative of the field number that was assigned by FORMSPEC at form design time. All fields as currently defined on a form return a negative of their assigned field number when touched. No additional definition of touch fields is necessary. The number returned is negative in order to distinguish it from the positive numbers returned for function keys. In this manner, fields on the form can be treated just like function keys.

Thus, touch applications can be designed using the same VPLUS/V intrinsic and the same forms file. The negative field number returned to the *lastkey* field in the *comarea* can then be interpreted for further processing.

Local Form Storage

Certain terminals allow forms to be stored locally in terminal memory, which can reduce the overhead of writing a form image from the form definition area of memory. Frequently displayed forms can be loaded into the terminal to be written directly to the screen. Only form images are loaded -- not the associated data. The HP 2626A terminal can store as many as four forms locally. The HP 2624B and HP 2394A terminal can store a maximum of 255 forms locally depending on the size of the forms and the size of terminal memory available.

The VPLUS/V intrinsic and communication area items activate and control local form storage. The same intrinsic and variables are used with local form storage terminals, but there are some differences as to how the form storage is handled. These differences, along with general information about using form storage, are described here. The specifics of the intrinsic involved are contained in the individual intrinsic descriptions later in this section.

The HP 2626A vs. the HP 2624B/HP 2394A

VPLUS/V utilizes the workspace/window feature of the HP 2626A terminal, configuring terminal memory into from one to four workspaces to store forms. Total available space is equal to 119 lines of forms, with the minimum size of a workspace being 26 lines. One full-screen window is used to display a form.

A form is displayed by attaching a workspace to the window. Any enhancements and data entered onto the form are part of the form as long as it is in the terminal -- a "fresh" form is not available in local storage. If a fresh copy of the form is to be displayed, it must be written from the form definition area of system memory.

One workspace is always attached to the window and displayed, even if there are no forms loaded. The workspace being displayed is frozen -- no form can be loaded into this frozen workspace. Therefore, only three forms can actually be loaded in a single call to `VLOADFORMS`. To get a fourth form into the terminal, display it in the window with `VSHOWFORM`.

Forms can be purged from the terminal if there is not room for the form you are trying to load. Forms are purged on a "least recently used" basis. During a load operation, each form is frozen until all forms have been loaded, thus preventing loading and purging in a single operation.

If you want to purge the currently displayed form from the terminal, it is not erased from the screen, but the form storage directory entry for this workspace is marked available. (The form storage directory is discussed later in this section.)

On the HP 2624B/HP 2394A terminals, forms stored locally are copied from terminal memory to the screen. A fresh copy of the form remains available in local storage. When the terminal is opened, enough memory is reserved to display the largest form in the forms file and this affects how many forms can actually be loaded.

Form Storage Directory

The *comarea* variable *formstoresize*, which must be set before opening the terminal and the forms file, indicates the maximum number of forms you are going to store locally. A form storage directory with an entry for each form is created on the user stack. This directory is used to keep track of loaded forms and of how much terminal memory each form is using. Note that setting this variable does not load any forms -- it just reserves space for them. `VSHOWFORM` searches the directory to determine if a form to be displayed is in local storage.

When a form is purged from local storage, the corresponding directory entry is marked available to indicate that the form is no longer in the terminal and is not taking up any terminal memory space.

NOTE

`VCLOSETERM` deallocates local form storage space -- both `VOPENTERM` and `VOPENFORMF` must be called if local form storage space is to be reallocated.

Loading Forms

Forms can be loaded into local form storage in the following ways:

- Preload the current form when `VSHOWFORM` is called by setting *showcontrol* bit 9 to one.
- Load the next form when `VREADFIELDS` is called by setting *lookahead* to zero.
- Call `VLOADFORMS`.

Using VPLUS/V Intrinsic

When VSHOWFORM is called to display a form, it searches the form storage directory to determine whether or not the form is already in the terminal.

If the form is in local storage, VSHOWFORM displays it on the screen. If the form is not in local storage, VSHOWFORM preloads it into the terminal before displaying it when bit 9 of the *comarea* variable *showcontrol* is set to one.

If there is not enough room in terminal memory for the form, the least recently used form (or forms) is purged from the terminal. If the form is not already in the terminal and is not preloaded into local form storage (bit 9 is zero), it is written to the screen from the form definition area of system memory.

(On the HP 2626A terminal, the form is written from memory when *showcontrol* bit 15 is set to force a write, no matter what the setting of *showcontrol* bit 9 or whether the form is already in the terminal.)

Look-ahead loading can be performed by setting the *comarea* variable *lookahead* to zero before the call to VREADFIELDS. The next form, named by *nfname* in VGETNEXTFORM, is loaded before or after the current form is read depending on the type of data communications being used. If point-to-point is being used, the next form is loaded before the current form is read. If multipoint is being used, the next form is loaded after the current form is read. The least recently used form (or forms) is purged from the terminal if there is not enough room to load the next form.

VREADFIELDS does not check whether or not the next form loads successfully. If the form does not load, the next call to VSHOWFORM retrieves it from the form definition area of system memory.

Forms can also be loaded into local form storage by a call to VLOADFORMS, which simply loads the forms named in the *forms* parameter. A form loaded by this intrinsic is thus already in the terminal when the call to VSHOWFORM is made (recall that VSHOWFORM always checks whether the form is in local storage). Forms are loaded in the order specified in the *forms* parameter as long as there is space in the terminal. Any remaining forms are ignored.

VLOADFORMS checks to see whether or not a form is loaded. During this check, the keyboard is locked briefly. To avoid possible loss of keystrokes and/or entered data do not call VLOADFORMS between VSHOWFORM and VREADFIELDS.

Form Families and Local Storage

VLOADFORMS loads forms strictly by form name so that multiple family members can be loaded at the same time. For look-ahead loading or preloading, to optimize performance, the form is not loaded if a related form is currently being displayed or is already in the terminal. The family member that is in local storage is changed into the required form and displayed by VSHOWFORM (unless *showcontrol* bit 15 is set to force a fresh copy of the form to be written).

Note that form family optimization displays any data associated with the required form. When optimization occurs on the HP 2626A terminal, the form storage directory is updated to reflect the name of the required form rather than the name of the family member that was used to make the change.

Appending Forms and Local Storage

The implementation of appending forms differs between the HP 2626A and the HP 2624B/HP 2394A terminals. On the HP 2626A, the appending form is always written to the screen from the form definition area of memory rather than being displayed from local storage. If, however, the appending

form has 24 or more lines (23 or more if a message window is defined), it is handled as an independent form rather than as an appending one. In this case, it can be displayed from local storage.

The workspace for the current form (the one being appended to) is marked available, so the current form cannot be displayed again from local storage.

Form family optimization occurs when the current form is an appended form and is related to the next form to be displayed. The current appended form is changed into the required form when it is displayed.

On the HP 2624B/HP 2394A terminals, the appending form can be copied from local storage.

Purging Forms From Local Storage

The VUNLOADFORM intrinsic is used to purge an unneeded form from local storage to make room for a new form. This intrinsic checks to see whether or not a form is purged. During this check, the keyboard is locked briefly. To avoid possible loss of keystrokes and/or entered data, do not call VUNLOADFORM between VSHOWFORM and VREADFIELDS. In addition, the intrinsics VCLOSEFORMF and VCLOSETERM also clear all forms from local storage.

With look-ahead loading, forms can be automatically purged from the terminal to make room for a new form. To protect forms from being purged from local storage, set *lookahead* to one before any other calls to VREADFIELDS or to VLOADFORMS. If *lookahead* is zero, the least recently used forms are purged if necessary. However, when *showcontrol* bit 9 is set to one, forms can be purged to make room even if look-ahead is not enabled.

INTRINSIC DESCRIPTIONS

The intrinsic on the following pages are described in alphabetic order for easy reference. However, this is not the order in which they are normally used. Table 6-7 is provided to group the intrinsic according to the functions they perform. Note that terminal access is performed by only seven intrinsic: VOPENTERM, VCLOSETERM, VPLACECURSOR, VSHOWFORM, VREADFIELDS, VLOADFORMS, and VUNLOADFORM. The remaining intrinsic interact with the form definition, data buffer, window, and key label areas of memory. An example of the order in which the intrinsic are used is shown in Figure 6-2.

Table 6-7. Intrinsic by Function Group

Function	Intrinsic	
Access to Terminal	VOPENTERM VCLOSETERM VSHOWFORM VPLACECURSOR	VREADFIELDS VLOADFORMS VUNLOADFORM
Access to Forms Definition	VOPENFORMF VGETFIELDINFO VGETFORMINFO VPRINTFORM	VCLOSEFORMF VGETFILEINFO VGETNEXTFORM VCHANGEFIELD
Data Processing	VINITFORM VFINISHFORM	VFIELDDEITS
Data Entry	VCLOSEBATCH VPOSTBATCH VWRITEBATCH	VOPENBATCH VREADBATCH
Programmatic Access to Data	VGETBUFFER VPUTBUFFER VGETtype	VGETFIELD VPUTFIELD VPUTtype
Access to Error/Status Window, Function Key Labels, Native Language ID	VERRMSG VGETKEYLABELS VSETKEYLABELS VGETLANG	VSETERROR VPUTWINDOW VSETKEYLABEL VSETLANG

All intrinsic require that three items in *comarea* (*cstatus*, *language* and *comarealen*) are set before the intrinsic is called. However, two of these items, *language* and *comarealen*, only need to be initially set, prior to the *first* intrinsic call, since they are not modified by any subsequent intrinsic. In contrast, *cstatus* should be reset before calling any intrinsic after an error occurs (except for the intrinsic VERRMSG that uses *cstatus* in retrieving the associated error message).

No running examples are provided with the intrinsic descriptions. Instead, Appendix A contains sample programs that use many of the intrinsic described in this section.

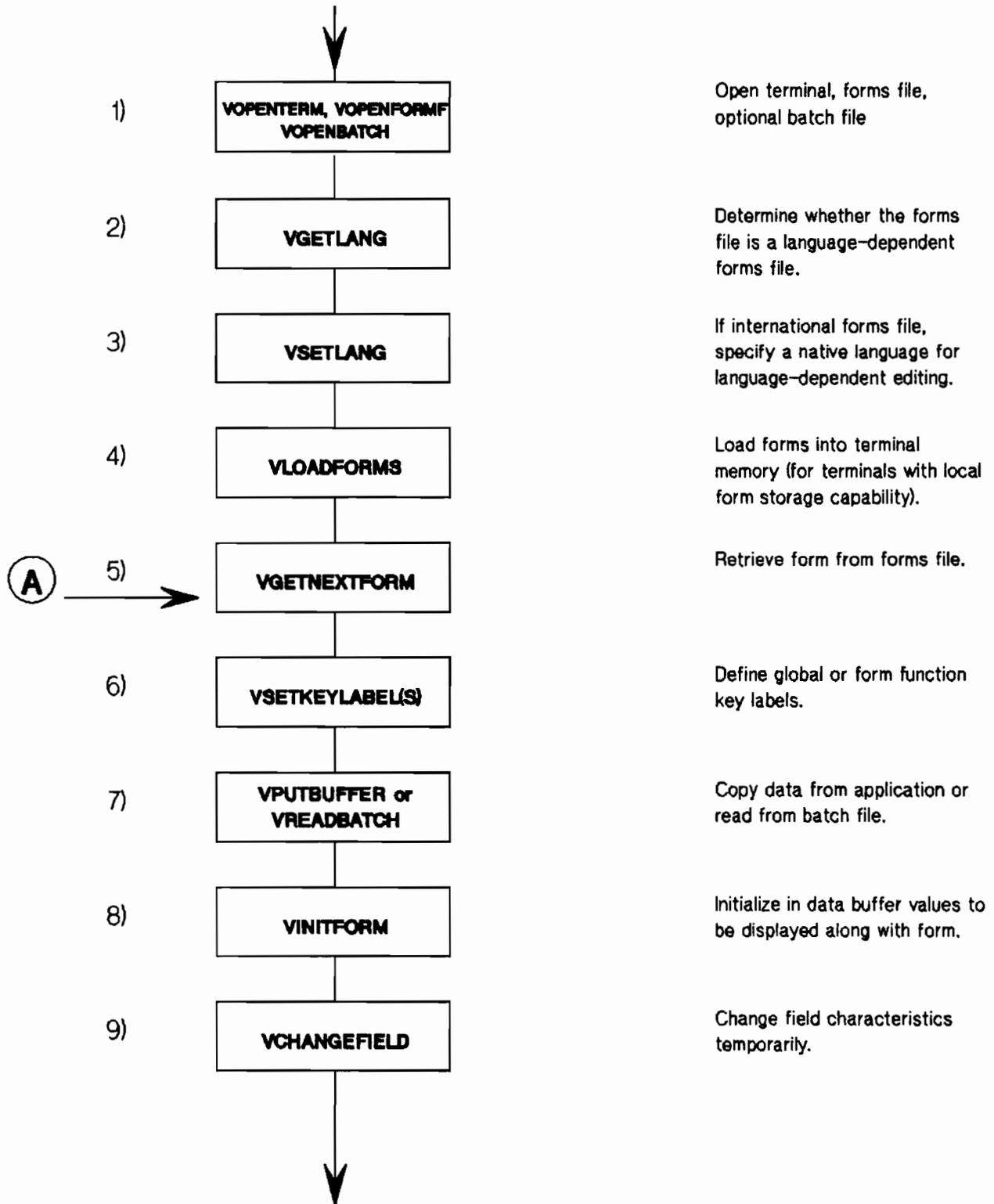


Figure 6-2. Intrinsic Flow

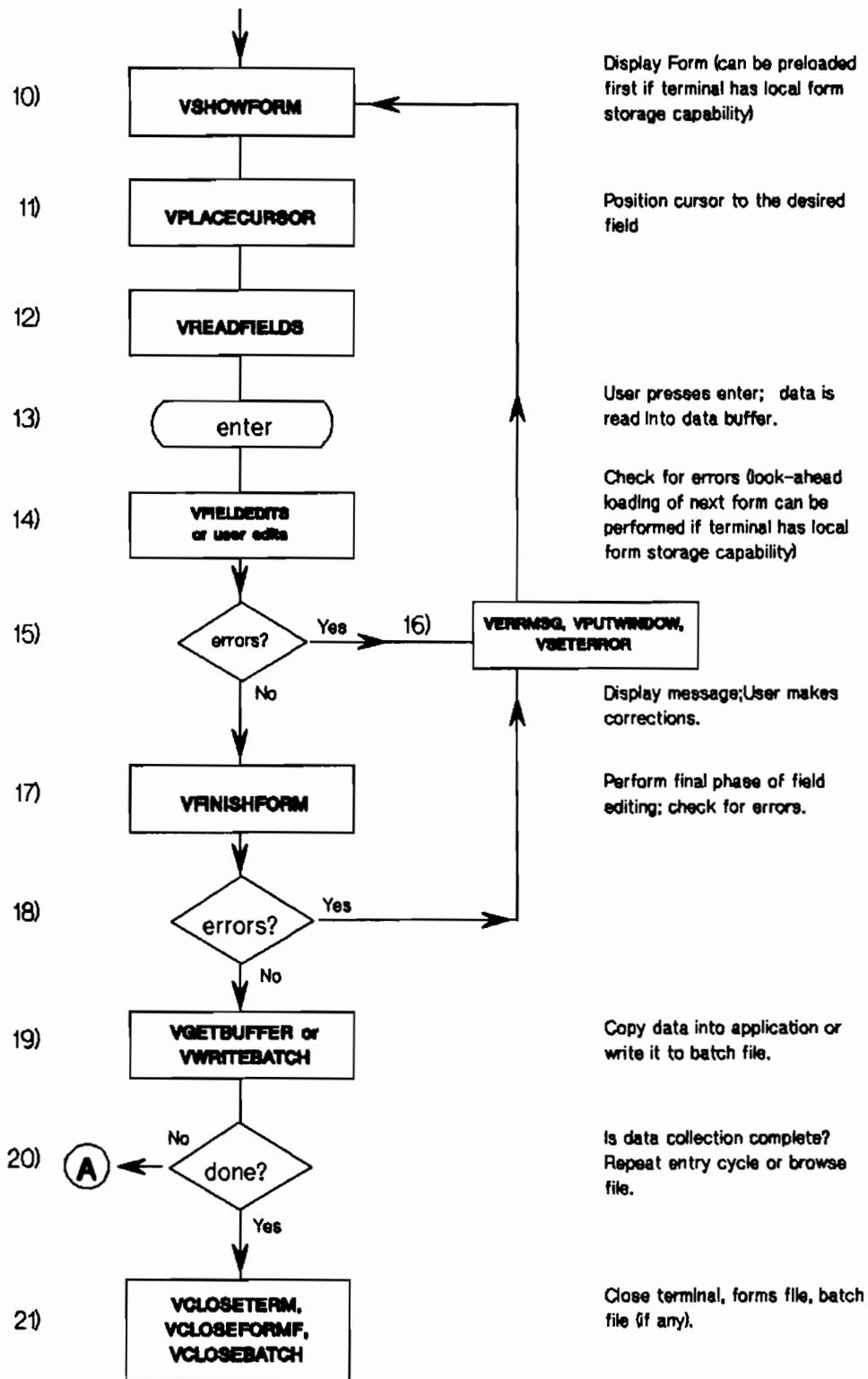


Figure 6-2. Intrinsic Flow (continued)

Dependency Between Intrinsic

Certain intrinsics must be called before other intrinsics can be executed. Figure 6-3 illustrates the standard dependencies among intrinsics. For example, the terminal file must be opened before any other intrinsics 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.

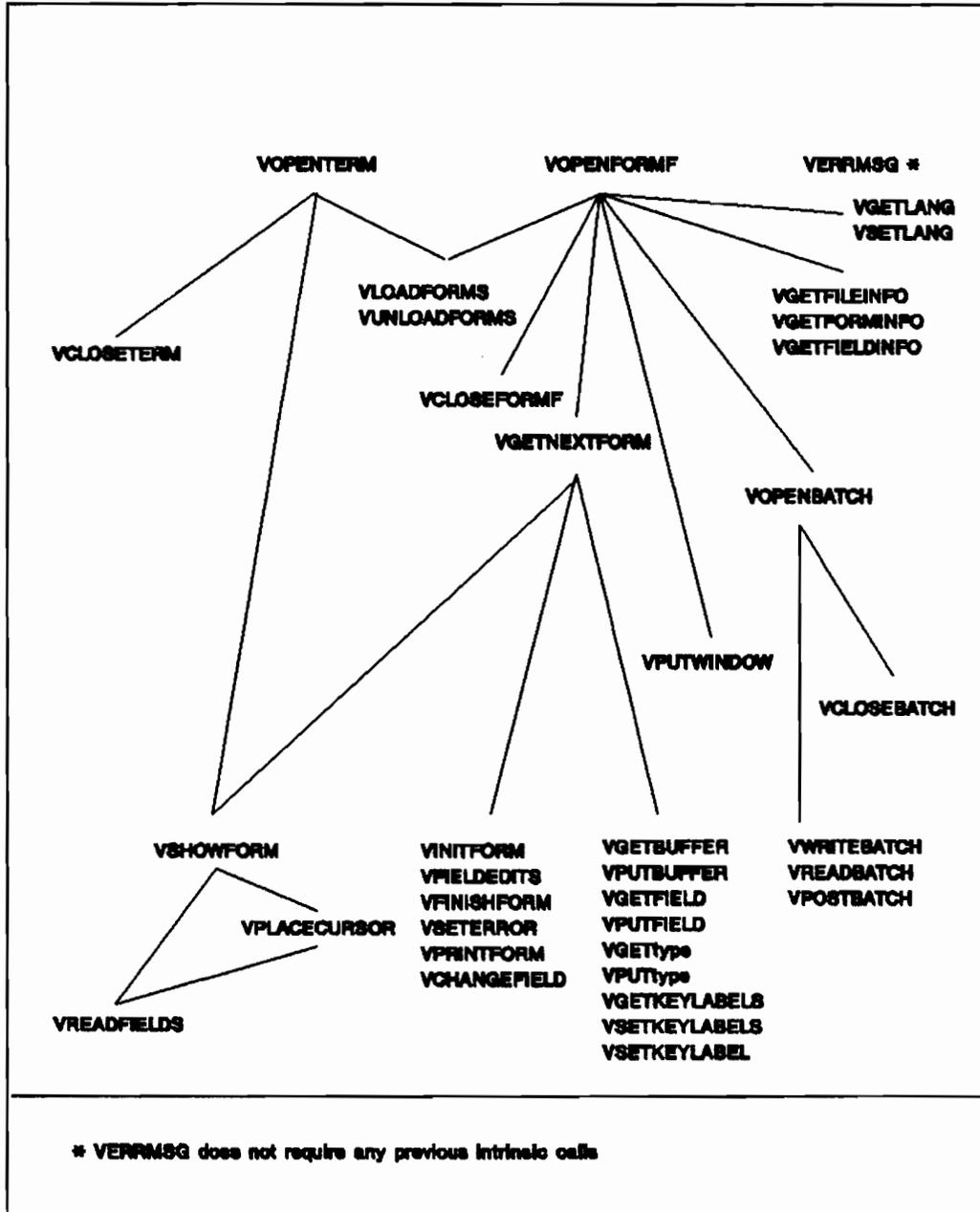


Figure 6-3. Intrinsic Dependencies

VCHANGEFIELD

Allows dynamic field attribute definition.

Syntax

```
VCHANGEFIELD {comarea,specbuffer,numentries}
```

Parameters

comarea Must be *comarea* specified when forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VCHANGEFIELD:

cstatus Set to zero.

language Set to code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VCHANGEFIELD may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

specbuffer A record; the specifications buffer is an array that provides VCHANGEFIELD with the information regarding the fields which are to be altered. The format of the array is as shown in table 6-8.

numentries A two-byte integer indicating the number of entries in the specifications buffer. The specifications buffer can contain up to three types of changes for each field in the form. That is, one call to VCHANGEFIELD can change the field type, data type and enhancement of every field in a form. For example, if a form contains four fields and an application is designed to make three changes to each field, then *numentries* should have a value of 12.

Discussion

VCHANGEFIELD alters the run-time copy of the current form. It does not modify the contents of the forms file. The next call to VGETNEXTFORM which actually retrieves a copy of the form from the forms file will reset the field specifications. The only exception to this is when two calls to VGETNEXTFORM are performed without resetting *nfname*; the second call will *not* retrieve the form definition from the forms file.

Please note that it is the responsibility of the programmer to ensure that a new data type is compatible with any initial values or processing specifications which may have been defined for the field.

VCHANGEFIELD

If the change type selected is a toggle type, (change type of 1, 2 or 3), the change specification buffer is returned with the "old" characteristics. Hence, the next VCHANGEFIELD call with the same buffer will toggle between the characteristics specified in the first VCHANGEFIELD call and the original characteristics. To toggle between two specific characteristics, VCHANGEFIELD can be used to set a field (change type of 4, 5, or 6) to certain starting characteristics without the toggle option at first.

Table 6-8. Specifications Buffer

Data Type	Position	Description
Integer (Two-byte)	1	positive field number or negative screen order number
Integer (Two-byte)	2	Change type 1 = toggle enhancement 2 = toggle field type 3 = toggle data type 4 = change enhancement 5 = change field type 6 = change data type
Character array (4-byte)	3-4	Change Specifications Type 1, 4 values = H, I, B, U, NONE Type 2, 5 values = O, D, P, R Type 3, 6 values = CHAR, DIG, IMP n , NUM $[n]$, DMY, MDY, YMD

VCHANGEFIELD has no effect on the security or color enhancements; an error is returned if the codes for these enhancements (1-8 or S) are specified. VCHANGEFIELD does not alter the field error enhancement nor is this enhancement code returned when the enhancement attribute is toggled (rather the current "normal" enhancement is altered and returned if toggling).

Example

```
COBOL  
CALL "VCHANGEFIELD" USING COMAREA,SPECBUF,ENTRIES.
```

```
BASIC  
CALL VCHANGEFIELD(C(*),B1,EN)
```

```
FORTRAN  
CALL VCHANGEFIELD(COMAREA,SPECBUF,ENTRIES);
```

```
SPL/PASCAL  
VCHANGEFIELD(COMAREA,SPECBUF,ENTRIES);
```

VCLOSEBATCH

Closes an open batch file.

Syntax

```
VCLOSEBATCH" {comarea}
```

Parameters

comarea Must be name of *comarea* specified in VOPENBATCH call that opened this batch file. If not already set, the following *comarea* items must be set before calling VCLOSEBATCH:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VCLOSEBATCH may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

filerrnum Set to file error code if MPE file error.

Discussion

The open batch file identified by *comarea* is closed when this intrinsic is called. The batch file must be closed before VCLOSEFORMF is called.

Example

```
COBOL  
CALL "VCLOSEBATCH" USING COMAREA.
```

```
BASIC  
200 CALL VCLOSEBATCH(C(*))
```

```
FORTRAN  
CALL VCLOSEBATCH(COMAREA)
```

```
SPL/PASCAL  
VCLOSEBATCH(COMAREA);
```

VCLOSEFORMF

Closes an open forms file.

Syntax

```
VCLOSEFORMF {comarea}
```

Parameters

comarea

Must be *comarea* specified when forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VCLOSEFORMF:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VCLOSEFORMF may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

filerrnum Set to file error code if MPE file error.

Discussion

The open forms file is closed when this intrinsic is executed. Once closed, the forms file is not available for further processing.

Example

```
COBOL  
CALL "VCLOSEFORMF" USING COMAREA.
```

```
BASIC  
200 CALL VCLOSEFORMF(C(*))
```

```
FORTRAN  
CALL VCLOSEFORMF(COMAREA)
```

```
SPL/PASCAL  
VCLOSEFORMF(COMAREA);
```

VCLOSETERM

Closes an open terminal file.

Syntax

```
VCLOSETERM {comarea}
```

Parameters

comarea Must be *comarea* named when file was opened by VOPENTERM. If not already set, the following *comarea* items must be set before calling VCLOSETERM:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VCLOSETERM may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

filerrnum Set to file error code if MPE file error.

Discussion

The terminal file opened with the specified *comarea* is closed when this intrinsic is executed. For terminals with this feature, the local forms storage of the terminal is cleared.

Example

```
COBOL  
CALL "VCLOSETERM" USING COMAREA.
```

```
BASIC  
200 CALL VCLOSETERM(C(*))
```

```
FORTRAN  
CALL VCLOSETERM(COMAREA)
```

```
SPL/PASCAL  
VCLOSETERM(COMAREA);
```

Returns a message corresponding to the error number of an edit error or an intrinsic call error.

Syntax

```
VERRMSG {comarea,buffer,buflen,actualen}
```



Parameters

comarea Must be *comarea* named when forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VERRMSG:

language Set to code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

errfilenum Contains MPE file number of VPLUS/V error message file; should be initialized to zero so that VERRMSG can open the error message file.

(Note that *cstatus* must not be cleared before calling this intrinsic.)

VERRMSG may set the following *comarea* items:

errfilenum If initialized to zero, VERRMSG opens VPLUS/V error message file and sets *errfilenum* to MPE file number of the opened file.

buffer Character string in an application to which message is returned by VERRMSG. In order to contain the message, this buffer must be defined as at least 72 bytes long.

buflen Two-byte integer variable set by an application to the length of the buffer.

actualen Two-byte integer to which VERRMSG returns the number of bytes in the message sent to buffer.

Discussion

If an error occurs in an intrinsic call or is detected by a VPLUS/V edit, a call to VERRMSG returns the message associated with the error. For an intrinsic call error, *cstatus* is set to a nonzero value. If *cstatus* indicates an error, VERRMSG returns the text explaining the type and cause of the error. If VINITFORM, VFIELDDEDITS, or VFINISHFORM detects a data error, *cstatus* is set to zero and *numerrs* is set to a nonzero value. If *numerrs* is set, VERRMSG returns a custom error message if there

VERRMSG

is one; otherwise, it returns the VPLUS/V error message associated with the edit error number in the VPLUS/V error message file. The error message, custom or VPLUS/V, is returned for the first field flagged in error.

The message describing the error is returned to an application 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.

The *errfilenum* contains the MPE file number of the VPLUS/V error message file. This file contains the error numbers and their associated messages for both intrinsic 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 when VERRMSG is called; it then sets *errfilenum* to the MPE file number of the file. The file is closed automatically when an application terminates.

If the VPLUS/V error message file cannot be opened and read by this intrinsic, default messages are generated. If *cstatus* is not equal to zero, the message is:

```
"VPLUS Error, COM'STATUS is nnn".
```

If *cstatus* is zero and *numerrs* is greater than zero, the message is:

```
"VPLUS Edit Error nnn".
```

Example

```
COBOL  
CALL "VERRMSG" USING COMAREA, BUFFER, BUFLen, ACTUALen.
```

```
BASIC  
225 CALL VERRMSG(C(*),M$,L,M)
```

```
FORTRAN  
CALL VERRMSG(COMAREA,BUFFER,BUFLen,ACTUALen)
```

```
SPL/PASCAL  
VERRMSG(COMAREA,BUFFER,BUFLen,ACTUALen);
```

Assume that *filerrnum* contains an MPE file error code. The calls to VERRMSG shown above return to your application a message describing the error and the length of this message in bytes.

VFIELDDEDITS

Edits data entered in each field of form and, if indicated, modifies data in the data buffer. If necessary, sets error flags.

Syntax

```
VFIELDDEDITS {comarea}
```

Parameters

comarea

Must be *comarea* name specified when forms file was opened with VOPENFORMF. If not set already, the following *comarea* items must be set before calling VFIELDDEDITS:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VFIELDDEDITS may set the following *comarea* items:

numerrs Set to total number of fields in which errors were detected.

cstatus Set to nonzero value if call unsuccessful.

nfname Set to new next form name if name changed by processing specifications.

repeatapp Set to new current form code if code changed by processing specifications.

freezapp Set to new next form code if code changed by processing specifications.

Discussion

This intrinsic checks the data content of each field. It checks that the data type and field type are correct, and if any special processing specifications were defined for this field in the Field Edit phase, it checks that the data conforms to these specifications. If any data formatting or data movement was specified for a field, VFIELDDEDITS performs these functions on the data in the data buffer.

VFIELDDEDITS

For each field that does not pass the edit checks, VFIELDDEDITS sets an error flag. These error flags are used by the VPLUS/V intrinsics, as shown in Table 6-9. In addition, these error flags can be accessed by an application with a call to VGETFORMINFO, which retrieves a table corresponding to the current error flag settings.

Table 6-9. Actions Used by Intrinsics

Intrinsic	Action
VGETNEXTFORM	Initializes all the error flags to zero.
VFIELDDEDITS VFINISHFORM VINITFORM VSETERROR	Each may set field error flags on, increment <i>numerrs</i> , 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 the form is displayed.
VPUTBUFFER VPUTFIELD VPUTtype	Each may clear field error flags and decrement <i>numerrs</i> if new data is entered into a field with an error.

After setting error flags for all fields with errors, VFIELDDEDITS saves the error number of 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.) VFIELDDEDITS sets *numerrs* to the total number of fields in which errors were found.

If requested by a call to VERRMSG, the text associated with the error number in the VPLUS/V error message file, or the associated custom error message, is returned to an application. If requested by a call to VPUTWINDOW, the message is copied to the window area of memory. Then, a call to VSHOWFORM can be used to display this message on the terminal screen and enhance the fields with errors.

Example

```
COBOL  
CALL "VFIELDDEDITS" USING COMAREA.
```

```
BASIC  
150 CALL VFIELDDEDITS(C(*))
```

```
FORTRAN  
CALL VFIELDDEDITS(COMAREA)
```

```
SPL/PASCAL  
VFIELDDEDITS(COMAREA);
```

VFINISHFORM

Performs any processing specifications defined for the final phase of fields editing.

Syntax

```
VFINISHFORM {comarea}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VFINISHFORM:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VFINISHFORM may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

numerrs Set to total number of fields in the form in which errors 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.

Discussion

All special processing defined as part of the "finish" phase of field editing is performed by this intrinsic. 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 4.)

Like VFIELDDEDITS, VFINISHFORM sets an error flag for each field that has an error as a result of processing the form. (Refer to the VFIELDDEDITS discussion.)

VFINISHFORM

Example

COBOL
CALL "VFINISHFORM" USING COMAREA.

BASIC
160 CALL VFINISHFORM(C(*))

FORTRAN
CALL VFINISHFORM(COMAREA)

SPL/PASCAL
VFINISHFORM(COMAREA);

The examples above perform all finish operations on a form.

Copies entire contents of data buffer from memory to an application.

Syntax

```
VGETBUFFER {comarea,buffer,buflen}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETBUFFER:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*. Must be at least 70 words in length if the ARB feature is used.

buffercontrol Set bit 15 to 1 to indicate that data is to be transformed according to the ARB specifications.

VGETBUFFER may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

bufferstatus Bit 15 set to 1 if data conversion successful.

buffer

Character string in an application to which the data in the data buffer is copied. Could also be a record describing the ARB, i.e., non-CHAR data.

buflen

Two-byte integer variable that specifies the number of bytes to be transferred to the user buffer.

Discussion

This intrinsic transfers data from the data buffer in memory to the area in an application specified by the *buffer* parameter. The data includes everything in the unprotected and display-only fields on a form. Previously, data in the data buffer was stored in the order of the fields on the form, and specific fields could be moved from the buffer with VGETFIELD or VGETtype. Now, the Application-ready Buffer (ARB) allows you to specify, using the FORMSPEC ARB feature, the order in which the application should receive the fields in the buffer, and the data type conversion to be performed on each (see the discussion on creating an ARB in Section 3.) You need no longer use VGETtype to convert individual fields; a call to VGETBUFFER accomplishes the task.

VGETBUFFER

The number of bytes moved from the data buffer is based on the number of bytes specified in the *buflen* parameter, or by the *dbuflen* in the *comarea* (refer to Table 6-5), whichever is less. The *dbuflen* item is set by VGETNEXTFORM when the current form is read into memory. For example, if there are 20 bytes in the data buffer (*dbuflen* is 20), and the user requests 50 bytes in the *buflen* parameter, only 20 bytes are transferred. Conversely, if the user requests 10 bytes through the *buflen* parameter, and *dbuflen* is 20, only 10 bytes are transferred.

Special Considerations

Designers using the ARB feature in VGETBUFFER should be aware that damaging run-time errors could occur if the application is inadvertently run on a system that has a VPLUS/V version earlier than B.05.00.

To prevent this, the designer should do three things:

1. Document the product with a clear warning that VPLUS/V version B.05.00 or later **MUST** be used.
2. Use the VPLUS/V intrinsic HP32209 in the code. This intrinsic checks to make sure that you are using the proper VPLUS/V version. If not, the application should terminate with an appropriate message.
3. Check offset 65 (*bufferstatus*) in the *comarea* on return from VGETBUFFER. Bit 15 will be set to 1 if VGETBUFFER performed the conversion successfully. In other words, the application must check both *status* and *bufferstatus* to be sure that the data was correctly converted.

Example

```
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 C
350 CALL VGETBUFFER(C(*),D$,L1)
```

```
FORTRAN
CALL VGETBUFFER(COMAREA,D1,DBFLEN)
```

```
SPL
BYTE ARRAY D1(0:36);
.
VGETBUFFER(COMAREA,D1,DBUFLEN);
```

These examples transfer the contents of the data buffer in memory to an application. The current value of *dbuflen* is specified as the user buffer length.

Copies contents of specified field from data buffer in memory to an application.

Syntax

```
VGETFIELD {comarea,fieldnum,fieldbuf,buflen,actualen,nextfldnum}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETFIELD:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VGETFIELD may set the following *comarea* items:

cstatus Set to nonzero if call unsuccessful, or if requested field has an error, or if *fieldnum* is unacceptable.

fieldnum Two-byte integer variable containing the number assigned to the field by FORMSPEC.

fieldbuf Character string in an application to which data entered in specified field is copied.

buflen Two-byte integer variable that specifies the number of bytes in *fieldbuf*.

actualen Two-byte integer to which VGETFIELD returns the number of bytes actually moved to *fieldbuf*.

nextfldnum Two-byte integer to which VGETFIELD returns the number of the next field in screen order. If there are no more fields, zero is returned. If *fieldnum* was set to zero or a negative number by an application, this is an error. In this case, VGETFIELD returns the number of the first field in screen order in *nextfieldnum*.

VGETFIELD

Discussion

VGETFIELD transfers the contents of the field specified by *fieldnum* from the data buffer to a variable in an application. This is in contrast to VGETBUFFER which retrieves data according to the current field layout.

When considering what is transferred by VGETFIELD, keep in mind that all the fields defined for a particular form in FORMSPEC are assigned numbers. The number assigned to a field by FORMSPEC does not change regardless of any changes to the field's position in the form or to its length and does not necessarily correspond to the screen order. The field numbers on a form can only be changed with the batch command, RENUMBER, as described in Section 7. The field number must not be confused with the field's position in the data buffer, which corresponds to its position in the form according to screen order, not assigned field number.

If the number of bytes specified by *buflen* is less than the field size, the rightmost bytes 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 bytes transferred to *fieldbuf*, the user buffer; *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.

Example

```
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.
CALL "VGETFIELD" USING COMAREA, FIELD-NUM, PART-DESCR OF ORDER-ENTRY,
ITEM-LEN, ACTUAL-LENGTH, NEXT-FIELD.
```

VGETFIELD

```
BASIC
350 F1=2
355 L1=12
360 CALL VGETFIELD(C(*),F1,P$,L1,A1,N1)
```

```
FORTRAN
FIELD=2
LEN=12
CALL VGETFIELD(COMAREA, FIELD, PARTDES, LEN, LENFLD, NXTFLD)
```

```
SPL/PASCAL
INTEGER
FIELD,
LEN;
BYTE ARRAY PARTDES(0:11);
:
FIELD:=2;
LEN:=12;
VGETFIELD(COMAREA, FIELD, PARTDES, LEN, ACTUAL'LEN, NEXT'FLD);
```

Assume that the contents of field number "2" is to be copied, and that the length of this field is 12 bytes. The calls shown above copy the contents of this field into a variable in an application.

VGETFIELDINFO

Returns information about specified fields to an application.

Syntax

```
VGETFIELDINFO {comarea,infobuf,infobuflen}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETFIELDINFO:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VGETFIELDINFO may set the following *comarea* item:

cstatus Set to nonzero value if call unsuccessful.

infobuf

A record through which you pass the request for field information and to which the intrinsic returns the specified information. This parameter must be initialized to spaces before filling in your request and calling VGETFIELDINFO. The layout of *infobuf* is shown in table 6-10. The three required fields of *infobuf* pass user-supplied parameters to VGETFIELDINFO as follows:

numofentries Specifies how many fields you want information about.

entrylength Indicates how many two-byte words of information (maximum 17 as shown in table 6-10) you want about each field.

formname The name of the form that contains the fields you are inquiring about.

The rest of *infobuf* (beginning with position 11) must consist of *entrylength* number of words for each field you want information about. Thus, the total length of *infobuf* in words must be as follows:

$$10 + (\textit{entrylength} * \textit{numofentries})$$

You may pass one or more of the three permissible keys, which must be passed in the position indicated in the table. Remember that *infobuf* must be initialized to spaces before filling in the parameters and any key.

infobuflen

Two-byte integer variable set to the number of two-byte words in *infobuf*.

Discussion

This intrinsic accesses an internal table and places information about one or more fields into *infobuf*. You tell VGETFIELDINFO how many fields, how much information about each field, and the name of the form containing the fields.

Table 6-10. Field Information Buffer

Data Type	Position	Contents	Comments
Integer (Two-byte)	1	<i>numofentries</i>	Required
	2	<i>entrylength</i>	Required
Character Array (16-byte)	3-10	<i>formname</i>	Required; second byte of position 10 is unused
	11-18	Field name	Permissible key; last byte of position 18 is unused
Integer (Two-byte)	19	Field number according to screen order.	Permissible key
	20	Field number	Permissible key; in order of creation
	21	Field length	In bytes
	22	Position of field in data buffer	In bytes, offset from zero
Character Array (4-byte)	23-24	Field enhancement	Combination of I, H, U, B, 1-8, or NONE.
	25-26	Data type of field	May be CHAR, DIG, IMP <i>n</i> , date (MDY, DMY, YMD), or NUM[<i>n</i>].
Character (Two-byte)	27	Field type	First byte of position 27; may be O, R, P or D.

Passing INFOBUF Without Entering Keys

Keys are optional. If you do not supply a key to indicate the first field you want information about, VGETFIELDINFO starts with the first field not already reported on by the current call.

As an example, suppose you want information about 10 fields, and you enter screen order numbers 3, 5, and 6 as keys. VGETFIELDINFO returns information about fields 3, 5, 6, and 7 through 13. If you enter no keys, VGETFIELDINFO returns information about fields 1 through 10.

VGETFIELDINFO

Table Wraparound

VGETFIELDINFO starts over with the first field in the form if you request information about more fields than are in the form or, if when you pass a key, your request goes beyond the end of the form.

Suppose a form has 12 fields. If you request information about 10 of the fields and enter screen order number 8 as a key, VGETFIELDINFO returns information about fields 8 through 12, and then goes to the beginning of the form and returns information about fields 1 through 5.

Example

```
COBOL
DATA DIVISION.
  .
WORKING-STORAGE SECTION.
01  INFOBUF.
    05  NUMBER-OF-ENTRIES          PIC S9(4) COMP.
    05  ENTRY-LENGTH              PIC S9(4) COMP.
    05  FORM-NAME                 PIC X(15).
    05  FILLER                    PIC X.
    05  ENTRY-TABLE OCCURS 10 TIMES.
        10  FIELD-NAME            PIC X(15).
        10  FILLER                PIC X.
        10  SCREEN-ORD-NUM        PIC S9(4) COMP.
        10  FIELD-NUM             PIC S9(4) COMP.
        10  FIELD-LENGTH          PIC S9(4) COMP.
        10  FIELD-POSITION        PIC S9(4) COMP.
        10  FIELD-ENHANCE         PIC X(4).
01  INFOBUFLEN                    PIC S9(4) COMP VALUE 80.
  .
PROCEDURE DIVISION.
  .
  MOVE SPACES TO INFOBUF.
  .
  MOVE 5 TO NUMBER-OF-ENTRIES.
  MOVE 14 TO ENTRY-LENGTH.
  MOVE "FORM1" TO FORM-NAME.
  MOVE 2 TO SCREEN-ORD-NUM(1).
  CALL "VGETFIELDINFO" USING COMAREA, INFOBUF, INFOBUFLEN.
```

The example shown above illustrates the data declaration of *infobuf* and *infobuflen* and the passing of parameters to VGETFIELDINFO. Note that before the intrinsic is called, *infobuf* is initialized to spaces (*not* zeros). The intrinsic copies 14 two-byte words of information about fields 2 through 6 of form FORM1 into *infobuf*.

Returns information about forms file to an application.

Syntax

```
VGETFILEINFO {comarea,infobuf,infobuflen}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETFILEINFO:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VGETFILEINFO may set the following *comarea* item:

cstatus Set to nonzero value if call unsuccessful.

infobuf

A record through which you pass the request for forms file information and to which the intrinsic returns the specified information. This parameter must be initialized to spaces before filling in your request and calling VGETFILEINFO. The layout of *infobuf* is shown in table 6-11. The two required fields of *infobuf* pass user-supplied parameters to VGETFILEINFO as follows:

numofentries Specifies how many times you want the information repeated (information is returned only about the forms file associated with *comarea*; if you specify a number greater than one, you get duplicate information).

entrylength Indicates how many two-byte words of information (maximum 19 as shown in table 6-11) you want about the file.

Remember that *infobuf* must be initialized to spaces before filling in the parameters.

infobuflen

Two-byte integer variable set to number of two-byte words in *infobuf*.

VGETFILEINFO

Discussion

This intrinsic accesses an internal table and places information about the open forms file into *infobuf*. You tell VGETFILEINFO how much information you want about the file.

Table 6-11. File Information Buffer

Data Type	Position	Contents	Comments
Integer	1	<i>numofentries</i>	Required
(Two-byte)	2	<i>entrylength</i>	Required
Integer Array (Four-byte)	3-4	File version number	A date/time stamp, recorded at last forms file compile.
Integer	5	Number of forms in file	In any one form Of any one form, plus up to one additional two-byte word.
(Two-byte)	6	Maximum number of fields	
	7	Maximum data buffer size in two-byte words	
	8	Number of save fields in file	
Character Array	9-16	Name of head form in file	Last byte of position 16 is unused.
	17-18	Global error enhancement	Combination of I, H, U, B, 1-8, or NONE.
	19-20	Global window enhancement	Combination of I, H, U, B, 1-8, or NONE.
Integer (Two-byte)	21	Position of window	Line number, 0-24. The line closest to the top of the form is 1; zero (0) means no window.

Example



```

COBOL
DATA DIVISION.
    .
WORKING-STORAGE SECTION.
01  INFOBUF.
    05  NUMBER-OF-ENTRIES                PIC S9(4) COMP.
    05  ENTRY-LENGTH                    PIC S9(4) COMP.
    05  ENTRY-TABLE OCCURS 1 TIMES.
        10  FILE-VERSION                PIC S9(8) COMP.
        10  NUMBER-OF-FORMS            PIC S9(4) COMP.
        10  MAX-FIELDS                  PIC S9(4) COMP.
        10  MAX-BUFSIZE                 PIC S9(4) COMP.
        10  SAVE-FIELDS                 PIC S9(4) COMP.
        10  HEAD-FORM                   PIC X(15).
        10  FILLER                      PIC X.
        10  ERROR-ENHANCE               PIC X(4).
        10  WINDOW-ENHANCE              PIC X(4).
        10  WINDOW-POSITION             PIC S9(4) COMP.
01  INFOBUFLN                            PIC S9(4) COMP VALUE 21.

PROCEDURE DIVISION.
    .
    MOVE SPACES TO INFOBUF.
    .
    MOVE 1 TO NUMBER-OF-ENTRIES.
    MOVE 19 TO ENTRY-LENGTH.
    CALL "VGETFILEINFO" USING COMAREA, INFOBUF, INFOBUFLN.
  
```

The example shown above illustrates the data declaration of *infobuf* and *infobuflen* and the passing of parameters to VGETFILEINFO. Note that before the intrinsic is called, *infobuf* is initialized to spaces (*not* zeros). The intrinsic copies 19 two-byte words of information about the open forms file into *infobuf*.

VGETFORMINFO

Returns information about specified forms to an application.

Syntax

```
VGETFORMINFO {comarea, infobuf, infobuflen}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETFORMINFO:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VGETFORMINFO sets the following field:

cstatus Set to nonzero value if call unsuccessful.

infobuf A record through which you pass the request for form information and to which the intrinsic returns the specified information. This parameter must be initialized to spaces before filling in your request and calling VGETFORMINFO. The layout of *infobuf* is shown in table 6-12. The two required fields of *infobuf* pass user-supplied parameters to VGETFORMINFO as follows:

numofentries Specifies how many forms you want information about.

entrylength Indicates how many two-byte words of information (maximum 36 as shown in table 6-12) you want about each form.

The rest of *infobuf* (beginning with word 3) must consist of *entrylength* number of two-byte words for each form you want information about. Thus, the total length of *infobuf* in two-byte words must be as follows:

$$2 + (\textit{entrylength} * \textit{numofentries})$$

You may pass one or both of the two permissible keys, which must be passed in the position indicated in the table. Remember that *infobuf* must be initialized to spaces before filling in the parameters and any key.

infobuflen Two-byte integer variable set to the number of two-byte words in *infobuf*.

Discussion

This intrinsic accesses an internal table and places information about forms into *infobuf*. You tell VGETFORMINFO how many forms and how much information you want about each form.

Table 6-12. Form Information Buffer

Data Type	Position	Contents	Comments
Integer (Two-byte)	1	<i>numofentries</i>	Required
	2	<i>entrylength</i>	Required
Character Array (16-byte)	3-10	Form name	Permissible key; last byte of position 10 is unused (15 bytes maximum).
Integer (Two-byte)	11	Form number	Permissible key; forms are ordered within the forms file in alphabetic order, thus this argument reflects the order of the form within the file, not a permanently assigned form numeric identifier. As a result, the number may change if the forms file is recompiled.
	12	Number of fields in form.	
	13	Data buffer length in bytes.	
Character Array (16-byte)	14-21	Name of next form.	Last byte of position 21 is unused.
Character (Two-byte)	22	Repeat option.	First byte of position 22; may be N, A, or R.
	22	Next form option	Last byte of position 22; may be C, A, or F.
Integer Array (32-byte)	23-38	Fields in error.	Bit map; bit is on if field is in error.

Passing INFOBUF Without Entering Keys

Keys are optional. If you do not supply a key to indicate the first form you want information about, VGETFORMINFO starts with the first form not already reported on by the current call.

As an example, suppose you want information about 3 forms, and you enter form number 6 as a key. VGETFORMINFO returns information about forms 6, 7, and 8. If you enter no keys, VGETFORMINFO returns information about forms 1 through 3.

VGETFORMINFO

Table Wraparound

VGETFORMINFO starts over with the first form in the file if you request information about more forms than are in the file or, if when you pass a key, your request goes beyond the end of the file.

Suppose a file has 10 forms. If you request information about 5 of the forms and enter form number 8 as a key, VGETFORMINFO returns information about forms 8 through 10, and then goes to the beginning of the file and returns information about forms 1 and 2.

Error Determination

To enable applications to determine which fields are in error for a given form at run-time, VGETFORMINFO optionally returns a 32-byte bit map containing these error flags to show which fields failed the edit checks.

Each field in a form is represented by a bit in the bit map according to its field creation number. The bit map is a 32-byte logical array to accommodate the maximum of 255 field numbers. The most significant (left most) bit of the first byte in the bit map array represents the field with the field number equal to 1 as assigned by FORMSPEC. For example, the field assigned a field number of 17 would be represented by the most-significant bit of the third byte in the array. If a field is in error, the corresponding bit representing that field will be set to a one. Otherwise, the value will be zero.

The bit map is valid for the current form only. If the bit map is requested for any other form, it will contain all zeros. Therefore, a value of zero should not be interpreted to mean the field is not in error for the current form, (i.e. the last form retrieved by VGETNEXTFORM). Furthermore, bit positions representing nonexistent field numbers for a given form will contain a value of zero.

To retrieve the bit map, the user-supplied parameter *entrylength* in *infobuf* should contain a value of 36. This is to inform VPLUS/V to supply 36 two-byte words of information pertaining to the form requested. The last 16 two-byte words will contain the bit map indicating which fields are in error.

The *infobuf* parameter normally passed to VGETFORMINFO is calculated using the *entrylength* of 36. To request the bit map for the current form, the calculation is as follows:

$$2 + (\text{entrylength} * \text{numofentries})$$

where *entrylength* is equal to 36 and *numofentries* is equal to 1. Thus, the *infobuf* parameter should contain a value of 38 when calling VGETFORMINFO.

For languages which provide bit manipulation capability, the bits can be shifted left or right to determine which bits are on. For other languages, a mask may be used with the bit map to determine whether a bit is on or off. An intrinsic called BITMAPCNV is available to help decode and set bits in the bit map. Refer to Appendix I for more information.

Example

```

COBOL
DATA DIVISION.
  .
  .
  .
WORKING-STORAGE SECTION.
01  INFOBUF.
    05  NUMBER-OF-ENTRIES          PIC S9(4) COMP.
    05  ENTRY-LENGTH              PIC S9(4) COMP.
    05  ENTRY-TABLE OCCURS 9 TIMES.
      10  FORM-NAME                PIC X15.
      10  FILLER                   PIC X.
      10  FORM-NUMBER              PIC S9(4) COMP.
      10  NUMBER-OF-FIELDS         PIC S9(4) COMP.
      10  BUF-LENGTH              PIC S9(4) COMP.
      10  NEXT-FORM                PIC X(15).
      10  FILLER                   PIC X.
      10  REPEAT-OPTION            PIC X.
      10  NFORM-OPTION             PIC X.
01  INFOBUFLEN                    PIC S9(4) COMP VALUE 62.
  .
  .
  .
PROCEDURE DIVISION.
  .
  .
  .
  MOVE SPACES TO INFOBUF.
  .
  .
  .
  MOVE 3 TO NUMBER-OF-ENTRIES.
  MOVE 20 TO ENTRY-LENGTH.
  MOVE "FORM1" TO FORM-NAME.
  CALL "VGETFORMINFO" USING COMAREA, INFOBUF, INFOBUFLEN.

```

The example shown above illustrates the data declaration of *infobuf* and *infobuflen* and the passing of parameters to VGETFORMINFO. Note that before the intrinsic is called, *infobuf* is initialized to spaces (*not* zeros). The intrinsic copies 20 two-byte words of information about 3 forms in collating sequence order, beginning with the form named FORM1, into *infobuf*.

VGETKEYLABELS

Retrieves global or current form function key labels.

Syntax

```
VGETKEYLABELS {comarea,formorglob,numoflabels,labels}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETKEYLABELS:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VGETKEYLABELS may set the following *comarea* item:

cstatus Set to nonzero value if call unsuccessful.

formorglob

Two-byte integer specifying which type of labels are to be retrieved.

- 0 - Retrieve global labels
- 1 - Retrieve current form labels

numoflabels

Two-byte integer indicating how many labels are to be retrieved. This value must be from 1 to 8, inclusive.

labels

A byte array in which the labels are passed back to an application. The length of the array must be at least *numoflabels* * 16. (Each label is 16 bytes long.)

Discussion

This intrinsic is used to copy global or current form function key labels that have been specified with FORMSPEC, VSETKEYLABEL, or VSETKEYLABELS into an application. Labels for some or for all eight function keys can be copy. The *labeloption* must be set to one prior to VOPENFORMF.

When this intrinsic is called, any active labels are retrieved from the key label buffers in memory into an application. If there are no active labels, a buffer of spaces is returned. The labels are displayed by calling VSHOWFORM.

VGETKEYLABELS

Example

```
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*2 FORMORGLOB,NUMLABELS
FORMORGLOB=1
NUMLABELS=2
CALL VGETKEYLABELS (COMAREA,FORMORGLOB,NUMLABELS,LABELS)
```

```
SPL/PASCAL
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);
```

VGETLANG

Returns the native language ID assigned to the forms file. For more information on Native Language Support, see Section 8.

Syntax

```
VGETLANG {comarea,langnum}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETLANG:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VGETLANG may set the following *comarea* items:

cstatus Set to nonzero value if call is unsuccessful

langnum Two-byte integer variable to which VGETLANG returns the native language ID assigned to the forms file.

Discussion

The forms file must be opened before calling VGETLANG. Otherwise, *cstatus* returns a nonzero value. The native language ID returned is always the number assigned to the forms file on the Terminal/Language Menu in FORMSPEC. If the international language ID (-1) is assigned, which means that VSETLANG may be used to specify the native language ID when the application executes, the native language ID returned by VGETLANG will be the number assigned to the forms file (-1), *not* the number specified with VSETLANG.

Example

COBOL
CALL "VGETLANG" USING COMAREA, LANGNUM.

BASIC
120 CALL VGETLANG(C(*),L)

FORTRAN
CALL VGETLANG(COMAREA,LANGNUM)

SPL/PASCAL
VGETLANG(COMAREA,LANGNUM);

VGETNEXTFORM

Reads the next form from an open forms file into the form definition area of memory.

Syntax

```
VGETNEXTFORM {comarea}
```

Parameters

comarea

Must be *comarea* name specified when the forms file, from which the form is to be retrieved, was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETNEXTFORM:

<i>cstatus</i>	Set to zero.
<i>comarealen</i>	Set to total number of two-byte words in <i>comarea</i> .
<i>cmode</i>	Set to zero.
<i>nfname</i>	Set to name of next form, or \$END, \$HEAD, \$RETURN, or \$REFRESH. See "Communication Area" earlier in this section for more information. (May also be set by prior call to VGETNEXTFORM, VOPENFORMF, VREADBATCH, VINITFORM, VFIELDEDITS, or VFINISHFORM.)
<i>repeatapp</i>	Set to 1 if current form is to be repeated, to 2 if repeated and appended. Note that <i>repeatapp</i> 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.)
<i>freezapp</i>	Set to 1 if next form is to be appended to current form, to 2 if current 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*:

<i>numerrs</i>	Set to zero.
<i>cfname</i>	Set to name of form just read from file.
<i>cstatus</i>	Set to nonzero value if call unsuccessful.
<i>filerrnum</i>	Set to file error code if MPE file error.

VGETNEXTFORM

multiusage

Set to 1 if current form is child or sibling of previous form, zero otherwise.

In addition, if a new form has been retrieved (*repeatapp* is zero), VGETNEXTFORM sets the following items:

<i>cfnmlines</i>	Set to number of lines in form just read (now the current form).
<i>nfname</i>	Set to name of next form to be read from file. See Discussion below for more information.
<i>repeatapp</i>	Set to value read from forms file for this (current) form.
<i>freezapp</i>	Set to value read from forms file for this (current) form.
<i>dbuflen</i>	Set to length (in bytes) of the current form just read from the forms file.

Discussion

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 *cfnmlines*, *nfname*, *repeatapp*, *freezapp*, or *dbuflen*. Note that a repeating form is repeated until *repeatapp* is set to zero, either by an application or, for ENTRY, when the user presses **NEXT FORM** 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.

Example

```
COBOL
CALL "VGETNEXTFORM" USING COMAREA.
```

```
BASIC
11 CALL VGETNEXTFORM(C(*))
```

```
FORTRAN
CALL VGETNEXTFORM(COMAREA)
```

```
SPL/PASCAL
VGETNEXTFORM(COMAREA);
```

The examples above call VGETNEXTFORM to retrieve the next form from the forms file and reset the *comarea* according to the values in the next form.

VGETNEXTFORM

Copies character coded data contents from data buffer into an application, converting numeric value to specified type.

Syntax

```
VGETtype {comarea,fieldnum,variable}[,numdigits,decplaces]*
```

Parameters

type

The *type* in VGETtype indicates that this intrinsic may be specified as:

VGETINT	converts value to two-byte integer
VGETDINT	converts value to four-byte integer
VGETREAL	converts value to four-byte real value
VGETLONG	converts value to eight-byte long value
*VGETPACKED	converts value to packed decimal format (COBOL <i>usage is comp-3</i> format); this intrinsic has two extra parameters, <i>numdigits</i> and <i>decplaces</i> . Both are two-byte integer variables that contain the number of digits and number of decimal places, respectively, specified by the COBOL <i>usage is comp-3</i> data declaration.
*VGETZONED	converts value to zoned decimal format (COBOL default format); has the parameters <i>numdigits</i> and <i>decplaces</i> , which are two-byte integer variables that contain the number of digits and number of decimal places, respectively, specified by the COBOL data declaration.
VGETYYMMDD	converts date to a six-byte character value

For example: CALL "VGETINT" USING COMAREA, FIELDNUM, VARIABLE

```
CALL "VGETPACKED" USING  
COMAREA, FIELDNUM, VARIABLE, NUMDIGITS, DECPLACES
```

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF.

If not already set, the following *comarea* items must be set before calling VGETtype:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VGETtype may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful or if requested field has an error.

numerrs May be decremented if new value replaces the value of a field that was in error.

fieldnum Two-byte integer variable containing the number assigned to the field by FORMSPEC.

variable Variable within application, of type specified in VGETtype, into which converted value is placed.

Discussion

If there is no ARB, all data is read from the unprotected fields on the screen as character strings and concatenated to create a data buffer. The VGETBUFFER intrinsic copies the entire buffer to an application program or VGETFIELD can be used to obtain the contents of an individual field.

NOTE

If you are using VGETBUFFER in conjunction with an ARB, you do not need to use VGETtype. VGETBUFFER performs all the required conversions on the screen data in the buffer.

You can use both VGETBUFFER with an ARB and VGETtype calls in the same program: the *buffercontrol* setting in the *comarea* that controls ARB processing can be switched on or off for each form.

The VGETtype intrinsics, VGETINT, VGETDINT, VGETREAL, VGETLONG, VGETYYMMDD, VGETPACKED, and VGETZONED, have been provided to perform conversion from character coded data to the seven indicated data types. This intrinsic copies the contents of 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 character coded form, is converted to the specified type and then copied to the variable in the application. (Refer to table 6-13 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.

VGETtype

Table 6-13. Numeric Type Conversion

Type	Format
INT	Two-byte word fixed-point; twos complement representation of positive and negative values; range from -32768 through +32767.
DINT	Four-byte word fixed-point format; twos complement representation of positive and negative range between approximately -2 billion and +2 billion. (Not used by BASIC programs.)
REAL	Four-byte 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 eight-bytes; 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.)
PACKED	COBOL only; <i>comp-3</i> . See COBOL II/3000 Reference Manual.
ZONED	COBOL only. See COBOL II/3000 Reference Manual.
YYMMDD	Six-byte ASCII, in format YYMMDD; for example, 870623.

VGETtype

The following chart will help you correlate VPLUS/V data types with VGETtype intrinsics and programming language data types.

Table 6-14. Correlation of VGETtype with the Data Types for Each Language

Language	Intrinsic	Data Type
FORTRAN	VGETINT VGETDINT VGETREAL VGETLONG	Integer*2 Integer*4 Real Double Precision
BASIC	VGETINT VGETREAL VGETLONG	Integer Real Long
SPL	VGETINT VGETDINT VGETREAL VGETLONG	Integer Double Real Long
COBOL	VGETINT VGETDINT	S9-S9(4)COMP S9(5) - S9(9)COMP
PASCAL	VGETINT VGETDINT VGETREAL VGETLONG	Subrange -32768..32767 Integer Real Longreal

1. If errors occur during conversion, the two-byte 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 VFIELDERRORS or VSETERROR), the conversion is performed, but the two-byte status word is also set to an error value.
3. An attempt to convert a number larger than 32767 using VGETINT returns an error value (504) in the two-byte status word and leaves the receiving value unchanged.
4. All numeric separators are stripped before conversion is performed.
5. Fields of type CHAR may be converted as long as they contain numeric characters (including signs and separators). Otherwise, an error value is returned in the two-byte status word.
6. VGETINT and VGETDINT 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 rightmost "n" characters are treated as a fraction.
7. Note that if VGETFIELD is used to pass a field containing a decimal point to a COBOL program, the decimal point is also passed and no arithmetic may be performed on the field.

VGETtype

8. Negative numbers can be zoned correctly for COBOL only by using the VGETINT and VGETDINT intrinsics. VGETBUFFER and VGETFIELD transfer the negative sign, but COBOL treats 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) truncates to the value of 2345.
10. VGETINT may be used to convert positive integers to type LOGICAL in SPL.

Example

```
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(COMAREA,FIELD,K1)
```

```
SPL/PASCAL
INTEGER
  FIELD,
  COUNT;
  .
FIELD:=5;
VGETINT(COMAREA,FIELD,COUNT);
```

The calls in this example convert a value contained in field 5 in the data buffer to two-byte integer representation and store it in an application.

VINITFORM

Initializes fields in data buffer according to specifications defined in the initialize phase of field definition. Both VOPENFORMF and VGETNEXTFORM must have been executed successfully prior to calling VINITFORM.

Syntax

```
VINITFORM {comarea}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VINITFORM:

cstatus Set to zero

comarealen Set to total number of two-byte 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 current form code, if code changed by processing specifications.

freezapp Set to new next form code, if code changed by processing specifications.

Discussion

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 child or sibling to the previous form, data from the previous form is transferred to this form (with conversion if necessary) before initializations occur. If a field in a child or sibling form must be initialized to blanks, use \$EMPTY as an initial value.

VINITFORM only resets the field error flag if the field is initialized explicitly by initialization phase (INIT) processing specifications.

Example

```
COBOL  
CALL "VINITFORM" USING COMAREA.
```

```
BASIC  
140 CALL VINITFORM(C1(*))
```

```
FORTRAN  
CALL VINITFORM(COMAREA)
```

```
SPL/PASCAL  
VINITFORM(COMAREA);
```

The calls shown above set initial values in the data buffer area of memory according to initialize specifications defined for each field in the current form.

VLOADFORMS

Loads specified forms into terminal local form storage memory.

Syntax

```
VLOADFORMS {comarea,numofforms,formsloaded,forms}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VLOADFORMS:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VLOADFORMS sets the following items in *comarea*:

cstatus Set to nonzero value if call unsuccessful.

numofforms Two-byte integer value, between 1 and 255, inclusive, indicating the number of forms to be loaded.

formsloaded Two-byte integer value indicating the number of forms that were successfully loaded.

forms The names of the forms to be loaded. Each name can be up to 15 characters and is stored in a 16-byte character array with a one byte filler that is not part of the name.

Discussion

This intrinsic is used on terminals having local form storage. VLOADFORMS loads the forms named in *forms* into terminal local form storage memory in the order they are specified. It may not be possible to load all the forms named because of a limit of available space in local form storage or in the form storage directory (set with *formstoresize* prior to VOPENTERM and VOPENFORMF). When there is no more space, any other forms named in *forms* are ignored. The value returned in the *formsloaded* parameter is:

formsadded + *formsalreadypresent*

where *formsadded* is the number of forms specified in the *forms* parameter that were *actually* loaded into the local storage of the terminal and *formsalreadypresent* is the number of forms specified in the *forms* parameter that were confirmed to *already* exist in the local storage.

VLOADFORMS

Because forms are loaded by form name, family relationships are ignored. Multiple family members can be loaded at the same time.

Note that the terminal keyboard may be locked briefly while VLOADFORMS verifies whether or not a form was loaded. In case keys are pressed during this time, the terminal beeps to indicate that the keystrokes and/or entered data are lost. You can avoid this by not calling VLOADFORMS between calls to VSHOWFORM and VREADFIELDS. Refer to the discussion on "Local Forms Storage" earlier in this section.

Example



```
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*2 NFORM,FLOAD
CHARACTER*16 FORMS
NFORM=1
FORMS="FORMA  "
CALL VLOADFORMS(COMAREA,NFORM,FLOAD,FORMS)
```

```
SPL/PASCAL
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 the specified file is new, creates a batch file and then opens it for processing.

Syntax

```
VOPENBATCH {comarea,batchfile}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VOPENBATCH:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VOPENBATCH sets the following *comarea* items:

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 nondeleted records in file if existing file opened.

VOPENBATCH may set the following *comarea* items:

cstatus Set to nonzero 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.

Discussion

VOPENBATCH opens the specified batch file for processing by the calling program. The batch file may be an existing file or a new file.

VOPENBATCH

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
- USASCII coded data
- Fixed-length records
- No carriage control
- :FILE command allowed
(use actual file designator)
- Update access
- Exclusive (nonshared) access
- No dynamic locking
- No multirecord access
- Normal buffering

Record Format

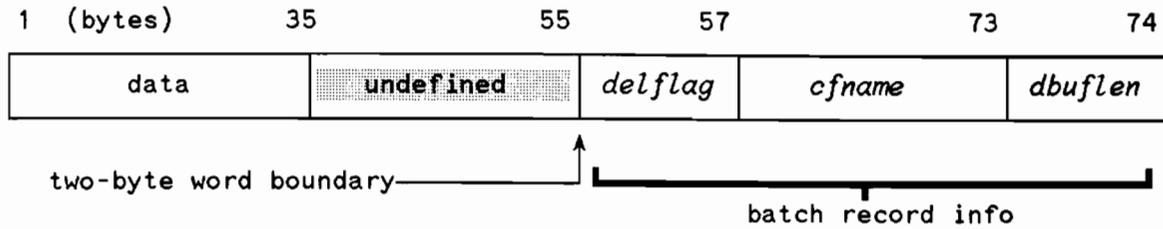
The size of the fixed-length batch file records must be large enough to hold the largest data buffer used by the forms file associated with the batch file, plus 10 two-byte words for batch record control information. If the largest data buffer is an even number of bytes, an additional two-byte word is added before the control information. This batch record information consists of:

2 bytes (logical)	-- Delete flag (TRUE if record deleted)
16 bytes (character)	-- Current form name + extra character
2 bytes (logical)	-- Data buffer length

Total = 20 bytes

VOPENBATCH

The above batch record information is written at the end of each record in the batch file, starting on a two-byte word boundary. To illustrate, assume the record size is 74 bytes, and the data only requires 35 bytes (bytes are numbered from 1):



Labels

In addition, VOPENBATCH creates sufficient user labels (each 256 bytes long) to hold any save field buffers, plus 176 bytes for the collection environment and the forms file version.

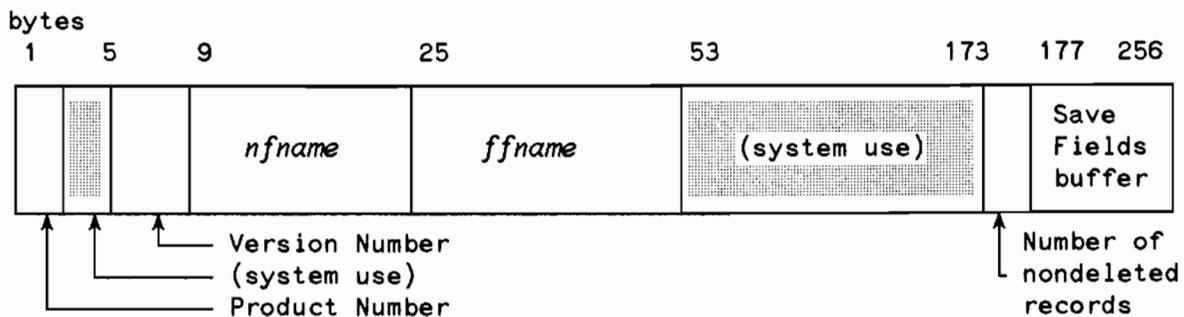
The length of the save field buffers depends on how many (if any) save fields were defined for the form and the length of each. The collection environment consists of the forms file name and version number, the next form name, and 122 bytes of system information. This information is stored as follows:

# of Bytes	
2	-- Product number
2	-- (System use)
4	-- Forms file version (date and time of compilation)
16	-- Next form name
28	-- Forms file name
120	-- Reserved for system use
4	-- Number of nondeleted records in batch file

Total = 176

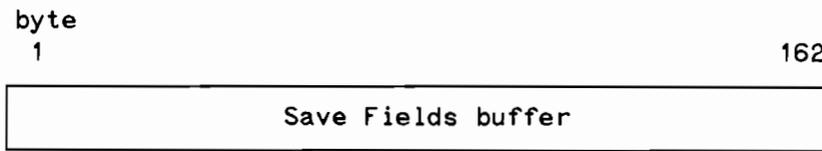
For example, assume that the save fields buffer requires 242 bytes. The following user labels are created by VOPENBATCH:

LABEL 0 -- Contains first 80 bytes of Save Fields buffer



VOPENBATCH

LABEL 1 -- Contains remaining 162 bytes of Save Fields buffer

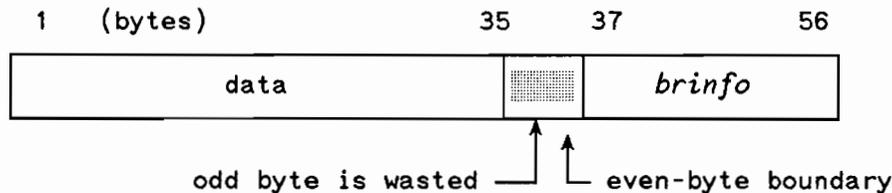


Note that the length of the save fields buffers is determined by taking the number of bytes in each save field, summing them together, adding one byte, and then, if odd, rounding the total up to an even number.

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. Undefined-length records are formatted like fixed-length records.

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 bytes of data:



The total size of this record is 56 bytes. 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 bytes) plus 20 bytes for the batch record information.

Example

```
COBOL
CALL "VOPENBATCH" USING COMAREA, BATCH.
```

```
BASIC
170 CALL VOPENBATCH(C(*),B1$)
```

```
FORTRAN
CALL VOPENBATCH(COMAREA, BATCH)
```

```
SPL/PASCAL
VOPENBATCH(COMAREA, BATCH);
```

If the requested batch file name is an existing file, it is opened and the user 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.

Syntax

```
VOPENFORMF {comarea,formfile}
```

Parameters

comarea

The *comarea* name must be unique for each open forms file. If not already set, the following *comarea* items must be set before calling VOPENFORMF:

<i>cstatus</i>	Set to zero.
<i>language</i>	Set to the code identifying the programming language of the calling program.
<i>comarealen</i>	Set to total number of two-byte words in <i>comarea</i> .
<i>usrbuflen</i>	BASIC programs only; set to number of two-byte words needed for <i>comarea</i> extension.
<i>labeloption</i>	Set to zero or one.
<i>formstoresize</i>	Set to minus one, zero, or one through 255 to indicate the number of forms allowed in the form storage directory.

VOPENFORMF sets the following *comarea* items:

<i>cstatus</i>	Set to nonzero value if call unsuccessful.
<i>lastkey</i>	Set to zero.
<i>numerrs</i>	Set to zero.
<i>recnum</i>	Set to zero.
<i>dbuflen</i>	Set to zero.
<i>cmode</i>	Set to zero (collect mode).
<i>repeatapp</i>	Set to zero (no repeat/append).
<i>freezapp</i>	Set to zero (clear current form).
<i>printfilnum</i>	Set to zero.

VOPENFORMF

<i>deleteflag</i>	Set to FALSE (all zeros) (used for batch file intrinsics).
<i>cfname</i>	Set to blank.
<i>nfname</i>	Set to name of head form.
<i>cstatus</i>	Set to nonzero value if call unsuccessful.
<i>filerrnum</i>	Set to file error code if MPE file error.
<i>windowenh</i>	Set to enhancement code defined in forms file.

If the data capture devices are used:

<i>errorlight</i>	Set to default or user specified value.
<i>splitmsgpause</i>	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.

Discussion

VOPENFORMF opens the specified forms file for processing by the calling application. If the calling application is BASIC, it must provide its own *comarea* extension immediately following the *comarea* and specify the size of this extension in *usrbuflen*. If the application is written in a programming language other than BASIC, a DL area is obtained for use as the *comarea* extension. In this case, the application must not use the DL area for other functions. Refer to Appendix E for more information on the DL area. The forms file listing gives an estimate of DL to DB area used. You use this estimate with the MAXDATA parameter at PREP or RUN time. The *labeloption* must be set to one if you are using function key labels.

Example

```
COBOL
CALL "VOPENFORMF" USING COMAREA, FNAME.
```

```
BASIC
100 CALL VOPENFORMF(C(*),F1$)
```

```
FORTRAN
CALL VOPENFORMF(COMAREA,FNAME)
```

```
SPL/PASCAL
BYTE ARRAY FNAME(0:34);
:
MOVE FNAME:="FORMSA ";
VOPENFORMF(COM1,FNAME);
```

VOPENTERM

Opens a VPLUS/V supported terminal.

Syntax

```
VOPENTERM {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 calling VOPENTERM:

<i>cstatus</i>	Set to zero.
<i>language</i>	Set to the code identifying the programming language of the calling program.
<i>comarealen</i>	Set to total number of two-byte words of <i>comarea</i> .
<i>formstoresize</i>	Set to minus one, zero, or one through 255 to indicate the number of forms allowed in the form storage directory.

VOPENTERM may set the following *comarea* items:

<i>cstatus</i>	Set to nonzero value if call unsuccessful.
<i>filerrnum</i>	Set to file error code if MPE file error.
<i>filen</i>	Set to MPE file number of terminal.
<i>identifier</i>	Set to appropriate VPLUS/V terminal type code.
<i>environ</i>	Set to logical device number of terminal.
<i>labinfo</i>	Set to appropriate number and length of labels used by VPLUS/V.

If the data capture devices are used:

<i>leftmodule</i>	Set to appropriate value defining terminal.
<i>rightmodule</i>	Set to appropriate value defining terminal.
<i>keyboard</i>	Set to appropriate value defining terminal.
<i>display</i>	Set to appropriate value defining terminal.

VOPENTERM

termfile

Character string of up to 36 characters (including a terminator) that identifies the terminal. If set to a blank, the session device is used to get the *ldev#* of the logon terminal. 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.

Discussion

This intrinsic 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 the session device is opened.

Once the terminal is opened for control by VPLUS/V, an application should not call system intrinsics to communicate with the terminal. All terminal I/O must be controlled through VPLUS/V intrinsics.

If you are using an HP 2640B or HP 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. The data capture devices are treated as character mode terminals.

Example

```
COBOL
77 T1 PIC X(8) VALUE "VTERM  ".
01 COMAREA.
   :
PROCEDURE DIVISION.
   :
OPENTERMINAL.
   CALL "VOPENTERM" USING COMAREA, T1.
```

This example opens the device referenced by the file equation VTERM. If no file equation exists, the default is the logon device.

```
BASIC
90 T1$=" "
100 CALL VOPENTERM(C(*),T1$)
```

```
FORTRAN
T1=" "
VOPENTERM(COMAREA,T1)
```

```
SPL/PASCAL
MOVE T1:=" ";
VOPENTERM(COMAREA,T1);
```

The examples shown above open the session device with *comarea* identified as COMAREA (C(60) for BASIC).

VPLACECURSOR

Allows an application to position the cursor to any input field at run-time.

Syntax

```
VPLACECURSOR {comarea,fieldnum}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPLACECURSOR:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VPLACECURSOR may set the following *comarea* items:

cstatus Set to non-zero value if call unsuccessful.

filerrnum Set to file error code if MPE file error.

fieldnum A two-byte integer that identifies the input field, where a positive number indicates the field number; a negative number indicates the screen order number.

Discussion

VPLACECURSOR allows an application to position the cursor to any input field at run-time. This intrinsic places the cursor to the input field specified by *fieldnum*. Calling VPLACECURSOR with a positive number indicates the field number; a negative number indicates the screen order number. Using the field number is preferable because if the fields in a form are rearranged, no modification to an application is necessary.

VPLACECURSOR must be called after calling VSHOWFORM. VPLACECURSOR returns an error if the field number or the screen order number does not exist. An error is also returned if the intrinsic is called to place the cursor to a display-only field.

VPLACECURSOR

Example

COBOL
CALL "VPLACECURSOR" USING COMAREA, FIELD-NUM.

BASIC
120 CALL VPLACECURSOR(C(*), FIELDNUM

FORTRAN
CALL VPLACECURSOR(COMAREA, FIELDNUM)

SPL/PASCAL
VPLACECURSOR(COMAREA, FIELDNUM);

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.

Syntax

```
VPOSTBATCH {comarea}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPOSTBATCH:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VPOSTBATCH will set the following field:

cstatus Set to nonzero if called unsuccessful, zero if successful.

Discussion

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. Refer to the discussion of VOPENBATCH for a description of the environmental information.

If a system crash or power failure occurs while the batch file is open, all data before the end-of-file mark is preserved, and data collection continues from that point. In ENTRY, VPOSTBATCH is called after every 20 records, though you may extend or shorten this posting interval. 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 intrinsic when the last record referenced is in the middle of the file, VPOSTBATCH posts 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 have been deleted since the last call to the VPOSTBATCH intrinsic.

VPOSTBATCH

Example

COBOL
CALL "VPOSTBATCH" USING COMAREA.

BASIC
290 CALL VPOSTBATCH(C(*))

FORTRAN
CALL VPOSTBATCH(COMAREA)

SPL/PASCAL
VPOSTBATCH(COMAREA);

VPRINTFORM

Prints the current form on an offline list device.

Syntax

```
VPRINTFORM {comarea,printcntl,pagecntl}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPRINTFORM:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

printfilnum Set to file number of the list file to which the form is to be printed. If set to zero, VPRINTFORM opens the device "LP" as the list file and sets *printfilnum* 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 nonzero value if call unsuccessful.

filerrnum Set to file error code if MPE file error.

printcntl

Two-byte 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

Two-byte 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 intrinsic, including the following:

61(octal) -- Page eject
320(octal) -- No line feed or carriage return
zero -- Carriage return/line feed

Discussion

VPRINTFORM prints the current form to a list file. It is analogous to VSHOWFORM, in that it prints the form and the current data buffer values, except that VPRINTFORM prints the form on a hardcopy device rather than on the terminal. Enhancements obviously cannot be shown directly; the window line and key labels are not printed. The form must have been read into the form definition area of memory by a prior call to VGETNEXTFORM. The information printed depends on what is in the data buffer, which may not necessarily be the same as what is displayed.

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 bytes. This is equivalent to using the file equation:

```
:FILE FORMLIST;DEV=LP;REC=-80
```

A user may change any of these characteristics with an MPE :FILE command.

Example

```
COBOL  
CALL "VPRINTFORM" USING COMAREA, UNDERLINE, PAGE.
```

```
BASIC  
135 CALL VPRINTFORM(C(*),U,P)
```

```
FORTRAN  
CALL VPRINTFORM(COMAREA,UNDRLN,PAGE)
```

```
SPL/PASCAL  
VPRINTFORM(COMAREA,UNDERLINE,PAGE);
```

Each of the calls shown above prints the current form on a list device; if not already open, it opens the device file.

VPRINTSCREEN

Prints the entire contents of a terminal screen to an offline list device or laser printer during VPLUS execution.

Syntax

```
VPRINTSCREEN {comarea, readsize}
```

Parameters

comarea

If not already set, the following *comarea* items must be set before calling VPRINTSCREEN:

cstatus Set to zero.

comarealen Set to the total number of two-byte words in *comarea*.

printfilenum Set to the file number of the list file to which the form is printed. If set to 0, VPRINTSCREEN opens the device "LP" as the list file and sets *printfilenum* to the file number of the opened list file.

VPRINTSCREEN may set the following *comarea* items:

printfilenum If VPRINTSCREEN opened the list file, set to the file number of the opened file.

cstatus Set to a nonzero value if call is unsuccessful.

fileerrnum Set to the MPE error code if an MPE file error is encountered.

readsize

Two-byte integer, reserved for system use. Must be set to 0.

Discussion

This intrinsic provides the capability for printing the current screen display with function keys, line drawing characters and appended forms, as well as the data on the screen. It differs from VPRINTFORM, which is limited to printing only the data in the form data buffer. However, VPRINTFORM offers the option of underlining fields, which VPRINTSCREEN in LP mode does not.

The programmer can produce copies of VPLUS screens in either of two ways: incorporating VPRINTSCREEN into an application so that screen images can be captured at run-time with their data, or developing a simple utility that allows data to be entered into the screens before calling VPRINTSCREEN to reproduce them. The utility has the advantage of removing the overhead caused by

VPRINTSCREEN

VPRINTSCREEN from the application, while still providing a way to reproduce screens and data for product literature.

VPRINTSCREEN always uses the Pascal HEAP procedures to perform stack allocation. This introduces the risk of conflict for programs written in COBOL, FORTRAN/66 or SPL, because the existing intrinsics would use DLSIZE for stack allocation in these cases. Applications written in these languages must, therefore, follow two rules when calling VPRINTSCREEN. They are:

1. The language id in the *comarea* must be set to 5.
2. The INTRINSIC calling mechanism must be used when calling the VPLUS intrinsics from the main and all interacting parts of the application.
For example, with COBOL the application must use:

```
:CALL INTRINSIC <intrinsic name>
```

Refer to Appendix E of this manual, and to the COBOL and Pascal reference manuals, for more information on calling mechanisms.

For applications that use a language id of 5 in the VPLUS *comarea*, including Pascal, FORTRAN/77 and HPBUSINESS BASIC, VPLUS uses the HEAP procedures for stack allocation, so VPRINTSCREEN can be called in the standard format.

The programmer may implement this feature by defining a function key that allows the user to print the screen contents at any time. This would be useful for providing immediate output during production.

Modes of Operation

VPRINTSCREEN operates in two modes, normal and documentation mode. A VPLUS-supported terminal is required for execution of this intrinsic.

Normal Mode

This is the default calling mode for VPRINTSCREEN. When called, the value in the *printnumfile* word of the *comarea* is used to determine the list device. If the calling program opens the list file, it must supply the file number of this file in *printfilenum*. VPRINTSCREEN 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
```

The user may change any of these characteristics with a :FILE command.

NOTE

It is recommended that VPRINTSCREEN and VPRINTFORM not be used in the same program. Since the same list file is used for both listings, output from the two calls will be intermixed.

Each time VPRINTSCREEN is called, a PAGE EJECT is performed at the end of the print operation.

VPRINTSCREEN

Documentation Mode

You require TDP in order to use VPRINTSCREEN in this mode. VPRINTSCREEN, in conjunction with TDP, provides the capability to print screen contents on a laser printer (HP2680A or HP2688A). In this mode, field highlighting other than color, borders, alternate character sets and active function keys are captured and converted to the requisite font for printing on the laser printer.

Documentation mode is enabled by setting a JCW before running the program. It is:

```
:SETJCW VPRINTJCW=1
```

When VPRINTJCW is set to 1, the list file FORMLIST is NOT opened. Instead, a temporary file called EPOCLIST is created (or appended to, if it already exists). EPOCLIST can be saved and renamed on completion of the screen capture, then input to TDP and 'finald'. The user can add text to the file or include it in a separate TDP file. Refer to the *TDP Reference Manual* (Part No. 36578-90004) for more information on use and include files.

In order to print the forms on a laser printer, an environment file must be created (if it does not already exist). Refer to the *IFS/3000 Reference Guide* (Part No. 36580-90001) for more information. The environment files for the HP2680 and HP2688 laser printers are VENV80 and VENV88 respectively. All environment files must include the font ids listed under "Limitations" below.

EPOCLIST uses VSETUP, a TDP include file, as the default file to reference the environment files, which must be accessible to TDP in order for EPOCLIST to be printed. If an environment file other than VENV80 or VENV88 is used, EPOCLIST must be modified to reference this file, and the font definitions from VENV80 and VENV88 must be included in it (see "Limitations" below).

Printing Screens from TDP. To print the contents of EPOCLIST to a laser printer, follow these steps:

1. :PURGE (or :RENAME) EPOCLIST
2. :SETJCW VPRINTJCW=1
3. :RUN <your application program>
4. Use the print function wherever it is available to save the screen contents, including data, in EPOCLIST.
5. Exit the application.
6. :SAVE EPOCLIST
7. :RENAME EPOCLIST, <new name> If you do not rename EPOCLIST, the output generated the next time the application is run will be appended to the existing file.
8. 6fd:RUN TDP.PUB.SYS
9. Text in the file and execute the command, "Final from <new name> to *HP2680" (or HP2688).

The screens will be printed out, one per page. Steps 8 and 9 may be specified in a job stream.

Merging Screens with a TDP File. You can include screens in an existing TDP document easily by using the file VEPOCUSE, a TDP use file that divides EPOCLIST into separate files containing one screen per file. The VSETUP file must be included as one of the first statements of the TDP document so that the correct environment file is referenced for printing the screens.

Follow these steps:

1. `:RUN TDP` Clear the workspace.
2. `:USE VEPOCUSE` Answer the prompts, and enter a PREFIX. VEPOCUSE separates the old file into new files, each containing one screen.
3. Text in the TDP document and include each screen file in the correct place.

It is recommended that screen files be included as separate, individual files in a document, rather than being incorporated directly into the text, because EPOCLIST has a record size of 168 bytes and most document files are set to 80 bytes.

Limitations

The following limitations pertain to the use of VPRINTSCREEN.

- Procedure calls must be modified in order to call the intrinsic from a language that does not use a VPLUS language id of 5 in the *comarea*; for example, COBOL, SPL and FORTRAN/66.
- It uses additional stack resources.
- Native Language support is NOT available for VPRINTSCREEN. To print a screen in another language, you require a LOCALIZED environment file which maps to the following fonts:

- `c` = full bright
- `g` = half-bright inverse video
- `d` = normal
- `l` = line-draw
- `m` = math

- The current environment files do not distinguish between full bright and half bright, but the code is set up to do so.
- Screen images cannot be scaled; fonts come in one size only.
- The maximum TDP record size is 168 characters. A single line in a screen may easily exceed this limit if it contains many escape sequences, in which case the line may be truncated.
- TDP macros 5-9 are used to minimize truncation, but some screens will reach the limit anyway. If you use your own macros 5-9, some inconsistencies may appear when text and screens are merged.
- TDP may indicate that errors have occurred when, in fact, there are none. The most common error messages are: "Unrecognizable command" and "n characters have been truncated".

VPRINTSCREEN

Example

This is a sample EPOCLIST file.

```
*>>DATE: FRI, MAY 3, 1987, 11:19AM
if main in hpvsetup.pub
image 28
need 28
>>>>>screen formatting commands<<<<<<
*>>END VPRINTSCREEN B.05.00
```

SAMPLE SCREEN v.FF

This is a sample form to demonstrate the VPRINTSCREEN intrinsic.

Name:	[]		
Street Address:	[]		
City:	[]	State:	[]
Phone:	[]		

ADD CHANGE DELETE REVIEW HELP PRINT EXIT

Figure 6-4. Sample EPOCLIST Screen from TDP

VPRINTSCREEN

VPUTBUFFER

Copies data from an application to the data buffer in memory.

Syntax

```
VPUTBUFFER {comarea,buffer,buflen}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPUTBUFFER:

<i>cstatus</i>	Set to zero
<i>language</i>	Set to the code identifying the programming language of the calling program.
<i>comarealen</i>	Set to total number of two-byte words in <i>comarea</i> . Must be at least 70 words in length if the ARB feature is used.
<i>buffercontrol</i>	Set bit 15 to 1 to indicate that data is to be transformed according to the ARB specifications.

VPUTBUFFER may set the following *comarea* items:

<i>cstatus</i>	Set to nonzero value if call unsuccessful.
<i>numerrs</i>	May be decremented as a result of new data replacing data in field with error.
<i>bufferstatus</i>	Bit 15 set to 1 if data conversion is successful.

buffer Character string in an application containing the data to be copied to the data buffer in memory.

buflen Two-byte integer variable that specifies the number of bytes to be copied to the data buffer in memory. The *comarea* item *dbuflen*, which contains the size of the data buffer in memory, may be used as this parameter.

Discussion

This intrinsic transfers data from a buffer in an application to the data buffer in memory. The length of the data moved is based on the number of bytes specified in the *buflen* parameter and the number of bytes in the largest data buffer in the forms file, whichever is less. The length of the buffer assigned to

VPUTBUFFER

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 bytes
Form B *dbuflen* = 200 bytes
Form C *dbuflen* = 75 bytes

In this case, the maximum data buffer length is 200 bytes. 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 VGETNEXTFORM to get form B and then VSHOWFORM to display form B with the 200 bytes of data moved to the data buffer with VPUTBUFFER.

Fewer bytes than the data buffer can hold may be transferred; the remaining space in the data buffer is not changed.

Previously, the data was moved to the data buffer exactly as it appeared in the application buffer, and you had to use VPUTtype to convert data in each field from the application data type to USASCII. Now, you can specify, using the FORMSPEC ARB feature, the mapping between the fields in the application buffer and those on the screen (see the discussion on creating an ARB in Section 3). Screen fields may be in a different order from those in the application buffer, and the data conversions will be performed when the data is written out to the screen. If any field being replaced by user data contained an error, VPUTBUFFER clears the error flag for the field and decrements *numerrs*.

Special Considerations

Designers using the ARB feature in VPUTBUFFER should be aware that damaging run-time errors could occur if the application is inadvertently run on a system that has a VPLUS/V version earlier than B.05.00.

To prevent this, the designer should do three things:

1. Document the product with a clear warning that VPLUS/V version B.05.00 or later **MUST** be used.
2. Use the VPLUS/V intrinsic HP32209 in the code. This intrinsic checks to make sure you are using the proper VPLUS/V version. If not, the application should terminate with an appropriate message.
3. Check offset 65 (*bufferstatus*) in the *comarea* on return from VPUTBUFFER. Bit 15 will be set to 1 if VPUTBUFFER performed the conversion successfully. In other words, the application must check both *status* and *bufferstatus* to be sure that the data was correctly converted.

Example

```
COBOL
01  DAT1.
   03  FIRSTNAME PIC X(6).
   03  LASTNAME  PIC X(18).
   ..
ACCEPT DAT1.
CALL "VPUTBUFFER" USING COMAREA, DAT1, DBUFLEN.
```

VPUTBUFFER

BASIC

235 L1=24

240 CALL VPUTBUFFER(C(*),D1\$,L1)

FORTRAN

CALL VPUTBUFFER(COMAREA,DAT1,DBLEN)

SPL/PASCAL

BYTE ARRAY DAT1(0:23);

⋮
VPUTBUFFER(COM1,DAT1,LEN);

The following calls transfer 24 bytes from an application area, DAT1 to the data buffer. In this example, the longest *dbuflen* is assumed to be 80 bytes.

VPUTBUFFER

VPUTFIELD

Copies data from an application into a specified field in the data buffer in memory.

Syntax

```
VPUTFIELD {comarea,fieldnum,fieldbuf,buflen,actualen,nextfldnum}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPUTFIELD:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VPUTFIELD may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

numerrs May be decremented if new value is moved to a field which has error flag set.

fieldnum Two-byte integer variable containing the number assigned to the field in the data buffer by FORMSPEC.

fieldbuf Character string in an application containing the data to be copied to the specified data buffer field.

buflen Two-byte integer containing the number of bytes to be copied from *fieldbuf* to the field identified by *fieldnum*.

actualen Two-byte integer that specifies the number of bytes actually copied.

nextfldnum Two-byte integer to which VPUTFIELD returns the number of the next field in screen order. If there are no more fields, it returns zero.

Discussion

The data in an application is copied 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. The number assigned to a field by FORMSPEC does not change regardless of any changes to the field's position in the form or to its length. The field number can only be changed with the batch

VPUTFIELD

command, RENUMBER, as described in Section 7. The field number must not be confused with the screen order number, which is the field's position in the data buffer and corresponds to its position in the form.

If the field is shorter than the data transferred to it, the data is truncated on the right. If the field is longer than the data transferred to it, the data, if any, in the remaining space in the field is not changed.

If the field whose data 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 from internal numeric representation to character strings, you must use VPUTtype, where type specifies the data type of the field in an application.

Example

```
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$="GADGET"
260 CALL VPUTFIELD(C(*),F1,I$,C,A,N)
```

```
FORTRAN
FLDNUM=1
ICOUNT=10
XITEM="GADGET"
CALL VPUTFIELD(COMAREA,FLDNUM,XITEM,ICOUNT,INUM,FLDNXT)
```

```
SPL/PASCAL
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);
```

The calls shown above copy a 10-byte value from an application to field number 1 in the data buffer.

VPUTtype

Copies a numeric value of specified type from an application to a field in the data buffer in memory, converting the value to character set coded external representation.

Syntax

```
VPUTtype {comarea,fieldnum,variable}[,numdigits,decplaces]*
```

Parameters

type

The *type* in VPUTtype indicates that this intrinsic may be specified as:

VPUTINT	converts value to two-byte integer
VPUTDINT	converts value to four-byte integer
VPUTREAL	converts value to four-byte real value
VPUTLONG	converts value to eight-byte long value
*VPUTPACKED	converts packed decimal format to character set coded ASCII; this intrinsic has two extra parameters, <i>numdigits</i> and <i>decplaces</i> . Both are two-byte integer variables that contain the number of digits and number of decimal places, respectively, specified by the COBOL <i>usage is comp-3</i> data declaration.
*VPUTZONED	converts zoned decimal format to character set coded ASCII; has the parameters <i>numdigits</i> and <i>decplaces</i> , which are two-byte integer variables that contain the number of digits and number of decimal places, respectively, specified by the COBOL data declaration.
VPUTYYMMDD	converts six-byte character value to YMD, DMY or MYD

For example:

```
CALL "VPUTINT" USING COMAREA, FIELDNUM, VARIABLE
```

```
CALL "VPUTPACKED" USING  
COMAREA, FIELDNUM, VARIABLE, NUMDIGITS, DECPLACES
```

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF.

VPUTtype

If not already set, the following *comarea* items must be set before calling VPUTtype:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VPUTtype may set the following *comarea* items:

cstatus Set to nonzero value if call unsuccessful.

numerrs May be decremented if new value replaces the value of a field with an error.

fieldnum Two-byte integer variable containing the field number assigned by FORMSPEC to the field in the data buffer to which the value is copied.

variable Variable of specified type in an application that contains the value to be converted to USASCII and copied to a field in the data buffer.

Discussion

Depending on how it is specified, this intrinsic converts integer, double, real, long, packed, zoned or yymmdd values to the external representation and copies 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 copied may be defined as a numeric field (data type NUM[n], IMPn, or DIG) or as a character field (data type CHAR).

NOTE

If you are using VPUTBUFFER in conjunction with an ARB, you do not need to use VPUTtype. VPUTBUFFER performs all the required conversions on the application data in the buffer.

You can use both VPUTBUFFER with an ARB and VPUTtype calls in the same program: the *buffercontrol* setting in the *comarea* that controls ARB processing can be switched on or off for each form.

Note that the field number never changes as long as the field exists. It is not changed when the position of the field in the form is changed, or its length or other characteristics are changed. The field number can only be changed with the batch command, RENUMBER, as described in Section 7. The field number should not be confused the screen order number, which is the position of the field in the data buffer and 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 6-13 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.

VPUTtype

Example

```
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(C(*),F1,I)
```

```
FORTRAN
FIELD=4
ITEM=25
CALL VPUTINT(COMAREA,FIELD,ITEM)
```

```
SPL/PASCAL
INTEGER FIELD,ITEM;
  .
FIELD:=4;
ITEM:=25;
VPUTINT(COMAREA,FIELD,ITEM);
```

The calls shown above convert an integer value of 25 in the application to the external representation and copy it to field 4 in the data buffer in memory.

VPUTWINDOW

Copies a message to the window area of memory.

Syntax

```
VPUTWINDOW {comarea,message,length}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPUTWINDOW:

language

Set to the code identifying the programming language of the calling program.

comarealen

Set to total number of two-byte words in *comarea*.

VPUTWINDOW may set the following *comarea* items:

cstatus

Set to nonzero value if call unsuccessful.

message

Character string containing the message to be copied to the window area of memory.

length

Two-byte integer that specifies the number of bytes in the message. If set to zero, any message in the window is cleared to blanks. The maximum length is 150 bytes, but only 80 can be printable characters.

Discussion

This intrinsic copies the specified message to the window area of memory for later display. Then, a 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* of *comarea*. A message copied by VPUTWINDOW overwrites any previous message in the window area, including any message copied 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.

Example

```
COBOL
MOVE "ENTER ORDERS ON THIS FORM" TO MESSAGE.
MOVE 25 TO MSG-LENGTH.
CALL "VPUTWINDOW" USING COMAREA, MESSAGE, MSG-LENGTH.
```

```
BASIC
310 M1$="ENTER ORDERS ON THIS FORM"
320 L1=25
330 CALL VPUTWINDOW(C(*),M1$,L1)
```

```
FORTRAN
MSG="ENTER ORDERS ON THIS FORM"
LEN=25
CALL VPUTWINDOW(COMAREA,MSG,LEN)
```

```
SPL/PASCAL
BYTE ARRAY MSG(0:24):="ENTER ORDERS ON THIS FORM";
INTEGER LEN;
:
LEN:=25;
VPUTWINDOW(COMAREA,MSG,LEN);
```

The calls shown above copy the message

ENTER ORDERS ON THIS FORM
to the window area of memory.



VREADBATC

Reads contents of current batch record into data buffer in memory.

Syntax

```
VREADBATC {comarea}
```

Parameters

comarea

Must be *comarea* name used when the batch file was opened with VOPENBATC. If not already set, the following *comarea* items must be set before calling VREADBATC:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

recnum Set to the number of the record in the open batch file from which data is to be read (records are numbered from zero).

VREADBATC may set the following *comarea* items:

nfname Set to the name of the form associated with the data read from the batch file (used by VGETNEXTFORM to retrieve this form from forms file).

dbuflen Set to length of data buffer (in bytes) based on length of data read from batch record.

deleteflag Set to TRUE (all ones) if delete flag in batch record indicates record is deleted; set to FALSE (all zeros) otherwise.

cstatus Set to nonzero value if call unsuccessful.

filerrnum Set to file error code if MPE file error.

Discussion

Use of this intrinsic is demonstrated by the browse/modify mode of ENTRY. It enables a user to view the data in the batch file. VREADBATC may also be used to bring the data from a batch file into the data buffer so that it can be retrieved by an application with the VGETBUFFER, VGETFIELD, or VGETtype intrinsics.

VREADBATCH

To display the data just read onto its associated form, VGETNEXTFORM must be called before the call to VSHOWFORM.

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 bytes which is moved to *dbuflen*. The data buffer is not updated if the *deleteflag* has been set.

In order to use VREADBATCH, the batch file must be on a direct-access device and must be created with fixed-length records, *not* variable-length records.

Example

```
COBOL  
CALL "VREADBATCH" USING COMAREA.
```

```
BASIC  
175 CALL VREADBATCH(C(*))
```

```
FORTRAN  
CALL VREADBATCH(COMAREA)
```

```
SPL/PASCAL  
VREADBATCH(COMAREA);
```

The calls shown above read the batch record specified by *recnum*, update the *comarea* according to the batch record information stored with the data, and put the data in the data buffer in memory.

VREADFIELDS

Accepts all user input from an open terminal, including data entered by pressing **ENTER**, or user requests made by pressing a function key. Allows look-ahead form loading.

Syntax

```
VREADFIELDS {comarea}
```

Parameters

comarea

Must be *comarea* named specified when terminal file was opened with VOPENTERM. If not already set, the following *comarea* items must be set before calling VREADFIELDS:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

The following *comarea* items can be set:

lookahead Set to zero to enable look-ahead form loading or to one to disable look-ahead form loading.

showcontrol Set to control the touch feature or to affect the display on data capture devices. The particular settings are described below.

VREADFIELDS may set the following *comarea* values:

numerrs Set to zero.

lastkey Set to code for terminal function key pressed by user.

cstatus Set to nonzero value if call unsuccessful.

Discussion

VREADFIELDS accepts user-entered data when **ENTER** 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. Note that the keyboard is locked after the data is read (the subsequent call to VSHOWFORM unlocks the keyboard). If a function key is pressed, VREADFIELDS sets *lastkey* to the corresponding number: 1 for **F1**, 2 for **F2**, and so forth. The screen is *not* read and data is *not* transferred when a function key is pressed. It is, however, possible to accomplish this by enabling the AUTOREAD feature. To enable

VREADFIELDS

AUTOREAD, a bit must be set in the *termoptions* position of the *comarea*. For further information regarding the AUTOREAD feature, consult the Terminal Communications Area section of Appendix G.

Application programs must supply the code to perform each of the functions that can be requested via a function key. 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 examples of applications using VREADFIELDS).

Normally, there is no time limit for VREADFIELDS, whether the intrinsic is to be terminated by the enter key or by a function key. Timeouts can be enabled in an application by setting bits in the *termoptions* word of the *comarea* and specifying a timeout interval in the *usertime* word of the *comarea*. For more information regarding user timeouts, consult the Terminal Communications Area section of Appendix G.

For any terminal with touch capability, an application activates the touch feature by setting bit 0 of *showcontrol* to one. Then, for each time the user touches a field, VREADFIELDS returns *lastkey* with the field number of the touched field (as a negative number). Refer to Appendix G for the "Supported Terminal Features" list.

For the data capture devices, VREADFIELDS, not VSHOWFORM, displays a form one field at a time and reads entered data, also one field at a time. Editing is performed after data for all the fields in the form has been read. VREADFIELDS sets *lastkey* as follows for these terminals: -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. If a function key is pressed when entering data on the data capture devices, VREADFIELDS returns to the calling program and does not display any remaining fields.

Bits 12 and 11 of the *comarea* item *showcontrol* affect form display on the data capture devices. These bits can be set as follows:

bit 12 = 0	Stop after displaying a form without fields.
1	Do not stop after displaying 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.

A form without fields can be used to display a message to a user. If bit 12 is zero, this type of form is frozen until **ENTER** or a function key is pressed so that the user has time to read the message. If bit 11 is one, a right bracket is displayed as a terminator on all input fields.

VREADFIELDS may also invoke look-ahead form loading for terminals with local form storage using the *comarea* item *lookahead*. If point-to-point data communications is being used, the next form (as defined in the forms file) is loaded into the terminal's local form storage memory while the user is typing in data, before **ENTER** is pressed. If multipoint or X.25 data communications is being used, the next form is loaded after the data has been read, after **ENTER** is pressed. If there is no room in local form storage for the next form, or if there are no available entries in the form storage directory (set with *formstoresize* prior to VOPENTERM and VOPENFORMF), the least recently used form could be purged from local form storage. Look-ahead form loading does not load multiple family members. If a family member of the next form is already in local form storage, this family member is changed into the required form when it is displayed. (Refer to Section 4 for more information on Local Form Storage.)

If the next form does not load successfully, no error is returned. The next form is simply displayed by VSHOWFORM.

VREADFIELDS

Example

COBOL
CALL "VREADFIELDS" USING COMAREA.

BASIC
130 CALL VREADFIELDS(C(*))

FORTRAN
CALL VREADFIELDS(COMAREA)

SPL/PASCAL
VREADFIELDS(COMAREA);

Each of the following calls accepts user input from the terminal, transfers any data to the data buffer, and sets *lastkey*.

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 copies a message to the window area of memory for later display.

Syntax

```
VSETERROR {comarea,fieldnum,message,msglen}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VSETERROR:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VSETERROR may set the following *comarea* items:

numerrs Contains number of fields in form with errors; incremented by VSETERROR.

cstatus Set to nonzero value if call unsuccessful.

fieldnum Two-byte integer variable containing the field number or screen order number of the data field to be flagged for error. If it is the screen order number, it must be a negative number representing the position of the field on the form.

message Character string containing an error message to be copied to the window area of memory for subsequent display by VSHOWFORM.

length Two-byte integer containing the length in bytes 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.

Discussion

This intrinsic 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 a field has an error detected by a prior call to VFIELDEDITS, a call to VSETERROR for that field does not increment *numerrs*. However, if the touch

VSETERROR

feature is enabled, VSETERROR *toggles* the error flag for a specified field to clear, if it was already set, and decrements *numerrs*. Refer to "Coding the Touch Feature" in Appendix E.

If this is the first field in the form (in screen order) that has an error, the specified message is copied 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. Using the screen order number is preferable because if the fields in a form are rearranged, no modification to an application is necessary.

Example

```
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$="THIS FIELD IS REQUIRED"
250 CALL VSETERROR(C(*),F1,M$,L1)
```

```
FORTRAN
FF=3
ML=22
MSG="THIS FIELD IS REQUIRED"
CALL VSETERROR(COMAREA,FF,MSG,ML)
```

```
SPL/PASCAL
INTEGER FF,ML;
BYTE ARRAY MESSAGE(0:21);="THIS FIELD IS REQUIRED";
.
FF:=3;
ML:=22;
VSETERROR(COMAREA,FF,MESSAGE,ML);
```

The examples above 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.

VSETERROR

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(C(*),F1,M$,L1)
```

FORTRAN

```
FF=3  
ML=-1  
CALL VSETERROR(COM1,FF,MSG,ML)
```

SPL/PASCAL

```
FIELD:=3;  
LEN:=-1;  
VSETERROR(COM1,FIELD,MESSAGE,LEN);
```

VSETKEYLABEL

Allows for temporarily setting, programmatically, a new label for a function key.

Syntax

```
VSETKEYLABEL {comarea,formorglob,keynum,label}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VSETKEYLABEL:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VSETKEYLABEL may set the following *comarea* items:

cstatus Set to nonzero value if call is unsuccessful.

formorglob Integer specifying which type of label is to be temporarily set.

0 - Set global label.

1 - Set current form label.

keynum Integer from 1 to 8 corresponding to function key to be set.

label A byte array containing the text for the label; must be 16 bytes long.

Discussion

VSETKEYLABEL is only a temporary setting of a new label for an individual function key. Use of this intrinsic does not change the label definition made in FORMSPEC. Note only one function key can be set with this intrinsic. The *labeloption* must be set to one prior to VOPENFORMF.

The temporary label is displayed after the next call to VSHOWFORM. If the temporary label is global, it remains 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 is replaced when the next form is retrieved or when a new current form label is set.

If no global or current form labels have been defined with FORMSPEC or no labels have been set with VSETKEYLABELS, the key label buffers are cleared before the label being defined with this intrinsic is set.

VSETKEYLABEL

Example

```
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
LABEL="LABEL      1      "
CALL VSETKEYLABEL (COMAREA,FORMORGLOB,KEYNUM,LABEL)
```

```
SPL/PASCAL
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.

Syntax

```
VSETKEYLABELS {comarea,formorglob,numoflabels,labels}
```

Parameters

comarea Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VSETKEYLABELS:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VSETKEYLABELS may set the following *comarea* items:

cstatus Set to nonzero value if call is unsuccessful.

formorglob Integer specifying which type of labels are to be temporarily replaced.

0 - Replace global labels.

1 - Replace current form labels.

numoflabels Integer from 0 through 8 indicating how many labels are to be temporarily set, where 0 indicates that the labels defined in FORMSPEC should be used.

labels A byte array in which the labels are defined. The length of the array must be at least *numoflabels* * 16.

Discussion

VSETKEYLABELS is only a temporary setting of new labels for the function keys. Use of it does not change the label definitions made in FORMSPEC. The *labeloption* must be set to one prior to VOPENFORMF.

The temporary labels are displayed after the next call to VSHOWFORM. If the temporary labels are global, they remain active until the forms file is closed or replaced by new global labels. If the temporary labels are current form labels, they are replaced when the next form is retrieved or when new current form labels are set.

VSETKEYLABELS

If no global or current form labels have been defined with FORMSPEC or no labels have been set with a prior call to VSETKEYLABELS, the key label buffers are cleared before the labels currently being defined are set.

Example

```
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 FORMORGLOBAL,NUMLABELS
CHARACTER*32 LABELS
FORMORGLOBAL=1
NUMLABELS=2
LABELS="LABEL 1 LABEL 2 "
CALL VSETKEYLABELS(COMAREA,FORMORGLOBAL,NUMLABELS,LABELS)
```

```
SPL/PASCAL
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);
```

VSETLANG

The VSETLANG intrinsic specifies the native language to be used with an international forms file. For more information on Native Language Support, see Section 8.

Syntax

```
VSETLANG {comarea,langnum,errorcode}
```

Parameters

<i>comarea</i>	Must be <i>comarea</i> name specified when the forms file was opened with VOPENFORMF. If not already set, the following <i>comarea</i> items must be set before calling VSETLANG:
<i>comarealen</i>	Set to the total number of two-byte words in <i>comarea</i> .
	VSETLANG may set the following <i>comarea</i> items:
<i>cstatus</i>	Set to nonzero value if call is unsuccessful.
<i>langnum</i>	A two-byte integer, <i>passed by value</i> , containing the native language ID number of the forms file language to be used by VPLUS/V.
<i>errorcode</i>	Two-byte integer to which the error code is returned.

Discussion

This intrinsic sets the native language to be used by VPLUS/V at run time for an international forms file. The forms file must be opened before calling VSETLANG. Otherwise, *cstatus* returns a nonzero value.

If VSETLANG is called to set the native language ID for a language-dependent or unlocalized forms file, an error code of -1 will be returned to *errorcode*. For international forms files, both *cstatus* and *errorcode* return a value of zero and the forms file is processed with the native language ID specified in *langnum*.

Example

COBOL
CALL "VSETLANG" USING COMAREA, \LANGNUM\, ERRORCODE

BASIC
Pass by value parameters not supported.

FORTRAN
CALL VSETLANG (COMAREA, LANGNUM, ERRORCODE)

SPL/PASCAL
VSETLANG(COMAREA, LANGNUM, ERRORCODE)

VSHOWFORM

Displays on the terminal screen the current form from local form storage or from the form definition buffer, any data in the data buffer, and any messages from the window buffer. Displays any labels from the key label buffer.

Syntax

```
VSHOWFORM {comarea}
```

Parameters

comarea Must be the *comarea* name specified when the terminal file was opened with VOPENTERM. If not already set, the following *comarea* items must be set before calling VSHOWFORM:

cstatus Set to zero

comarealen Set to total number of two-byte words in *comarea*.

windowenh Set to window enhancement code; may be set before call to code for nondefault 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.)

The following *comarea* item can be set:

showcontrol Set to change VSHOWFORM control options. The particular settings are described below.

VSHOWFORM may set the following *comarea* values:

cstatus Set to nonzero value if call unsuccessful.

filerrnum Set to file error code if MPE file error.

(Note that *showcontrol* is not cleared by VSHOWFORM.)

Discussion

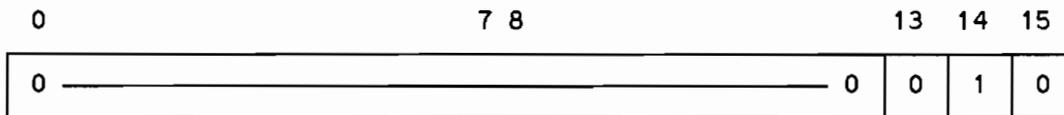
VSHOWFORM displays, on an open terminal screen, the form currently stored as a form image in the terminal's local form storage or in the form definition area of memory. Any enhancements specified for the form are used for the display. Data currently in the data buffer in memory is moved to the appropriate 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. Also, any current form or global function key labels in the key label buffer are displayed.

Performance Optimization

In order to optimize VSHOWFORM performance, only changed information is written to the terminal. Thus, if the window has not been changed by VPUTWINDOW or VSETERROR 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 the *comarea* item *showcontrol* as follows:

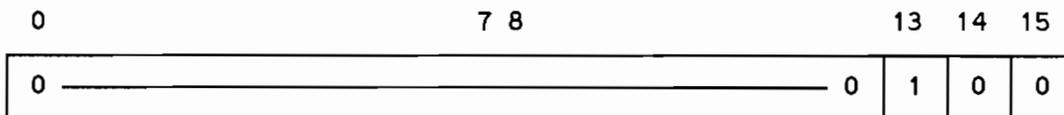
- bit 15 = 1 Force form to be written to the terminal screen.
- 14 = 1 Force data and field enhancements to be written to the terminal screen.
- 13 = 1 Force window line to be written to the terminal screen.

Depending on the bits set, VSHOWFORM writes a form or data or the window to the terminal whether or not it has changed. Anything that has changed is always written to the terminal regardless of *showcontrol*. Any combination of these bits may be set. For example, if you want to force a write of all data:



showcontrol = octal 2

Or, if you want to force the window to be written:



showcontrol = octal 4

Display and Local Form Storage

Prior to display on terminals with local form storage, the form can be loaded into the terminal with any of the following methods:

- by a call to VLOADFORMS,
- by the previous call to VREADFIELDS if look-ahead form loading is enabled, or
- by VSHOWFORM if preload is enabled.

A prior call to VGETNEXTFORM reads the form into the form definition area of memory prior to display by VSHOWFORM. The form definition includes the form image, any 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.

VSHOWFORM

The data buffer may contain data as a result of initialization by VINITFORM, retrieval of user-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, either from an application with VPUTBUFFER or VPUTtype, or from a batch file with VREADBATCH.

The window contains any message set by VSETERROR or VPUTWINDOW.

Controlling the Keyboard

As soon as the form is displayed, VSHOWFORM normally enables the keyboard so the user can enter data. The next call is usually to VREADFIELDS, which locks the keyboard after the entered data is read. In case of consecutive calls to VSHOWFORM, the following *showcontrol* bit can be set:

- bit 10 = 0 Enable the keyboard.
- 1 Do not enable the keyboard.

The scenario is as follows:

- the last VREADFIELDS call locked the keyboard,
- for each of a series of consecutive calls to VSHOWFORM; set bit 10 of *showcontrol* to one so the keyboard is not enabled,
- for the last of the consecutive calls to VSHOWFORM, set bit 10 to zero to enable the keyboard.

This ensures that keystrokes do not change a form while it is being displayed. Consecutive calls to VSHOWFORM could be used to display a form with no fields, append another form to it, and display this second form without doing a read in between.

Controlling Preload of Forms

On terminals with local form storage, VSHOWFORM first determines if the current form is already in the terminal. If the form is in local storage, it is displayed from local storage. If the form is not in local storage, it is preloaded into the terminal from the form definition area of memory depending on *showcontrol*, which can be set as follows:

- bit 9 = 0 Do not preload the form.
- 1 Preload the form.

If bit 9 is zero, the form is written directly from the form definition area of memory to the terminal screen. If bit 9 is one, the form is preloaded into local storage and then displayed from local storage. One or more forms could be purged from local storage if there is not room for the form that is being loaded. Note that purging could occur even when *lookahead* is set to one.

Controlling the Touch Feature

On terminals with the touch feature, touch can be enabled or disabled with *showcontrol*, which can be set as follows:

- bit 0 = 0 Do not enable the touch feature.
- 1 Enable the touch feature.

Controlling the Bell

By default (*termoptions* bit 15 is set to zero), VSHOWFORM sounds the bell if neither the screen nor the window line have been redisplayed. The bell can be suppressed in all cases by setting the *termoptions* bit to one.

Example

```
COBOL
CALL "VSHOWFORM" USING COMAREA.
```

```
BASIC
120 CALL VSHOWFORM(C(*))
```

```
FORTRAN
CALL VSHOWFORM(COMAREA)
```

```
SPL/PASCAL
VSHOWFORM(COMAREA);
```

The calls shown above display a form with optional data and enhancements on the terminal screen opened with the *comarea*, COM1.

VUNLOADFORM

Unloads the specified form from local form storage memory.

Syntax

```
VUNLOADFORM {comarea,whichform}
```

Parameters

comarea

Must be *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VUNLOADFORM:

cstatus Set to zero.

language Set to the code identifying the programming language of the calling program.

comarealen Set to total number of two-byte words in *comarea*.

VUNLOADFORM may set the following *comarea* values:

cstatus Set to nonzero value if call unsuccessful.

whichform

The name of the form to be removed from local form storage. Each name can be up to 15 characters and is stored in a 16-byte character array with a one byte filler that is not part of the name.

Discussion

This intrinsic is used on terminals having local form storage. VUNLOADFORM purges the form named by the *whichform* parameter from terminal local form storage memory. Note that the terminal keyboard may be locked briefly while VUNLOADFORM verifies whether or not the form was purged. In case keys are pressed during this time, the terminal beeps to indicate that the keystrokes and/or entered data are lost. You can avoid this by not calling VUNLOADFORM between calls to VSHOWFORM and VREADFIELDS.

Example

```
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 VUNLOADFROM(C[*],W$)
```

```
FORTRAN
CHARACTER*16 WFORM
WFORM="FORMA           "
CALL VUNLOADFORM(COMAREA,WFORM)
```

```
SPL/PASCAL
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.

Syntax

```
VWRITEBATCH {comarea}
```

Parameters

comarea Must be *comarea* name specified when the batch file was opened with VOPENBATCH. If not already set, the following *comarea* items must be set before calling VWRITEBATCH:

<i>cstatus</i>	Set to zero.
<i>comarealen</i>	Set to total number of two-byte words in <i>comarea</i> .
<i>recnum</i>	Set to record number in batch file to which data is to be written (record numbers start with zero). The <i>recnum</i> is initialized to zero by VOPENBATCH (or to the next sequential record number if the file is an existing file).
<i>deleteflag</i>	Set to TRUE (all ones) if record is to be deleted.

(The *cfname* and *dbuflen* should be set by a prior call to VGETNEXTFORM.)

VWRITEBATCH may set the following *comarea* values:

<i>cstatus</i>	Set to nonzero value if call unsuccessful.
<i>filerrnum</i>	Set to file error code if MPE file error.
<i>numrecs</i>	Incremented each time a new record is written; decremented if record is deleted.

Discussion

VWRITEBATCH writes the contents of the data buffer to the record specified by *recnum* in an open batch file. (The *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 may be used in both the data collection and 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*.

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 ones) by an application. 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 intrinsic.

In ENTRY, a function key pressed by the user not only determines which record is to be viewed, but also specifies which record is to be deleted.

VWRITEBATCH

Example

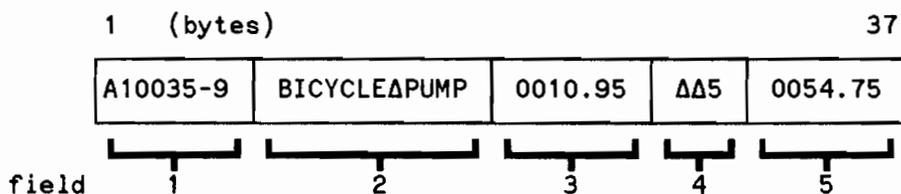
COBOL
CALL "VWRITEBATCH" USING COMAREA.

BASIC
165 CALL VWRITEBATCH(C(*))

FORTRAN
CALL VWRITEBATCH(COMAREA)

SPL/PASCAL
VWRITEBATCH(COMAREA);

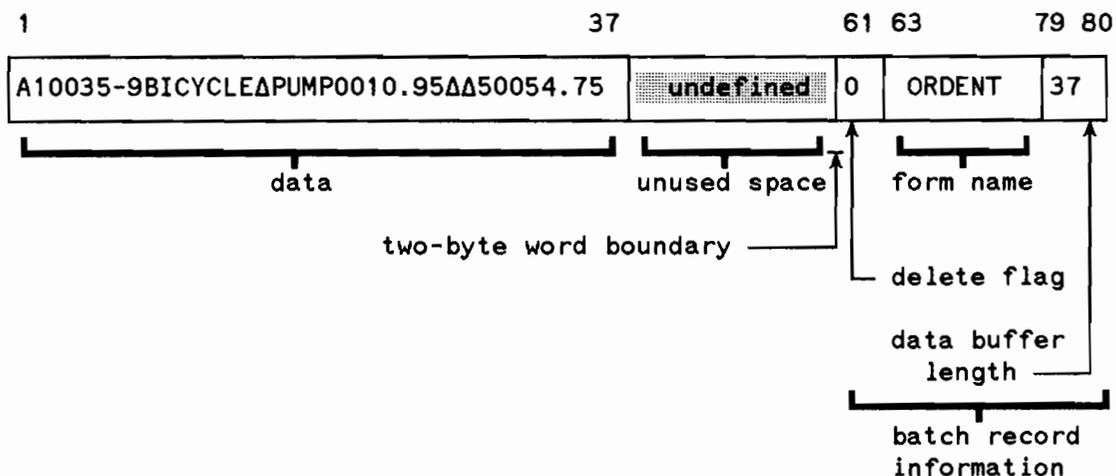
The calls shown above write the contents of the data buffer to the batch record identified by *recnum*. Assume the following data is in the data buffer and that it was entered on form ORDENT. (Note that the data entered on separate fields of a form is concatenated in the data buffer, with no separators.)



Assume the *comarea* is set as follows:

recnum = 5
cfname = ORDENT
dbuflen = 37

In addition, assume the batch file opened by VOPENBATCH has fixed-length records, 80 bytes long. The call to VWRITEBATCH writes the following record as the sixth record in the batch file:



FORMSPEC in batch mode allows you to manage a forms file without tying up your terminal. With this feature, forms may be deleted from or copied to a forms file, compiled, and listed on a printer by accepting commands from a job stream or from a disc file or I/O device.

Compiling Forms File in Batch Mode

Compiling a forms file can be very time-consuming, especially if there are many forms, fields, and field specifications defined in the file. By using FORMSPEC in batch mode, long compilations can be executed from a job stream. You do not have to wait for compilation to complete in order to use your terminal for other purposes.

Updating Forms in Batch Mode

A forms file can be updated systematically by using FORMSPEC in batch mode. Entire forms can be copied to or deleted from a forms file, field attributes may be updated, and function key labels may be created and updated. Individual screens, save fields, and some global information, however, cannot be defined in batch mode. Also, save fields cannot be copied from one forms file to another.

Listing Forms Files in Batch Mode

By using the batch mode LIST command, forms can be listed on the list device from a streamed job. This means forms file listings may be automated in a job that updates a forms file. For example, a job that updates a forms file could also print all or specific forms for future reference. The form listings may also be directed to a disc file to be examined.

Relating Forms in Batch Mode

Form family relationships can be established using the RELATE command in batch mode. The RELATE command requires that the forms exist and their screen designs be identical. Furthermore, the forms to be related must be distinct forms and the proposed child form cannot already be a member of a family of forms.

INVOKING FORMSPEC IN BATCH MODE

FORMSPEC in batch mode can be invoked from within a job stream or from within a session. When invoked, FORMSPEC can accept batch mode commands from a command file or from \$STDIN.

To invoke FORMSPEC to accept batch mode commands from a command file, use the MPE `:RUN` command with the `INFO=filename` parameter, where *filename* is the name of the command file. For example:

```
:RUN FORMSPEC.PUB.SYS;INFO="COMMANDS.PUB.MFG"
```

will execute FORMSPEC in batch mode using `COMMANDS.PUB.MFG` as the command file. Remember, before you can invoke FORMSPEC in batch mode to accept commands from a command file, you must first create the command file using a text editor and enter the batch mode commands into the file. (See the end of this section for a sample of a command file.)

The `INFO=filename` parameter directly invokes batch mode. In addition, FORMSPEC will always execute in batch mode within a job stream. When the `INFO=filename` parameter is omitted within a job stream, \$STDIN becomes the command file by default and commands are accepted from within the job stream. In order to use \$STDIN as the command file within a session, the `INFO=filename` parameter is required. For example:

```
:RUN FORMSPEC.PUB.SYS;INFO="$STDIN"
```

In response to the greater than (>) prompt, enter the batch mode commands. Although this usage of batch mode will not free up a terminal, users will have the ability to react to any error conditions which may occur.

When FORMSPEC detects an error in batch mode, the error message is sent to the output file (\$STDLIST by default). The number of errors can be checked with `FORMSPECERRORJCW`; number of warnings can be checked with `FORMSPECWARNJCW`. An error will cause one of the following actions:

- If any batch mode command except the `FILE` command causes an error, then FORMSPEC will skip that command and continue to the next command.
- If a `FILE` command caused the error or the error occurred because FORMSPEC did not recognize a batch mode command, then all batch mode commands, except the `FORMS` command, will be skipped until the next `FILE` or the `EXIT` command is encountered.

Batch mode output (errors, warnings, or messages) will normally be directed to \$STDLIST. However, output can be redirected to another file by using the MPE `:FILE` command to redefine the formal file designator, `FORMOUT`. For example, since \$STDLIST in a session is the terminal, you may want to redirect output to the line printer as follows:

```
:FILE FORMOUT;DEV=LP
```

When the output file is redirected to a disc file or the list device, the batch mode commands will also be echoed to the output file as they are executed. This will help to determine the cause of any errors that may occur.

BATCH MODE COMMANDS

The following pages show the syntax and description of the batch mode commands that FORMSPEC accepts in a command file or job stream. Table 7-1 gives a summary of the batch mode commands, a brief description of their functions, and the interactive counterpart for each command.

Note that each command must be entered on a separate line and a command may not carry over to the next line. Leading blanks may be used for indentation. You can also use blank lines between commands to make the command file easier to read. Comments may be inserted anywhere in a command file. They may be more than one line long, but they must be delimited by double arrows <<*comment*>>.

Table 7-1. FORMSPEC in Batch Mode Commands

Command	Function	Interactive Counterpart
COMPILE	Compiles the current forms file.	Selection (X) on the FORMSPEC Main Menu.
COPY	Copies forms within the current forms file, or from another file to the current forms file.	Selection (C) on the FORMSPEC Main Menu.
DELETE	Deletes a form from the current forms file.	Selection (D) on the FORMSPEC Main Menu.
EXIT	Terminates FORMSPEC in batch mode.	Function key (F6) in any menu.
FIELD	Updates field attributes.	Field attributes specified on the FORMSPEC Field Menu.
FILE	Declares the current forms file.	Forms file name on the Forms File Menu.
FKLABELS	Creates or updates function key label strings.	FORMSPEC form-specific Function Key Menu.
FORMS	Prints summary information on the current forms file as well as for each or all the forms therein.	No interactive counterpart for this command.
LIST	Lists a form or all the forms on the standard list device.	Selection (L) on the FORMSPEC Main Menu.

Table 7-1. FORMSPEC in Batch Mode Commands, cont'd

Command	Function	Interactive Counterpart
RELATE	Establish family relation between two forms.	Selection R on the FORMSPEC Main Menu.
RENUMBER	Renumbers fields within a form.	No interactive counterpart for this command.
SELECTLANG	Updates native language specifications.	FORMSPEC Terminal/Language Menu.
SELECTTERM	Updates terminal and device specifications.	FORMSPEC Terminal/Language Menu.
ARB Commands	Function	Interactive Counterpart
ARBTOSCREEN	Set data type conversion defaults from ARB to screen.	Select defaults on FORMSPEC Data Type Conversions Menu.
SCREENTOARB	Sets data type conversion rules from screen to ARB.	Select defaults on FORMSPEC Data Type Conversions Menu.
DELARB	Deletes an ARB.	Select DEL on the FORMSPEC ARB Menu.
GENARB	Generates an ARB from a form.	Select GEN on the FORMSPEC ARB Menu.
ADDARBFIELD	Adds a field to the ARB.	Select ADD on the FORMSPEC Restructure ARB Menu.
DELARBFIELD	Deletes a field from the ARB.	Select DEL on the FORMSPEC Restructure ARB Menu.
MODARBFIELD	Modifies the attributes of an ARB.	Select MOD on the FORMSPEC ARB Menu.
MOVEARBFIELD	Moves a field or group of fields around on the ARB.	Select MOVE on the FORMSPEC Restructure ARB Menu.
RENAMEARBFIELD	Changes the name of an ARB field.	Select RENAME on the FORMSPEC Restructure ARB Menu.

Compiles the current forms file.

Syntax

```
COMPILE [INTO fastfile]
```



Parameters

fastfile

is the name of the fast forms file to which all code records are copied. Since a fast forms file contains only records needed by VPLUS/V intrinsics at run-time, it may be advantageous to compile the forms file into a fast forms file. If *fastfile* already exists and it is a fast forms file, it is replaced with this version. If *fastfile* already exists and it is *not* a fast forms file, an error occurs. The current forms file still compiles, but the code records are not copied to a fast forms file. If *fastfile* does not exist, it is created for you as a fast forms file.

Discussion

Normally the COMPILE command only compiles the forms that have been modified in the current forms file. However, when certain global attributes (such as save fields) are modified, all forms in the current forms file are compiled. The fast forms file, if specified, contains code records for all forms in the current forms file.

Example

```
COMPILE
```

compiles the current forms file.

```
COMPILE INTO INVFFORM.PUB
```

compiles the current forms file and creates a fast forms file named INVFFORM in the PUB group.

COPY

Copies a form within the current forms file, or copies a form from another forms file to the current forms file.

Syntax

```
COPY form {TO newform  
           IN file [TO newform]}
```

Parameters

- form* is the name of the form to be copied.
- newform* is the name of the new form in the current forms file to which *form* will be copied. The *newform* cannot already exist in the forms file. If *newform* already exists, an error will occur and no copying will be done. If *TO newform* is omitted, the new form created in the current forms file will have the same name as the form it was copied from.
- file* is the name of the forms file, if *form* is not in the current forms file.

Example

```
COPY PARTS TO PARTS2
```

copies FROM the form PARTS in the current forms file TO the new form PARTS2.

```
COPY PARTS IN INVFORMS TO PARTS2
```

copies FROM the form PARTS in the forms file INVFORMS TO the new form PARTS2 in the current forms file.

```
COPY PARTS IN INVFORMS
```

copies FROM the form PARTS in the forms file INVFORMS TO the new form PARTS in the current forms file.

Removes a form from the current forms file.

Syntax

```
DELETE form
```

Parameters

form

is the name of the form to be removed from the current forms file. If *form* does not exist, FORMSPEC will issue a warning and proceed to the next command.

Example

```
DELETE ORDER2
```

removes the form ORDER2 from the current forms file.

EXIT

Terminates FORMSPEC in batch mode.

Syntax

```
EXIT
```

Discussion

The EXIT command is optional. FORMSPEC will also terminate after the last command in the command file is executed.

Updates field attributes.

Syntax

$\text{FIELDform } \left\{ \begin{array}{l} \text{fieldtag} \\ \text{oldfieldname} \end{array} \right\} \left\{ \begin{array}{l} \text{newfieldname} \\ \text{,enhancement} \\ \text{,fieldtype} \\ \text{,fedatatype} \\ \text{,initialvalue} \end{array} \right\}$
--

Parameters

<i>form</i>	the name of an existing form within the current forms file.
<i>fieldtag</i>	the tag assigned to the field in form screen design.
<i>oldfieldname</i>	the name currently assigned to the field in the field menu.
<i>newfieldname</i>	replacement field name.
<i>enhancement</i>	replacement display enhancement code.
<i>fieldtype</i>	replacement field type.
<i>datatype</i>	replacement data type.
<i>initialvalue</i>	replacement initial value. Leading blanks are treated as part of the <i>initialvalue</i> . If \$EMPTY or \$empty is specified, the existing initial value is cleared.

Discussion

The first two parameters of this command identify the field to be updated. The remaining parameters are the actual updates. The field menu update parameters (*newfieldname ... initialvalue*) may be longer than the corresponding number of character positions available on the Field Menu in interactive FORMSPEC, in which case the input parameter is truncated.

EXAMPLE

```
FIELD PARTS a_b_c A_B_C_CODE,NONE,R,,D
```

Updates field menu selections for field *a_b_c* of form PARTS. The field name is set to *A_B_C_CODE*, the field display enhancement code is set to *NONE* (for no enhancements), the field type value is set to *R* (for required), and the data type value is unchanged. The initial value for the field is set to the character *D*.

FILE

Specifies the current forms file to be used.

Syntax

```
FILE file
```

Parameters

file is the name of the forms file to be used by subsequent batch mode commands. The *file* is referred to as the current forms file.

Discussion

The FILE command must precede all other commands in a command file or job stream. This command opens the specified forms file to be operated on by the other batch mode commands. Only one forms file can be modified at a time (only one current forms file at a time). However, more than one forms file can be operated on in one command file or job stream as long as a FILE command precedes any operations on that file. Note that a FILE command specifying a new current forms file closes the previous current forms file and opens the new current forms file. Termination of FORMSPEC automatically closes the current forms file.

If the file specified does not already exist, it is created. A message indicating that the file was created is issued.

Example

```
FILE INVFORMS.PUB
```

opens the forms file INVFORMS in the PUB group to be operated on by other batch mode commands.

Updates function key labels.

Syntax

```
FKLABELS form [\keylabel ... \keylabel]
```

Parameters

form is the name of an existing form within the current forms file.

keylabel literal which is to appear in the function key label. If \$EMPTY is entered as the first six characters of *keylabel*, the existing key label is cleared.

Discussion

The FKLABELS command creates (if none exist) or updates function key labels for the specified form. The first 16 characters of the *keylabel* literal are used for the label. Only the first eight *keylabels* are used.

Example

```
FKLABELS STAFF_SALARY \\ Start Over\\\$EMPTY\\ Exit  
Planner
```

Updates labels for function keys 2, 5 (which is cleared), and 8 in form STAFF_SALARY. Existing labels for function keys 1,3,4,6, and 7 are not updated.

FORMS

Prints a summary listing for the forms in the current forms file.

Syntax

```
FORMS [form]
      [@ ]
```

Parameters

- form* The name of an existing form within the current forms file for which summary information is to be printed. If omitted, a summary of the current forms file is printed.
- @ The commercial at sign (@) specifies that summary information for all forms, as well as the current forms file, is printed.

Discussion

The FORMS command prints the name of each form, the number of fields in the form, the number of bytes in the screen image, the number of children that the form has, and the name of the form's parent form, if any. If a form name (*form*) is included with the FORMS command, an extended summary is generated for the form. The extended summary includes all of the form information listed above. In addition, for each field in the form, the FORMSPEC assigned field number, data buffer offset (in bytes, one relative), and field length (in bytes) is printed. If the commercial at sign (@) is used for *form*, an extended summary is generated for each form in the forms file. See the examples below for samples of the three types of summary listings available.

Although all FORMSPEC output is normally directed to \$STDLIST, you may want to redirect this output separately to the line printer. To do so, use the MPE :FILE command to redefine the formal file designator, FORMOUT, as follows:

```
:FILE FORMOUT;DEV=LP
```

The FORMS command prints a maximum of 60 lines per page. This maximum can be changed by setting the JCW

```
FORMSPECLINESPERPAGE
```

See Appendix E for more information.

The FORMS command is a very useful batch mode command. When FORMS is used after the forms file has been updated, the listing can be used to quickly check that the resulting forms in the file are the correct ones.

This command also provides useful reference information when coding (or modifying) an application data buffer that has to match the layout of the fields in a form.

Example

FORMS

prints a summary of the current forms file, SYSADMIN in this example:

Forms File: SYSADMIN.ACCTG.FINANC Page 1
 File last modified: WED, MAY 14, 1986, 1:58 PM
 File last compiled: WED, MAY 14, 1986, 1:58 PM

FORM NAME	NUMBER LARGEST		NEXT FORM	NUMBER OF:		
	OF	FIELD		BYTES IN	CHILDREN	PARENT FORM
FORM NAME	FIELDS	NUMBER	NEXT FORM	SCREEN	CHILDREN	PARENT FORM
ACCOUNT_MGRS	5	5	\$HEAD	114	0	
SYSTEM_USERS	5	5	\$HEAD	98	0	

Example

FORMS SYSTEM_USERS

prints a summary of the form specified, SYSTEM_USERS in this example:

Forms File: SYSADMIN.ACCTG.FINANC Page 1
 File last modified: WED, MAY 14, 1986, 1:58 PM
 File last compiled: WED, MAY 14, 1986, 1:58 PM

FORM NAME	NUMBER LARGEST		NEXT FORM	NUMBER OF:		
	OF	FIELD		BYTES IN	CHILDREN	PARENT FORM
FORM NAME	FIELDS	NUMBER	NEXT FORM	SCREEN	CHILDREN	PARENT FORM
SYSTEM_USERS	5	5	\$HEAD	98	0	

Forms File: SYSADMIN.ACCTG.FINANC Page 2

COMPILED FORM NAME	SCREEN LENGTH (bytes)	BUFFER LENGTH (bytes)	FIELD NAME	FIELD NUMBER	LENGTH (bytes)	BUFFER OFFSET
SYSTEM_USERS	168	33	**** Compiled for: HP264X			
			USER_NAME	1	8	0
			ACCOUNT_NAME	2	8	8
			HOME_GROUP	3	8	16
			PORT_NUMBER	4	4	24
			PHONE_EXTENSION	5	5	28

**** End of compiled forms listing. ****

FORMS

Example

FORMS @

prints a summary of the current forms file, (SYSADMIN) as well as the summary information for each form in the forms file, two in this example:

Forms File: SYSADMIN.ACCTG.FINANC Page 1
 File last modified: WED, MAY 14, 1986, 1:58 PM
 File last compiled: WED, MAY 14, 1986, 1:58 PM

FORM NAME	NUMBER OF FIELDS	LARGEST FIELD NUMBER	NEXT FORM	NUMBER OF BYTES IN SCREEN	NUMBER OF CHILDREN	PARENT FORM
ACCOUNT_MGRS	5	5	\$HEAD	114	0	
SYSTEM_USERS	5	5	\$HEAD	98	0	

Forms File: SYSADMIN.ACCTG.FINANC Page 2

COMPILED FORM NAME	SCREEN LENGTH (bytes)	BUFFER LENGTH (bytes)	FIELD NAME	FIELD NUMBER	LENGTH (bytes)	BUFFER OFFSET
ACCOUNT_MGRS	155	68	**** Compiled for: HP264X			
			ACCOUNT_NAME	1	8	0
			MGR_LAST_NAME	2	20	8
			MGR_FIRST_NAME	3	15	28
			DEPARTMENT	4	20	43
			PHONE_EXTENSION	5	5	63
SYSTEM_USERS	168	33	**** Compiled for: HP264X			
			USER_NAME	1	8	0
			ACCOUNT_NAME	2	8	8
			HOME_GROUP	3	8	16
			PORT_NUMBER	4	4	24
			PHONE_EXTENSION	5	5	28

**** End of compiled forms listing. ****

Number of errors = 0 Number of warnings = 0

Lists a specified form or all forms on the list device.

Syntax

```
LIST [form]
```

Parameters

form is the name of the form in the current forms file to be listed. When *form* is not specified, all forms in the current forms file are listed.

Discussion

The LIST command can be used to provide documentation for a forms file. The forms are usually listed on the line printer. The LIST command can also be used to list forms to a disc file to be examined. To do so, use the MPE :FILE command to redefine the formal file designator FORMLIST as shown below:

```
:FILE FORMLIST=MYFILE;DEV=DISC;SAVE;NOCCTL
```

will list forms to the disc file, MYFILE. The SAVE keyword specifies that MYFILE be saved as a permanent file. The NOCCTL keyword indicates that no carriage control characters are written to the file. (See the *MPE Commands Reference Manual* (30000-90009), for more information on the MPE:FILE command.)

The LIST command prints a maximum of 60 lines per page. This maximum may be changed by setting the JCW

```
FORMSPECLINESPERPAGE
```

See Appendix E for more information.

Example

```
LIST
```

lists all forms in the current forms file on the list device.

```
LIST PARTS
```

lists the form PARTS in the current forms file on the list device.

RELATE

Creates a family relationship between two existing forms.

Syntax

```
RELATE childform TO parentform
```

Parameters

childform is the name of the form which is to become the child form. It must already exist, and not be related to any other form.

parentform is the name of the form which is to become the parent form. It must already exist, and have the identical form design as the *childform* specified with the *childform* parameter.

Example

```
RELATE FORM1 TO FORM2
```

Establishes a family relationship between FORM1 and FORM2 where FORM1 is the child form, and the FORM2 is the parent form.

Renumber fields to screen order.

Syntax

```
RENUMBER form
```

Parameters

form is the name of the form containing the fields to be renumbered.

Discussion

The fields in the form specified by *form* are renumbered in accordance with screen order, left to right, top to bottom. If your application depends on field numbers, be sure to update your application to reflect the changed field numbers. If the fields are already numbered in screen order, a message is returned and the fields are not renumbered.

For family forms, the parent form must be specified; all family members are then automatically renumbered. Again, if your application depends on field numbers, be sure to update your application, in this case, for all family members. If you specify the name of a child form, the fields are not renumbered. Instead, a message is returned stating the form was not renumbered because it is a child form and indicating the name of the parent form.

Example

```
RENUMBER FORM1
```

Renumbers the fields in FORM1 in screen order.

SELECTLANG

Updates native language specification.

Syntax

```
SELECTLANG [formsfilelangID] [,FORMSPEClangID]
```

Parameters

formsfilelangID Set forms file native language specification.

FORMSPEClangID Set FORMSPEC native language specification.

Discussion

Only the first three characters of each parameter are used. Refer to "Setting Language ID Number" in Section 8 for more information. At least one parameter must be specified.

Example

```
SELECTLANG 8
```

Native language ID 8, German, is specified for the forms file.

```
SELECTLANG ,0
```

Native language ID 0, Native-3000, is specified for FORMSPEC. The native language specified for the forms file is retained.

Updates forms file terminal or device specifications.

Syntax

```
SELECTTERM [262xand239x][,color][,264x][,307x]
```

Parameters

<i>262xand239x</i>	Configure forms file for the HP 262X and HP 239X terminal families; also includes the HP 150.
<i>color</i>	Configure forms file for color terminals.
<i>264x</i>	Configure forms file for the HP 264X terminal family. Specifying yes (the value Y or y) enables function key labeling.
<i>307x</i>	Configure forms file for the data capture device family. Specifying eight (the value 8) enables support of the extended features of the HP 3081A.

Discussion

The parameters may be any valid non-blank character to select the terminal family. Only the first character of each parameter is used.

At least one parameter must be specified. Providing a USASCII blank as the specification for a given device/terminal family does not change the existing specification. Specifying \$EMPTY or \$empty clears an existing specification.

Example

```
SELECTTERM x,,y
```

Forms file is configured for HP 262X/239X terminals, the HP 150, and HP 264X terminals, with HP 264X function key labeling enabled. Existing color specification, if any, is retained.

```
SELECTTERM ,x,$EMPTY,8
```

Forms file is configured for color terminals and HP 307x terminals, with HP 3081A features enabled. HP 264X specification is cleared and existing HP 262X/239X/150 specification, if any, is retained.

ARB BATCH MODE COMMANDS

The following batch mode commands support the ARB feature. They are based on the ARB Menu options provided in interactive FORMSPEC. The commands are summarized below, and each one is described in detail in the rest of this section.

NOTE

In the ARB commands, *destination* is:

[BEFORE fieldname] or [AFTER fieldname]

The data type conversion commands are:

ARBTOSCREEN and ARBTOSCREEN

The ARB commands are:

DELARB and GENARB

The ARB field commands are:

ADDARBFIELD, DELARBFIELD, MODARBFIELD, MOVEARBFIELD, and RENAMEARBFIELD

Their functions are summarized in Table 7-1.

Add a field to the ARB.

Syntax

```
ADDARBFIELD arbname fieldname destination
```

Parameters

<i>arbname</i>	Name of the ARB to which the field will be added.
<i>fieldname</i>	The name of the new field.
<i>destination</i>	Position at which the field is to be inserted. May be left blank.

Discussion

If the *arbname* doesn't exist, an ARB with this name will be created for the associated form. The ARB cannot contain the *fieldname* already. If *fieldname* is not a field on the form, it becomes a filler field, with no datatype and with a length of 1 (this length may be changed using the MODARBFIELD command). If *fieldname* is on the form, its ARB *length* and *data type* are created from its screen attributes and the Data Conversion table (set up using the Data Conversion Menu). If *destination* is specified, the field will be inserted at that position, otherwise it will be appended to the end of the ARB.

ARBTOSCREEN

Create/update the datatype conversion rules for converting ARB field contents to screen field contents.

Syntax

```
ARBTOSCREEN [chartype, inttype, realttype, packtype, zonetype]
```

Parameters

<i>chartype</i>	The conversion type for ARB fields with type CHAR.
<i>inttype</i>	The conversion type for ARB fields with type INT or DINT.
<i>realttype</i>	The conversion type for ARB fields with type REAL or LONG.
<i>packtype</i>	The conversion type for ARB fields with type PACK, PACK0, PACK1, ... PACK9, SPACK, SPACK0, SPACK1, ... SPACK9.
<i>zonetype</i>	The conversion type for ARB fields with type ZONE, ZONE0, ZONE1, ... ZONE9.

Discussion

This command creates/updates the datatype conversion rules for converting ARB data types to screen data types. If a data type update is omitted, then the current default for that type is retained. If no update parameters are specified, all ARB-to-screen conversion types are set to CHAR, the default. Leading commas must be supplied.

Valid data type updates are: CHAR, YMD, MDY, DMY, DIG, NUM[n], NUMA, IMPn, IMPA

EXAMPLE

The command:

```
ARBTOSCREEN , DIG, NUM
```

sets the conversion type for INT and DINT ARB fields to DIG and for REAL and LONG ARB fields to NUM.

Deletes the entire specified ARB.

Syntax

```
DELARB arbname
```

Parameters

arbname Name of the existing ARB.

Discussion

All the fields are deleted. You can generate another ARB for the specified form if required. If extensive changes are made to a form, it is easier to delete the existing ARB and create a new one than to revise the old ARB.

DELARBFIELD

Deletes a specified field from the ARB.

Syntax

```
DELARBFIELD arbname fieldname
```

Parameters

arbname Name of the existing ARB.

fieldname Name of the specified field on the ARB.

Discussion

The *arbname* must exist in the forms file and the *fieldname* must exist on the ARB.

Generates an ARB from the form specified by the formname.

Syntax

```
GENARB arbname
```

Parameters

arbname Same as the name of the associated form.

Discussion

The ARB will contain a field for every field on the form. The data type and length of each ARB field are determined from the screen attributes and the Data Conversion table (set up using the Data Conversion Menu).

MODARBFIELD

Modify the attributes of a field on an ARB.

Syntax

```
MODARBFIELD arbname fieldname {[length, type]}
```

Parameters

<i>arbname</i>	Name of the ARB that contains the field to be modified.
<i>fieldname</i>	The name of the field to be modified.
[<i>length, type</i>]	The new ARB field length and/or data type.

Discussion

The *arbname* must exist in the forms file and the *fieldname* must exist on the ARB. You can change both the parameters *length* and *type*, but you must specify at least one. They must conform to the edit rules (see Section 3).

EXAMPLE

The command:

```
MODARBFIELD EMPLOYEE SSNUM , CHAR
```

changes the ARB data type to CHAR.

Move a field or group of fields to a specified position on an ARB.

Syntax

```
MOVEARBFIELD arbname range [destination]
```

Parameters

<i>arbname</i>	The name of the ARB.
<i>range</i>	The field or fields to be moved, expressed as <i>fieldname</i> or <i>fieldname/fieldname</i>
[<i>destination</i>]	The position in the ARB to which the field(s) will be moved. Can be left blank.

Discussion

The *arbname* must exist. If the *destination* is not specified, the field(s) will be appended at the end of the ARB.

EXAMPLE

Suppose the fields in the ARB EMPLOYEE are in this order:

NAME
ADDRESS
PHONE
SSNUM
SUPERVISOR

The command:

```
MOVEARBFIELD EMPLOYEE ADDRESS/PHONE AFTER SSNUM
```

or the command:

```
MOVEARBFIELD EMPLOYEE SSNUM BEFORE ADDRESS
```

will rearrange the fields in this order:

NAME
SSNUM
ADDRESS
PHONE
SUPERVISOR

RENAMEARBFIELD

Change the name of *oldfieldname* to *newfieldname* in the ARB *arname*.

Syntax

```
RENAMEARBFIELD arname oldfieldname newfieldname
```

Parameters

<i>arname</i>	Name of the ARB that contains the field.
<i>oldfieldname</i>	The current name of the field.
<i>newfieldname</i>	The new name you want to give the field.

Discussion

This command does not change the ARB data type or length. The *arname* must exist in the forms file, and the *oldfieldname* must exist on the ARB. The ARB cannot already contain a field with the *newfieldname*. If *newfieldname* is not a field on the form, the renamed field becomes a filler field.

Create/update the datatype conversion rules for generating an ARB from a form.

Syntax

```
SCREENTOARB [chartype, datetype, digtype, numtype, imptype]
```

Parameters

<i>chartype</i>	The conversion type for screen fields with type CHAR.
<i>datetype</i>	The conversion type for screen fields with a date type.
<i>digtype</i>	The conversion type for screen fields with type DIG.
<i>numtype</i>	The conversion type for screen fields with type NUM, NUM0, NUM1, ... NUM9.
<i>imptype</i>	The conversion type for screen fields with type IMP, IMP0, IMP1, ... IMP9.

Discussion

This command creates/updates the datatype conversion rules for generating an ARB from a form. If an update parameter is omitted, then the current conversion type for that parameter is retained. If no update parameters are specified, all screen-to-ARB conversion types are set to CHAR, the default. Leading commas must be supplied.

Valid type updates are: CHAR, YYMMDD, INT, DINT, REAL, LONG, PACKn, SPACKn, ZONE n, PACKA, SPACKA, ZONEA

EXAMPLE

The command:

```
SCREENTOARB ,INT, REAL, REAL
```

sets the conversion type for DIG screen fields to INT, and for NUM and IMP screen fields to REAL.

SAMPLES OF FORMSPEC IN BATCH MODE

The following are samples of how FORMSPEC in batch mode is used, first with a command file and then with the commands directly in the job stream.

Sample Command File

FORMSPEC in batch mode can be executed with the following commands:

```
:FILE FORMOUT;DEV=LP  
:RUN FORMSPEC.PUB.SYS;INFO="FORMCMDS.PUB.MFG"
```

The MPE :FILE command directs batch mode output (commands and error messages) to the list device. The MPE :RUN command with the INFO=*filename* parameter executes FORMSPEC in batch mode, taking batch mode commands from the command file FORMCMDS.PUB.MFG. Remember, you must first create the command file using a text editor.

Sample Job Stream

To stream a job using FORMSPEC in batch mode, you must first create the job file using a text editor. Then, stream the job using the MPE :STREAM command.

The following is a listing of a sample job stream which compiles a forms file and documents its contents:

```
!JOB FORMUPDT, MIKE.MFG,FORMS  
!FILE FORMOUT; DEV=LP  
!RUN FORMSPEC.PUB.SYS  
FILE FORMSF1  
COMPILE  
FORMS  
EXIT  
!EOJ
```

VPLUS/V Native Language Support (NLS) enables an applications designer to create interactive user applications which reflect both the user's native language and the local custom for numeric and date information in the supported languages. (See *Native Language Support Reference Manual* (32414-90001) for an explanation of supported languages.) NLS provides these specific features in VPLUS/V:

- Native decimal and thousands indicators.
- Native language month names for dates.
- Alphabetic upshifting of native characters.
- Native characters in single value comparisons, table checks, and range checks.
- Native collating sequence in range checks and single value comparisons.



VPLUS/V does not support the application design process in native languages. Form names, field identifiers and field tags support only USASCII characters.

REFSPEC and REFORMAT do not use NLS features. These programs interact with users in NATIVE-3000 only.

LANGUAGE ATTRIBUTE

VPLUS/V contains an NLS native language attribute option which allows the applications programmer to design a forms file which reflects the native language characteristics of the application. Each forms file has a global native language ID number. The application may be unlocalized, language-dependent, or international. Refer to the *Native Language Support Reference Manual* for a list of the available native language ID numbers and for examples of these applications.

Unlocalized (NATIVE-3000)

If a native language ID number is not assigned to a forms file, it will default to 0 (NATIVE-3000). NLS features do not impact an unlocalized forms file.

Language-Dependent

Language-dependent forms files are used when the associated application only operates in a single native language context. The native language ID number for the specific native language desired is assigned when the forms file is designed.

International

Multinational corporations may need to maintain a business language for commands, titles, and menus in addition to accommodating the native language of the user for the actual data retrieved or displayed. For this application, -1 is selected as the native language ID number for the forms file. The VPLUS/V intrinsic VSETLANG must be called at run-time to assign the appropriate native language ID.

NLS CAPABILITIES

The components of a form which can be language-dependent are the initial values of fields and the field edit rules. The text is fixed in a single native language by the forms designer. The native language ID number determines the context for data editing, conversion, and formatting. There are two native language IDs assigned for each forms file. The FORMSPEC language controls the context when the forms file is designed. The forms file language controls the context when the forms file is executed.

Setting the Native Language ID Numbers

The forms designer sets both native language ID number values for the forms file via the FORMSPEC Terminal/Language Selection Menu. NATIVE-3000 is currently the only selection available for FORMSPEC language. This means that initial values and processing specifications must be defined with the month names and numeric conventions of NATIVE-3000.

The native language ID for the forms file language defaults to 0 (NATIVE-3000) if no native language ID number is specified on the Terminal/Language Selection Menu. On this menu, the forms designer can assign or change the native language ID for the forms file language at any time. The value specified must be a positive number or a zero for a single native language application. If the value specified is acceptable, but the native language is not configured on the system used for forms design, FORMSPEC issues a warning message but does not reject the native language ID number. Instead, the designer is prompted to confirm the value or change it.

For applications that are used with multiple native languages, the forms designer specifies the international language ID number -1. The international language ID number allows the intrinsic VSETLANG to be called at run-time to select the actual native language ID number for the forms file. If an application uses an international forms file without calling VSETLANG, it is executed in the default, NATIVE-3000. If VSETLANG is called for an unlocalized or language-dependent forms file, an error code will be returned.

The designer has three options in designing an application to work effectively with multiple languages:

- Develop several language-dependent forms files.
- Create one international forms file.
- Produce a combination of language-dependent files and an international forms file.

If the text needs to be in the native language, unique versions of a forms file are required for each native language supported.

VGETLANG may be used to determine whether a language-dependent forms file or an international forms file is being executed. If VGETLANG indicates an international forms file, VSETLANG must be called to select the actual native language. Refer to the VGETLANG and VSETLANG intrinsics in Section 6.

FIELD EDITS

NATIVE-3000 must be used to specify date and numeric fields within FORMSPEC. When the forms file is executed, VPLUS/V will convert the value to be consistent with the native language selected. Single value comparisons (GT,GE,LT,LE,EQ,NE), as well as table and range checks (IN, NIN), specified within FORMSPEC may contain any character in the 8 bit extended character set consistent with the selected native language ID number. When the form is executed at run-time, the collating table for the native language specified is used to check whether the field is within a range.

Date Handling

VPLUS/V supports several date formats and three date orders: MDY, DMY,YMD. Any format is acceptable as input when the form is executed, provided that the field length can accommodate the format. The forms designer specifies the order of each date-type field. With NLS, the alphabetic month names are edited and converted to numeric destinations using the month names corresponding to the native language of the forms file. The format and the date order are not related to the native language of the forms file.

Numeric Data

Decimal and thousands indicators are native language-dependent in the NUM[n] and IMPn fields. When data is moved between fields and automatic formatting occurs for data entered in any field, recognition, removal or insertion of these decimal and thousands indicators is native language-dependent. The optional decimal symbol in constants is also native language-dependent.

NOTE

VPLUS/V edit processing specifications and terminal edit processing statements are separate and are not checked for compatibility. There will be no check to see that the forms designer has specified a terminal local edit (DEC_TYPE_EUR, DEC_TYPE_US) in the configuration phase which is consistent with the native language-dependent decimal indicator for the native language specified in the forms file.

Native Language Characters

If a native language ID number has been specified in the forms file, the UPSHIFT formatting statement uses native language upshift tables.

Native Language Support

Range checks and the single value comparisons GT, GE, LT and LE involve collating sequences. When the form is executed, the native language collating sequence table designated by the native language ID number is used to check whether the field passes the edit.

NLS features in VPLUS/V do not include support for pattern matching with native characters. MATCH uses USASCII specifications.

ENTRY AND LANGUAGE ID NUMBER

The native language ID assigned for the forms file language determines the native language used by ENTRY unless the file is international (-1). ENTRY uses the intrinsic VGETLANG to identify the native language ID assigned by the forms designer for the forms file language.

If the forms file is international, ENTRY calls the NLS intrinsic NLGETLANG (mode 1). If it returns a value of UNKNOWN, the user is prompted for a native language ID number. Once a valid native language ID number is determined, ENTRY calls the VSETLANG intrinsic to specify the corresponding native language.

The batch file does not have a native language indicator. Users with different native languages may collect data in the same batch file if the associated forms file is international.

SAMPLE PROGRAMS

APPENDIX

A

The VPLUS/V intrinsics can be called from the programming languages listed in Table A-1. The reference manuals to consult for each language are also listed.

Table A-1. Programming Languages and References

Language	Reference Manual
BASIC	<i>BASIC/3000 Interpreter Reference Manual (30000-90026)</i> <i>BASIC/3000 Compiler Reference Manual (32103-90001)</i>
HP BUSINESS BASIC	<i>HP BUSINESS BASIC Reference Manual (32115-90001)</i> <i>HP BUSINESS BASIC Programming Guide (32115-90003)</i>
COBOL	<i>COBOL/3000 Reference Manual (32213-90001)</i> <i>COBOLII/3000 Reference Manual (32233-90001)</i> <i>COBOLII/3000 Conversion Guide (32233-90005)</i> <i>Using COBOL (32213-90003)</i>
FORTRAN	<i>FORTRAN/3000 Reference Manual (30000-90040)</i> <i>HP FORTRAN 77 Reference Manual (5957-4685)</i> <i>FORTRAN 77 Supplement (30000-90294)</i>
Pascal	<i>Pascal/3000 Reference Manual (32106-90001)</i>
SPL	<i>System Programming Language Reference Manual (30000-90024)</i>
TRANSACT	<i>Transact/3000 Reference Manual (32247-90001)</i> <i>Getting Started with Transact Manual (32247-90007)</i>

In addition, this appendix contains sample programs, with the VPLUS/V intrinsic calls highlighted, for each of the following programming languages:

Language	Page
SPL	A-2
COBOL	A-26
FORTRAN 77	A-38
BASIC	A-57
TRANSACT	A-66
PASCAL	A-75

SPL

```
$PAGE "HP32209B.04.17 VPLUS/V S40S209B, ENTRY"
$COPYRIGHT "
$ "
$ " (C) COPYRIGHT HEWLETT-PACKARD. 1986
$ "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 USLIMIT, LIST, MAP, CODE
<<*****>>
<< >>
<< ENTRY--VPLUS/V 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/
                        COLLECT

>>
```

```

$PAGE "          ENTRY DECLARATIONS"
<<*****>>
<<                                     >>
<<          ENTRY Global Declarations          >>
<<                                     >>
<<*****>>
BEGIN
DEFINE
  VERSION          = "B.04.17" #
  ,ID'MSG=("HP32209",VERSION," ENTRY (C) HEWLETT-PACKARD CO. 1986")#
  ;

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'DBUFLLEN        = 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

```

SPL

```
,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
;
EQUATE  << NATIVE LANGUAGE SUPPORT EQUATES >>
  INTERNATIONAL    = -1
;
INTEGER ARRAY
  COMAREA (0:COMAREALEN-1) := COMAREALEN (0)
;
BYTE ARRAY
  COMAREA'B (*) = COMAREA
;
DOUBLE ARRAY
  COMAREA'D (*) = COMAREA
;
LOGICAL
  ERRORS := FALSE
,BATCH
;
ARRAY
  MESSAGE'WBUF (0:MAXWINDOWLEN/2)
;
BYTE ARRAY
  MESSAGE'BUF (*) = MESSAGE'WBUF
;
INTEGER
  PARMVAL := 20
,UNDERLINE := 1
,MESSAGE'BUF'LEN := MAXWINDOWLEN
,MSGLEN
,PAGE'EJECT := %61
;
DOUBLE
  LAST'REC'NUM
;
```

```

$PAGE "                VPLUS/V INTRINSIC DECLARATIONS"
<<*****>>
<<                                                                >>
<<                VPLUS/V INTRINSICS                            >>
<<                                                                >>
<<*****>>
INTRINSIC
  VCLOSEBATCH
  ,VCLOSEFORMF
  ,VCLOSETERM
  ,VERRMSG
  ,VFIELDEDITS
  ,VFINISHFORM
  ,VGETNEXTFORM
  ,VINITFORM
  ,VOPENBATCH
  ,VOPENFORMF
  ,VOPENTERM
  ,VPOSTBATCH
  ,VPRINTFORM
  ,VPUTWINDOW
  ,VREADBATCH
  ,VREADFIELDS
  ,VSHOWFORM
  ,VWRITEBATCH
  ,VGETKEYLABELS
  ,VSETKEYLABELS
  ,VSETKEYLABEL
  ,VSETLANG
  ,VGETLANG
  ;

<<*****>>
<<                                                                >>
<<                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  "

```

SPL

```
        << 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;

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 >>

```



SPL

```
$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 >>

```

SPL

```
$PAGE "          ENTRY INITIALIZATION PROCEDURE"
<<*****>>
<<                                                    >>
<<                INIT                                >>
<<                                                    >>
<<*****>>
PROCEDURE INIT;
  BEGIN

    EQUATE
      VERSIONS'DIFF = 70
      ,DIF'FF       = 73
      ,FILENAMELEN  = 36
      ,LANGID'LEN   = 17
    ;
    EQUATE
      BLANK'LINE    = 0
      ,GET'FF'NAME  = 1
      ,GET'BF'NAME  = 2
      ,DIF'FF'WARN  = 3
      ,VERS'DIF'WARN = 4
      ,Y'TO'CONT    = 5
      ,PRODUCT'ID  = 6
      ,GET'LANGID  = 7
      ,NOT'CONFIG  = 8
      ,NOT'INSTALL = 9
    ;
    INTEGER
      INDEX
      ,READ'LEN
      ,LANGID
      ,VERROR
    ;
    LOGICAL
      CONTINUE
    ;
    LOGICAL ARRAY
      NLERROR(0:1)
    ;
    LOGICAL ARRAY
      LANGID'STR'L(0:9)
    ;
    BYTE ARRAY
      LANGID'STR(*) = LANGID'STR'L
    ;
    BYTE ARRAY
      FILENAME (0:FILENAMELEN)
    ;
    INTRINSIC
      TERMINATE
      ,QUIT
      ,PRINT
      ,READ
    ;
```

```
INTRINSIC
  NLGETLANG
  ,NLINFO
  ;
```

```
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, BLANK'LINE >>
      MOVE MESSAGE'BUF:=" ",2;

      << 1, FF'NAME'PROMPT >>
      MOVE MESSAGE'BUF:=" ENTER FORMS FILE NAME AND PRESS RETURN: ",
        2;

      << 2, BF NAME PROMPT >>
      MOVE MESSAGE'BUF:=" ENTER BATCH FILE NAME AND PRESS RETURN: ",
        2;

      << 3, DIFFERENT FF WARNING >>
      MOVE MESSAGE'BUF:=(" WARNING: A different forms file was used",
        " to create this batch."),2;

      << 4, FF MOD WARN >>
      MOVE MESSAGE'BUF:=(" WARNING: Forms File was recompiled since",
        " this batch was created."), 2;

      << 5, Y'TO'CONTINUE >>
      MOVE MESSAGE'BUF := (" Enter ""Y"" to continue: "), 2;
```

SPL

```
<< 6, PRODUCT'ID >>
MOVE MESSAGE'BUF := ID'MSG, 2;

<< 7, GET'LANGID >>
MOVE MESSAGE'BUF :=(" ENTER LANGUAGE ID NUMBER AND PRESS",
                    " RETURN: "),2;

<< 8, NOT'CONFIG >>
MOVE MESSAGE'BUF := " Specified language is not configured ",2;

<< 9, NOT'INSTALL >>
MOVE MESSAGE'BUF := " Native language Software not installed",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 OR FILENAME = " " 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
          END
        END
      END
    END
  END

```

SPL

```
        ELSE << IS REAL ERROR >>
            WRITE'MSG;
        END;

IF COM'STATUS = 0 THEN
    BEGIN

    VGETLANG( COMAREA, LANGID );
    IF COM'STATUS <> 0 THEN WRITE'MSG
    ELSE IF LANGID = INTERNATIONAL THEN BEGIN

    << IF INTERNATIONAL FORMS FILE PROMPT FOR LANGID >>

        CONTINUE := TRUE;
        LANGID := NLGETLANG( 1, NLERROR );
        IF NLERROR = 0 THEN BEGIN
            VSETLANG( COMAREA, LANGID, VERROR );
            IF VERROR = 0 THEN CONTINUE := FALSE;
            COM'STATUS := 0;
        END;

        WHILE CONTINUE DO BEGIN

            PRINT'TO'TERM( BLANK'LINE, %40 );
            PRINT'TO'TERM( GET'LANGID, %320 );
            READ'LEN := READ'FROM'TERM( LANGID'STR, LANGID'LEN );
            IF READ'LEN = 0 THEN CONTINUE := FALSE
            ELSE BEGIN
                LANGID'STR( READ'LEN ) := " ";
                NLINFO( 22, LANGID'STR'L, LANGID, NLERROR );
                IF NLERROR = 0 THEN BEGIN
                    VSETLANG( COMAREA, LANGID, VERROR );
                    IF VERROR = 0 AND COM'STATUS = 0 THEN
                        CONTINUE := FALSE;
                    IF COM'STATUS <> 0 THEN WRITE'MSG;
                END
                ELSE IF NLERROR = 1
                    THEN PRINT'TO'TERM(NOT'INSTALL,%40)
                    ELSE PRINT'TO'TERM(NOT'CONFIG,%40);
            END;
        END;
    END; << WHILE CONTINUE >>
    END; << IF LANGID = INTERNATIONAL >>

END; << IF COM'STATUS = 0 >>

<< 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 >>
```

SPL

```
$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 >>
  COM 'STATUS := 0;
  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
      else
        if errors then
          print (message 'wbuf, -msglen, 0); << msg from collect >>

  << 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
      ,UNDERLINE := 1
      ,DIRECTION
    ;
    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 (COM'AREA);
        CHECK'ERROR;

        IF COM'DELETEFLAG = FALSE THEN << NOT DELETED >>
          BEGIN

```

SPL

```
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
COM'SHOWCONTROL.(10:1) := 1;
VSHOWFORM (COMAREA);
COM'SHOWCONTROL.(10:1) := 0;
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;
```



SPL

```
<< 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
COM'SHOWCONTROL.(10:1) := 1;
VSHOWFORM (COMAREA);
COM'SHOWCONTROL.(10:1) := 0;
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

          COM'SHOWCONTROL.(10:1) := 1;
          << TO SUPPRESS KEYBORAD ENABLE >>
          VSHOWFORM (COMAREA);
          COM'SHOWCONTROL.(10:1) := 0;
          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;

```

SPL

```
DO << WHILE ERRORS >>
  BEGIN

  ERRORS := FALSE;

  VSHOWFORM (COMAREA);
  CHECK'ERROR;

  COM'SHOWCONTROL := 0; << CLEAR >>

  IF COM'DBUFLN <= 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
                      VFIELDDEDITS (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;

      COM'SHOWCONTROL.(14:1):=1;
      BROWSE;
      COM'SHOWCONTROL.(14:1):=0;

      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'DBUFLN > 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.

COBOL

\$CONTROL LIST, MAP, VERBS

IDENTIFICATION DIVISION.

PROGRAM-ID. COBOL-EXAMPLE.

```
*****
***** This application collects employee payroll deduction
***** transactions and places the edited transactions into
***** a file.
*****
***** For this application:          Enter key = edit and file
*****                               transaction;
*****
*****                               f8 = exit application;
*****
*****                               all other f keys = redo transaction.
*****
***** Each transaction entered by the operator is subjected to the
***** data edits embedded within the input form.
*****
***** The application continues to collect transactions until either
***** the operator signals to exit or a system error is detected.
*****
```

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT TXN-ENTRY ASSIGN TO "PAYTXN".

DATA DIVISION.

FILE SECTION.

```
FD TXN-ENTRY
   RECORD CONTAINS 200 CHARACTERS
   DATA RECORDS ARE TXN-REC.
```

```
01 TXN-REC.
   05 FILLER          PIC X(200).
WORKING-STORAGE SECTION.
```

```
01 COMAREA.
   05 CSTATUS        PIC S9(4) COMP VALUE 0.
   05 LANGUAGE        PIC S9(4) COMP VALUE 0.
   05 COMAREALEN      PIC S9(4) COMP VALUE 0.
   05 USERBUFLEN      PIC S9(4) COMP VALUE 0.
   05 CMODE           PIC S9(4) COMP VALUE 0.
   05 LASTKEY         PIC S9(4) COMP VALUE 0.
```

```

05 NUMERRS          PIC S9(4) COMP VALUE 0.
05 WINDOWENH       PIC S9(4) COMP VALUE 0.
05 MULTIUSAGE      PIC S9(4) COMP VALUE 0.
05 LABELOPTIONS    PIC S9(4) COMP VALUE 0.
05 CFNAME          PIC X(16) VALUE SPACES.
05 NFNAME          PIC X(16) VALUE SPACES.
05 REPEATAPP       PIC S9(4) COMP VALUE 0.
05 FREEZEAPP       PIC S9(4) COMP VALUE 0.
05 CFNUMLINES      PIC S9(4) COMP VALUE 0.
05 DBUFLEN         PIC S9(4) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 LOOKAHEAD       PIC S9(4) COMP VALUE 0.
05 DELETEFLAG      PIC S9(4) COMP VALUE 0.
05 SHOWCONTROL     PIC S9(4) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 PRINTFILENUM    PIC S9(4) COMP VALUE 0.
05 FILERRNUM       PIC S9(4) COMP VALUE 0.
05 ERRFILENUM      PIC S9(4) COMP VALUE 0.
05 FORMSTORESIZE   PIC S9(4) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 NUMRECS         PIC S9(8) COMP VALUE 0.
05 RECNUM          PIC S9(8) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 TERMFILLEN     PIC S9(4) COMP VALUE 0.
05 FILLER          PIC S9(4) COMP VALUE 0.
05 RETRIES         PIC S9(4) COMP VALUE 0.
05 TERMOPTIONS    PIC S9(4) COMP VALUE 0.
05 ENVIRON         PIC S9(4) COMP VALUE 0.
05 USERTIME        PIC S9(4) COMP VALUE 0.
05 IDENTIFIER      PIC S9(4) COMP VALUE 0.
05 LABELINFO       PIC S9(4) COMP VALUE 0.
01 FIELDINFO.
05 NUM-ENTRIES     PIC S9(4) COMP.
05 ENTRY-LEN       PIC S9(4) COMP.
05 FORM-NAME       PIC X(16).
05 FIELD-ENTRY     OCCURS 3 TIMES.
    10 FIELD-NAME   PIC X(16).
    10 SCREEN-ORDER PIC S9(4) COMP.

01 FIELDSPECS.
05 SPEC-ENTRY      OCCURS 3 TIMES.
    10 FIELD-ID     PIC S9(4) COMP.
    10 CHANGE-TYPE  PIC S9(4) COMP.
    10 CHANGE-SPEC  PIC X(4).

01 DATABUF         PIC X(200).

```

COBOL

```
01 DATABUFLEN          PIC S9(4) COMP.
01 DONE-WITH-TRANSACTIONS PIC X.
01 ERROR-LOCATION       PIC X(70).
01 FILENAME           PIC X(86).
01 FOUND-DATA-ERRORS  PIC X.
01 INFOBUFLEN         PIC S9(4) COMP.
01 MSGBUF             PIC X(150).
01 MSGBUFLEN          PIC S9(4) COMP.
01 ERRMSGLEN          PIC S9(4) COMP.
01 NBR-TXN-COLLECTED  PIC 9(4).
01 NUMSPECS           PIC S9(4) COMP.
01 STOP-NOW           PIC X.
PROCEDURE DIVISION.
```

A-000-START-PROGRAM.

```
MOVE "N" TO STOP-NOW
      DONE-WITH-TRANSACTIONS.
```

```
MOVE ZERO TO NBR-TXN-COLLECTED.
```

```
PERFORM A-100-SETUP-FOR-WORK.
```

```
PERFORM A-500-COLLECT-TRANSACTIONS
      UNTIL STOP-NOW          = "Y"
      OR   DONE-WITH-TRANSACTIONS = "Y".
```

```
PERFORM A-900-CLEANUP-AFTER-WORK.
```

```
DISPLAY " ".
```

```
DISPLAY "Deduction transactions collected this session = "
      NBR-TXN-COLLECTED.
```

```
IF STOP-NOW = "Y"
      PERFORM Z-900-DISPLAY-SYSTEM-ERROR.
```

```
STOP RUN.
```

A-100-SETUP-FOR-WORK.

***** Finish Comarea initialization.

***** (Note Comarea value clauses.)

***** Set Language for COBOL.

MOVE ZERO TO LANGUAGE OF COMAREA.

***** Set Comarealen to 60 words (120 bytes).

MOVE 60 TO COMAREALEN OF COMAREA.

***** Activate function key labeling.

MOVE 1 TO LABELOPTIONS OF COMAREA.

***** Disable form background loading on Vreadfields.

MOVE ZERO TO LOOKAHEAD OF COMAREA.

***** Set size of local form storage directory.

MOVE 4 TO FORMSTORESIZE OF COMAREA.

***** Open the Transaction File

OPEN OUTPUT TXN-ENTRY.

***** Open the Forms File.

MOVE "PAYROLL.WORK.ADMIN" TO FILENAME.

CALL "VOPENFORMF" USING COMAREA
FILENAME.

IF CSTATUS OF COMAREA NOT = 0

MOVE "Y" TO STOP-NOW

MOVE

"***** Paragraph: A-100-SETUP-FOR-WORK - Forms File Open"

TO ERROR-LOCATION

PERFORM Z-100-GET-ERROR-MESSAGE.

COBOL

```
IF STOP-NOW NOT = "Y"
```

```
*****
```

```
***** Open the Terminal.
```

```
*****
```

```
MOVE "HPTERM" TO FILENAME
```

```
CALL "VOPENTERM" USING COMAREA  
FILENAME
```

```
IF CSTATUS OF COMAREA NOT = 0
```

```
MOVE "Y" TO STOP-NOW
```

```
MOVE
```

```
"***** Paragraph: A-100-SETUP-FOR-WORK - Terminal Setu
```

```
"p"
```

```
TO ERROR-LOCATION
```

```
PERFORM Z-100-GET-ERROR-MESSAGE.
```

```
IF STOP-NOW NOT = "Y"
```

```
*****
```

```
***** Translate field names to screen orders.
```

```
*****
```

```
***** Three of the fields in the form used by this  
***** application need to be toggled from "display  
***** only" to "input allowed". In order to do this,  
***** we first translate field names to screen orders.  
*****
```

```
MOVE 3 TO NUM-ENTRIES OF FIELDINFO
```

```
MOVE 9 TO ENTRY-LEN OF FIELDINFO
```

```
MOVE "DEDUCTION" TO FORM-NAME OF FIELDINFO
```

```
*****
```

```
***** The value 8224, which is moved to Screen Order in  
***** the following statements is equal to two ASCII blanks.  
*****
```

```
MOVE "BADGE_NUMBER" TO FIELD-NAME  
OF FIELD-ENTRY (1)  
MOVE 8224 TO SCREEN-ORDER  
OF FIELD-ENTRY (1)
```

```
MOVE "LAST_NAME" TO FIELD-NAME  
OF FIELD-ENTRY (2)  
MOVE 8224 TO SCREEN-ORDER  
OF FIELD-ENTRY (2)
```

```
MOVE "SUR_NAME" TO FIELD-NAME  
OF FIELD-ENTRY (3)  
MOVE 8224 TO SCREEN-ORDER  
OF FIELD-ENTRY (3)
```

```

*****
***** Now determine the length of the entire Fieldinfo
***** Buffer.
*****

```

```

MULTIPLY NUM-ENTRIES OF FIELDINFO
  BY ENTRY-LEN OF FIELDINFO
  GIVING INFOBUFLEN

```

```

ADD 10 TO INFOBUFLEN

```

```

CALL "VGETFIELDINFO" USING COMAREA
                                FIELDINFO
                                INFOBUFLEN

```

```

IF CSTATUS OF COMAREA NOT = 0
  MOVE "Y" TO STOP-NOW
  MOVE
    "**** Paragraph: A-100-SETUP-FOR-WORK - Field Informa
      tion Retrieval"
    TO ERROR-LOCATION
  PERFORM Z-100-GET-ERROR-MESSAGE.
A-500-COLLECT-TRANSACTIONS.

```

```

*****
***** Setup for and get transaction data entry form.
*****

```

```

MOVE ZERO TO REPEATAPP OF COMAREA
  FREEZEAPP OF COMAREA.

```

```

MOVE "DEDUCTION" TO NFNAME OF COMAREA.

```

```

CALL "VGETNEXTFORM" USING COMAREA.

```

```

IF CSTATUS OF COMAREA NOT = 0
  MOVE "Y" TO STOP-NOW
  MOVE
    "**** Paragraph: A-500-COLLECTION-TRANSACTIONS - Form R
      etrieval"
    TO ERROR-LOCATION
  PERFORM Z-100-GET-ERROR-MESSAGE.

```

```

IF STOP-NOW NOT = "Y"

```

```

*****
***** Toggle three fields in form to "input allowed".
*****
***** Screen order is indicated to field change
***** intrinsic as negative number.
*****
***** Change field type is indicated by a 5.
*****
***** "Input allowed" is indicated by an "0"

```

COBOL

***** (for input/output).

MULTIPLY SCREEN-ORDER OF FIELD-ENTRY (1)
BY -1
GIVING FIELD-ID OF SPEC-ENTRY (1)
MOVE 5 TO CHANGE-TYPE OF SPEC-ENTRY (1)
MOVE "0" TO CHANGE-SPEC OF SPEC-ENTRY (1)

MULTIPLY SCREEN-ORDER OF FIELD-ENTRY (2)
BY -1
GIVING FIELD-ID OF SPEC-ENTRY (2)
MOVE 5 TO CHANGE-TYPE OF SPEC-ENTRY (2)
MOVE "0" TO CHANGE-SPEC OF SPEC-ENTRY (2)

MULTIPLY SCREEN-ORDER OF FIELD-ENTRY (3)
BY -1
GIVING FIELD-ID OF SPEC-ENTRY (3)
MOVE 5 TO CHANGE-TYPE OF SPEC-ENTRY (3)
MOVE "0" TO CHANGE-SPEC OF SPEC-ENTRY (3)

MOVE 3 TO NUMSPECS
CALL "VCHANGEFIELD" USING COMAREA
FIELD SPECS
NUMSPECS

IF CSTATUS OF COMAREA NOT = 0
MOVE "Y" TO STOP-NOW
MOVE
"**** Paragraph: A-500-COLLECT-TRANSACTIONS - Field
- "Type Updates"
TO ERROR-LOCATION
PERFORM Z-100-GET-ERROR-MESSAGE.

IF STOP-NOW NOT = "Y"

***** Load window message.

MOVE 79 TO MSGBUFLEN

MOVE
"Fill in Deduction Transaction according to worksheet."
TO MSGBUF

CALL "VPUTWINDOW" USING COMAREA
MSGBUF
MSGBUFLEN

IF CSTATUS OF COMAREA NOT = 0
MOVE "Y" TO STOP-NOW
MOVE
"**** Paragraph: A-500-COLLECT-TRANSACTIONS - Window

```

      " Load"
      TO ERROR-LOCATION
      PERFORM Z-100-GET-ERROR-MESSAGE.

      IF STOP-NOW NOT = "Y"

*****
***** Initialize form.
*****

      CALL "VINITFORM" USING COMAREA

      IF CSTATUS OF COMAREA NOT = 0
      MOVE "Y" TO STOP-NOW
      MOVE
      "***** Paragraph: A-500-COLLECT-TRANSACTIONS - Form I
      "nit "
      TO ERROR-LOCATION
      PERFORM Z-100-GET-ERROR-MESSAGE.
      IF STOP-NOW NOT = "Y"

*****
***** Show form.
*****

      CALL "VSHOWFORM" USING COMAREA

      IF CSTATUS OF COMAREA NOT = 0
      MOVE "Y" TO STOP-NOW
      MOVE
      "***** Paragraph: A-500-COLLECT-TRANSACTIONS - Form D
      "isplay"
      TO ERROR-LOCATION
      PERFORM Z-100-GET-ERROR-MESSAGE.

      IF STOP-NOW NOT = "Y"

*****
***** Setup and loop on transaction until it can be filed.
*****

      MOVE "Y" TO FOUND-DATA-ERRORS

      PERFORM B-100-READ-EDIT-AND-FILE
      UNTIL FOUND-DATA-ERRORS = "N"
      OR STOP-NOW = "Y"
      OR DONE-WITH-TRANSACTIONS = "Y".
      B-100-READ-EDIT-AND-FILE.

*****
***** Read form.
*****

      CALL "VREADFIELDS" USING COMAREA.

```

COBOL

```
IF CSTATUS OF COMAREA NOT = 0
  MOVE "Y" TO STOP-NOW
  MOVE
    "**** Paragraph: B-100-READ-EDIT-AND-FILE - Terminal Rea
-    "d"
    TO ERROR-LOCATION
  PERFORM Z-100-GET-ERROR-MESSAGE.
```

```
IF STOP-NOW NOT = "Y"
```

```
*****
***** Determine if operator wants to stop transaction collection.
*****
```

```
IF LASTKEY OF COMAREA = 1
  MOVE "Y" TO DONE-WITH-TRANSACTIONS.
```

```
IF STOP-NOW NOT = "Y"
  AND DONE-WITH-TRANSACTIONS NOT = "Y"
```

```
*****
***** Edit data read from terminal
*****
```

```
CALL "VFIELDDEDITS" USING COMAREA
```

```
IF CSTATUS OF COMAREA NOT = 0
  MOVE "Y" TO STOP-NOW
  MOVE
    "**** Paragraph: B-100-READ-EDIT-AND-FILE - Data Edit
-    " "
    TO ERROR-LOCATION
  PERFORM Z-100-GET-ERROR-MESSAGE
```

```
ELSE
```

```
*****
***** Determine if edit errors detected.
*****
```

```
IF NUMERRS OF COMAREA < 1
  MOVE "N" TO FOUND-DATA-ERRORS.
IF STOP-NOW NOT = "Y"
  AND DONE-WITH-TRANSACTIONS NOT = "Y"
  AND FOUND-DATA-ERRORS NOT = "Y"
```

```
*****
***** Finish form data.
*****
```

```
CALL "VFINISHFORM" USING COMAREA
```

```
IF CSTATUS OF COMAREA NOT = 0
  MOVE "Y" TO STOP-NOW
```

```

MOVE
  "***** Paragraph: B-100-READ-EDIT-AND-FILE - Data Fini
  "shing"
  TO ERROR-LOCATION
PERFORM Z-100-GET-ERROR-MESSAGE

```

```
ELSE
```

```
*****
```

```
***** Determine if data finishing errors detected.
```

```
*****
```

```

IF NUMERRS OF COMAREA > 0
  MOVE "Y" TO FOUND-DATA-ERRORS.
IF STOP-NOW NOT = "Y"
  AND DONE-WITH-TRANSACTIONS NOT = "Y"

```

```
*****
```

```
***** Do we have a transaction that can be filed?
```

```
*****
```

```
IF FOUND-DATA-ERRORS NOT = "Y"
```

```
IF LASTKEY OF COMAREA = 0
```

```
*****
```

```
***** Get transaction from form and file it.
```

```
*****
```

```
MOVE SPACES TO DATABUF
```

```
MOVE 200 TO DATABUFLEN
```

```
CALL "VGETBUFFER" USING COMAREA
                                DATABUF
                                DATABUFLEN
```

```
IF CSTATUS OF COMAREA NOT = 0
```

```
MOVE "Y" TO STOP-NOW
```

```
MOVE
```

```

  "***** Paragraph: B-100-READ-EDIT-AND-FILE - Dat
  "a Get"

```

```

  TO ERROR-LOCATION
PERFORM Z-100-GET-ERROR-MESSAGE

```

```
ELSE
```

```
WRITE TXN-REC FROM DATABUF
```

```
ADD 1 TO NBR-TXN-COLLECTED.
```

```

IF STOP-NOW NOT = "Y"
  AND DONE-WITH-TRANSACTIONS NOT = "Y"

```

COBOL

```
*****  
***** Do we need to prompt the operator to correct errors?  
*****
```

```
IF FOUND-DATA-ERRORS = "Y"  
    IF LASTKEY OF COMAREA = 0  
        PERFORM B-200-PROMPT-OPERATOR.
```

```
IF STOP-NOW NOT = "Y"  
    AND DONE-WITH-TRANSACTIONS NOT = "Y"
```

```
*****  
***** Do we need to refresh the display?  
*****
```

```
IF FOUND-DATA-ERRORS = "Y"  
    IF LASTKEY OF COMAREA NOT = 0
```

```
*****  
***** The operator pressed some key other than Enter  
***** or Exit so clear data error flag to break loop.  
*****
```

```
MOVE "N" TO FOUND-DATA-ERRORS.  
B-200-PROMPT-OPERATOR.
```

```
*****  
***** Get message text associated with first field flagged  
***** with a data error.  
*****
```

```
PERFORM Z-100-GET-ERROR-MESSAGE.
```

```
CALL "VPUTWINDOW" USING COMAREA  
    MSGBUF  
    ERRMSGLEN.
```

```
IF CSTATUS OF COMAREA NOT = 0  
    MOVE "Y" TO STOP-NOW  
    MOVE  
    "***** Paragraph: B-200-PROMPT-OPERATOR - Window Load"  
    TO ERROR-LOCATION  
    PERFORM Z-100-GET-ERROR-MESSAGE.
```

```
IF STOP-NOW NOT = "Y"
```

```
*****  
***** Display highlighted form and updated window message.  
*****
```

```
CALL "VSHOWFORM" USING COMAREA.
```

```

IF CSTATUS OF COMAREA NOT = 0
  MOVE "Y" TO STOP-NOW
  MOVE
    "**** Paragraph: B-200-PROMPT-OPERATOR - Display Upd
      "ates"
    TO ERROR-LOCATION
  PERFORM Z-100-GET-ERROR-MESSAGE.

```

A-900-CLEANUP-AFTER-WORK.

```

*****
***** Note that this paragraph unconditionally attempts to
***** close the Forms File and Terminal.
*****

```

CLOSE TXN-ENTRY.

MOVE 0 to CSTATUS OF COMAREA.

CALL "VCLOSEFORMF" USING COMAREA.

MOVE 0 to CSTATUS OF COMAREA.

CALL "VCLOSETERM" USING COMAREA.

Z-100-GET-ERROR-MESSAGE.

MOVE SPACES TO MSGBUF.
 MOVE 150 TO MSGBUFLN.

CALL "VERRMSG" USING COMAREA
 MSGBUF
 MSGBUFLN
 ERRMSGLEN

Z-900-DISPLAY-SYSTEM-ERROR.

```

DISPLAY "**** Transaction entry facility detected system erro
-       "r at:".
DISPLAY ERROR-LOCATION.
DISPLAY "**** The error message returned is:".
DISPLAY "**** "
      MSGBUF.

```

FORTRAN 77

\$CONTROL list on, tables on

```
!  
! This application collects employee payroll deduction  
! transactions and places the edited transactions into  
! a file.  
!  
! For this application:      Enter key = edit and file  
!                          transaction;  
!  
!                          f8 = exit application;  
!  
!                          all other f keys = redo transaction.  
!  
! Each transaction entered by the operator is subjected to the  
! data edits embedded within the input form.  
!  
! The application continues to collect transactions until either  
! the operator signals to exit or a system error is detected.  
!
```

\$TITLE ' Main Program'

```
!*****!  
! Main Program !  
!*****!
```

PROGRAM FTNEXMP

IMPLICIT NONE

```
COMMON /COM01/ COMAREA  
COMMON /COM11/ STOP_NOW  
COMMON /COM12/ DONE_WITH_TXNS  
COMMON /COM13/ NBR_TXN_COLLECTED  
COMMON /COM21/ FIELDINFO  
COMMON /COM22/ INFOBUFLEN  
COMMON /COM81/ MSGBUF  
COMMON /COM82/ MSGBUFLEN  
COMMON /COM83/ ERRMSGLEN  
COMMON /COM91/ ERROR_LOCATION
```

```
INTEGER*2    COMAREA(60)  
INTEGER*2    STOP_NOW  
INTEGER*2    DONE_WITH_TXNS  
INTEGER*2    NBR_TXN_COLLECTED  
INTEGER*2    FIELDINFO(37)  
INTEGER*2    INFOBUFLEN  
CHARACTER*150 MSGBUF
```

```

INTEGER*2    MSGBUFLEN
INTEGER*2    ERRMSGLEN
CHARACTER*70 ERROR_LOCATION

```

```

!
STOP_NOW = 0
DONE_WITH_TXNS = 0

```

```

!
NBR_TXN_COLLECTED = 0

```

```

!
CALL SETUP_FOR_WORK

```

```

!
DO WHILE (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0)
    CALL COLLECT_TXNS
END DO

```

```

!
CALL CLEANUP_AFTER_WORK

```

```

!
PRINT *,
+ "Deduction transactions collected this session = ",
+ NBR_TXN_COLLECTED

```

```

!
IF (STOP_NOW.EQ.1) THEN
    CALL DISPLAY_SYSTEM_ERROR
END IF

```

```

!
STOP
END

```



FORTRAN 77

```
$TITLE '          Setup For Work'
!*****!
!
!          Setup For Work
!
!*****!
!
SUBROUTINE SETUP_FOR_WORK
!
IMPLICIT NONE
!
COMMON /COM01/ COMAREA
COMMON /COM11/ STOP_NOW
COMMON /COM21/ FIELDINFO
COMMON /COM22/ INFOBUFLEN
COMMON /COM81/ MSGBUF
COMMON /COM82/ MSGBUFLEN
COMMON /COM83/ ERRMSGLEN
COMMON /COM91/ ERROR_LOCATION
!
SYSTEM INTRINSIC VOPENFORMF,
+          VOPENTERM,
+          VGETFIELDINFO
!
INTEGER*2   COMAREA(60)
INTEGER*2   CSTATUS
INTEGER*2   LANGUAGE
INTEGER*2   COMAREALEN
INTEGER*2   LABELOPTIONS
INTEGER*2   LOOKAHEAD
INTEGER*2   FORMSTORESIZE
EQUIVALENCE (COMAREA(1), CSTATUS),
+          (COMAREA(2), LANGUAGE),
+          (COMAREA(3), COMAREALEN),
+          (COMAREA(10), LABELOPTIONS),
+          (COMAREA(32), LOOKAHEAD),
+          (COMAREA(39), FORMSTORESIZE)
INTEGER*2   STOP_NOW
INTEGER*2   FIELDINFO(37)
INTEGER*2   NUM_ENTRIES
INTEGER*2   ENTRY_LEN
CHARACTER*16 FORM_NAME
EQUIVALENCE (FIELDINFO(1), NUM_ENTRIES),
+          (FIELDINFO(2), ENTRY_LEN),
+          (FIELDINFO(3), FORM_NAME)
CHARACTER*18 FIELD_NAME (1,3)
EQUIVALENCE (FIELDINFO(11), FIELD_NAME)
INTEGER*2   INFOBUFLEN
CHARACTER*150 MSGBUF
INTEGER*2   MSGBUFLEN
INTEGER*2   ERRMSGLEN
CHARACTER*70 ERROR_LOCATION
INTEGER*2   ARRAY_INDEX
CHARACTER*86 FILENAME
```

```

!
! Init Comarea to all zeros.
!
  ARRAY_INDEX = 1
  DO WHILE (ARRAY_INDEX.LE.60)
    COMAREA(ARRAY_INDEX) = 0
    ARRAY_INDEX = ARRAY_INDEX + 1
  END DO
!
! Set Language for FORTRAN-77.
!
  LANGUAGE = 5
!
! Set Comarealen for 60 words (120 bytes).
!
  COMAREALEN = 60
!
! Activate function key labeling.
!
  LABELOPTIONS = 1
!
! Disable form background loading on Vreadfields.
!
  LOOKAHEAD = 0
!
! Set size of local form storage directory.
!
  FORMSTORESIZE = 4
!
! Open the Transaction File:
!
  OPEN (UNIT = 10,
+      FILE = 'PAYTXN',
+      ACCESS = 'DIRECT',
+      RECL = 200,
+      FORM = 'UNFORMATTED',
+      STATUS = 'NEW',
+      ERR = 110)
!
  GOTO 120
!
110 STOP NOW = 1
  ERROR_LOCATION =
+ "**** Routine: Setup For Work - Open Transaction File"
  MSGBUF =
+ "**** File open failed!"
!
! Open the Forms File.
!
120 IF (STOP_NOW.EQ.0) THEN
  FILENAME = "PAYROLL.WORK.ADMIN"
!
  CALL VOPENFORMF (COMAREA,
+ FILENAME)

```

FORTRAN 77

```
!
      IF (CSTATUS.NE.0) THEN
        STOP_NOW = 1
        ERROR_LOCATION =
+       "**** Routine: Setup For Work - Forms File Open"
        CALL GET_ERROR_MESSAGE
      END IF
    END IF
!
! Open the Terminal.
!
      IF (STOP_NOW.EQ.0) THEN
        FILENAME = "HPTERM"
!
        CALL VOPENTERM (COMAREA,
+        FILENAME)
!
      IF (CSTATUS.NE.0) THEN
        STOP_NOW = 1
        ERROR_LOCATION =
+       "**** Routine: Setup For Work - Terminal Setup"
        CALL GET_ERROR_MESSAGE
      END IF
    END IF
!
! Translate field names to screen orders.
!
! Three of the fields in the form used by this
! application need to be toggled from "display
! only" to "input allowed". In order to do this,
! we first translate field names to screen orders.
!
      IF (STOP_NOW.EQ.0) THEN
!
        NUM_ENTRIES = 3
        ENTRY_LEN   = 9
        FORM_NAME   = "DEDUCTION"
!
! Note that because the FIELD_NAME element is defined as 18
! characters long, each occurrence of FIELD_NAME overlaps
! the position of each occurrence of the SCREEN_ORDER element
! in the infobuf. Thus setting each FIELD_NAME element to
! a literal that is 16 characters long or less results in each
! SCREEN_ORDER element being implicitly filled with blanks.
!
        FIELD_NAME(1,1) = "BADGE_NUMBER"
!
        FIELD_NAME(1,2) = "LAST_NAME"
!
        FIELD_NAME(1,3) = "SUR_NAME"
!
! Now determine the length of the entire Fieldinfo buffer.
!
```

```

!
      INFOBUFLEN = (NUM_ENTRIES * ENTRY_LEN) + 10
!
      CALL VGETFIELDINFO (COMAREA,
+      FIELDINFO,
+      INFOBUFLEN)
!
      IF (CSTATUS.NE.0) THEN
        STOP_NOW = 1
        ERROR_LOCATION =
+      "**** Routine: Setup For Work - Field Info Retrieval"
        CALL GET_ERROR_MESSAGE
      END IF
    END IF
!
  END

```

FORTRAN 77

```
$TITLE '      Collect Transactions'
!*****!
!
!      Collect Transactions
!
!*****!
!
SUBROUTINE COLLECT_TXNS
!
IMPLICIT NONE
!
COMMON /COM01/  COMAREA
COMMON /COM11/  STOP_NOW
COMMON /COM12/  DONE_WITH_TXNS
COMMON /COM13/  NBR_TXN_COLLECTED
COMMON /COM21/  FIELDINFO
COMMON /COM81/  MSGBUF
COMMON /COM82/  MSGBUFLN
COMMON /COM83/  ERRMSGLEN
COMMON /COM91/  ERROR_LOCATION
COMMON /COM101/ FOUND_DATA_ERRS
!
SYSTEM INTRINSIC VCHANGEFIELD,
+           VPUTWINDOW,
+           VINITFORM,
+           VSHOWFORM
!
INTEGER*2    COMAREA(60)
INTEGER*2    CSTATUS
INTEGER*2    REPEATAPP
INTEGER*2    FREEZEAPP
CHARACTER*16 NFNAME
EQUIVALENCE (COMAREA(1),  CSTATUS),
+           (COMAREA(27), REPEATAPP),
+           (COMAREA(28), FREEZEAPP),
+           (COMAREA(19), NFNAME)
INTEGER*2    STOP_NOW
INTEGER*2    DONE_WITH_TXNS
INTEGER*2    NBR_TXN_COLLECTED
INTEGER*2    FIELDINFO(37)
INTEGER*2    NUM_ENTRIES
INTEGER*2    ENTRY_LEN
CHARACTER*16 FORM_NAME
EQUIVALENCE (FIELDINFO(1), NUM_ENTRIES),
+           (FIELDINFO(2), ENTRY_LEN),
+           (FIELDINFO(3), FORM_NAME)
INTEGER*2    FIELD_NAME (9,3)
INTEGER*2    SCREEN_ORDER (9,3)
EQUIVALENCE (FIELDINFO(11), FIELD_NAME),
+           (FIELDINFO(11), SCREEN_ORDER)
CHARACTER*150 MSGBUF
INTEGER*2    MSGBUFLN
INTEGER*2    ERRMSGLEN
CHARACTER*70 ERROR_LOCATION
```

```

INTEGER*2    FIELDSPECS(12)
              INTEGER*2    FIELD_ID(4,3)
              INTEGER*2    CHANGE_TYPE(4,3)
              CHARACTER*4   CHANGE_SPEC(2,3)
              EQUIVALENCE (FIELDSPECS(1), FIELD_ID),
+                          (FIELDSPECS(1), CHANGE_TYPE),
+                          (FIELDSPECS(1), CHANGE_SPEC)
INTEGER*2    NUMSPECS
INTEGER*2    FOUND_DATA_ERRS
!
! Setup for and get transaction data entry form.
!
REPEATAPP = 0
FREEZEAPP = 0
!
NFNAME = "DEDUCTION"
!
CALL VGETNEXTFORM (COMAREA)
!
IF (CSTATUS.NE.0) THEN
  STOP_NOW = 1
  ERROR_LOCATION =
+  "**** Routine: Collect Transactions - Form Retrieval"
  CALL GET_ERROR_MESSAGE
END IF
!
! Toggle three fields in form to "input allowed".
!
! Screen order is indicated to field change intrinsic
! as a negative number.
!
! Change field type is indicated by a 5.
!
! "Input allowed" is indicated by an "0" (for input/output).
!
IF (STOP_NOW.EQ.0) THEN
  FIELD_ID(1,1)    = (SCREEN_ORDER(9,1) * (-1))
  CHANGE_TYPE(2,1) = 5
  CHANGE_SPEC(2,1) = "0"
!
  FIELD_ID(1,2)    = (SCREEN_ORDER(9,2) * (-1))
  CHANGE_TYPE(2,2) = 5
  CHANGE_SPEC(2,2) = "0"
!
  FIELD_ID(1,3)    = (SCREEN_ORDER(9,3) * (-1))
  CHANGE_TYPE(2,3) = 5
  CHANGE_SPEC(2,3) = "0"
!
  NUMSPECS = 3
!
  CALL VCHANGEFIELD (COMAREA,
+                   FIELDSPECS,
+                   NUMSPECS)

```

FORTTRAN 77

```
!
      IF (CSTATUS.NE.0) THEN
        STOP_NOW = 1
        ERROR_LOCATION =
+       "**** Routine: Collect Transactions - Field Type Updates"
        CALL GET_ERROR_MESSAGE
      END IF
    END IF

!
! Load window message.
!
      IF (STOP_NOW.EQ.0) THEN

        MSGBUFLN = 79

        MSGBUF =
+       "Fill in Deduction Transaction according to worksheet."

        CALL VPUTWINDOW (COMAREA,
+                       MSGBUF,
+                       MSGBUFLN)

        IF (CSTATUS.NE.0) THEN
          STOP_NOW = 1
          ERROR_LOCATION =
+         "**** Routine: Collect Transactions - Window Load"
          CALL GET_ERROR_MESSAGE
        END IF
      END IF

!
! Initialize form.
!
      IF (STOP_NOW.EQ.0) THEN

        CALL VINITFORM (COMAREA)

        IF (CSTATUS.NE.0) THEN
          STOP_NOW = 1
          ERROR_LOCATION =
+         "**** Routine: Collect Transactions - Form Init"
          CALL GET_ERROR_MESSAGE
        END IF
      END IF

!
! Show form.
!
      IF (STOP_NOW.EQ.0) THEN

        CALL VSHOWFORM (COMAREA)

        IF (CSTATUS.NE.0) THEN
          STOP_NOW = 1
          ERROR_LOCATION =
+         "**** Routine: Collect Transactions - Form display"
```

```

        CALL GET_ERROR_MESSAGE
      END IF
    END IF
!
! Setup and loop on transaction until it can be filed.
!
    FOUND_DATA_ERRS = 1
!
    DO WHILE (FOUND_DATA_ERRS.EQ.1
+      .AND.STOP_NOW.EQ.0
+      .AND.DONE_WITH_TXNS.EQ.0)
!
      CALL READ_EDIT_AND_FILE
!
    END DO
!
  END

```

FORTRAN 77

```
$TITLE '          Read Edit and File'
!*****!
!
!          Read Edit and File          !
!
!*****!
!
SUBROUTINE READ_EDIT_AND_FILE
!
IMPLICIT NONE
!
COMMON /COM01/  COMAREA
COMMON /COM11/  STOP_NOW
COMMON /COM12/  DONE_WITH_TXNS
COMMON /COM13/  NBR_TXN_COLLECTED
COMMON /COM81/  MSGBUF
COMMON /COM82/  MSGBUFLN
COMMON /COM83/  ERRMSGLEN
COMMON /COM91/  ERROR_LOCATION
COMMON /COM101/ FOUND_DATA_ERRS
!
SYSTEM INTRINSIC VREADFIELDS,
+          VFIELDEDITS,
+          VFINISHFORM,
+          VGETBUFFER
!
INTEGER*2    COMAREA(60)
INTEGER*2    CSTATUS
INTEGER*2    LASTKEY
INTEGER*2    NUMERRS
EQUIVALENCE (COMAREA(1),  CSTATUS),
+          (COMAREA(6),  LASTKEY),
+          (COMAREA(7),  NUMERRS)
INTEGER*2    STOP_NOW
INTEGER*2    DONE_WITH_TXNS
INTEGER*2    NBR_TXN_COLLECTED
CHARACTER*150 MSGBUF
INTEGER*2    MSGBUFLN
INTEGER*2    ERRMSGLEN
CHARACTER*70  ERROR_LOCATION
INTEGER*2    FOUND_DATA_ERRS
CHARACTER*200 DATABUF
INTEGER*2    DATABUFLN
!
! Read form.
!
CALL VREADFIELDS (COMAREA)
!
IF (CSTATUS.NE.0) THEN
  STOP_NOW = 1
  ERROR_LOCATION =
+  "**** Routine: Read Edit and File - Terminal Read"
  CALL GET_ERROR_MESSAGE
END IF
```

```

!
! Determine if operator wants to stop transaction collection.
!
      IF (STOP_NOW.EQ.0) THEN
        IF (LASTKEY.EQ.8) THEN
          DONE_WITH_TXNS = 1
        END IF
      END IF
!
! Edit data read from terminal.
!
      IF (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0) THEN
!
        CALL VFIELDEDITS (COMAREA)
!
        IF (CSTATUS.NE.0) THEN
          STOP_NOW = 1
          ERROR_LOCATION =
+ "**** Routine: Read Edit and File - Data Edit"
          CALL GET_ERROR_MESSAGE
        END IF
      END IF
!
! Determine if edit errors detected.
!
      IF (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0) THEN
!
        IF (NUMERRS.LT.1) THEN
          FOUND_DATA_ERRS = 0
        END IF
      END IF
!
! Finish form data.
!
      IF (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0
+ .AND.FOUND_DATA_ERRS.EQ.0) THEN
!
        CALL VFINISHFORM (COMAREA)
!
        IF (CSTATUS.NE.0) THEN
          STOP_NOW = 1
          ERROR_LOCATION =
+ "**** Routine: Read Edit and File - Data Finishing"
          CALL GET_ERROR_MESSAGE
        END IF
      END IF
!
! Determine if data finishing errors detected.
!
      IF (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0

```

FORTRAN 77

```
!
+ .AND.FOUND_DATA_ERRS.EQ.0) THEN
!
  IF (NUMERRS.GT.0) THEN
    FOUND_DATA_ERRS = 1
  END IF
END IF
!
! Do we have a transaction that can be filed?
!
  IF (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0) THEN
!
  IF (FOUND_DATA_ERRS.EQ.0
+ .AND.LASTKEY.EQ.0) THEN
!
! Get transaction from form and file it.
!
  DATABUF = " "
!
  DATABUFLEN = 200
!
  CALL VGETBUFFER (COMAREA,
+ DATABUF,
+ DATABUFLEN)
!
  IF (CSTATUS.NE.0) THEN
    STOP_NOW = 1
    ERROR_LOCATION =
+ "**** Routine: Read Edit and File - Data Get"
    CALL GET_ERROR_MESSAGE
!
  ELSE
!
! Write Databuf to Transaction File.
!
  WRITE (UNIT = 10,
+ ERR = 310) DATABUF
!
  GOTO 320
!
310  STOP_NOW = 1
  ERROR_LOCATION =
+ "**** Routine: Read Edit and File - File Write"
  MSGBUF =
+ "**** Write to Transaction File failed!"
!
320  IF (STOP_NOW.EQ.0) THEN
    NBR_TXN_COLLECTED = NBR_TXN_COLLECTED + 1
  END IF
  END IF
  END IF
  END IF
!
! Do we need to prompt the operator to correct errors?
```

```

!
  IF (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0) THEN
!
    IF (FOUND_DATA_ERRS.EQ.1
+ .AND.LASTKEY.EQ.0) THEN
!
      CALL PROMPT_OPERATOR
!
    END IF
  END IF
!
! Do we need to refresh the display?
!
  IF (STOP_NOW.EQ.0
+ .AND.DONE_WITH_TXNS.EQ.0) THEN
!
    IF (FOUND_DATA_ERRS.EQ.1
+ .AND.LASTKEY.NE.0) THEN
!
! The operator pressed some key other than <ENTER>
! or <EXIT> so clear data error flag to break loop.
!
      FOUND_DATA_ERRS = 0
!
    END IF
  END IF
!
  END

```

FORTRAN 77

```

$TITLE ' Prompt Operator'
!*****!
!
! Prompt Operator !
!*****!
!
SUBROUTINE PROMPT_OPERATOR
!
IMPLICIT NONE
!
COMMON /COM01/ COMAREA
COMMON /COM11/ STOP_NOW
COMMON /COM81/ MSGBUF
COMMON /COM82/ MSGBUFLN
COMMON /COM83/ ERRMSGLEN
COMMON /COM91/ ERROR_LOCATION
!
SYSTEM INTRINSIC VPUTWINDOW,
+ VSHOWFORM
!
INTEGER*2 COMAREA(60)
INTEGER*2 CSTATUS
EQUIVALENCE (COMAREA(1), CSTATUS)
INTEGER*2 STOP_NOW
CHARACTER*150 MSGBUF
INTEGER*2 MSGBUFLN
INTEGER*2 ERRMSGLEN
CHARACTER*70 ERROR_LOCATION
!
! Get message text associated with first field flagged
! with a data error.
!
CALL GET_ERROR_MESSAGE
!
CALL VPUTWINDOW (COMAREA,
+ MSGBUF,
+ ERRMSGLEN)
!
IF (CSTATUS.NE.0) THEN
STOP_NOW = 1
ERROR_LOCATION =
+ "**** Routine: Prompt Operator - Window Load"
CALL GET_ERROR_MESSAGE
END IF
!
! Display highlighted form and updated window message.
!
IF (STOP_NOW.EQ.0) THEN
CALL VSHOWFORM (COMAREA)
!

```

```
      IF (CSTATUS.NE.0) THEN
        STOP_NOW = 1
        ERROR_LOCATION =
+       "**** Routine: Prompt Operator - Display Updates"
        CALL GET_ERROR_MESSAGE
      END IF
!     END IF
      END
```

FORTRAN 77

```
$TITLE '          Cleanup After Work'
!*****!
!          Cleanup After Work          !
!*****!
!
!   SUBROUTINE CLEANUP_AFTER_WORK
!
!   IMPLICIT NONE
!
!   COMMON /COM01/ COMAREA
!
!   SYSTEM INTRINSIC VCLOSEFORMF,
+           VCLOSETERM
!
!   INTEGER*2    COMAREA(60)
!   INTEGER*2    CSTATUS
!   EQUIVALENCE (COMAREA(1), CSTATUS)
!
! Note that this routine unconditionally attempts to close
! the Forms File and Terminal
!
!   CLOSE (UNIT = 10)
!
!   CSTATUS = 0
!
!   CALL VCLOSEFORMF (COMAREA)
!
!   CSTATUS = 0
!
!   CALL VCLOSETERM (COMAREA)
!
!   END
```

```

$TITLE '          Get Error Message'
!*****!
!
!          Get Error Message          !
!
!*****!
!
SUBROUTINE GET_ERROR_MESSAGE
!
IMPLICIT NONE
!
COMMON /COM01/ COMAREA
COMMON /COM01/ MSGBUF
COMMON /COM02/ MSGBUFLN
COMMON /COM03/ ERRMSGLEN
!
SYSTEM INTRINSIC VERRMSG
!
INTEGER*2    COMAREA(60)
CHARACTER*150 MSGBUF
INTEGER*2    MSGBUFLN
INTEGER*2    ERRMSGLEN
!
MSGBUF = " "
MSGBUFLN = 150
!
CALL VERRMSG (COMAREA,
*           MSGBUF,
*           MSGBUFLN,
*           ERRMSGLEN)
!
END

```



FORTRAN 77

```
$TITLE '          Display System Error'
!*****!
!
!          Display System Error          !
!
!*****!
!
SUBROUTINE DISPLAY_SYSTEM_ERROR
!
IMPLICIT NONE
!
COMMON /COM81/ MSGBUF
COMMON /COM91/ ERROR_LOCATION
!
CHARACTER*150 MSGBUF
CHARACTER*70 ERROR_LOCATION
!
PRINT *,
+ "**** Transaction entry facility detected system error at:"
PRINT *, ERROR_LOCATION
PRINT *,
+ "**** The error message returned is:"
PRINT *, MSGBUF
!
END
```

BASIC

```
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 and
75 REM opens it if possible. If all is OK, it displays the HEAD FORM
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 with
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 B0$[16] - Saves B$ 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
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)
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
```

BASIC

```
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
440 REM
450 REM
451 REM      *****
452 REM
500 REM2 ** DECLARATIONS **
510 DIM B$(16),B0$(16),B1$(16),C$(2)
520 DIM E$(60),M$(150),S$(150),U$(150),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
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
2305     create this batch."
2310   IF C[1]=73 THEN PRINT "WARNING: FORMS file was recompiled since
2315     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

```

BASIC

```
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$,B0,I)
3065 E=1
3070 RETURN
3075 DOEND
3080 IF C[1] THEN GOSUB 9100
3085 F1=0
3090 CALL VINITFORM(C[*])
3095 IF C[1] OR C[7] THEN GOSUB 9100
3100 IF NOT E THEN GOSUB 8000
3105 REM1 *SOFTKEY LOOP*
3110 E=0
3115 CALL VSHOWFORM(C[*])
3120 IF C[1] THEN GOSUB 9100
3125 C[34]=0
3130 IF C[30]<=0 AND C[27]=0 AND C[28]<>0 THEN DO
3132 IF NOT E AND B1 THEN DO
3135 CALL VWRITEBATCH(C[*])
3140 IF C[1] THEN GOSUB 9100
3145 IF NOT E THEN DO
3146 C[46]=C[46]+1
3147 P1=20
3148 P=C[46] MOD P1
3149 IF P=0 THEN CALL VPOSTBATCH(C[*])
3150 DOEND
3151 DOEND
3152 DOEND
3155 ELSE DO
3160 CALL VREADFIELDS(C[*])
3165 IF C[1] THEN GOSUB 9100
3170 IF NOT E AND C[6]=8 THEN RETURN
3175 IF NOT E THEN GOSUB C[6]+1 OF 3300,3350,3400,3450,3500,3550,3600,370
3180 DOEND
3185 IF C[6]=8 THEN RETURN
3190 IF E OR C[6]=3 THEN 3105
3195 GOSUB 8100
3200 IF B1$<>"$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
```

BASIC

```
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>
```

BASIC

```
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
```

TRANSACTION

```
SYSTEM TRANS,FILE=TXNFILE(APPEND);
```

```
<<
```

This application collects employee payroll deduction transactions and places the edited transactions into a file.

For this application: Enter key = edit and file transaction;

 f8 = exit application;

 all other f keys = redo transaction.

Each transaction entered by the operator is subjected to the data edits embedded within the input form.

The application continues to collect transactions until either the operator signals to exit or a system error is detected.

```
>>
```

```
DEFINE(ITEM) MAXWINDOWLEN I(4),INIT=150;
```

```
DEFINE(ITEM) COMAREA 60 I(4):
```

```
    CSTATUS I(4)=COMAREA:
```

```
    LANGUAGE I(4)=COMAREA(3):
```

```
    COMAREALEN I(4)=COMAREA(5):
```

```
    USERBUFLN I(4)=COMAREA(7):
```

```
    CMODE I(4)=COMAREA(9):
```

```
    LASTKEY I(4)=COMAREA(11):
```

```
    NUMERRS I(4)=COMAREA(13):
```

```
    WINDOWENH I(4)=COMAREA(15):
```

```
    MULTIUSAGE I(4)=COMAREA(17):
```

```
    LABELOPTIONS I(4)=COMAREA(19):
```

```
    CFNAME X(16)=COMAREA(21):
```

```
    NFNAME X(16)=COMAREA(37):
```

```
    REPEATAPP I(4)=COMAREA(53):
```

```
    FREEZEAPP I(4)=COMAREA(55):
```

```
    CFNUMLINES I(4)=COMAREA(57):
```

```
    DBUFLN I(4)=COMAREA(59):
```

```
    LOOKAHEAD I(4)=COMAREA(63):
```

```
    DELETEFLAG I(4)=COMAREA(65):
```

```
    SHOWCONTROL I(4)=COMAREA(67):
```

```
    PRINTFILENUM I(4)=COMAREA(71):
```

```
    FILERRNUM I(4)=COMAREA(73):
```

```
    ERRFILENUM I(4)=COMAREA(75):
```

```
    FORMSTORESIZE I(4)=COMAREA(77):
```

```
    NUMRECS I(8)=COMAREA(85):
```

```
    RECNUM I(8)=COMAREA(89):
```

```
    TERM-FILEN I(4)=COMAREA(97):
```

```
    RETRIES I(4)=COMAREA(109):
```

```
    TERM-OPTIONS I(4)=COMAREA(111):
```

```
    ENVIRON I(4)=COMAREA(113):
```

```
    USERTIME I(4)=COMAREA(115):
```

```
    IDENTIFIER I(4)=COMAREA(117):
```

```
    LABELINFO X(2)=COMAREA(119):
```

```
    DATABUF X(200):
```

```

DATABUFLEN I(4):
DONE-WITH-TRANS I(4):
ERROR-LOCATION X(70):
FIELDINFO X(80):
  NUM-ENTRIES I(4)=FIELDINFO:
  ENTRY-LEN I(4)=FIELDINFO(3):
  FORM-NAME X(16)=FIELDINFO(5):
  FIELDDATA 3 X(20)=FIELDINFO(21):
  FIELDENTRY X(20)=FIELDDATA:
    FIELD-NAME X(16)=FIELDENTRY:
    SCREEN-ORDER I(4)=FIELDENTRY(17):
    FIELD-NUM I(4)=FIELDENTRY(19):
FIELDSPECS-ITEM 3 X(8):
  FIELDSPECS X(8)=FIELDSPECS-ITEM:
  FIELD-ID I(4)=FIELDSPECS:
  CHANGE-TYPE I(4)=FIELDSPECS(3):
  CHANGE-SPEC X(4)=FIELDSPECS(5):
FILENAME X(86):
FOUND-DATA-ERRS I(4):
INFOBUFLEN I(4):
MSGBUF X(150):
MSGBUFLEN I(4):
NBR-TXN-COLLECT I(8):
NUMENTRIES I(4):
NUMSPECS I(4):
STOP-NOW I(4):
GLOBAL 3 I(4):
  ZERO I(4)=GLOBAL:
  ONE I(4)=GLOBAL(3):
  EIGHT I(4)=GLOBAL(5):
  FALSE I(4)=ZERO:
  TRUE I(4)=ONE;

```

```

LIST MAXWINDOWLEN;
LIST COMAREA,INIT:      << Initialize to all zeros >>
  DATABUF:
  DATABUFLEN:
  DONE-WITH-TRANS:
  ERROR-LOCATION:
  FIELDINFO:
  FIELDSPECS-ITEM:
  FILENAME:
  FOUND-DATA-ERRS:
  INFOBUFLEN:
  MSGBUF:
  MSGBUFLEN:
  NBR-TXN-COLLECT:
  NUMENTRIES:
  NUMSPECS:
  STOP-NOW:
  GLOBAL;
SET(OPTION) NOHEAD;

```

<< Sample program main line >>

TRANSACTION

```
PERFORM SETUP-FOR-WORK;
LEVEL;    <<setup loop until done collecting transactions>>
  IF (STOP-NOW) = (TRUE) THEN
    END(LEVEL)
  ELSE IF (DONE-WITH-TRANS) = (TRUE) THEN
    END(LEVEL)
  ELSE
    DO
      PERFORM COLLECT-TRANSACTIONS;
    END;    <<loop to next transaction>>
  DOEND;
PERFORM CLEANUP-AFTER-WORK;
DISPLAY "Deduction transactions collected this session =":
  NBR-TXN-COLLECT;
IF (STOP-NOW) = (TRUE) THEN
  PERFORM DISPLAY-SYSTEM-ERROR;
EXIT;

DISPLAY-SYSTEM-ERROR:
<<*****>>

  DISPLAY "**** Transaction collection facility detected system "
    "error at:":
    ERROR-LOCATION:
    "**** The error message returned is:",line=1:
    "****",line=1:
    MSGBUF;
  RETURN;

CLEANUP-AFTER-WORK:
<<*****>>

  FILE(CLOSE) TXNFILE;
  LET (CSTATUS) = (ZERO);
  PROC VCLOSEFORMF((COMAREA));
  LET (CSTATUS) = (ZERO);
  PROC VCLOSETERM((COMAREA));
  RETURN;

GET-ERROR-MESSAGE:
<<*****>>

  LET (MSGBUFLen) = (MAXWINDOWLEN);
  MOVE (MSGBUF) = " ";
  PROC VERRMSG((COMAREA),%(MSGBUF), (MSGBUFLen), (MSGBUFLen));
  RETURN;

PROMPT-OPERATOR:
<<*****>>

  PERFORM GET-ERROR-MESSAGE;
  PROC VPUTWINDOW((COMAREA),%(MSGBUF), (MSGBUFLen));
  IF (CSTATUS) <> (ZERO) THEN
    DO
```

```

LET (STOP-NOW) = (TRUE);
MOVE (ERROR-LOCATION) = "**** procedure: Prompt Operator - "
                      "Window Load";
PERFORM GET-ERROR-MESSAGE;
RETURN;
DOEND;

```

```

PROC VSHOWFORM((COMAREA));
IF (CSTATUS) <> (ZERO) THEN
DO
LET (STOP-NOW) = (TRUE);
MOVE (ERROR-LOCATION) = "**** procedure: Collect Transactions"
                      " - Display Update";
PERFORM GET-ERROR-MESSAGE;
DOEND;
RETURN;

```

```

READ-EDIT-FILE-TRANSACTION:
<<*****>>

```

```

PROC VREADFIELDS((COMAREA));
IF (CSTATUS) <> (ZERO) THEN
DO
LET (STOP-NOW) = (TRUE);
MOVE (ERROR-LOCATION) = "**** procedure: Read, Edit, and File"
                      " - Terminal Read";
PERFORM GET-ERROR-MESSAGE;
RETURN;
DOEND;

```

```

<< Determine if operator wants to stop transaction collection >>

```

```

IF (LASTKEY) = (EIGHT) THEN
DO
LET (DONE-WITH-TRANS) = (TRUE);
RETURN;
DOEND;

```

```

IF (LASTKEY) <> (ZERO) THEN

  << Operator pressed some key other than Enter or Exit
    so clear data error flag to break loop          >>

DO
LET (FOUND-DATA-ERRS) = (FALSE);
RETURN;
DOEND;

```

```

<< Edit data >>

```

```

PROC VFIELDDEDITS((COMAREA));
IF (CSTATUS) <> (ZERO) THEN
DO
LET (STOP-NOW) = (TRUE);
MOVE (ERROR-LOCATION) = "**** procedure: Read, Edit, and"

```

TRANSACTION

```
                                " File - data Edit";
PERFORM GET-ERROR-MESSAGE;
RETURN;
DOEND;

<< Determine if edit errors >>

IF (NUMERRS) < (ONE) THEN
  LET (FOUND-DATA-ERRS) = (FALSE);
IF (FOUND-DATA-ERRS) = (FALSE) THEN
  DO

    << Finish form data >>

PROC VFINISHFORM((COMAREA));
IF (CSTATUS) <> (ZERO) THEN
  DO
    LET (STOP-NOW) = (TRUE);
    MOVE (ERROR-LOCATION) = "**** procedure: Read, Edit,"
                          " and File - Data Finishing";
    PERFORM GET-ERROR-MESSAGE;
    RETURN;
    DOEND;
  IF (NUMERRS) > (ZERO) THEN
    LET (FOUND-DATA-ERRS) = (TRUE);
  DOEND; << Finish form data >>

<< Do we have a transaction that can be filed? >>

IF (FOUND-DATA-ERRS) = (FALSE) THEN
  DO

    << get transaction from form and file it >>

    MOVE (DATABUF) = " ";
    LET (DATABUFLEN) = 200;
PROC VGETBUFFER((COMAREA),%(DATABUF), (DATABUFLEN));
IF (CSTATUS) <> (ZERO) THEN
  DO
    LET (STOP-NOW) = (TRUE);
    MOVE (ERROR-LOCATION) = "**** procedure: Read, Edit, and"
                          " File - Data Get";
    PERFORM GET-ERROR-MESSAGE;
    RETURN;
    DOEND;
  PUT TXNFILE,LIST=(DATABUF);
  LET (NBR-TXN-COLLECT) = (NBR-TXN-COLLECT) + 1;
  DOEND << Get transaction from form and file it >>
ELSE

<< Prompt the operator to correct errors >>

  PERFORM PROMPT-OPERATOR;
```

RETURN;

COLLECT-TRANSACTIONS:

<<*****>>

<< setup form and get transaction entry form >>

```
LET (REPEATAPP) = (ZERO);
LET (FREEZEAPP) = (ZERO);
MOVE (NFNAME) = "DEDUCTION";
PROC VGETNEXTFORM((COMAREA));
IF (CSTATUS) <> (ZERO) THEN
  DO
    LET (STOP-NOW) = (TRUE);
    MOVE (ERROR-LOCATION) = "**** procedure: Collect Transactions"
                          " - Form Retrieval";
    PERFORM GET-ERROR-MESSAGE;
    RETURN;
  DOEND;
```



<< Toggle three fields in form to "input allowed" >>

<< Screen order is indicated to field change intrinsic as negative number >>

```
LET OFFSET(FIELDSPECS) = 0;
LET OFFSET(FIELDENTRY) = 0;
LET (FIELD-ID) = (SCREEN-ORDER) * [-1];
LET (CHANGE-TYPE) = 5;
MOVE (CHANGE-SPEC) = "0";
LET OFFSET(FIELDSPECS) = 8;
LET OFFSET(FIELDENTRY) = 20;
LET (FIELD-ID) = (SCREEN-ORDER) * [-1];
LET (CHANGE-TYPE) = 5;
MOVE (CHANGE-SPEC) = "0";
LET OFFSET(FIELDSPECS) = 16;
LET OFFSET(FIELDENTRY) = 40;
LET (FIELD-ID) = (SCREEN-ORDER) * [-1];
LET (CHANGE-TYPE) = 5;
MOVE (CHANGE-SPEC) = "0";
LET (NUMSPECS) = 3;
PROC VCHANGEFIELD((COMAREA), (FIELDSPECS-ITEM), (NUMSPECS));
IF (CSTATUS) <> (ZERO) THEN
  DO
    LET (STOP-NOW) = (TRUE);
    MOVE (ERROR-LOCATION) = "**** procedure: Collect Transactions"
                          " - Field Type Updates";
    PERFORM GET-ERROR-MESSAGE;
    RETURN;
  DOEND;
```

<< Load window message >>

TRANSACTION

```
LET (MSGBUFLN) = 79;
MOVE (MSGBUF) = "Fill in Deduction Transaction according to "
               "worksheet";
PROC VPUTWINDOW((COMAREA),%(MSGBUF),(MSGBUFLN));
IF (CSTATUS) <> (ZERO) THEN
  DO
  LET (STOP-NOW) = (TRUE);
  MOVE (ERROR-LOCATION) = "**** procedure: Collect Transactions"
                       " - Window Load";
  PERFORM GET-ERROR-MESSAGE;
  RETURN;
DOEND;
```

<< Init form >>

```
PROC VINITFORM((COMAREA));
IF (CSTATUS) <> (ZERO) THEN
  DO
  LET (STOP-NOW) = (TRUE);
  MOVE (ERROR-LOCATION) = "**** procedure: Collect Transactions"
                       " - Form Init";
  PERFORM GET-ERROR-MESSAGE;
  RETURN;
DOEND;
```

<< Show form >>

```
PROC VSHOWFORM((COMAREA));
IF (CSTATUS) <> (ZERO) THEN
  DO
  LET (STOP-NOW) = (TRUE);
  MOVE (ERROR-LOCATION) = "**** procedure: Collect Transactions"
                       " - Form Display";
  PERFORM GET-ERROR-MESSAGE;
  RETURN;
DOEND;
```

<< Setup and loop on transaction until it can be filed >>

```
LET (FOUND-DATA-ERRS) = (TRUE);
LEVEL;
IF (FOUND-DATA-ERRS) = (FALSE) THEN
  END(LEVEL)
ELSE IF (STOP-NOW) = (TRUE) THEN
  END(LEVEL)
ELSE IF (DONE-WITH-TRANS) = (TRUE) THEN
  END(LEVEL)
ELSE
  DO
  PERFORM READ-EDIT-FILE-TRANSACTION;
  END;
DOEND;
RETURN;
```

SETUP-FOR-WORK:
 <<*****>>

```
LET (ZERO) = 0;
LET (ONE) = 1;
LET (EIGHT) = 8;
LET (STOP-NOW) = (FALSE);
LET (DONE-WITH-TRANS) = (FALSE);
LET (NBR-TXN-COLLECT) = (ZERO);
```

<< Init Comarea >>

<< Set language to SPL. This is the default language if TRANSACTION opens the formfile. However, in this language, all character arrays must be passed as byte addresses, ie %(NAME) >>

```
LET (LANGUAGE) = 3;
LET (COMAREALEN) = 60;
LET (LABELOPTIONS) = 1;
MOVE (CFNAME) = " ";
MOVE (NFNAME) = " ";
LET (FORMSTORESIZE) = 4;
```

<< Open of Transaction file done in SYSTEM statement. Opened so that new records are appended to those already in the file >>

<< Open forms file >>

```
MOVE (FILENAME) = "PAYROLL.VPLUS.MILLER";
PROC VOPENFORMF((COMAREA),%(FILENAME));
IF (CSTATUS) <> (ZERO) THEN
  DO
    LET (STOP-NOW) = (TRUE);
    MOVE (ERROR-LOCATION) = "**** procedure: Setup For Work - "
                          "Forms File Open";
    PERFORM GET-ERROR-MESSAGE;
    RETURN;
  DOEND;
```

<< Open terminal >>

```
MOVE (FILENAME) = "HPTERM";
PROC VOPENTERM((COMAREA),%(FILENAME));
IF (CSTATUS) <> (ZERO) THEN
  DO
    LET (STOP-NOW) = (TRUE);
    MOVE (ERROR-LOCATION) = "**** procedure: Setup For Work - "
                          "Terminal Open";
    PERFORM GET-ERROR-MESSAGE;
    RETURN;
  DOEND;
```

<< Translate field names to screen order >>

TRANSACTION

<< Three of the fields in the form used by this application need to be toggled from "display only" to "input allowed". In order to do this, we first translate field names to screen order. >>

<< Setup to retrieve screen order for three specified fields >>

```
MOVE (CFNAME) = "DEDUCTION";
MOVE (FIELDINFO) = " ";
LET (NUM-ENTRIES) = 3;
LET (ENTRY-LEN) = 10;
MOVE (FORM-NAME) = "DEDUCTION";
LET OFFSET(FIELDENTRY) = 0;
MOVE (FIELD-NAME) = "BADGE NUMBER";
LET OFFSET(FIELDENTRY) = 20;
MOVE (FIELD-NAME) = "LAST NAME";
LET OFFSET(FIELDENTRY) = 40;
MOVE (FIELD-NAME) = "SUR_NAME";
```

<< Set length of entire info buffer >>

```
LET (INFOBUFLN) = (NUM-ENTRIES) * (ENTRY-LEN) + 10;
```

```
PROC VGETFIELDINFO((COMAREA), (FIELDINFO), (INFOBUFLN));
```

```
IF (CSTATUS) <> (ZERO) THEN
```

```
DO
```

```
LET (STOP-NOW) = (TRUE);
```

```
MOVE (ERROR-LOCATION) = "**** procedure: Setup For Work - "
"Field Info Retrieval";
```

```
PERFORM GET-ERROR-MESSAGE;
```

```
DOEND;
```

```
RETURN;
```

```
$TITLE 'VPLUS/V Data Entry Sample Program'$
```

```
{
This application collects employee payroll deduction transactions
and places the edited transactions into a file.
```

```
For this application:      Enter key = edit and file transaction;

                           f8 = exit application;

                           all other f keys = redo transaction.
```

```
Each transaction entered by the operator is subjected to the data
edits embedded within the input form.
```

```
The application continues to collect transactions until either
the operator signals to exit or a system error is detected.
}
```

```
PROGRAM Pascal_Sample (output);
```

```
CONST
```

```
  MAXWINDOWLEN      = 150;
  FILENAMELEN       = 86;
```

```
TYPE
```

```
  SMALL_INT         = -32768..32767

  PAC_2             = PACKED ARRAY [1..2] OF CHAR;
  PAC_4             = PACKED ARRAY [1..4] OF CHAR;
  PAC_8             = PACKED ARRAY [1..8] OF CHAR;
  PAC_16            = PACKED ARRAY [1..16] OF CHAR;
  PAC_70            = PACKED ARRAY [1..70] OF CHAR;
  PAC_80            = PACKED ARRAY [1..80] OF CHAR;
  PAC_200           = PACKED ARRAY [1..200] OF CHAR;
  PAC_FILENAME      = PACKED ARRAY [1..FILENAMELEN] OF CHAR;
  PAC_MAXWINDOWLEN = PACKED ARRAY [1..MAXWINDOWLEN]
                    OF CHAR;
```

```
TWO_BYTE_SUB_RANGE = PACKED ARRAY [1..2] OF 0..255;
```

```
COMAREA_REC = RECORD
```

```
  CSTATUS          : SMALL_INT;
  LANGUAGE          : SMALL_INT;
  COMAREALEN       : SMALL_INT;
  USERBUFLN       : SMALL_INT;
  CMODE            : SMALL_INT;
  LASTKEY          : SMALL_INT;
  NUMERRS          : SMALL_INT;
  WINDOWENH       : SMALL_INT;
  MULTIUSAGE       : SMALL_INT;
  LABELOPTIONS     : SMALL_INT;
```

PASCAL

```
CFNAME      : PAC_16;  
NFNAME      : PAC_16;  
REPEATAPP   : SMALL_INT;  
FREEZEAPP   : SMALL_INT;  
CFNUMLINES  : SMALL_INT;  
DBUFLEN     : SMALL_INT;  
SKIP_31     : SMALL_INT;  
LOOKAHEAD   : SMALL_INT;  
DELETEFLAG  : SMALL_INT;  
SHOWCONTROL : SMALL_INT;  
SKIP_35     : SMALL_INT;  
PRINTFILENUM : SMALL_INT;  
FILERRNUM   : SMALL_INT;  
ERRFILENUM  : SMALL_INT;  
FORMSTORESIZE : SMALL_INT;  
SKIP_40     : SMALL_INT;  
SKIP_41     : SMALL_INT;  
SKIP_42     : SMALL_INT;  
NUMRECS     : INTEGER;  
RECNUM      : INTEGER;  
SKIP_47     : SMALL_INT;  
SKIP_48     : SMALL_INT;  
TERM_FILEN  : SMALL_INT;  
SKIP_50     : SMALL_INT;  
SKIP_51     : SMALL_INT;  
SKIP_52     : SMALL_INT;  
SKIP_53     : SMALL_INT;  
SKIP_54     : SMALL_INT;  
RETRIES     : SMALL_INT;  
TERM_OPTIONS : SMALL_INT;  
ENVIRON     : SMALL_INT;  
USERTIME    : SMALL_INT;  
IDENTIFIER  : SMALL_INT;  
LABELINFO   : TWO_BYTE_SUB_RANGE;  
END;
```

FIELDENTRY_REC = PACKED RECORD

```
FIELD_NAME      : PAC_16;  
SCREEN_ORDER    : SMALL_INT;  
END;
```

FIELDINFO_REC = PACKED RECORD

```
NUM_ENTRIES     : SMALL_INT;  
ENTRY_LEN       : SMALL_INT;  
FORM_NAME       : PAC_16;  
FIELDENTRY      : PACKED ARRAY [1..3] OF FIELDENTRY_REC;  
END;
```

FIELDSPECS_REC = RECORD

```
FIELD_ID        : SMALL_INT;  
CHANGE_TYPE     : SMALL_INT;  
CHANGE_SPEC     : PAC_4;  
END;
```

CONST

```

LABELINFO_INIT = TWO_BYTE_SUB_RANGE [2 OF 0];
{ Comarea initialization constant record }

COMAREA_INIT = COMAREA_REC
[ CSTATUS      : 0,
  LANGUAGE      : 5,          {Pascal  }
  COMAREALEN    : 60,
  USERBUFLLEN  : 0,
  CMODE        : 0,
  LASTKEY      : 0,
  NUMERRS      : 0,
  WINDOWENH    : 0,
  MULTIUSAGE   : 0,
  LABELOPTIONS : 1,          {activate labels}
  CFNAME       : '',
  NFNAME       : '',
  REPEATAPP    : 0,
  FREEZEAPP    : 0,
  CFNUMLINES   : 0,
  DBUFLEN      : 0,
  SKIP_31      : 0,
  LOOKAHEAD    : 0,          {no form background loading}
  DELETEFLAG   : 0,
  SHOWCONTROL  : 0,
  SKIP_35      : 0,
  PRINTFILENUM : 0,
  FILERRNUM    : 0,
  ERRFILENUM   : 0,
  FORMSTORESIZE : 4,          {local form storage}
  SKIP_40      : 0,
  SKIP_41      : 0,
  SKIP_42      : 0,
  NUMRECS      : 0,
  RECNUM       : 0,
  SKIP_47      : 0,
  SKIP_48      : 0,
  TERM_FILEN   : 0,
  SKIP_50      : 0,
  SKIP_51      : 0,
  SKIP_52      : 0,
  SKIP_53      : 0,
  SKIP_54      : 0,
  RETRIES      : 0,
  TERM_OPTIONS : 0,
  ENVIRON      : 0,
  USERTIME     : 0,
  IDENTIFIER   : 0,
  LABELINFO    : LABELINFO_INIT ];

VAR
COMAREA      : COMAREA_REC;
DATABUF     : PAC_200;
DATABUFLEN  : SMALL_INT;
DONE_WITH_TRANSACTIONS : BOOLEAN;

```

PASCAL

```
ERROR_LOCATION      : PAC_70;
FIELDINFO           : FIELDINFO_REC;
FIELDSPECS          : ARRAY [1..3] OF FIELDSPECS_REC;
FILENAME            : PAC_FILENAME;
FOUND_DATA_ERRORS  : BOOLEAN;
INFOBUFLEN         : SMALL_INT;
MSGBUF              : PAC_MAXWINDOWLEN;
MSGBUFLen          : SMALL_INT;
ERRMSGLEN           : SMALL_INT;
NBR_TXN_COLLECTED  : INTEGER;
NUMSPECS            : SMALL_INT;
STOP_NOW            : BOOLEAN;
TXN_FILE            : TEXT;
```

{ Procedure Declarations }

```
PROCEDURE VCHANGEFIELD ; INTRINSIC;
PROCEDURE VCLOSEFORMF ; INTRINSIC;
PROCEDURE VCLOSETERM ; INTRINSIC;
PROCEDURE VERRMSG ; INTRINSIC;
PROCEDURE VFIELDEDITS ; INTRINSIC;
PROCEDURE VFINISHFORM ; INTRINSIC;
PROCEDURE VGETBUFFER ; INTRINSIC;
PROCEDURE VGETFIELDINFO ; INTRINSIC;
PROCEDURE VGETNEXTFORM ; INTRINSIC;
PROCEDURE VINITFORM ; INTRINSIC;
PROCEDURE VOPENFORMF ; INTRINSIC;
PROCEDURE VOPENTERM ; INTRINSIC;
PROCEDURE VPUTWINDOW ; INTRINSIC;
PROCEDURE VREADFIELDS ; INTRINSIC;
PROCEDURE VSHOWFORM ; INTRINSIC;
```

```
PROCEDURE DISPLAY_SYSTEM_ERROR;
```

```
BEGIN
```

```
WRITELN ('**** Transaction collection facility detected system ',
         'error at: ',
         ERROR_LOCATION);
```

```
WRITELN ('**** The error messaged returned is:');
WRITELN ('**** ', MSGBUF : MSGBUFLen);
```

```
END;
```

```
PROCEDURE CLEANUP_AFTER_WORK;
```

```
{ Note that this procedure unconditionally attempts to close
  the forms file and terminal }
```

```
BEGIN
```

```
CLOSE (TXN_FILE);
```

```
COMAREA.CSTATUS := 0;
```

```
VCLOSEFORMF (COMAREA);
```

```
COMAREA.CSTATUS := 0;
```

```
VCLOSETERM (COMAREA);
```

```
END;
```

```
PROCEDURE GET_ERROR_MESSAGE;
```

```
BEGIN
```

```
MSGBUFLen := MAXWINDOWLEN;
```

```
VERRMSG (COMAREA,
         MSGBUF,
         MSGBUFLen,
         ERRMSGLEN);
```

```
END;
```

```
PROCEDURE PROMPT_OPERATOR;
```

```
BEGIN
```

```
GET_ERROR_MESSAGE;
```

```
VPUTWINDOW (COMAREA,
            MSGBUF,
            ERRMSGLEN);
```

```
IF COMAREA.CSTATUS <> 0 THEN
```

```
  BEGIN
```

```
    STOP_NOW := TRUE;
```

```
    ERROR_LOCATION
```

```
      := '**** procedure: Prompt Operator - Window Load';
```

```
    GET_ERROR_MESSAGE;
```

```
  END;
```

```
IF NOT (STOP_NOW) THEN
```

```
  BEGIN
```

PASCAL

```
{ Display update }
```

```
VSHOWFORM (COMAREA);
```

```
IF COMAREA.CSTATUS <> 0 THEN  
  BEGIN  
    STOP_NOW := TRUE;  
    ERROR_LOCATION  
      := '**** procedure: Collect Transactions - Display Update';  
    GET_ERROR_MESSAGE;  
    END;
```

```
END; { Display update }
```

```
END; {Procedure Prompt Operator }
```

```
PROCEDURE READ_EDIT_FILE_TRANSACTION;
```

```
BEGIN
```

```
VREADFIELDS (COMAREA);
```

```
IF COMAREA.CSTATUS <> 0 THEN  
  BEGIN  
    STOP_NOW := TRUE;  
    ERROR_LOCATION  
      := '**** procedure: Read, Edit, and File - Terminal Read';  
    GET_ERROR_MESSAGE;  
    END;
```

```
IF NOT (STOP_NOW) THEN
```

```
  { Determine if operator wants to stop transaction collection }
```

```
  IF COMAREA.LASTKEY = 8 THEN  
    DONE_WITH_TRANSACTIONS := TRUE;
```

```
IF (NOT (STOP_NOW))  
AND (NOT (DONE_WITH_TRANSACTIONS)) THEN  
  BEGIN
```

```
  { Edit data }
```

```
VFIELDDEDITS (COMAREA);
```

```
IF COMAREA.CSTATUS <> 0 THEN  
  BEGIN  
    STOP_NOW := TRUE;  
    ERROR_LOCATION  
      := '**** procedure: Read, Edit, and File - Data Edit';  
    GET_ERROR_MESSAGE;  
    END;
```

```
IF NOT (STOP_NOW) THEN
```

```

    { Determine if edit errors }

    IF COMAREA.NUMERRS < 1 THEN
        FOUND_DATA_ERRORS := FALSE; { Was true before }

    END; { Edit data }
    IF (NOT (STOP_NOW))
    AND (NOT (DONE_WITH_TRANSACTIONS))
    AND (NOT (FOUND_DATA_ERRORS)) THEN
        BEGIN

            { Finish form data }

            VFINISHFORM (COMAREA);

            IF COMAREA.CSTATUS <> 0 THEN
                BEGIN
                    STOP_NOW := TRUE;
                    ERROR_LOCATION
                        := '**** procedure: Read, Edit, and File - Data Finishing';
                    GET_ERROR_MESSAGE;
                    END;

            IF NOT (STOP_NOW) THEN

                { Determine if finishing errors }

                IF COMAREA.NUMERRS > 0 THEN
                    FOUND_DATA_ERRORS := TRUE; { Was false before }

                END; { Finish form data }
                IF (NOT (STOP_NOW))
                AND (NOT (DONE_WITH_TRANSACTIONS)) THEN

                    { Do we have a transaction that can be filed? }

                    IF NOT (FOUND_DATA_ERRORS) THEN

                        IF COMAREA.LASTKEY = 0 { Enter key } THEN
                            BEGIN

                                { Get transaction from form and file it }

                                DATABUF := '';

                                DATABUFLEN := 200;

                                VGETBUFFER (COMAREA,
                                            DATABUF,
                                            DATABUFLEN);

                                IF COMAREA.CSTATUS <> 0 THEN
                                    BEGIN
                                        STOP_NOW := TRUE;

```

PASCAL

```
        ERROR_LOCATION
          := '**** procedure: Read, Edit, and File - Data Get';
        GET_ERROR_MESSAGE;
        END;

    IF NOT (STOP_NOW) THEN
        BEGIN

            WRITELN (TXN_FILE, DATABUF : DATABUFLEN);

            NBR_TXN_COLLECTED := NBR_TXN_COLLECTED + 1;

            END;

        END; { Get transaction from form and file it }
    IF (NOT (STOP_NOW))
        AND (NOT (DONE_WITH_TRANSACTIONS)) THEN

        { Do we need to prompt the operator to correct errors? }

        IF FOUND_DATA_ERRORS THEN

            IF COMAREA.LASTKEY = 0 { Enter key } THEN
                PROMPT_OPERATOR;

    IF (NOT (STOP_NOW))
        AND (NOT (DONE_WITH_TRANSACTIONS)) THEN

        { Do we need to refresh the display? }

        IF COMAREA.LASTKEY <> 0 THEN

            { Operator pressed some key other than Enter or Exit
              so clear data error flag to break loop }

            FOUND_DATA_ERRORS := FALSE;

    END; { Procedure Read, Edit, and File Transaction }

    PROCEDURE COLLECT_TRANSACTIONS;

    BEGIN

        { Setup for and get transaction entry form }

        COMAREA.REPEATAPP := 0;
        COMAREA.FREEZEAPP := 0;

        COMAREA.NFNAME := 'DEDUCTION';

        VGETNEXTFORM (COMAREA);

        IF COMAREA.CSTATUS <> 0 THEN
            BEGIN
```

```

STOP_NOW := TRUE;
ERROR_LOCATION
:= '**** procedure: Collect Transactions - Form Retrieval';
GET_ERROR_MESSAGE;
END;

IF NOT (STOP_NOW) THEN
BEGIN

{ Toggle three fields in form to "input allowed" }

{ Screen order is indicated to field change intrinsic
as negative number }

FIELDSPECS [1].FIELD_ID
:= FIELDINFO.FIELDENTRY [1].SCREEN_ORDER * (-1);
FIELDSPECS [1].CHANGE_TYPE := 5; { change field type }
FIELDSPECS [1].CHANGE_SPEC := '0'; { input and output }

FIELDSPECS [2].FIELD_ID
:= FIELDINFO.FIELDENTRY [2].SCREEN_ORDER * (-1);
FIELDSPECS [2].CHANGE_TYPE := 5; { change field type }
FIELDSPECS [2].CHANGE_SPEC := '0'; { input and output }

FIELDSPECS [3].FIELD_ID
:= FIELDINFO.FIELDENTRY [3].SCREEN_ORDER * (-1);
FIELDSPECS [3].CHANGE_TYPE := 5; { change field type }
FIELDSPECS [3].CHANGE_SPEC := '0'; { input and output }

NUMSPECS := 3;

VCHANGEFIELD (COMAREA,
              FIELDSPECS,
              NUMSPECS);

IF COMAREA.CSTATUS <> 0 THEN
BEGIN
STOP_NOW := TRUE;
ERROR_LOCATION
:= '**** procedure: Collect Transactions - Field Type Updates';
GET_ERROR_MESSAGE;
END;

END; { Toggle three fields in form }
IF NOT (STOP_NOW) THEN
BEGIN

{ Load window message }

MSGBUFLen := 79;

MSGBUF
:= 'Fill in Deduction Transaction according to worksheet.';

```

PASCAL

```
VPUTWINDOW (COMAREA,  
            MSGBUF,  
            MSGBUFLN);
```

```
IF COMAREA.CSTATUS <> 0 THEN  
  BEGIN  
    STOP_NOW := TRUE;  
    ERROR_LOCATION  
      := '**** procedure: Collect Transactions - Window Load';  
    GET_ERROR_MESSAGE;  
    END;
```

```
END; { Load window message }
```

```
IF NOT (STOP_NOW) THEN  
  BEGIN
```

```
{ Init form }
```

```
VINITFORM (COMAREA);
```

```
IF COMAREA.CSTATUS <> 0 THEN  
  BEGIN  
    STOP_NOW := TRUE;  
    ERROR_LOCATION  
      := '**** procedure: Collect Transactions - Form Init';  
    GET_ERROR_MESSAGE;  
    END;
```

```
END; { Init form }
```

```
IF NOT (STOP_NOW) THEN  
  BEGIN
```

```
{ Show form }
```

```
VSHOWFORM (COMAREA);
```

```
IF COMAREA.CSTATUS <> 0 THEN  
  BEGIN  
    STOP_NOW := TRUE;  
    ERROR_LOCATION  
      := '**** procedure: Collect Transactions - Form Display';  
    GET_ERROR_MESSAGE;  
    END;
```

```
END; { Show form }
```

```
IF NOT (STOP_NOW) THEN  
  BEGIN
```

```
{ Setup and loop on transaction until it can be filed }
```

```
FOUND_DATA_ERRORS := TRUE;
```

```

WHILE      (FOUND_DATA_ERRORS)
  AND      (NOT (STOP_NOW))
  AND      (NOT (DONE_WITH_TRANSACTIONS)) DO
  READ_EDIT_FILE_TRANSACTION;

END;

END; { Procedure Collect Transactions }

PROCEDURE SETUP_FOR_WORK;

BEGIN

{ Init Comarea }

COMAREA := COMAREA_INIT;

{ Open Transaction File so that new transactions are
  added to those already in the file }

APPEND (TXN_FILE, 'PAYTXN');

{ Open forms file }

FILENAME := 'PAYROLL.WORK.ADMIN';

VOPENFORM (COMAREA,
           FILENAME);

IF COMAREA.CSTATUS <> 0 THEN
  BEGIN
  STOP_NOW := TRUE;
  ERROR_LOCATION
    := '**** procedure: Setup For Work - Forms File Open';
  GET_ERROR_MESSAGE;
  END;

IF NOT (STOP_NOW) THEN
  BEGIN

  { Open terminal }

  FILENAME := 'HPTERM';

  VOPENTERM (COMAREA,
            FILENAME);

  IF COMAREA.CSTATUS <> 0 THEN
    BEGIN
    STOP_NOW := TRUE;
    ERROR_LOCATION
      := '**** procedure: Setup For Work - Terminal Open';
    GET_ERROR_MESSAGE;
    END;

```

PASCAL

```
END; { Open terminal }
IF NOT (STOP_NOW) THEN
BEGIN

  { Translate field names to screen orders }

  { Three of the fields in the form used by this application
  need to be toggled from "display only" to "input allowed".
  In order to do this, we first translate field names to
  screen orders. }

  { Setup to retrieve screen order for three specified fields }

  FIELDINFO.NUM_ENTRIES := 3;

  FIELDINFO.ENTRY_LEN := 9; { Field name key and screen order }

  FIELDINFO.FORM_NAME := 'DEDUCTION';

  FIELDINFO.FIELDENTRY [1].FIELD_NAME := 'BADGE_NUMBER';
  FIELDINFO.FIELDENTRY [1].SCREEN_ORDER := 8224; {ASCII blanks}

  FIELDINFO.FIELDENTRY [2].FIELD_NAME := 'LAST_NAME';
  FIELDINFO.FIELDENTRY [2].SCREEN_ORDER := 8224; {ASCII blanks}

  FIELDINFO.FIELDENTRY [3].FIELD_NAME := 'SUR_NAME';
  FIELDINFO.FIELDENTRY [3].SCREEN_ORDER := 8224; {ASCII blanks}

  { Set length of entire info buffer }

  INFOBUFLen
    := (FIELDINFO.NUM_ENTRIES * FIELDINFO.ENTRY_LEN) + 10;

  VGETFIELDINFO (COMAREA,
                FIELDINFO,
                INFOBUFLen);

  IF COMAREA.CSTATUS <> 0 THEN
  BEGIN
    STOP_NOW := TRUE;
    ERROR_LOCATION
      := '**** procedure: Setup For Work - Field Info Retrieval';
    GET_ERROR_MESSAGE;
    END;

  END; { Translate field names to screen orders }

END; { Procedure Setup For Work }

BEGIN

{ Sample program outer block }
```

```
STOP_NOW := FALSE;
DONE_WITH_TRANSACTIONS := FALSE;

NBR_TXN_COLLECTED := 0;

SETUP_FOR_WORK;

WHILE (NOT (STOP_NOW))
  AND (NOT (DONE_WITH_TRANSACTIONS)) DO
  COLLECT_TRANSACTIONS;

CLEANUP_AFTER_WORK;

WRITELN ('Deduction transactions collected this session = ',
  NBR_TXN_COLLECTED);

IF STOP_NOW THEN
  DISPLAY_SYSTEM_ERROR;

END.
```



VPLUS/V ERROR MESSAGES

APPENDIX

B

Any error diagnosed during execution of VPLUS/V 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. When the number following the word FSERR is negative, refer to the appropriate group (FORMSPEC, REFSPEC, etc.) of "Access Method Errors" listed below. The word COM'STATUS indicates an error executing one of the VPLUS/V intrinsics.

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 encountered, most users should call the system manager for advice.

ERROR MESSAGES

These messages are issued by:

FORMSPEC
REFSPEC
User callable VPLUS/V intrinsics
REFORMAT

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. Increase the MAXDATA parameter on the PREP or RUN command. Check the FORMSPEC listing for directory size.
- 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/Action
101	Form is too big, overflowed the 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.

VPLUS/V Error Messages

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. (COM'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	Error writing the screen source. File full? (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 intrinsics 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	Error writing the global screen record. File full? (FSERR !!!!)	See error 123.
129	Internal error: Allocate of STBL failed.	Could not allocate stack space for symbol table.
130	Error writing STBL to the forms file. File full? (FSERR !!!!)	See error 123.
131	INTERNAL ERROR: changing screen of family.	
132	Only 128 fields are allowed in a form.	Reduce the number of fields in the form.

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?	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 unreferenced 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.
324	Internal error: CFG ALLOCATE for form ! failed.	
325	The field number cannot be more than 256.	Too many modifications have been made to the screen for this form. The form must be rebuilt from scratch or renumbered in batch.

Menu Processing Utility Errors

Error	Message	Cause/Action
601	Must be B, H, I, U, 1-8, 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	Error writing globals buffer. File full? (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.

VPLUS/V Error Messages

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.	Refer to the list of the FORMSPEC reserved words.
614	You can specify only one color for the field.	
615	Color pair value must be between 1 and 8.	

Menu Init 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.
(FSERR !!!!!)
- 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. In batch mode, form not found in current forms file.
- 723 Selection must be one of the characters listed below. Enter one of the specified characters in the MAIN menu selection box.

VPLUS/V Error Messages

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	Err writing form int'm. rec for form !. File Full? (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.
728	File does not exist. Press ENTER to create file.	
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 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 VPLUS 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. File full? (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 VPLUS Fast Forms File.	The fast forms file name you specified is not an existing VPLUS fast forms file.
761	This file is not a VPLUS 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.

VPLUS/V Error Messages

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 child 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.
768	List operation failed.	A write to file FORMLIST failed. Check for a valid file equation or for spooling problems.

FORMSPEC in batch mode errors

Error	Message	Cause/Action
769	Invalid batch mode command.	FORMSPEC expected a valid batch mode command or parameter.
770	Warning: Forms file modified and not compiled.	The forms file must be compiled before it can be used.
771	Could not open command file.	Check file name, security, or if in use elsewhere.
772	Could not open list file.	Check file name, security, or if in use elsewhere.
744	In form !, field !, "!" is undefined:	An error was found in a field during compilation. (See next error message printed.)
775	Internal error in form !, field !:	An error was found in a field during compilation. (See next error message printed.)
776	Form !, field ! has semantic error:	An error was found in a field during compilation. The syntax of the statement in the field may be correct, but its meaning is wrong. (See next error message printed.)
777	Form !, field ! has a data type error:	An error was found in a field during compilation. Data type inconsistent with specification. (See next error

778	Form !, field ! has a syntactic error:	Statement syntax is incorrect. (See next error message printed.)
780	Commands must be preceded by a FILE command.	Current forms file must be specified.
781	Missing form name.	Form name must be specified.
782	Missing forms file name.	Forms file name must be specified.
783	Missing fast forms file name.	Fast forms file name must be specified when "INTO" is used.
784	Too many arguments.	Too many parameters were specified for the command.
785	Expecting "TO form-name".	The form name must be specified when "TO" is used.
786	Expecting "IN forms-file" or "TO form-name".	COPY command must include either "IN file" or "TO newform".
787	Expecting "INTO fast-forms-file".	The only option for the COMPILE command is "INTO" followed by the fast forms file name.
788	Internal error in FORMSPEC processing.	See System Manager.
789	All commands ingored until next FILE command.	Either a command was not recognized or an error occured with a FILE command.
790	Compile failed.	

RELATE Command Errors

791	Child cannot have the same name as parent.	Same form name was specified as the parent and child form.
792	Form is already child of that parent.	The forms are already related.
793	Child form is already related to another parent.	A child cannot have more than one parent.
794	Form specified as child cannot have any children.	A child form cannot be a parent.
795	Form layout of child must be the same as parent.	The form layouts must be identical, including field tags.

VPLUS/V Error Messages

796	Field numbers of child must be the same as parent.	When the field numbers do not match, the forms cannot be related.
797	Correct syntax is: "RELATE child-form-name TO parent-form-name".	Arguments in the batch mode command were mis-typed.
798	KSAM forms files can no longer be modified - use CONVERT.PUB.SYS.	Refer to Appendix H for instructions.
799	Cannot change color forms file on a non-color terminal.	
800	Internal error in color conversion.	

Menu Controller Errors

Error	Message	Cause/Action
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 VGETBUF. (COM'STATUS !!!!)	
809	Error writing source to forms file. File full? (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
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.
- 823 Enter "Y" to go to
function key labels menu To define global function key
labels, enter "Y" on the last
line of the GLOBALS menu. To
define form function key labels,
enter "Y" on the Function Key
Label line of the FORM menu.
- 824 Pause interval may not
be negative. The Split Message Pause must
be between 1 and 99.
- 825 Invalid error light
(A ... P, @). Enter a valid error light (a
letter of the alphabet or the
@ character).
- 826 Invalid Multifunction
reader definition (MARKS
or HOLES). Enter either of the valid
Multifunction reader definitions.
- 827 Field may only have the
values "YFS" or "NO". The valid options for Corner
Cut Required are YES or NO.

VPLUS/V Error Messages

828	Illegal combination of options for the Multifunction reader.	The combinations HOLES with Clock On and MARKS with NONE are not valid.
829	Invalid Multifunction reader definition (COD, CAD, NONE).	Enter one of the valid Multifunction reader definitions.
830	Invalid barcode type (UPC, EAN, I25, I39, MAT, ILV).	Enter one of the valid Barcode Reader types.
831	Internal error: Writing device intermediate record (FSERR !!!!!)	
832	Invalid combination of PAUSE INTERVAL and WAIT FOR USER	If Split Message Pause is specified, Wait for User must be NO.
833	Enter "CONTINUE" only, or press a function key	Only CONTINUE may be entered in this field.
834	NEXT is not appropriate until the forms file has been adjusted	The specified forms file is an old version and must be adjusted before going further.
835	Place name of form to be listed within List Form field.	
836	Enter "Y" or "N" only.	
837	Enter "Y" to go to Data Type Conversion menu.	

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. (COM'STATUS !!!!!)	

905

Internal error closing
the terminal file.
(COM'STATUS !!!!)

- 909 Error writing the global code record. File full? (FSERR !!!!!) See error 123.
- 911 Error writing form code record for form !. File full? (FSERR !!!) See error 123.
- 912 Internal error reading screen source for form !. (FSERR !!!)
- 913 Internal error closing your forms file. (FSERR !!!!!)
- 915 Initial value contains system constant.
- 916 In form !, the field name ! is not unique. Duplicate field name found in indicated form; use FIELD menu to change field name.
- 917 Error writing field names table for form !. File full? (FSERR !!!)
- 918 Error writing save field table for form !. File full? (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, 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.

VPLUS/V Error Messages

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 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	More left braces than right braces were found.	Braces must match; add right brace, or delete extra left brace.
1011	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.

VPLUS/V Error Messages

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 a comment starting with a \.
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 is 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 expression.	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 data 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 a 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 CHAR 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.

VPLUS/V Error Messages

- | | | |
|------|--|--|
| 1120 | Too many statements, code for entire form is too big. | At compilation, the form information generated does not fit into the maximum record size of 12000 bytes. Reduce the number of fields or processing specifications. |
| 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 inconsistent. | |
| 1925 | Internal error: MEM size is inconsistent with ARRAYSIZE. | |
| 1926 | Internal error: XEVAL recursion underflow. | |
| 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. | |
| 1931 | Internal error: INEXT conversion failed. | |
| 1932 | Internal error: Operand must be character string constant. | |

- 1932 Internal error: Operand must be character string constant.
- 1933 Internal error: Invalid operand in APPLY.
- 1934 Internal error: Invalid operand in type check.
- 1935 Internal error: Invalid operand in MATCH.
- 1936 Internal error: Invalid operand in CDIGIT.
- 1937 Internal error: Invalid right operand.
- 1938 Internal error: Invalid left operand.
- 1939 Internal error: Invalid source operand.
- 1940 Internal error: Invalid Dtype in type checking.
- 1941 Internal error: EXTIN conversion failed.
- 1942 Internal error: Bad id number.
- 1943 Internal error: Bad screen order number.
- 1944 Internal error: MEM dispose failed.
- 1945 Internal error: Lexic level underflow.
- 1946 Internal error: Invalid leading blanks.
- 1947 Internal error: Invalid Dtype in scanner.
- 1948 Internal error: MEM allocate failed.
- 1949 Internal error: Invalid production.

VPLUS/V Error Messages

- 1950 Internal error: Scanner error.
- 1951 Invalid local edit combination. An invalid combination of local edits has been specified in the CONFIG phase of the field processing specifications.

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	Invalid character found in pattern.	A character in the expression is not recognized as a valid character in the pattern.
2104	End of pattern found prematurely.	Pattern characters were found after the end of the pattern was found.
2105	Second character in range is lower than first.	A range (a:b) is transposed, or else a colon was not preceded by an exclamation point.
2203	Pattern is too big.	Either split the pattern into smaller patterns (with IF statements), or simplify the pattern.
2900	Internal error: COMPILE'PATTERN failed.	

FORMSPEC B.05.xx Release Errors

Error	Message	Action
3097	Invalid data type conversion notation.	Enter the correct data type to which the specified data type should be converted.
3098	Data Type Conversion Record has not been defined.	Use the Data Type Conversions Menu to set the default data type conversions.

List Process Errors

Error	Message	Action
3100	Internal error: Could not create list process.	
3101	Internal error: Could not send mail to list process.	
3102	Internal error: Could not activate list process.	
3103	Internal error: Mail from list was lost.	
3104	Formsfile reopen failed. Press PREV to reopen. (FSERR !!!!!)	
3105	List could not open the forms file. (FSERR !!!!!)	
3106	Internal error : List could not close the formsfile.	
3107	Internal error: could not flush current form to the forms file.	
3108	Internal error: could not flush the field table to the forms file.	
3109	Internal error: could not flush the globals to the forms file.	
3110	Internal error: could not close the formsfile.	
3111	List could not open the FORMSPEC formsfile. (FSERR !!!!!)	
3112	Internal error: List could not close the FORMSPEC formsfile.	
3113	List could not find form ! in the FORMSPEC formsfile.	
3114	List write failed. (FSERR !!!!!)	

VPLUS/V Error Messages

3115 FORMSPECLINESPERPAGE JCW must
be between 1 and 150.
List failed.

3116 FORMSPECLINESPERPAGE JCW must
be between 1 and 150.

Application-ready Buffer (ARB) Errors

Error	Message	Cause/Action
3403	Internal error: ARB detail record missing.	
3405	This field must not be blank.	Enter the required information in the field.
3406	This field accepts only contiguous digits.	Remove any spaces between the digits.
3407	Must specify number of decimals here.	
3409	Internal error: could not update ARB Detail record.	
3412	Internal error: Field Symbol Table record missing.	
3413	The form has no fields.	You cannot reproduce the form fields on the ARB because they do not exist.
3414	Internal error: Data Type Conversion record missing.	
3416	Internal error: could not create ARB Detail record.	
3418	Internal error: could not create ARB Map record.	
3420	An ARB already exists.	You cannot create two ARBs for one form. If you want to create a new ARB, you must delete the old one first.

- 3423 Warning: run-time errors may result from default data conversions. You may have specified incompatible data conversions. Check your data type conversion record to make sure the data conversions are legitimate. If your code has made provision for these anomalies, you may disregard the warning.
- 3424 Warning: ARB length adjusted to match data type. You specified a length for an ARB field that was too long or too short.
- 3425 Length must be greater than zero.
- 3426 Field was not found in ARB.
- 3429 ARB does not exist.
- 3432 Selection must be one of the abbreviations listed below.
- 3450 First field of range is missing; specify a name or number.
- 3451 Number specified is not in current ARB range.
- 3452 ARB is full; no more fields may be added. An ARB may hold up to 256 fields.
- 3453 Please enter a valid ARB field name. Check the syntax of the name you have entered.
- 3455 Range is reversed; try 'leading-field/trailing-field' You have reversed the order of the fields you want to manipulate. The field that appears first on the ARB must appear first in the command.
- 3456 Please enter a valid command (Add, Move, Rename or DELETE).
- 3457 That command cannot be abbreviated. Please try again.
- 3458 ARB field name is too long; maximum length is 15 characters.

VPLUS/V Error Messages

- | | | |
|------|---|--|
| 3459 | Please specify Before or After, or leave blank. | |
| 3460 | ARB field name to be added already exists in ARB. | You cannot duplicate an ARB field name; choose a new name or delete the other field first. |
| 3461 | Internal error: Selection routine approved an unknown command. | |
| 3462 | New {Destination} name already exists in ARB. | Select a name that does not already exist. |
| 3465 | A destination entry requires that Before or After be specified. | If you do not want to specify Before or After, you must leave the Destination field blank. |
| 3466 | An ARB field name must start with an alphabetic character. | |
| 3467 | No imbedded blanks are allowed in an ARB field name. | |
| 3468 | Legal ARB field name characters are A-Z, 0-9, and ' - '. | |
| 3469 | An ARB field name cannot be a reserved word. | Choose a new name for the field. |
| 3473 | ARB field name must be specified; numeric reference is not allowed. | |
| 3474 | Second field of range is missing; specify ARB field name or number. | |
| 3475 | Internal error: SearchRec did not locate field in ARB Add | |
| 3476 | A range cannot be moved to a destination within itself. | |
| 3498 | Internal error: VGETBUFFER failed (COM'STATUS !!!!) | |

3499 Internal error: VPUTBUFFER
failed. (COM'STATUS !!!!!)

NOTE

In most cases, the corrective action is clearly implied by the message. "Internal Errors" are serious; they imply a corrupted forms file and/or a program error. Call your SE.

ARB Compile Errors

Error	Message	Action
3500	In ARB !, field ! has no corresponding screen field.	Either delete the ARB field, or add a field to the associated screen.
3501	In ARB !, field ! has an incorrect length.	Change the ARB field length to match the length of the corresponding screen field.
3502	In ARB !, field ! has an incorrect ARB type.	Check the permitted conversion types and correct the ARB data type.
3503	Internal error: In ARB !, missing source record for field !.	

More Menu Init and Processing Errors

Error	Message	Cause/Action
7001	Correct syntax is: "FIELD formname fieldtag/name" then menu specs.	
7002	Unable to open screen file.	
7003	WARNING: No new screenfile specified. Old screenfile was closed.	
7004	Correct syntax is: "ADD formname [\comments]".	
7005	Correct syntax is: "READD formname [newformname] [\comments]".	
7006	A screen layout file is not currently open.	
7007	End of file when reading screen layout file.	
7008	File system error when reading screen layout file.	
7009	Form name in screen file record record not equal to current form name.	
7010	Form contains no fields.	
7011	Cannot find screen record for form.	
7012	Write of screen record to screen layout file failed.	
7013	WARNING: if ADD command, form created with no screen	
7014	WARNING: if READD, screen not updated (form renamed?).	
7015	Correct syntax is: "FKLABELS formname [\keystring1...\keystringn]".	
7016	Screenfile contains an invalid character.	
7017	Missing datatype parameter.	

- 7018 Missing field type parameter.
- 7019 Missing field enhancement parameter.
- 7020 SELECTTERM update parameters must be separated by commas.
- 7021 Missing new FORMSPEC language ID.
- 7022 FIELD update parameters must be separated by commas.
- 7023 Renumbered !.
- 7024 Renumber was not necessary for !.
- 7025 Form is family child; renumber family by specifying parent form !.
- 7026 Internal error: unable to fetch form head.
- 7027 There are no forms in the current file.
- 7028 Creating new forms file.
- 7029 Recovering file -- PLEASE WAIT
- 7030 Compiling form !
- 7031 Number of errors = !!!!
Number of warnings = !!!!
- 7032 There are no compiled forms in the current file.
- 7033 No update values specified.
Command ignored.
- 7034 ***** End of compiled forms listing. *****
- 7035 Missing color parameter.
- 7036 Missing 264x parameter.
- 7037 Missing 307x parameter.
- 7038 New function key labels must be preceded by \.

VPLUS/V Error Messages

7039 End of command file.

7040 Forms file is being converted to new format.

7041 Missing ARB name.

7042 Sorry. That command has not been implemented.

7043 Missing ARB field name.

7044 Correct syntax is:

7045 ADDARBFIELD ARBname fieldname
[BEFORE/AFTER fieldname]

7046 Missing field name in
"BEFORE/AFTER fieldname".

7047 MODARBFIELD ARBname fieldname { length, type }

7048 MOVEARBFIELD ARBname field [/ field] [BEFORE/AFTER field]

7049 SCREENTOARB [char, date, dig, num, imp]

7050 ARBTOSCREEN [char, int, real, pack, zone]

7051 RENAMEARBFIELD ARBname oldfieldname newfieldname

Native Language Support Errors

Error	Message	Cause/Action
8000	This edit statement is not valid for international forms files.	
8001	Edit contains data inconsistent with language of this forms file.	
8002	Edit contains language-dependent data.	
8003	Edit contains data not compatible with the language of this forms file.	
8004	Invalid initial value for international forms file.	
9001	Native language support software not installed.	

- 9002 Language specified is not configured on this system.
- 9003 Character set specified is not configured on this system.
- 9004 Internal error: National table is not present.
- 9005 Internal error: Bad NLT extra data segment.
- 9006 Internal error: Bad LDST extra data segment.
- 9007 Error calling native language support intrinsic.
- 9008 Error calling native language support intrinsic.
- 9009 Error calling native language support intrinsic.
- 9010 Internal error: NLINFO item out of range.
- 9011 WARNING: Language not configured, change or hit ENTER to proceed.
- 9012 This edit requires a configured language.
- 9014 Attempted setting a language-dependent forms file to another language.
- 9015 NATIVE-3000 is currently the only selection available.
- 9016 Invalid date value for international forms file.
- 9017 Internal Error: Error updating language ID - try again.
- 9070 Internal Error: writing terminal selection source record failed.
- 9500 Language of forms file is not configured on this system.

VPLUS/V Error Messages

9998 Language ID must be 0 to 999
or -1 for international forms
file.

9999 Call to native language support
intrinsic 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.	
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

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. |



VPLUS/V Error Messages

- | | | |
|-----|---|--|
| 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 have 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 specified 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.
163	Output record size must be between 1 and 8192.	The output record size exceeds 8192 characters. Make the output record smaller.

Validation Errors

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.

VPLUS/V Error Messages

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 identifier on another menu.
204	Internal error: Data type is bad.	See error 206.
205	Output record length is greater than 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. Set'next failed.	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.

VPLUS/V Error Messages

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 unreferenced 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 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 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 form 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. | |

Menu Processing Errors

Error	Message	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.

VPLUS/V Error Messages

502	Reformat identifier ! not found.	A reformat id entered on the MAIN menu is not in the reformat file. Check the INPUT FORMS menu 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 form 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 in the lefthand box.
510	Listing is not done on a 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

Error	Message	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", "C", or " ".	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 checkdigit, 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.

VPLUS/V Error Messages

- | | | |
|-----|--|--|
| 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 and IMP 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 Init 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 err 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.

VPLUS/V Error Messages

- 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 !!!!) If trying to change the forms file name in REFSPEC make sure that the original forms file does not exist by temporarily renaming it.
- 714 Internal error writing the window. (COM'STATUS !!!!)
- 716 Internal error writing the error cursor pos'n. (COM'STATUS !!!!)
- 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
initializing output
record menu.
(COMSTATUS !!!!)
- 753 Internal error: VSETERROR
failed. (COMSTATUS !!!!)
- 754 Internal error: FGETINFO
failed.
- 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 The file name specified on the
file. REFORMAT file menu does not
identify an existing reformat
file.
- 757 Internal error: Could not Intrinsic FGETINFO failed.
get information from
reformat file.
- 758 Internal error: Could not Intrinsic FGETINFO failed.
get information from
forms file.

Menu Controller Errors

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 not meaning on the current menu.
805	Error getting next form !. (COM'STATUS !!!)	
806	Internal error in VSHOWFORM. (COM'STATUS !!!!)	

VPLUS/V Error Messages

- 807 Internal error in
VREADFIELDS.
(COM'STATUS !!!!)
- 808 Internal error in VGETBUF.
(COM'STATUS !!!!)
- 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 file 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 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.

Init and Compile Errors

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 !!!!)	

VPLUS/V Error Messages

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 Error

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 !!!!!) | |
-

VPLUS INTRINSIC MESSAGES

VOPENTERM and VCLOSETERM Messages

Error	Message	Cause/Action
001	Internal error: Terminal file initialization failed.	FCONTROL or FSETMODE intrinsic failed.
002	Your terminal is not supported by VPLUS.	VPLUS works on the HP terminals listed in Appendix G.
003	VPLUS does not run on a series I.	VPLUS does not run on an HP Series I computer system.
004	The terminal status request failed.	VOPENTERM's request for terminal status failed. Check that strapping is correct. May also be data comm problem.
005	The terminal allocation failed.	Message used only with the 2640A terminal, which is not supported by VPLUS. Reserved words in the COMAREA are not initialized to zeroes.
006	The terminal needs the H strap in (closed).	Refer to appendix G for instructions on how to strap your terminal for VPLUS.
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	The terminal close failed. (FSERR !)	FCLOSE intrinsic failed.
011	Sorry, this device is not supported by VPLUS.	Refer to Appendix G for a list of supported terminals.

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 (see App. G).
013	The HP2644A needs straps A, C, H in (closed); D, G out.	You must strap this terminal (and the 2640B) yourself; other terminals are strapped automatically (see App. 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 VPLUS.
015	Terminal type 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.
017	Comarea too short for run-time HP3075/6 terminals.	The Comarea must be 85 words for HP 3075/6 terminals. Set Comarealen to 85.
018	Non-page/block mode terminal types not supported.	Redefine terminal type. (Note: VPLUS requires a terminal which supports page/block mode.)
019	Internal error: FDEVICECONTROL should not return ccg.	See System Manager.
021	Invalid terminal type.	TERMTYPE used is a printer terminal type.

General Intrinsic Error Messages

Error	Message	Cause/Action
020	Error detected by host language.	Check the Comarea item FILERRNUM for the host language error number.
021	Invalid terminal type.	See 021 above.

VOPENTERM Error Messages (continued)

Error	Message	Cause/Action
023	Environment control file access failed	

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
042	Formsfile name not passed to VOPENFORMF.	The name of the forms file to be opened must be included in the call to VOPENFORMF.
043	Comarea length may not be less than 60.	The Comarea length as specified by the item Comarealen must be 60 words for HP terminals and 85 words for data capture devices.
044	Comarea length is too large.	The Comarea length as specified by the item comarealeng must not be longer than the longest length documented.
045	TERMINAL or FORMSFILE has not yet been opened.	
046	Unrecognized Comarea language code passed.	
050	Forms File open failed. (FSERR !)	FOPEN intrinsic failed. Check forms file name.
051	The file is not a VPLUS 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 !)
- 060 The program supplied
COMAREA extension is too
small.
- 061 Internal error: Failure
to obtain required PASCAL
Heap area.
- Run FORMSPEC and compile the
forms file.
- If you are using BASIC, check
word 4 of COMAREA (USRBUFLN),
and increase the length of the
COMAREA extension.

- | | | |
|------|--|---|
| 062 | Internal error: Failure to return PASCAL Heap area. | |
| 063 | Failure to close Forms File. (FSERR !) | An error occurred while trying to close the forms file. Check that the forms file is open before VCLOSEFORMF is called. |
| 067 | Invalid form store size specified. | An number outside the range of -1 to 255 is specified in the FORM'STORE'SIZE variable, which is the 39th. word of the VPLUS communication area. |
| 068 | VPLUS internal error; there is not enough DL area space. | For languages other than BASIC, VPLUS uses the DL area as the COMAREA extension. See error 040. |
| 069 | VPLUS internal error; the stack is frozen. | See error 041. |
| 9001 | Native Language Support software not installed | Check with System Manager to install Native Language Support software. |
| 9002 | Language specified is not configured on this system. | Select another language or check with System Manager to configure the desired language. |
| 9500 | Language of forms file is not configured on this system. | See System Manager to configure the language or use forms files on a system with that language configured. |



VOPENBATCH and VCLOSEBATCH Messages

- | Error | Message | Cause/Action |
|-------|--|---|
| 070 | Forms 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 specified 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. |

VPLUS/V Error Messages

073	A different forms file created this batch.	You opened a forms file that is 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

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 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 USERBUFLN to this number. If the application is not written in BASIC, increase the MAXDATA parameter on the PREP or RUN command.
- 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.

VPLUS/V Error Messages

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
104	Formsfile not compiled for this run-time terminal.	The type of terminal is being used has not been selected for use with the forms file. Specify the terminal type in the FORMSPEC Terminal Selection menu, then recompile the forms file.
105	HP3075/6 terminal with numeric only display is not supported.	VPLUS does not run on the numeric display HP3075/6 terminals.
106	HP3077 terminal is not supported except for opening and closing.	VPLUS only opens and closes the HP3077 terminal; no other functions can be performed.

VSHOWFORM Messages

Error	Message	Cause/Action
130	Internal Error: Terminal write failed. (FSERR !)	FWRITE intrinsic failed.
131	Internal error: Color conversion failed.	

VREADFIELDS Messages

Error	Message	Cause/Action
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 a 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. If user timeouts were enabled in the user application, a time out has occurred.
161	Field too long for HP3075/6 terminals.	The maximum length of a field on the HP 3075/6 terminals is 200 characters.
162	Terminal Power Failure detected.	A power failure has been detected on an HP 3075/6 terminal.
163	Invalid error light	Valid error lights are letters of the alphabet or the @ character.
164	HP3075/6 Terminal printer is out of paper.	Load the printer with a roll of paper.
165	Device is not supported by VPRINTLOCAL.	The terminal must be an HP 3075A or HP 3076A.
166	HP3075/6 Terminal does not have a printer.	Your terminal is not equipped with a printer.

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

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 field can only contain digits.	The field is specified as type DIG, but it contains characters other than 0-9.
205	The field can only contain a number.	The field is specified as type NUM, but it contains characters other than 0-9, decimal point, comma, or plus or minus sign.
206	The field can only contain a number, with max ! decimal places.	Reduce the number of digits to the right of the decimal point to n.
207	The field can only contain a number (! decimal places implied).	The field is specified as type IMPn, but it contains characters other than 0-9, decimal point, comma, or plus or minus sign. Also, there could be more than n digits to the right of the decimal point.
208	The field 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 field 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 field 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.

212	Error during data conversion between family members.	A data conversion error has been detected by VINITFORM when converting between family members. Check for field type compatibility problems in the form family in the forms file.
220	Internal error: Invalid destination.	Code execution problem.
222	An edit test failed.	The default field edit 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 date 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 is 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.

VPLUS/V Error Messages

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.	
337	Check digit validation string contains only one character.	A field whose value is to be checked by a modulus calculation must contain at least two characters.
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 an add operation.	Value is outside 4-word REAL range. Range of 4-word REAL is from approximately 8.636×10^{-78} to 1.158×10^{77} .
346	Overflow in a subtract operation.	See error 345.
347	Overflow in a multiply operation.	See error 345.
348	Overflow in a divide operation.	See error 345.
349	Overflow in a percent 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 a 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.

VPLUS/V Error Messages

- | | | |
|-----|--|--|
| 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 a character string 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). |
| 507 | Function key labels option is disabled. | The COMAREA item LABEL'OPTION must be set to one before opening the forms file. |
| 508 | Error reading function key values from forms file (FSERR !!!!) | An error occurred while trying to read the function key label definitions from the forms file. |
| 509 | Invalid input parameter to function key intrinsics. | An invalid value has been passed as the FORM-OR-GLOB or KEY-NUM parameter to VGETKEYLABELS, VSETKEYLABEL, or VSETKEYLABELS intrinsics. |
| 510 | A field with the screen order supplied does not exist. | |

VGET/PUTBUFFER ARB Messages

Error	Message	Cause/Action
519	ARB flag is set but no ARB record is found.	Application-specified conversion but the forms file does not contain an ARB record. <ul style="list-style-type: none"> • Initialize ARB conversion flag to 0 if conversion is not desired. • Use FORMSPEC to add the necessary ARB record. • Check for possible corruption of the formsfile.
520	ARB conversion type specified is not supported.	Check for possible corruption of the formsfile.
521	ARB record length exceeded maximum limit.	Check for possible corruption of the formsfile.
522	IEEE conversion is not supported in this version.	Application-specified IEEE conversion which is not yet supported. Check the IEEE conversion flag in the Comarea.
523	ARB record does not contain the minimum amount of information required.	Check for possible corruption of the formsfile.

VGETLANG and VSETLANG Messages

Error	Message	Cause/Action
9002	Language specified is not configured on this system.	Select another language or check with System Manager to configure the desired language.



9014 Attempted setting a language dependent forms file to another language. VSETLANG can only be used with international forms files.

VCHANGEFIELD

Error	Message	Cause/Action
850	Invalid field type specification.	Field type incorrect; use one of the listed codes.
851	Invalid data type specification.	Data type incorrect; use one of the listed codes - n indicates number of decimal positions.
852	Invalid enhance type specification.	Enhancement code is incorrect; check and correct.
853	Internal error: invalid field type.	Internal.
854	Internal error: invalid data type.	Internal.
855	Internal error: invalid enhance type.	Internal.
856	Invalid VCHANGEFIELD specification type.	Value must be 1-6.

INFO Intrinsic Messages

Error	Message	Cause/Action
601	Invalid input parameters to INFO 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. | Make sure that the field name is correctly spelled; that the field name parameter is 16 bytes long and padded with ASCII blanks. Make sure the field number is the actual field number. |
| 607 | Invalid form # specified to FORMINFO intrinsic. | |

Forms Loading Errors

- | Error | Message | Cause/Action |
|-------|---|---|
| 700 | No such form in terminal form storage memory. | The specified form is not purged because it does not exist. |
| 701 | Terminal form storage memory is full. | The specified form is not loaded because the form storage memory is full. |
| 702 | Requested load of blank form name. | |

Reformat Messages

- | 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 a 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 ABNORMALLY.	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 information from batch file. (FSERR !!!!)	FGETINFO failed.
24	File specified was not a reformat file.	Check reformat file name in :FILE command.

Allocation Errors

Error	Message	Cause/Action
30	Internal error: Batch data space too small. Allocation failed.	
31	Internal error allocating batch information buffer.	

VPLUS/V Error Messages

- 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

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 !!!!!)	

Intrinsic 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 USERBUFLN to this number.
096	Access method: Internal error. Parent block not found.	
097	Access method: Internal error. Illegal entry number.	

VPLUS/V Error Messages

- 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. Data truncated.	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 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 truncated. | 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. |

VPLUS/V Error Messages

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 checkdigit is to be inserted is not large enough to contain the extra digit. The checkdigit is not added. |
| 320 | Illegal character in field. Checkdigit not generated. | In performing the checkdigit 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 are 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. |

VPLUS/V Error Messages

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 data 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

Number	Message	Cause/Action
500	*****	
501	*	*
502	*	REFORMAT HP32209! *
503	*	! *
504	PAGE !!!!	
505	!	

506

507 REFORMAT/3000 HP32209!

Testlist Errors

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.	

Native Language Support

Error	Message	Cause/Action
8000	This edit statement is not valid for international forms files.	
8001	Edit contains data inconsistent with language of this forms file.	
8002	Edit contains language dependent data.	
8003	Edit contains data not compatible with the language of this forms file.	
8004	Invalid initial value for international forms file.	
9001	Native Language Support Software not installed	Check with System Manager to install NLS software.
9002	Language specified is not configured on this system.	Select another language or check with System Manager to configure the desired language.
9003	Character set specified is not configured on this system.	

9004	Internal error: National table is not present.	Internal error from NLS.
9005	Internal error: Bad NLT extra data segment.	Internal error from NLS.
9006	Internal error: Bad LDST extra data segment.	Internal error from NLS.
9007	Error calling native language support intrinsic.	Internal error from NLS.
9008	Error calling native language support intrinsic.	Internal error from NLS.
9009	Error calling native language support intrinsic.	Internal error from NLS.
9010	Internal error: NLINFO item out of range.	Internal error from NLS.
9011	WARNING: Language not configured, change or hit ENTER to proceed.	Language specified is not configured on the system, forms file produced will only run on a system configured with that language.
9012	This edit requires a configured language.	
9013	Encountered invalid Asian (2-byte) character.	
9014	Attempted setting a language dependent forms file to another language.	VSETLANG can only be used with international forms files.
9015	NATIVE-3000 is currently the only selection available.	FORMSPEC language can only be 0 in this version.
9016	Invalid date value for international forms file.	
9017	Internal error: Error updating language ID - try again.	

VPLUS/V Error Messages

- 9070 Internal error: writing terminal selection source record failed.
- 9500 Language of forms file is not configured on this system.
- 9998 Language ID must be 0 to 999 or -1 for international forms file. Forms file language ID must be between -1 and 999.
- 9999 Call to native language support intrinsic failed.

VPLACECURSOR intrinsic

- 800 Cannot place cursor to display only fields. Field number must be for an input field.

Native Mode Access Error Messages

- 998 Negative argument passed to VPLUS intrinsic. See Appendix J.
- 999 Attempt to access Compatibility Mode VPLUS services failed. See Appendix J.
-

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 USASCII and EBCDIC meaning of the character code, the USASCII<->EBCDIC conversion code, and the Hollerith representation (punched card code) for the USASCII character.

The following examples describe several ways of using the table:

Example 1: Suppose you want to determine the USASCII code for the \$ character. Scan down the USASCII 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 USASCII 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 USASCII (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 USASCII, 044. As a check, find 044 (oct) in the character code column, look right to the USASCII graphic column and note that \$ converted to EBCDIC is 133 (oct), which equals 5B (hex).

USASCII Character Set in Collating Sequence

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	11-1-3-9	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

CHAR CODE			ASCII		EBCDIC		
Dec	Oct	Hex	Cntl/ Gph	to EBCDIC (Oct)	Hollerith	Cntl/ Gph	to ASCII (Oct)
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
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	9B		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

USASCII Character Set in Collating Sequence

CHAR CODE			ASCII		EBCDIC		
Dec	Oct	Hex	Cntl/ Gph	to EBCDIC (Oct)	Hollerith	Cntl/ Gph	to ASCII (Oct)
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		172
170	252	AA		122	12-11-2-9	z	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
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
189	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	11
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	J	35
205	315	CD		234	12-11-4-8		35
206	316	CE		235	12-11-5-8		354
207	317	CF		236	12-11-6-8	K	355

CHAR CODE			ASCII		EBCDIC		
Dec	Oct	Hex	Cntl/ Gph	to EBCDIC (Oct)	Hollerith	Cntl/ Gph	to ASCII (Oct)
208	320	D0		237	12-11-7-8	l	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	O	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	rl	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 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 VPLUS/V 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.

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	96438 <u>7</u>

MODULUS 11

Modulus 11 detects single digit errors, single transpositions, and double transpositions. Unlike other checkdigit 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.

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	$20+3+12+7+54+10+32+21+4=163$
Subtract remainder from 11	$11-9=2$
Check digit =	2
Self-checking number	516192872 <u>2</u>

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.

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	Δ	(Δ=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 checkdigit 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	34G12H <u>1</u>

APPLICATION HINTS

APPENDIX

E

This appendix provides hints and guidelines on the following topics

- Designing your Forms File
- Rules for the DL Area
- Coding the Touch Feature

DESIGNING YOUR FORMS FILE

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. On an HP 264X terminal, 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 area of the screen), you can define a field with a display only field type and any enhancements you desire. Using the INIT processing phase specification, text can be initialized to appear in the field. For example:

```
Num [1 ] Len [22 ] Name [TEXT ] Enh [HI ] FType [D] DType [CHAR]
Initial Value [ ]
*** Processing Specifications ***
INIT
SET TO "ENTER NAME HERE"
```

This method can also be used to alter "text" on the screen during execution. Define a display only field with the desired enhancements. You can either use the processing specifications phases (INIT, FIELD or FINISH) to alter the contents of the field or you can programmatically change the field contents using VPUTBUFFER or VPUTFIELD intrinsic calls.

NOTE

Processing specifications only execute if the application calls the "editing" intrinsics: VINITFORM, VFIELDEDITS, VFINISHFORM.

4. If you want to define blank lines at the bottom of a form, define a display only field of at least one character at the end of the blank lines. Delimit the field by `(ESCAPE)` [and `(ESCAPE)`], with the field enhancements of NONE. This forces the display of seemingly blank lines and does not affect the application programs. Otherwise FORMSPEC would delete trailing blank lines from a form.
5. If you use save fields to accumulate totals and you expect the user to browse and correct the fields used for summation, special care must be taken to ensure 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.

Application Hints

7. Sometimes in data entry applications, the user 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 a user 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 (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 named 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. There are three file equations for which VPLUS/V checks during execution. The use of these file equations provides the user with a way to override the defaults used by VPLUS/V. These file equations are also useful when attempting to debug an application with trace messages or when DEBUG is used. The VPLUS/V intrinsic VOPENTERM, used by FORMSPEC, ENTRY, and REFSPEC, searches for a file equation named A264X. VOPENTERM attempts to open the device referenced by A264X to display VPLUS/V forms and accept their associated input. For example:

```
:FILE A264X; DEV=28
:RUN ENTRY.PUB.SYS
```

causes logical device 28 to be opened by VOPENTERM, and subsequent screen displays by VSHOWFORM appear on LDEV 28. In this example, LDEV 28 must be a logged off terminal in an available state. Applications can make use of this feature by using the *termfile* parameter of VOPENTERM and a file equation. If the second parameter to VOPENTERM contained the value TERMINAL, the following file equation will cause logical device 28 to be opened for forms display:

```
:FILE TERMINAL;DEV=28
```

Trace and abort messages will be displayed to the command terminal -- the one which initiated the application. This would be the main reason to use this feature as abort messages are frequently impossible to read because they are displayed in the unprotected fields of a VPLUS/V application.

The second file equation which VPLUS/V searches for is FORMLIST. The VPRINTFORM intrinsic uses FORMLIST to list forms (as in ENTRY). FORMLIST defaults to the system printer, but the user can save the list file on disc using the following file equation, which creates a file named FORMLIST containing the form listing.

:FILE FORMLIST; DEV=DISC;SAVE

If your system has both a page printer and a line printer (default), the file equation:

:FILE FORMLIST;DEV=PP

sends the form listing to the page printer.

The third file equation which VPLUS/V uses deals with a summary listing of the forms file from batch mode FORMSPEC. Batch mode prints to a file named FORMOUT. A file equation for FORMOUT can use the same parameters as discussed for the file FORMLIST.

10. The LIST command and the batch mode FORMS command print a maximum of 60 lines per page. This maximum may be altered by setting the JCW

FORMSPECLINESPERPAGE

For example, the MPE command

:SETJCW FORMSPECLINESPERPAGE=45

sets the maximum number of lines per page to 45. This JCW is useful when more white space is desired at the bottom of the page, or when special paper is used.

FORMSPECLINESPERPAGE must be in the range 1 to 150. Any value outside this range will cause a LIST command to fail, and will cause batch mode FORMSPEC to halt with the error message

FORMSPECLINESPERPAGE must be between 1 and 150

No lines will be printed in either case, to ensure that FORMSPEC does not waste paper printing a listing or FORMS reference in an unwanted format.

RULES FOR THE DL AREA

VPLUS/V makes use of the DL area for buffers and internal tables. Therefore, if the DL area is put to any other use during the execution of a VPLUS/V application, such use must not conflict with that of VPLUS/V. To insure against conflict, observe the following rules:

1. The first use of the DL area by an application, including any procedures, routines, or intrinsics executed on behalf of the application, should be the call to VOPENFORMF that opens the forms file and allocates the DL area needed by VPLUS/V.
2. The call to VCLOSEFORMF that frees up the DL area used by VPLUS/V must not occur until the application has released all other DL space allocated subsequent to the VOPENFORMF call.

These restrictions need not be followed if the application is wholly coded in any combination of Pascal, HPFORTRAN 77, and HPBUSINESS BASIC and sets the *comarea* item *language* to 5, which causes VPLUS/V to use the heap procedures provided by Pascal instead of managing the DL area directly.

VPLUS/V applications coded in other languages need to take great care if they pass the *language* identifier of 5 as a means of getting around the restrictions stated above. Specifying an incorrect *language* identifier can cause other aspects of the VPLUS/V interface, such as addressing of byte parameters, to function unreliably. In such a case, the application must use the INTRINSIC mechanism (CALL INTRINSIC in COBOL) for the main and all interacting parts of the application. Check the Pascal and COBOL reference manuals for more information.

In all cases, check for any other use of the DL area by intrinsics such as the DSG intrinsics.

CODING THE TOUCH FEATURE

In order to use the touch feature effectively, a method is needed to provide feedback to the user when a field is touched. Changing the field enhancement is one of the best methods to indicate that a field has been touched. The enhancement of a field can be changed by using the `VCHANGEFIELD` intrinsic, or family forms. If neither of these methods is appropriate and the error count is not important, `VSETERROR` can be used instead. `VSETERROR` toggles the error flag when called successively; i.e. if `VSETERROR` is called the second time for a particular field, the error flag is cleared. Therefore, the error enhancement can be used with `VSHOWFORM` as feedback to the user to indicate that a field has been touched. This feature is only activated if the *showcontrol* bit 0 is set to 1.

When designing touch applications, the designer should keep the idea of compatibility in mind. Applications should be able to run on HP 262X/239X terminals, or on a touch terminal/workstation using keyboard input. For `VPLUS/V` applications, this objective can be achieved by designing one single form interface for all HP 262X/239X terminals.

To design a touch application with compatibility in mind, the interface should be built so that the user has the option of responding via the keyboard or the touch screen. This consideration is important in two respects. First, it provides an alternative path for the user when an application is run on touch terminals. Second, an application can be run on non-touch terminals as well as touch terminals.

For example, if an application is designed to display a form with many items for the user to select from, the programmer may wish to include one input field for the users to type in their choice. This will allow for one interface for both touch and non-touch terminals.

Example

```

IF COM-LASTKEY < 0 THEN
    CALL "VSHOWFORM" USING COMAREA
    CALL "VREADFIELDS" USING COMAREA
    IF COM-LASTKEY IS NOT EQUAL TO -999 THEN
        NEXT-FIELD = -(COM-LASTKEY)
        CALL "VSETERROR" USING COMAREA, NEXT-FIELD, MSG, MSGLEN
        CALL "VSETERROR" USING COMAREA, PREVIOUS-FIELD, MSG, MSGLEN
        PREVIOUS-FIELD = NEXTFIELD.

```

The first `VSETERROR` toggles the error flag of the new field touched, and the second `VSETERROR` toggles the error flag of the previous field touched. When `VSHOWFORM` is called, `NEXT-FIELD` will be highlighted with the error enhancement, while `PREVIOUS-FIELD` reverts to the normal enhancement. The `VPLACECURSOR` intrinsic can be used to position the cursor if the default cursor position set by `VSETERROR` needs to be overridden.



STATE CODES

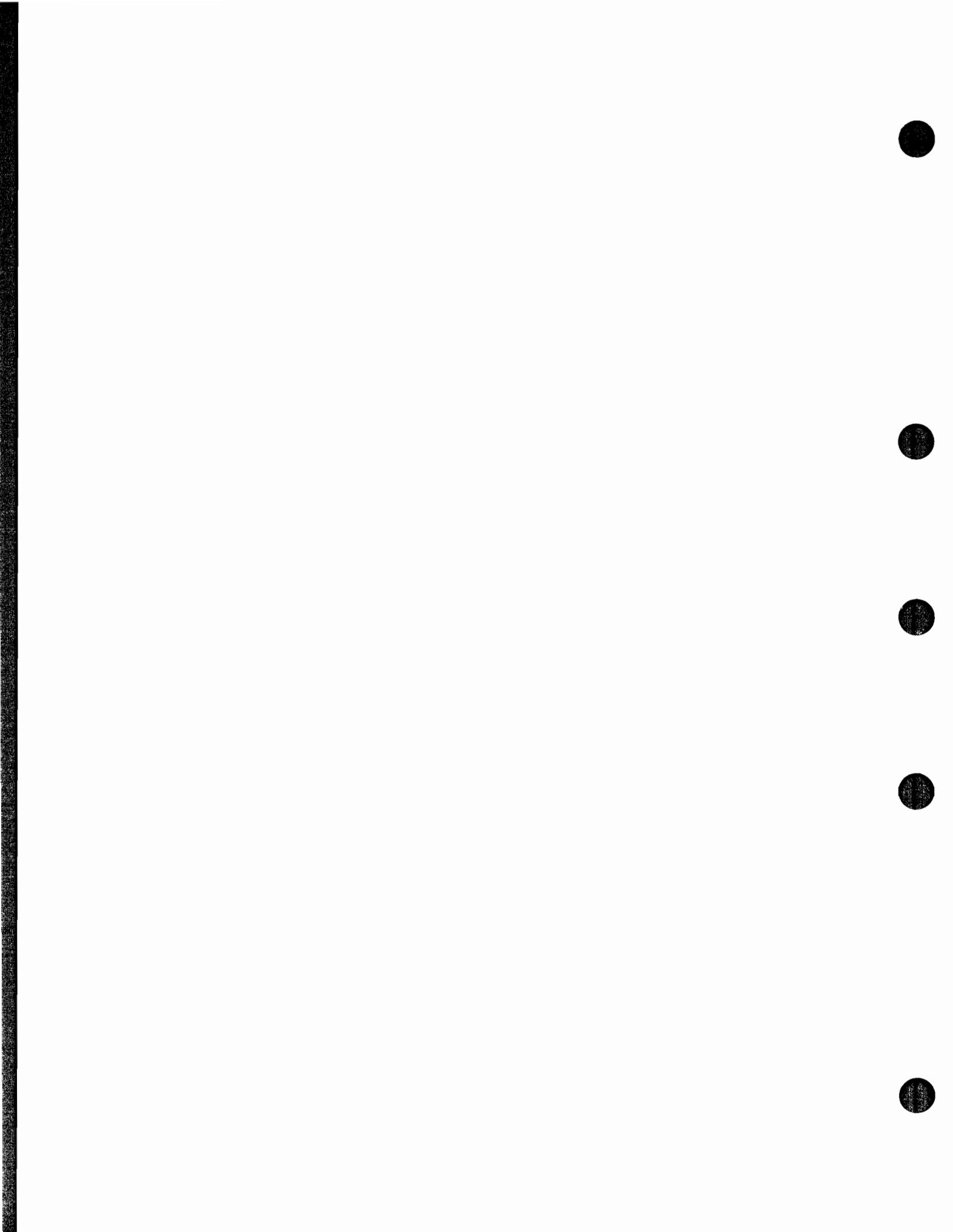
APPENDIX

F

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 lowercase letter. 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 uppercase 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

APPENDIX

G

This appendix contains information about terminals that affects the operation of VPLUS/V. An overview of the supported terminals and their main features is followed by information on terminals, grouped according to terminal family and model within the family. This appendix also contains information about:

- The VPLUS/V *comarea* items that relate to terminals
- Configuring terminal buffers
- Recovering from unexpected program interruption
- The user environment control file
- Advanced terminal I/O procedures; VTURNON/VTURNOFF

SUPPORTED TERMINALS AND FEATURES

VPLUS/V can be used with the following terminals:

HP 150	HP 2392A	HP 2622A	HP 2640B	HP 3075A
	HP 2393A	HP 2623A	HP 2641A	HP 3076A
HP 2382A	HP 2394A	HP 2624A	HP 2642A	HP 3081A
	HP 2397A	HP 2624B	HP 2644A	
		HP 2625A	HP 2645A	
		HP 2626A	HP 2647F	
		HP 2627A	HP 2648A	
		HP 2628A		

Applicable special features of these terminals are shown in Table G-1 and are described below.

Termtypes

As a user of an HP 3000 system, you may choose to specify a *termtyp*e when you log onto MPE; for example,

```
:HELLO username.acctname;TERM=nn
```

However, VPLUS/V only supports a subset of the available *termtyp*es. They are *termtyp*es 10, 12, 14, and 24.

A *termtyp*e that is not supported by VPLUS/V may be:

- Rejected as an error if VPLUS/V knows that it cannot continue, or
- Mapped to a supported *termtyp*e if this will allow VPLUS/V to continue.

When VPLUS/V maps an unsupported *termtyp*e to one that is supported, the terminal driver is reconfigured to the original *termtyp*e when VPLUS/V closes the terminal. Finally, VPLUS/V defers to the Workstation Configurator when editing *termtyp*e if that facility is available.

Modified Data Tag

You do not have to take any action to utilize this feature. Only the fields which have been modified are transmitted to the computer. This feature is supported on the following terminals:

HP 150	HP 2393A	HP 2624A/B
	HP 2394A	HP 2625A
	HP 2397A	HP 2628A

Extended Local Edits

This feature provides for the editing of information by the terminal as it is typed into the form. You specify the editing to be done by using the command `LOCALEDITS` in the field processing specification section. The HP 2624B and HP 2394A terminals support this feature.

Relabeling Function Keys

This feature allows you to specify during forms creation or modification time function key labels that will be used during data entry. Function key labeling is supported by `VPLUS/V` on all terminals except the data capture devices. On HP 264X terminals, `VPLUS/V` uses the last two screen lines to emulate function key labels, which are available as a terminal feature on all other HP terminals.

Security Display Enhancement

This feature allows you to inhibit the display of the data in certain fields. When this feature is used, the characters entered at the terminal are displayed as blanks on the screen. The security display enhancement is supported on the following terminals:

HP 150	HP 2393A	HP 2624B
	HP 2394A	HP 2625A
		HP 2626A
		HP 2628A

Table G-1. Terminals Supported by VPLUS/V

ID	✓	MODEL	GRAPH	COLOR	TOUCH	LFS	MDT	LOCAL EDITS	SEC ENH	X.25 PAD
0	✓	✓✓								
1		2640B								
2		2644A								
3		2645A								
4		2641A								
5		2648A	□							
6		2647A	□							
7	✓	2621A								
8		2626A				□			□	
9		2624A					□	□	□	
10		2642A								
11		2622A								*
12		2623A	□							*
13		2624B				□	□	□	□	*
14		2382A								
15		3075A								
15		3081A								
16		3076A								
17	+	3077A								
18	✓	2621B								
31		2647F	□							
51		2703A	□							
52		2627A	□	□						□
55		2622E								
56		2392A								□
57		2394A				□	□	□	□	□
61		2625A					□	□	□	□
62		2628A					□	□	□	□
63		2625A	□				□	□	□	□
64		2628A	□				□	□	□	□
65		2393A	□				□	□	□	□
66		2397A	□	□			□	□	□	□
67		2393A	□		□		□	□	□	□
68		2397A	□	□	□		□	□	□	□
70		HP150	□		□		□	□	□	□

✓ Not a supported terminal

✓✓ Unknown

+ Can only be opened and closed by VPLUS/V

* Check ROMS for auto keyboard lock

Local Form Storage Capabilities

This feature allows forms to be stored locally in the terminal's memory. Local form storage can reduce the data communication overhead associated with frequently displayed forms. This feature is supported on the HP 2394A, HP 2624B and HP 2626A terminals, and is fully described in Section 6.

X.25 Capability

This feature allows X.25 block mode to be used via a PAD interface with the following terminals:

HP 150	HP 2392A	HP 2622A*	HP 2625A
	HP 2393A	HP 2623A*	HP 2627A
	HP 2394A	HP 2624B*	HP 2628A
	HP 2397A		

* Check ROMS for auto keyboard lock.

Color Enhancement

This feature allows color to be used for field, error, and window enhancements with the HP 2397A and HP 2627A terminals.

Data Capture Device Special Features

The HP 3075A, HP 3076A and HP 3081A terminals are data capture devices. Special features of the data capture devices that VPLUS/V supports are as follows:

Terminal Input Options

- Type V 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 are not supported.

THE HP 264X TERMINALS

The HP 264X terminal family is the terminal default--you do not need to access FORMSPEC's Terminal/Language Selection Menu *unless* you need to change the default. However, if you want function key labeling for HP 264X terminals, you must access the Terminal/Language Selection Menu and specify Y in the HP 264X Family field. Also, when designing forms to run on these terminals, the following constraints must be considered:

- Do not use column 79 if the form is to be part of a form family.
- Do not use column 80 if the form may have another form appended to it or is part of a form family.
- A multiline field must go all the way through column 80 in order to continue on the next line.
- The security display enhancement is not supported on HP 264X terminals.

All HP 264X terminals, except the HP 2640A and HP 2644A terminals described below, run VPLUS/V with the default terminal strapping. However, if the defaults have been altered, you must *close* straps G and H. These terminals are automatically configured by VPLUS/V so you do not have to press the block mode key.

HP 2640B Terminal

This terminal must be placed in block mode. When the terminal is opened by FORMSPEC, ENTRY, REFSPEC, or an application, a message is displayed asking you to press the block mode key. Also, when the terminal is closed, a message reminds you to unlatch the block mode key. The HP 2640B must be strapped as follows to use VPLUS/V:

Switch	Setting
A	closed
B	-----
C	closed
D	open
E	open
F	open
G	open
H	closed

Strapping the terminal in this manner allows BLOCK MODE/PAGE operation and allows you to use the function keys without also using **CONTROL**. The backtab feature, which allows you to use **CONTROL** **TAB** to position the cursor to the beginning of the previous field, is not available on this terminal.

HP 2644A Terminal

This terminal must be placed in block mode. When the terminal is opened by FORMSPEC, ENTRY, REFSPEC, or an application, a message is displayed asking you to press the block mode key. Also, when the terminal is closed, a message reminds you to unlatch the block mode key. The HP 2644A must be strapped as follows to use VPLUS/V:

Switch	Setting
A	closed
B	-----
C	closed
D	open
E	-----
F	-----
G	open
H	closed

Strapping the terminal in this manner allows BLOCK MODE/PAGE operation.

THE HP 262X TERMINALS

VPLUS/V runs on the HP 262X family without any special action by you (without specifying the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu) *except* in the special cases noted in the following paragraphs.

To use X.25 block mode via a PAD interface, available with all HP 262X terminals except the HP 2624A, the transmit and receive pacing must be set correctly, the appropriate ROMS must be used, and the terminal must be strapped correctly. You must check that the correct ROMS are used with the HP 2622A, HP 2623A, HP 2624B and the HP 2627B terminals. The HP 2625A and HP 2628A terminals always have the correct ROMS. Also, the correct terminal type must be used when logging on to the terminal. For more detailed information, refer to the *DSN/X.25 HP 3000 Reference Manual*.

HP 2624A Terminal

To utilize the local edits and security display enhancement features of the HP 2624A, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

At run time, when using a form, you could receive the message:

LINE IS FULL - RETURN TO CLEAR.



This message is issued by the terminal, not by VPLUS/V. It means that there are too many characters on the line, and you must redesign and recompile the form.

HP 2624B Terminal

To utilize the local edits and security display enhancement features of the HP 2624B, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

This terminal also has the capability of storing as many as 255 forms locally in terminal memory depending on the size of the forms and the memory available. Use of this feature is activated and controlled by the VPLUS/V intrinsics and *comarea* items as discussed in Section 6.

To use VPLUS/V and local form storage on this terminal, you must alter the default datacomm configuration menu. Change the strip NULL and DEL field from YES to NO.

At run time, when using a form, you could receive the message:

LINE IS FULL - RETURN TO CLEAR.

This message is issued by the terminal, not by VPLUS/V. It means that there are too many characters on the line, and you must redesign and recompile the form.

HP 2625A, HP 2628A Terminals

To utilize the security display enhancement of these terminals, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

HP 2626A Terminal

To utilize the security display enhancement feature of the HP 2626A, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

The HP 2626A terminal can store as many as four forms locally depending on the size of the forms. VPLUS/V utilizes the workspace/window feature of this terminal and configures terminal memory into from one to four workspaces. Local form storage is activated and controlled by the VPLUS/V intrinsics and *comarea* items as discussed in Section 6. Only port 1 of the HP 2626A terminal can be utilized with local form storage. If you wish to run an application on port 2 of the terminal without local form storage, the *formstoresize* item of the *comarea* should be modified as described in Section 6 in the *comarea* item description.

HP 2627A Terminal

To utilize the color display enhancement feature of the HP 2627A, you must specify both HP262X, HP239X Families and HP2627A, HP2397A on FORMSPEC's Terminal/Language Selection Menu.

THE HP 239X AND HP 150 TERMINALS

VPLUS/V runs on the HP 239X and the HP 150 terminals without any special action by you (without specifying the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu) *except* in the special cases noted in the following paragraphs.

To use X.25 block mode via a PAD interface, available with all HP 239X and HP 150 terminals, the transmit and receive pacing must be set correctly and the terminal must be strapped correctly. (The HP 239X and HP 150 terminals always have the correct ROMS.) Also, the correct terminal type must be used when logging on to the terminal. For more detailed information, refer to the *DSN/X.25 HP 3000 Reference Manual*.

HP 2392A Terminal

To utilize the security display enhancement feature of the HP 2392A, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

HP 2393A Terminal

To utilize the security display enhancement feature of the HP 2393A, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

HP 2394A Terminal

To utilize the local edits and security display enhancement features of the HP 2394A, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

This terminal also has the capability of storing as many as 255 forms locally in terminal memory depending on the size of the forms and the memory available. Use of this feature is activated and controlled by the VPLUS/V intrinsics and *comarea* items as discussed in Section 6.

HP 2397A Terminal

To utilize the color display enhancement feature of the HP 2397A, you must specify both HP262X, HP239X Families and HP2627, HP2397A on FORMSPEC's Terminal/Language Selection Menu.

HP 150 Terminal

To utilize the security display enhancement feature of the HP 150, you must specify the HP262X, HP239X Families on FORMSPEC's Terminal/Language Selection Menu.

THE DATA CAPTURE DEVICES

To use the HP 3075A, HP 3076A or HP 3081A devices with VPLUS/V, you must specify the HP307X Family on FORMSPEC's Terminal/Language Selection Menu. To take advantage of the HP 3081A features, you must specify an eight (8) in the selection box for the HP307X Family on the Terminal/Language Selection Menu. VPLUS/V only opens or closes the HP 3077A device; no other functions are available.

TERMINAL COMMUNICATION AREA

Positions 49 through 58 (relative to 1) of the VPLUS/V communication area (*comarea*) are used for terminal-related information, which includes the terminal file number, terminal type, terminal allocation and error logging, and terminal options. It is not necessary for you to access this area as VPLUS/V manages it automatically. However, if you need 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 *comarea* items described in the following paragraphs perform these functions.

Parameters

Position 49 - *filen* MPE file number used to identify the terminal.

Position 56 - *termoptions* Terminal control options:

bits: 0 - 8	Reserved for system use.
9 - 10	01 = Enable ENTER or function key key timeout in VREADFIELDS. 00 = Disable ENTER or function key timeout in VREADFIELDS (default).
13 - 14	01 = Enable AUTOREAD in VREADFIELDS. 00 = Disable AUTOREAD in VREADFIELDS (default)
15	0 = Display mode message (default) 1 = Suppress mode message.

Normally, there is no time limit for VREADFIELDS, whether the intrinsic is terminated by **ENTER** or a function key. Timeouts can be enabled in applications by setting bit 10 of the *termoptions* item of the *comarea* to 1, and specifying a timeout interval in the *usertime* item of the *comarea*. When a timeout is enabled, if **ENTER** or a function key is not pressed within the number of seconds specified in *comarea* position 58, the *cstatus* of 160 will be returned to the *comarea*. At this point, control will then be passed to the statement following the VREADFIELDS call in the application.

For example, in COBOL:

```
INITIALIZE-COMAREA
  MOVE ZERO TO COM-OPTIONS.
  ADD 32 TO COM-OPTIONS.
  MOVE 60 TO COM-USERTIME.
```

if included in the initialization section of the application will indicate that all calls to VREADFIELDS should timeout after 60 seconds. Note that an ADD instruction was used to turn on the timeout feature. All changes to the *termoptions* item following VOPENTERM should be careful to set only single bits, as VPLUS/V uses bits 0-7. A MOVE instruction will destroy the configuration of these bits.

When VREADFIELDS is terminated by a function key, VPLUS/V does not transmit any data entered on the screen to the user buffer. If the programmer wishes for data to be transmitted to the data buffer when a function key is pressed, the AUTOREAD feature should be enabled. The following COBOL example shows how to enable the autoread feature, and transfer data to the buffer when the function key 4 is pressed:

```
CALL "VREADFIELDS" USING COMAREA.
IF COM-LASTKEY = 4 THEN
  ADD 2 TO COM-OPTIONS
  CALL "VREADFIELDS" USING COMAREA
  SUBTRACT 2 FROM COM-OPTIONS.
```

Note that two VREADFIELDS calls are necessary to complete the process. The first VREADFIELDS detects that a function key was pressed, the second call actually transmits the data to the data buffer. The second call to VREADFIELDS does not wait for the user to press enter, but rather sends an "ESC d" to the terminal to simulate an automatic enter. As noted above in the description of user timeouts, the COBOL SUBTRACT and ADD statements should always be used when setting bits in the *termoptions* item of the *comarea*.

The mode message is displayed on terminals not automatically configured by VPLUS/V. The mode message for VOPENTERM is BLOCK MODE/PAGE IS SET; for VCLOSETERM it is REMEMBER TO UNLATCH THE BLOCK MODE KEY.

Position 58 - *usertime*

If enabled, the value in this word is used as the number of seconds to wait for the ENTER or function key to be pressed. The maximum timeout value is 32,767 seconds (9.1 hours).

Position 59 - *identifier*

VPLUS/V identifier for the HP terminal model being used, as shown earlier in Table G-1 in the column labeled ID.

TERMINAL BUFFER CONFIGURATION

When using FORMSPEC, ENTRY, REFSPEC, or the intrinsics VREADFIELDS or VSHOWFORM, there must be sufficient terminal buffers available for all concurrently executing terminal I/O operations. The number of terminal buffers must be at least 150 and it is strongly recommended that the number of terminal buffers be set to the maximum allowed on your system, shared by all processes. (See the configuration dialogue in the *MPE V Systems Operation and Resource Management Reference Manual* (32033-90005).

RECOVERING FROM UNEXPECTED PROGRAM INTERRUPTION

In case of unexpected interruption due to hitting **BREAK** or a terminal power failure, control returns to MPE. To recover from such a situation, you take the following steps:

1. Perform a hard reset.
 - On an HP 264X terminal, you press the RESET TERMINAL key twice, and then press **RETURN** to display the colon prompt.
 - On HP 262X, HP 239X, and HP 150 terminals, you press **SHIFT CONTROL**RESET and then press **RETURN** to display the colon prompt.
2. If echo is turned off, press **ESCAPE**, then press **:** to restore echo.
3. Type RESUME and you should see the message READ PENDING.
4. Press **REFRESH** 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, modifying, or entering data at the time of interruption.

If you were in the unformatted portion of a FORMSPEC Field Menu, the procedure is slightly different, as follows:

1. Perform a hard reset, as described above.
2. Type **CONTROL**R ABORT**CONTROL**R (Letters must be upper case as shown).
3. If echo is turned off, press **ESCAPE**, then press **:** to restore echo.
4. Issue the RUN command for FORMSPEC to continue operation.

USER ENVIRONMENT CONTROL FILE

The user environment control file is called `VENVCNTL.PUB.SYS`, and is designed to let the programmer assert limited control over terminal activities. It is opened and read on the `VOPENTERM` call, and it controls these two items in the user environment.

1. Abbreviating a terminal query. When `VOPENTERM` is called, it usually goes through a complicated process to identify the terminal type. The `VENVCNTL` file shortens the identification process. However, it should only be used if it is known that a query will elicit a valid terminal ID. To shorten the terminal query, enter 1 in the first column of the `VENVCNTL` file.
2. Enabling `(AIDS)/(MODES)/(USERKEYS)` without a program. Instead of setting the `SHOWCONTROL` word of the user's *comarea* in the application, enter 1 in the second column of the `VENVCNTL` file. This option is very useful if you do not have the source code for the software but you want to enable `(AIDS)/(MODES)/(USERKEYS)`.

To implement a user environment control file, build this file:

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

Use any editor to enter 1 in column 1 and/or 2. When `VOPENTERM` executes, it looks for the `VENVCNTL` file. If it does not find it, `VPLUS/V` carries out its usual operations. If the file exists and columns 1 and/or 2 are set, `VPLUS/V` performs either or both of the prescribed actions.

The `VENVCNTL` file need not necessarily be in `PUB.SYS`. However, if you have the file residing in another group and account, you must set up the file equation:

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

ADVANCED TERMINAL I/O PROCEDURES

In general, the VPLUS/V terminal input/output procedures such as VOPENTERM, VCLOSETERM, VREADFIELDS and VSHOWFORM are sufficient for most applications that communicate with a user via a terminal interface.

In building complex user interfaces, however, application designers have a limited number of tools for implementation. Hence, they resort to using FREAD, FWRITE, and input/output verbs of their programming language to display and retrieve information from the terminal when running VPLUS/V. The situation becomes more complicated when the application controls the terminal settings as well as the driver settings, and is likely to result in an application that runs in a restricted environment; for example, on an ATP and on one kind of terminal only.

VPLUS/V provides two additional procedures that help application designers to write applications with complex user interfaces that are more portable and easier to maintain. They are VTURNON and VTURNOFF.

VTURNON/VTURNOFF

VOPENTERM configures the driver and the terminal for block mode access. VCLOSETERM configures the terminal and driver for character mode access. These procedures initialize the terminal in many ways, including clearing the screen image. VTURNON and VTURNOFF reconfigure the terminal and driver but leave the terminal screen image intact. VTURNOFF allows the application to switch momentarily to character mode from block mode without disturbing the screen. A call to VTURNON will reconfigure the terminal back to block mode. An example of such use is an application which performs graphics input/output after the terminal has been opened by VOPENTERM and a form has been displayed. VTURNOFF can then be called to prepare for graphics operations. A subsequent VTURNON call will switch the application back to VPLUS/V block mode. The use of VOPENTERM, VCLOSETERM, VTURNON and VTURNOFF provides for a systematic approach to controlling the terminal and driver settings.

The recommended practice for programming VPLUS/V applications is to use only VPLUS/V procedures to perform terminal input/output operations when in VPLUS/V. If the application must perform terminal input/output operations using other input/output methods, either VCLOSETERM or VTURNOFF must be used to switch out of VPLUS/V before doing so.

This simple programming guideline assures that the application will be compatible with the different HP3000 drivers and terminals, and improves its maintainability.

VTURNOFF

Turns off VPLUS/V block mode and enables character mode access without disturbing the terminal screen.

Syntax

```
VTURNOFF {comarea}
```

Parameters

comarea Must be *comarea* named when the file was opened by VOPENTERM. If not already set, the following *comarea* items must be set before calling VTURNOFF.

cstatus Set to zero.

comarealen Set to total number of 2-byte words of *comarea*.

VTURNOFF may set the following *comarea* items:

cstatus Set to nonzero value if call is unsuccessful.

fileerrnum Set to file error code if there is an MPE file error.

Discussion

VTURNOFF is used for momentarily switching from VPLUS/V block mode to character mode. This procedure is designed for use after a terminal has been previously opened by VOPENTERM or after a VTURNON.

VTURNOFF reconfigures the terminal and driver for character mode operations without disturbing the screen image on the terminal. The following operations normally performed in VCLOSETERM are not performed in VTURNOFF:

- Clear local form storage
- Enable the USER/SYSTEM keys
- Disable touch reporting, delete touch fields
- Clear screen
- Unlock keyboard
- Close terminal file

VTURNOFF

NOTE

VTURNOFF does not close the terminal file. To close the file and completely reset the driver and the terminal, VCLOSETERM must be used.

Example

The following examples illustrate a call to VTURNOFF:

```
COBOL  
CALL "VTURNOFF" USING COMAREA.
```

```
BASIC  
200 CALL VTURNOFF(C(*))
```

```
FORTRAN  
CALL VTURNOFF(COMAREA)
```

```
SPL  
VTURNOFF(COMAREA);
```

```
PASCAL  
VTURNOFF(COMAREA);
```

Turns on VPLUS/V block mode without disturbing the terminal screen.

Syntax

```
VTURNON {comarea,TERMFILE}
```

Parameters

comarea The *comarea* name must be unique for each open forms file. The *comarea* must be the same one used in VOPENTERM. The following *comarea* items must be set before the call if not already set:

cstatus Set to zero.

language Set to code that identifies the programming language of the calling program.

comarealen Set to total number of 2-byte words in *comarea*.

VTURNON may set the following *comarea* fields:

cstatus Set to nonzero value if call is unsuccessful.

fileerrnum Set to file error code if there is an MPE file error.

filen Set to MPE file number of terminal.

identifier Set to appropriate VPLUS/V terminal ID.

lab'info Set to appropriate number and length of labels.

termfile Must be the same terminal file name used in VOPENTERM.

Discussion

VTURNON is normally used in an application when the terminal has already been opened by VOPENTERM, and VTURNOFF was called to switch out of VPLUS/V block mode. VTURNON switches the application back to VPLUS/V block mode without disturbing the image on the terminal screen.

VTURNON

VTURNON reconfigures the terminal and the driver without performing the following operations, which are usually carried out by VOPENTERM:

- Initialize local form storage
- Clear screen
- Enable the `USERKEYS` function keys
- Disable or enable the USER/SYSTEM key as specified in the SHOWCONTROL word

Unlike VOPENTERM, VTURNON will not ask you to press the BLOCK MODE key if you are using an HP2640B or HP2644 terminal and the terminal is not in block mode.

Example

The following examples illustrate a call to VTURNON:

```
COBOL
CALL "VTURNON" USING COMAREA, T1.
```

```
BASIC
90 T1$=" "
100 CALL VTURNON(C(*),T1$)
```

```
FORTRAN
T1=" "
VTURNON(COMAREA,T1);
```

```
SPL
MOVE T1:=" ";
VTURNON(comarea,T1);
```

```
PASCAL
T1:= ' ';
VTURNON(COMAREA,T1);
```

VERSION CONTROL

APPENDIX

H

This appendix provides information on:

- KSAM file control and conversion to MPE files
- Adjust menu
- The HP32209 intrinsic
- The HP32209B utility

Convert KSAM File

Before converting a KSAM file to the now standard MPE files, you should first use FCOPY to physically remove any records which have been logically deleted in the file. This can be accomplished as follows:

```
:FCOPY FROM=ORDFORM;TO=(ORDFORM2,ORDKEY2)
```

This will copy only the active records in the KSAM file named ORDFORM to the new KSAM file named ORDFORM2 whose key file is ORDKEY2, also a new file.

The utility program CONVERT.PUB.SYS will create an MPE file from the KSAM file created above:

```
:RUN CONVERT.PUB.SYS  
PLEASE ENTER NAME OF THE OLD FORMS FILE.  
ORDFORM  
PLEASE ENTER THE NEW FORMS FILE NAME.  
ORDNEW  
DIRECTORY LENGTH=1264 ← system response  
END OF PROGRAM
```

The directory is found in an extension to the *comarea*, and consists of information regarding the location of the data records in the forms file. Its length varies from forms file to forms file and is indicated in the system response as in the example.

ADJUST MENU

The Adjust Menu is automatically displayed from the Forms File Menu the first time a forms file created before VPLUS/3000 version B. 03. 03 is accessed from FORMSPEC version B. 03. 03 or beyond.

```
FORMSPEC v.uu.ff Adjust Menu

This forms file must be adjusted for this version
                                and cross-checked to ensure its integrity.

BE SURE YOU HAVE A BACK-UP COPY OF THIS FORMS FILE.

Press  F5  to specify a new forms file.
Press  F8  to exit FORMSPEC.
Enter "CONTINUE" to resume cross-checking.
[        ]

PREV  NEXT  REFRESH  PREV  NEXT  MAIN/  EXIT
FORM  FORM  FORMSPEC  FORM  FORM  RESUME  FORM
```

Figure H-1. The Adjust Menu

You are given a choice of exiting FORMSPEC by pressing **F8**, or continuing with the adjustment process by entering CONTINUE into the only field on the Adjust Menu. It is recommended that you create a backup copy of the forms file before the adjustment. Once the forms file is converted to the B. 03. 03 (or later) format, it is not compatible with VPLUS/V versions prior to B. 03. 03.

Returns the version identifier for the installed version of the VPLUS/V intrinsics.

Syntax

```
HP32209 {buffer}
```

Parameters

buffer Fourteen-byte character string defined in an application to which the version identifier is returned.

Discussion

The version identifier is in the format `HP32209v.uu.ff` where *v* denotes the major enhancement level, *uu* the update level, and *ff* the fix level.

Example

```
COBOL  
CALL HP32209 USING BUFFER
```

```
BASIC  
200 CALL HP32209(B$)
```

```
FORTRAN  
CALL HP32209(BUFFER)
```

```
SPL/PASCAL  
HP32209(BUFFER)
```

THE HP32209B UTILITY

The HP32209B utility provides you with a simple way to confirm the version of the VPLUS/V software that you are currently running on the system. HP32209B resides in the PUB.SYS group and account. You can use it by typing the MPE command

```
:RUN HP32209B.PUB.SYS
```

The screen will display the product name and version as well as the version number of each segment of the software. This is to indicate any patches that may have a version number different from the product.

Refer to Appendix J for more information on the use of the MPE/XL version of this utility, called HP32209S, with HP Precision Architecture.

Example

When you run HP32209B, the screen displays the information in this format:

```
VPLUS INTRINSICS VERSION:  HP32209B.05.00
```

SEGMENT	VERSION
01	B'04'22'S01'00
02	B'04'22'S02'00
03	B'04'22'S03'00
04	B'04'22'S04'00
05	B'04'22'S05'00
06	B'04'22'S06'00
07	B'04'22'S07'00
08	B'04'22'S08'00
09	B'04'22'S09'00
10	B'04'22'S10'00
11	B'04'22'S11'00
12	B'04'22'S12'00
13	B'04'22'S13'00

An intrinsic called BITMAPCNV is provided to help decode and set bits in programming languages which do not have bit operations. It is particularly helpful in COBOL with the bit map returned by VGETFIELDINFO.

Syntax



```
BITMAPCNV {bitmap,bytemap,numbytes,function,cnvrr}
```

Parameters

<i>bitmap</i>	Source bit array containing bits to be converted to a byte array, or destination bit array for conversion from a byte array.
<i>bytemap</i>	Destination or source byte array. (For COBOL use @BYTEMAP)
<i>numbytes</i>	Two-byte integer set to number of bytes to be converted from or to.
<i>function</i>	Two-byte integer set to conversion type; 1 = bit to byte, 2 = byte to bit
<i>cnvrr</i>	Two-byte integer variable containing conversion error return; 0 = conversion successful, -1 = conversion failed, -2 = invalid input parameter

Discussion

This intrinsic can be used to set bits in the *comarea*. It can also be used to decode the error bit map returned by VGETFORMINFO. The BITMAPCNV intrinsic decodes a bit map to a byte array if the *function* parameter contains a value of one. Each byte in the byte array represents the corresponding bit in the bit map. If a bit in the bit map is on (value of one), the corresponding byte will contain the character of one (1). If the bit is off, the byte will contain the character of zero (0).

If the *function* parameter passed contains a value of two, the conversion will be from byte to bit. A bit map will be produced to reflect the 1's and 0's in a byte array.

When converting bytes to bits, anything in a byte other than the character zero (0) causes the corresponding bit in the bit map to be set to one (1).

Bit Map Conversion

In COBOL, BITMAPCNV can be called as shown in the example below.

Example

```
CALL "BITMAPCNV" USING BITMAP,@BYTEMAP,NUMBYTES,FUNCTION,CNVERR.
```

This appendix offers some guidelines for the migration of programs developed on MPE/V to MPE/XL systems, and a short list of peripherals and languages supported on MPE/XL systems.

It also provides information on those aspects of VPLUS/V that pertain exclusively to MPE/XL running on the HP Precision Architecture (HPPA) systems. They are:

- The intrinsics VGETIEEEREAL/VGETIEEELONG and VPUTIEEEREAL/VPUTIEEELONG
- The HP32209S utility

MIGRATION ISSUES

Programs that were originally developed on MPE/V must be rewritten to run in native mode on the MPE/XL. For details, refer to the following reference manuals:

- *Migration Process Guide* (Part No. 30367-90007); describes how to migrate existing applications to MPE/XL.
- *Programmers' Skills Migration Guide* (Part No. 30367-90005); explains new program development on MPE/XL and serves as a reference to the operating system.
- *COBOL II/XL Migration Guide* (Part No. 31500-90004); explains how to migrate COBOL II/V programs to MPE/XL.
- *FORTRAN 77/XL Migration Guide* (Part No. 31501-90004); details the migration of FORTRAN 77/V applications to MPE/XL.
- *HP Pascal/XL Migration Guide*; (Part No. 31502-90004); details the migration of Pascal applications to MPE/XL. migration of SPL programs to MPE/XL.

Some points to be noted when applications developed on VPLUS/V are run on MPE/XL are described below.

Terminals. When you run an application on MPE/XL, the terminal must be configured for the XON/XOFF handshake. If it is not, large or complex forms will appear to be corrupted when displayed on the screen.

Floating Point Numbers. The type conversion of floating point numbers in VPLUS/V assumes the HP 3000 floating point format, whereas native mode applications store floating point values in IEEE format. The type conversion will work, but the values will be wrong.

Native Mode Pascal Applications. The MPE/XL native mode Pascal compiler allocates 4 bytes for the integer subrange -32768..32767, compared with the 2 bytes allocated by the MPE/V Pascal compiler. You should use type SHORTINT in native mode Pascal applications to get a 2-byte allocation for this subrange.

Record padding in native mode Pascal will cause data misalignment problems for programmers using the VPLUS/V intrinsics, VGETFIELDINFO, VGETFORMINFO, and VGETFILEINFO. The problem can be avoided by packing the *infobuf* records passed as parameters.

Incorrect Parameters. Existing VPLUS/V applications written in COBOL and recompiled into MPE/XL native mode may fail to function. The application either fails with a VPLUS/V error code of 998 or 999, or it aborts with a Memory Fault error. The reason for this is that the programmer has coded the application to pass a display numeric data type as the parameter for a VPLUS/V intrinsic, instead of a binary numeric data type. That is, the COBOL data type PIC 9 has been coded instead of PIC 9 COMP.

This kind of mistake is not caught when the application is compiled, because most COBOL VPLUS/V applications do not use the compiler intrinsic mechanism, "CALL INTRINSIC... USING...". So, the error must be caught at run-time, if at all. It is possible for the defective program to appear to work correctly

in MPE/V and MPE/XL compatibility mode, because VPLUS/V is more forgiving when parameters of the wrong type are passed to it. However, native mode applications on MPE/XL call the VPLUS intrinsics through switch stubs, which require the correct parameter types. The wrong parameter causes a run-time error because the VPLUS native mode access layer does not have the capacity to edit the data passed through it.

If your application fails with a VPLUS error 998 or 999, or a memory fault, you can take the following steps:

1. Activate the VPLUS trace facility and examine the trace, especially the entering and leaving messages from the VPLUS native mode stub software.
2. Check the parameter content messages of the last VPLUS intrinsic call before the program failed.
3. Examine and correct VPLUS parameter declarations as needed, both in native mode code and in compatibility mode code if this is a migrated application.

Supported Peripherals and Applications

The following are currently supported on MPE/XL.

Printers-HPiB	2565, 2566, 2680, 2688
Terminals	2392, 2393, 2394, 2397, 2622, 2624B, 2627, HP 150, Vectra, Port+
Native Mode Languages	COBOL II/XL, FORTRAN 77/XL, HP Pascal/XL
Compatibility Mode Languages	SPL/V, RPG/V, Business BASIC/V, Transact/V

HPPA INTRINSICS

Introduction

On the MPE/V and previous systems, Hewlett Packard represented real or floating point numbers in a format unique to each machine. However, on the MPE/XL systems, these numbers are represented in a format that conforms with an IEEE standard. This means that programs running in native mode on MPE/XL systems will deal with numbers stored in the IEEE format. VPLUS/V will be running in compatibility mode, however, and will thus deal with numbers stored in the MPE/V format.

Currently, VPLUS/V offers intrinsics that will accept a real number in the MPE/V format, convert it to its equivalent external representation and put it into a form's data buffer. The four HPPA intrinsics described below have been developed to allow programs running in native mode to perform the same operations on a number stored in the IEEE format.

NOTE

Use these intrinsics only if you are programming entirely in native mode. If you are accessing a database in compatibility mode, you must use the VPLUS/V intrinsics, VGET/PUTtype.

VGETIEEEREAL

Copies character coded data contents from data buffer into an application, converting numeric value to IEEE floating point format.

Syntax

```
VGETIEEEREAL {comarea,fieldnum,variable}
```

Parameters

comarea Must be the *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETIEEEREAL:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VGETIEEEREAL may set the following *comarea* items:

cstatus Set to nonzero value if call is unsuccessful or if requested field has an error.

fieldnum Two-byte integer variable containing the number assigned to the field by FORMSPEC. The field itself may not be longer than 80 characters.

variable Variable within application, of type IEEE floating point real (32 bits), into which converted value is placed.

For example:

```
VGETIEEEREAL (COMAREA, FIELDNUM, VARIABLE);
```

Discussion

The content of the field denoted by the field number is read from the data buffer. This content must be numeric but its data type need not be; numbers in a field of CHAR type can be converted.

NOTE

The field number is a unique number assigned to each field by FORMSPEC and is completely independent of the field's position in the data buffer or the form. It only changes if the batch command, RENUMBER, is used. You can alter the length, position or other characteristics of a field without

VGETIEEEREAL

affecting the field number. The field number provides a way to locate fields regardless of their position.

The numeric value, stored in the buffer in character coded form, is converted to IEEE floating point format and then copied to the variable in the application. If errors occur during conversion, *cstatus* is set to the appropriate error code. If the requested field's error code is set, its value is moved to the variable but *cstatus* is set to an error code.

VGETIEEELONG

Copies character coded data contents from data buffer into an application, converting numeric value to IEEE floating point long format (64-bit).

Syntax

```
VGETIEEELONG {comarea,fieldnum,variable}
```

Parameters

comarea Must be the *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VGETIEEELONG:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VGETIEEELONG may set the following *comarea* items:

cstatus Set to nonzero value if call is unsuccessful or if requested field has an error.

fieldnum Two-byte integer variable containing the number assigned to the field by FORMSPEC. The field itself may not be longer than 80 characters.

variable Variable within application, of type IEEE floating point long (64 bits), into which converted value is placed.

For example:

```
VGETIEEELONG(COMAREA, FIELDNUM, VARIABLE);
```

Discussion

The content of the field denoted by the field number is read from the data buffer. The content of the field must be numeric, but its data type need not be; numbers in a field of CHAR type can be converted.

NOTE

The field number is a unique number assigned to each field by FORMSPEC and is completely independent of the field's position in the data buffer or the form. It only changes if the batch command, RENUMBER, is used. You can alter the length, position or other characteristics of a field without

VGETIEEELONG

affecting the field number. The field number provides a way to locate fields regardless of their position.

The numeric value, stored in the buffer in character coded form, is converted to IEEE floating point long format and then copied to the variable in the application. If errors occur during conversion, *cstatus* is set to the appropriate error code. If the requested field's error code is set, its value is moved to the variable but *cstatus* is set to an error code.

VPUTIEEEREAL

Writes a floating point number in IEEE standard format from an application to a specified field in the form data buffer, converting the value to character set coded external representation.

Syntax

```
VPUTIEEEREAL {comarea,fieldnum,variable}
```

Parameters

comarea Must be the *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPUTIEEEREAL:

estatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VPUTIEEEREAL may set the following *comarea* items:

estatus Set to nonzero value if call is unsuccessful.

numerrs May be decremented if a new value replaces the old value of a field with an error.

fieldnum Two-byte integer variable containing the number assigned by FORMSPEC to the field in which the value is written. The field itself may not be longer than 80 characters.

variable Variable within application, of type IEEE floating point real (32 bits), that contains the value to be converted to character set coded external representation and copied to a field in the data buffer.

For example: VPUTIEEEREAL (COMAREA, FIELDNUM, VARIABLE);

Discussion

This intrinsic converts an IEEE floating point real number to its character coded form and writes the converted value to a particular field in the data buffer, right justified. The exact format of the written data depends on the type of the destination field. For example, if the number "34.56" were put to a field of type DIG, the result would be "34" since such a field may contain only the digits 0 through 9. The destination field is identified by the field number assigned by FORMSPEC, and must be defined as a numeric field, that is, type NUM, IMP or DIG.

If the specified field had an error, VPUTIEEEREAL will clear the field's error flag and decrement *numerrs*.

VPUTIEEEREAL

NOTE

The field number is a unique number assigned to each field by FORMSPEC and is completely independent of the field's position in the data buffer or the form. It only changes if the batch command, **RENUMBER**, is used. You can alter the length, position or other characteristics of a field without affecting the field number. The field number provides a way to locate fields regardless of their position.

VPUTIEEEELONG

Writes a floating point number in IEEE standard long format from an application to a specified field in the form data buffer, converting the value to character set coded external representation.

Syntax

```
VPUTIEEEELONG {comarea,fieldnum,variable}
```

Parameters

comarea Must be the *comarea* name specified when the forms file was opened with VOPENFORMF. If not already set, the following *comarea* items must be set before calling VPUTIEEEELONG:

cstatus Set to zero.

comarealen Set to total number of two-byte words in *comarea*.

VPUTIEEE long may set the following *comarea* items:

cstatus Set to nonzero value if call is unsuccessful.

numerrs May be decremented if a new value replaces the old value of a field with an error.

fieldnum Two-byte integer variable containing the number assigned by FORMSPEC to the field to which the value is written. The field itself may not be longer than 80 characters.

variable Variable within application, of type IEEE floating point long (64 bits), that contains the value to be converted to character set coded external representation and copied to a field in the data buffer.

For example: VPUTIEEEELONG(COMAREA, FIELDNUM, VARIABLE);

Discussion

This intrinsic converts an IEEE floating point long number to its character coded form and writes the converted value to a particular field in the data buffer, right justified. The exact format of the written data depends on the type of the destination field. For example, if the number "34.56" were put to a field of type DIG, the result would be "34" since such a field may contain only the digits 0 through 9. The destination field is identified by the field number assigned by FORMSPEC, and must be defined as a numeric field, that is, type NUM, IMP or DIG.

If the specified field had an error, VPUTIEEEELONG will clear the field's error flag and decrement *numerrs*.

NOTE

The field number is a unique number assigned to each field by FORMSPEC and is completely independent of the field's position in the data buffer or the form. It only changes if the batch command, RENUMBER, is used. You can alter the length, position or other characteristics of a field without affecting the field number. The field number provides a way to locate fields regardless of their position.

THE HP32209S UTILITY

The HP32209S utility provides you with a simple way to confirm the versions of both the native mode software and the compatibility mode software that you are currently running on the system. HP32209S resides in the PUB.SYS group and account. You can use it by typing the MPE command

```
:RUN HP32209S.PUB.SYS
```

The screen will display the product name and version as well as the version number of each segment of the software. This is to indicate any patches that may have a version number different from the product.

This is a quick way to test whether native mode is moving through to compatibility mode on the MPE/XL system, and hence to confirm that the VPLUS/V software is accessible in compatibility mode.

Example

When you run HP32209S, the screen displays the information in this format, first for native mode and then for compatibility mode:

```
VPLUS INTRINSICS VERSION:  HP32209S.05.00
```

SEGMENT	VERSION
01	B'04'22'S01'00
02	B'04'22'S02'00
03	B'04'22'S03'00
04	B'04'22'S04'00
05	B'04'22'S05'00
06	B'04'22'S06'00
07	B'04'22'S07'00
08	B'04'22'S08'00
09	B'04'22'S09'00
10	B'04'22'S10'00
11	B'04'22'S11'00
12	B'04'22'S12'00
13	B'04'22'S13'00



A

Access, using VPLUS/V intrinsics, 6-28
Access, method errors, B-2
Add Check Digit option, REFSPEC, 5-43
ADDARBFIELD, description, 7-19b
Adding,
 a form, FORMSPEC, 3-3, 3-8, 3-37
 a reformat, REFSPEC, 5-19
 a save field, FORMSPEC, 3-38
Adjust Menu, H-4
Advanced,
 forms design, overview, 4-1
 levels of, 4-2
 processing, overview, 1-6, 1-11, 4-1
 specifications, 4-36
 statements, 1-13
Alphabetic check digits, calculation, D-4
Alter forms sequence, FORMSPEC, 4-50
Appended forms,
 ENTRY, 2-9
 FORMSPEC, 3-18
 with local form storage, 6-26
Application-ready Buffer, 1-13, 3-53a
 batch mode commands, 7-4, 7-19a
 benefits, 1-13a
 creating, 3-3, 3-7, 3-39
 data types, 3-15
 generating, 3-53d
 trace facility, 3-53i
 use in VGETBUFFER, 6-44
 using and defining, 3-53j
Applications hints, E-1
Applications programmer, using manual, 1-4
Applications user, 1-5
ARB, see Application-ready Buffer
ARBTOSCREEN, description, 7-19c
Arithmetic calculation, 1-13
Arithmetic expressions, FORMSPEC syntax, 4-11
ASCII, see USASCII
Assigning value to field, FORMSPEC, 4-38
Automatic data formatting, FORMSPEC, 4-41
AUTOREAD, 6-19, 6-109, G-11

B

Bar Code Reader, 3-32
BASIC,
 interface to VPLUS/V, A-1



- intrinsic call, 6-7
- parameters, 6-8
- sample program, A-58
- Batch file,
 - close, VPLUS/V intrinsic, 6-34
 - concatenating, 5-1, 5-45
 - creating, VPLUS/V intrinsic, 6-75
 - current record number, see RECNUM
 - deleting a record, VPLUS/V intrinsic, 6-115
 - expanding, 2-7
 - number of records, see RECNUM
 - open, VPLUS/V intrinsic, 6-81
 - protection against system crash, 5-3, 6-92
 - read, VPLUS/V intrinsic, 6-106
 - relation to other files, 5-2
 - REFSPEC, 5-1
 - Removing deleted records, 2-7
 - specification in ENTRY, 2-5
 - write to, VPLUS/V intrinsic, 6-124
- Bar code,
 - Defining from Data Capture Device Specifications Menu, 3-57
 - Defining from CONFIG phase, 4-7
- Batch mode, FORMSPEC in, 7-1
 - commands, 7-3
 - errors, B-12
- Batch record, read, 6-94
- Bit Map conversion, 1-1
- Blank replacement,
 - in output, REFSPEC, 5-25
 - FORMSPEC, 4-48
- Blink,
 - FORMSPEC, 3-14, 3-46
 - VPLUS/V intrinsic code, 6-13
- BLOCK MODE,
 - use in ENTRY, 2-8
 - use in FORMSPEC, 3-2
- Bracketed field, ENTRY, 2-11
- Brackets, field delimiters, FORMSPEC, 3-9
- BREAK key,
 - recovery from in,
 - ENTRY, 2-24
 - FORMSPEC, 3-7
 - REFSPEC, 5-13
 - steps to recovery, G-14
- Browse data, VPLUS/V intrinsic, 6-94
- BROWSE/COLLECT key, ENTRY, 2-3
- BROWSE/MODIFY
 - key, ENTRY, 2-16
 - requesting mode, 2-16
 - use of, 1-18
- Buffer,
 - configuration, terminal, G-13
 - control for ARB, 6-19, 6-43
 - data area, 6-4, 6-6

length, see DBUFLEN
read data from, VPLUS/V intrinsic, 6-43
write data to, VPLUS/V intrinsic, 6-86
BUILD command, with forms files, 3-27
Byte array, VPLUS/V intrinsic, 6-8

C

Calling VPLUS/V procedures, 6-7
Carriage return, block mode, 2-8
CDIGIT, Edit Statement, FORMSPEC, 4-29
Center data, FORMSPEC, 4-52
CFNAME, definition, 6-9, 6-14
CFNUMLINES, definition, 6-15
Change,
 a field or form, FORMSPEC, 3-8, 3-25
 forms file name, REFSPEC, 5-20
 statement, FORMSPEC, 4-9, 4-50
CHAR type, FORMSPEC, 3-15
Character,
 comparison, FORMSPEC, 4-23
 constants, specification of, 4-10
 data types, FORMSPEC, 3-15
 deletion,
 FORMSPEC, 4-4
 REFSPEC, 5-38
 formatting, FORMSPEC, 4-49
 set, USASCII, C-1
 string, data type, 6-8
Check digit,
 adding, REFSPEC, 5-43
 alphabetic, D-6
 calculation, D-1
 validation, 4-29
 validation, FORMSPEC, 4-29
Choice pattern, MATCH statement, 4-31
Close, VPLUS/V procedure for,
 batch file, 6-34
 forms file, 6-35
 terminal, 6-36
CMODE, definition, 6-12
COBOL,
 interface to VPLUS/V, A-1
 parameters, 6-8
 sample program, A-26
Coding, Touch Feature, E-7
Collating sequence, USASCII characters, C-1
Collection of data, simple, 1-6
COMAREA,
 definition, 6-9
 extension, 6-12
 initialization, 6-12
 size of extension, 6-12

- terminal information, G-11
- words, 6-12
- COMAREALEN, definition, 6-12
- Combining data in forms, REFSPEC, 5-7
- Command file, 7-2, 7-20
- Commas, in numeric data, FORMSPEC, 3-15
- Comments,
 - FORMSPEC field processing, 4-5
 - in FORMSPEC forms, 3-42
 - use of in command files, 7-3
 - use of in forms families, 3-31
- Communication area, 6-9, 6-11
- Comparisons, single value, 4-23
- Compiling,
 - forms file, 3-27
 - FORMSPEC in batch mode, 7-3
- Concatenating Batch Files, 5-45
- Conditional processing, FORMSPEC, 4-52
- Condition, FORMSPEC IF statement, 1-13, 4-52
- CONFIGuration statements, 4-7, 4-13
- Constant,
 - as field processing operand, 4-10
 - output file, REFSPEC, 5-32
- Continuing lines, field processing, 4-5
- Control statements, FORMSPEC, 4-9
- Conversion,
 - of data, REFSPEC, 5-35
 - in data movement, FORMSPEC, 4-43
 - numeric, VPLUS/V intrinsic, 6-65
- Convert,
 - buffer data, 6-65
 - output data to EBCDIC, REFSPEC, 5-25
- Converting KSAM to MPE forms files, H-3
- Copying a form,
 - FORMSPEC, 3-25, 3-39
 - FORMSPEC in batch mode, 7-1
- Correcting errors, ENTRY, 2-14
- Create batch file, 6-75
- CSTATUS, definition, 6-12
- Custom Error Message, 4-5

D

- Data buffer,
 - area, 6-4
 - length, see DBUFLEN
 - read, VPLUS/V intrinsic, 6-43
 - write to, VPLUS/V intrinsic, 6-86
- Data capture devices,
 - COMAREA items, 6-11
 - configuration phase, 4-13, 4-59
 - Data Capture Device Specifications Menu, 3-56
 - special features, 1-8, G-1

- Terminal/Language Selection Menu, 3-54, 6-5, 6-115
- Data Capture Device Specifications Menu, data capture devices, 3-56
- Data collection,
 - resumption in ENTRY, 2-16
 - use of, 1-17
- Data conversion,
 - FORMSPEC, 4-43
 - REFSPEC, 5-38
- Data editing,
 - simple, 1-8
 - VPLUS/V intrinsic, 6-44
- Data entry,
 - ENTRY, 2-1, 2-12
 - program, 1-17
 - VPLUS/V procedures, 6-40
- Data formatting,
 - FORMSPEC, 4-41
 - REFSPEC, 1-13, 5-36
- Data interpretation, FORMSPEC, 3-16
- Data justification, FORMSPEC, 5-56
- Data modification,
 - ENTRY, 2-16, 2-17
- Data movement, FORMSPEC, 1-13, 4-40
- Data processing, VPLUS/V procedures, 6-31
- Data reformatting,
 - REFORMAT, 5-1
 - overview, 1-19
- Data type,
 - ARB, 3-17a
 - conversion, FORMSPEC, 4-43
 - formatting, REFSPEC, 5-36
 - FORMSPEC field, 3-15
 - REFSPEC output field, 5-35
 - save fields, 3-53
 - screen, 3-15
 - use of, 1-9
 - VPLUS/V procedures, 6-8
- Data,
 - deletion, ENTRY, 2-7, 2-17
 - modify, VPLUS/V intrinsic, 6-115
 - read from buffer, VPLUS/V intrinsic, 6-43
 - source, 1-1
 - viewing it in ENTRY, 2-17
 - write to batch file, VPLUS/V intrinsic, 6-115
 - write to buffer, VPLUS/V intrinsic, 6-86
- Date,
 - comparisons, FORMSPEC, 4-24
 - constant, specification of, 4-10
 - conversion, FORMSPEC, 4-37
 - data types, FORMSPEC, 3-16
 - formatting, FORMSPEC, 4-42
 - type conversion, 3-53c
- Data Conversions, define, 3-50
- Date fields, interpretation, FORMSPEC, 3-17

DBUFLEN, definition, 6-15, 6-43
Decimal point, in numeric data, FORMSPEC, 3-15, 3-47
Default formatting, FORMSPEC, 4-44
DELARB, description, 7-19d
DELARBFIELD, description, 7-19e
DELETE key, ENTRY, 2-7
Delete record, 2-3
Delete sign, REFSPEC, 5-58
DELETEFLAG, definition, 6-16
Deleting,
 a form, FORMSPEC, 3-25, 3-38
 a form, FORMSPEC in batch mode, 7-3
 characters, FORMSPEC, 4-46
 characters, REFSPEC, 5-38
 data, ENTRY, 2-7, 2-17
 REFORMAT, 5-21
Delimiters,
 displaying, FORMSPEC, 3-9
 examples, 3-10, 3-13
 mixing printing and nonprinting, 3-10
 of fields in ENTRY, 2-11
Designer of forms, using manual, 1-4, 1-6, 1-22
Designing forms file, E-2
Designing form, FORMSPEC, 3-8
Destination field, FORMSPEC SET statement, 4-40
Detecting errors, in ENTRY, 2-1
DEVICE, 4-14
DIG field, FORMSPEC, 3-15
Directory area, H-3
Display,
 current form, VPLUS/V intrinsic, 6-108
 data at terminal, VPLUS/V intrinsic, 6-108
 enhancement,
 default, FORMSPEC, 3-14, 3-46
 ENTRY, 2-4
 forms, FORMSPEC, 3-5
 message, VPLUS/V intrinsic, 6-108
Display-only field, FORMSPEC definition, 3-14
DL area, use by VPLUS/V procedures, 6-11, E-6
DMY field, FORMSPEC, 3-16
Double (four-byte) integer,
 convert from USASCII, VPLUS/V intrinsic, 6-64
 convert to USASCII, VPLUS/V intrinsic, 6-90
 VPLUS/V intrinsic call, 6-8

E

EBCDIC, conversion to, REFSPEC, 5-25
Edit,
 errors, 6-22
 field, overview of, 1-6, 1-11
 fields, VPLUS/V intrinsic, 6-39
 statements,

- field processing summary, 4-18
- FORMSPEC, summary of, 4-8
- Editing, simple, 1-6, 1-8
- Edits, local, 4-16, 4-57, G-2
- End-of-file mark, batch file, 6-82
- Enhance window, VPLUS/V intrinsic, 6-92
- Enhancement,
 - codes,
 - FORMSPEC, 3-14, 3-34, 3-46
 - window, VPLUS/V procedures, 6-92
 - display,
 - default FORMSPEC, 3-14, 3-46
 - in ENTRY, 2-4
 - error FORMSPEC, 3-49
 - field, FORMSPEC, 3-46
- ENTER, key, 2-4
 - use in ENTRY, 2-11
- Entering data, ENTRY, 2-1, 2-11
- ENTRY,
 - files, 2-5
 - function keys, 2-3
 - local form storage, 2-8
 - program,
 - executing, 2-5
 - interrupting, 2-15
 - overview, 2-1
 - terminating, 2-15
 - use of, 1-17
- Environment, definition of, 6-19
- EQ statement, FORMSPEC, 4-23
- ERRFILENUM, definition, 6-17
- Error,
 - correction, ENTRY, 2-14
 - detection in ENTRY, 2-1
 - intrinsic, 6-1
 - enhancement, FORMSPEC, 3-49
 - flag, set with VPLUS/V intrinsic, 6-99
 - handling, VPLUS/V intrinsic, 6-22
 - in batch mode FORMSPEC, 7-2
 - light, HP3075/6 terminals, 4-15
 - logical, ENTRY, 2-14
 - messages,
 - custom, FORMSPEC, 4-5
 - ENTRY, 2-1
 - in FORMSPEC, B-2
 - file, REFORMAT, 5-44
 - list of VPLUS/V, B-1
 - return, VPLUS/V intrinsic, 6-37
 - VPLUS/V intrinsic, 6-22
 - number, file error, 6-17
 - system, ENTRY, 2-14
- Exit from ENTRY, 2-15
- EXIT key,
 - ENTRY, 2-3, 2-7, 2-9

- FORMSPEC, 3-4
- REFSPEC, 5-14
- Expanding files,
 - batch, 2-7
 - forms, 3-27
- Expressions, arithmetic, FORMSPEC syntax, 4-11
- Extension of COMAREA, BASIC, 6-12

F

- FAIL statement, FORMSPEC, 4-55
- Families of forms, 3-30
- Fast forms file, FORMSPEC, 3-39
- FCOPY,
 - use with ENTRY FILES, 2-7
 - with forms files, 3-27
- Features, VPLUS/V, 1-2
- Field,
 - delimiters,
 - ENTRY, 2-11
 - FORMSPEC, 3-9
 - edit,
 - overview, 4-20
 - phase, FORMSPEC, 4-60
 - statements, summary, 4-20
 - use of, 1-6, 1-10
 - VPLUS/V intrinsic, 6-39
 - enhancement,
 - FORMSPEC, 3-14
 - use of, 1-9
 - formatting output, 5-35
 - heading, FORMSPEC, 4-57
 - information, obtaining, 6-48
 - length,
 - as constant value, 4-10, 4-21
 - FORMSPEC, 3-12
 - REFSPEC output field, 5-32
 - save fields, 4-14
 - menu,
 - custom error message example, 4-5
 - FORMSPEC, 3-22
 - FORMSPEC processing specifications, 4-3
 - REFSPEC output, 5-34
 - sample of, 1-9
 - name,
 - as field processing operand, 4-9
 - FORMSPEC, 3-13
 - REFSPEC input field, 5-27, 5-31
 - REFSPEC output field, 5-31
 - Required/Optional, 5-31
 - save fields, 3-53
 - use of, 1-9
 - number,

- FORMSPEC, 3-12
- VPLUS/V procedures, 6-47, 6-88
- processing specifications, 4-20
 - correcting, 4-4
 - entering, 4-3
 - overview, 4-2
- processing statements,
 - conditional, FORMSPEC, 4-52
 - summary, 4-7
- REFSPEC output, 5-34
- separator, REFSPEC, 5-26
- specifications, 5-6
 - reformat file, 5-3
 - REFSPEC, 5-3
- standard formatting, 5-36
- substring 5-31
- symbol, table errors, B-6
- tag, FORMSPEC, 3-11
- TOGGLE key, FORMSPEC, 3-4
- type, use of, 1-8
- write to buffer, VPLUS/V intrinsic, 6-88
- Fields,
 - entering data, 2-11
 - maximum number on one form, 3-9
 - protected, definition of, 2-11
 - read from terminal, VPLUS/V intrinsic, 6-45
 - unprotected, definition of, 2-11
- File,
 - batch mode command, 7-10
 - command,
 - with ENTRY file, 2-7
 - with forms files, 3-27
 - error number (FCHECK code), see FILERRNUM
 - errors, 6-22
 - number,
 - error message file, see ERRFILENUM
 - print file, see ERRFILENUM
- File Equations,
 - FORMSPEC, 7-15, E-3
 - ENTRY, Intrinsic, E-4
 - REFSPEC, 5-44
- FILEN, definition of, 6-18
- FILERRNUM, definition, 6-17
- Files,
 - ENTRY, 2-5
 - for REFORMAT program, 5-3
 - FORMSPEC,
 - compiled forms file, 3-27
 - fast forms file, 3-39
 - forms file, 3-25
 - REFSPEC, 5-3
- FILL,
 - option, REFSPEC, 5-42
 - statement, FORMSPEC, 4-48

- Finish,
 - heading, FORMSPEC, 4-57
 - phase processing, VPLUS/V intrinsic, 6-41
- First record marker, REFSPEC output file, 5-32
- Float sign, REFSPEC, 5-40
- Floating decimal point, FORMSPEC, 3-15
- Forced,
 - data formatting, FORMSPEC, 4-44
 - failure, FORMSPEC edits, 4-55
- Form,
 - adding a, 2-9, 3-8, 3-37
 - appended, FORMSPEC, 3-18, 6-26
 - copy a, 3-25, 3-39, 7-1
 - definition area, 6-4
 - deleting a, 3-25, 3-38, 7-1
 - design,
 - advanced, overview, 4-1
 - processing, 1-11
 - FORMSPEC, 3-8
 - levels, of, 1-6
 - preparation for, 3-8
 - resuming, 3-7
 - display at terminal, VPLUS/V intrinsic, 6-108
 - families, 4-71, 6-26
 - function key labels, 3-43
 - frozen,
 - ENTRY, 2-9
 - FORMSPEC, 3-18, 3-41
 - initialize, VPLUS/V intrinsic, 6-68
 - layout,
 - example of, 1-8
 - FORMSPEC, 3-3, 3-44
 - modification, 3-8
 - planning for, 3-22
 - listing of, 3-28, 3-38
 - loading, 6-25
 - local storage, 2-8, 6-24, 6-26
 - menu, example, 1-7
 - modification, FORMSPEC, 3-8
 - name,
 - current, FORMSPEC, 3-41
 - current, FORMSPEC in batch mode, 7-10
 - head form, 3-41, 3-49
 - next, FORMSPEC, 3-41
 - print with, VPLUS/V intrinsic, 6-84
 - printing of in ENTRY, 2-4
 - processing,
 - advanced, 1-12
 - conditional, 4-52
 - phases, 4-57
 - STOP statement, 4-56
 - purging from local storage, 6-27
 - repeating,
 - ENTRY, 2-9

- FORMSPEC, 3-18, 3-41
- sequence, 1-13
 - altering, 4-50
 - ENTRY, 2-9
 - REFSPEC input file, 5-27
- storage directory, 6-25
- Format, VPLUS/V intrinsic calls, 6-7
- Formatting,
 - by data type, REFSPEC, 5-36
 - by command,
 - FILL, 4-48
 - JUSTIFY, 4-47
 - SET, 4-41
 - STRIP, 4-46
 - UPSHIFT, 4-49
 - character, FORMSPEC, 4-41
 - date, FORMSPEC, 4-42
 - numeric, FORMSPEC, 4-41
 - output data, REFSPEC, 5-35
 - specifications,
 - FORMSPEC, 4-50
 - REFSPEC, 5-38
- Forms and output records, REFSPEC, 5-5
- Forms designer, using manual, 1-4
- Forms file,
 - close, VPLUS/V intrinsic, 6-35
 - compiled version, 3-27
 - compilation, 3-27, 3-39
 - design, E-2
 - fast version, 3-39
 - layout of, 3-25
 - menu,
 - FORMSPEC, 3-2, 3-35
 - REFSPEC, 5-18
 - name,
 - altering, REFSPEC, 5-20
 - changing in FORMSPEC, 3-2, 3-35
 - changing in REFSPEC, 5-20
 - reformat file, 5-44
 - REFSPEC, 5-3
 - open, VPLUS/V intrinsic, 6-76
 - printing, 3-21
 - sequencing, 1-13, 2-9
 - source version, 3-25
 - specification,
 - in ENTRY, 2-5
 - in FORMSPEC, 3-2, 3-35
 - in REFSPEC, 5-18
- Forms files,
 - converting KSAM to MPE, H-3
 - expanding, 3-27
 - in batch mode FORMSPEC, 7-1
 - obtaining information, 6-54
 - printing, 3-28, 3-38

- purging, H-2
- recompiling, 3-27, 3-39
- renaming, H-2
- Formsloading error, B-57
- FORMSPEC,
 - adjust menu, H-4
 - advanced forms design, overview, 4-1
 - advanced processing, overview, 4-2
 - apply errors, B-20
 - batch mode
 - errors, B-12
 - overview, 7-1
 - error messages list, B-2
 - field edits, 1-10
 - form menu, 1-7
 - function keys, 3-4
 - hints for using, E-1
 - init and compile, B-16
 - menu controller errors, B-13
 - menus, 1-7, 3-5
 - parser errors, B-20
 - program,
 - executing, 3-2
 - overview, 3-1
 - terminating, 3-7
 - use of, 1-6
 - scanner errors, B-18
 - screen design, 1-8
- FORMSTORE'SIZE, definition, 6-17
- FORTRAN,
 - interface to VPLUS/V, overview, 1-14, 6-1, A-1
 - parameters, 6-8
 - sample program, A-38
- Freeze form, VPLUS/V intrinsic code, 6-15
- Freeze/Append option, FORMSPEC, 3-18, 3-41
- FREEZAPP, definition, 6-15
- Frozen forms,
 - ENTRY, 2-9
 - FORMSPEC, 3-18, 41
- Function Key Labels,
 - Accessing by intrinsics, 6-58, 6-104
 - Defining labels, 3-43, 3-51
- Function keys,
 - ENTRY, 2-3, 2-4
 - FORMSPEC, 3-4
 - define labels for, 3-43, 3-51
 - read, VPLUS/V intrinsic, 6-96
- REFSPEC, 5-14
- sequence control,
 - in FORMSPEC, 3-18
 - in REFSPEC, 5-14

G

GE statement, FORMSPEC, 4-23
 GENARB, description, 7-19f
 Get,
 next form, 6-62
 with VPLUS/V procedures, 6-62
 data from memory, 6-43
 form from forms file, 6-62
 Global,
 adding specifications, 5-20
 defining in REFSPEC, 5-24
 information, reformat file, 5-3
 specification,
 FORMSPEC, 3-48
 REFSPEC, 5-20
 use of, 1-13
 Globals,
 define function key labels, 3-51
 menu,
 FORMSPEC, 3-48
 REFSPEC, 5-24
 Grouping, MATCH statement, 4-32
 GT statement, FORMSPEC, 4-23

H

Half bright,
 FORMSPEC, 3-14, 3-46
 VPLUS/V intrinsic code, 6-13
 Head form,
 key, ENTRY, 2-3
 name, FORMSPEC, 3-27
 Header messages, B-66
 Hints for using FORMSPEC, E-1
 HP32209B intrinsic, H-5
 HP32209B utility, H-6
 HP32209S utility, J-11

I

IF statement, FORMSPEC, 4-52
 Implied decimal point, FORMSPEC, 3-15, 3-47
 IMPn field, FORMSPEC, 3-15, 3-47
 IN statement,
 FORMSPEC,
 range check, 4-27
 table check, 4-25
 Index retrieve operand, FORMSPEC syntax, 4-12
 Information procedures, 6-48
 INIT heading, FORMSPEC, 4-57
 Initial,

- phase processing, VPLUS/V procedures, 6-68
- value,
 - in field, FORMSPEC, 3-17, 3-47
 - save field, 3-53
 - use of, 1-10
- Initialization phase, FORMSPEC, 4-58
- Initialize,
 - COMAREA, 6-12
 - form, VPLUS/V intrinsic, 6-68
- Input,
 - field specification, REFSPEC, 5-27
 - FORMS,
 - changing menu, 5-20
 - menu, REFSPEC, 5-27
 - sequence modifying, 5-27
 - sequence, reformat file, 5-5
 - sequence, specification of, 5-36
- Integer,
 - convert,
 - from USASCII, VPLUS/V intrinsic, 6-64
 - to USASCII, VPLUS/V intrinsic, 6-90
 - VPLUS/V intrinsic call, 6-8
- Interface to programs, use of, 1-14
- Interpreting entered data, FORMSPEC, 3-16
- Interrupting,
 - data entry, ENTRY, 2-15
 - forms design, 3-7
 - recovery, G-14
 - REFSPEC, 5-13
- Intrinsics
 - calling, 6-7
 - dependency, 6-31
 - description of, 6-29
 - flow 6-29
 - operation, 6-5
 - summary of, 6-2
 - using, 6-1
- Inverse video,
 - FORMSPEC, 3-14, 3-46
 - VPLUS/V intrinsic code, 6-13

J

- JUSTIFY,
 - option, REFSPEC, 5-39
 - statement, FORMSPEC, 4-47

K

- Keyboard, terminal, 2-2
- Key definitions, REFSPEC, 5-14
- Key label area of memory, 6-6

KSAM file management,
 convert, H-3
 purge, H-2
 rename, H-2
 KSAM forms files, 2-6, 3-36, H-2

L

Label, batch file, 6-74
 Label option, COMAREA definition, 6-14
 LANGUAGE, COMAREA definition, 6-11
 LASTKEY, COMAREA definition, 6-13
 LE statement, FORMSPEC, 4-23
 Left justify,
 data, FORMSPEC, 4-47
 output data, REFSPEC, 5-39
 Length check, FORMSPEC edits, 4-21
 Levels of advanced forms design, 4-2
 LIGHT, configuration command, 4-15
 LIGHTSON definition of, 6-20
 Lines, continuation of, in field processing, 4-5
 Lines per page, adjustment, E-5
 Listing,
 forms, FORMSPEC, 3-28, 3-38
 forms, FORMSPEC in batch mode, 7-1
 reformat, REFSPEC, 5-21
 reformatted data, REFORMAT, 5-21
 Local form function key labels, 3-43
 form families, 6-26
 Local form storage,
 form families, 6-26
 form storage directory, 6-25
 FORMSTORESIZE, 6-17, 6-76
 HP 2624B terminal, 6-17, 6-24, G-3
 HP 2626A terminal, 6-24, G-3
 HP 2394A terminal, 6-24, G-3
 loading forms, 6-25
 look-ahead loading, 6-26
 LOOK'AHEAD, 6-15, 6-26, 6-96
 overview, 6-24
 preloading, 6-25
 purging, 6-26
 SHOWCONTROL, 6-16, 6-26, 6-108
 VLOADFORMS, 6-26, 6-70
 VUNLOADFORM, 6-112
 with appending forms, 6-27
 with ENTRY, 2-8
 with form families, 6-26
 LOCALEDITS, configuration command, 4-16
 Lockword,
 for forms file, 3-36
 on reformat file, 5-13
 Logic errors, ENTRY, 2-14

Logical, data type, 6-8
 Long data type,
 convert,
 from USASCII, VPLUS/V intrinsic, 6-64
 to USASCII, VPLUS/V intrinsic, 6-90
 usage, 6-8
 Look-ahead loading, 6-26
 LOOK'AHEAD, definition, 6-15
 LT statement, FORMSPEC, 4-23

M

Main Menu,
 FORMSPEC, 3-37
 REFSPEC, 5-19
 MAIN/RESUME key,
 FORMSPEC, 3-4
 REFSPEC, 5-14
 Manual, use of, 1-4
 MATCH,
 operator hierarchy, FORMSPEC, 4-34
 statement, FORMSPEC, 4-30
 MDY field, 3-16
 Memory, use by VPLUS/V procedures, 6-4
 Menu,
 definition, 2-9
 FORMSPEC,
 Data Capture Device Specifications, 3-56
 Default formatting, 4-41
 Field,
 description, 3-11, 3-45
 with processing specifications, 4-3
 with custom error messages, 4-5
 Form, 3-18, 3-40
 Form Function Key Labels, 3-43
 Forms File, 3-35
 Globals, 3-48
 Global Function Key Labels, 3-51
 INIT and Processing errors, B-7
 Main, 3-37
 Processing Table errors, B-7
 sample form designs, 3-23
 Save Field, 3-52
 Terminal/Language Selection, 3-54
 REFSPEC,
 Forms File, 5-18
 Globals, 5-24
 Input Forms, 5-27
 Main, 5-19
 Output Field, 5-34
 Output Record, 5-29, 5-30
 Reformat File, 5-17
 Menus,

- ease of forms design, 3-8
- FORMSPEC program, 3-5
- REFSPEC program, 5-15
- sequence in,
 - FORMSPEC, 3-5
 - REFSPEC, 5-15
- Message,
 - display at terminal, VPLUS/V intrinsic, 6-108
 - error return, VPLUS/V intrinsic, 6-37
 - FORMSPEC custom error, 4-5
 - move to window, VPLUS/V intrinsic, 6-92
- MINLEN statement, FORMSPEC, 4-21
- Mixing printing and nonprinting delimiters, 3-10
- MODARB, description, 7-19g
- Modifying,
 - data, ENTRY, 2-1, 2-16
 - field processing specifications, 4-4
 - form, FORMSPEC, 3-25
 - forms file, FORMSPEC, 3-25
 - reformat, REFSPEC, 5-20
- Modulus 10, check digit calculation, D-2
- Modulus 11, check digit calculation, D-3
- MOVEARBFIELD, description, 7-19h
- Movement of data, FORMSPEC, 4-40
- MPE/XL, migration issues, J-3
 - native mode intrinsics, J-4
 - Pascal applications, J-2
 - supported peripherals, J-3
 - terminals, J-2
- MULTIFUNCTION READER, 3-57
- Multilanguage, 6-1
- Multipurpose, 6-1
- Multiusage, definition, 6-14

N

- Naming files, ENTRY, 2-6
- Native Language Support,
 - date handling, 4-28, 8-3
 - ENTRY, 2-6, 8-4
 - errors, B-67
 - field edits, 8-3
 - international forms file, 8-2
 - intrinsics,
 - VGETLANG, 6-2, 6-60, B-55
 - VSETLANG, 6-2, 6-106, B-55
 - language attribute, 8-1
 - MATCH, 4-30
 - NATIVE-3000, 4-24, 8-1
 - numeric data, 3-16, 3-47, 8-4
 - overview, 8-1
 - pattern matching, 4-30
 - range checks, 4-28

- setting language ID number, 8-2
- single value comparisons, 4-24, 8-3
- table checks, 4-25
- Terminal/Language Selection Menu, 3-54
- upshifting, 4-49
- NE statement, 4-23
- NEXT FORM key,
 - ENTRY, 2-3
 - FORMSPEC, 3-4
- Next form name, FORMSPEC, 3-41
- NEXT key,
 - FORMSPEC, 3-4
 - REFSPEC, 5-14
- NEXT REFORMAT key, REFSPEC, 5-14
- NFNAME, definition, 6-14
- NIN statement,
 - range check, 4-27
 - table check, 4-27
- Nonprinting delimiters, 3-9
- Number of,
 - errors, set by VPLUS/V intrinsic, 6-99
 - lines in current form, see CFNUMLINES
 - records in batch file, see NUMRECS
- Numeric,
 - comparisons, FORMSPEC, 4-24
 - constants, specification of, 4-10
 - conversion,
 - FORMSPEC, 4-43
 - VPLUS/V intrinsic, 6-64, 6-90
 - data types, FORMSPEC, 3-15, 3-47
 - formatting, FORMSPEC, 4-41
- NUMERRS, definition, VPLUS/V intrinsic, 6-13
- NUM, standard formatting, 5-37
- NUMn field, FORMSPEC, 3-15, 3-47
- NUMRECS, definition, VPLUS/V intrinsic, 6-18

O

- Open,
 - batch file, VPLUS/V intrinsic, 6-72
 - forms file, VPLUS/V intrinsic, 6-76
 - terminal, VPLUS/V intrinsic, 6-78
- Operands, field processing statements, 4-9
- Operands, hierarchy, 4-34
- Operators, MATCH statement, 4-34
- Optional,
 - field,
 - ENTRY, 2-12
 - FORMSPEC definition, 3-14
 - specification, MATCH statement, 4-32
- Output Field Menu, REFSPEC, 5-34
- Output field specification,
 - reformat file, 5-6

- REFSPEC output record, 5-34
- Output file,
 - name, REFORMAT, 5-44
 - REFSPEC, 5-3
- Output record,
 - definition, 5-6
 - changing, 5-21
 - modifying, 5-30
 - reformat file, 5-3
 - REFSPEC, 5-6
- Output Record Menu, REFSPEC, 5-29

P

- Parameter types, VPLUS/V intrinsic, 6-8
- Parent form, 3-30
- PASCAL,
 - interface to VPLUS/V, overview, 1-14, A-1
 - parameters, 6-6
 - sample program, A-75
- Pattern match, FORMSPEC edit, 4-30
- Performance, with form families, 3-30
- Phase specification statements, FORMSPEC, 4-57
- Phases of form processing, FORMSPEC, 1-13, 4-57
- Preloading, 6-25, 6-108
- Preparing for forms design, 3-22
- PREVIOUS FORM key, FORMSPEC, 3-4
- PREVIOUS key,
 - ENTRY, 2-3
 - FORMSPEC, 3-4
 - REFSPEC, 5-14
- Print,
 - file number, see PRINTFILENUM
 - form, VPLUS/V intrinsic, 6-84
- PRINT key, ENTRY, 2-3
- Printed delimiters, 3-9
- PRINTFILNUM, definition, 6-17
- Printing data and forms, ENTRY, 2-4
- Process field, FORMSPEC definition, 3-14
- Processing specification, how to enter, 4-3, 4-13
 - Advanced Processing statement, 1-12, 1-13
 - Processing statement, 4-7
- Program generated data, ENTRY, 2-13
- Program,
 - interface, 1-15
 - interruption,
 - ENTRY, 2-15
 - FORMSPEC, 3-7
 - recovery, G-14
 - REFSPEC, 5-13
 - terminating,
 - ENTRY, 2-15
 - FORMSPEC, 3-7

- REFSPEC, 5-13
- Programmatic interface, use of, 1-14
- Programmer, using manual, 1-4
- Programming in BASIC, COMAREA extension, 6-12
- Programs, samples using VPLUS/V intrinsics, A-1
- Protected fields, definition of, 2-1
- Protecting batch file from system crash, 6-82
- Purging forms file, H-2
- Put, field in buffer, VPLUS/V intrinsic, 6-88

R

- Range checks, FORMSPEC, 4-27
- Read and convert field, VPLUS/V intrinsic, 6-64, 6-90
- Read,
 - batch record, VPLUS/V intrinsic, 6-94
 - data buffer, VPLUS/V intrinsic, 6-43
 - data from terminal, VPLUS/V intrinsic, 6-96
 - field,
 - and convert from USASCII, VPLUS/V intrinsic, 6-64
 - from memory, VPLUS/V intrinsic, 6-45
 - form from forms file, VPLUS/V intrinsic, 6-62
- Real value,
 - convert from USASCII, VPLUS/V intrinsic, 6-64
 - convert to USASCII, VPLUS/V intrinsic, 6-90
 - VPLUS/V intrinsic call, 6-8
- RECNUM, definition, 6-18
- Recompiling forms file, 3-27
- Record,
 - definition, reformat file, 5-3
 - delete from batch file, VPLUS/V intrinsic, 6-114
 - format,
 - batch file, 6-73
 - output file, REFSPEC, 5-24
 - length, output file, REFSPEC, 5-25
 - marker, REFSPEC, 5-6, 5-32
 - number,
 - batch file, see RECNUM
 - terminator, REFSPEC, 5-25
 - read from batch file, VPLUS/V intrinsic, 6-94
 - REFSPEC output definition, 5-31
 - write to batch file, VPLUS/V intrinsic, 6-114
- Records in batch file, see NUMRECS
- Recovery from program interruption,
 - ENTRY, 2-15
 - FORMSPEC, 3-7
 - REFSPEC, 5-13
 - steps to, G-14
- REFLIST file, REFORMAT, 5-45
- Reformat,
 - adding to reformat file, 5-19
 - deletion of, 5-21
 - design, resuming, 5-21

- error messages, list of, B-57, B-63
- file,
 - creates MPE, 5-3
 - compiling, 5-19
 - diagram of, 5-4
 - name,
 - REFORMAT, 5-44
 - REFSPEC, 5-19
 - REFSPEC, 5-3
 - specification, REFSPEC, 5-17
 - listing, 5-21
 - modification of, 5-20
- Reformat File Menu, REFSPEC, 5-17
- REFORMAT program,
 - executing, 5-44
 - streaming, 5-46
 - use of, 5-1
- Reformatting,
 - data, 2-1
 - capability, 1-19
 - conversion and movement, 5-36
 - REFORMAT, 2-1
 - order of, 5-35
 - repeating forms, REFSPEC, 5-10
 - overview, 5-1
 - standard 5-36
- REFRESH key,
 - ENTRY, 2-3
 - FORMSPEC, 3-4
 - REFSPEC, 5-14
- REFSPEC,
 - Access Method, B-24
 - error messages, list of, B-24
 - field table errors, B-30
 - function keys, 5-17
 - interruption, 5-13
 - menu processing, B-32
 - menus, 5-20
 - output sequence definition, B-25
 - program,
 - executing, 5-13
 - overview, 5-1
 - terminating, 5-13
 - use of, 1-18, 5-13
 - standard formatting, 5-36
 - validation error, B-28
- RELATE command,
 - use, 7-16
 - errors, B-14
- Relating forms,
 - FORMSPEC, 3-31, 3-39, 3-42
 - FORMSPEC in batch mode, 7-1, 7-4, 7-16
 - REFSPEC, 5-5
- Remote terminals, via DS facility, E-3

- Removing deleted records, 2-7
- RENAMEARBFIELD, description, 7-19i
- Repeat option, FORMSPEC form definition, 3-18, 3-41
- REPEATAPP, definition, 6-15
- Repeating form,
 - ENTRY, 2-9
 - FORMSPEC, 3-18, 3-41
 - reformatting, 5-10
 - VPLUS/V intrinsic code, 6-15
- Repetition, MATCH statement, 4-33
- Replace blanks,
 - in data, FORMSPEC, 4-48
 - in output, REFSPEC, 5-42
- Reproduced From field, 3-42
- Required field,
 - ENTRY, 2-12
 - FORMSPEC definition, 3-14
- Reserved word list, FORMSPEC, 3-13
- Resuming,
 - data collection, ENTRY, 2-16
 - design, 5-21
 - execution, FORMSPEC, 3-7
 - forms design, FORMSPEC, 3-25
 - reformat design, REFSPEC, 5-21
- Retries, definition of, 6-18
- Right justify,
 - data, FORMSPEC, 4-47
 - output data, REFSPEC, 5-39
- Root form, 3-30
- Running,
 - ENTRY, 2-5
 - REFORMAT, 5-44
 - REFSPEC, 5-13

S

- Sample batch data entry programs, A-1
- Sample command file, 7-2
- Save field,
 - adding to forms file, 3-38
 - data type, specification of, 3-53
 - deleting from forms file, 3-38
 - impossible to copy, 3-25
 - length, specification of, 3-53
 - menu, FORMSPEC, 3-52
 - name, specification of, 3-53
 - overview, 3-52
 - specification of, 3-53
 - use of, 1-14
- Screen definition errors, B-3
- SCREENTOARB, description, 7-19j
- Security display enhancement, 3-14, 3-33, 3-46, 3-49
- Selecting a function,

- FORMSPEC, 3-4
- REFSPEC, 5-14
- Separating data,
 - in forms, REFSPEC, 5-9
 - into several files, REFSPEC, 5-9, 5-12
- Separator, output field, REFSPEC, 5-26
- Sequence of USASCII characters, **ascending order, C-1**
- Sequence of forms,
 - altering in FORMSPEC, 4-50
- ENTRY, 2-9
- ENTRY, example, 2-10
- REFSPEC input file, 5-5
- Session, terminating, 2-15
- SET statement, syntax, FORMSPEC, **4-37**
- Set error flag, VPLUS/V intrinsic, 6-99
- Shift data to uppercase, FORMSPEC, **4-49**
- SHOWCONTROL,
 - data capture devices, 6-96
 - definition, 6-16
 - local form storage, 6-16
 - VREADFIELDS, 6-96
 - VSHOWFORM, 6-108
- Sign,
 - in entered data, FORMSPEC, 3-15, **3-47**
 - option, REFSPEC, 5-40
 - placement by REFSPEC, 5-40
- Simple collection, forms design, 1-6
- Single value comparisons, 4-23
- Source data entry, 1-1
- Source field, FORMSPEC SET statement, **4-40**
- Specifications, processing,
 - correction of, 4-4
 - how to enter, 4-3
 - specifying entry, 2-5
- SPL,
 - interface to VPLUS/V overview, 1-14, **6-1**
 - parameters, 6-8
 - sample program, A-1
- SPLIT MESSAGE PAUSE, 3-56
- SPLITPAUSE, definition of, 6-19
- Standard formatting, REFSPEC, 5-36
- Starting column,
 - input field, REFSPEC, 5-30
 - output field, REFSPEC, 5-31
- State code,
 - as constant value, 4-10
 - list of, F-1
- Statement summary, FORMSPEC editing, **4-8**
- Statements, field processing,
 - summary, 4-7
 - syntax, 4-5
- Status, VPLUS/V procedures, 6-12
- STOP statement, FORMSPEC, 4-56
- Strapping terminal, G-5

Stream REFORMAT program, 5-46
 STRIP,
 option, REFSPEC, 5-38
 statement, FORMSPEC, 4-46
 syntax, 4-46
 Substring specification, REFSPEC, 5-31
 Supported terminal features, G-2
 Syntax,
 FORMSPEC,
 edit statements, 4-8
 processing specifications, 4-5
 System constants,
 output file, REFSPEC, 5-32
 record terminator, REFSPEC, 5-25
 System crash, recovery, 2-15
 System errors, ENTRY, 2-14

T

Table checks, FORMSPEC, 4-25
 Terminal,
 buffer configuration, G-13
 close, VPLUS/V intrinsic, 6-36
 color enhancement, G-4
 communication area, 6-9, G-11
 data capture, G-4, G-10
 edits, 4-16, G-3
 features,
 defining in FORMSPEC, 3-33
 supported, 2-2, G-2
 information, G-1
 keyboard, 2-2
 local form storage, 6-3
 modified data tag, G-2
 open, VPLUS/V intrinsic, 6-89
 operator, using manual, 1-23
 relabelling function keys, G-2
 reset function, VOPENTERM, 6-78
 security display enhancement, G-2
 strapping, G-5
 supported by VPLUS/V, 2-2, G-2,3,5
 use, ENTRY, 2-2
 X.25 operation, G-4,7,9
 Terminal/Language Selection Menu, 3-54
 Terminating,
 ENTRY, 2-15
 FORMSPEC, 3-7
 REFSPEC, 5-13
 Terminator, output record, REFSPEC, 5-25
 errors, B-67
 TERMOPTIONS, definition of, 6-18
 Termtypes, choice of, G-2
 TESTLIST file, REFORMAT 5-45

- Text fields, FORMSPEC, 3-9
- Timeouts, 6-19, 6-97, G-12
- Today's date, as constant value, 4-10
- Touch Feature, coding, E-7
- Transaction Processing, 1-1
- Transact/3000, VPLUS/V interface, 1-1
- TRANSACT, sample program, A-66
- Transfer data,
 - from buffer, VPLUS/V intrinsic, 6-43
 - to buffer, VPLUS/V intrinsic, 6-86
- Transparency operator, MATCH statement, 4-31

U

- Underline,
 - FORMSPEC code for, 3-14, 3-46
 - VPLUS/V intrinsic code, 6-13
- Unexpected interruption,
 - ENTRY, 2-15
 - FORMSPEC, 3-7
 - REFSPEC, 5-13
 - Terminal recovery, G-14
- Unprotected fields, definition, 2-1, 2-11
- Uppercase,
 - shift data to,
 - FORMSPEC, 4-49
 - REFSPEC, 5-25
- Upshift data, output file, REFSPEC, 5-25
- User-defined commands, 5-45
- User environment control file, G-14
- Ustime, 6-19
- Using REFSPEC, 5-13
- UPSHIFT statement, FORMSPEC, 4-49
- USASCII character set, C
 - convert from, VPLUS/V intrinsic, 6-72
 - convert to, VPLUS/V intrinsic, 6-101
- USRBUFLN, definition, 6-12

V

- VPLUS/V, intrinsic messages, B-43
 - error messages, B-1
 - intrinsic,
 - by function, 6-28
 - call format, 6-7
 - dependency between, 6-31
 - error messages, list of, B-43
 - flow, 6-29
 - how to call, 6-7
 - information, 6-48
 - operational overview, 6-4
 - overview, 6-1

Index

- sample programs using, A-1
- status, COMAREA item, 6-12
- summary, 6-2
- use of, 1-14
- overview, 1-1, 6-1
- users, 1-14
- Value comparisons, FORMSPEC edits, 4-23
- VCHANGEFIELD, description of, 6-32
- VCLOSEBATCH, description, 6-34
- VCLOSEFORMF, definition, 6-35
- VCLOSETERM, description, 6-36
- VENVCNTL.PUB.SYS, G-14
- Verify check digit, FORMSPEC, 4-33
- VERRMSG, description, 6-42
- Version control, H-1
- VFIELDDEDITS, description, 6-39
- VFINISHFORM, description, 6-41
- VGETBUFFER, description, 6-43
- VGETFIELD, description, 6-45
- VGETFIELDINFO, description, 6-48
- VGETFILEINFO, description, 6-51
- VGETFORMINFO, description, 6-54
- VGETIEEELONG, description, J-5
- VGETIEEEREAL, description, J-3
- VGETKEYLABELS, description, 6-58
- VGETLANG, description, 6-60
- VGETNEXTFORM, description, 6-62
- VGETPACKED, description, 3-17a, 6-64
- VGETtype, description, 6-64
- VGETYYMMDD, description, 3-17a, 6-64
- VGETZONED, description, 3-17a, 6-64
- Viewing collected data, ENTRY, 2-17
- VINITFORM, description, 6-68
- VLOADFORMS, description, 6-70
- VOPENBATCH, description, 6-72
- VOPENFORMF, description, 6-76
- VOPENTERM, description, 6-78
- VPLACECURSOR, description, 6-80
- VPOSTBATCH, description, 6-82
- VPRINTFORM, description, 6-84
- VPRINTSCREEN, description, 6-85a
- VPUTBUFFER, description, 6-86
- VPUTFIELD, description, 6-88
- VPUTIEEELONG, description, J-9
- VPUTIEEEREAL, description, J-7
- VPUTPACKED, description, 6-90
- VPUTtype, description, 6-90
- VPUTWINDOW, description, 6-92
- VPUTYYMMDD, description, 6-90
- VPUTZONED, description, 6-90
- VREADBATCH, description, 6-94
- VREADFIELDS, description, 6-96
- VSETERROR, description, 6-99
- VSETKEYLABEL, description, 6-102

VSETKEYLABELS, description, 6-104
VSETLANG, description, 6-106
VSHOWFORM, description, 6-108
VTURNOFF, description, G-16
VTURNON, description, G-18
VUNLOADFORM, description, 6-112
VWRITEBATCH, description, 6-114

W

Window,
 area, 6-6
 enhancement, VPLUS/V code, 6-13
 line,
 enhancement, FORMSPEC, 3-49
 placement, FORMSPEC, 3-49
 move message to, VPLUS/V intrinsic, 6-92
WINDOWENH, definition, 6-13
Write data to,
 batch file, VPLUS/V intrinsic, 6-114
 buffer, VPLUS/V intrinsic, 6-86
Write field and convert to USASCII, 6-90
Write field to buffer, VPLUS/V intrinsic, 6-88

X

X.25 operation, G-4,7,9

Y

YMD field, FORMSPEC, 3-16, 3-47

\$

\$EMPTY, field processing constant, 4-10
\$END, last form FORMSPEC, 3-41, 6-14
\$HEAD, head form FORMSPEC, 3-41, 6-14
\$LENGTH, field processing constant, 4-10
\$REFRESH, refresh form FORMSPEC, 3-42, 6-14
\$RETURN, next form FORMSPEC, 3-41, 6-14
\$STATE, field processing constant, 4-10
\$TODAY, field processing constant, 4-10

