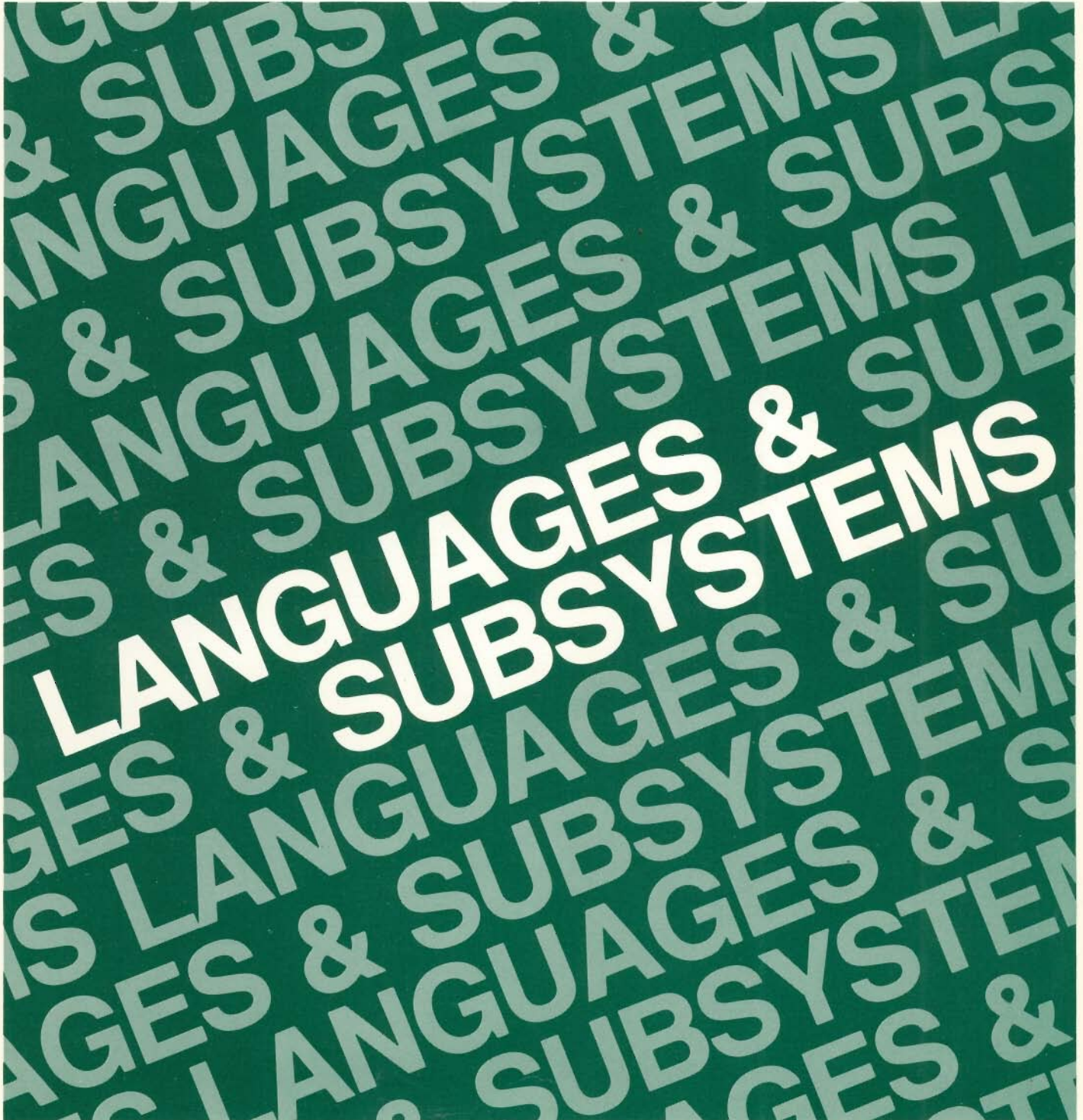


HP 3000 Computer Systems



V/R

RPG
reference manual



HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HP 3000 Computer Systems

V/R



RPG

Reference Manual



19447 PRUNERIDGE AVENUE, CUPERTINO, CALIFORNIA 95014

Part No. 32033-90058
Product No. 32104A

Printed in U.S.A. 07/86
E0786

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 program language without the prior written consent of 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 date on the title page and back cover of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated. No information is incorporated into a reprinting unless it appears as a prior update. This edition does not change.

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

First Edition	Feb 1975
Update Package #1	Aug 1976
Second Edition	Feb 1977
Update Package #2	Aug 1977
Update Package #2 Incorporated	Dec 1977
Update Package #3	Nov 1978
Update Package #3 Incorporated	Apr 1979
Update Package #4	May 1980
Update Package #5	Jan 1983
Update Package #5 Incorporated	May 1983
Update Package #6	Aug 1984
Third Edition	Aug 1985

NOTE

This manual corresponds with
the software version of MPE V/R.

PREFACE

This manual explains how to use the Report Program Generator for the HP 3000 Computer Systems (RPG/3000). Specifically, it shows how to write source programs in the RPG/3000 programming language and compile them into object programs with the RPG/3000 compiler.

To use this manual, you should understand the fundamental techniques of programming. Experience with the RPG language for another computer system, while not mandatory, will help. For information about RPG on a more introductory or tutorial level, please see any of the excellent texts available. Among these are:

Bernard, Solomon Martin, *System/360 Report Program Generator*. New Jersey, Prentice-Hall, Inc., 1970.

Brightman, Richard W. and Clark, John R., *RPG I and RPG II, System/3 and System/360*. Ontario, The Macmillan Company, Collier-Macmillan Canada, Ltd., 1970.

The RPG/3000 compiler operates under control of the HP 3000 Multiprogramming Executive Operating System (MPE/3000). Directions on how to use the operating system to access the computer, specify input/output files, invoke the compiler, and execute object programs, appear in Section XI of this manual. If you are working with the MPE-III Operating System, you can find more information about MPE in the following manuals:

Manual Title	Part No.
<i>HP 3000 Computer Systems MPE Commands Reference Manual</i>	30000-90009
<i>HP 3000 Computer Systems MPE Intrinsics Reference Manual</i>	30009-90010
<i>HP 3000 Computer Systems Console Operator's Guide</i>	30000-90013

If you are working with the MPE-C Operating System, you can learn more about MPE in:

Manual Title	Part No.
<i>MPE/3000 Operating System Reference Manual</i>	32000-90002
<i>MPE/3000 Operating System Console Operator's Guide</i>	32000-90004

The RPG/3000 compiler may generate programs that interface with the Keyed Sequential Access Method (KSAM/3000), the IMAGE Data Base Management System (IMAGE/3000), and the source data entry system, V/3000. For information about these subsystems beyond that described in this manual, please read:

Manual Title	Part No.
<i>IMAGE/3000 Data Base Management System Reference Manual</i>	30000-90041
<i>KSAM/3000 Reference Manual</i>	30000-90079
<i>V/3000 Reference Manual</i>	32209-90001

CONTENTS

Section I	Page		
INTRODUCTION TO RPG/3000			
Applications	1-1		
Capabilities and Advantages	1-1		
New Features	1-2		
Using RPG/3000	1-2		
Compatibility with Other Systems	1-11		
System Requirements	1-11		
Section II	Page		
COMMON ENTRIES			
General Information	2-1		
Coding Lines	2-1		
Sequence Number (Columns 1 through 5)	2-3		
Form Type (Column 6)	2-4		
Comment Record Indicator (Columns 6 or 7)	2-4		
Program Name (Columns 75 through 80)	2-5		
Section III	Page		
CONTROL RECORD SPECIFICATION			
Fields	3-1		
Sequence Number (Columns 1 through 5)	3-1		
Form Type (Column 6)	3-1		
Error Dump Filename (Columns 7 through 14)	3-3		
Debug (Column 15)	3-3		
USWITCH Source (Column 16)	3-4		
UPDATE Source (Column 17)	3-9		
Line Number Option (Column 20)	3-11		
Inverted Print (Column 21)	3-11		
Record Number Adjust (Column 22)	3-13		
Program Name Logging (Column 25)	3-13		
Alternate Collating Sequence (Column 26)	3-14		
BUFCHK Defaults (Column 28)	3-18		
Table/Array Look-Up (Column 34)	3-19		
EBCDIC Zone/Digit Tests (Column 39)	3-21		
Sign Process (Column 40)	3-21		
Form Positioning (Column 41)	3-22		
Indicator Setting (Column 42)	3-23		
File Translation (Column 43)	3-25		
Non-Numeric Digits (Column 44)	3-29		
Dollar-Sign Substitute (Column 46)	3-30		
Skip Suppress (Column 47)	3-32		
Record Length Check (Column 49)	3-33		
Page Overflow Test (Column 50)	3-33		
*PLACE Method (Column 51)	3-34		
Cross-Reference Listing (Column 52)	3-34		
Carriage Control Type (Column 53)	3-35		
Textfile Sequence Check (Column 54)	3-38		
Error Log (Column 55)	3-39		
Error Response (Columns 56 through 71)	3-40		
Program Name (Columns 75 through 80)	3-41		
Default Summary	3-41		
Section IV	Page		
FILE DESCRIPTION SPECIFICATIONS			
Fields	4-1		
Sequence Number (Columns 1 through 5)	4-1		
Form Type (Column 6)	4-3		
File Name (Columns 7 through 14)	4-3		
File Type (Column 15)	4-5		
File Designation (Column 16)	4-6		
End-of-File (Column 17)	4-7		
Input Sequence Check (Column 18)	4-8		
Record Format (Column 19)	4-9		
Block Length (Columns 20 through 23)	4-9		
Logical Record Length (Columns 24 through 27)	4-11		
Processing Mode (Column 28)	4-12		
Record Address Field Length (Columns 29 through 30)	4-14		
Record Address Type (Column 31)	4-15		
File Organization/Additional I/O Area (Column 32)	4-15		
Overflow Indicator (Columns 33 through 34)	4-18		
Key Field Starting Locations	4-19		
Summary of KSAM/RSAM Field Formats	4-19		
Extension Code (Column 39)	4-20		
Device Class Name (Columns 40 through 46)	4-22		
Interface Type (Column 47)	4-23		
Interface Control (Columns 48 through 52)	4-24		
Disc Labels (Column 53)	4-24		
Name of Label Exit (Columns 54 through 59)	4-25		
File Addition (Column 66)	4-26		
Extents (Columns 68 through 69)	4-27		
File Conditioner (Columns 71 through 72)	4-28		
Program Name (Columns 75 through 80)	4-28		
Continuation Records	4-29		
Sequence Number (Columns 1 through 5)	4-29		
Form Type (Column 6)	4-29		
Long Name Option Target (Columns 20 through 51)	4-29		
Continuation Code (Column 53)	4-29		
File Processing Option (Columns 54 through 74)	4-29		
General Group Fields	4-30		
File-sharing Group Fields	4-35		
Data Base Management Group Fields	4-36		
Data Base Name (IMAGE) Records	4-38		
Dummy Item Name (ITEMXX) Records	4-42		
Item Name (ITEM) Records	4-42		
Password (LEVEL) Records	4-43		
Data Set Name (DSNAME) Records	4-43		
Key File Name (KEYFL) Records	4-44		
Default Summary	4-47		

CONTENTS (continued)

Section V	Page	Channel Number Option Fields	6-2
FILE EXTENSION SPECIFICATIONS		Sequence Number (Columns 1 through 5)	6-2
Fields	5-1	Form Type (Column 6)	6-2
Sequence Number (Columns 1 through 5)	5-1	File Name (Columns 7 through 14)	6-2
Form Type (Column 6)	5-1	Line Number (Columns 15 through 17)	6-3
Chaining File Record Sequence (Columns 7 through 8)	5-3	Channel Number/OL (Columns 18 through 19)	6-3
Chaining Code Identifier (Columns 9 through 10)	5-3	Line Number and Channel Number/OL/FL (Columns 20 through 74)	6-5
From Filename (Columns 11 through 18)	5-5	Program Name (Columns 75 through 80)	6-5
To Filename (Columns 19 through 26)	5-8	Line Number Option Fields	6-5
Table, Array, or Routine Name (Columns 27 through 32)	5-9	Sequence Number (Columns 1 through 5)	6-5
Entries Per Record (Columns 33 through 35)	5-10	Form Type (Column 6)	6-5
Entries Per Table/Array (Columns 36 through 39)	5-12	File Name (Columns 7 through 14)	6-5
Entry Length (Columns 40 through 42)	5-13	Line Number (Columns 15 through 17)	6-6
Data Format (Column 43)	5-14	OL/FL (Columns 18 through 19)	6-6
Decimal Positions (Column 44)	5-14	Line Number and Channel Number/OL/FL (Columns 20 through 24)	6-6
Table/Array Sequence (Column 45)	5-15	Default Summary	6-7
Table/Array Name (Columns 46 through 51)	5-16		
Entry Length (Columns 52 through 54)	5-17	Section VII	Page
Data Format (Column 55)	5-18	INPUT SPECIFICATIONS	
Decimal Positions (Column 56)	5-18	Fields	7-1
Table/Array Sequence (Column 57)	5-19	Sequence Number (Columns 1 through 5)	7-1
Comments (Columns 58 through 74)	5-19	Form Type (Column 6)	7-1
Program Name (Columns 75 through 80)	5-19	Record Description Fields (Columns 7 through 42)	7-1
Required/Optional/Prohibited Entry Summary	5-19	File Name (Columns 7 through 14)	7-1
Default Summary	5-21	Group Sequence (Columns 15 through 16)	7-3
Rules For Using Tables and Arrays	5-22	Number of Records (Column 17)	7-4
Creating Tables and Arrays	5-22	Option (Column 18)	7-6
Rules For Creating All Table and Arrays	5-22	Local Data Area/User Data Structure	7-6
Rules For Creating Compile Time and Pre-Execution Time Table and Arrays	5-22	Record Indicator/Look-Ahead/Trailer/Data Structure (Columns 19 through 20)	7-8
Defining Table and Arrays	5-23	Indicators	7-9
For All Tables and Arrays	5-23	Look-Ahead Fields	7-13
For Compile Time or Pre-Execution Time Tables and Arrays	5-23	Spread Records	7-13
Loading Tables and Arrays	5-24	Data Structures	7-16
Compile Time Tables and Arrays	5-24	Record Identification Codes (Columns 21 through 41)	7-16
Loading From Records Appended to the Source Program	5-25	Position (Columns 21 through 24, 28 through 31, and 35 through 38)	7-16
Loading From Disc Files	5-27	Not (Columns 25, 32 and 39)	7-18
Loading From Mixed Media	5-29	Portion (Columns 26, 33 and 40)	7-18
Pre-Execution Time Tables and Arrays	5-30	Character (Columns 27, 34 and 41)	7-20
Execution Time Arrays	5-30	AND Lines	7-20
Searching Tables and Arrays	5-31	OR Lines	7-22
Using Tables and Arrays	5-33	Stacker Select (Column 42)	7-22
Writing Tables and Arrays to Output Files	5-34	Field Description Fields (Columns 43 through 70)	7-24
		Data Format (Column 43)	7-24
		Unpacked Decimal and Alphanumeric Formats	7-26
Section VI	Page	Unpacked Decimal Format with Leading Sign	7-26
LINE COUNTER SPECIFICATIONS		Unpacked Decimal Format with Trailing Sign	7-26
Channel Number Option	6-1		
Line Number Option	6-1		
Form Overflow and Length	6-1		
Fields	6-2		

CONTENTS (continued)

Packed Decimal Format	7-27	Searching a Single Table	8-52
Binary Format	7-27	Searching Related Tables	8-52
Conversion Method	7-28	Searching an Array	8-52
Specifying Data Formats	7-29	File Operations	8-56
From Field Position (Columns 44 through 47)	7-29	Debug Operation	8-78
To Field Position (Columns 48 through 51)	7-30	Record Number Conversion Operations	8-80
Decimal Positions (Column 52)	7-31	Internal Conversion Routines	8-80
Field Name (Columns 53 through 58)	7-31	External Conversion Routines	8-82
Data Structures	7-35	Default Summary	8-89
Control Level (Columns 59 through 60)	7-38		
Matching/Chaining Fields (Columns 61		Section IX	Page
through 62)	7-39	OUTPUT SPECIFICATIONS	
Matching Fields	7-41	Fields	9-1
Chaining Fields	7-46	Sequence Number (Columns 1 through 5)	9-1
Field Record Relation (Columns 63		Form Type (Column 6)	9-1
through 64)	7-51	Record Description Fields (Columns 7	
Field Indicator Field (Columns 65		through 31)	9-1
through 70)	7-54	File Name (Columns 7 through 14)	9-3
Program Name (Columns 75 through 80)	7-55	Type (Column 15)	9-4
Default Summary	7-55	Record Addition/Deletion (Columns 16	
		through 18)	9-5
Section VIII	Page	Stacker Select (Column 16)	9-6
CALCULATION SPECIFICATIONS		Fetch Overflow (Column 16)	9-8
Field Descriptions	8-2	Release Column (Column 16)	9-9
Sequence Number (Columns 1 through 5)	8-2	Space (Columns 17 through 18)	9-10
Form Type (Column 6)	8-2	Skip (Columns 19 through 22)	9-12
Control Level (Columns 7 through 8)	8-2	Output Indicators (Columns 23 through 31)	9-13
Control Level Indicators (L0 through L9)	8-4	General Indicators	9-14
Last Record Indicator (LR)	8-4	Command Key Indicators	9-15
Subroutine Identifier (SR)	8-4	Function Key Indicators	9-15
AN/OR Lines	8-4	Control Level Indicators	9-15
Indicators Field (Columns 9 through 17)	8-5	Last Record Indicator	9-15
Factors 1 and Factor 2 Fields (Columns		Matching Record Indicator	9-15
18 through 27 and 33 through 42)	8-8	Overflow Indicator	9-15
Operation (Columns 28 through 32)	8-12	First-Page Indicator	9-17
Result Field (Columns 43 through 48)	8-15	Halt Indicators	9-17
Field Length (Columns 49 through 51)	8-16	User Indicators	9-17
Decimal Positions (Column 52)	8-16	Error Conditions	9-17
Half Adjust (Column 53)	8-17	Not (Columns 23, 26 or 29)	9-18
Resulting Indicators (Columns 54		AND Lines	9-18
through 59)	8-17	OR Lines	9-18
Testing Results of Operations	8-19	Field Description Fields (Columns 32	
Setting Indicators	8-22	through 70)	9-19
Comments (Columns 60 through 74)	8-22	Field Name (Columns 32 through 37)	9-19
Program Name (Columns 75 through 80)	8-22	Edit Code (Column 38)	9-25
Operations	8-22	Simple Edit Codes	9-26
Arithmetic Operations	8-22	Complex Edit Codes	9-26
Move Operations	8-24	Blank After (Column 39)	9-29
Compare and Test Operations	8-31	End Position (Columns 40 through 43)	9-29
Branching and Exiting Operations	8-34	RPG Screen Interface From Name	
Branching Operations	8-34	Specification (Columns 42 through 43)	9-31
Exiting Operations	8-36	Output for Printed Reports, Punched	
Internal Subroutines	8-38	Cards or Disc	9-32
External Subroutines	8-41	Output for Card Reader Punch	9-32
Indicator and Bit Setting Operations	8-48	Packed/Binary (Column 44)	9-34
Lower-Limit Operation	8-50	Constant/Edit Word (Columns 45	
Table/Array Look-Up Operation	8-50	through 70)	9-35

CONTENTS (continued)

Constants	9-35	Output Files	11-25
Edit Words	9-36	Newfile	11-25
Comments (Columns 71 through 74)	9-41	Uslfile	11-26
Program Name (Columns 75 through 80)	9-41	User File Characteristics	11-28
Default Summary	9-44	Compiler Secondary Entry Points	11-29
Printer Spacing Chart	9-45		
Section X	Page	Section XII	Page
RPG/3000 PROGRAMMING APPLICATIONS		COMPILER SUBSYSTEM COMMANDS	
A Simple RPG/3000 Program	10-1	Syntax and Format	12-1
Analyzing the Application	10-1	Parameters	12-2
Writing the Program	10-2	Comments	12-3
Control Record and File Description		Continuation Records	12-3
Specifications	10-2	Effects of Commands	12-4
File Extension and Line Counter		Command Summary	12-5
Specifications	10-4	Listing and Compilation Options (\$CONTROL	
Input Specifications	10-4	Command)	12-5
Calculation Specifications	10-4	COPYLIB Processing (\$COPY Command)	12-8
Printer Spacing Chart and Output		COPYLIB File Naming (\$INCLUDE Command)	12-9
Specifications	10-9	Conditional Compilation (\$IF Command)	12-9
Compiling, Preparing and Executing the		Software Switches for Conditional Compilation	
Program	10-9	(\$SET Command)	12-10
KSAM File Applications	10-20	Page Title in Standard Listing	
Creating KSAM Files	10-20	(\$TITLE Command)	12-11
KSAM Files and Chain Operation	10-23	Page Title and Ejection (\$PAGE Command)	12-12
KSAM Files and Chaining Codes	10-27	Source Text Merging and Editing (\$EDIT	
KSAM Files, RAFs, and Processing		Command)	12-13
Between Limits	10-31	Merging	12-13
KSAM Files, RAFs, and Random		Checking Sequence Fields	12-15
Processing	10-36	Editing	12-15
IMAGE File Applications	10-36		
Generating a Report through IMAGE	10-39	Section XIII	Page
Creating an IMAGE Data Base	10-41	RPG INTERFACE TO V/3000	
Updating a Data Base	10-43	Overview	13-1
CARD-READER/PUNCH Applications	10-44	Operation of the V/3000 Interface	13-1
		WORKSTN File	13-1
		Other Files	13-2
Section XI	Page	Actions and Events	13-2
OPERATING THE RPG/3000 COMPILER		Runtime Errors	13-2
Command Elements	11-3	Break Key Disabled	13-3
Positional Parameters	11-4	Forms Downloading	13-3
Keyword Parameters	11-4	Using The V/3000 Interface	13-4
Continuation Characters	11-5	Data Entry Through RPG	13-4
Command Errors	11-5	Events	13-4
Reference Specifications	11-6	Actions	13-5
:RPG	11-8	Record Types	13-9
:RPGGO	11-10	Input Record Formats	13-9
:RPGPREP	11-11	Output Record Formats	13-11
Sample Jobs and Sessions	11-13	Handling Runtime Errors	13-14
Compiler Listing Output	11-14	Error Dump	13-14
Program Listing	11-15	Error Message Display	13-14
Symbol Table Listing	11-17	Break Key	13-15
Error Messages	11-18	Trace File	13-15
Cross-Reference Listing	11-18	Special WORKSTN Fields	13-15
Compiler Input/Output File Characteristics	11-21	Function Key Labels Enable	13-15
Input Files	11-21	Error Message Interval	13-15
Textfile	11-21	BREAK Key Enable	13-15
Master File	11-24	Example Using the V/3000 Interface	13-16

CONTENTS (continued)

Program Plan	13-16		Page
Control Record Specification	13-19	Appendix E	
File Description Specification	13-19	OBJECT PROGRAM LOGIC FLOW CHART	E-1
Input Specifications	13-20		
Calculation Specifications	13-21	Appendix F	Page
Output Specifications	13-23	SPECIFICATION SHEET SUMMARIES	
Compiled Version	13-24	Control Record Specification	F-1
		File Description Specifications	F-4
Section XIV	Page	File Extension Specifications	F-10
RPG COPYLIB PROCESSOR (RPGCOPY)		Line Counter Specifications	F-12
Inserting Copylib Records	14-1	Input Specifications	F-14
Modifying Copylib Records	14-2	Calculation Specifications	F-17
		Output Specifications	F-20
Appendix A	Page		
COMPATIBILITY WITH OTHER RPG SYSTEMS		Appendix G	Page
Functional Differences Requiring Conversion	A-1	ASCII CHARACTER SET AND COLLATING	
Printer Files	A-1	SEQUENCE	G-1
Card Reader/Punch/Interpreter	A-1		
Edit Words	A-1	Appendix H	Page
Differences in Character Codes	A-2	SIMULATING ISAM FUNCTIONS WITH	
Device Class Names	A-2	IMAGE/3000	
Rewind Operations	A-2	Performance	H-1
Quotation Marks	A-2	ISAM Simulation Rules	H-1
File and Program Names	A-2	Coding Specifications for ISAM Simulation	H-3
Non-Supported Features	A-3	File Description Specifications	H-3
Sterling Notation	A-3	File Description Specifications Continuation	
Telecommunications	A-3	Records	H-4
ULABL Operation	A-3	Data Base Name Records	H-4
Control Record Operations	A-3	Item Name Records	H-4
RPG/3000 Extensions	A-3	Applications	H-4
External Subroutine Call Parameters	A-3	Creating and Listing an Indexed File	H-5
Run-Time Error Options	A-4	Processing an Indexed File Between Limits	H-5
Cross-Reference Listing Option	A-4	Setting the Lower Limit with the SETLL	
Automatic Program Segmentation	A-4	Operation	H-8
Character Translation	A-4		
Partial Field Translation	A-4	Appendix I	Page
Combined Input/Output (Terminal) File	A-4	USER-WRITTEN ISAM PROCEDURES	I-1
Calculation Indicator Repetition	A-4		
Compile-Time Tables/Arrays on Separate		Appendix J	Page
Disc Files	A-5	RPG CODING AND PLANNING FORMS	J-1
Appendix B	Page	Appendix K	Page
DIAGNOSTIC MESSAGES		RPG SCREEN INTERFACE	K-1
Compile-Time Messages	B-1		
Run (Execution) Time Errors	B-76	Appendix L	Page
RPG-Detected Errors	B-76	MPE USER MESSAGE SYSTEM	
IMAGE-Detected Errors	B-79	MPE User Message System	L-1
USWITCH Command Errors	B-79	Message Catalog	L-1
		MAKECAT.PUB.SYS Program	L-2
Appendix C	Page		
DEBUGGING TECHNIQUES		Appendix I	Page
Formatted Portion	C-1	INDEX	Index-1
Unformatted Portion	C-2		
Appendix D	Page		
OBJECT PROGRAM OPERATING CYCLE	D-1		

ILLUSTRATIONS

Title	Page	Title	Page
Steps for Preparing and Running an RPG/3000 Program	1-3	Coding Input Specifications for Chaining	7-50
RPG/3000 Source Program	1-5	Using Indicators to Relate Field to Record Types	7-53
Printer Spacing Chart	1-6	Using Field Indicators	7-56
RPG/3000 Source Program Input Records	1-8	Calculation Specifications Sheet	8-3
RPG/3000 Source Program Listing	1-9	Using Three General Indicators	8-9
RPG/3000 Cross-Reference Listing	1-10	Using a Field Indicator	8-10
RPG/3000 Object Program Output (Report)	1-12	Literals	8-12
Control Record and File Description Specifications Sheet	2-2	Specifying Field Length and Roundings	8-18
Control Record and File Description Specifications Sheet	3-2	Using Resulting Indicators	8-21
Specifying an Alternate Collating Sequence in Hexadecimal Code	3-15	Using the DIV and MVR Operations	8-25
Specifying an Alternate Collating Sequence in Octal Code	3-20	Compare Operations	8-33
Specifying File Translation in Hexadecimal Code	3-28	Testing Bit Status	8-35
Specifying File Translation in Octal Code	3-31	Conditional Branching	8-37
Cross-Reference Listing	3-37	General Subroutine Structure	8-39
Control Record and File Description Specification Sheet	4-2	Using Internal Subroutines	8-41
Descriptions for KSAM Files	4-21	RPG Main Program and Symbol Table Listing	8-45
File Sharing	4-37	SPL Procedure for Address Conversion	8-46
File Extension and Line Counter Specification Sheet	5-2	COBOL Subprogram Called by RPG Main Program	8-46
Specifying Chaining Record Sequence	5-4	MPE Batch-Job Session for Compiling, Preparing, and Executing Programs	8-47
Specifying Chaining Field Code	5-6	Result of Program Execution	8-47
File Extension Specifications for Three Arrays	5-24	Using External Subroutines	8-49
Loading Compile-Time Tables and Arrays from Job Stream	5-26	Indicator and Bit Setting Operations	8-51
Loading Compile-Time Data from Disc File	5-28	Searching Tables and Arrays	8-53
File Extension and Input Specifications for Arrays	5-32	Contents of Arrays After SORTA	8-54
File Extension and Line Counter Specification Sheet	6-4	Using the SORTA Operation	8-55
Input Specification Sheet	7-2	Using the EXCPT Operation	8-58
Specifying a Record Group Sequence	7-5	Using the FORCE Operation	8-59
Input Specifications for User Data Structure	7-7	Using SET Operation	8-60
Typical Flow Using Local Data Area	7-7	Message File Records	8-62
Command Keys on Keyboard	7-11	Sample Array	8-62
Using a General Record Indicator	7-12	Using the DSNLY Operation	8-66
Specifying Look-Ahead Fields	7-14	Reading a Demand File	8-68
Specifying Spread Records	7-17	Chaining to an Input File	8-70
Record Identification Codes	7-21	Creating a Chained File	8-72
AND and OR Lines	7-23	Using the DEBUG Operation	8-81
Stacker Selection	7-25	Using Conversion Routines	8-83
Defining Data Fields	7-34	Using TIME and TIME2	8-86
Breaking Down an Input Field	7-37	Output Specifications	9-2
Grouping Two Separate Fields	7-37	Output Record Types	9-4
Input Specifications for Voter Count Program	7-40	Stacker Selection	9-7
Multi-file Processing	7-43	Fetching Overflow	9-11
Record Sequencing	7-45	Using Output Indicators	9-16
Record Matching	7-47	Using the *PLACE Field	9-23
Chaining to a Direct-Access File	7-48	RSI Form Name Specification	9-31
		Printing and Punching Data	9-33
		Specifying Constants	9-37
		Examples of Edit Words	9-42
		Printer Spacing Chart	9-46
		Sample Program Control Record and File Description Specification Sheet	10-3
		Sample Program File Extension and Line Counter Specification Sheet	10-5

ILLUSTRATIONS (continued)

Title	Page	Title	Page
Sample Program Input Specification Sheet	10-6	Cross-Reference Listing (Indicators Used Portion)	11-19
Sample Program Input Data Deck	10-7	Cross-Reference Listing (Field Names Used Portion)	11-20
Sample Program Calculation Specifications Sheet	10-8	Cross-Reference Listing (File Names Used Portion)	11-21
Sample Program Printer Spacing Chart	10-10	Creating a New Source File by Merging and Editing	12-13
Sample Program Output Specifications Sheet	10-11	Relation of RPG Action Codes of V/3000 File Description Specification for Sample Program	13-8
Batch Job Output	10-13	Input Specifications for Sample Program	13-20
Chaining to a KSAM File	10-21	Calculation Specifications for Sample Program	13-21
Creating a New KSAM File	10-22	Output Specifications for Sample Program	13-23
Specifying Chaining to a KSAM File	10-23	Compile-Time Error Messages	B-2
Specifying Chaining Codes for a KSAM File	10-27	Error Dump	C-4
RAF File Entries for Processing Between Limits	10-32	RPG/3000 Object Program Operating Cycle	D-3
KSAM File Processing by RAF, Between Limits	10-32	Object Program Logic Flow Chart	E-2
KSAM File Processing by RAF, Random Access	10-37	ISAM File Creation Program (Source Code)	H-6
Printing a Report	10-40	ISAM File Creation Program (Output)	H-6
Data-Base Schema	10-42	Limits Program (Source Code)	H-7
Data-Base Create Job Stream	10-42	Limits Program (Record Address File)	H-7
Indexed Update Program	10-43	Limits Program (Output)	H-8
Card-Reader/Punch Application	10-45	SETLL Program (Source Code)	H-9
RPG/3000 Source Program Input Records	11-2	SETLL Program (Output)	H-9
Program Listing	11-16		
Symbol Table Listing	11-18		

TABLES

Title	Page	Title	Page
Inverted Print Option Formats	3-12	Item Name (ITEM) Record Fields	4-42
ALTSEQ Record Description for Hexadecimal Format	3-15	Password (LEVEL) Record Fields	4-43
ALTSEQ Record Description for Octal Format	3-16	Status Array (STATUS) Record Fields	4-44
Sign-Forcing Options	3-22	Keyfile (KEYFL)	4-45
Translation Record Description with Hexadecimal Codes	3-26	WORKSTN Interface Group Fields	4-45
Translation Record Description with Octal Codes	3-26	File Description Specification Default Values	4-47
Carriage Control Initialization	3-36	File Extension Specification Required/Optional/ Prohibited Entries	5-20
RPG-Detected Errors	3-42	File Extension Specification Default Values	5-21
Control Record Specification Default Options	3-44	Table/Array File Name Record Format	5-27
Processing Direct Access Storage Files	4-17	Line Counter Specifications Default Values	6-7
General Group Fields	4-30	Overpunch Conversion Chart	7-29
File-Sharing Group Fields	4-35	Input Specification Default Values	7-57
Relationship of Access Type of KSAM/RSAM Currency Pointers	4-36	Move Operation Results	8-26
Data-Base Name (IMAGE) Record Fields	4-41	Move Zone Operations	8-31
		LOCK/UNLCK Summary	8-76
		LOCK/UNLCK Resulting Indicators	8-77

TABLES (continued)

Title	Page	Title	Page
Summary Table for PUTJW and FINDJW	8-88	Common Errors	B-3
Calculation Specification Default Values	8-89	Control Record (H) Specification Errors	B-5
Printer File Types	9-3	File Description (F) Specification Errors	B-13
Release File Actions	9-9	File Extension (E) Specification Errors	B-26
Complex Edit Code Summary	9-27	Line Counter (L) Specification Errors	B-33
Effects of Edit Codes	9-28	Input Specification (I) Errors	B-35
Output Specifications Default Values	9-44	Calculation Specification (C) Errors	B-50
Functional List of Commands	11-6	Output Specification (O) Errors	B-62
Reference Notes for Command Definitions	11-7	Compiler Subsystem Command Errors	B-71
RPGTEXT Characteristics	11-23	Other Errors	B-73
RPGMAST Characteristics	11-24	RPG-Detected Errors	B-77
RPGNEW Characteristics	11-25	USWITCH Command Errors	B-79
RPGUSL Characteristics	11-26	File Table Format	C-14
RPGLIST Characteristics	11-27	IMAGE/3000 Table Format	C-18
User File Characteristics	11-28	Chaining Table Format	C-19
Compiler Subsystem Command Summary	12-5	Table/Array Control Table Format	C-19
Event Codes	13-5	R'ISAM Type and F Array Parameter Values	I-1
Action Codes	13-6	Result Values Returned by R'ISAM Procedure	I-3
Explanation of Program	14-4		

INTRODUCTION TO RPG/3000

SECTION

I

The HP 3000 Report Program Generator (RPG/3000) helps you create programs to produce reports, update files, and perform many other general file applications. It consists of:

- The RPG/3000 symbolic programming language, used to write source programs that specify your requirements to an HP 3000 Computer.
- The RPG/3000 compiler that translates the source programs into object programs in HP 3000 machine code. These object programs then produce the reports, manipulate the files, or perform the other functions you desire.
- The RPG/3000 library, consisting of run-time procedures for certain functions such as input/output operations, output-field editing, and specialized calculations such as deriving square root. These procedures are invoked by compiler-generated code. (For most operations, however, the compiler generates in-line all code required.) Because the library procedures are furnished for your program automatically, you need not consider them during source coding nor make any special provisions for them.

The RPG/3000 compiler, object programs it generates, and the RPG/3000 library all operate on any HP 3000 Computer under control of the HP 3000 Multiprogramming Executive Operating System (MPE/3000).

1-1. APPLICATIONS

Programmers most often use RPG/3000 in business and commercial applications, although it is not restricted to these areas. In fact, it is flexible enough for almost any file application. It handles a wide range of jobs, from simple ones (such as printing mailing-address labels) to very complex tasks (such as running an entire payroll process, complete with printing paychecks, payroll registers, and various allied reports). RPG/3000 is ideal for producing such documents as inventory lists, billings, invoices, insurance benefit notices, or summaries of sales, profits and losses, and customer transactions. It is also excellent for updating master files used in producing these documents.

1-2. CAPABILITIES AND ADVANTAGES

RPG/3000 can produce object programs to handle almost any business data processing task. Such programs can:

- Process large tables and arrays of data.
- Read data from input files on various devices.
- Perform extensive calculations and store results in new files on discs, tapes, terminals, or punched cards.
- Update large files on magnetic tape or disc.
- Generate multiple reports of varying complexity in a single process.

Introduction

- Process records by random (direct) or sequential access through the HP 3000 Keyed Sequential Access Method (KSAM/3000), RPG Indexed Sequential Access Method (INDEX/3000), or IMAGE/3000 subsystems. (KSAM is not available to programs running under control of MPE-C.)

The RPG/3000 language is a machine-independent, problem-oriented language that is easy to learn, use, and code. It allows you to specify many important operations with a minimum of effort, by making simple entries on specially-formatted coding sheets. Because RPG is a standard language available on many different machines, you can submit programs coded in another manufacturer's RPG directly to the RPG/3000 compiler with little or no re-coding for conversion. (Conversion requirements are noted in Appendix A.) In addition, the RPG/3000 compiler helps you detect errors at the source language level with extensive diagnostic messages.

1-3. NEW FEATURES

The following new features have been added to RPG since the previous edition of this manual:

- Support of indexed-sequential files through the Keyed Sequential Access Method (KSAM/3000) and the INDEX/3000 indexed file processing subsystem. INDEX/3000 is available only to programs running under control of MPE-C. (Prior to its release as a supported product, INDEX had been in use at some sites under the name RSAM or R'ISAM.)
- Partial field translation where translation of packed decimal and binary data fields is suppressed during input/output operations. (This feature is used when converting files from ASCII to EBCDIC code and vice-versa; packed decimal and binary data fields must not be translated because they must have the same data representation in ASCII as in EBCDIC.)
- Locking and unlocking of shared files to control access.
- File-sharing, where a single file can be identified by several different names. (This allows you to bypass the use of MPE :FILE commands for equating file names to one another.)
- Increased compatibility with IBM System/3 RPG II; this includes the setting of Blank-After Indicators controlled by Control Record Specification entries (Column 42); the option to specify that all direct file processing begins with Relative Record No. 1, also requested through Control Record Specification entries (Column 22); and the entry of tables and arrays into the system as part of the source program.
- Locking and unlocking of IMAGE/3000 at the Data Base, Data Set, and Data Record levels.
- A new category of Calculation Specifications operations called System Operations. The first entries are TIME and TIME2 for access to the system time and date.

1-4. USING RPG/3000

In preparing and running any RPG/3000 program, you normally follow the seven steps outlined below and illustrated in Figure 1-1.

Step 1: Analyze the Task.

Carefully consider the data to be input and its format, the calculations and other operations needed, the number and type of totals to be accumulated, and the data to be output and its format. You must also decide what devices to use for your input and output files.

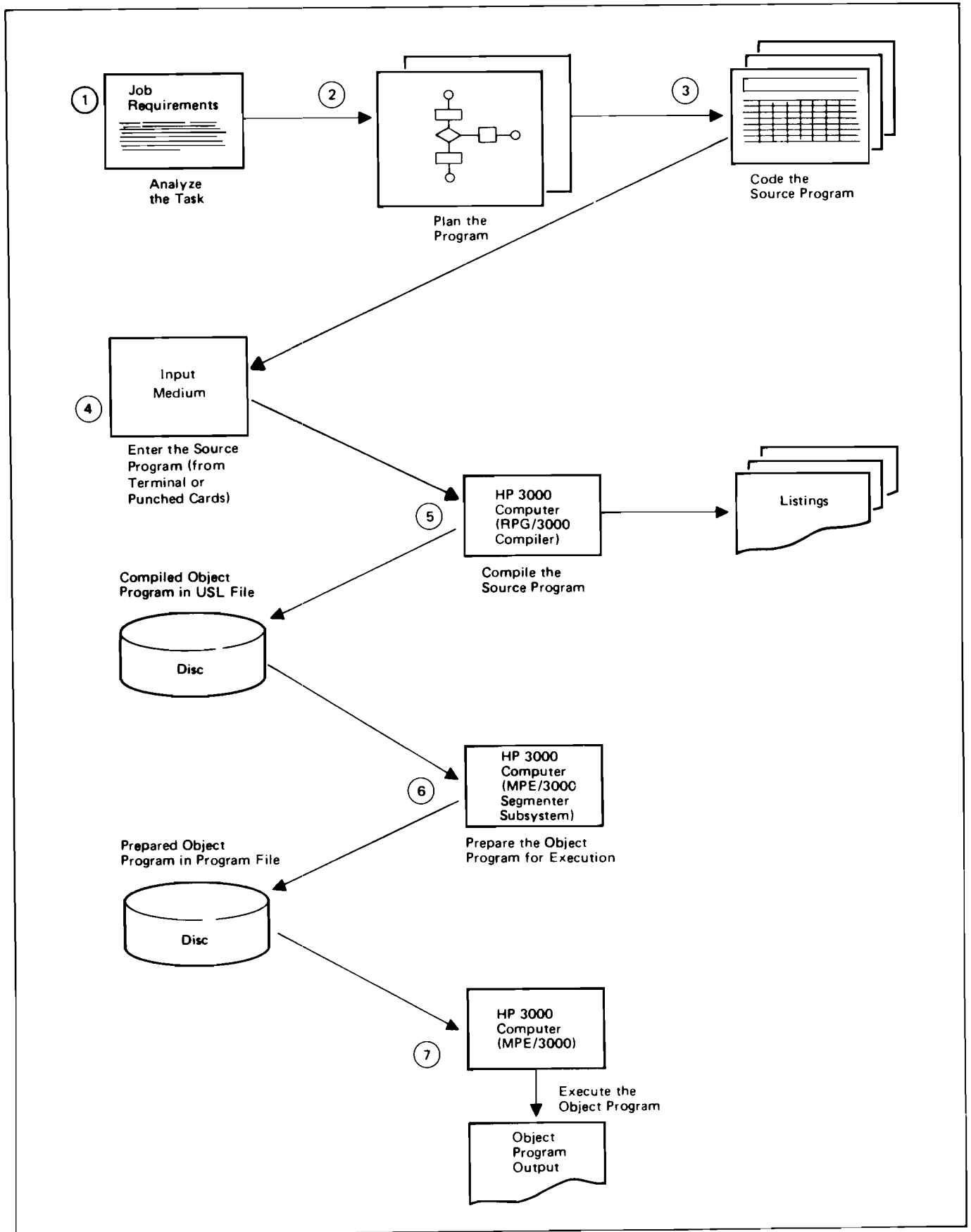


Figure 1-1. Steps for Preparing and Running an RPG/3000 Program

Introduction

Step 2: Plan the Program.

Plan the general steps your object program must execute to produce the desired output. In doing this, you may want to prepare an overall flow-chart of the program.

Step 3: Code the Source Program.

Write the source program from which your object program will be compiled on the RPG/3000 Specifications Sheets. These are five special coding forms, each with a distinct purpose, designed to help you organize information and code it efficiently. These sheets are discussed briefly below, in the order in which you enter their contents into the system; they are described in detail in Sections III through IX. A sample RPG/3000 program, written on these sheets, appears in Figure 1-2. Blank specifications sheets appear in Appendix J.

- The **Control Record and File Description Specifications Sheet** includes two types of specifications. One type is a **Control Record Specification** on which you supply general information about your source program. This includes directions on how to handle certain editing codes, what indicators (switches or flags) to set, whether to print a cross-reference listing showing where all field names, file names, and indicators are referenced in the program, where run-time error messages are to be transmitted, and additional information. The other type is comprised of the **File Description Specifications** you supply to describe the characteristics of all files used by the program — their names and types, the sizes of the logical and physical records they contain, and the formats in which these records are written.
- The **File Extension and Line Counter Specifications Sheet** also includes two types of specifications. In the **File Extension Specifications**, you describe tables and arrays of data and record-address files used by the program; you may also supply input chaining information. (Tables and arrays are groups of related data specially arranged for efficient reference; a record-address file is an input file that specifies which records are to be read from another file on disc, and in which order; chaining refers to a method for direct loading and random access of an input file, as described in Section VII.) In the **Line Counter Specifications** you provide information for line printer files. This information specifies form length and associates line numbers on such files with channel numbers on the printer's carriage control tape. This association allows reports to be stored on disc for later output to a line printer.
- The **Input Specifications Sheet** allows you to identify the types of input records used by the program, and the fields they contain.
- The **Calculations Specifications Sheet** permits you to describe all data manipulation, arithmetic, and branching operations required, and all calls to subroutines. On this sheet, you can also directly control program input/output.
- The **Output Specifications Sheet** enables you to define all output records produced by the object program, and the fields they contain. This data can be printed on reports, punched on cards, or stored on disc or tape. (If preparing a report, you can lay out the fields on a preliminary planning form before writing the output specifications. This form, called a **Printer Spacing Chart**, is shown in Figure 1-3 and explained in Section IX.)

Note: For each RPG/3000 source program, you must include at least one **Control Record and File Description Specifications Sheet**. All other sheets are optional, although without at least one **Calculations Specifications Sheet** or **Output Specifications Sheet** your program will not actually do anything (except possibly begin an infinite looping of the Logic Cycle if the LR indicator or a Halt indicator is not set on in Calculations). The only way to write a program that does not include at least one **Input Specifications Sheet** or **Output Specifications Sheet** is to define all files on the **File Description Specifications Sheet** as **Display type files**.

HEWLETT PACKARD
RPG OUTPUT SPECIFICATIONS
Page 5 of 5

HEWLETT PACKARD
RPG CALCULATION SPECIFICATIONS
Page 4 of 5

HEWLETT PACKARD
RPG INPUT SPECIFICATIONS
Page 3 of 5

HEWLETT PACKARD
RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS
Page 2 of 5

HEWLETT PACKARD
RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS
Page 1 of 5

Author: **B FRANKLIN**
Date: **1/3/75**
Printing Instructions:

Title: **TEXTBOOK SALES**
Program Name: **TEXTSL**

Error Response (0-1, 2-4)										
Error Numbers										
0	1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76
77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98
99										

File Description Specifications

File No.	File Name	File Type	File Format	File Length	File Record Length	File Record Count	File Record Type	File Record Length	File Record Count	File Record Length	File Record Count	File Record Length	File Record Count	File Record Length	File Record Count
0020	CARDS	IP	F	80						CARD					
0030	TABFILE	IT	F	80						EDISC					
0040	REPORT	IO	F	132		OF				LLP					

Order of Specifications:

- Control Record (Header) — one (and only one)**
- File Descriptions — at least one is required**
- File Extensions — optional**
- Line Counter — optional**
- Input — at least one Input or Output Specification is required unless all files are Display-type files**
- Calculations — optional**
- Output — at least one Input or Output Specification is required unless all files are Display-type files**

Figure 1-2. RPG/3000 Source Program

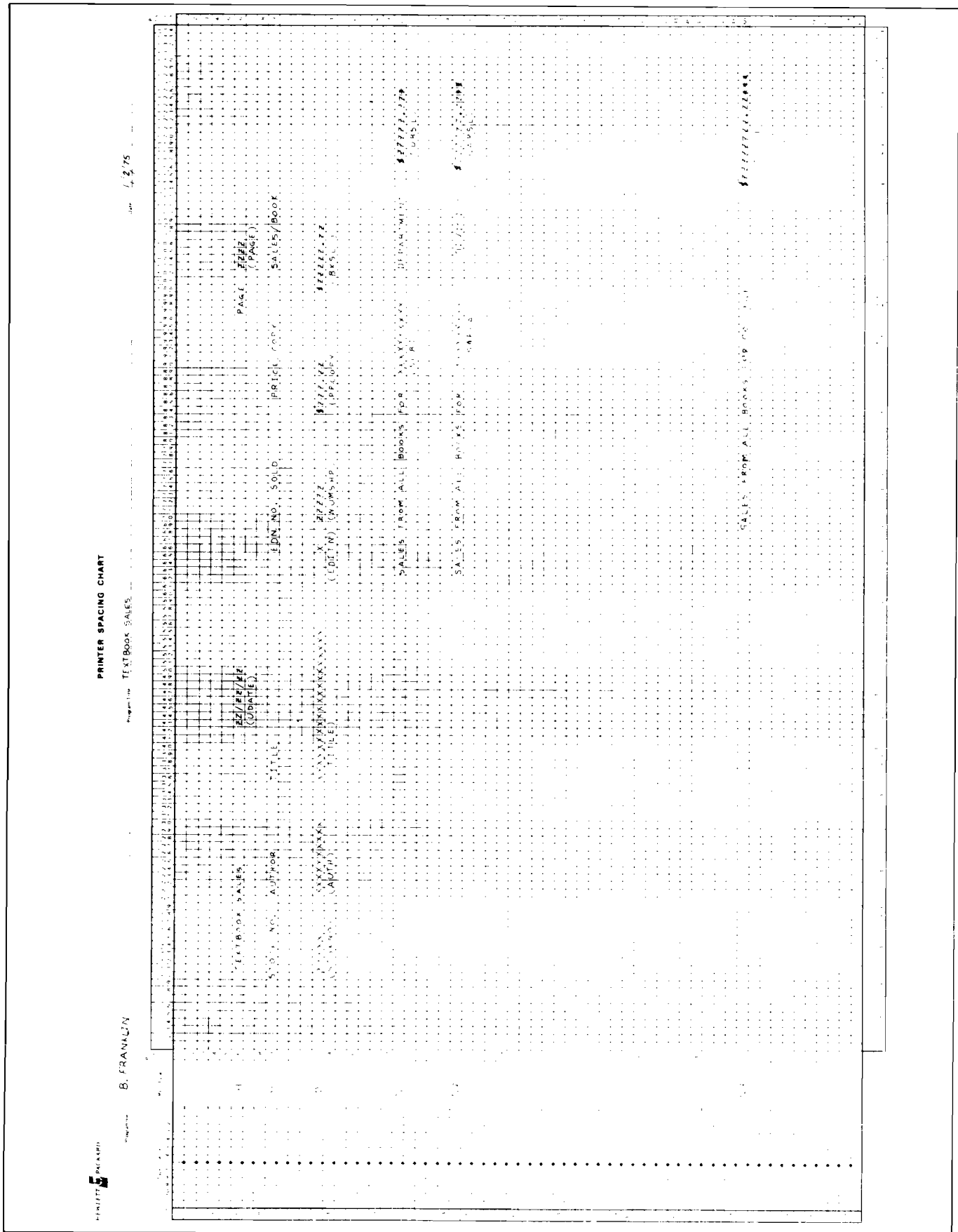


Figure 1-3. Printer Spacing Chart

After coding, it is wise to desk-check all specification sheets for any obvious errors in coding or program logic.

Step 4: Enter the Source Program

For many applications, the RPG/3000 compiler reads source program input from a disc file or card images, generates the object program onto disc, and produces listings on the line printer. Whether entering your source program from a terminal or on punched cards, enter one logical record (card-image or card) for each line on a specification sheet. (You can also submit special instructions to the compiler, called **compiler subsystem commands**, with your program. These commands are described in Section XII.)

Note: RPG/3000 allows input/output on many devices. Specifically, you can input your source program and compiler subsystem commands from cards, terminals, disc, or magnetic tape; you can output listings to printers, terminals, disc, or magnetic tape. (However, you must output your object program to disc only.)

When entering your source program, be sure that the records making up the program are arranged in correct order, as shown in Figure 1-4. As part of the program you must include:

- One (and **only one**) Control (Header) Specification record.
- At least one File Description Specification record.
- At least one Input Specification or Output Specification record (unless all files are Display type).

All other records, such as Line Counter or Calculation Specification records, are optional.

Step 5: Compile the Source Program

Access the computer and compile your program with the MPE/3000 commands discussed in Section XI. The RPG/3000 compiler works in conjunction with MPE/3000 to assign disc storage areas and create routines to handle input/output, and to translate the source program into HP 3000 machine code stored in a user sub-program library (USL) file. (USL files are described in the MPE Reference Manuals.) **At your option**, the compiler provides source program and cross-reference listings. It also outputs compile-time messages concerning any errors encountered during compilation; these are discussed in Appendix B. A source program listing appears in Figure 1-5; a cross-reference listing appears in Figure 1-6.

These listings and messages help you correct any errors detected. If corrections are necessary, compile the program again until all errors are resolved.

Note: MPE allows you to request compilation (Step 5), preparation (Step 6, below), and execution (Step 7, below) as separate, discrete operations using three separate commands; as a collective sequence of operations using a single command; or as several combinations of operations using various other commands.
(See Section XI.)

Step 6: Prepare the Object Program for Execution

The program on the USL file is not directly executable. Instead, you must **prepare** it for execution by linking the module to any other subroutines and procedures required for its support. (These subroutines and procedures, when used, support special files and the RPG/3000 EXIT command, discussed in Section VIII. They are typically written in other languages, such as HP 3000 Systems Programming Language

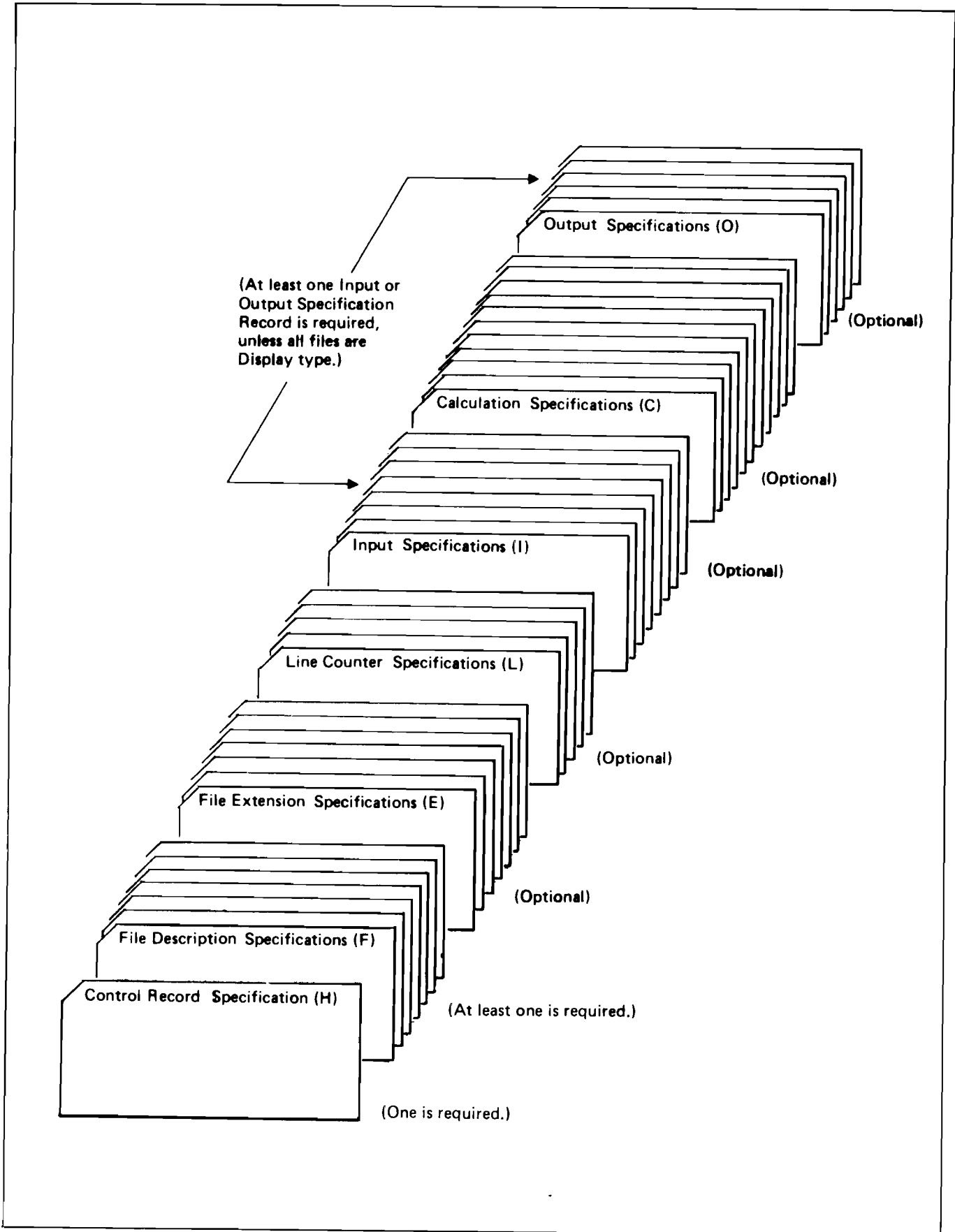


Figure 1-4. RPG/3000 Source Program Input Records

```

PAGE 0001  HEWLETT PACKARD 32104A,P2,03 RPG/3000  THU, JAN  2, 1975,  4:39 PM

0001  0010 M                                XLS                                TEXT5L

0002  0020 FCARDS  IP  F      80          CARD
0003  0030 FTABFILE IT  F      80          EDISC
0004  0040 FREPORT  O  F     132        OF  LLP

0005  0050 E  TABFILE          TABA  B 140 5  ATARB  5 0 ALTERNATING TABLE

0006  0060 LREPORT  66FL 550L

0007  0070 ICARDS  AA  15
0008  0080 I                                1  5 STOKNO
0009  0090 I                                6 15 OSAREAL2
0010  0100 I                                16 25 SUBJ  L1
0011  0200 I                                26 35 AUTH
0012  0300 I                                36 55 TITLE
0013  0400 I                                56 58 EDITN
0014  0500 I                                59 632PPCOPY
0015  0600 I                                64 680NUMSHP

0016  0800 C          PPCOPY  MULT NUMSHP  BKSL  72          SALES/BOOK
0017  0900 C          BKSL  ADD  SUBSL  SURSL  92          SALES/SUBJECT
0018  1000 C          BKSL  ADD  GARSL  GARSL 102         SALES/GEN AREA
0019  1010 C          BKSL  ADD  TOTSL  TOTSL 122         TOTAL SALES
0020  1020 C          STOKNO LOKUPTABA  TABB          10
0021  1030 C  10      TABB  SUR  NUMSHP  TABB          UPDATE INV TAB

0022  1040 OREPORT  H          1P
0023  1050 O                                24 "TEXTBOOK SALES"
0024  1060 O                                UDATE Y  50
0025  1070 O                                PAGE     100 "PAGE"
0026  1080 O                                PAGE     106
0027  1090 O          H  6  OF
0028  1100 O                                PAGE     100 "PAGE"
0029  1110 O                                PAGE     106
0030  1120 O          H 33  1P
0031  1130 O          OR  OF
0032  1140 O                                18 "STOCK NO.,"
0033  1150 O                                26 "AUTHOR"
0034  1160 O                                40 "TITLE"
0035  1170 O                                68 "EDN"
0036  1180 O                                77 "NO. SOLD"
0037  1190 O                                95 "PRICE/COPY"
0038  1200 O                                112 "SALES/BOOK"
0039  1210 O          D 1  15

```

Figure 1-5. RPG/3000 Source Program Listing

Introduction

PAGE 0003	TEXTSL	INDICATORS USED
INDICATOR 10 DEFINED	0020	
REFERENCED	0021	
INDICATOR 15 DEFINED	0007	
REFERENCED	0039	
INDICATOR L1 DEFINED	0010	
REFERENCED	0047	
INDICATOR L2 DEFINED	0009	
REFERENCED	0053	
INDICATOR LR NOT DEFINED		
REFERENCED	0059	
INDICATOR OF DEFINED	0004	
REFERENCED	0027 0031	
INDICATOR 1P NOT DEFINED		
REFERENCED	0022 0030	

PAGE 0004	TEXTSL	FIELD NAMES USED
AUTH (FIELD) 10	0011	
REFERENCED	0041	
BKSL (FIELD) 7.2	0016	
REFERENCED	0017 0018 0019 0046	
EDITN (FIELD) 3	0013	
REFERENCED	0043	
GARSL (FIELD) 10.2	0018	
REFERENCED	0057	
OSAREA (FIELD) 10	0009	
REFERENCED	0055	
NUMSWP (FIELD) 5.0	0015	

PAGE 0006	TEXTSL	FILE NAMES USED
CARDS	0002	
REFERENCED	0007	
REPORT	0004	
REFERENCED	0006 0022	
TABFILE	0003	
REFERENCED	0005	

Figure 1-6. RPG/3000 Cross-Reference Listing

(SPL/3000) or COBOL/3000.) Prepare the program by entering commands directed to the MPE/3000 **Segmenter** subsystem, as discussed in Section XI. When preparation is complete, the object code and subroutine and procedure linkages reside in an object program file on disc. (Program files are described in the MPE Reference Manuals.)

Step 7: Execute the Object Program

Request execution through the MPE/3000 commands noted in Section XI. In response, MPE/3000 links the code in the program file to the RPG/3000 library, and the object program begins execution. The object program processes your input, performs calculations, merges and manipulates information, and produces reports and other output files requested. At this time, diagnostic messages again appear if program errors are encountered. These run-time messages, like compile-time messages, are discussed in Appendix B. Techniques for debugging the object program appear in Appendix C.

Your object program can use various devices on the system for input/output, as you desire. (MPE allows you to specify these devices at program-execution time, without requiring changes to the source or object program.) An example of object program output, a report written to a line printer, appears in Figure 1-7.

Every RPG/3000 program executes on the same general cycle, from the time it reads the first input record until it writes the last line of output. Although you do not actually observe this cycle, you can write and debug your programs more effectively if you understand the cycle and the general logic it involves. From a broad standpoint, the cycle entails reading an input record, operating on the data it supplies, and writing the results to an output device, in that order — and then repeating this sequence until the last record is processed. As an example, a program that prints paychecks for a company's employees would read the first employee's time card, compute his earnings by multiplying the hours he worked by his pay rate and taking various deductions and withholdings, print his paycheck, and then read the next employee's time card. The details of the object program operating cycle are described in Appendix D. A complete understanding of these details depends on knowledge of terms defined in Sections II through X. Thus, if you have no previous experience with RPG, you should read these sections before turning to Appendix D. Complex programs require you to understand the object program operating cycle even more thoroughly. Before writing such programs, please see the detailed RPG object program logic flowchart in Appendix E.

1-5. COMPATIBILITY WITH OTHER SYSTEMS

The RPG/3000 compiler also processes programs written in IBM System/3 RPG II, IBM System/360 Disk Operating System (DOS) RPG II, or any RPG compatible with these languages. A complete list of extensions and minor functional differences appears in Appendix A.

1-6. SYSTEM REQUIREMENTS

The RPG/3000 compiler and library can be installed as disc-resident systems and operated on any HP 3000 computer run under control of the HP 3000 Multiprogramming Executive Operating System (MPE/3000). For compilation of large programs, RPG/3000 may require additional main memory up to a maximum of 131,072 bytes.

The object programs generated by RPG/3000 will support any input/output devices supported by MPE/3000. These include card readers, card punches, line printers, and terminals. On Pre-Series II Computer Systems, object program execution time can be upgraded by using a machine with the **Extended Instruction Set Decimal Arithmetic Option**.

Introduction

TEXTBOOK SALES		1/02/75	PAGE 0001		
STOCK NO.	AUTHOR	TITLE	EDM NO, SOLD	PRICE/COPY	SALES/BOOK
00001	WAKEFIELD	PRINCIPLES OF ACCT	1 00010	\$8.75	\$87.50
00002	SMITH	ACCOUNTING FOR MGT	1 00005	\$5.75	\$28.75
SALES FROM ALL BOOKS FOR ACCOUNTING DEPARTMENT					\$116.25*
00003	WOODRY	MANAGEMENT BY OBJECT	3 00007	\$9.00	\$63.00
SALES FROM ALL BOOKS FOR ADMINISTRATION DEPARTMENT					\$63.00*
00007	SIGMA	APPLIED STAT FOR BUS	2 00010	\$8.50	\$85.00
00008	OLCOTT	BAYESIAN STATISTICS	1 00011	\$8.75	\$96.25
SALES FROM ALL BOOKS FOR STATISTICS DEPARTMENT					\$181.25*
SALES FROM ALL BOOKS FOR BUSINESS DIVISION					\$360.50**
00020	RAPHAEL	ROMANTICISM IN ART	1 00015	\$15.00	\$225.00
00021	RAND	FIGURE DRAWING	1 00021	\$11.00	\$231.00
SALES FROM ALL BOOKS FOR ART DEPARTMENT					\$456.00**

00051	MESSICK	CIRCUITS	2 00015	\$11.00	\$165.00
00057	MESSICK	BASIC ELECTRONICS	1 00020	\$6.00	\$120.00
SALES FROM ALL BOOKS FOR ELEC ENG DEPARTMENT					\$285.00*
00060	GRUNDY	MODERN MATH IDEAS	2 00003	\$8.00	\$24.00
SALES FROM ALL BOOKS FOR MATH DEPARTMENT					\$24.00*
00070	BROWN	MODERN PHYSICS	1 00018	\$15.00	\$270.00
00072	DARWELL	NUCLEAR PHYSICS	1 00005	\$7.50	\$37.50
SALES FROM ALL BOOKS FOR PHYSICS DEPARTMENT					\$307.50*
SALES FROM ALL BOOKS FOR SCI & TECH DIVISION					\$616.50**
SALES FROM ALL BOOKS FOR COLLEGE					\$1,734.50***

Figure 1-7. RPG/3000 Object Program Output (Report)

COMMON ENTRIES

SECTION

II

Before writing an RPG/3000 source program, you first must learn the rules for making entries on the RPG/3000 Specifications Sheets. Rules that apply to all specification sheets appear in this section; rules unique to individual sheets are discussed in Sections III through IX. After you have learned the rules from the details presented in these sections, you still may want to refresh your memory when coding. The specification sheet summaries in Appendix F are ideal for this purpose.

2-1. GENERAL INFORMATION

All RPG/3000 Specification Sheets contain spaces for entering the page number of the current sheet, the total number of sheets making up your program, your name, the title of your program, and the current date.

EXAMPLE

In the Control Record and File Descriptions Specifications Sheet in Figure 2-1, examples of these entries appear as the circled item numbers 1 through 5 respectively. This is the first page (Item 1) of the five-sheet program (Item 2), written by G. Boole (Item 3). He has entitled the program PAYRUN (Item 4) and dated the sheet November 2, 1974 (Item 5).

In addition, all specification sheets include an area for special keypunching instructions. In this area, you indicate what combinations of punches should be used to represent characters that do not appear on the keypunch or characters that correspond to different keys than those indicated on the keyboard.

EXAMPLE

To request the keypunch operator to represent the percent sign (%) by a 0-4-8 punch, make the entry shown in Figure 2-1, Item 6.

2-2. CODING LINES

Next, the area in which you code your RPG program appears. On RPG Specifications Sheets, as on most coding forms, each line of code contains several fields, each relating to a particular type of information. Entries in these fields can describe an operation (such as addition, multiplication, or movement of data); an operand (data to be operated upon, such as an employee's tax status or the number of hours he worked during a pay period); the name of the file containing this data; the type of device on which the file resides; and many other kinds of information.

On each specification sheet, the name identifying each field appears above that field. For example, note field names Sequence Number, Form Type, and Debug in the Control Record portion of the Control Record and File Description Specifications Sheet in Figure 2-1. The shaded fields on the specification sheets are not used in RPG/3000, although they may be required or supported by other RPG's; if entries appear in these fields, the compiler issues warning messages but takes no other action. (All entries consist of characters selected from the American Standard Code for Information Interchange (ASCII) Character Set and Collating Sequence shown in Appendix G.)

While most fields appear on only one or two particular specification sheets, four are common to all sheets: the Sequence Number, Form Type, Comment Record Indicator, and Program Name Fields. The rules covering these fields follow.

Note: In this manual, the notation **(Required)** following a field name indicates that you must make an entry in that field.

2-3. Sequence Number (Columns 1 through 5)

In this field, you optionally can assign a unique sequence number to each line of code (record) in your source program. This compiler allows the following entries:

Columns 1 through 5 Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

During compilation, sequence numbers allow the compiler to check the source records for proper order. If you submit your source program on cards, these numbers help anyone handling the card deck to keep it in proper order. To aid in debugging, they also appear on the Source Program Listing.

You may omit leading zeros from sequence numbers of less than five digits. Within any sequence number, RPG/3000 treats leading, trailing, or embedded blanks as zeros.

Write sequence numbers in ascending numerical order (although they need not be consecutive) — otherwise, an error will occur during sequence checking. If you leave a Sequence Number Field entirely blank, the record on which it appears will not be checked for proper sequence. If you use sequence numbers but do not want them checked, specify N in the Textfile Sequence Check Field (Column 54) of the Control Record Specification or leave that field blank, as discussed in Section III.

Some programmers treat the Sequence Number Field so that Columns 1 through 2 indicate the page number of the specification sheet, Columns 3 through 4 designate the line number of the current entry, and Column 5 specifies an inserted record. This facilitates keeping the sheets as well as individual records in order. In such a case, the control record would have page number 01 and line number 01; its overall sequence number would be 0101 (implying 01010). A record inserted between Records 0101 and 0102 could have the overall sequence number 01011.

Common Entries

In the examples in this manual, the first four digits in the Sequence Number Field reflect the sequence of the record with no regard to specification sheet page number; the fifth digit, when present, indicates an inserted record.

EXAMPLE

You could indicate the sequence number of the first four records in your program as shown in Figure 2-1, Item 7.

2-4. Form Type (Column 6) (Required)

This field denotes the type of specification you are currently entering, as follows:

Column 6 Entry	Meaning
H	Control Record (Header) Specification
F	File Description Specifications
E	File Extension Specifications
L	Line Counter Specifications
I	Input Specifications
C	Calculation Specifications
O	Output Specifications

The entry must appear in all specifications (as indicated by the notation (Required) in the heading above), and is pre-printed on the specification sheets for convenience.

Note: Comment records (defined next), which are not truly RPG/3000 specifications, do not require a Form Type entry.

EXAMPLE

H and F form type entries appear in the Control Record and File Description Specifications, respectively, in Figure 2-1, Items 8 and 9.

2-5. Comment Record Indicator (Column 6 or 7)

An asterisk in Column 6 or 7 of any record indicates that the remainder of the record contains comments or notations that are not part of the source program.

Column 6 or 7 Entry	Meaning
*	Comment record.

Comments do not affect compilation or object program execution, but they do appear in the Source Program Listing. Programmers typically use comment records to document the purpose of coding or program logic, and to aid in checking and debugging programs.

Although a comment record does not contain RPG/3000 specifications, it still can include a sequence number in Columns 1 through 5 and is still subject to the sequence-checking rules applying to all specification records.

If you enter an asterisk in Column 7, you need not enter one in Column 6. Furthermore, an asterisk in Column 7 always indicates a comment record, even if a legal Form Type entry appears in Column 6.

The comment itself can appear in any of the remaining columns through Column 80, and can include any character from the ASCII character set.

EXAMPLE

A comment record indicating the end of the File Description Specifications for the program appears in Figure 2-1, Item 10. Note that the asterisk overrides the Form Type F.

2-6. Program Name (Columns 75 through 80)

In the Control Record Specification, this field assigns a name to your program so that it can be uniquely identified by the system; in other specifications, this field can contain any information you desire. The Program Name Field appears only once on each specification sheet (in the upper-right portion); this allows you to specify, with only one entry, information to appear on every record coded on the sheet.

For the Control Record Specification, these entries are allowed:

Columns 75 through 80 Entry	Meaning
Valid program name. (Program names can contain up to six characters, beginning with one of the letters A through Z. The remaining characters can be letters, any of the digits 0 through 9 or an apostrophe ('). Embedded blanks are not allowed in the name.)	Name assigned to source program, also appearing on Source Program Listing.
blank	Name RPGOBJ is assigned to object program, and also appears on Source Program Listing.

Alternatively, you may assign the source program name using a \$CONTROL NAME= entry, which also would allow you to use shorter source records – 74 bytes long instead of 80 – resulting in a 7.5% savings in storage space required. See paragraph 12-7 for a description of this option.

Common Entries

For all other specifications, the following entries are permitted:

Columns 75 through 80 Entry	Meaning
Any characters.	Name identifying program or part of program on Source Program Listing, or any other meaning desired.
blank	None.

When entered in a Control Record Specification, the name specifically applies to the relocatable binary module (RBM) containing the object program outer block stored on the user subprogram library (USL) file. (For further information about RBM s and outer blocks, please see the **MPE Reference Manual**.) If you omit this entry, the compiler assigns the name **RPGOBJ**. The name assigned also appears on the Source Program Listing.

In any other specifications, you can enter the same program name assigned in the Control Record Specification in this field. Alternatively, you can enter any other information you desire — for example, a name that describes a part of the program or a block of code, or a brief comment or notation. This entry appears on the Source Program Listing but has no meaning to the system.

EXAMPLE

To assign the name **PAYRUN** to the object program, enter the name in the Program Name Field as shown in Figure 2-1, Item 11.

CONTROL RECORD SPECIFICATION

SECTION

III

Every RPG/3000 source program must begin with a Control Record Specification. With this specification, you can select various options that satisfy the general requirements of your program by directing the actions of the RPG/3000 compiler. Among other functions, these options allow you to:

- Suppress debugging operations requested by DEBUG commands in Calculation Specifications. (See Section VIII.)
- Use a collating sequence other than the ASCII Collating Sequence for certain comparison operations.
- Force data in numeric input fields to be converted to signed values when output.
- Set various input and calculation flags, called indicators.
- Translate input files from one code into another (for instance, EBCDIC to ASCII) prior to processing.
- Print a Cross-Reference Listing.
- Check source records for proper sequence.

Write the Control Record Specification on the Control Record and File Description Specifications Sheet, which appears in Figure 3-1. Because each source program requires only one Control Record Specification, the specification sheet provides a single line for this purpose. The shaded fields on this line are not used in RPG/3000.

Note: The notation **Required** following a field name in this manual indicates that the field requires an entry.

3-1. FIELDS.

The RPG/3000 Control Record Specification contains the fields described below. The discussion of each field introduced in this section includes an example keyed to one of the item numbers circled in Figure 3-1.

3-2. Sequence Number (Columns 1 through 5).

This field contains the source record sequence number, described in Paragraph 2-3.

3-3. Form Type (Column 6) (Required).

This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
H	Control Record (Header) Specification.

H Control Record Specification

HEWLETT PACKARD

RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS

Page of

Programmer G. BOOLE Date 1/3/75

Program Title TEXTSK

Punching instructions

Graphic	
Punch	

75 76 77 78 79 80

Program Name TEXTSK

Control Record Specification

Sequence Number	Form Type (H)	Line Number Option (N)	Inverted Print (I, J, D)	Print Record No. (I)	A, Z, Copying Spec. (S, O, E)	Table Array Look Up (B)	Sign Process (S, N, I, O, B)	Form Positioning (I)	Indicator Setting (S, B, T)	File Translation (F, O)	Library Substructure	Sub-address (S)	Cross Reference Using (X)	Carriage Control Type (U)	Textile Sequence Check (S, N)	Error Log (N, S)	Error Numbers
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7

Diagram showing callouts 1 through 18 pointing to specific fields in the Control Record Specification table.

File Description Specifications

Sequence Number	Form Type (F)	File Name	File Type (I, O, U, D, C)	Designation (P, S, R, C, T, D)	End of File (E)	Input Sequence Check (A, D)	Record Format (F, V)	Block Length	Logical Record Length	Processing Mode (R, L)	Key Field Record Address	Field Length	Record Address Type (A, P, K, I)	File Org/Addr (O, U, X, S, W, D, T, I, J)	Overflow Ind (O, A, O, G, O, V)	Key Field Starting Location (0-9999)	Extension Code (E, L)	Device Class Name	Disk Labels Cont. (S, E, K, 2, B)	Name of Label Exit Option Type (EBCDIC to ASCII File)	Option Target	File Addition (A)	Extent (1-32)	File Cond. (U, I, U, B)
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	

Figure 3-1. Control Record and File Description Specifications Sheet

Note: You must submit a specifications record with an H in Column 6 at the beginning of every RPG/3000 program, even if all other fields in this record are blank.

This entry is the only one required in the Control Record Specification. Because it is always used, the entry is pre-printed on the specification sheet for your convenience (Figure 3-1, Item 1).

3-3A. Error Dump Filename (Columns 7 through 14)

A filename placed in columns 7 through 14 specifies the file to which the error dump will be directed. If no filename is specified, or if the filename that is specified does not exist, the error dump is directed to the standard list device (\$STDLIST).

3-4. Debug (Column 15).

In this field, you determine whether or not debugging operations specified within your program are actually executed. The entries allowed are:

Column 15 Entry	Meaning
1	Execute all DEBUG operations; as explained below.
blank	Suppress all DEBUG operations; the compiler will treat records requesting these operations as comment records.

Through the RPG/3000 DEBUG operation in the Calculation Specifications (discussed in Section VIII), you can request listing of various indicator (flag) settings at different points during object program execution; this information helps you debug and correct a program that is not working properly. Once you correct the program, however, you may want to execute it without listing this information. You can enable or disable all requests for the debugging operation without changing the Calculation Specifications by making the appropriate entry in the Debug Field in the Control Record Specification. This is particularly efficient where a program contains many DEBUG requests, all of which can be enabled or disabled by this single entry in the Control Record Specification.

EXAMPLE

To execute all DEBUG operations in your program, enter 1 in the Debug Field (Figure 3-1, Item 2).

H Control Record Specification

3-4A. USWITCH Source (Column 16)

In this field, you determine the source of initialization values for all User Indicators (U1 through U8) used by your program. The entries allowed are:

Column 16 Entry	Meaning
F	Initialize User Indicators from USWITCH records read from the file whose formal designator is USWITCH (see "USWITCH Record Mode").
J	Initialize User Indicators from the Job/Session Job Control Word "JCW" (See "JCW Mode").
blank	If running in a batch job, initialize User Indicators from USWITCH records read from the job stream — \$STDIN (See "USWITCH Record Mode"). If running in an interactive session, initialize User Indicators by prompting the User (on \$STDIN) for initial settings (see "Interactive Mode").

If you specify any of the User Indicators U1 through U8 anywhere in your program, you must indicate how they are to be initialized. The initializing will be done at the beginning of execution of your object program in one of the following ways, depending on the contents of Header column 16 and on whether your program is run in a batch job or an interactive session: USWITCH Record Mode, Interactive Mode, or JCW Mode.

- **USWITCH Record Mode:** If Header column 16 contains "F", USWITCH records are read from the file whose formal designator is USWITCH. If Header column 16 contains a blank and your program is run in a batch job, USWITCH records are read from the batch job stream immediately following the :RPGGO, :PREPRUN, or :RUN command that requests program execution. In either case, USWITCH records can be written in either of two formats — Long Format or Short Format:

LONG FORMAT USWITCH RECORDS

USWITCH: [Ui=resp] ,[Ui=resp] . . . [Ui=resp]

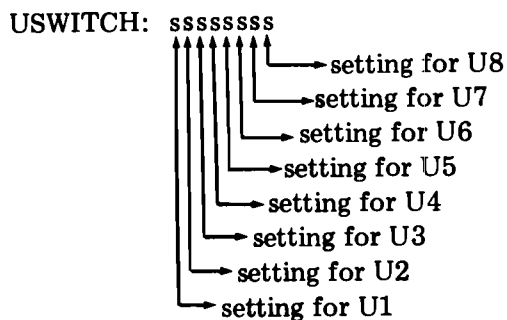
i = Any User Indicator number, 1 through 8.

resp = ON — to set the specified indicator ON.
OFF — to set the specified indicator OFF.
JCW — to set the specified indicator to the value previously established for it in the Job Control Word "JCW" (See "Use of JCW").

The brackets indicate that the parameters are all optional; do not write them as part of the record. Begin the USWITCH: entry in Column 1 and end it in Column 8; otherwise, your program will abort. The U_i =resp parameters can appear in any order (for instance, $U_3=ON,U_4=ON,U_1=ON . . .$). If you omit a U_i =resp parameter for any of the eight indicators, that indicator is set OFF. If you enter U_i = but omit resp, or if you use an entry other than ON, OFF, or JCW for resp, the corresponding indicator is set OFF. A redundant U_i =resp parameter overrides the preceding corresponding parameter. (For instance, $U_1=ON,U_3=OFF, U_1=OFF$ is equivalent to $U_1=OFF,U_3=OFF$.)

You can write Long Format records in free-format; that is, any number of blanks can appear between the USWITCH: entry and the parameter list, or between individual parameters and/or sub-parameters. You can continue the parameter list onto one or more additional USWITCH records, without any special continuation code or character. You cannot, however, divide a parameter such as U_1 or resp between two records — no parameter is allowed to span more than one record.

SHORT FORMAT USWITCH RECORDS



- s = 1 — to set the User Indicator ON
 0 — to set the User Indicator OFF
 X — to set the User Indicator to the value previously established for it in the Job Control Word “JCW” (See “Use of JCW”).

This format allows entry of USWITCH settings in a concise form that is compatible with other RPG compilers. User Indicator settings are coded in an 8-character string in which each character represents one User Indicator (U_1-U_8). While any number of blanks can appear between the USWITCH: entry and the 8-character string, no blanks can be imbedded within the 8-character string.

When this format is used, only one USWITCH record can be specified.

ADDITIONAL RULES FOR USWITCH RECORD MODE

For each execution of a program you can specify either:

- one or more Long Format records, or
- one (and only one) Short Format record

H Control Record Specification

If USWITCH records are Long Format records and are being read from a batch job stream, you must follow the last USWITCH record with a record that contains a double asterisk (**) in Columns 1 and 2. A Short Format USWITCH record read from a job stream SHOULD NOT be followed by a (**) record. If the USWITCH records are being read from a file instead of a job stream, the (**) record is optional. If present, any data following the (**) record will be ignored and can be used by you for comments.

If your program is run in a batch job, the initialized settings of all User Indicators will be printed on the job stream listing immediately after the :RPGGO, :PREPRUN, or :RUN command used to execute your program.

In USWITCH Record Mode, various error messages may follow the attempted execution of a USWITCH command. These messages are discussed in Appendix B.

EXAMPLES

In a batch job, initialize User Indicators using three different methods — Long Format records in the job stream, a Short Format record in the job stream, and Long Format records in a USWITCH file.

!JOB USER.ACCT	(Begin job)
.	
.	
.	
!RUN PROG1	(Execute PROG1: Header col-16 = blank)
USWITCH: U1=ON,U2=JCW	(Set U1 ON and U2 to previous setting)
USWITCH: U8=ON	(Set U8 ON; U3-U7 default to OFF)
**	(Terminate reading of USWITCH records)
!RUN PROG2	(Execute PROG2: Header col-16 = blank)
USWITCH: 1X000001	(Set U1,U8 ON and U2 to PROG1 setting)
	(No ** record after Short Format record)
!FILE USWITCH=SWITCH2	(USWITCH recs read from file SWITCH2)
!RUN PROG3	(Execute PROG3: Header col-16 = "F")
	(No USWITCH recs read from job stream)
!EOJ	(Terminate job)

In an interactive session, initialize User Indicators from Long Format records in a USWITCH file.

```

:HELLO USER.ACCT      (Begin session)
.
.
:FILE USWITCH=SWITCH2 (USWITCH recs read from file SWITCH2)
:RUN PROG4            (Execute PROG4: Header col-16 = "F")
                     (No prompting for USWITCH values)
.
.
:BYE                  (End session)

```

Example contents of file SWITCH2:

```

USWITCH: U1=ON
USWITCH: U2=JCW
USWITCH: U8=ON
**

```

Definition of SWITCH2 initializations:

U1=ON ==> Process current, not historical, data files

U2=JCW ==> Print detailed report only if previously run program
says it's OK (U2=ON)

U8=ON ==> Print summary totals page

} Comments

- **Interactive Mode.** If Header column 16 contains a blank and your program is run in an interactive session, you will be prompted for the initial User Indicator settings immediately after you enter the :RPGGO, :PREPRUN, or :RUN command used to execute your program. The prompt appears on the terminal as follows:

PLEASE INPUT USER SWITCH Ui=?

As in USWITCH Record Mode (Long Format), *i* is the User Indicator number. Prompting is repeated until the settings for all User Indicators specified have been entered. You must respond to each prompt with ON, OFF, JCW or a carriage return (which indicates OFF). Terminate each response by entering a carriage return. If you enter a response other than ON, OFF, or JCW, the indicator is set OFF. In any case, only the first three characters of any response are read. After you reply to the last prompt, your program begins execution.

See "Use of JCW" for further discussion on using "JCW" for initialization. No error messages are output for USWITCH initialization in Interactive Mode.

EXAMPLE

Set User Indicators U1 and U2 ON by responding interactively to the prompts shown in the following session:

H Control Record Specification

```
:HELLO USER.ACCT          (Begin session)
.
.
.
:RUN PROG5                (Execute PROG5: )
                           (Header col-16 = blank)
PLEASE INPUT USER SWITCH U1=? ON (Set User Indicator U1 ON)
PLEASE INPUT USER SWITCH U2=? ON (Set User Indicator U2 ON)
.
.
.
:BYE                      (End session)
```

- **JCW Mode.** If Header column 16 contains “J”, User Indicator settings are taken directly from the Job Control Word “JCW”. If your program is run in a batch job, no USWITCH records will be read from the job stream, and if your program is run in an interactive session, there will be no prompting of the user for initial User Indicator settings.

If your program is run in a batch job, the initialized settings of all User Indicators will be printed on the job stream listing immediately after the :RPGGO, :PREPRUN, or :RUN command used to execute your program.

See “Use of JCW” for further discussion on using “JCW” for initialization. No error messages are output for USWITCH initialization in JCW Mode.

EXAMPLE

Set User Indicators U1, U2, and U8 ON by setting ON Bits 8, 9, and 15 in the Job Control Word “JCW”. Then, pass all User Indicator settings from PROG6 to PROG7.

```
:HELLO USER.ACCT          (Begin session)
.
.
.
:SETJCW JCW = %301        (JCW = %301 = Binary [11 000 001])
:RUN PROG6                (Execute PROG6: Header col-16 = “J”)
                           (No prompting for USWITCH values)
:RUN PROG7                (Execute PROG7: Header col-16 = “J”)
                           (No prompting for USWITCH values)
.
.
.
:BYE                      (End session)
```

Since the 16-Bit pattern corresponding to (Octal) 301 is [00000000 11000001], User Indicators U1, U2, and U8 will be initialized ON by PROG6. The final settings of all User Indicators used by PROG6 will be used for initialization in PROG7.

- **Use of JCW.** You can use "JCW" to initialize User Indicators if your job/session contains a previous program in which User Indicator settings were established, or if you use the :SETJCW command, AND if you use the :RUN command (not :RPGGO or :PREPRUN) to execute your program. If the previous program was an RPG/3000 program, specifying use of JCW is sufficient to pass the User Indicator settings from the previous program to the current one. But, if the previous program was written in another language, that program must store the User Indicator settings in Bits 8 through 15 of the Job Control Word. (RPG/3000 programs implicitly store the User Indicator settings specified by your program into the Job Control Word, and, at the start of each program, check the job control word for settings established by previous programs.)

In the Job Control Word, Bits 8 through 15 indicate the settings for User Indicators 1 through 8, respectively. That is, if Bit 8 is ON, User Indicator 1 is ON; and if Bit 9 is ON, User indicator 2 is ON; and so forth. These "JCW" Bits can be set by a previously executed program or by the :SETJCW command (see example under "JCW Mode").

3-4B UDATE SOURCE (Column 17)

In this field, you determine the source of initialization values for the UDATE, UDAY, UMONTH, and UYEAR special fields. The entries allowed are:

Column 17 Entry	Meaning
F	Initialize UDATE, UDAY, UMONTH, and UYEAR from a date record read from the file whose formal designator is RPGUDATE.
blank	Initialize UDATE, UDAY, UMONTH, and UYEAR from the system date.



- **Date Records.** If Header column 17 contains "F", initial date settings are taken from a Date Record instead of from the system date.

Use an MPE FILE command, prior to running your program, to equate the formal file designator RPGUDATE to the actual file that holds your data record. Equating RPGUDATE to \$STDIN allows you to enter the date record interactively at your terminal or to imbed it in your job stream after the :RPGGO, :PREPRUN or :RUN command used to execute your program (and after USWITCH records, if any). This usage in a job stream allows you to simulate the IBM//DATE OCL command. For example:

```
!FILE RPGUDATE = $STDIN
!RUN MYPROG
USWITCH: U1 = ON
**
//DATE 041080
```

In the date record, the first numeric character indicates the beginning of the date values (this does not have to be in column 1). The date may be entered on the date record in either of the following formats:

H Control Record Specification

mmddy — mm is month (01 to 12)
— dd is day (01 to 31)
— yy is year (00 to 99)
All six digits are required

mm#dd#yy — # represents a separator consisting of one or more non-numeric characters. In this format, leading zeros may be omitted.

If mm, dd, and yy are all equal to zero, or the formal file RPGUPDATE is equated to \$NULL, then RPG initializes the date from the system date. This feature allows you to default back to the system date without removing the "F" from column 17 and recompiling your program.

Assuming the system date is 4/10/80, all of the following examples of date records will cause UPDATE to be initialized to 041080 (shaded digits are those which RPG uses to determine the date — all other characters on the record are ignored):

000000==>DEFAULT TO SYSTEM DATE. (041080)
041080
//DATE 041080 IBM FORMAT
DATE IS 041080 = APRIL 10, 1980
FORMAT OF DATE IS MMDDYY = 041080
4/10/80
MONTH IS 4; DAY IS 10; YEAR IS 80;
0 0 0==>DEFAULT TO SYSTEM DATE; (041080)

The following are examples of invalid date records:

4/10 (year is missing)
4/10/1980 (year must be 0 to 99)
040080 (day must be 01 to 31)

If the date file does not exist, does not contain records, or does not contain a correctly formatted data record, then your RPG program will abort immediately with one of the following error messages:

"UNABLE TO OPEN FILE RPGUPDATE TO OBTAIN DATE."
 "UNABLE TO READ DATE RECORD IN FILE RPGUPDATE."
 "INVALID DATE RECORD IN FILE RPGUPDATE."
 "INVALID MONTH ON DATE RECORD IN RPGUPDATE."
 "INVALID DAY ON DATE RECORD IN FILE RPGUPDATE."
 "INVALID YEAR ON DATE RECORD IN RPGUPDATE."

3-4C. (Column 18 is not used.)

3-4D. (Column 19 is not used.)

3-4E Line Number Option (Column 20)

In this field, you determine whether or not the compiler returns a statement number containing a run-time error. The entries allowed are:

Column 20 Entry	Meaning
N	Disable line number option.
blank	Enable line number option.

When the line number option is enabled, the RPG compiler stores the statement number it is executing. When a run-time error occurs, the compiler can then display the line number containing the error. Faster execution of large programs will occur when the line number option is disabled.

EXAMPLE

To enable the line number option in your program, leave column 20 blank in the control record specification (Figure 3-1, Item 2A).

3-5. Inverted Print (Column 21)

This field determines the format and punctuation for numeric literals, numeric data controlled by edit codes, and dates supplied by the RPG feature UDATE:

Column 21 Entry	Meaning
blank	Domestic Format.
I	European Format.
J	European Format with leading zero.
D	United Kingdom Format.

H Control Record Specification

Numeric literals are numbers (constant values) specified directly in Calculation Specification operations, discussed in Section VIII. Edit codes are entries requesting special punctuation, made in output specifications, discussed in Section IX. UDATE is a special field name, also referenced in Output Specifications, that makes the current date available.

The formats specified result in the following action (when used in combination with the Y edit code discussed in Section IX).

- **Domestic (United States) Format:** periods are used as decimal points in numeric literals and edited data; thousand-units are separated by commas; all leading zeros preceding a decimal point are suppressed; dates are printed in the format mm/dd/yy, where:

mm

is the month;

dd

is the day of the month; and

yy

is the year.

- **European Format:** commas are used as decimal points; thousands-units are separated by periods; all leading zeros preceding a decimal point are suppressed; dates are printed as dd.mm.yy.
- **European Format with Leading Zero:** commas are used as decimal points; thousand units are separated by periods; leading zeros are suppressed up to one position before a decimal point; dates are printed as dd.mm.yy.
- **United Kingdom Format:** periods are used as decimal points; thousand-units are separated by commas; all leading zeros preceding a decimal point are suppressed; dates are printed as dd/mm/yy.

Examples illustrating these formats appear in Table 3-1, below:

Table 3-1. Inverted Print Option Formats

Column 21 Entry	Format	Numeric Literal, Period or Comma as Decimal Point	Edit Code Field, Period or Comma as Decimal Point or Separator	Zeros Suppressed Left of Decimal Point	UDATE/Slash or Period
blank	Domestic	1974.19	9,762.55	.14	06/24/74
I	European	1974,19	9.762,55	.14	24.06.74
J	European (w/Lead Zero)	1974,19	9.762,55	0,14	24.06.74
D	United Kingdom	1974.19	9,762.55	.14	24/06/74

EXAMPLE

To request output of data in the United Kingdom Format, enter D in the Inverted Print Field (Figure 3-1, Item 3).

As a result, the numeric literal 03.01, the edited field 33,306.00, and the date August 21, 1974, are printed as:

3.01 33,306.00 21/08/74

3-6. Record Number Adjust (Column 22)

In this field you declare the adjustment, if any, to be made by RPG to Relative Record Numbers prior to their use in retrieving records from direct-access MPE and KSAM files.

Column 22 Entry	Meaning
1	RPG will subtract the value 1 from all Relative Record Numbers prior to using them for direct file access (unless the file is KSAM built with the Firstrec=1 option).
+	RPG will add the value 1 to all Relative Record Numbers prior to using them for direct file access.
0 or blank	RPG will not adjust Relative Record Numbers prior to using them for direct File Access.

This field is used to request that RPG automatically adjust the Relative Record Numbers supplied by your program prior to their use in accessing records, and applies to all **MPE** and **KSAM** direct-access files ("1" in this field **does not** apply to **KSAM** files built with the **Firstrec=1** option).

The presence of a "1" in this field causes the value 1 (one) to be subtracted from each Relative Record Number obtained from an **ADDROUT** file, chaining operation, or chaining file, prior to its use by RPG in retrieving (or adding) a record. This option **does not** apply to **KSAM Firstrec=1** files.

The presence of a "+" in this field causes the value 1 (one) to be added to each Relative Record Number obtained from an **ADDROUT** file, chaining operation, or chaining file, prior to its use by RPG in retrieving (or adding) a record. This option **does** apply to **KSAM Firstrec=1** files.

This field has no effect on **IMAGE/3000** files.

3-6A (Columns 23-24 are not used.)**3-6B Program Name Logging (Column 25)**

In this field you determine whether a program identification line is to be printed to the standard list device (**\$STDLIST**) at the beginning of program execution, and at the end (unless the program terminates abnormally). This line identifies both the source program name (from Header spec columns 75-80, or from a **\$CONTROL NAME=** entry) and the object program file name. The formats of the beginning and ending identification lines are:

H Control Record Specification

Program: source = object.group.account day, mon dd, yyyy, hh:mm xM
Pgm-End: source = object.group.account day, mon dd, yyyy, hh:mm xM

Column 25 Entry

Meaning

L	Print a program identification line to \$STDLIST at the beginning and ending of program execution.
blank	Do not print program identification lines.

3-7. Alternate Collating Sequence (Column 26)

An entry in this field allows you to specify an alternative to the ASCII (JIS-Japanese Industrial Standard-Katakana characters) Collating Sequence for use in certain comparison operations:

Column 26 Entry

Meaning

S	Alternate collating sequence, specified in hexadecimal format, applies.
O	Alternate collating sequence, specified in octal format, applies.
E	EBCDIC (Extended Binary Coded Decimal Interchange Code) collating sequence applies.
K	EBCDIK collating sequence applies.
blank	Normal ASCII (JIS) collating sequence applies.

RPG/3000 normally performs various compare operations based on the ASCII Collating Sequence shown in Appendix G. In this sequence, each alphabetic, numeric, or special character holds a unique position in relation to all other characters. This position is determined by the internal representation of that character — the particular combination of bits denoting the character. You can, however, specify a different collating sequence so that characters will be treated as if ranked in a different order, or as if two or more characters occupy the same sequential position. This alternate collating sequence will then be used for alphanumeric comparison and matching field operations. It will not, however, affect control levels, numeric comparison, or table look-up operations (as defined later in this manual).

The **Alternate Collating Sequence Field** on the Control Record Specification allows you to specify that an alternate collating sequence is to be used. If you specify an alternate sequence other than EBCDIC or EBCDIK, you must define it in either hexadecimal or octal format on special ALTSEQ records, as described below. As an aid on coding these records, enter the new sequence numbers opposite the ASCII sequence on a special form — the **Translation Table and Alternate Collating Sequence Coding Sheet** (Figure 3-2). Then, simply transcribe these new values onto ALTSEQ records, using as many records as you need. Place these records within your program as shown below and in Paragraph 5-31, Section V. To specify the sequence in hexadecimal format, write each ALTSEQ record as shown in Table 3-2.

H Control Record Specification

To specify the sequence in octal format, write each ALTSEQ record as shown in Table 3-3.

Table 3-3. ALTSEQ Record Description for Octal Format

Column	Entry	Meaning
1-6	ALTSEQ	This is an ALTSEQ record.
7-8	blank	None.
9-14	Two ASCII characters, specified by sequence numbers in octal.	The first character (Columns 9-11) replaces the second character (Columns 12-14) in the ASCII Collating Sequence. (An example appears below.)
15-80	Two ASCII characters, specified by sequence numbers in octal.	Same meanings as Columns 9-14, repeated as needed.

Enter the ALTSEQ records as a table, using one of the methods noted below and described in detail in Section V:

1. Load the records as part of your job stream, or
2. Load the records from a disc file. When you do this, you must name this disc file on a special **Table/Array File Name Record** entered in your RPG program. (An example of a Table/Array File Name Record appears below; complete information on this type of record appears in Section V.)

Note: This method provides an extension to the standard method available with other RPG compilers (Method 1, above). It requires that you create and load the ALTSEQ records only once, even though you may run several programs that all use those records.

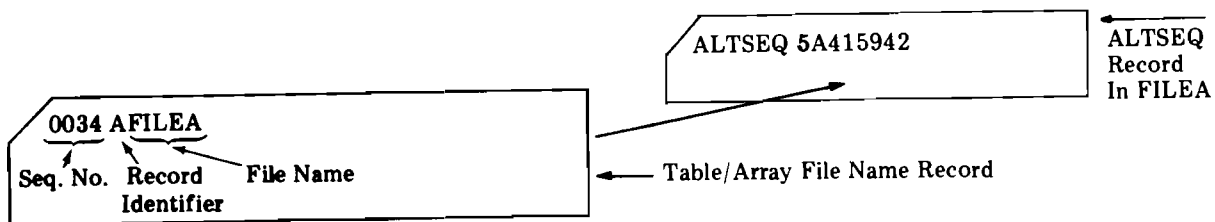
If you specify the collating sequence based on the Extended Binary Coded Decimal Interchange Code (EBCDIC), you need only enter **E** in the **Alternate Collating Sequence Field**; no ALTSEQ records apply. This entry requests translation of data from ASCII to EBCDIC characters, with comparison on the basis of EBCDIC. You would use this entry when your program reads data prepared on EBCDIC-based computer systems or when your data, for any other reason, depends on the EBCDIC sequence. The primary differences between the EBCDIC and ASCII sequences are:

1. In the EBCDIC sequence, alphabetic characters fall before numeric characters; in the ASCII sequence, the converse is true.
2. All special characters differ between these two sequences.

EXAMPLES

To supply an Alternate Collating Sequence so that the ASCII Z (hexadecimal 5A) replaces the ASCII A (hexadecimal 41) and the ASCII Y (hexadecimal 59) replaces the ASCII B (hexadecimal 42), with Y and Z also still retaining their normal places in the sequence, follow the directions below. (In this example, the ALTSEQ records are loaded from disc.)

1. Enter S in Column 26 of the Control Record Specification, as shown in Figure 3-1, Item 4.
2. Make the entries noted as Items 1 and 2 on the Translation Table and Alternate Collating Sequence Coding Sheet in Figure 3-2.
3. Translate these entries onto ALTSEQ records as follows. (Assume that these records will be stored in FILEA.)



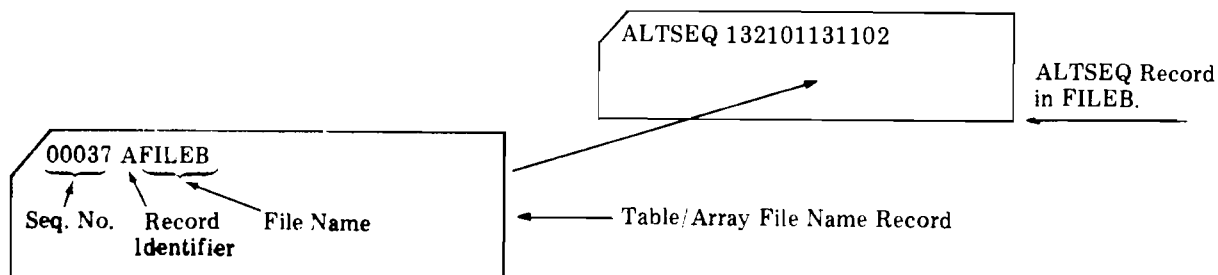
The resulting sequence is:

Z
Y
C
D
E
.
.
.
W
X
Y
Z

H Control Record Specification

To supply an Alternate Collating Sequence so that the ASCII Z (octal 132) replaces the ASCII A (octal 101) and ASCII Y (octal 131) replaces ASCII B (octal 102), with Y and Z also retaining their normal places, proceed as follows: (The ALTSEQ records are all loaded from disc.)

1. Enter O in Column 26 of the Control Record Specification.
2. Make the entries noted as Items 1 and 2 on the Translation Table and Alternate Collating Sequence Coding Sheet in Figure 3-3.
3. Translate these entries onto ALTSEQ records, as follows. (Assume that these records will reside in FILEB.)



The resulting sequence is the same as noted in the previous example.

3-7A (Column 27 is not used.)

3-7B BUFCHK Defaults (Column 28)

In this field you define the default settings for the three BUFCHK options -- Current Data Checking (CDC), No-Read Checking (NRC), and Update-Protect Checking (UPC). These defaults will be applied to all files, according to the definitions of the three options, and can be overridden for specific files using BUFCHK Continuation records.

Column 28

Entry	Meaning
C	Enable CDC for all Sequential (MPE) and KSAM files that specify a LOCK Continuation record, and are linked to another CDC file at runtime by MPE file equations.
N	Enable NRC for all update files
B	Enable both CDC and NRC (as defined above).
U	Enable UPC and NRC for all update files.
X	Enable CDC, NRC, and UPC (as defined above).

Refer to the discussion of BUFCHK Continuation records on page 4-30b for complete descriptions of Current Data Checking (CDC), No-Read Checking (NRC) and Update-Protect Checking (UPC).

3-7C (Columns 29-33 are not used.)

3-8. Table/Array Look-Up (Column 34)

If an object program references data contained in sequential tables or arrays, you can specify, in the Table/Array Look-Up Field, how you want the program to search for this data:

Column 34 Entry	Meaning
blank	Sequential look-up applies.
B	Binary look-up applies.

RPG/3000 provides two data-searching methods:

- **Sequential Look-Up** is most effective if the tables or arrays are small (less than 16 entries), or if the entries used most often are few and appear at the beginning of the table or array.
- **Binary Look-Up** is excellent for large tables and arrays.

EXAMPLE

To request a binary search, enter B in the Table/Array Look-Up Field (Figure 3-1, Item 5).



RPG TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	Character	Hexa Decimal Entry	Octal Entry	Replace/ Replaced By
00000000	NUL	00	000	
00000001	SOH	01	001	
00000002	STX	02	002	
00000003	ETX	03	003	
00000004	EOT	04	004	
00000005	END	05	005	
00000006	ACK	06	006	
00000007	BEL	07	007	
00000008	BS	08	010	
00000009	HT	09	011	
00000010	LF	0A	012	
00000011	VT	0B	013	
00000012	FF	0C	014	
00000013	CR	0D	015	
00000014	SO	0E	016	
00000015	SI	0F	017	
00000016	DLE	10	020	
00000017	DC1	11	021	
00000018	DC2	12	022	
00000019	DC3	13	023	
00000020	DC4	14	024	
00000021	NAK	15	025	
00000022	SXA	16	026	
00000023	ETB	17	027	
00000024	CAN	18	030	
00000025	EM	19	031	
00000026	SUB	1A	032	
00000027	ESC	1B	033	
00000028	FS	1C	034	
00000029	GS	1D	035	
00000030	RS	1E	036	
00000031	JS	1F	037	
00000032	SP	20	040	
00000033		21	041	
00000034		22	042	
00000035		23	043	
00000036		24	044	
00000037		25	045	
00000038		26	046	
00000039		27	047	
00000040		28	050	
00000041		29	051	
00000042		2A	052	
00000043		2B	053	
00000044		2C	054	
00000045		2D	055	
00000046		2E	056	
00000047		2F	057	
00000048		30	060	
00000049		31	061	
00000050		32	062	
00000051		33	063	
00000052		34	064	
00000053		35	065	
00000054		36	066	
00000055		37	067	
00000056		38	070	
00000057		39	071	
00000058		3A	072	
00000059		3B	073	
00000060		3C	074	
00000061		3D	075	
00000062		3E	076	
00000063		3F	077	
00000064		40	100	
00000065		41	101	
00000066		42	102	
00000067		43	103	
00000068		44	104	
00000069		45	105	
00000070		46	106	
00000071		47	107	
00000072		48	110	
00000073		49	111	
00000074		4A	112	
00000075		4B	113	
00000076		4C	114	
00000077		4D	115	
00000078		4E	116	
00000079		4F	117	
00000080		50	120	
00000081		51	121	
00000082		52	122	
00000083		53	123	
00000084		54	124	
00000085		55	125	
00000086		56	126	
00000087		57	127	
00000088		58	130	
00000089		59	131	
00000090		5A	132	
00000091		5B	133	
00000092		5C	134	
00000093		5D	135	
00000094		5E	136	
00000095		5F	137	
00000096		60	140	
00000097		61	141	
00000098		62	142	
00000099		63	143	
00000100		64	144	
00000101		65	145	
00000102		66	146	
00000103		67	147	
00000104		68	150	
00000105		69	151	
00000106		6A	152	
00000107		6B	153	
00000108		6C	154	
00000109		6D	155	
00000110		6E	156	
00000111		6F	157	
00000112		70	160	
00000113		71	161	
00000114		72	162	
00000115		73	163	
00000116		74	164	
00000117		75	165	
00000118		76	166	
00000119		77	167	
00000120		78	170	
00000121		79	171	
00000122		7A	172	
00000123		7B	173	
00000124		7C	174	
00000125		7D	175	
00000126		7E	176	
00000127		7F	177	
00000128		80	200	
00000129		81	201	
00000130		82	202	
00000131		83	203	
00000132		84	204	
00000133		85	205	
00000134		86	206	
00000135		87	207	
00000136		88	210	
00000137		89	211	
00000138		8A	212	
00000139		8B	213	
00000140		8C	214	
00000141		8D	215	
00000142		8E	216	
00000143		8F	217	
00000144		90	220	
00000145		91	221	
00000146		92	222	
00000147		93	223	
00000148		94	224	
00000149		95	225	
00000150		96	226	
00000151		97	227	
00000152		98	230	
00000153		99	231	
00000154		9A	232	
00000155		9B	233	
00000156		9C	234	
00000157		9D	235	
00000158		9E	236	
00000159		9F	237	
00000160		A0	240	
00000161		A1	241	
00000162		A2	242	
00000163		A3	243	
00000164		A4	244	
00000165		A5	245	
00000166		A6	246	
00000167		A7	247	
00000168		A8	250	
00000169		A9	251	
00000170		AA	252	
00000171		AB	253	
00000172		AC	254	
00000173		AD	255	
00000174		AE	256	
00000175		AF	257	
00000176		B0	260	
00000177		B1	261	
00000178		B2	262	
00000179		B3	263	
00000180		B4	264	
00000181		B5	265	
00000182		B6	266	
00000183		B7	267	
00000184		B8	270	
00000185		B9	271	
00000186		BA	272	
00000187		BB	273	
00000188		BC	274	
00000189		BD	275	
00000190		BE	276	
00000191		BF	277	

Figure 3-3. Specifying an Alternate Collating Sequence in Octal Code

3-8A. (Columns 35–38 are not used.)**3-8B. EBCDIC Zone/Digit Tests (Column 39)**

This field determines whether or not record identification data will be converted to EBCDIC format for identifications that use just the Zone (leftmost 4 bits) or Digit (rightmost 4 bits) portion of the data.

Column 39 Entry	Meaning
E	For Input specifications record identification codes "Z" and "D", convert the identification characters to EBCDIC format for comparison of just the Zone or Digit portions.
blank	Use the original ASCII representations of all characters in record identification testing. Note: Do not specify this option if record identification data contains packed numeric data.

This option provides compatibility with other vendors' EBCDIC-based systems.

3-9. Sign Process (Column 40)

This field determines whether sign-forcing is to take effect in your program. Sign-Forcing means that numeric signs will be changed when fields are moved, as shown in Table 3-3a. Minus signs in the data are not affected. As an example, if you wish a numeric field containing + 1 to be output as the ASCII character 1, you would request sign-forcing (or editing as described in Section IX); otherwise, the value + 1 would be output as the ASCII character A.

Column 40 Entry	Meaning
B or blank	Sign-forcing takes place only on Output to unpacked numeric fields. Original signs are retained on all other field moves.
S	Standard sign-forcing occurs on all field moves. Thus, sign-forcing takes place on input of all unpacked numeric fields; on all Calculation specification moves* of alphanumeric fields to numeric fields; and on output to all unpacked numeric fields.
I	Same as S (standard sign-forcing), except that no sign forcing occurs on input of unpacked numeric fields.
O	Same as S (standard sign-forcing), except that no sign forcing occurs on output to unpacked numeric fields.
N	No sign-forcing takes place. Thus, original signs are retained whenever fields are moved.

Sign-Forcing

Column-40	Input unpacked numeric fld	Move* alphanumeric to numeric fld	Output unpacked numeric fld
B or blank	none	none	C→F
S	C→F	F→C	C→F
I	none	F→C	C→F
O	C→F	F→C	none
N	none	none	none

Legend

C = Positive Sign (Sign bits = 1100)

F = Unsigned (Sign bits = 1111)

C→F = Positive signs changed to Unsigned by the move

F→C = Unsigned field changed to Positive by the move

none = Signs not changed by the move

Table 3-3a. Sign-Forcing Options

*Refers to the MOVE, MOVEL, MLLZO, MHHZO, MLHZO, and MHLZO Calculations operators. NOTE — No sign-forcing occurs for the MOVEL operator when the Result Field is longer than or the same length as the Factor-2 Field.

Note: Information on field data formats, essential in understanding the complete ramifications of sign-forcing, appears in Section VII.

EXAMPLE

To prevent sign-forcing on both input and output, enter N in the Sign-Process Field (as in Figure 3-1, Item 6).

3-10. Form Positioning (Column 41)

You can request verification of printer form position by an entry in this field:

Column 41 Entry	Meaning
1	Program repeats a forms-alignment record on the printer until operator verifies correct form position.
blank	Form position is not verified.

Your object program may write output upon special forms, such as paychecks or accounting forms, mounted on a line printer. Before printing begins, you can ensure that these forms are properly aligned on the printer by making an entry in the Form Positioning Field. This requests your object program to print a special forms-alignment record on the printer, send a message to the computer operator at the console asking him to verify the alignment of your forms, and wait. This pause allows the operator to reposition the forms, if necessary, and signal the system to resume. If the operator verifies that the forms are correctly positioned, the object program then prints the first line (according to the specified carriage-control directive) and continues. Otherwise, the object program repeats the forms-alignment record on a new page, again requests the operator to verify form position, and pauses. This procedure is repeated until the operator actually verifies correct positioning. The object program does not begin incrementing the report page count until the forms are positioned properly. (Form-positioning operates as described above for both spooled and non-spooled output files.) Further information on operator/program interaction for forms alignment appears in the MPE reference manuals and console operator's guide.

Note: RPG has no way to determine if an output file will be printed on a line printer unless the user program includes vertical spacing controls, such as space or skip within the output specifications for the file.

EXAMPLE

Request the operator to verify that the forms used for output are positioned properly by entering 1 in the Form Positioning Field (Figure 3-1, Item 7).

3-11. Indicator Setting (Column 42)

This field establishes the setting of RPG indicators used to test data fields for blank or zero contents (Blank/Zero Indicators) at program initialization and following RPG Blank-After operations.

Column 42 Entry	Meaning
S	Blank/Zero Indicators are set on at program initialization and following Blank-After operations.
T	Blank/Zero Indicators are set on at program initialization, but not following Blank-After operations.
B	Blank/Zero Indicators are not set on at program initialization nor following Blank-After operations.
blank	Blank/Zero Indicators are not set on at program initialization; they are set on, however, following Blank-After operations.

Programmers frequently use various indicators (special flags or switches) in an object program to test for certain conditions and to branch to different operations on the basis of these conditions. One of these is the First-Page (1P) Indicator that determines when the first line of output is printed; this indicator is initially set on, and directs the program to print the report heading when it is referenced in the Output Specifications. When the specified output is complete, the 1P indicator is turned off. Another indicator is the L0 Indicator, used to indicate the highest total level; this indicator is also initially set on, and is used as described in Section VII. (Further information on these indicators appears in Appendix D.)

H Control Record Specification

Other indicators are used to indicate the contents of data fields -- for example, **Blank/Zero Indicators** are turned on during input or calculation operations to indicate fields filled with blanks or zeros; initially, however, all these indicators are shut off, even though alphabetic fields are filled with blanks and numeric fields are filled with zeros when they are initialized by RPG. Thus, **Blank/Zero Indicators** used in Input or Calculation Specifications do not initially reflect blank or zero field contents unless you specifically direct RPG to set these indicators on; to do this, you must enter S or T in the Indicator Setting Field of the Control Record Specification.

The following rules determine which indicators are defined by the compiler as **Blank/Zero Indicators**:

Input Specifications — Any indicator named in columns 69-70 (**Zero or Blank Indicator**) of Field Descriptions.

Calculation Specifications — Any indicator named in columns 58-59 (**Equal/Zero Resulting Indicator**) of specifications that have one of the Arithmetic operations (ADD, Z-ADD, SUB, Z-SUB, MULT, DIV, MVR, and XFOOT; excluding SQRT) in columns 28-32.

The **Blank-After** request (Column 39 of the Output Specifications) allows you to fill a field with zeros or blanks after that field is output, and to set **Blank/Zero Indicators** on to reflect this action. In RPG/3000, when an input or calculation operation specifies a **Blank/Zero Indicator** with an input or result field, the indicator is set on when the field is blanked after it is output. (When a group of indicators is used to test such a field, only the first indicator specified is set on.) In IBM's System/3 RPG, however, no indicator is set on in such a case. To suppress this RPG/3000 feature when converting a System/3 program for compilation under RPG/3000, enter B or T as indicated above.

Further information on all types of indicators appears throughout the remainder of this manual. In summary, all indicators except the 1P and L0 Indicators are initially shut off unless S or T appears in the Indicator Setting Field.

EXAMPLE

To set all **Blank/Zero Indicators** on at program initialization and following **Blank-After** operations, enter S in the Indicator Setting Field (Figure 3-1, Item 8).

3-12. File Translation (Column 43)

This field allows you to request translation of information in a file from one code format into another.

Column 43 Entry	Meaning
F	Translate characters according to file translation table specified in hexadecimal format.
O	Translate characters according to file translation table specified in octal format.
blank	Translation is not required.

Occasionally, input files contain information coded in a format that cannot be read by your object program; conversely, you may want to write data to an output file in a form other than that used by your program. This occurs, for instance, when your object program processes data originally prepared for another machine, or when you want to output data in some confidential code understood by only a few of those who actually handle the data. In such cases, you must request the object program to translate the data before it is processed. When you request translation of data on combined input/output files, or files to be updated, RPG/3000 translates both the input and output portions of the file.

When you request translation of one or more files, characters read from input files (external characters) are translated into a form usable by your program (internal characters); conversely, characters used by your program (internal characters) are translated to another form (external characters). If you request translation, you must also specify what external characters correspond to which internal characters for translation purposes. You do this on special records entered as a file translation table (as discussed below and in Section V), in either hexadecimal or octal format. Before preparing the records in this table, you can plan their contents on the Translation Table and Alternate Collating Sequence Coding Sheet, introduced previously in Figure 3-2. Then, transcribe these items onto as many records as they require. To specify in hexadecimal the characters requiring translation, write the records as shown in Table 3-4.

Table 3-4. Translation Record Description with Hexadecimal Codes

Column	Entry	Meaning
1-8	<p>*FILES</p> <p>Valid file name, (as used in File Description Specifications). (File names can contain up to eight characters, beginning with one of the letters A through Z. The remaining characters can be letters and/or any of the digits 0 through 9. Embedded blanks are not allowed.)</p>	<p>Translate all files referenced by the program.</p> <p>Translate the file identified by this file name. (Use a different record for each file named.)</p>
9-12	Two ASCII characters specified in hexadecimal.	The first character is an external ASCII character; the second character is the corresponding internal ASCII character.
13-80	Sets of two ASCII characters, specified in hexadecimal.	Same meaning as Columns 9-12, repeated as needed.

To specify in octal the characters requiring translation, write the translation records as shown in Table 3-5.

Table 3-5. Translation Record Description with Octal Codes

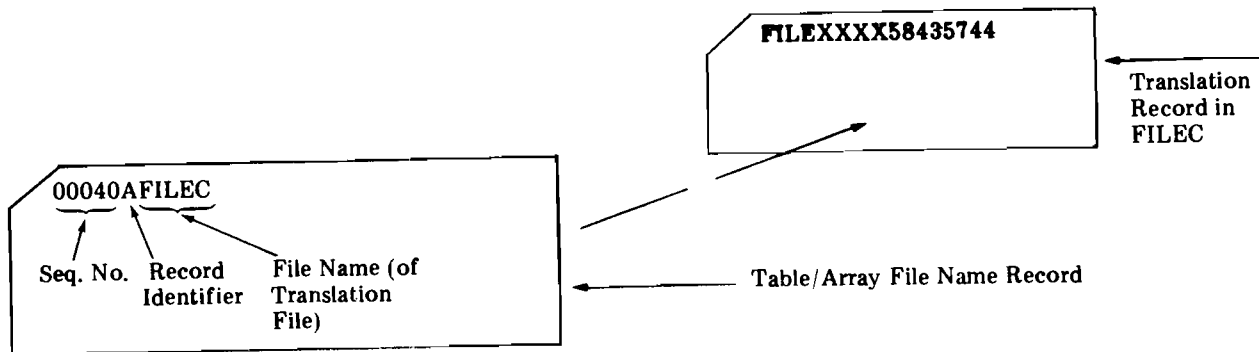
Column	Entry	Meaning
1-8	<p>*FILES</p> <p>Valid file name, subject to the above rules.</p>	<p>Translate all files referenced by the program.</p> <p>Translate the file referenced by this file name.</p>
9-14	Two ASCII characters, specified in octal.	The first character is an external ASCII character; the second is the corresponding internal ASCII character.
15-80	Sets of two ASCII characters, specified in octal.	Same meanings as Columns 9-14, repeated as needed.

Enter these records as a table, using the same techniques summarized for ALTSEQ records in paragraph 3-7. Remember that if you load the table from disc, you must name the file containing this table on a Table/Array File Name Record (discussed in detail in Section V). The table can contain as many records as necessary. Furthermore, the program can reference as many translation tables as necessary but the records for each table must be contiguous.

EXAMPLES

An RPG/3000 program prints a report dealing with the status of various company employees. The report makes reference to wage and salary curves C and D -- curves whose meanings are general knowledge to many employees. To help restrict salary information to proper personnel only, you could establish translation criteria so that the internal characters C and D (hexadecimal 43 and 44) are translated into the external characters X and W (hexadecimal 58 and 57,) respectively on the report, and give the field in which they appear a meaningless "dummy" heading. To accomplish this with a hexadecimal translation table, proceed as shown below. (In this case, the translation table is stored in a file that is loaded from disc.)

1. Enter F in Column 43 of the Control Record Specification (Figure 3-1, Item 9).
2. Make the entries noted as Item 1 and 2 on the Translation Table and Alternate Collating Sequence Coding Sheet in Figure 3-4.
3. Transcribe these entries onto a translation record, as follows. (Assume that this record will reside in a translation table in the file named FILEC.)



Note: For updated or combination input/output files, X and W become C and D on input, and C and D return to X and W on output.



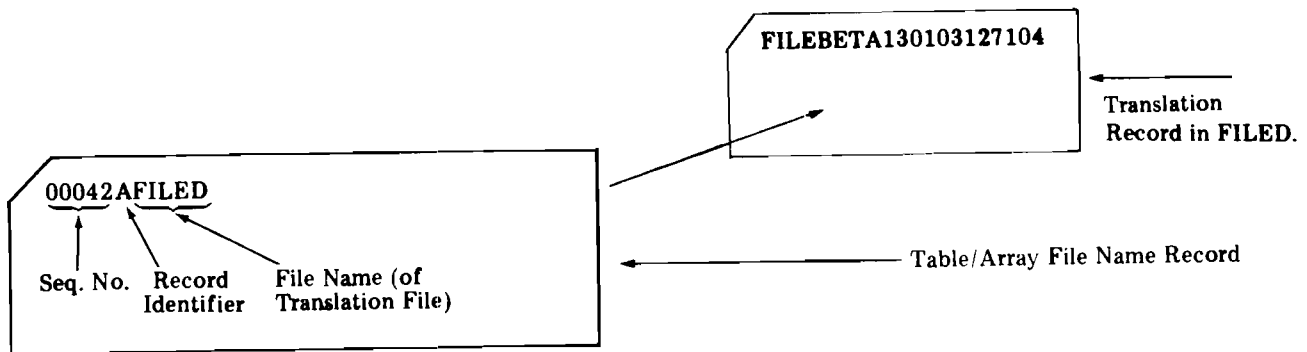
RPG TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Main table with columns: Code, Character, Hexa decimal Entry, Octal Entry, Replace/ Replace By. The table is divided into four sections, each representing a different collating sequence. Handwritten annotations '58' and '57' are present in the second section.

Figure 3-4. Specifying File Translation in Hexadecimal Code

To accomplish the same task with an octal translation file:

1. Enter O in Column 43 of the Control Record Specification.
2. Make the entries noted as Items 1 and 2 on the Translation Table and Alternate Collating Sequence Coding Sheet in Figure 3-5.
3. Transcribe these entries onto a translation record as follows. (Assume that this record will be stored as a translation table in the file named FILED.)



3-12A. Non-Numeric Digits (Column 44)

This field determines whether or not non-numeric data (special characters, such as !, #, \$, % ?, etc.) will be allowed to be moved into numeric fields when Input field moves, Data Structure field moves, or Calculations MOVE or MOVEL operations are performed.

Column 44 Entry	Meaning
N	Allow non-numeric data ("special" characters) to be moved into numeric fields.
blank	Do not allow non-numeric data ("special" characters) to be moved into numeric fields. If such an attempt is made, RPG will generate runtime error #13 "INVALID NUMERICAL DATA".

This option provides compatibility with other vendors' RPG systems.

3-12B. (Column 45 is not used.)

3-13. Dollar-Sign Substitute (Column 46)

In this field you can request that another character be treated as a dollar sign in edit words and edit codes.

Column 46 Entry	Meaning
Any special character (not a letter or digit).	Treat this character as a dollar sign when encountered in edit words and codes.
blank	Do not substitute any character for the dollar sign.

This entry increases the flexibility of edit words and edit codes used in Output Specifications. (**Edit words** are fields that permit you to specify the contents of an output field character by character; **edit codes** are single-character codes used to insert special characters, suppress leading zeros, and enter floating dollar signs in numeric output fields.) In edit words and codes, the dollar sign has a unique meaning, as described in Section IX. You may, however, want to assign this meaning to some other character so that your object program effectively treats that character as a dollar sign when it is encountered in edit words and codes. You may use this feature, for example, in a program that prints a report dealing with foreign currency.

For dollar-sign substitution, first specify the substitute character in Column 46 of the Control Record Specification. When you write the Output Specifications, enter the substitute character in place of the dollar sign in edit words, or in place of a floating dollar sign in edit codes.

EXAMPLE

Indicate that the number sign (#) is to be treated as a dollar sign, by entering # in the Dollar-Sign Substitute Field (Figure 3-1, Item 10).



HEWLETT PACKARD RPG TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	Character	Hexa decimal Entry	Octal Entry	Replaces Replaced By
000000	A	41	100	
000001	B	42	101	
000002	C	43	102	
000003	D	44	104	
000004	E	45	105	
000005	F	46	106	
000006	G	47	107	
000007	H	48	110	
000008	I	49	111	
000009	J	4A	112	
000010	K	4B	113	
000011	L	4C	114	
000012	M	4D	115	
000013	N	4E	116	
000014	O	4F	117	
000015	P	50	120	
000016	Q	51	121	
000017	R	52	122	
000018	S	53	123	
000019	T	54	124	
000020	U	55	125	
000021	V	56	126	
000022	W	57	127	
000023	X	58	130	
000024	Y	59	131	
000025	Z	5A	132	
000026	[5B	133	
000027	\	5C	134	
000028]	5D	135	
000029	^	5E	136	
000030	_	5F	137	
000031	`	60	140	
000032	a	61	141	
000033	b	62	142	
000034	c	63	143	
000035	d	64	144	
000036	e	65	145	
000037	f	66	146	
000038	g	67	147	
000039	h	68	150	
000040	i	69	151	
000041	j	6A	152	
000042	k	6B	153	
000043	l	6C	154	
000044	m	6D	155	
000045	n	6E	156	
000046	o	6F	157	
000047	p	70	160	
000048	q	71	161	
000049	r	72	162	
000050	s	73	163	
000051	t	74	164	
000052	u	75	165	
000053	v	76	166	
000054	w	77	167	
000055	x	78	170	
000056	y	79	171	
000057	z	7A	172	
000058	[7B	173	
000059	\	7C	174	
000060]	7D	175	
000061	^	7E	176	
000062	_	7F	177	
000063	`	80	180	
000064	a	81	181	
000065	b	82	182	
000066	c	83	183	
000067	d	84	184	
000068	e	85	185	
000069	f	86	186	
000070	g	87	187	
000071	h	88	190	
000072	i	89	191	
000073	j	8A	192	
000074	k	8B	193	
000075	l	8C	194	
000076	m	8D	195	
000077	n	8E	196	
000078	o	8F	197	
000079	p	90	200	
000080	q	91	201	
000081	r	92	202	
000082	s	93	203	
000083	t	94	204	
000084	u	95	205	
000085	v	96	206	
000086	w	97	207	
000087	x	98	210	
000088	y	99	211	
000089	z	9A	212	
000090	[9B	213	
000091	\	9C	214	
000092]	9D	215	
000093	^	9E	216	
000094	_	9F	217	
000095	`	A0	220	
000096	a	A1	221	
000097	b	A2	222	
000098	c	A3	223	
000099	d	A4	224	
000100	e	A5	225	
000101	f	A6	226	
000102	g	A7	227	
000103	h	A8	230	
000104	i	A9	231	
000105	j	AA	232	
000106	k	AB	233	
000107	l	AC	234	
000108	m	AD	235	
000109	n	AE	236	
000110	o	AF	237	
000111	p	B0	240	
000112	q	B1	241	
000113	r	B2	242	
000114	s	B3	243	
000115	t	B4	244	
000116	u	B5	245	
000117	v	B6	246	
000118	w	B7	247	
000119	x	B8	250	
000120	y	B9	251	
000121	z	BA	252	
000122	[BB	253	
000123	\	BC	254	
000124]	BD	255	
000125	^	BE	256	
000126	_	BF	257	
000127	`	C0	260	
000128	a	C1	261	
000129	b	C2	262	
000130	c	C3	263	
000131	d	C4	264	
000132	e	C5	265	
000133	f	C6	266	
000134	g	C7	267	
000135	h	C8	270	
000136	i	C9	271	
000137	j	CA	272	
000138	k	CB	273	
000139	l	CC	274	
000140	m	CD	275	
000141	n	CE	276	
000142	o	CF	277	
000143	p	D0	280	
000144	q	D1	281	
000145	r	D2	282	
000146	s	D3	283	
000147	t	D4	284	
000148	u	D5	285	
000149	v	D6	286	
000150	w	D7	287	
000151	x	D8	290	
000152	y	D9	291	
000153	z	DA	292	
000154	[DB	293	
000155	\	DC	294	
000156]	DD	295	
000157	^	DE	296	
000158	_	DF	297	
000159	`	E0	300	
000160	a	E1	301	
000161	b	E2	302	
000162	c	E3	303	
000163	d	E4	304	
000164	e	E5	305	
000165	f	E6	306	
000166	g	E7	307	
000167	h	E8	310	
000168	i	E9	311	
000169	j	EA	312	
000170	k	EB	313	
000171	l	EC	314	
000172	m	ED	315	
000173	n	EE	316	
000174	o	EF	317	
000175	p	F0	320	
000176	q	F1	321	
000177	r	F2	322	
000178	s	F3	323	
000179	t	F4	324	
000180	u	F5	325	
000181	v	F6	326	
000182	w	F7	327	
000183	x	F8	330	
000184	y	F9	331	
000185	z	FA	332	
000186	[FB	333	
000187	\	FC	334	
000188]	FD	335	
000189	^	FE	336	
000190	_	FF	337	

Figure 3-5. Specifying File Translation in Octal Code

H Control Record Specification

3-14. Skip-Suppress (Column 47)

This field allows you to prevent the line printer from skipping to a new page at the beginning of program output.

Column 47 Entry	Meaning
S	Suppress skip so that output begins at current position of paper in printer.
blank	Skip to a new page.

When an RPG/3000 object program begins, the line printer normally skips to a new page so that any report output starts at a convenient place. You can prevent printing from starting on a new page, however, by entering S in the Skip-Suppress Field. (The line on the new page where the output begins is determined by a punch in the first channel of the printer's carriage-control tape; an S in the Skip-Suppress Field suppresses the initial skip to channel 1 of the tape.) Use this option if you have entered 1 in the Form Positioning Field (Column 41). This will minimize the paper used until the operator verifies that the forms are properly aligned in the printer.

EXAMPLE

Suppress the normal skip to channel 1, by entering S in the Skip-Suppress Field (Figure 3-1, Item 11).

3-14A. DSPLY Options (Column 48)

This field determines how DSPLY operations will be performed.

Column 48 Entry	Meaning
N	No "DSPLY" literal; Input on same line Suppress printing of the "DSPLY" literal on each use of DSPLY and enter new Result Field value on the same line as where the old value is displayed instead of on the next line, below the old value.
D	Use "DSPLY" literal; Input on same line Print the "DSPLY" literal on each use of DSPLY and enter new Result Field value on the same line as where the old value is displayed instead of on the next line, below the old value.
B	No "DSPLY" literal; Input on line below Suppress printing of the "DSPLY" literal on each use of DSPLY and enter new Result Field value directly below display of the old value.
blank	Use "DSPLY" literal; Input on line below (Default option) Print the "DSPLY" literal on each use of DSPLY and enter new Result Field value directly below display of the old value.
S	Suppress new DSPLY operations If it is necessary to recompile a DSPLY program using the A.05.00 or later compiler, but you need to run it using a pre-A.05.00 runtime library, you can avoid an SL BINDING ERROR by entering an "S" in Header column 48.

Refer to Section VIII for a full description of DSPLY operations.

3-14B. Record Length Check (Column 49)

This field determines how the runtime "Record Length" error will be handled. That error displays the warning message "ACTUAL < LOGICAL RECORD LENGTH FOR FILE xxxx" and will halt your program with a runtime error #2 "Unidentified Record" unless an appropriate entry is made in the Record Length Check field, as described below. (Refer to Section IV, 4-11, for further discussion of the Record Length error.)

Column 49 Entry	Meaning
E	For each "Record Length" error, a warning message will be displayed and your program will halt with a runtime error #2 "Unidentified Record". You may respond or pre-respond to this error in the same manner as other runtime errors. (Refer to paragraphs 3-18 and 3-19 for details on how to respond to runtime errors.)
N	No warning message will be displayed and no runtime error will occur for "Record Length" errors.
blank	For each "Record Length" error, a warning message will be displayed and your program will continue execution. No runtime error will occur.

3-14C. Page Overflow Test (Column 50)

This field specifies how RPG/3000 tests for page overflow.

Column 50 Entry	Meaning
P	Page overflow occurs when the next print line* is greater than the defined overflow line (i.e. the overflow line has been passed. This option provides compatibility with IBM System/3 RPG.)
blank	Page overflow occurs when the next print line* is equal to or greater than the defined overflow line (i.e. the overflow line has been either reached or passed.)

*"Next Print Line" is the line to which RPG has advanced its line counter after performing skip-after and/or space-after operations for the line just printed.

H Control Record Specification

3-14D. *PLACE METHOD (Column 51)

This field determines how *PLACE operations will be performed.

Column 51 Entry	Meaning
1	<p>*PLACE repeats only those fields preceding it that are not produced by *PLACE entries.</p> <p>This option allows RPG/3000 to perform repeated *PLACE operations in the same manner as the IBM System/3. That is, when you request *PLACE only once, RPG/3000 repeats the fields only once. When you use consecutive *PLACE entries, RPG/3000 repeats only the initial fields; it does not repeat any of the fields that were produced by preceding *PLACE entries.</p>
blank	<p>*PLACE repeats all those fields which precede it, including those fields produced by *PLACE entries.</p>

Refer to Section IX for a full description of *PLACE operations.

3-15. Cross-Reference Listing (Column 52)

You can request a Cross-Reference Listing indicating where all file names, field names, and indicators are referenced in your program.

Column 52 Entry	Meaning
X	Print a Cross-Reference Listing.
blank	Do not print a Cross-Reference Listing.

The Cross-Reference Listing shows:

- The sequence numbers of the records that define each file name, field name, and indicator.
- The sequence numbers of records that reference each file name, field name, and indicator.
- The memory location and length of each field name.
- The kind of data (alphanumeric or numeric) that each field contains.

NOTE: A Cross-Reference Listing can also be requested by using the \$CONTROL MAP compiler subsystem command option, (see Section XII.)

EXAMPLE

Request a Cross-Reference Listing by entering X in Column 52 (Figure 3-1, Item 12). The resulting listing appears in the format shown in Figure 3-6. The individual items in such a listing are explained in Section XI.

3-16. Carriage Control Type (Column 53)

This field specifies the method used to determine the line to which the printer skips in response to a skip request (Skip requests are specified in the Skip Field of the Output Specifications Sheet.):

Column 53 Entry	Meaning
blank	**CHANNEL OPTION** (Logical channels) Skip requests refer to logical carriage control channel numbers, whose associated line numbers are defined on Line Counter Specifications. RPG assumes that initial forms position is at the line specified by Channel 1 (default value is 6). Each page top-of-form is reached by a form-feed (i.e. a hardware page-eject, as controlled by the printer control tape channel-1 punch).
1	**LINE OPTION** (Simulates IBM System/3 usage) Skip requests refer to actual line numbers. RPG assumes that initial forms position is at line 1. Each page top-of-form is reached by a calculated series of line-feeds.
L	**LINE OPTION** (HP usage - First line is #6). Skip requests refer to actual line numbers. RPG assumes that initial forms position is at line 6. Each page top-of-form is reached by a calculated series of line-feeds (including page one if the first skip request is to a line number less than the assumed starting point of 6).

To associate channel numbers with print lines, leave column 53 of the Control Record Specification blank. Then, in the Line-Counter Specifications, equate the channels you plan to reference with the line numbers desired, as directed in section VI. Then, as an example, if line 10 is associated with channel 2, the printer skips to line 10 whenever channel 2 is referenced in a skip request.

Rather than referencing channel numbers in your skip requests, you can directly reference the lines to which the printer should skip. To use this method, enter L or 1 in Column 53 of the Control Record Specification. Then, use the Line Counter Specification Sheet to specify the length of the printed form, and the line at which it overflows. (See section VI.).

Since RPG controls all vertical forms movement by line counting, it cannot allow any movement where the number of lines is unknown — as would occur if use of the punched carriage control tape were allowed. The only punched channel ever referenced is Channel 1, to perform form-feeds (top-of-forms for Channel Option printing only).

H Control Record Specification

The internal line counter is initialized to the value which RPG has assumed to be the initial forms position (typically 6 or 1) and is only reset to that value after a form-feed.

Table 3-5a describes the initial carriage control values as determined by the contents of Control Record (Header) Specification Column 53, and the Line Counter Specifications.

Table 3-5a. Carriage Control Initializations

Header Col-53 Entry	INITIALIZATIONS			
	Line Counter & Assumed Top-of-Form	Chan2 – Chan12	Overflow Line	Forms Length
blank	=CHAN1 value as defined on Line Counter Specifications (Default is 6)	=Channel values as defined on Line Counter Specifications (Default is) (Channel# X 5)	=CHAN12 or "OL" as defined on Line Counter Specifications (Default is 60)	Not Used-
1	=1	-Not Used-	="OL" as defined on Line Counter Specifications (Default is 60)	="FL" as defined on Line Counter Specifications (Default is 66)
L	=6			

After the opening of each print file, a form-feed is done (unless Header Specifications column 47 contains "S" or column 41 contains "1") to ensure that initial forms position is at the hardware top-of-form.

All Skip requests, except form-feeds (Channel Option only), are done by use of a calculated series of line-feeds to reach the desired print line.

EXAMPLE

Equate top-of-form with line 1 on the printed page and begin line-counting at line 1 by entering 1 in Column 53 to simulate IBM System/3 line skipping (Figure 3-1, Item 13).

Other examples, illustrating the interrelationship of Control Record, Line Counter, and Output Specifications with respect to line-skipping, appear in Section IX.

PAGE 0003 TEXTSL INDICATORS USED

INDICATOR 10 DEFINED	0020
REFERENCED	0021
INDICATOR 15 DEFINED	0007
REFERENCED	0039
INDICATOR L1 DEFINED	0010
REFERENCED	0047
INDICATOR L2 DEFINED	0009
REFERENCED	0053

PAGE 0004 TEXTSL FIELD NAMES USED

AUTH (FIELD) 10	0011
REFERENCED	0041
BKSL (FIELD) 7.2	0016
REFERENCED	0017 0018 0019 0046
EOITN (FIELD) 3	0013
REFERENCED	0043
GARSL (FIELD) 10.2	0010
REFERENCED	0057
GSAREA (FIELD) 10	0009
REFERENCED	0055
NUMSWP (FIELD) 5.0	0015
REFERENCED	0016 0021 0044

PAGE 0006 TEXTSL FILE NAMES USED

CARDS	0002
REFERENCED	0007
REPORT	0004
REFERENCED	0006 0022
TABFILE	0003
REFERENCED	0005

Figure 3-6. Cross-Reference Listing

H Control Record Specification

3-17. Textfile Sequence Check (Column 54)

This field enables you to request sequence-checking of textfile input:

Column 54 Entry	Meaning
S	Check all records in textfile for proper sequence, based on the sequence numbers appearing in Columns 1-5 of these records. Print a message if a sequence error occurs.
N or blank	Do not perform sequence checking.

RPG/3000 source programs can be compiled in two ways:

1. By reading all compiler subsystem commands and source program records from a single primary input file, called a **textfile**.
2. By merging the primary compiler subsystem command and source program record input on a **masterfile** with additional command and/or source record input on a **textfile** during compilation. In this case, the commands and source records in the **textfile** provide editing capability by superceding and/or supplementing those in the **masterfile**.

In the first case, you can request or suppress sequence checking by making the entries described above in the **Textfile Sequence Check Field** of the **Control Record Specification Sheet**. In the second case, sequence-checking takes place automatically (as described in Sections XI and XII).

If you enter **S**, the compiler will verify that once a record with a valid sequence number appears, all subsequent records contain sequence numbers in ascending numerical order. Subsequent records with blank sequence number fields will not result in errors. If **all** records contain blank sequence-number fields, the compiler will assume that they are in proper sequence.

Embedded blanks **within** a sequence number are interpreted as zeros, and thus affect the validity of the sequence.

EXAMPLE

Suppose you submit a source program whose records are arranged in the following order.

```
0001
0002
0003
.
.
0014
0016
0015      (Note the Sequence Error.)
.
.
0025
0026
```

You can request sequence-checking by entering S in Column 54 (Figure 3-1, Item 14). When the sequence error is detected, a diagnostic message will appear.

3-18. Error Log (Column 55)

With the Error Log Field, you can re-direct or suppress transmission of run-time error messages:

Column 55 Entry	Meaning
blank	When an error occurs and a pre-selected response is specified in the Error Response Field , (described next), send the message to the standard list device. (This device is typically a printer for a batch job, or a terminal for a session.)
N	When an error occurs and a pre-selected response is specified in the Error Response Field , do not output an error message.
S	When an error occurs, whether or not a pre-selected response is specified in the Error Response Field , send an error message to the operator's console (in a job) or terminal (in a session), print an error dump (described in Appendix C), and terminate the program. Ignore the Error Response Field .

Normally, when an RPG/3000 object program encounters an error, a **run-time message** describing the error appears on the operator's console (in a batch job) or terminal (in an interactive session), and the program pauses. The operator can then direct the program to take any of three options: (1) continue processing in-line, (2) skip the erroneous record and continue processing, or (3) terminate. If you want to avoid this interaction with the operator—for instance, when anticipating an error that you want to ignore—you can:

1. Re-direct or suppress transmission of run-time messages, by an entry in the **Error Log Field** (Column 55) of the **Control Record Specification Sheet**, and
2. Specify a **pre-selected response** to run-time errors by an entry in the **Error Response Field** (Columns 56-70) of the **Control Record Specification Sheet**.

Note: You can re-direct or suppress transmission of run-time error messages only if you also specify a pre-selected response.

An Example showing how the **Error Log Field** is used appears under the discussion of the **Error Response Field** (because of the interrelationship of these two fields).

3-19. Error Response (Columns 56 through 71)

You can specify a pre-selected response to run-time errors (detected by RPG-generated code) in the Error Response Field. (In this field, each column corresponds to a particular error, as described in Table 3-6 and in Appendix B. Each error is identified by a particular number (2 through 17), shown above the column numbers on the specification sheet. Enter the pre-selected response under the appropriate column/error number that pertains to the error that you wish handled according to that response.)

Columns 56 through 71	Pre-Selected Response
0	Re-direct or suppress error message (in accordance with the Error Log Field) and continue program execution in-line.
1	Re-direct or suppress error message, skip the erroneous input record, and continue execution.
2	Re-direct or suppress error message, and terminate program (by executing normal termination code). This procedure turns on all level indicators (L1 through L9, and LR, as defined in Section VI), calculates all totals, prints all totals and table output, and closes all files.
3	Re-direct or suppress error message, close all files, and terminate program immediately. (Do not turn on level indicators, calculate totals, or print total and table output.)
4	Re-direct or suppress error message, print an error dump, and terminate program. This includes turning on all level indicators, calculating all totals, printing all totals, and closing all files.
5	Re-direct or suppress error message, print an error dump, and terminate program immediately. (Do not turn on level indicators, calculate totals, or print total and table output.)
blank	Transmit error message to operator — no preselected response is requested.

Note: All run-time errors are described in Appendix B. The above pre-selected responses do not all apply to all run-time errors. For the exceptions, see Appendix B.

If you specify S in the Error Log Field, RPG/3000 ignores the Error Response Field.

The seventeen possible RPG-detected error messages are listed in Table 3-6. This table shows:

- The error number (1 through 17).
- The message content.
- The probable cause of the error.
- The recovery procedure suggested.
- The column number (56 through 71) in the Error Response Field that corresponds to this error. (On the specification sheet, you make an entry (0 through 5) in this column to determine how you want the error handled.)
- The *ERROR Code. (This is a letter written into the one-byte, alphanumeric field named *ERROR. Whenever this error occurs, and the continue (0) or bypass (1) option is chosen in the Error Response Field, the *ERROR field can be tested by using Calculation Specifications.)
- The entries (0-5) allowed in the Error Response Field. (Not all pre-selected responses apply to all RPG-detected errors.)

Note: Error No. 1, which is a terminal error on a file, is always treated under Error Response No. 5, automatically. There is no provision for specifically requesting a pre-selected response for this error in the Error Response Field.

EXAMPLE

Suppose you wish to suppress the normal error message and continue program execution in-line whenever an arithmetic overflow occurs during your program. (In arithmetic overflow, the result of a calculation is too large to fit in the field in which it is to be entered.) Enter N in the Error Log Field and 0 in Column 65 (under Message No. 11) of the Error Response Field (Figure 3-1, Items 15 and 16).

3-20. Program Name (Columns 75 through 80)

This field contains the program name, discussed in Paragraph 2-6.

3-21. DEFAULT SUMMARY

If you leave all fields but Form Type (Column 6, the only required entry) blank, the Control Record Specification options shown in Table 3-7 apply.

Table 3-6. RPG-Detected Errors

Error Number	Message	Probable Cause	Recovery Procedure	Error Response Column	*ERROR Code	Error Response Entries
1	FATAL FILE ERROR	Check MPE error in File Information Display or IMAGE error message.	Determine cause of file error and correct the program.	NONE	F	NONE
2	UNIDENTIFIED RECORD, FILE-NAME = xxxx, RECORD NUMBER = nnn	Record number nnn on the input file named xxxx was not identifiable.	Choose one of the error response options. Either include this record type on Input Specifications or delete record from the file.	56	A	0,1,2,3,4,5
3	MATCHING RECORD SEQUENCE ERROR, FILENAME = xxxx, RECORD NUMBER = nnn	The matching record with record number = nnn and filename = xxxx was not in sequence.	Choose one of the error response options and correct the record sequence.	57	B	1,2,3,4,5
4	INPUT SEQUENCE ERROR, FILENAME = xxxx, RECORD NUMBER = nnn	Record number nnn on the input file named xxxx was not in proper sequence.	Rearrange the record in the proper sequence and execute the program again.	58	C	1,2,3,4,5
5	INDEX INVALID, LINE NUMBER = nnn, INDEX = nnn	The array index is 0 or greater than the number of array elements.	Choose one of the error response options and correct the program.	59	D	0,2,3,4,5
6	NEGATIVE SQUARE ROOT, LINE NUMBER = nnn	Your program calculated a square root that was a negative value.	Choose one of the error response options and correct the data.	60	E	0,2,3,4,5
7	TABLE/ARRAY SEQUENCE BAD DATA ERROR, OVERFULL FILE = xxxx, LINE NUMBER = nnn, VALUE = xxx	The table or array on file xxxx with a line number = nnn and value = xxx had a sequence, bad data, or overfull error.	Choose one of the error response options and correct the table or array.	61	H	0,2,3,4,5
8	RECORD NOT FOUND, FILENAME = xxxx, STATUS = xxxx	The record sought was not found (STATUS portion appears only for IMAGE file).	Choose one of the error response options and make the appropriate changes.	62	U	0,2,3,4,5
9	SPECIAL FILE ERROR, FILE-NAME = xxxx, STATUS = xxxx	The special file processing routine returned an error.	Check the error status and make the appropriate changes.	63	J	0,1,2,3,4,5

Table 3-6. RPG-Detected Errors (Continued)

Error Number	Message	Probable Cause	Recovery Procedure	Error Response Column	*ERROR Code	Error Response Entries
10	RDEXIT ERROR, FILENAME = xxxx, STATUS = xxxx	The Read Exit file processing routine returned an error.	Choose one of the error response options and make the appropriate changes.	64	J	0,1,2,3,4,5
11	ARITHMETIC OVERFLOW, LINE NUMBER = nnn, VALUE = nnn	The arithmetic result exceeded the number of digits the field could hold.	Correct the program (make the field size larger).	65	N	0,2,3,4,5
12	DIVIDE BY ZERO, LINE NUMBER = nnn	The number was divided by zero.	Choose one of the error response options and correct the program.	66	O	0,2,3,4,5
13	INVALID NUMERICAL DATA, LINE NUMBER = [nnn, VALUE = nnn, RECORD NUMBER = nnn, COLUMN NUMBER = nnn], or [VAL at DB + xxx ($\frac{L}{R}$)]	Either an input field or an alphanumeric field, being moved to a numeric field, contained "special" characters or embedded blanks.	Choose one of the error response options and correct the program.	67	P	0,2,3,4,5
14	BINARY CONVERSION OVERFLOW, LINE NUMBER = nnn, VALUE = nnn	The binary output field was not large enough to hold the value.	Define a larger binary field.	68	Q	0,2,3,4,5
15	INDICATOR H X IS ON (H0-H9)	These Halt indicators were either set on programatically or the RPG program has set on H0 because of a file error.	Choose one of the error response options and correct the program.	69	NONE	0,2,3,4,5
16	REL REC# INVALID, FILE NAME = xxxx	An attempt was made to read or write a relative record number past the end of the file or before the beginning of the file.	Choose one of the error response options and correct the program.	70	M	0,2,3,4,5
17	DUPLICATE KEY, FILENAME = xxxx, KEY = kkkk	The program attempted to write a record having the same key as a record already in the file, but duplicate keys are not allowed.	Re-define the file to permit duplicate keys, or delete one of the records.	71	Y	0,2,3,4,5

Table 3-7. Control Record Specification Default Options

Columns	Field	Default Specifications
1-5	Sequence Number	No sequence number applies.
7-14	Error Dump Filename	Error dump directed to the standard list device (\$STDLIST).
15	Debug	Suppress all DEBUG operations.
16	USWITCH Source	Initialize from job stream USWITCH records or by interactive prompting.
17	UPDATE Source	Initialize from the system date.
20	Line Number Option	Line number option applies.
21	Inverted Print	Domestic print format applies.
22	Record Number Adjust	No adjustment to be made to relative record numbers.
25	Program Name Logging	Do not print program identification lines.
26	Alternate Collating Sequence	Normal ASCII Collating Sequence applies.
28	BUFCHK Defaults	No defaults.
34	Table/Array Look-Up	Sequential Look-Up applies.
39	EBCDIC Zone/Digit tests	Use ASCII representation for record identification testing.
40	Sign Process	Do not force signs.
41	Form Positioning	Do not print forms-alignment record nor await verification of proper form position by operator.
42	Indicator Setting	Set only the 1P and L0 Indicators on.
43	File Translation	Do not translate files.
44	Non-numeric digits	Do not allow non-numeric data to be moved into numeric fields.
46	Dollar Sign Substitute	Do not substitute any other character for the dollar sign in edit words or codes.
47	Skip Suppress	Skip to Channel 1 on the printer's carriage control tape, so that output begins on a new page.
48	DSPLY Options	Print the "DSPLY" literal on each use of DSPLY and enter new Result Field value directly below display of the old value.
49	Record Length Check	For each "Record Length" error, display a warning message and continue program execution.
50	Page Overflow Test	Overflow when overflow line is reached or passed.
51	*PLACE Method	Repeat all preceding fields, including those produced by *PLACE entries.
52	Cross-Reference Listing	Do not print a Cross-Reference Listing.
53	Carriage Control Type	Skip requests refer to channel numbers on carriage control tape.
54	Textfile Sequence Check	Do not sequence-check textfile.
55	Error Log	When a run-time error occurs, and a pre-selected response is specified in the Error Response Field, log the error on the standard list device. Otherwise, send message to operator.
56-71	Error Response	Send error message to operator; no alternate response is required.
75-80	Program Name	Assign program name RPGOBL (unless a \$CONTROL NAME= was specified).

FILE DESCRIPTION SPECIFICATIONS

SECTION

IV

RPG/3000 programs read all input from, and write all output to, collections of information called files, stored on various peripheral devices. For instance, a payroll processing program might accept input from a file that contains wage and salary information for all employees on the company payroll; this program might also write new output to this same file during updating operations. Within a file, all information is organized into units of related data called logical records that are similar in form, purpose, and general content. For example, in the payroll file, each logical record could contain the wage and salary data related to a particular employee; there would be one record for each employee. Within each record, individual items of data are called fields. As an example, a payroll record for an employee might contain a field for each of the following items: the employee's name, social security number, marital status, gross pay, tax exemptions, individual deductions, total deductions, and net pay.

Your RPG/3000 program can reference any number of files, stored on media such as cards, magnetic tape, or disc. For each file (except those containing compile-time tables and arrays, discussed in Section V), you must write a File Description Specification on the Control Record and File Description Specification Sheet (Figure 4-1). Since every RPG/3000 program uses at least one file, you must enter at least one of these specifications as part of your program. On each specification, you define such information as:

- Name of the file.
- Type of file — for instance, input, output, or both.
- Size of the logical records in the file.
- Format of the records in the file — fixed-length or variable-length.
- Class name of the device on which the file resides, used to designate cards, disc, magnetic tape, and so forth.

You may define special additional information for the file by continuing the File Description Specification onto the next line of the sheet. This information and the Continuation Lines on which it appears, are described later in this section.

Note: Recall that the notation Required following a field name in this manual indicates that the field requires an entry.

4-1. FIELDS

The RPG/3000 File Description Specification contains the fields described below:

4-2. Sequence Number (Columns 1 through 5)

This field contains the source record sequence number, described in Paragraph 2-3.

F File Description Specifications

RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS

Page 1 of 4

Author: G SMITH Date: 1/3/75

Job Name: MYPROG

Program Name: MYPROG

Control Record Specification

Record Number	Record Length	Record Type	Record Name	Record Code	Record Position	Record Data	Record Status	Record Date	Record Time
0001	H								

File Description Specifications

File Number	File Name	File Type	File Code	File Position	File Data	File Status	File Date	File Time
0002	CARD1	IPEAF	80		CARDS			
0003	DISKA	ISF	64		FDISK	K E R R O R C A R D E R L A B		
0004	MSTR	CSF	4		MDISK			
0005	OUTX	OFF	132		LPRINTER			
0006	DISKB	O F	256		FDISK			
0007	RAFILE	IR F	256 R		FDISK			
0008	TAPE1	IS F	80		TAPE			

Figure 4-1. Control Record and File Description Specification Sheet

4-3. Form Type (Column 6) (Required)

This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
F	File Description Specification.

Because this entry is required and is always the same, it is pre-printed on the specification sheet for your convenience (Figure 4-1, Item 1).

4.4. File Name (Columns 7 through 14) (Required)

You must assign a unique name to each file used by your program, by making an entry in the File Name Field:

Columns 7 through 14 Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.) For an IMAGE/3000 file containing a data set, this entry indicates the data set name.	File name.

Later in the program, you use this same file name on other specification sheets to reference the file.

Note: Files containing compile-time tables and arrays are assigned names on the Table/Array Specification Sheet described in Section V; they must NOT be named on the File Description Specification Sheet.

In addition to naming sequential and direct RPG files in this entry, you can also name files created under the Keyed Sequential Access Method (KSAM/3000), the INDEX/3000, and the IMAGE/3000 subsystems.

KSAM/3000 allows you to create a file that can be processed efficiently by both random and sequential access, on the basis of a primary record key or one of fifteen alternate record keys (fields within records), record numbers, or chronological order (as the records were written to the file). A KSAM file is actually comprised of two files—one containing the data to be accessed, and the other containing a record directory (index) that points to the individual data records; from the standpoint of your program, however, both files are treated as a single file. KSAM files are analogous to the Indexed Sequential Access Method (ISAM) files created and used on other computer systems, such as those manufactured by IBM. KSAM is not available to programs running under control of MPE-C. For information about the structure and details of KSAM files, please see the **KSAM/3000 Reference Manual**.

F File Description Specifications

INDEX/3000 is a subset of IBM's System/3 ISAM. INDEX is only available on the pre-Series II 3000 Computer Systems. (RPG/3000 provides a special interface that supports both INDEX and any user-written routines for handling indexed-sequential files.)

IMAGE consists of a set of programs that create and maintain complex data structures (data bases) and a set of library procedures for accessing, modifying, and reporting on the data in the data bases. Further details about IMAGE appear in IMAGE/3000 Data Base Management Subsystems Reference Manual.

When accessing IMAGE/3000 through RPG/3000, the following restrictions apply:

- Because data is retrieved on a record basis, you must have an access-level equal to or higher than the highest access-level field of the records to be read. (See the IMAGE Reference Manual for a discussion of Access Levels.)
- Because IMAGE/3000 files are accessible only at execution time, the RPG/3000 compiler does not accept source records from an IMAGE/3000 file.

RPG/3000 supports all input modes (except Re-read) of the DBGET operation used by IMAGE. Since some operations in turn imply DBFIND operations, RPG/3000 allows an additional input mode -- reading down a chain as long as the requesting key does not change.

If you are processing an IMAGE file, you may enter in the File Name Field the name of the data set contained in this file; if you do not name the data set here, then you must name it on the DSNAME record (discussed later in this section). In addition, you must name the data base containing this data set on a File Description Specification continuation record.

Note: IMAGE can be used to simulate the sequential read operations of ISAM. A discussion of these specialized IMAGE functions appears in Appendix H. However, the coding required for any general IMAGE operation not related to ISAM is discussed throughout this section of the manual.

EXAMPLES

Assign the name CARD1 to an RPG input file by entering this name in the File Name Field (Figure 4-1, Item 2).

Other examples of valid file names are:

PRINTER
A
TAPE2
D1234567
LIMBO

Note: Examples of entries for all types of files -- including KSAM, INDEX, and IMAGE files -- appear later in this Section and in Section X.

4-5. File Type (Column 15) (Required)

This field specifies the type of file being described.

Column 15 Entry	Meaning
I	Input file.
O	Output file.
U	Update file.
D	Display file.
C	Combined file.

In your RPG/3000 program, you can reference five different types of files. For all of these, you must indicate the type of file in the **File Type Field**. The file type, in turn, determines how your program can use a file.

- **Input Files** contain the source data for your program — the data that the program processes to produce its output. They include pre-execution time (but not compile-time) table and array files, and record-address files. Your program reads records from these files, but cannot write records to them. These files reside on cards, disc, or magnetic tape, or they are entered from a terminal.

Note: You must further define pre-execution time table and array input files and record-address files on the File Extension Specifications Sheet (Section V), and all other input files on the Input Specifications Sheet (Section VI).

- **Output Files** receive the data that your program outputs (prints, punches, or writes). These files also include pre-execution time table and array files. Your program can write records to these files, but it cannot read records from them. Output files reside on disc or magnetic tape, or are printed on terminals or line-printers, or punched on cards.

Note: You must further define pre-execution time table and array output files on the File Extension Specifications Sheet (Section V), and all other output files on the Output Specifications Sheet (Section IX).

If your program is creating a **KSAM** file, you must define the **key file name** and **build parameters** using the **KEYFL** continuation specification. See Paragraphs 4-38 and 10-10A.

- **Update Files** are files from which your program reads a record, updates fields in that record, and writes the altered record back into the same location. These files must reside on disc or a device with the class name **SPECIAL** (as described later in this section).

Note: You must further define all update files on both Input and Output Specifications Sheets.

- **Display Files** are used in conjunction with the **DSPLY** operation on the Calculation Specification Sheet (Section VIII) to perform on-line entry and display of individual data items directly from Calculations. Specification of Device Class Names (in Columns 40 through 46) is optional since **DSPLY** operations never reference an opened Display File to do input and output. In fact, a Display File name (as specified in Columns 7 through 14) can be equated (by an external File Equation) to \$NULL or any other reference, as long as that file reference can be successfully opened. The **DSPLY** operation still works correctly because all input and output is done **directly** to the user's terminal (if used in an interactive session) or to the System Console (if used in a batch job), rather than through an open file.

Note: These files cannot be referenced on Input or Output Specifications Sheets.

F File Description Specifications

- **Combined Files** are used for both input and output. Your program can both read and write records on these files. The difference between combined and update files lies in the output written to them:
 1. The output for a **combined** file consists of new, selected data only.
 2. The output for an **update** file consists of the data read from the file (the input record) plus new, selected data.

You must assign these files to either terminals or combined card reader/punch devices.

Note: You must further describe these files on both Input and Output Specification Sheets.

Do not specify Type D or C for KSAM, INDEX, or IMAGE files.

EXAMPLE

Define a file as an input file by entering I in the File Type Field (Figure 4-1, Item 3).

4-6. File Designation (Column 16)

If a file is an input, update, or combined file, or an output file used for chaining operations, you must further describe it with one of the following entries:

Column 16 Entry	Meaning
P	Primary file.
S	Secondary file.
R	Record address file (input only).
C	Chained file.
T	Pre-execution table/array file (input only).
D	Demand file.
blank	Sequential output or display file.

These entries further qualify the function of a file:

- **Primary Files** are the main files from which programs read their input; only one per program is allowed. They can be input, update, or combined files. During multi-file processing, a program uses its primary file to control the order in which records from all files are processed. In programs that read records from only one file, that file is usually a primary file; this file might otherwise be a chained or demand file. With multi-file processing and no record-matching operation requested, the program always processes its primary file first. (See Section VII for a discussion of processing priority during matching.)
- **Secondary Files** are any input, update, or combined files, other than a primary file, used in multfile processing. There can be any number of these. If your program does not specify record matching, it first processes the primary file followed by the secondary files in the order in which they are specified on the File Description Specification Sheet, until processing is complete.

- **Record Address Files** are sequential input files that designate which records are to be read from another file on disc during random accessing, and the order in which they are to be read. Your program can only specify one such file, further defined on the File Extension Specification Sheet. Such files cannot be referenced on the Input Specification Sheet or have the device class name SPECIAL. Record Address files are often used to control the processing of KSAM/3000, INDEX/3000, and IMAGE/3000 files.
- **Chained Files** are input, output, or update files on disc accessed randomly through chaining fields specified on the Input Specifications Sheet or loaded directly with the CHAIN operation specified on the Calculation Specifications Sheet. (If the CHAIN operation is used, it requests reading of an input or update file, or writing of an output file; update files are written through the EXCPT operation or normal output specifications.) These files cannot reside on devices with the class name SPECIAL.
- **Pre-Execution Table/Array Files** are files that contain tables or arrays read into memory just before your program begins execution. You must further define these files on the File Extension Specifications Sheet.
- **Demand Files** are input or update files processed sequentially by the READ operation code on the Calculation Specification Sheet. Input demand files are accessed **only** through the READ operation. Update demand files are input through the READ operation but output through normal output-selection indicators.

Leave this field blank for ordinary sequential output or display files.

EXAMPLE

Specify a primary input file by entering P in the File Designation Field (Figure 4-1, Item 4).

4-7. End-of-File (Column 17)

For an input, update, or combined file, this field specifies whether your program can terminate before it reads all records from this file.

Column 17 Entry	Meaning
E	The program cannot end until it reads all records from this file.
blank	If this field contains E for any other files, the program can end whether or not it reads all records from this file. If this field is blank for all input, update, or combined files, the program cannot end until it reads all records from all of these files.

Normally, if your program performs multifile processing, you desire it to process all input, update, and combined files before terminating. If you do not require this, however, place an E in the End-of-File Field for those files that must be fully-read but leave this field blank for all other files.

F File Description Specifications

As a simple example, consider a program that reads a file of cards representing students attending a special class, matches each card against records in a master file of all students enrolled, updates a year-to-date attendance file, and prints an attendance report stating, for each student, whether or not he attended the class. If not all students attended, the program might complete processing the attendance card file before the master file. Since you would still want the remaining students on the master file to be listed on the report, you would want the program to continue until the master file had been completely read. To permit the program to continue, you could enter E in the End-of-File Field in the specification for the master file and leave this field blank for all other specifications.

If the End-of-File Field is blank for all files, or if it contains an E for all files, all records from every file must be read before the program can end.

Note: The entry E is not allowed for demand, chained, or pre-execution time table/array files.

EXAMPLE

To indicate that your program must continue until all records in the file CARD1 are processed, enter E in the End-of-File Field (Figure 4-1, Item 5).

4-8. Input Sequence Check (Column 18)

In this field, you request sequence-checking to determine whether the records in matching files are in proper ascending or descending order.

Column 18 Entry	Meaning
A or blank	If this is a matching file, check the records for proper ascending order.
D	If this is a matching file, check the records for proper descending order.

On the Input Specifications Sheet discussed in Section VII, you can request your program to compare two or more input, update, or combined files to determine when records or individual fields within records match. In the files to be compared (called **matching files**), records must appear in the same sequence, according to the fields to be matched. You can specify ascending or descending sequence by making an entry in the **Input Sequence Check Field** of the File Description Specifications.

When your program encounters a record that is out of sequence on a matching file, it prints an error message. Depending on the option you selected in the **Error Response Field** of the Control Record Specification, your program then either bypasses this record and continues, or it terminates.

This field applies only to update, combined, and primary and secondary input files. Leave it blank for all other files.

If you do not define matching fields in the Input Specifications for this file, your program ignores the Input Sequence Check Field in the File Description Specifications.

The program uses the normal ASCII Collating Sequence for input sequence checking, unless you have defined an alternate collating sequence or requested the EBCDIC sequence as discussed in Paragraph 3-7.

EXAMPLE

Request sequence-checking of a matching input file based on ascending order, by entering A in the Input Sequence Check Field (Figure 4-1, Item 6).

4-9. Record Format (Column 19) (Required)

This field indicates whether the logical records in the file are written in fixed-length or variable-length format:

Column 19 Entry	Meaning
F	Fixed-length record format.
V	Variable-length record format.

Each file used by your program contains logical records written in either of two formats:

- **Fixed-Length Format**, where all records in the file are the same length, or
- **Variable-Length Format**, where records in the file can differ in length.

All files on unit-record devices (such as card readers and punches, and line printers) and random-access files on tape or disc contain fixed-length records. Thus, a file input from a card reader contains records all 80 characters long; a random-access disc file could contain records all 256 characters long. Sequentially-accessed files on tape or disc contain variable-length records.

Note: Specify fixed-length (F) records for all **IMAGE** Files. Specify variable-length (V) records for all **WORKSTN**, **WORKSTNR** and **WORKSTNC** Files.

EXAMPLE

Specify a fixed-length format for the records in the file CARD1 by entering F in the Record Format Field, (Figure 4-1, Item 7).

4-10. Block Length (Columns 20 through 23)

In this field, you determine the length of the blocks (physical records) read from or written to a magnetic tape or disc file.

F File Description Specifications

Columns 20 through 23 Entry	Meaning
1 through 9999 (ending in Column 23).	The length of the block (physical record) in characters.
blank	No blocking; block and logical record lengths are identical.

Your program reads data from files, and writes data to them, in units known as blocks (or physical records). A block is the basic unit that can be transferred between the device on which the file is stored and main memory. For files on magnetic tape or disc, a block can consist of one logical record or a group of several logical records — for instance, 2, 16, or 256 logical records could be grouped into one block. For tape files, this blocking is normally done to improve program execution time or to conserve file space by reducing the number of interrecord gaps on the tape. To request blocking, make the appropriate entry in the **Block Length Field** as noted below.

For old files on disc, records are blocked according to blocking information in the file label. RPG/3000 ignores entries in the **Block Length Field** for old disc files.

For files on card readers, card punches, line printers, and terminals, each block is identical to each logical record, and its length is determined by the type of device. Thus, each block/logical record read from a card reader consists of one 80-character punched card; each block/logical record written to a line printer consists of one line of print (typically 132 characters). RPG/3000 ignores entries in the **Block Length Field** for files on these devices.

To request blocking for magnetic tape or disc files, enter the block length, in characters, in the **Block Length Field**. Specify any value, up to 9999 characters. For files containing variable-length records, enter the length of the largest blocked combination of logical records you anticipate. The entry must end in Column 23; leading zeros are permitted but not required.

You will enter the logical record length in the next field; on the basis of the block length and logical record length you specify, the RPG/3000 compiler determines the number of logical records to be transferred in each block. This number is called the **blocking factor** and is always an integer.

If you precede the command that executes your RPG program with an MPE/3000 **:FILE** command specifying a different block length for a file (as noted in Section XI), this length overrides that in the **File Description Specification**.

EXAMPLE

For the tape file TAPE1, specify a block length of 240 characters by entering 240 in the **Block Length Field** (Figure 4-1, Item 8).

4-11. Logical Record Length (Columns 24 through 27) (Required)

You must specify the length of the logical records in each file used by your program:

Columns 24 through 27 Entry	Meaning
1 through 9999 (ending in Column 27).	For files containing fixed-length records, the length of each record. For files with variable-length records, the length of the longest record in the file.

If the file contains variable-length records, specify the size of the longest logical record in this file.

For fixed length records, if the actual record length is less than the logical record length, and if Header specification column 49 contains a blank or an "E", the following warning message will be displayed:

"UNIDENTIFIED RECORD . . . ACTUAL < LOGICAL RECORD LENGTH FOR FILE XXXX"

This is detected at the initialization phase of the RPG cycle and so it appears only once.

In addition, if Header specification column 49 contains an "E", runtime error #2 "Unidentified Record" occurs. You may respond or prerespond to this with 0 to 5. If you respond "0", normal RPG processing continues, and the error message is essentially just a runtime warning.

To prerespond to this error, put a 0 to 5 in column 56 of the Header Specification Record.

WARNING: If you respond with a "0" to continue processing, since the actual record length is less than the logical record length, RPG will fill the extra buffer space with zeros (in numeric fields) and blanks (in alphanumeric fields).

If Header specification column 49 contains an "N", no warning message will be printed and no runtime error will occur.

In the Logical Record Length Field, an entry must end in Column 27; leading zeros are permitted but not required.

EXAMPLE

To specify a logical record length of 80 characters for the file TAPE1, enter 80 in the Logical Record Length Field (Figure 4-1, Item 9). Notice that this entry, together with the previous one, implies a blocking factor of 3.

F File Description Specifications

4-12. Processing Mode (Column 28)

This field specifies whether records are processed sequentially from beginning to end, sequentially between specified limits, or randomly.

Column 28 Entry	Meaning
blank	Sequential processing for entire file.
L	Sequential processing between limits.
R	Random processing (by record number or key).

Your program can process files in the following modes:

- **Sequential Mode (Entire File).** In this mode, a sequential file (indicated by a blank, a digit from 1 through 7, or a letter T or C in the **File Organization Field** in Column 32), is processed consecutively in the order in which its records physically reside in the file. A KSAM, INDEX, or IMAGE file (indicated by I or X, S, or M, respectively, in Column 32) is processed sequentially by record key. (A key is a string of characters that identifies an individual record by matching the contents of a field in that record.) The program continues reading the file until it encounters the end-of-file or terminates. Request this access mode by leaving the **Processing Mode Field** (Column 28) blank.
- **Sequential Mode (Between Limits).** In this mode, selected portions of a KSAM, INDEX, or IMAGE file can be processed sequentially by key between limits (boundaries) supplied by the program. The limits can be specified within the program; see the discussion of the SETLL command in Section VIII. Alternatively, they can be supplied in a record address file (RAF). The limits specified need not reflect identical keys in the file because the system searches for approximate keys if identical keys are not found; thus, if the beginning key specified in the RAF is not in the file to be processed, the record in that file with the next highest key is selected. (See Appendix H for special considerations for IMAGE files.) The file to be processed between limits is related to the controlling RAF in the **File Extension Specifications**. Request this mode by entering L in Column 28.
- **Random Mode.** In this mode, KSAM, INDEX, and IMAGE files can be accessed by key or relative record number; direct files can be accessed by relative record number. A KSAM, INDEX, or IMAGE file processed by key is indicated by I or X, S, or M, respectively, in Column 32. The key for accessing records on these files can be obtained from a RAF, chaining operation, or chaining file. Any file referenced by relative record number is indicated by a blank, a digit from 1 through 7, or the letter D in Column 32. The relative record number for accessing this file may be obtained from an ADDROUT file (see below), chaining operation, or chaining file. Request random processing mode by entering R in column 28.

A relative record number is a string of digits that identifies the position of each record relative to the first record in the file. In an IMAGE or INDEX file, the first record is designated Record 1. In a KSAM file, the first record may be designated either Record 0 or Record 1, depending on the option chosen when the file is created. In all other files, the first record is designated Record 0. In such a file, for example, 0, 5, and 15 denote the first, sixth, and sixteenth records.

Note: By entering a 1 in Column 22 of the **Control Record Specification** you indicate that all Relative Record Numbers being used to directly access files begin with Relative Record No. 1. (See the description of **Control Record Specification** Column 22 on page 3-5).

An ADDROUT file is a special type of RAF that contains four-character relative record numbers in fixed-length, binary records; such a file is output by SORT/3000.

When an RPG program processes records randomly under the direction of a RAF, the program reads the file until it reaches the end of the RAF or terminates. When a chaining file is used, the program reads the file until the end of the chaining file is encountered. The RAF and chaining files are related to the files they control in the File Extension Specifications.

EXAMPLE

Specify random processing by entering R in the Processing Mode Field (Figure 4-1, Item 10).

Note: Examples dealing with KSAM, INDEX, and IMAGE files appear in Section X.

Because of their power and flexibility, KSAM and INDEX files have special requirements. In these files, records can be accessed randomly or in ascending sequence by **record key**. As the records are written to the file, KSAM or INDEX builds an index that contains the record keys and their associated addresses. When the records are processed, the index eliminates the need for your program to step through the entire file, record by record, to find the data sought. Instead, the program simply locates the record keys in the index and reads the associated records.

The format of a record key can be alphanumeric, packed numeric, or unpacked data, and is specified in the **Record Address Type Field** (Paragraph 4-14, below). The length of the key field and its starting position are defined in the **Record Address or Key Field Length Field** and the **Key Field Starting Location Field** (Paragraphs 4-13 and 4-17, respectively).

Inactive records in a KSAM or INDEX file are treated as deleted, although they actually remain in the file. These records are identified by a special code (all bits set on) in the first and second positions (columns). They are actually dropped from the file only when the file is re-organized. The program that re-organizes the file effectively reads the data in the file and reloads it onto another file; this automatically removes all records flagged as deleted from the KSAM/INDEX file. In most applications using KSAM/INDEX files, it is normal to frequently encounter records flagged as deleted. Thus, any user program that processes such files consecutively, should include tests to recognize deleted records.

F File Description Specifications

4-13. Record Address or Key Field Length (Columns 29 through 30)

If this input file is a RAF, you must specify the length of the fields containing the relative record numbers of keys in the file; if this input file is a KSAM, INDEX, or IMAGE file, you must indicate the number of positions to be occupied by each record key. Note: If the input file is being controlled by an RAF file the number you indicate here must be the same number defined for the RAF file. Leave these columns blank for a KSAM or MPE file being controlled by an ADDRROUT RAF.

Columns 29 through 30 Entry	Meaning
1 through 99, (ending in Column 30).	For an RAF, length (in characters) of the relative record number or key fields in the RAF; for ADDRROUT files, this is always 4. For a KSAM, INDEX, or IMAGE file, length of the key fields, maximum key length is 99.
blank	This is not an RAF, KSAM, INDEX, or IMAGE file, or this is a KSAM or MPE file being controlled by an ADDRROUT RAF.

This entry must end in Column 30; leading zeros are permitted but not required. If the file is not an RAF, KSAM, INDEX, or IMAGE file, leave this field blank.

EXAMPLE

To specify the length of the relative record address records for an ADDRROUT file, enter 4 (Figure 4-1, Item 11).

4-14. Record Address Type (Column 31)

If this is a direct-access or indexed (KSAM, INDEX, or IMAGE) file whose records are read by way of chaining or a RAF, you must make an entry in the Record Address Type Field.

Column 31 Entry	Meaning
A	This file is a KSAM, INDEX, or IMAGE file processed by alphanumeric keys. (When a numeric field is used as the key, this field is unpacked prior to chaining.)
P	This file is a KSAM, INDEX, or IMAGE file processed by packed decimal numeric keys. Split chaining (described in Section VII) is not allowed with such keys. Note: For an IMAGE FILE, the sign of all numeric input fields is forced to C or D. All numeric result fields specified by Calculation Specifications have a C or D sign. Thus, if the sign of the key field in the IMAGE file is not C or D, a record-not-found error occurs when chaining to that file is specified.
K	This file is a direct (random)-access file processed by using record keys. (If K is specified for a KSAM, INDEX, or IMAGE file, the file is processed using packed decimal numeric keys; the system assumes an entry of P.)
I	This file is a direct-access, KSAM, INDEX, or IMAGE file processed by relative record number through chaining or under direction of a RAF.
blank	This file is a sequential file; or if Column 28 contains R, this is a random file processed by relative record number.

If this file is a KSAM file processed by key, enter A or P. If this file is a direct-access file processed by key, enter K. If this file itself is an RAF, you can either enter I or leave this field blank. If it is a sequential file, leave the field blank.

EXAMPLE

Indicate that a file is processed by an RAF by entering I in the Record Address Type Field (Figure 4-1, Item 12).

4-15. File Organization/Additional I/O Area (Column 32)

This field distinguishes between ADDROUT, KSAM, INDEX, and IMAGE files and all other types of input files, and indicates the number of input/output areas (buffers) available for the file.

F File Description Specifications

Column 32 Entry	Meaning
I X	This is a KSAM file. (The X entry is allowed for compatibility with other RPG systems.) A terminal error occurs at runtime if the file that is opened is not actually a KSAM file.
S	The program will use a special interface to user-supplied routines for handling indexed-sequential access-method files, or to INDEX software optionally supplied by HP.
T	This is an ADDROUT file, with two input/output buffers.
M	This is an IMAGE file; I, X or any other entry defaults to M if IMAGE continuation records are present (as they should be).
1 through 7	This is a sequential or random-access (direct) file; assign the specified number of buffers (1 through 7) for the file. Accepted (but not used) for IMAGE files. The default number of buffers is 2.
blank 8 or 9	This is a sequential or random-access (direct) file; assign two buffers for the file. (Entries 8 and 9 are allowed for compatibility with other RPG systems.)
D	This is a random-access (direct) file; assign two buffers for the file. (This entry is not required to specify direct files, and is available only for compatibility with other RPG systems.)
C	This is a KSAM input only file. The records are read chronologically.

Note that a file referenced by relative record numbers is so indicated by a blank, a digit (1-7), or the letter D in column 32. I, X, S, or M should be used when accessing a file by key value.

You need not enter D to specify a random-access (direct) file. However, RPG/3000 allows this entry so that programs written for other RPG compilers that require it can be compiled on the HP 3000 Computer without changing this field.

ADDROUT files are always assigned two input/output buffers.

RPG supplies a special interface to software other than KSAM for handling indexed sequential access method (ISAM) files. These programs may be user-written ISAM procedures, or the INDEX software optionally-supplied by HP. INDEX accepts standard IBM file definitions and provides all the features of KSAM except as noted below:

1. INDEX does not provide the file-locking capability of KSAM, described later in this section.
2. INDEX uses only one record key, whereas KSAM may use up to 16.

If KSAM is not present in the MPE File Management System when your program encounters an indexed-sequential file of any kind, then this special interface is used. But if KSAM is present when your program encounters an indexed sequential file, KSAM is used unless you explicitly request the special interface; to make this request in such a case, you must enter S in Column 32. In any case, if the special interface is requested but neither a user-supplied procedure nor INDEX is available, the special interface calls a procedure in the RPG Library that prints the following error message:

USER ISAM PROCEDURE NOT SUPPLIED

Further information about user-written ISAM procedures appears in Appendix I.

Note: A summary of the coding combinations for Columns 28, 31, and 32 appears in Table 4-1.

EXAMPLE

Request four buffers for the random-access file MSTR by entering 4 in the appropriate field (Figure 4-1, Item 13).

Table 4-1. Processing Direct Access Storage Files

File Organization (Column 32)	Record Address Type (Column 31)	Processing Mode (Column 28)
Sequential (blank)	Not acceptable (blank)	Complete file (blank)
IMAGE (M)	Alphanumeric key (A) Packed key (P) Record number (I) Record key (K)	Complete file (blank) Within limits (L) Randomly (R)
Direct (D)	Record number (I)	Randomly (R)
Tag (ADDROUT) File (T)	Tag file (blank)	Tag file (blank)
KSAM (I or C)	Alphanumeric key (A) Packed key (P) Record number (I) Record key (K)	Complete file (blank) Within limits (L) Randomly (R)
INDEX(S)	Alphanumeric key (A) Packed key (P) Record number (I) Record key (K)	Complete file (blank) Within limits (L) Randomly (R)

F File Description Specifications

4-16. Overflow Indicator (Columns 33 through 34)

If you plan to use an Overflow Indicator to direct or suppress (condition) output of overflow lines on a printer, you must assign such an indicator to the output file by making an entry in this field.

Columns 33 through 34 Entry	Meaning
OA	Assign the indicator named.
OB	
OC	
OD	
OE	
OF	
OG	
OV	
blank	Assign no indicator.

You determine the lines on which your RPG program output normally begins and ends on a printed page by making entries in the Line Counter Specifications or referring to punched channels on the carriage-control tape that are associated with specific lines. The last line is called the **Overflow Line**. When file output reaches the overflow line, the object program automatically checks to determine if you have assigned an Overflow Indicator to this file and if so, whether the indicator is set on or off. (Overflow indicators are set on or off through Input or Calculation Specifications; they are set on automatically each time the overflow line is reached.) Based on these factors, the program selects one of the following actions.

1. If an indicator is assigned and set on, the program prints special records at the bottom of the current page and/or the top of the next page. (These records are associated with the particular indicator by entries in the Output Specifications.) This feature is normally used to condition the printing of totals and sub-totals at the bottom of a page, or to print headings at the top of the following page.
2. If an indicator is assigned and set off, the program continues its normal output on the line following the overflow line; it does not print the special records nor skip to the next page.
3. If no indicator is assigned, the program skips to the top of the next page (normally associated with Channel 1 of the carriage-control tape) and continues printing its normal output.

An entry in the Overflow Indicator Field applies only to line-printer files or to disc files governed by Line Counter Specifications and ultimately destined for a printer. Only one indicator can be assigned to a file, and this indicator cannot be associated with any other file used by your program. Select one of the eight indicators noted in the box above.

Note: More details on the relationship of Overflow Indicators to Input, Calculation and Output Specifications, detail and total output lines, and page-skipping, appear in Sections VII, VIII, and IX.

EXAMPLE

Assign the Overflow Indicator OC to the file OUTX by entering OC in the Overflow Indicator Field (Figure 4-1, Item 14).

4-17. Key Field Starting Location (Columns 35 through 38)

If your program uses a KSAM file processed by a key field, you must indicate the starting location of that key field within the data records in that file.

Columns 35 through 38 Entry	Meaning
1 through 9999	Starting position of key field.
blank	No key field applies.

Note: This entry is not required for IMAGE or INDEX files; if present, it is accepted but not used.

Although an INDEX file is limited to one key field, a KSAM file may have up to 16 key fields. Through RPG, you may process a KSAM file by any of the keys assigned to it. In your program, the keys are identified by the key field starting location.

An example showing how to specify the key field starting location appears in Paragraph 4-18, below.

4-18. Summary of KSAM/INDEX Field Entries.

KSAM and INDEX files, because of their power and flexibility, require special attention. The primary fields used to describe these files appear in Columns 28 through 38 of the **File Description Specifications**. For these files, other fields as well are governed by certain rules.

- **File Type Field (Column 15):** KSAM and INDEX files can be specified as **input**, **output**, or **update** files by entering **I**, **O**, or **U** respectively.
- **File Designation Field (Column 16):** KSAM and INDEX files can be designated as **primary**, **secondary**, **demand**, or **chained** files by entering **P**, **S**, **D**, or **C**, respectively. (If you specify **R** or **T** in this field for a KSAM file, you must enter a digit 1 through 7 in the **File Organization Field (Column 32)** or leave that field blank; the file is then read in chronological order.)
- **Record Format Field (Column 19):** INDEX files must be written in **fixed-length** record format (requested by entering **F**). KSAM files may be written in **fixed length** format (**F**), or in **variable length** format (requested by entering **V**). KSAM files containing variable-length records addressed by relative number must be addressed by relative **word** number.
- **Block Length Field (Columns 20 through 23):** KSAM and INDEX files may be specified as **blocked** or **unblocked** files, denoted respectively by entering a block length or leaving this field blank. (Since both KSAM and INDEX files must be created through utility routines before they are used, this entry is not required except for sequential output files created by RPG.)

F File Description Specifications

- **File Organization Field (Column 32):** KSAM files must always be identified by entering I (or X) in this field. INDEX files must be identified by entering S.

Note: If Columns 28 through 38 are blank for a KSAM or INDEX file, that file is processed sequentially from beginning to end.

For a KSAM file, if **file type** (column 15) is **Output**, and **file addition** (column 66) is **BLANK**, a **KEYFL** continuation specification should be provided.

EXAMPLES

To specify sequential processing for an entire KSAM file, leave column 28 blank (Figure 4-2, Item 1). In this file, the record with the lowest key is processed first. Then, records with successively higher keys are processed, until the end-of-file is encountered. To designate this a secondary input file, enter I and S in Columns 15 and 16, respectively. (Item 2). To indicate that a ten-position packed key is used for processing, enter 10 and P in Columns 29 through 31 (Item 3). To indicate an indexed file, enter I in Column 32 (Item 4). To show that the key field begins in the fourth position of each input record, enter 4 in Column 38 (Item 5).

To specify sequential processing between limits for another KSAM file, enter L in Column 28 (Item 6). In this file, processing begins at a specified key and continues sequentially until the high key is reached. (These limits are specified by a RAF.) To designate this as a primary input file, enter I and P in columns 15 and 16 respectively. (Item 7). The records are blocked three forty-word logical records per block (Item 8). To specify that a six-position alphanumeric key is used, enter 6 and A in columns 29 through 31 (Item 9). To indicate an indexed file, enter X (or I) in Column 32 (Item 10). To denote that the key field begins in the first position of the record, enter 1 in Column 38 (Item 11).

To specify random processing for a KSAM file, enter R in Column 28 (Item 12). (The keys of the records chosen are supplied in a RAF.) To indicate that this file is a primary update file, enter U and P in Columns 15 and 16. (Item 13). To specify that a two-position alphanumeric key is used for this file, enter 2 and A in Columns 29 through 31 (Item 14). To indicate that the key field begins in the tenth position of the record, enter 10 in Columns 37 through 38 (Item 15).

Further information on applications using KSAM files appears in Section X.

4-19. Extension Code (Column 39)

If your program contains File Extension or Line Counter Specifications for this file, you must enter the appropriate code in this field:

Column 39 Entry	Meaning
E	File Extension Specifications further describe the file.
L	Line Counter Specifications further describe the file.
blank	Neither File Extension nor Line Counter Specifications apply to this file.

HEWLETT-PACKARD **RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS** Page 1 of 3

Programmer **BOWEN, R. A.** Date _____

Program Title **MY P** Program Name **MY P**

Control Record Specification

Sequence Number	Form Type (M)	Design	Line Number Option (N)	Starting Point (D)	End Record (E)	Logical Record Length	Processing Method (L)	Key Field Report Address	Field Length	Record Address Type (A, P, K, D)	File Orig/Mod (O, U, X, S, M, D, T, I, Z)	Overflow Int. (OA, OB, OV)	Key Field Starting Location (0-9999)	Extension Code (E, L)	Device Class Name	File Length (L)	File Address (A)	Extent (E, Z)	File Comp. (U, I, U, B)
0001	H																		

File Description Specifications

Sequence Number	File Name	File Type (I, O, U, C)	Designation (P, S, R, C, I, D)	Input Sequence Check (A, D)	Record Format (P, V)	Block Length	Logical Record Length	Processing Method (L)	Key Field Report Address	Field Length	Record Address Type (A, P, K, D)	File Orig/Mod (O, U, X, S, M, D, T, I, Z)	Overflow Int. (OA, OB, OV)	Key Field Starting Location (0-9999)	Extension Code (E, L)	Device Class Name	File Length (L)	File Address (A)	Extent (E, Z)	File Comp. (U, I, U, B)
0009	FILE1	I	S	F			80			10	P									
0010	FILE2	I	P	F		120	40			6	X									
0011	FILE3	U	P	F			104			2	A									

Figure 4-2. Descriptions for KSAM Files

F File Description Specifications

You must enter E in this field and supply File Extension Specifications if this is a table or array file to be read during program execution, RAF, or chaining file. You must enter L and supply Line Counter Specifications if your program depends upon overflow sensing. Otherwise, leave this field blank; in this case, if your program actually contains a File Extension or Line Counter Specification, the compiler will accept it but will also print a warning message.

EXAMPLE

Indicate that you will supply Line Counter Specifications for the file OUTX by entering L in the Extension Code Field. (Figure 4-1, Item 15).

4-20. Device Class Name (Columns 40 through 46)

This field must contain the device class name associated with the file. (Leading blanks are allowed. If present, they will be removed, the device class name will be left justified in the field, and a warning will be issued.)

Column 40 through 46 Entry	Meaning
Device identifier. (This can consist of up to seven letters, digits, and/or special characters.)	Class name or logical device number of device on which the file resides.
SPECIAL	This is a file on a device that requires a user-defined external subroutine to handle input/output. You supply the name of this routine in the Name of Label Exit Field (Columns 54 through 59), and also furnish the routine itself.
\$STDIN	A special device class; if an input file is declared on \$STDIN, it will be opened as \$STDIN.
\$STDLIST	A special device class; if an output or display file is declared on \$STDLIST, it will be opened as \$STDLIST.
WORKSTN	<p>WORKSTN is a special device class file that provides an interface between RPG and the RPG Screen Interface or V/3000. The file assigned to this device can have any legal file name, but must be specified as an Update or Combined file. (WORKSTNC can be any file type.) The file is usually designated as a demand file, but may be a primary file. For example, an update primary file might be used when an application is limited to a single cycle such as showing a form, reading data entered on the form, and writing data to a file. For most applications, however, the WORKSTN file is specified as update demand. For update demand files, the program stays within the calculation section, and uses demand reads and exception output.</p> <p>There may be only one file per program assigned to the device file WORKSTN. This file must be defined with variable length records.</p> <p>WORKSTN is a special name that tells RPG to use the RPG Screen Interface or the RPG/VPLUS Interface, and is not an actual device class as defined in the system configuration.</p>

blank

Device Class defaults to DISC.

The device identifier entry can be either a device class name or a logical device number. A device class name represents the general class to which the device on which the file resides belongs. For new files, this name permits you to request any device of a general type, such as any disc or any tape unit. A logical device number refers to a specific device, such as a particular printer or tape unit. Device class names and numbers

are determined and assigned during system configuration by the MPE/3000 System Supervisor, or Console Operator, who can tell you what names and numbers are related to which devices at your site.

When you run your program, you can override the name assigned in this field by making an entry in the DEV-field of an MPE/3000 :FILE command referencing the file (See Section XI.)

Note: MPE/3000 permits device class names of up to eight characters, whereas RPG/3000 limits these names to seven. If your program references a file related to an eight-character name (under MPE), use a name of less than eight characters in your program and equate this name to the MPE name with a :FILE command at run-time.

The compiler does not verify that you have entered a valid device class name.

The entry SPECIAL denotes that you plan to supply your own subroutine for handling input/output for the file. This subroutine, for instance, might be a driver for a special device not supported by HP; such drivers are typically written in HP 3000 Systems Programming Language (SPL/3000).

EXAMPLES

To assign the files CARD1, DISKA, MSTR, and OUTX the device class names CARDS, FDISC, MDISC, and PRINTER, respectively, enter these names in the Device Class Name Field (Figure 4-1, Item 16).

4-20A. Interface type (Column 47)

In this field, you specify the WORKSTN Interface to be used by your program.

Column 47 Entry	Meaning
R	This is an RPG Screen Interface WORKSTN file.
C	This is an RPG Screen Interface WORKSTN console file.
blank	This is an RPG/VPLUS Interface WORKSTN file.
V	(Reserved for future use — do not specify this option.)



F File Description Specifications

4-20B. Interface Control (Columns 48-52).

These fields are reserved for entry of special options that are unique to the WORKSTN Interface you are using. Refer to Section XIII for a description of WORKSTN control options for the RPG/VPLUS Interface.

4-21. Disc Labels (Column 53)

In this field, you specify what labels are to be checked on existing disc files or written on new disc files:

Column 53 Entry	Meaning
S or blank	Process standard labels.
E	Process standard labels and either call a user-supplied subroutine to process one user label, or bypass this label if it exists.
2 through 9	Process standard labels and either call a user-supplied subroutine to process the number of user labels specified by this entry or bypass this many labels if they exist.
K	This is a File Description Specification Continuation Record — see Paragraph 4-27.

MPE/3000 always reads or writes standard labels for files on disc during allocation of the devices on which these files reside. (The format and content of these labels are described in the MPE/3000 reference manuals.) In addition to these labels, you can read or write up to nine other labels of your own choosing. To do this, indicate in the Disc Labels Field the number of these user labels that you want to process. Then, indicate in the Name of Label Exit Field, discussed next, the name of the external subroutine you will supply to process the labels. Alternatively, you can direct the program to skip over existing user labels. To do this, enter in the Disc Labels Field the number of labels to skip but leave the Name of Label Exit Field blank. In the Disc Labels Field, the entry E always signifies one user label.

EXAMPLE

To request the processing of one user label, enter E in the Disc Labels Field. (Figure 4-1, Item 17) Then, enter the name of the label-handling subroutine in the Name of Label Exit Field, as shown in the next example.

4-22. Name of Label Exit (Columns 54 through 59)

If your program is to process user labels or files on SPECIAL devices, enter the name of the external subroutine that will handle this function.

Columns 54 through 59 Entry	Meaning
Subroutine name. (This may be a name of up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	The name of the external subroutine that will process the labels or SPECIAL files.
blank	The program processes no user labels or SPECIAL files.

If your program requires processing of user labels for disc files, you must have entered E or a digit (2 through 9) in the Disc Labels Field and you must also place the name of the label-handling subroutine in the Name of Label Exit Field. If you specify an E or an appropriate digit in the Disc Labels Field but leave the Name of Label Exit Field blank, label-processing will not occur. SPL/3000 subroutines that process labels on disc files must read the labels with the FREADLABEL intrinsic and write them with the FWRITELABEL intrinsic.

Although standard labels are not supported for magnetic tape, you can process user labels on tape simply by writing an appropriate subroutine and entering its name in the Name of Label Exit Field. (Do not make an entry in the Disc Labels Field.) SPL/3000 subroutines that process tape labels read them with the FREAD intrinsic and write them with the FWRITE intrinsic.)

If your program requires the processing of a SPECIAL file, you must have entered SPECIAL in the Device Class Name Field and you must also place, in the Name of Label Exit Field, the name of the subroutine that opens and closes this file and performs input/output with it.

If you name an SPL/3000 procedure as the subroutine for handling labels in the Name of Label Exit Field, write the procedure head in the following format, and follow this with the procedure body:

```

PROCEDURE exitname (ptr);
VALUE ptr;
INTEGER POINTER ptr;
.
.
.
(Procedure Body)

```

In this procedure head, exitname is the name of the procedure and ptr is the name of the pointer to the entry in the file table for this procedure. (The file table is an integer array described in Appendix C. Directions for writing SPL/3000 procedures appear in HP 3000 Systems Programming Language.)

F File Description Specifications

If `exitname` refers to a procedure for handling SPECIAL files, write the procedure head in this format:

```
PROCEDURE exitname (ptr, type, returncode);  
VALUE ptr;  
INTEGER POINTER ptr;  
INTEGER type, returncode;
```

In this procedure head, `exitname` and `ptr` have the same meanings noted above; `type` is an INTEGER parameter containing one of the following codes:

```
0 = To read a record  
1 = To write a record  
2 = To close the file  
3 = To open the file
```

The `returncode` parameter denotes a word to which the procedure returns one of the following codes:

```
0 = Normal file-processing occurred.  
-1 = Procedure encountered an error.  
+1 = Procedure encountered an end-file indicator while reading an input file.
```

After your program reads a record, it should place that record in the appropriate buffer so that the record can be processed by RPG in the normal way. This is the record buffer reserved for you by RPG via the Block Length Field of the File Description Specifications. A word-pointer to this buffer is found in the 14th word (13th word relative to Word 0) of the file table.

For special files with carriage controls specified in the Output Specifications (Section IX), the `fparam` entry (Word 36) in the File Table points to a four-word integer array containing the skip-before, space-before, skip-after, and space-after options respectively for the current record.

EXAMPLE

To indicate that a subroutine named `DSKLAB` will be used to process labels on the disc file `DISKA`, enter `DSKLAB` in the Name of Label Exit Field (Figure 4-1, Item 18).

4-23. File Addition (Column 66)

In this field, you specify whether new records output to an existing sequential file are written at the beginning of the file (over-writing any information already there), or are appended to the end of the file (following any previously-written information). You must also use this field to specify the addition of records to a KSAM, INDEX, or IMAGE input or update file.

Column 66 Entry	Meaning
A	Append new records at end of sequential or KSAM file, or add new records to INDEX or IMAGE file.
blank	Write records at beginning of sequential or KSAM file; do not add records to INDEX or IMAGE input file; do not add records to any update file.

Normally, your object program opens any sequential output file at its beginning, so that records added to it begin at the head of the file. This, of course, can result in overwriting records previously stored on the file. To preserve such records, you can request that the file be opened immediately following the last record written; any new records will then be appended at this point. Make this request by entering A in the File Addition Field (Column 66) of the File Description Specification. Then, enter ADD in the field of the same name (Columns 16 through 18) on the Output Specifications for all records to be appended. Alternatively, you can avoid making this entry for individual record descriptions simply by specifying an MPE/3000 :FILE command with the ACC=APPEND parameter (Section XI); this results in appending all records described. You can also use the same :FILE command and parameter to override a blank File Addition Field in the File Description Specification.

When you specify the addition of records to a KSAM, INDEX, or IMAGE input or update file, the records may be added in any order, and the key used need not be higher than the highest key on the file.

EXAMPLE

Request appending of new records to the file MSTR by entering A in the File Addition Field (Figure 4-1, Item 19). (Also make the appropriate entry ADD in the File Addition Field of the Output Specification.)

4-24. Extents (Columns 68 through 69)

This field specifies the number of extents required by a new file on disc.

Columns 68 through 69 Entry	Meaning
1 through 15.	Allow the specified number of disc extents.
blank	Allow eight disc extents.

MPE/3000 manages each disc file as a set of extents; each extent is an integral number of contiguously-located disc sectors. All extents (except possibly the last) are the same size. You can specify the maximum number of extents that can be assigned for a new file by entering this number in the Extents

F File Description Specifications

Field; if you make no entry, eight extents are assigned. You can override this entry with the numextents entry in the MPE/3000 :FILE command. For old files, the disc label overrides both the Extents Field specification and the :FILE command. The entry must end in Column 69; leading zeros are permitted.

EXAMPLE

Assign two extents for the file DISKB by entering 2 in the Extents Field (Figure 4-1, Item 20).

4-25. File Conditioner (Columns 71 through 72)

This field specifies any user indicators used to condition this file.

Columns 71 through 72 Entry	Meaning
U1 through U8	Use this file only when the indicator named is set on.
blank	Use this file unconditionally.

Eight user indicators, identified as U1 through U8, are available to allow you to condition various object program operations. You can apply the indicators to any file except Display, WORKSTN, and SPECIAL files to determine whether or not such files can be used by the program during a particular run. For instance, your program may require two files on one run and only one on a subsequent run. You could avoid re-coding your program by specifying a user indicator in the File Description Specification for the optional file, and setting it on to request file usage or off to suppress it. When use is prohibited by an indicator setting, information cannot be read from the affected file or written to it.

The user indicators are external indicators that are set by a special USWITCH record entered at run-time, rather than by an RPG/3000 specification. Although their setting can be altered by the object program, it is the initial setting that determines whether a file is to be used.

You can also use these same indicators in Input, Calculation, and Output Specifications, as described in the sections covering those specifications. The program passes the final settings for all eight user indicators to the system job control word "JCW" when the program terminates. (Refer to section III, 3-3a, for a complete discussion of USWITCH records and the use of "JCW".)

EXAMPLE

Specify that the file DISKB is conditioned by the user indicator U2 by entering the name of this indicator in the File Conditioner Field (Figure 4-1, Item 21).

4-26. Program Name (Columns 75 through 80)

This field contains either the program name or any other information, as discussed in Paragraph 2-6.

4-27. CONTINUATION RECORDS

You can define special additional information for a file by continuing the File Description Specification for that file onto the next line of the specifications sheet. The format of this continuation record is discussed below.

4-28. Sequence Number (Columns 1 through 5)

This field contains the source record sequence number, described in Paragraph 2-3.

4-29. Form Type (Column 6) (Required)

This field contains the Form Type, F, as discussed in Paragraph 4-3.

Note: Columns 7 through 19 and Column 52 of the Continuation Record must remain blank.

4-29A. Long Name Option Target (Columns 20 through 51)

This field is used as a longer alternative to the Option Target field (Columns 60 through 74) when more than 15 positions are needed for the Option Target. Only the DSNAME, ITEM, and ITEMXX Option Types may use this for specification of a longer Option Target - up to 16 characters (in Columns 20-35) maximum. You must enter an asterisk "*" in Column 60 to tell RPG to use the Long Name Option Target field instead of the Option Target field.

4-30. Continuation Code (Column 53) (Required)

This field always contains the following entry:

Column 53 Entry	Meaning
K	Continuation record.

EXAMPLE

Specify that the File Description Specification for the file CARD1 continues onto the next line by entering K in the Continuation Code Field on that line. (Figure 4-1, Item 22).

4-31. File Processing Options (Columns 54 through 74)

You can request various file processing options in Columns 54 through 74, which actually include several groups of fields.

- **General Group** allows you to specify error-handling procedures, exit routines, file locking, ASCII/EBCDIC conversion, and partial field translation for a file.
- **File-Sharing Group** allows you to specify more than one name for a single KSAM, INDEX, or IMAGE file.

F File Description Specifications

- **Data-Base Management Group** allows you to define KSAM files, IMAGE data bases, and IMAGE data sets.
- **WORKSTN Interface Group** Files and fields associated with the RPG Screen Interface and RPG/VPLUS Interface.

4-32. GENERAL GROUP FIELDS. The primary field in this group is the **Option Type Field** (Columns 54 through 59). Certain entries in this field are used in conjunction with those in other fields. For instance, if the characters **ERROR** appear in the **Option Type Field**, (requesting transfer to an error-handling routine if an input/output error occurs), then the name of the routine must appear in the **Option Target Field** (Columns 60 through 65). All related fields are summarized in Table 4-2.

Table 4-2. General Group Fields

Option Type Field		Option Target Field	
Columns 54 through 59 Entry	Meaning	Columns 60 through 65 Entry	Meaning
ERROR	When an input/output error occurs, transfer control to a user-defined external routine that handles the error. (The name of the routine must appear in the Option Target Field.)	Subroutine name. (This may be a name of up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	If ERROR appears in the Option Type Field, the Option Target Field contains the name of the subroutine that handles the error processing.
BYPASS	When an input/output error occurs, bypass the current logical record and increment a special counter by one. (The counter is a five-digit numeric field containing no decimal positions; its name must appear in the Option Target Field.)	Field name. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters or digits.)	If BYPASS appears in the Option Type Field, the Option Target Field contains the name of the field to be used for the error counter.
RDEXIT	Whenever the object program reads a record, pass control to an external routine that determines whether the record is to be bypassed. (The name of this routine must appear in the Option Target Field.)	Subroutine name. (This may be a name of up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	If RDEXIT appears in the Option Type Field, the Option Target Field contains the name of the subroutine that determines if the input record is to be bypassed.
ASCII	The file is in ASCII format. (RPG/3000 does not require this entry for ASCII files but allows it so that programs written for other RPG compilers that do use it can be compiled on the HP 3000.)	(Not applicable.)	(Not applicable.)

Table 4-2. General Group Fields (Continued)

Option Type Field		Option Target Field	
Columns 54 through 59 Entry	Meaning	Columns 60 through 65 Entry	Meaning
EBCDIC	The file is in EBCDIC format. If it is an input file, the object program reads the input and RPG/3000 translates it to ASCII code prior to processing. If it is an output file, the object program creates the output and RPG/3000 translates the output to EBCDIC code before it is written to the file.	P (in Column 60).	If EBCDIC appears in the Option Type Field, a partial field translation is accomplished — packed decimal and binary fields on input and/or output are <i>not</i> translated.
EBCDIK	The file is in EBCDIK format. If it is an input file, the object program reads the input and RPG/3000 translates it to JIS code prior to processing. If it is an output file, the object program creates the output and RPG/3000 translates the output to EBCDIK code before it is written to the file.	P (in Column 60).	If EBCDIK appears in the Option Type Field, a partial field translation is accomplished — packed decimal and binary fields on input and/or output are <i>not</i> translated.
PARTTR	When you are entering a user-defined file translation table that contains packed decimal or binary fields, those fields are not translated — a partial file translation occurs. For EBCDIC file input, use the EBCDIC entry, above.	(Not applicable.)	(Not applicable.)
LOCK	Lock this file during an input or output operation. (Applies only to KSAM and MPE files.)	(Not applicable.)	(Not applicable.)
NOLOCK	Opens this file so that other concurrent users of the file can specify LOCK; however, this option will not dynamically lock and unlock the file. (Applies only to KSAM and MPE files.)	(Not applicable)	(Not applicable.)
BUFCHK	Enable/Disable the three buffer integrity checking options-CDC, NRC, UPC.	N (in column 60) N (in column 61) N (in column 62)	Disables Current Data Checking (CDC). Blank enables. Disables No-Read Checking (NRC). Blank enables. Disables Update-Protect Checking (UPC). Blank enables.
blank	No options are selected.	(Not applicable.)	(Not applicable.)

F File Description Specifications

In the **Option Type Field**, you can use the entries **ERROR** or **BYPASS** to indicate steps to be taken when input/output errors occur. **ERROR** requires that you enter the name of an error-handling subroutine in the

ERROR, BYPASS, and RDEXIT Options

Option Target Field; **BYPASS** requires that you enter the name of an error-counter field in the **Option Target Field**. Furthermore in the **Option Type Field**, you can use the entry **RDEXIT** to determine whether to bypass individual input records; this entry requires you to specify the name of the routine that makes this determination in the **Option Target Field**. If you name an SPL/3000 procedure as an **ERROR** or **RDEXIT** subroutine, write the procedure head in the following format, and follow this with the procedure body:

```
PROCEDURE exitname (ptr, returncode);  
  VALUE ptr;  
  INTEGER POINTER ptr;  
  INTEGER returncode;  
  .  
  .  
  (Procedure Body)
```

In this format, **exitname** is the name of the procedure; **ptr** is the pointer to the pertinent file table for this procedure; **returncode** is a word to which the procedure returns one of the following codes:

- 0 = Process or continue without re-writing the record
- 1 = Bypass or re-write the record
- 2 = Abort the program

(The file table format is discussed in Appendix C. SPL/3000 procedures are discussed in **HP 3000 Systems Programming Language**.)

EXAMPLE

To indicate that an input/output error on the file **CARD1** should result in transfer of control to the routine named **CARDER**, enter **ERROR** in the **Option Type Field** and **CARDER** in the **Option Target Field**. (Figure 4-1, Item 23.)

ASCII, EBCDIC, and EBCDIK Options

You may also use the **Option Type Field** to convert input from the **EBCDIC** file to **ASCII** code, or to convert output to **EBCDIC** code. Do this by entering **EBCDIC** in this field. If packed decimal or binary fields occur on an input file, or if they are written to an output file, the results are unpredictable when these fields are translated from one code to another. To avoid such results, you can request a partial field translation by entering **P** in Column 60 of the **Option Target Field**. Then, the packed decimal and binary fields are not translated, but all other fields are.

PARTTR Option

In a similar way, you can request partial field translation of user-defined file translation tables that contain packed-decimal or binary fields that you do not wish translated. Do this by entering **PARTTR** in the **Option Type Field**. Be careful to define your translation table consistently for all 256 characters — do not equate two characters in one code to a single character in another code; otherwise, the partial field translation may produce improper results.

Note: If your input is in EBCDIC, use the EBCDIC entry in the **Option Type Field**.

LOCK Option (KSAM and MPE Files Only)

The **LOCK** option is used to obtain exclusive access to a file during input/output operations; this temporarily locks the file against other input/output requests each time you access it. You may wish to do this, for instance, when you are running a program concurrently with other programs that update the same file. This would prevent simultaneous update of the same records on the file; no other programs could access the file whenever your program was accessing a record on it.

When the **LOCK** option is specified, RPG opens the file with the dynamic locking facility enabled (to allow shared use of the file with other programs that enable locking) and performs automatic locking and unlocking of the file, for each record, as follows:

- An **input file** is locked before a record is read, and unlocked after it is read.
- An **output file** is locked before a record is written and unlocked after it is written.
- An **update file** is locked before a record is read and unlocked either after it is updated or before the next lock and read.

If an update file has been locked and read but not updated (and thus not unlocked), RPG unlocks the file when the program next attempts to lock and read from it. This allows other programs, which may be waiting for the file, to lock and access it.

NOLOCK Option (KSAM and MPE Files Only)

When the **NOLOCK** option is specified, RPG opens the file with the dynamic locking facility enabled (to allow shared use of the file with other programs that enable locking), but refrains from actually locking and unlocking the file for you (as described for the **LOCK** option above). If you need to lock and unlock your file, you must do so manually by using the **LOCK** and **UNLCK** Calculation operations.

LOCK and NOLOCK on DSNAMES Files

DSNAME continuation records are used to link several file access paths (specified by separate File Descriptions) to a single physical file. Only the first File Description of such a DSNAMES group will be used to determine how the file will be opened at runtime. This determination includes whether or not the dynamic locking facility is to be enabled.

F File Description Specifications

Therefore, to enable dynamic locking for a DSNAME file, you must specify a LOCK or NOLOCK continuation record for the first File Description specification of the DSNAME group. If, instead, you specify LOCK or NOLOCK for a succeeding File Description in the same DSNAME group, the following runtime abort will occur:

```
FATAL FILE ERROR, FILENAME= (Name of file)
KLOCK/KNOLOCK NOT SPECIFIED FOR THIS FILE TO
ENABLE LOCKING FOR OTHER DSNAME'D FILE ACCESS
```

BUFCHK Option

With this option, you specify whether each of the three buffer integrity options (described below) is to be enabled or disabled for the current file. If column 60, 61, or 62 (for CDC, NRC, or UPC, respectively) is blank, then the option **will be enabled**. If the column contains "N", the option will be disabled. Any other entry results in compiler error 286W, 287W, or 289W, and the option defaults to blank (enabled). If no BUFCHK Continuation record is included for the file, then the default values determined by Header column-28 (BUFCHK Defaults) will be used.

Current Data Checking (CDC)

When two or more files named on File Description Specs are equated to the same KSAM or MPE file on disc using MPE FILE commands, RPG opens the file once for each of those File Description Specs. This establishes **multiple access paths** from the program to the file. Since the MPE file system maintains separate data buffers for each of these access paths, output and update operations performed on one access path may not be reflected in the data buffers of the other paths. The posting of data from one access buffer to the physical file could inadvertently overlay data (including records and record pointers) previously posted from one of the other buffers. This could result in corruption of the KSAM or MPE file and/or in the use of incorrect data by the RPG program.

Current Data Checking causes RPG to automatically perform additional I/O operations between the physical file and the data buffers of all the access paths to ensure that all buffers have current data when needed.

This option is used at runtime only for KSAM and MPE files that have the locking facility enabled (that is, LOCK or NOLOCK Continuation records are specified) and that are equated to a common physical file by MPE FILE commands.

No-Read Checking (NRC)

With this option enabled, if the Output specs for an update operation are attempted to be executed prior to a first record being Input for the file, the program will abort with a NO RECORD FOUND runtime error #8, with the message "Attempted update before input of first record."

Update-Protect Checking (UPC)

This option ensures that the Output specs for an update operation will NOT be executed using a "dirty" RPG file buffer. A buffer is considered "dirty" if an Output operation (update or add) has been done, and new data has not yet been Input from the file. The following situations are prevented:

1. Two or more consecutive Update operations with no new Input.
2. For an Update-Add file (File Description spec col-15 = "U" and col-66 = "A") performing an Add to the file between the Input and Output phases of an Update to the same file. That is: Input a record to be updated; Add another record to the file; then format and Update (Output) the original record. Since both the Add and the Update operations use the same RPG buffer, the Add record replaces the Input record data in the buffer, meaning that the Output specs for the update would make modifications on the Add record data (rather than the original Input record) prior to writing out the Update.

Update-Protect Checking prevents the final output of the Update record because the Add operation "dirties" the RPG buffer. Note that if Update-Protect Checking is NOT specified for an Update-Add file, the compiler generates error 281W "FOR UPDATE-ADD FILES W/OUT 'UPDATE-PROTECT CHECK', ENSURE THAT NO UPDATE IS INTERRUPTED BY AN ADD."

If either of the above situations occurs, the program will abort with a NO RECORD FOUND runtime error #8, with the message "Attempted update on same record, or on intervening 'Add' record."

4-33. FILE-SHARING GROUP FIELDS. You may assign the same file name to two or more files by using the related fields shown in Table 4-3. (Records containing these fields are known as DSNAME Records.)

You may wish to process the same file in more than one manner in your RPG program; for instance, you may wish to process it both randomly by relative record number, and sequentially by key. When you do this, you must specify the file twice in your program — once for each type of access, specifying a different name each time. You may equate the file names in two different ways:

1. You may equate the file names by using MPE :FILE commands (Section XI), which causes RPG to open two logically-separate files, each with its own unique file number (assigned by MPE).
2. You may use an RPG DSNAME record to equate the file names, causing RPG to open one logical file (identified by one file number), to which both names apply. In this case, the file is opened only once by RPG and all references to it use the same file number. This allows your program to access the file in several different ways. In doing so, however, you must always remain aware of how different access methods affect the current-record pointers that keep track of your accesses. For instance, your program cannot read a KSAM file sequentially by key and then chain to it by key without altering your sequential current-record pointer. However, the program can read a KSAM file sequentially and chain to it by relative record number without affecting the sequential current-record pointer. A chart showing how the type of access affects the current-record pointer setting appears in Table 4-4.

Table 4-3. File-Sharing Group Fields

DSNAME Field		File Name Field	
Columns 54 through 59 Entry	Meaning	Columns 60 through 74 Entry	Meaning
DSNAME	This is a DSNAME record for file sharing.	Qualified file name. (This can contain from one to 15 characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Override the file name specified in Columns 7 through 14 for the preceding file specification when opening the file; share the file number (assigned by MPE) with all files with the same DSNAME file name. Thus, all references to this file number, by whatever file name, access the same logical file.
		*(in column 60)	Tell RPG that the File Name (up to 16 characters) is specified in Columns 20 through 35 of the Long Name Option Target field.

F File Description Specifications

Table 4-4. Relationship of Access Type to KSAM/RSAM Currency Pointers

Input/Output Type	Current-Record Pointer Setting	Next Update Applies To:
Sequential Read	Record read	Record read
Random Access by Relative Record Number	Not affected	Record read
Random Access by Record Key	Record read	Record read
Write	Record written	Record written
Update	Record updated	Record previously updated. (Each successive update updates same record.)
Delete	Next key sequential record.	Next record
Add	Not affected	Record added

Note: IMAGE differs from the Table 4-4 entries in only one respect: random access of an IMAGE file by relative record number affects the current-record pointer.

If you do not wish different references to the file to affect the current-record pointer, equate the file names by using a :FILE command.

To use the DSNNAME method of assigning multiple names to a single file, enter a DSNNAME record after each description of the file in the File Description Specifications.

To access a single IMAGE data set by multiple access paths, use a KIMAGE and KDSNAME extension entry for each file access. Code the KIMAGE entries with different dummy DB names, and code the KDSNAME entries with the name of the desired data set. Then use file equations to equate all the dummy DB names to the actual data base name. A separate DBOPEN will be executed for each access path to the data base/data set and separate current-record pointers will be established.

EXAMPLE

Suppose that you have a KSAM file that you want to access in two different ways. You must describe the file twice, perhaps naming it FILEA in the first description (Figure 4-3, Item 1) and FILEB in the second description (Item 2). (In this example, FILEA is processed sequentially by key; FILEB is processed randomly by relative record number.) After each description, you also include a DSNNAME record that assigns a common DSNNAME file name to each record; in this case, the DSNNAME file name is FILEK (Item 3). Any other file descriptions that are followed by a DSNNAME record specifying FILEK will also share the same file number.

4-34. DATA BASE MANAGEMENT GROUP FIELDS. In this group you specify information required for KSAM files or IMAGE files used in data-base management operations. The records and entries required to interface with these files through RPG programs are summarized below. Full information about KSAM files and operations appear in **KSAM/3000 Reference Manual**. Full information about IMAGE files and operations appear in **IMAGE/3000 Data-Base Management Subsystem Reference Manual**.

HEWLETT-PACKARD **RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS** Page 1 of 3

Programmer: **GREEN, DOROTHY** Date: **2/7/77**

Program Title: **MYPROG** Project: Program Name: **MYPROG**

Control Record Specification

Sequence Number	File Name	File Type	File Format	File Length	File Organization	File Attributes	File Location	File Status	File Control
1	FILEA	IP	F	80	AI	10	DISC ①		
2	FILEB	IC	F	80R	II		DISC ②		

File Description Specifications

Sequence Number	File Name	File Type	File Format	File Length	File Organization	File Attributes	File Location	File Status	File Control
1	FILEA	IP	F	80	AI	10	DISC ①		
2	FILEB	IC	F	80R	II		DISC ②		
3	FILEA	IP	F	80	AI	10	DISC ①		
4	FILEB	IC	F	80R	II		DISC ②		

Figure 4-3. File Sharing

F File Description Specifications

4-35. Data-Base Name (IMAGE) Records. These records are used to define an IMAGE data base. In addition to the fields common to all File Description Specification continuation records, these records must contain the fields summarized in Table 4-5.

Note: Complete examples of RPG/3000 programs that use IMAGE data sets appear in Section X. Use of IMAGE to simulate ISAM operations is described in Appendix H.

OPEN MODES

The open-mode entry (Column 66) indicates the access-mode in which the file is opened, as follows:

- **Locking Modes, Read/Write Shared Access.** All other programs attempting to read or write to the data base must do so in one of these modes, or in Mode 5 (below). The six locking modes are described in detail in the **LOCKING MODES** section below.
- **Mode 2, Update-Shared Access.** All other programs attempting to update the data base must do so in this mode, or in Mode 6 (below). Your program cannot add or delete records, or perform locking, in this mode.
- **Mode 3, Exclusive Access.** Your program alone must be using the data base in order for this access to succeed. Your program can read, write, and update the data base.
- **Mode 4, Semi-Exclusive Modify Access.** Your program has the same access as in Mode 3, except that programs in Mode 6 can also read the data base.
- **Mode 5, Shared Read Access.** Your program has read-only access to the data base, but concurrent programs running with Mode 1 (Locking Modes) have read/write shared access to it.
- **Mode 6, Shared Read Access.** Your program has read-only access to the data base, but concurrent users must be running in Mode 2, 4, 6, or 8.
- **Mode 7, Exclusive Read Access.** Your program alone must be using the data base for this access to succeed. Your program can read the data base, but cannot write to it.
- **Mode 8, Shared Read Access.** Your program has read-only access to the data-base, but concurrent programs running with Mode 6 or using IMAGE DBSTORE routines also have read-only access to it.

LOCKING MODES

The following special open modes (column 66) allow both automatic and user-controlled locking and un-locking at the data base, data set, and data record levels:

- **Mode B, Data Base Locking for Duration.** The data base is locked for the duration of program execution.
- **Mode S, Data Set Locking for Duration.** The data set is locked for the duration of program execution.
- **Mode 1, Data Base Locking per Record.** The data base is locked and unlocked whenever a record from that data base is accessed.
- **Mode 9, Data Set Locking per Record.** The data set is locked and unlocked whenever a record from that data set is accessed.

- **Mode R, Record Locking.** The specified data record is locked and unlocked when it is accessed. **Input/Output Mode** (column 67) must be 5, 6, C, or R. If not, the data base is opened in **Mode 1** (Data Base Locking per Record). A KITEM continuation record must also be included with this File Description Specification.
- **Mode L, Locking Enabled.** The data base is opened with the dynamic locking facility enabled to allow the user to control all locking and unlocking. No automatic locking/unlocking will be done by RPG. (This mode is equivalent to using the KNOLOCK continuation record for KSAM and other non-IMAGE or INDEX files.)

Modes B and S cause locking to be done at program initialization time and to remain in effect for the duration of program execution.

Modes 1, 9, and R cause locking to be done on each record access as follows:

- An input file (or record) is locked before the record is read, and unlocked after it is read.
- An output file (or record) is locked before the record is written, and unlocked after it is written.
- An update file (or record) is locked before the record is read, and unlocked either after it is updated or before the next lock and read. If the file has been locked and read, but not updated (and thus not unlocked), RPG unlocks it before the next lock and read.

Locking Precedence

If your program defines the same data base or data set more than once, all locking modes specified for that data base or data set must be at the same precedence level as the first locking mode specified.

For example, if a data base is first defined with data base locking for duration and later defined with record level locking, a logic error exists because there is no reason to perform record level locking on the data base when the entire data base is already locked for the duration of the program. The compiler will issue a warning and default the record level locking to the first locking mode specified (data base locking for duration).

The locking modes have the following precedence in relation to each other:

Highest level → B
 Middle level → S
 Lowest level → 1, 9, R, L

When an error occurs because all locking modes are not at the same level as the first locking mode specified for the data base or data set, the compiler will adjust as follows:

- (1) If the first locking mode specified for a data base or data set is of higher precedence than all of the succeeding locking modes specified for the same data base or data set, the compiler will issue a warning and default all lower precedence locking modes to the first mode specified.
- (2) If the first locking mode specified for a data base or data set is of lower precedence than any of the succeeding locking modes specified for the same data base or data set, the compiler will issue a terminal error.

F File Description Specifications

In summary, the rules used by the compiler for multiple definitions of the same data base and/or set are:

At the first occurrence of a uniquely named data base or data set:

- If locking Mode B is specified, ensure that all other definitions of the same data base also specify locking Mode B.
- If locking Mode S is specified, ensure that all other definitions of the same data base and data set specify locking Mode S.
- If locking Mode 1, 9, R, or L is specified, ensure that all other definitions of the same data base and/or data set specify one of the locking Modes 1, 9, R, or L.

Note: If your program is to execute with more than one outstanding lock you must have **Multiple RIN Special Capability**. Extreme caution should be taken to prevent deadlock when executing with Multiple RIN. (See page D-1 of the IMAGE/3000 Reference Manual for an explanation of MR capability.)

Also, see pages 4-14 to 4-21 of the IMAGE/3000 Reference Manual for more information on locking levels and strategy.

INPUT/OUTPUT MODES

The input/output mode (Column 67) defines the order in which records are read from or written to the data base, as follows:

- **Mode 2, Serial Read.** The records are read in the physical sequence in which they appear in the data set. (This is not necessarily the same as the key sequence.)
- **Mode 3, Backward Serial Read.** The same as Serial Read, except that the records are read in reverse physical sequence.
- **Mode 4, Directed Read.** Records are read by relative record numbers. The chain field or record address fields must contain numeric relative record numbers. If no record exists with the requested record number, the H0 indicator is set on. (For a chain operation requested through Calculation Specifications, the error indicator is set on.) The first record in the file is designated Record No. 1 (unlike an MPE/3000 file, where the first record in the file is Record No. 0).
- **Mode 5, Chained Read.** You must provide a key for each record read. If the same key is provided twice, this causes the same record to be read again. A DBFIND operation is done before each record is read. An ITEM continuation record is required for this mode. If you wish to read down a chain, use the **Chained Sequential Read Mode**, below.
- **Mode 6, Backward Chained Read.** Same as Chained Read, except that if more than one of the same key exists, only the last record with this key will be read. An ITEM continuation record is required for this mode. If you wish to read up a chain, use the **Backward Chained Sequential Read Mode**, below.
- **Mode 7, Calculated Read (Formerly Associative Read).** Applies to master data set only. A record is read with a matching key entry.

Table 4-5. Data-Base Name (IMAGE) Record Fields

Record Type Field	Data Base Name Field	Open Mode Field	Input/Output Mode Field
Columns 54 through 59 Entry Meaning	Columns 60 through 65 Entry Meaning	Column 66 Entry Meaning	Column 67 Entry Meaning
IMAGE This is an IMAGE Data-base name record.	Data base name. Name of the data base to which the data set to be accessed belongs. (Maximum of six characters long.)	B, S, 1, 9 R or L* 2 3 or blank 4 5 6 7 8 Update Shared Mode Exclusive Mode Semi-exclusive Modify Access Mode Shared Read Access Mode (Concurrent with Mode 1) Shared Read Access Mode (Concurrent with Mode 4) Exclusive Read Access Mode Shared Read Access Mode (Concurrent with Mode 6)	2 3 4 5 6 7 8 C R S, B blank Serial read. Backward serial read. Directed read. Chained read. Backward chained read. Calculated read. Primary Calculated read. Chained sequential read. Backward chained sequential read. ISAM simulation modes (See Appendix H) Write, Output File.

*Locking Modes (see discussions under OPEN MODES and LOCKING MODES on pages 4-34 through 4-34b)

F File Description Specifications

- **Mode 8, Primary Calculated Read (Formerly Primary Associative Read).** Applies to master data sets only. The record key is used to find the place that a record with this key should occupy, and the record at that location is read. Because of the way that IMAGE/3000 files are written (duplicates can occur in hashing the key), the record retrieved may not have the same key as that requested.
- **Mode C, Chained Sequential Read.** With this type of read, you can use chaining or record address files. With chaining, consecutive duplicate keys result in reading sequentially down the chain. If no more records remain in the chain, the end-of-chain indicator (in Columns 56 through 57 of the CHAIN operation in the Calculation Specifications) will be set on. For input chaining, H0 will be set on. With record address files, each key results in reading the entire chain before the next key is used. An ITEM continuation record is required for the Input/Output Mode.
- **Mode R, Backward Chained Sequential Read.** Same as Chained Sequential Read, except that the chains will be read in reverse sequence.
- **Mode S, Sequential Read.** Used for ISAM simulation (See Appendix H).
- **Mode B, Backward Sequential Read.** Used for ISAM simulation (See Appendix H).
- **(blank entry) Write, Output File.** Data is written to the output file.

Initial Chain Key

The Initial Chain Key entry (columns 68-69) is used to specify the Dummy Chain Key value used for ISAM simulation (See Appendix H).

4-36. **Dummy Item Name (ITEMXX) Records.** These records define Dummy Chain Key names for ISAM simulation (See Appendix H).

4-37. **Item Name (ITEM) Records.** These records define IMAGE item (key) names. They must contain the fields summarized in Table 4-6.

Table 4-6. Item Name (ITEM) Record Fields

Record Type Field		Item Name Field	
Columns 54 through 59 Entry	Meaning	Columns 60 through 74 Entry	Meaning
ITEM	This is an IMAGE item (key) name record.	Item (key) name. *(in column 60)	The name of the key used in referencing this file. Tells RPG that the Item Name (up to 16 characters) is specified in Columns 20 through 35 of the Long Name Option Target field.

This record is required if your program accesses the data via a Data-Base Name (IMAGE) Record that contains any of the following entries in the **Input/Output Mode Field** (Column 67): 5, 6, 7, 8, C, or R.

The entry in the **Item Name Field** (Columns 60 through 74) is the same name as the key named in your IMAGE schema when you defined the data base. However, it need not be the same name used in a CHAIN operation on this file.

4-38. **Password (LEVEL) Records.** These records define passwords that permit your program to access the data base. They contain the fields shown in Table 4-7.

Table 4-7. Password (LEVEL) Record Fields

Record Type Field		Password Field	
Columns 54 through 59 Entry	Meaning	Columns 60 through 67 Entry	Meaning
LEVEL	This is a password record.	Password. (Maximum length of eight characters.)	A password that establishes a user-class identification that permits your program to access the data base.

If no password is provided by the RPG program, then User Class 0 is assumed. (See the IMAGE/3000 Reference Manual for further information.)

Because RPG processes entire IMAGE entries (records) rather than individual fields, your password must include permission to read or write all items in an entry (record).

4-39. **Data Set Name (DSNAME) Records.** These records have a two-fold purpose:

1. They can be used to allow you to process the same data set in more than one manner — for instance, randomly by relative record number and sequentially by key. This requires you to specify the data set as more than one file in your program, and then use DSNAME records to equate these specifications. To do this, you use the same techniques (and DSNAME record fields) described in Paragraph 4-33, under File-Sharing.
2. They can also be used to specify a longer data set name than the eight-character file name allowed in the **File Name Field** (Columns 7 through 14) of the **File Description Specifications** record. This is done by entering the longer name in the **File Name Field** (Columns 60 through 74) of the DSNAME record, which allows names of up to 15 characters, or by entering an asterisk "*" in Column 60 and a 16-character name in Columns 20–35 of the **Long Name Option Target** field. This entry then overrides that in Columns 7 through 14 of the **File Description Specifications** record.

4-40. **Input/Output Status Array (STATUS) Records.** These records define the names of IMAGE status arrays. They contain the fields noted in the Table 4-8.

F File Description Specifications

Table 4-8. Status Array (STATUS) Record Fields

Record Type Field		Array Name Field	
Columns 54 through 59 Entry	Meaning	Columns 60 through 65 Entry	Meaning
STATUS	This is an input/output status array record.	Array name	The name of the array.

In the **Array Name Field**, enter the name of a six-element numeric array with each element containing ten digits and zero decimal positions. This array need not be defined further in File Extension Specifications. If you use this option, then during the Input Cycle, RPG/3000 will not automatically terminate the job on unusual positive errors. (Negative IMAGE/3000 errors are always serious and the program is always terminated immediately when they are detected.) Positive exceptional conditions will set the H0 indicator on. You must then process these errors yourself or let the H0 indicator remain on, in which case the program will end on the next cycle (unless you select the Continue Control Record option). Errors detected during the Output Cycle always cause the program to terminate automatically.

The six array entries are:

1. **Condition word.** This is zero if no exception conditions occurred. (Such conditions are defined in the IMAGE/3000 Reference Manual.) Otherwise, this word contains the IMAGE error number (See Appendix B.)
2. **Length of Record** (in words) read or written.
3. **Record Number** of the record read or written.
4. **Zero** (for read operations on non-primary entries).
Synonym Chain Count (for read operations on primary entries).
Synonym Chain Length (for write operations).
5. **Record Number of Predecessor Entry** in this chain of the current path.
6. **Record Number of the Successor Entry** in this chain of the current path.

For further information, see Section X.

4-40A. Key File Name (KEYFL) Records. These records define the name to be used for the KSAM Keyfile and specify the Duplicate Key Options when a KSAM file is being created by your program. They contain the fields noted in the Table 4-8A.

In the **Key File Name** field, enter the eight character name to be used for the KSAM key file. The name supplied must comply with standard MPE file naming conventions.

In the **Duplicate Key Option** field (column 68), enter a "D" to indicate that duplicate keys are allowed. Leave this field blank if duplicate keys are not to be allowed.

If you have specified that duplicate keys are allowed, you may enter a "C" in the **Chronological Option** field (column 69). Leave this field blank if your file will not have duplicate keys, or if you desire random key additions to the KSAM key file. The random (default) option provides a performance advantage over the chronological option. For a detailed discussion of this topic, refer to **KSAM/3000 Reference Manual**.

Table 4-8A. Keyfile (KEYFL)

RECORD TYPE FIELD		KEY FILE NAME FIELD		FIRST RECORD OPTION FIELD		DUPLICATE KEY OPTION FIELD		CHRONOLOGICAL OPTION FIELD	
Columns 54 through 59		Columns 60 through 67		Column 68		Column 69		Column 70	
ENTRY	MEANING	ENTRY	MEANING	ENTRY	MEANING	ENTRY	MEANING	ENTRY	MEANING
KEYFL	This is a KSAM Key file name record	Key File Name	Name to be given to the KSAM key file.	1	First record 1.	D	Allow duplicate keys.	R	Add duplicate keys randomly to key file.
				0 or blank	First record 0	blank	Do not allow duplicate keys	C or blank	Maintain chronological order of duplicate keys.

4-40B. WORKSTN Interface Group Fields. The primary field in this group is the option type field (Columns 54 through 59). Certain entries in this field are used in conjunction with the option target field (Columns 60 through 74) to specify control parameters, files and field names associated with WORKSTN file processing. All related fields are summarized in Table 4-8b. Further discussion of the RPG/VPLUS Interface Fields can be found in Section XIII.

Table 4-8B. WORKSTN Interface Group Fields

OPTION TYPE FIELD		OPTION TARGET FIELD	
Columns 54 – 59 Entry	Meaning	Columns 60 – 74 Entry	Meaning
FORMDL	The number of forms specified in the option target field determines the size of the VPLUS "form storage directory" used in forms downloading.	3-digit number right-justified in columns 60 – 62. (Value must be in range of 1 – 255.)	Maximum number of forms to be held simultaneously in terminal memory.
FORMS	The file name specified in the option target field is the VPLUS forms file created with Formspec or is the RPG Screen Interface form Library created with RPGSIG.	File Name. (This can be up to 8 characters beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.	File name of the forms library
BATCH	The file name specified in the option target field is the file to which data entered on the form is written during the standard VPLUS data entry operations.	File Name.	File name of the VPLUS batch file.

Table 4-8B. WORKSTN Interface Group Fields (Continued)

OPTION TYPE FIELD		OPTION TARGET FIELD	
Columns 54 - 59 Entry	Meaning	Columns 60 - 74 Entry	Meaning
TRACE	The file name specified in the option target field is the VPLUS trace file which will contain a record for every action or event or runtime error during execution of the RPG/VPLUS interface.	File Name	File name of VPLUS trace file
STATUS	The array name specified in the option target field will be used by the RPG Screen Interface (Not RPG/VPLUS Interface) to return status information. This is automatically defined as a six element, ten digit array with zero decimal places.	Array name (This may be a name up to six characters beginning with a letter or @, \$ or #; the remaining characters can be letters or digits.)	Array in which the RPG Screen Interface status is returned.
TRMID	The field name specified in the option target field is the field which the RPG Screen Interface (Not RPG/VPLUS Interface) will initialize with the terminal identification of the WORKSTN file. This is automatically defined as a two character alphanumeric field.	Field Name. (This may be a name up to six characters beginning with a letter or @, \$, or #; the remaining characters can be letters or digits.)	Field in which the terminal ID is returned.
START	The field name specified in the option target field determines the starting line number for RPG Screen Interface (Not RPG/VPLUS Interface) forms which have a variable starting line number. This is automatically defined as a two-digit numeric field.	Field Name. (This may be a name up to six characters beginning with a letter or @, \$, or #; the remaining characters can be letters or digits.)	Starting line number field.

4-41. DEFAULT SUMMARY

If you leave the optional fields in the File Description Specifications blank, the default specifications shown in Table 4-9 apply:

Table 4-9. File Description Specification Default Values

Columns	Field	Default Specifications
1-5	Sequence Number	No sequence number applies.
16	File Designation	None; the file is a sequential output or display file.
17	End-of-File	The program can end whether or not it reads all records from this file (unless this field is blank for all input, update, and combined records).
18	Input Sequence Check	If this is a matching file, check the records for proper ascending order.
20-23	Block Length	No blocking; block and logical record lengths are identical, unless this default is overridden by file label contents or a :FILE command.
28	Processing Mode	Sequential processing.
29-30	Record Address Field Length	This is not a RAF.
31	Record Address Type	The file is a direct-access file not processed by a RAF, nor a sequential file.
32	File Organization/Additional I/O Area	This is a non-ADDROUT random-access file; assign two buffers for the file.
31-34	Overflow Indicator	Assign no overflow indicator.
39	Extension Code	Neither File Extension nor Line Counter Specifications apply to this file.
53	Disc Labels	Process standard labels.
54-59	Name of Label Exit	The program processes no user labels nor SPECIAL files.
66	File Addition	Write new records at beginning of file.
68-69	Extent	Allow eight disc extents.
71-72	File Conditioner	Use the file unconditionally.
75-80	Program Name	None.



FILE EXTENSION SPECIFICATIONS

SECTION

V

For each table or array file, record address file (RAF), and chaining file used in your program, you must enter a **File Extension Specification**. This allows you to specify such items as:

- Names of tables and arrays, characteristics of entries in these data areas, and names of the files on which they reside.
- Record sequences and chaining fields by which your program processes chained files.
- Names of RAF's and record-number conversion routines for accessing records in them.

Enter these specifications on the **File Extension and Line Counter Specifications Sheet**, shown in Figure 5-1.

5-1. FIELDS

The RPG/3000 File Extension specification contains the fields described below:

5-2. Sequence Number (Columns 1 through 5)


This field contains the source record sequence number, described in Paragraph 2-3.

5-3. Form Type (Column 6) (Required)

This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
E	File Extension Specification

Because this entry is required and is always the same, it is pre-printed on the specification sheet for your convenience (Figure 5-1, Item 1).

HEWLETT  PACKARD RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS

Page 2 of 4

Programmer D. BROWN Date 1/2/75

Program Title PAYRUN

Punching Instructions
 Graphic

 Punch

Program Name

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Seq	Chaining File Code (C) (9)	From Filename	To Filename	Table Array or Routine Name	Entries Per Record (1 999)	Entries Per Table Array (1 9999)	Entry Length (1 256)	Data Format (P,B,L,R)	Decimal Positions (0 9)	Table Array Sequence (A,D)	Table Array Name	Entry Length (1 256)	Data Format (P,B,L,R)	Decimal Positions (0 9)	Table Array Sequence (A,D)	Comments
0009	E	01	C	CHAIN.F	CHAIN.D													CHAINED FILE
	E	03																
	E	04																
	E																	
0010	E					TABLEA	03	0500	10	L2			TABLEB	12	R4			TABLE FILE
	E																	
	E																	
	E																	

Line Counter Specifications

Sequence Number	Form Type (L)	Filename	1		2		3		4		5		6		7		8		9		10		11		12	
			Line Number	Channel Number (O, F)	Line Number	Channel Number (O, L, F)	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
	L																									
	L																									
	L																									
	L																									

Figure 5-1. File Extension and Line Counter Specifications Sheet

5-4. Chaining File Record Sequence (Columns 7 through 8)

If this is a specification for a chaining file, you can enter in this field the sequence in which records in the file should appear. If you make an entry in this field, it should be the same as that appearing in the Group Sequence Field in the Input Specifications (Section VII).

Columns 7 through 8 Entries	Meaning
Two digits (00 through 99) or two letters (A through Z).	The same record sequence specified for the chaining file in the Input Specifications (Section VII).
Blank	None.

Note: RPG/3000 does not actually check or use this entry, but allows it so that programs written for other RPG s that require it can be compiled on the HP 3000 without change.

This field applies only to chaining files. Complete information about these files and their record sequences appears in Paragraph 5-5, 7-6, 7-35, and 7-37.

EXAMPLE

Assign the sequence number 01 to all records in the file CARDS, by entering these characters in the Chaining File Record Sequence Field of the File Extension Specifications. (This entry also appears in the Group Sequence Field of the Input Specifications.) (See Figure 5-2.)

A further example of chaining record sequence numbers appears in Figure 5-1, Item 2.

5-5. Chaining Code Identifier (Columns 9 through 10)

For a chaining file, enter the code identifying the chaining field upon which chaining is based.

Columns 9 through 10 Entry	Meaning
C1 through C9	The code number identifying chaining field, as defined in the Matching or Chaining Field entry in the Input Specifications.
Blank	The file is not a chaining file.

HEWLETT PACKARD RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS Page 2 of 4

Programmer D. BROWN Date 1/12/75

Program Title PGX

Punching Instructions	
Graphic	
Punch	

Program Name PGX

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Seq.	Chaining File Code (C1-C9)	From Filename	To Filename	Table Array or Routine Name	Entries Per Record (1-999)	Entries Per Table Array (1-9999)	Entry Length (1-256)	Data Format (P B L R)	Decimal Positions (0-9)	Table Array Sequence (A-D)	Table Array Name	Entry Length (1-256)	Data Format (P B L R)	Decimal Positions (0-9)	Table Array Sequence (A-D)	Comments
1			01	CARDS	DISK A													

HEWLETT PACKARD RPG INPUT SPECIFICATIONS Page 3 of 4

Programmer D. BROWN Date 1/12/75

Program Title PGX

Punching Instructions	
Graphic	
Punch	

Program Name PGX

RPG INPUT SPECIFICATIONS

Sequence Number	Form Type (I)	File Name	Group Sequence	Number of Records (1-N)	Option (O)	Record Indicator	Look Ahead Trailer	Record Identification Codes									Field Position		Field Name	Control Level (1-15)	Matching or Chaining Field (M1-M9 C1-C9)	Field Reciprocal Relation	Field Indicators															
								Position (1-9999)	Not (N)	Position (C Z D)	Character	Position (1-9999)	Not (N)	Position (C Z D)	Character	Position (1-9999)	Not (N)	Position (C Z D)					Character	From (1-9999)	To (1-9999)	Plus	Minus	Zero or Blank										
1	I	CARDS	01					1																														
2	I							2																														
3	I							3																														
4	I							4																														
5	I							5																														
6	I							6																														
7	I							7																														
8	I							8																														
9	I							9																														
10	I							10																														
11	I							11																														
12	I							12																														
13	I							13																														
14	I							14																														

Figure 5-2. Specifying Chaining Record Sequence

This field applies only to **chaining files**. Such files allow retrieval of records from a **chained file** on disc, based on the content of **chaining fields** in records stored in the chaining file. For instance, suppose that a program reads a card file denoting customer transactions, obtains customer addresses from another file on disc, and prints a bill to be mailed to each customer. If the card file was used as a chaining file, each record in that file could contain a chaining field with a record number pointing to the location containing the customer's address on the chained file on disc.

Your program can identify up to nine chaining fields or field combinations, indicated by the characters C1 through C9. These codes appear in the **Matching or Chaining Field** entry in the **Input Specifications**, opposite the applicable field name. Split chaining (assigning the same code to more than one field) is permitted. Other rules and examples appear in Section VII.

If you wish to use a chaining field in the **Input Specifications**, you must enter the chaining field code in columns 9 through 10 of the File Extension Specifications. The same chaining field code can appear more than once in the File Extension Specification if more than one chaining file chains to the same **chained file**; in this case, the **From Filename** entries associated with the code must be different in each instance, but the **To Filename** entries must be the same. (The **From** and **To Filename** entries are described below.)

EXAMPLE

Assign the chaining field code C1 to the field named CHNFLD in the file CARDS by:

1. Entering C1 in the Chaining Field Code Field on the File Extension Specifications Sheet. (Figure 5-3.)
2. Entering C1 in the Matching or Chaining Field Field in the Input Specifications, opposite the field name CHNFLD. (Also, Figure 5-3.)

A further example of chaining field codes appears in Figure 5-1, Item 3.

5-6. From Filename (Columns 11 through 18)

If this is a chaining, pre-execution time table or array, or record-address file to be read by your program, enter its name in this field.

Columns 11 through 18 Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Name of chaining file used in Input Specification with chaining numbers in Matching or Chaining Field ,
	or
	Name of file from which pre-execution time table or array is read.
	or
	Name of record address file (RAF) containing relative record numbers or record keys to be processed.

HEWLETT PACKARD

RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS

Page 2 of 4

Programmer: D. BROWN Date: 1/12/75
 Program Title: PGNONE

Punching Instructions	
Graphic	
Punch	

75	76	77	78	79	80
PGNONE					

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Seq	Chaining File Code (C I C9)	From Name	To Name	Table Array or Routine Name	Entries Per Record (1-999)	Entries Per Table Array (1-9999)	Entry Length (1-756)	Data Format (P B L R)	Decimal Positions (0-9)	Table Array Sequence (A D)	Table Array Name	Entry Length (1-756)	Data Format (P B L R)	Decimal Positions (0-9)	Table Array Sequence (A D)	Comments	
1				01	CARDS	DISKA													

HEWLETT PACKARD

RPG INPUT SPECIFICATIONS

Page 3 of 4

Programmer: D. BROWN Date: 1/12/75
 Program Title: PGNONE

Punching Instructions	
Graphic	
Punch	

75	76	77	78	79	80
PGNONE					

Sequence Number	Form Type (E)	File Name	Group Sequence	Number of Records (1-9)	Record Description Codes										Field Position	Field Name	Control Level (L U)	Matching or Chaining Fields (M I M9 C I C9)	Field Record Relation									
					1	2	3	4	5	6	7	8	9	0					Plus	Minus	Zeros	Blank						
1		CARDS		0											1	SOCHNIFLD												

Figure 5-3. Specifying Chaining Field Code

Columns
11 through 18
Entry

Meaning

Blank.

This is a **compilation-time table or array** if an entry appears in the **Entries Per Record Field** (Columns 33-35); it is an **execution-time array** if no entry appears in the **Entries Per Record Field**.

Chaining files are described in Paragraph 5-5. Record address files (RAF's) are discussed in Paragraphs 4-12 through 4-15.

Table and array files contain groups of related data arranged for efficient, systematic reference. Tables and arrays differ primarily in the way in which your program searches them and references items within them. Both can be searched sequentially, one at a time, for the item that matches another item that you specify. For example, a program that writes bills for a medical supply company might read each item ordered by individual customers from a card file, search a table on disc for the price of that item (as matched with its part number), and use this value in calculating the bill. In arrays, however, items can be located by directly referencing their position relative to the first item in the array (through indexing). Also, entire arrays can be processed by referencing the array name alone.

The general types of tables and arrays are:

1. **Compile-time tables and arrays**, loaded during compilation and compiled with your source program. (They are loaded in the order specified in the File Extension Specifications.) They become a permanent part of your object program, and can only be permanently altered by re-compiling the program with the revisions you desire.
2. **Pre-execution time tables and arrays**, loaded by the object program before it actually begins the RPG/3000 program cycle. The files containing these tables and arrays are thus available before any other input files are read, and before calculations or output are done.
3. **Execution-time arrays**, loaded during program execution, when they are read as input data (specified in the Input Specifications) or created by calculation operations (specified in the Calculation Specifications). (There are no execution-time tables in RPG.)

The creation and loading of tables and arrays are discussed later in this section. The uses of arrays and methods of accessing them are described in Paragraph 5-39. Compile-time table and array files loaded from disc must also be defined in Table/Array File Name Records, discussed in Paragraph 5-34.

In the **From Filename Field**, all entries must begin in Column 11. When you request reading of more than one pre-execution time table or array from the same file, they are read in the order specified in the File Extension Specifications. For any file, the **From Filename** entry must be identical to that specified in the **Filename Field** in the File Description Specifications.

E File Extension Specifications

EXAMPLE

To assign the name CHAINF to a chaining file, enter this name in the From File Field (Figure 5-1, Item 4).

5-7. To Filename (Columns 19 through 26)

In this field, enter the name of any file related to that referenced by the **From Filename Field**; this can be a chained file, or file processed by a RAF. It can also be an output file for a compile time or pre-execution time table or array.

Columns 19 through 26 Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	The name of the direct-access chained file processed by the chaining file noted in the From Filename Field , or The name of the KSAM, INDEX, IMAGE or direct-access input or update file to be processed by the RAF named in the From Filename Field , or The name of a sequential output file to which the compile-time or pre-execution time table or array named in the Table, Array or Routine Name Field , (Columns 27 through 32) is copied when your program terminates.
Blank.	None. Tables and arrays, if any, are not copied.

This field relates the file specified to the file noted in the **From Filename Field**. You must make an entry in this field if you name a chaining file or RAF in the **From Filename Field**. Otherwise, include this entry only if you want a compile-time or pre-execution time table or array written to a disc or tape file, punched on cards, or printed on a line printer or terminal. Execution-time arrays cannot be output at program termination.

When you name an output file for tables and arrays in this field, all tables and arrays output (except those destined for a printer or governed by Line Counter Specifications) are written in the same format in which they are read. If you specify this output file more than once, the tables and arrays output are separated

by a special record with two asterisks and a blank in Columns 1 through 3. However, alternating tables (described later in this section) are **not** separated by such a record, but are written as alternating fields. If your program writes data to this file during execution, tables and arrays copied at program termination immediately follow this data with no separating records.

Tables and arrays output to a line printer or governed by Line Counter Specifications, are written separately in a special format with a heading identifying each table or array.

You can re-arrange or re-format table or array output by using Output Specifications, discussed in Section IX.

The size of records specified in the File Description Specifications for the file named in the **To Filename Field** must be large enough to accommodate the number of entries per record you will specify in the **Entries Per Record Field** (Columns 33 through 35) of the File Extension Specifications.

EXAMPLE

To assign the name CHAIND to the chained file corresponding to CHAINF, enter this name in the **To Filename Field** (Figure 5-1, Item 5).

5-8. Table, Array, or Routine Name (Columns 27 through 32)

In this field, enter the name of the table or array being described, or the name of a conversion routine for relative record numbers on RAF's or chaining files.

Columns 27 through 32 Entry	Meaning
Table, array, or routine name. (Table names can contain from three to six characters, beginning with TAB; array and routine names can contain from one to six characters, beginning with a letter or one of the special characters @, \$, or =. In both table and array names, the remaining characters can be letters, digits, or @, \$, or =. Embedded blanks are not allowed.	The name of an array or table searched by your program, or The name of a conversion routine used to calculate relative record numbers retrieved from a RAF or chaining file.
Blank.	None.

E File Extension Specifications

If you specify a table or array name, this must be the name used to reference the table or array elsewhere in your program. (Tables are referenced in Calculation and Output Specifications; arrays are referenced in Input, Calculation, and Output Specifications.) The name in this field can apply to any type of table or array--compile time, pre-execution time, and execution time.

Your program can use a conversion routine that hashes keys to relative record numbers from a RAF or chaining file, and converts them to new values to be used in direct-accessing another file. If this is done, you must enter the name of the conversion routine in the **Table, Array, or Routine Name Field**; this entry must be the same as the routine label specified as **Factor 1** of the RPGCV or EXTVCV operation in the Calculation Specifications (discussed in Section VIII). The program calls this routine just prior to obtaining the record. (If a conversion routine is not used, the program takes the relative record numbers directly from the RAF or chaining fields on the chaining file.)

The table, array, or conversion routine name must begin in Column 27. No two tables or arrays can have the same name. If this specification names alternating tables or arrays, this entry must indicate the table or array containing the item found first on the input records read.

EXAMPLE

Assign the name TABLEA to the first table used by a program, by entering this name in the **Table, Array, or Routine Name Field** (Figure 5-1, Item 6).

5-9. Entries Per Record (Columns 33 through 35)

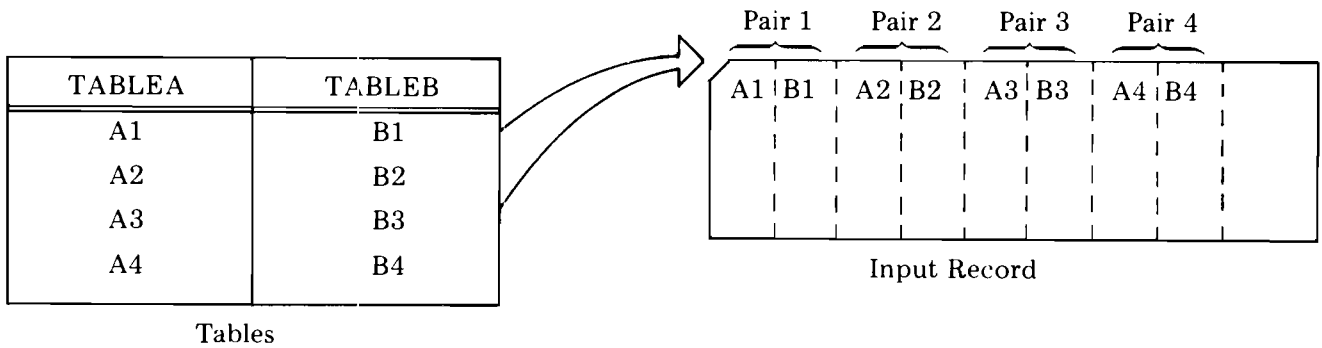
If specifying a compile time or pre-execution time table or array, enter the number of table or array entries in each input record.

Columns 33 through 35 Entry	Meaning
1 through 999	The number of table/array entries in each table or array input record.
Blank.	This is an execution-time array.

In a compile time or pre-execution time table or array, each record except the last must contain exactly the number of entries specified; the last record may contain fewer entries than the number specified, but never more.

Your program can reference a pair of related tables or arrays that are used together. In such cases, each entry item in the first table or array is associated with a corresponding entry in the second table or array. The input records for these two tables or arrays must contain the corresponding entries written in **Alternating form**.

For example, the entries from TABLEA and TABLEB, below, are punched on the same input card; A1 corresponds to B1, A2 corresponds to B2, and so forth.



Each pair of corresponding entries (for example, A1 and B1) must appear on the same record--such a pair cannot be split between records. The Entries Per Record Field applies to each table/array of an alternating pair, so that corresponding entries are treated as one entry. Thus, the input record shown above contains four entries for each table; you would enter 4 in the Entries Per Record Field for this table.

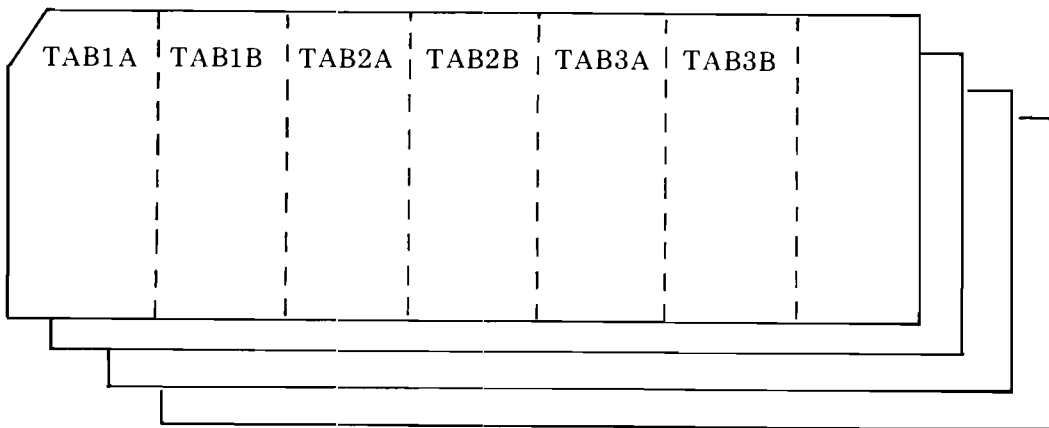
When the Entries Per Record Field is blank, compile time and pre-execution time tables and arrays can only be written at the end of your program if a printer file or file controlled by Line Counter Specifications is named in the To Filename Field (Columns 33 through 35).

The entry in this field must end in Column 35. Leading zeros are permitted but not required.

On input records, comments can follow table or array entries whenever room permits.

EXAMPLE

For the alternating table records shown below, specify the number of entries per record by entering 3 in the Entries Per Record Field (Figure 5-1, Item 7).



E File Extension Specifications

5-10. Entries Per Table/Array (Columns 36 through 39)

In this field, enter the maximum number of entries required by the table or array.

Columns 36 through 39 Entry	Meaning
1 through 9999	The number of entries reserved by the compiler for table or array.
Blank.	This is not a table or array.

When you supply data for all table/array entries available, a **full table or array** results to which no more data can be added. You can however, input less than the maximum amount of data allowed; in this case, not all of the entries contain data, resulting in a **short table or array**. Each short table or array must contain at least one entry. In alphanumeric tables and arrays, unused entries are filled with blanks; in numeric tables and arrays, they are filled with zeroes. The program can add or replace entries during execution by reading them from input records or creating them through calculations.

The entry in the **Entries Per Table/Array Field** can apply to a single table or two related tables in alternating format; in related tables, two corresponding entries are considered as one entry. Entries in this field must end in Column 39; leading zeros are permitted but not required.

Non-alternating tables, when used together in a LOKUP operation, (discussed in Section VIII) should be the same length; if they are not, take care in using them, because LOKUP searching stops at the end of the shorter table or array, and does not complete scanning of the longer one.

EXAMPLE

The alternating tables shown below are used by an insurance company to specify rates for a certain type and amount of coverage by state. Specify the maximum length of these tables as 50 entries each by entering this value in the **Entries Per Table/Array Field** (Figure 5-1, Item 8).

50 Entries (One per State)	TABLEA (State)	TABLEB (Rate)
	ALABAMA	15.00
	ALASKA	17.00
	ARIZONA	14.00
	ARKANSAS	14.50
	WYOMING	18.00

5-11. Entry Length (Columns 40 through 42)

Enter the length of each entry in the table or array.

**Columns
40 through 42
Entries**

Meaning

1 through 256

Entry length (space allowed for each item entered).

Blank.

This is not a table or array entry.



In tables and arrays, all entries must be the same length. The maximum length allowed for alphabetic entries is 256 characters. The maximum length for numeric entries is 15 digits. If you specify (in the next field) ASCII format with leading or trailing signs, the length must include the sign position. For numeric tables or arrays in packed decimal format, enter the unpacked decimal length. For numeric tables or arrays in binary format, enter the number of digits required for storage--for example, a 2-byte field requires 5 digits. For further information on data formats, see Paragraph 7-22.

This entry must end in Column 42; leading zeros are permitted but not required. For related tables or arrays, this entry applies to the table or array named in the **Routine, Table or Array Name Field** (Columns 27 through 32). You will specify the length of the other table or array in the **Entry Length Field** (Columns 52 through 54); that entry will apply to the table or array named in the **Table/Array Name Field** (Columns 46 through 51).

All entries within a table or array must contain the same number of characters. If specific numeric entries differ in length, add leading zeros or blanks to pad them for equal length; if specific alphabetic entries differ in length, add leading or trailing blanks to pad them.

EXAMPLE

Specify that the entries in TABLEA are 10 characters long by entering 10 in the **Entry-Length Field** (Figure 5-1, Item 9).

E File Extension Specifications

5-12. Data Format (Column 43)

This field specifies the data format for compile time or pre-execution time tables or arrays that contain numeric entries:

Column 43 Entry	Meaning
P	Packed decimal format.
B	Binary format.
L	ASCII format with preceding arithmetic (plus or minus) sign.
R	ASCII format with trailing arithmetic sign.
Blank.	ASCII format with no arithmetic sign, or alphanumeric table/array, or execution time array.

For related tables or arrays, the entry applies to the table or array with the first item encountered on an input record.

Leave this entry blank for alphabetic tables or arrays, or execution time arrays.

Data formats are discussed in detail in Paragraph 7-22.

EXAMPLE

To specify that the table named TABLEA contains ASCII data with leading arithmetic signs, enter L in the Data Format Field (Figure 5-1, Item 10).

5-13. Decimal Positions (Column 44)

If this is a specification for a table or array that contains numeric fields, enter the number of digit positions to appear to the right of the decimal point in the fields.

Column 44 Entry	Meaning
0 through 9	This is a numeric table or array with the specified number of decimal positions in each field.
Blank.	This is an alphanumeric table or array.

Numeric fields in tables and arrays do not actually contain decimal points; you must specify where decimal points occur by making an entry in the **Decimal Positions Field**. When these numeric fields are used in calculations, they will be treated as though they contained the decimal point specified. When the fields are printed with edit codes, the decimal points will appear as specified. (Edit codes are discussed in Section IX.)

The number of decimal positions must not exceed the number of digits in the field. For numeric fields with no decimal positions, you must enter 0. Leave the **Decimal Positions Field** blank for tables and arrays containing alphanumeric data. If you are describing two alternating tables or arrays, this entry applies to the table/array containing the item which appears first on the input record.

EXAMPLE

Specify that the entries in the following array are to include decimal points before the two right-most digits, by entering 2 in the **Decimal Positions Field** (Figure 5-1, Item 11):

ARRAY

0000
0100
0150
0200
0250
.
.
.

The entries in this array would be printed as follows if an edit code of 1 were used. (This edit code suppresses leading zeros but prints balancing zeros.)

.00 1.00 1.50 2.00 2.50 . . .

5-14. Table/Array Sequence (Column 45)

In this field, you can request sequence checking of compile time or pre-execution time tables or arrays.

Column 45 Entry	Meaning
A	Sequence check the table or array for ascending order.
D	Sequence check the table or array for descending order.

E File Extension Specifications

Blank Do not sequence check the table or array; high or low LOKUP operations prohibited for unsequenced tables or arrays, and LOKUPs will be sequential.

Entries in tables and arrays can be arranged in ascending or descending order, or in no order at all. In ascending order, the lowest data entry appears first, followed by entries that consecutively increase in value according to the ASCII Collating Sequence. In descending order, the highest data entry appears first, followed by entries that consecutively decrease in value. When you specify an ascending or descending sequence in the **Table/Array Sequence Field**, RPG/3000 checks the table or array during loading to verify that it really is arranged in that order. If the compiler detects a sequence error, processing halts immediately. The computer operator can now direct the compiler to resume processing in spite of the error, or to terminate it completely.

In ascending or descending sequences, RPG/3000 allows two or more consecutive entries of equal value. Thus, in an ascending sequence, you can pad the beginning of a table or array with blanks or zeros with validity.

If you are describing alternating tables or arrays in this specification, the **Table/Array Sequence Field** applies to the table/array containing the entry that appears first on the input record.

Although RPG/3000 does not sequence check execution time arrays, you must enter **A** or **D** in the **Table/Array Sequence Field** if you plan to request LOKUP operations using high or low indicators, or if you desire binary searching. (See Section VIII for details on the LOKUP operation, and Paragraph 3-8 for a discussion of binary searching.)

NOTE: Table/array sequence checking is NOT affected by alternate collating sequences specified in Column 26 of the Control Record Specification.

EXAMPLE

Request table/array sequence checking in ascending order, by entering **A** in the **Table/Array Sequence Field** (Figure 5-1, Item 12).

5-15. Table/Array Name (Columns 46 through 51)

If this specification describes alternating tables or arrays, enter the name of the second table or array input in alternating format with the table or array named in the **Table, Array, or Routine Name Field** (Columns 27 through 32).

Columns 46 through 51 Entry	Meaning
Table or array name, as defined in Paragraph 5-8.	The name of a second alternating table or array.

Blank. The name in Columns 27 through 32 does not apply to an alternating table or array.

Use this field (and the fields in Columns 52 through 57) only if you are describing a second table or array entered in alternating format with the table or array named in Columns 27 through 32. You must describe such pairs of alternating tables and arrays on the same specification line. Both members of the pair must contain the same number of entries per table/array and entries per record. When compile time or pre-execution time tables or arrays in alternating format are loaded, the entries must in fact alternate as the specification implies. A set of related table/array entries cannot be split between two input records.

An execution time array cannot alternate with another array, and must NOT be noted in this field.

The table or array name must begin in Column 46. Other rules governing this field parallel those for the **Table, Array, or Routine Name Field**, discussed in Paragraph 5-8.

NOTE: If you name an alternating table or array in the **Table/Array Name Field** (Columns 46 through 51), you must further describe it in the following fields:

Columns	Fields
52 through 54	Entry Length
55	Data Format
56	Decimal Position
57	Table/Array Sequence

EXAMPLE

To assign the name **TABLEB** to the second alternating table used by your program, enter this name in the **Table/Array Name Field** (Figure 5-1, Item 13).

5-16. Entry Length (Columns 52 through 54)

Enter the length of each entry in the second alternating table or array.

Columns 52 through 54 Entry	Meaning
1 through 256	Entry length (space allowed for each item entered).
Blank	No entry is specified.

This entry must end in Column 54. Other rules governing this field parallel those for the previous **Entry Length Field** (Columns 40 through 42) on this line, applying to the first alternating table or array, discussed in Paragraph 5-11.

E File Extension Specifications

EXAMPLE

To specify that the entries in TABLEB are each 12 characters long, enter 12 in the Entry-Length Field (Columns 52 through 54), (Figure 5-1, Item 14).

5-17. Data Format (Column 55)

This field specifies the data format for a second alternating table or array that contains numeric entries:

Column 55 Entry	Meaning
P	Packed decimal format.
B	Binary format.
L	ASCII format with preceding arithmetic (plus or minus) sign.
R	ASCII format with trailing arithmetic sign.
Blank	ASCII format with no arithmetic sign, or alphanumeric table/array, or no alternating table/array specified.

Rules governing this field parallel those for the previous Data Format Field (Column 43), applying to the first alternating table or array, discussed in Paragraph 5-12.

EXAMPLE

To specify that the table named TABLEB contains ASCII data with trailing arithmetic signs, enter R in the Data Format Field (Figure 5-1, Item 15).

5-18. Decimal Positions (Column 56)

If this is a specification for a second alternating table or array that contains numeric fields, enter the number of digit positions to appear to the right of the decimal point in the fields.

Column 44 Entry	Meaning
0 through 9	This is a numeric table or array with the specified number of decimal positions in each field.
Blank	This is an alphanumeric table or array.

Rules governing this field parallel those for the **Decimal Positions Field** (Column 44), applying to the first alternating table or array, discussed in Paragraph 5-13.

EXAMPLE

Specify that the entries in **TABLEB** are to include decimal points before the four right-most digits, by entering 4 in the **Deciaml Positions Field** (Figure 5-1, Item 16).

5-19. Table/Array Sequence (Column 57)

In this field, you can request sequence checking of a second alternating table or array.

Column 45 Entry	Meaning
A	Sequence check the table or array for ascending order.
D	Sequence check the table or array for descending order.
Blank	Do not sequence check the table or array.

Rules governing this field parallel those for the **Table/Array Sequence Field** (Column 45), discussed in Paragraph 5-14.

EXAMPLE

Request table/array sequence checking for **TABLEB** in ascending order, by entering A in the **Table/Array Sequence Field** (Figure 5-1, Item 17).

5-20. Comments (Columns 58 through 74)

You may use the spaces in Columns 58 through 74 for comments of any kind.

5-21. Program Name (Columns 75 through 80)

This field contains the program name or any other information, as discussed in Paragraph 2-6.

5-22. REQUIRED/OPTIONAL/PROHIBITED ENTRY SUMMARY

In a **File Extension Specification**, certain entries are required, others are optional, and still others are prohibited, depending upon the type of file you are describing. The rules governing these entries for each type of file are summarized in Table 5-1.

E File Extension Specifications

Table 5-1. File Extension Specification Required/Optional/Prohibited Entries

Columns	Field	Chaining File	Record Address File	Compile Time Table or Array File	Pre-Execution Time Table or Array File	Execution Time Array File
7 through 8	Chaining File Record Sequence	Optional	Prohibited	Prohibited	Prohibited	Prohibited
9 through 10	Chaining Field Number	Required	Prohibited	Prohibited	Prohibited	Prohibited
11 through 18	From Filename	Required	Required	Prohibited	Required	Prohibited
19 through 26	To Filename	Required	Required	Optional	Optional	Prohibited
27 through 32	Table, Array, or Routine name	Optional	Optional	Required	Required	Required
33 through 35	Entries Per Record	Prohibited	Prohibited	Required	Required	Prohibited
36 through 39	Entries Per Table/Array	Prohibited	Prohibited	Required	Required	Required
40 through 42	Entry Length	Prohibited	Prohibited	Required	Required	Required
43	Data Format	Prohibited	Prohibited	Optional	Optional	Optional
44	Decimal Position	Prohibited	Prohibited	Required for Numeric Entries; Prohibited for all others.	Required for Numeric Entries; Prohibited for all others.	Required for Numeric Entries; Prohibited for all others.
45	Table/Array Sequence	Prohibited	Prohibited	Optional	Optional	Optional
46 through 51	Table/Array Name	Prohibited	Prohibited	Required for alternating Table/Array.	Required for alternating Table/Array.	Required for alternating Array.
52 through 54	Entry Length	Prohibited	Prohibited	Required for alternating Table/Array.	Required for alternating Table/Array.	Required for alternating Array.
55	Data Format	Prohibited	Prohibited	Optional	Optional	Optional
56	Decimal Position	Prohibited	Prohibited	Optional	Optional	Optional
57	Table/Array Sequence	Prohibited	Prohibited	Optional	Optional	Optional

5-23. DEFAULT SUMMARY

If you leave the optional fields of the File Extension Specifications blank, the default specifications shown in Table 5-2 apply:

Table 5-2. File Extension Specification Default Values

Columns	Field	Default Specifications
1 through 5	Sequence Number	No sequence number applies.
7 through 8	Chaining File Record Number	None.
9 through 10	Chaining Field Number	This file is not a chaining file.
11 through 18	From Filename	This is a compilation-time table or array if an entry appears in Columns 33 through 35; it is an execution time array if no entry appears in Columns 33 through 35.
19 through 26	To Filename	Tables and arrays, if any, are not output.
27 through 32	Table, Array, or Routine Name	None.
33 through 35	Entries Per Record	This is an execution time array.
36 through 39	Entries Per Table/Array	This is not a table or array.
40 through 42	Entry Length	This is not a table or array entry.
43	Data Format	ASCII format with no arithmetic sign, or alphanumeric table/array or execution time array.
44	Decimal Positions	This is an alphanumeric table or array.
45	Table/Array Sequence	Do not sequence-check the table or array; high or low LOKUP is prohibited for unsequenced tables and arrays, and LOKUP will be sequential. SORTA operation assumes ascending sequence.
46 through 51	Table/Array Name	The name in Columns 27 through 52 does not apply to an alternating table or array.
52 through 54	Entry Length	No entry is specified.
55	Data Format	ASCII format with no arithmetic sign, or alphanumeric table/array, or no alternating table/array specified.
56	Decimal Positions	This is an alphanumeric table or array.
57	Table or Array Sequence	Do not sequence-check the table or array.
75 through 80	Program Name	None.

E File Extension Specifications

5-24. RULES FOR USING TABLES AND ARRAYS

In addition to writing descriptions for files that contain tables and arrays, you must also understand how to prepare the tables and arrays themselves, load them into the computer system, and use them in your RPG/3000 program. These operations are explained below.

5-25. Creating Tables and Arrays

You create compile time or pre-execution time tables and arrays by entering them on records residing on files loaded from input devices before your program begins. You create execution time arrays by entering them on records read from input files by your program, or by requesting various calculation operations, during program execution.

5-26. RULES FOR CREATING ALL TABLES AND ARRAYS. When creating any compile time or pre-execution time table, or any compile time, pre-execution time, or execution time array, follow these rules:

1. Ensure that within a particular table or array, each entry (item or element) has the same field length, data type (alphanumeric or numeric), and decimal positions as every other entry.
2. Limit each alphanumeric entry to 256 characters or less.
3. Limit each numeric entry to 15 digits or less.
4. Do not allow any entry to span more than one record. For example you must not continue the last entry on a record onto the next record.
5. Do not allow blanks between entries; all entries must be continuous on each record. (You can, however, embed blanks as **part** of an entry.)
6. Arrange the entries in ascending or descending sequence, or in no sequence at all, as you desire.

5-27. RULES FOR CREATING COMPILE TIME AND PRE-EXECUTION TIME TABLES AND ARRAYS. In addition to the rules in Paragraph 5-26, follow the rules described below when creating any compile time or pre-execution time table or array:

1. Begin the first entry in each record in Position 1.
2. Ensure that all records except the last in the table or array contain the same number of entries; the last record can contain fewer entries than the other records, but it cannot contain more. For example, if the first record contains seven entries, all following records but the last must also contain seven entries; the last record can contain from one to seven entries.

3. Ensure that the table or array to be loaded contains the exact number of entries specified in the **Entries Per Table/Array Field** (Columns 36 through 39) in the File Extension Specifications, or fewer entries. (Tables and arrays in which not all of the entries contain data are called **short tables and arrays**; in numeric tables and arrays, the unused entries contain zeros; in alphanumeric tables and arrays, the unused entries contain blanks. Typically, you create short tables and arrays when you have only a few data entries available initially, but plan to include more items later; you must include at least one entry. Tables and arrays that contain the total number of entries specified are called **full tables and arrays**.)
4. Enter as many entries on each record as you like, in accordance with the **Entries Per Record Field**, (Columns 33 through 35) in the File Extension Specifications and Rule 2 above. This means you can enter from one entry up to the total number required to fill a record. Leave any remaining spaces on the record blank, or fill them with comments.
5. In alternating tables or arrays, begin each record with an entry from the first table or array to be used (specified in Columns 27 through 45 of the File Extension Specifications). Terminate all records, including the last, with an entry from the second table or array to be used (specified in Columns 46 through 57 of the File Extension Specifications).
6. Be certain that the input records containing the table or array entries are fixed-length records that will reside in a sequential (as opposed to random-access) file.

5-28. Defining Tables and Arrays

Be certain to describe all tables and arrays in the File Extension Specifications as discussed earlier in this section. Some of the rules applying to the File Extension Specifications, as well as other specifications, are repeated below for your convenience. (You must also define files containing compile-time tables and arrays in **Table/Array File Name Records**, discussed in Paragraph 5-32.)

5-29. FOR ALL TABLES AND ARRAYS. For any numeric table or array, enter a digit (specifying decimal positions) in the **Decimal Positions Field** (Column 44). Specify the format of the table/array data in the **Data Format Field** (Column 43).

5-30. FOR COMPILE TIME OR PRE-EXECUTION TIME TABLES AND ARRAYS. For tables and arrays loaded at compile or pre-execution time, these rules apply:

1. If you enter A (for ascending sequence) or D (for descending sequence) in the **Table/Array Sequence Field** (Column 45) of the File Extension Specifications, the table or array will be sequence-checked upon loading.
2. If you plan to request the LOKUP operation with high and low indicators in the Calculation Specifications, you must specify a sequence (A or D) in the **Table/Array Sequence Field**. The table or array, of course, must be in the sequence specified.

E File Extension Specifications

3. If you specify a sign to the left or right of the data in the **Data Format Field**, (Column 43), be certain to consider this sign when calculating the entry length that you specify in the **Entry Length Field** (Columns 40 through 42).
4. For table and array files defined in the **File Designation Field** (Column 16) of the File Description Specification, specify fixed-length record format in the **Record Format Field** (Column 19) of the same specification.

EXAMPLES

File Specifications for three arrays appear in Figure 5-4. ARRA is a compile time array containing 10 items (2 per record), each 12 digits long with two decimal places. ARRB is a pre-execution time array, read from the disc file DISKER, containing 300 elements, 10 per record, each 6 positions long. ARRC is an execution time array with 10 alphanumeric elements, each 2 positions long, read from input records.

HEWLETT PACKARD
RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS
Page 2 of 4

Programmer: D. BROWN Date: 1/12/75

Program Title: PROGB

Punching Instructions	
Graphic	
Punch	

75	76	77	78	79	80
Program Name: PROGB					

File Extension Specifications

Sequence Number	Form Type (E)	Changing File Rec. Seq.	Changing File Group (C)	From Filename	To Filename	Table Array, or Routine Name	Entries Per Record (1-999)	Entries Per Table Array (1-9999)	Entry Length (1-256)	Data Format (P, B, L, R)	Decimal Positions (0-9)	Table Array Sequence (A, D)	Table Array Name	Entry Length (1-256)	Data Format (P, B, L, R)	Decimal Positions (0-9)	Table Array Sequence (A, D)	Comments
1	E					ARRA	2	10	12	2								COMPILE TIME
2	E			DISKER		ARRB	10	300	6	0								PRE-EXEC TIME
3	E					ARRC	10	2	0A									EXEC TIME
4	E																	
5	E																	
6	E																	
7	E																	
8	E																	
9	E																	
10	E																	

Figure 5-4. File Extension Specifications for Three Arrays

5-31. Loading Tables and Arrays

Before your program can use tables and arrays, they must be loaded into the computer system as follows:

5-32. **COMPILE TIME TABLES AND ARRAYS.** RPG/3000 compiles all compile time tables and arrays along with your RPG source program; they then become part of the object program. The operating system loads them into main memory when the object program is loaded. (ALTSEQ Records defining alternate collating sequences (Paragraph 3-7) and File Translation Records (Paragraph 3-12) are also loaded in this manner.)

The RPG compiler allows you to load the tables and arrays from records appended to your source program or from files on disc. They can all be loaded from a single file, each from a separate file, or using various combinations of tables/arrays per file, as long as you specify them in the File Extension Specifications in the same order as they appear in the job stream or table/array files. If you have also defined ALTSEQ or File Translation Records, these are loaded in a manner similar to tables and arrays in the order shown below:

1. ALTSEQ (Alternate Collating Sequence) Records.
2. Compile-time tables and arrays, in the order described in the File Extension Specifications.
3. File Translation Records.

In any file that contains more than one table or array, separate these tables/arrays with a record containing two asterisks and a blank in Columns 1 through 3. (This record is a delimiter only, and is not treated as data.)

5-33. Loading from Records Appended to the Source Program. To load ALTSEQ records, tables and arrays, and file translation records from records within your job stream or to append the data to your source program:

1. Place a record containing '**␣' (where ␣ represents a blank) in Columns 1 through 3 after the last record of your source program.
2. After the record containing '**␣', place all ALTSEQ records, if any, as described in Table 3-2 or Table 3-3.
3. After the last ALTSEQ record, place a record containing '**␣' in Columns 1 through 3. (Only do this if there are ALTSEQ records.)
4. Enter your compile-time tables and arrays according to the format described in their Extension specs. Separate each table or array with a record containing '**␣' in Columns 1 through 3.
5. After the last table or array, place a record containing '**␣' in Columns 1 through 3. (Only do this if you have file translation records to append.)
6. Add the file translation records, if any, as described in Table 3-4 or Table 3-5. Separate each new filename with a record containing '**␣' in Columns 1 through 3.

EXAMPLE

A job stream containing an RPG source program plus alternate collating sequence records, four array records, and a file translation record is shown below. All of the data is in the job stream.

```

      .
      .
      .
**␣
ALTSEQ 5A415942
ALTSEQ 5843
**␣
ALPHA
BETA
GAMMA
DELTA
**␣
OUTPUT 58445745
!EOD

```

} Source program

} Alternate collating sequence records

} Compile-time array

} File translation record

} MPE end-of-data command

A job stream containing alternate collating sequence records, two tables, one array, and file translation records is shown in Figure 5-5.

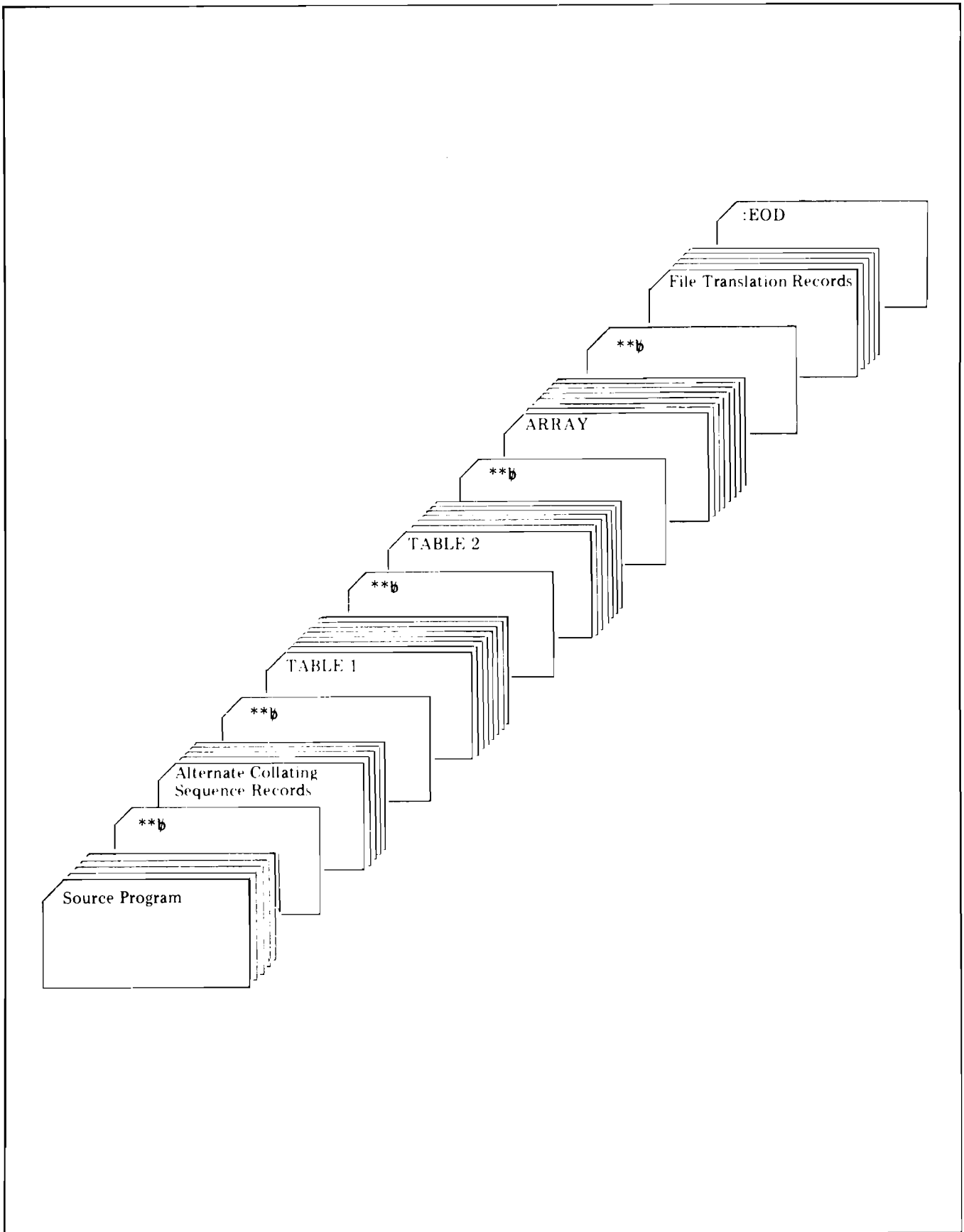


Figure 5-5. Loading Compile-Time Tables and Arrays from Job Stream

5-34. Loading from Disc Files. To load ALTSEQ records, tables and arrays, and file translation records from disc files:

1. Enter the ALTSEQ records, tables and arrays, and file translation records into a disc file. Be sure to use the order specified in Paragraph 5-32 and use the separator records as needed. Do not begin your disc file with a separator record.
2. Place the File Name Records (as described in Table 5-3) after the last specification record in your RPG program. For instance, if your program contains Output Specification records, the File Name Records follow the last of these. Do not place a separator record between the end of your source and the first File Name Record.
3. Enter your source input into the system. When your program is compiled, the compiler loads the ALTSEQ records, tables and arrays, and file translation records as part of your program.

EXAMPLE

A disc file containing three File Name Records is shown below. All of the data is in several different files.

```

.
.
.
} Source program
ATABFILE1
ATABFILE2
ATABFILE3

```

A disc file containing alternate collating sequence records, two tables, one array, and file translation records is shown in Figure 5-6. The Table/Array File Name Record following the program points to the disc file.

Table 5-3. File Name Record Format

Column	Entry	Meaning
1 through 5	C0000 through 99999	Sequence number, as defined in Paragraph 2-3.
6	A	This is a File Name Record.
7 through 14	Valid file name	Name of a file containing one or more compile-time alternate collating sequence records, tables and arrays, and file translation records.
15 through 80	Blank	None

E File Extension Specifications

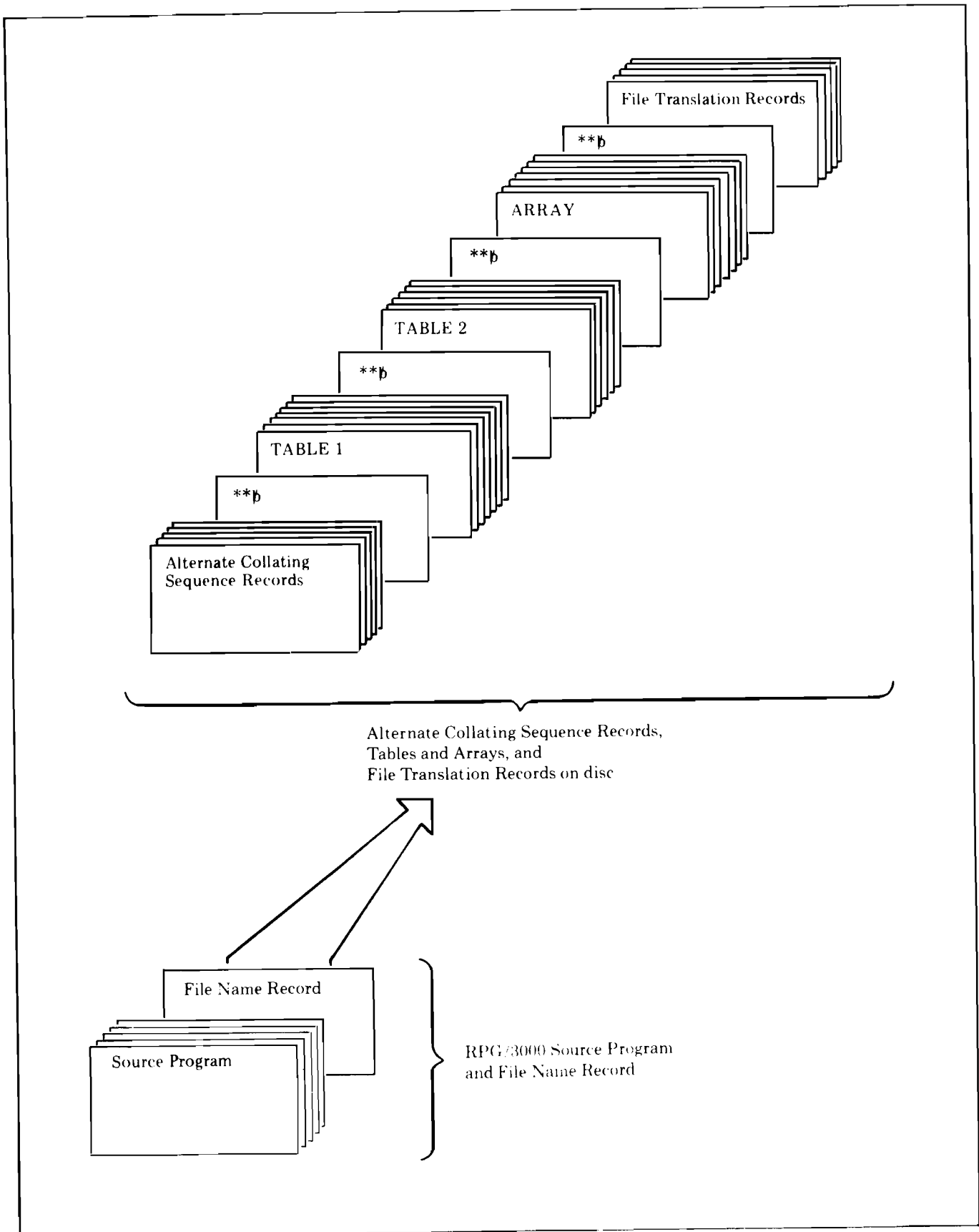


Figure 5-6. Loading Compile Time Data from Disc File

5-35. Loading from Mixed Media. To load ALTSEQ records, tables and arrays, and file translation records both from your source program and from disc files, follow the rules in Paragraphs 5-33 and 5-34. Place the File Name Records after your source program, followed by a separator record, followed by additional ALTSEQ records, tables and arrays, and file translation records.

Make sure that the order of the data records, whether loaded from the source program or from disc is: ALTSEQ records, table and array records, followed by file translation records. When your program is compiled, the data records are loaded in the following sequence:

1. Those on disc, named in File Name Records.
2. Those residing on any source program master-file designated in the MPE command that calls the RPG compiler, as described in Section XI. These ALTSEQ records, tables and arrays, and file translation records must be preceded by a separator record.
3. Those input within your source program textfile, as described in Section XI.

NOTE: If ALTSEQ records are included in a disc file, they must be in the first file named.

Once the compiler detects a separator record in the source program, all further records are read from the source program. If File Name Records are specified after a separator record, they are not recognized as File Name Records; instead, they are treated as data.

EXAMPLE

Shown below is specification of a disc file containing alternate collating sequence records, plus compile-time array records, a file translation records appended to the source program.

.	Source program
.	File containing ALTSEQ records
.	(can also contain compile-time
.ATABFILE	tables or arrays).
**B	Another compile-time array
ABLE	
BAKER	
CHARLIE	
DOG	
**B	File translation record
OUTPUT 58445745	
**B	File translation record
INPUT 59426043	

5-36. PRE-EXECUTION TIME TABLES AND ARRAYS. All pre-execution time tables and arrays used by your program can reside on the same device or on different devices. The operating system loads them in the order they appear in the File Extension Specifications just before the object program begins execution. The object program uses the files containing these tables and arrays like any other data files. In preparing these tables and arrays, follow these rules:

1. If you request sequence-checking for any table or array, be certain that the entries in the last record in the table/array are not all blank. (For proper sequence, the last entry for a table/array in ascending sequence must contain the highest number in the sequence; the last entry for a table/array in descending sequence must contain the lowest number.)
2. Ensure that the table or arrays reside in their input files in the same order as described on the File Extension Specification Sheet; they will be loaded in this order.
3. Follow each table or array in the input file with a record containing two asterisks and a blank in Columns 1 through 3. This record is a delimiter only, and is not treated as data.
4. When regular input data follows tables/arrays on the same file, be sure to place a delimiting record (two asterisks and a blank) between the data and the last table/array.

5-37. EXECUTION TIME ARRAYS. The object program loads execution time arrays by reading them from records in input files or by creating them in memory with calculation operations. In the first case, you must describe the arrays on the Input Specifications Sheet as described in Section VII and summarized below; in the second case, you must enter the appropriate operations on the Calculation Specifications Sheet as discussed in Section VIII.

For arrays loaded through Input Specifications, elements can occupy consecutive positions on the records or can be scattered throughout the records with one or more intervening spaces between items. When the array occupies only one record, consecutive elements are loaded with only one specification (coding line) and scattered elements are each loaded with an individual specification.

For each Input Specification, follow these rules:

1. Enter 1 in the File Type Field (Column 6).
2. Leave columns 7 through 42 blank.
3. Enter P in the Data Format Field (Column 43) if the array is in packed decimal format; enter B if it is in binary format; enter L if it is in ASCII format with leading sign; enter R if it is in ASCII format with trailing sign; otherwise, leave this field blank.
4. Enter the field location of the entire array (if the array is entered as consecutive items), or that of an individual entry (if the items are not consecutive) in Columns 44 through 51. Enter the beginning location in the From Field (Column 44 through 47) and the ending location in the To Field (Columns 48 through 51).

5. If you have specified a decimal point in the **Decimal Positions Field** in the **File Extension Specifications**, enter this same value in the **Decimal Positions Field** of the **Input Specifications (Column 52)**. Otherwise, leave that field blank.
6. Enter the name of the array in the **Field Name Field (Columns 53 through 58)**. This name must agree with the entry made in the **Table, Array, or Routine Name Field** in the **File Extension Specifications**. To define an individual entry or beginning element number in the array, also enter the **index** as described later in this section.
7. Leave Columns 59 through 62 blank.
8. Enter the field record relation indicator, if any is used, in the **Field Record Relation Field (Columns 63 through 64)**. This indicator is described in Section VII.

When array information appears on two or more records, you can define the data in these records collectively or individually in the **Input Specifications** (depending on how it is entered). Use variable indexes and/or record-identifying indicators to avoid overlaying the data from one record with the data from a later record.

Since an RPG object program processes only one record at a time, it cannot process an entire multi-record array until all records containing array information have been read and all array information has been moved into the array fields. For this reason, you may need to suppress calculation and output operations until the entire array has been read into the system. This is done using the indicators described in Section VII.

EXAMPLE

The **File Extension and Input Specifications** for several arrays appear in Figure 5-7. **ARR1** is an array of eight entries (elements), 10 positions each, loaded from a single record from the file **FILEA**. **ARR2** is an array of 5 elements, five positions each, with intervening blanks, loaded from a single record. Note that with **ARR2**, the elements are listed individually on the **Input Specifications Sheet**. **ARR3** is an array of 30 elements, 10 positions each, loaded from five records. The first four array input records each contain seven elements (in columns 3 through 72); the fifth array input record contains two elements (in columns 3 through 22).

5-38. Searching Tables and Arrays

You can direct your program to search tables and arrays for specific entries by requesting the **LOKUP** operation in the **Calculation Specifications (Section VIII)**.

The program searches tables one entry at a time (unless the entries are in sequence and you have requested binary searching) until it finds an entry that matches a specified identifier. It then makes that entry immediately available for use in calculations; when the table name is used as an operand in a calculation operation, the entry most recently found in the **LOKUP** operation on that table is used in the calculation.

You can ensure that an array is sequenced properly for a **LOKUP** operation by using the **SORTA** operation in the **Calculations Specifications**.

Your program can search arrays in the same way it searches tables, or it can find an entry by a specific reference to its location in the array (through indexing).

E File Extension Specifications

HEWLETT PACKARD RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS Page 2 of 4

Programmer D. BROWN Date 1/12/75

Program Title PROGYX

Punching Instructions
 Graphic

--	--	--	--	--	--

 Punch

--	--	--	--	--	--

Program Name

75	76	77	78	79	80
P	R	O	G	Y	X

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Seq	Chaining File Cont. (C1 (9))	From Filename	To Filename	Table Array or Routine Name	Entries Per Record (1-999)	Entries Per Table Array (1-999)	Entry Length (1-256)	Data Format (P/B/L/R)	Decimal Positions (0-9)	Table Array Sequence (A-D)	Table Array Name	Entry Length (1-256)	Data Format (P/B/L/R)	Decimal Positions (0-9)	Table Array Sequence (A-D)	Comments	
1	E					ARR 1		8	10										
2	E					ARR 2		5	5										
3	E					ARR 3		30	10										

HEWLETT PACKARD RPG INPUT SPECIFICATIONS Page 3 of 4

Programmer D. BROWN Date 1/12/75

Program Title PROGYX

Punching Instructions
 Graphic

--	--	--	--	--	--

 Punch

--	--	--	--	--	--

Program Name

75	76	77	78	79	80
P	R	O	G	Y	X

Record Identification Codes

Sequence Number	Form Type (I)	File Name	Group Sequence Number of Records (1/N)	Option (O)	Record Indicator/ Look Ahead/Trailer	Record Identification Codes			Field Position		Decimal Positions (0-9)	Control Level (L1 (9))	Matching or Chaining Fields (M1 M5 C1 (9))	Field Indicators				
						1	2	3	From (1-9999)	To (1-9999)				Plus	Minus	Zero or Blank		
1	I	FILEA	AA	01					1	80								
2	I	FILEA	AA	01					7	5								
3	I	FILEA	AA	01					13	17								
4	I	FILEA	AA	01					19	23								
5	I	FILEA	AA	01					25	29								
6	I	FILEA	AA	01						20								
7	I	FILEA	AA	01					3	72								

Figure 5-7. File Extension and Input Specifications for Arrays

5-39. Using Tables and Arrays

You can read arrays through Input Specifications, operate on tables and arrays through Calculation Specifications, and write them to output files through Output Specifications. Some of the rules governing these functions are summarized below; others appear under the discussions of the appropriate specifications.

When referencing a table in Calculation or Output Specifications, use the table name assigned in the File Extension Specifications; your program will access the most recent entry sought in the table.

When referencing an entire array, use the array name assigned in the Calculation Specifications. When referencing an individual entry (element) in the array, use the array name assigned, followed by a comma, followed by the index number or the name of a numeric field containing the index number. Indexes begin with the value 1--zero cannot be used for an index. Variable fields used for indexes must be assigned zero decimal positions, and be no more than 9 digits in length.

EXAMPLE

To reference the fourth element in the array named XARY, write:

XARY,4

Suppose that N was the name of a numeric field containing the number 4; you could then reference the fourth element in XARY by writing:

XARY,N

The combination of the array name, comma, and index is limited in length by the width of the operand field in which it appears on the specification sheet. If you use a numeric field for the index, this must be a field with zero decimal positions and a value of one or greater.

When reading data into an array through Input Specifications, be certain to specify that complete elements (not portions of elements) be read.

You can reference an entire array or an individual element (the index is optional) when you use an array name with the following operations, discussed in Section VIII:

ADD	MHHZO	MOVE	SQRT
DEBUG	MHLZO	MOVEA	SUB
DIV	MLHZO	MOVEL	Z-ADD
LOKUP	MLLZO	MULT	Z-SUB

You must reference array elements individually (include an index) with the following operations, also discussed in Section VIII:

BITOF	DSPLY
BITON	TESTE
COMP	TESTZ

You must reference an entire array with the XFOOT operation.

E File Extension Specifications

When you use array names without an index in Calculation Specifications, the object program follows these rules:

1. Whenever the factors and the result field reference arrays with the same number of elements, the operation is performed sequentially. The first element from every array is used first, then the second element, and so forth, until all elements in the arrays have been processed. If the arrays do not contain the same number of elements, the operation ends when the last element of the array with the fewest elements has been processed.
2. Whenever one of the factors references a field or constant, and the other factor and the result field both reference arrays, the operation is performed until every element in the shorter array has been used. The same field or constant is used in all of the operations.
3. Since multiple operations are being performed, resulting indicators cannot be used except for the XFOOT and LOKUP operations.

When you use a table or array name in the result field in an arithmetic or move operation (Section VIII), you temporarily change the table or array entries during object program execution. Additionally, you can add to or change any type of table or array by using the Input or Calculation Specifications. These changes, however, remain in effect only until object program execution is complete; at that time, any changes are lost.

You can make temporary changes permanent only by writing or punching out the modified entries onto new records and using these records instead of the original ones in the table or array input file. Alternatively, you can modify the original records themselves.

You can add entries to short tables and arrays through calculation operations or read them from new input records during object program execution.

5-40. Writing Tables and Arrays to Output Files

You can copy a table or array to an output medium such as a card deck, disc, or print-out in either of two ways:

1. To copy an entire table or array (except an execution time array), simply enter the name of the table or array output file in the To Filename Field (Columns 19 through 26) of the File Extension Specifications. The object program will write out the entire table or array, including all temporary modifications. To copy an execution time array, use the Output Specifications described in Section IX.
2. To request output of only selected items from the table or array, describe these items in the Output Specifications just as you would normal fields of data. When you use a table name in Output Specifications, the last entry found (not the entire table) is written out.

When you request editing of an entire array, the editing specified applies to all elements in the array. If differing editing requirements exist for individual elements, reference these elements individually. When an edit code is specified in Output Specifications for an entire array, edited array elements are separated by two blanks each on the output file so that you can distinguish them from each other more easily. However, if output is directed to a line printer device, array elements are separated by two blanks even when the edit code does not exist.

LINE COUNTER SPECIFICATIONS

SECTION

VI

Programmers use Line Counter Specifications to indicate, for printable output files, where overflow is to occur and where output is to end on a printed page. Line Counter Specifications are basic to the RPG/3000 overflow feature and must be used if your program depends upon overflow sensing.

With these specifications, you can use either of two options for handling overflow and specifying page length. The option you select depends upon whether line-skipping requests in your program refer to logical channel numbers, or to actual line numbers. (Line-skipping requests appear in the Skip Field. (Columns 19 through 22) of the Output Specifications. Your entry in the Carriage Control Type Field (Column 53) of the Control Record Specification determines what they refer to--a blank indicates logical channel numbers and an L or 1 signifies actual line numbers, as discussed earlier in Paragraph 3-16.) The two options — the Channel Number Option and the Line Number Option — are discussed below.

6-1. CHANNEL NUMBER OPTION

If skipping requests refer to channel numbers, use the Line Counter Specifications to associate a line number with each channel number referenced and to specify the line where form overflow occurs (overflow line). This option permits your object program to count each line as it is output and to keep track of the line number being skipped to, so that the program can detect the overflow line when it is reached. (The Line Counter Specifications are so named because they request this line counting.) Select this option by entering a blank in Control Record Specification Column 53.

6-2. LINE NUMBER OPTION

If skipping requests refer directly to line numbers, use the Line Counter Specifications to specify only the overflow line and the total number of print lines available on each page or form (form length). Select this option by entering either L or 1 in Control Record Specification Column 53.

6-3. FORM OVERFLOW AND LENGTH

When your program spaces, skips, or attempts to print beyond the overflow line (but does not advance beyond the current page), the Overflow Indicator is set on. This indicator signals that the end of the page is near and normally directs the program to print the following items at the bottom of the page (unless the Fetch Overflow routine is used to control output of overflow data):

- Detail lines
- Total lines (including those conditioned by the Overflow Indicator)

For a full discussion of detail and total lines, setting the Overflow Indicator, and using the overflow feature and the Fetch Overflow routine, please see Section IX.

When you specify the overflow line and the form length in the Line Counter Specifications and do not plan to use Fetch Overflow, be sure to allow enough space between the overflow and form length lines for all expected detail and total output to appear. Since you know in advance the maximum amount of detail and total lines your program produces, you can judge the required space accordingly.

L Line Counter Specifications

6-4. FIELDS

The RPG/3000 Line Counter Specifications are described below. Each specification (coding line) applies to only one file.

Because you use the specification fields differently, depending on the option you select, the fields as they apply to the Channel Number Option are discussed together; then, the fields as they apply to the Line Number Option are discussed.

6-5. Channel Number Option Fields

For this option, the **Sequence Number** (Columns 1 through 5) and **Program Name**(Columns 75 through 80) **Fields** are optional and the **Form Type** (Column 6) and **File Name** (Columns 7 through 14) **Fields** are required. Columns 15 through 74 contain twelve five-position groups that each allow you to associate one line number with one channel number or with the overflow line.

6-6. SEQUENCE NUMBER. (COLUMNS 1 THROUGH 5). This field contains the source record sequence number, described in Paragraph 2-3.

6-7. FORM TYPE (COLUMN 6) (REQUIRED). This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
L	Line Counter Specification.

Because this entry is required and is always the same, it is printed on the specification sheet for your convenience. (Figure 6-1, Item 1).

6-8. FILENAME (COLUMNS 7 THROUGH 14) (REQUIRED). Enter the name of the output file to which this specification applies, as described in the File Description Specifications.

Columns 7 through 14 Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Name of printable file to which this Line Counter Specification applies.

LINE COUNTER SPECIFICATIONS

SECTION

VI

Programmers use Line Counter Specifications to indicate, for printable output files, where overflow is to occur and where output is to end on a printed page. Line Counter Specifications are basic to the RPG/3000 overflow feature and must be used if your program depends upon overflow sensing.

With these specifications, you can use either of two options for handling overflow and specifying page length. The option you select depends upon whether line-skipping requests in your program refer to logical channel numbers, or to actual line numbers. (Line-skipping requests appear in the Skip Field. (Columns 19 through 22) of the Output Specifications. Your entry in the Carriage Control Type Field (Column 53) of the Control Record Specification determines what they refer to—a blank indicates logical channel numbers and an L or 1 signifies actual line numbers, as discussed earlier in Paragraph 3-16.) The two options — the Channel Number Option and the Line Number Option — are discussed below.

6-1. CHANNEL NUMBER OPTION

If skipping requests refer to channel numbers, use the Line Counter Specifications to associate a line number with each channel number referenced and to specify the line where form overflow occurs (overflow line). This option permits your object program to count each line as it is output and to keep track of the line number being skipped to, so that the program can detect the overflow line when it is reached. (The Line Counter Specifications are so named because they request this line counting.) Select this option by entering a blank in Control Record Specification Column 53.

6-2. LINE NUMBER OPTION

If skipping requests refer directly to line numbers, use the Line Counter Specifications to specify only the overflow line and the total number of print lines available on each page or form (form length). Select this option by entering either L or 1 in Control Record Specification Column 53.

6-3. FORM OVERFLOW AND LENGTH

When your program spaces, skips, or attempts to print beyond the overflow line (but does not advance beyond the current page), the Overflow Indicator is set on. This indicator signals that the end of the page is near and normally directs the program to print the following items at the bottom of the page (unless the Fetch Overflow routine is used to control output of overflow data):

- Detail lines
- Total lines (including those conditioned by the Overflow Indicator)

For a full discussion of detail and total lines, setting the Overflow Indicator, and using the overflow feature and the Fetch Overflow routine, please see Section IX.

When you specify the overflow line and the form length in the Line Counter Specifications and do not plan to use Fetch Overflow, be sure to allow enough space between the overflow and form length lines for all expected detail and total output to appear. Since you know in advance the maximum amount of detail and total lines your program produces, you can judge the required space accordingly.

L Line Counter Specifications

6-4. FIELDS

The RPG/3000 Line Counter Specifications are described below. Each specification (coding line) applies to only one file.

Because you use the specification fields differently, depending on the option you select, the fields as they apply to the Channel Number Option are discussed together; then, the fields as they apply to the Line Number Option are discussed.

6-5. Channel Number Option Fields

For this option, the **Sequence Number** (Columns 1 through 5) and **Program Name**(Columns 75 through 80) **Fields** are optional and the **Form Type** (Column 6) and **File Name** (Columns 7 through 14) **Fields** are required. Columns 15 through 74 contain twelve five-position groups that each allow you to associate one line number with one channel number or with the overflow line.

6-6. SEQUENCE NUMBER. (COLUMNS 1 THROUGH 5). This field contains the source record sequence number, described in Paragraph 2-3.

6-7. FORM TYPE (COLUMN 6) (REQUIRED). This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
L	Line Counter Specification.

Because this entry is required and is always the same, it is printed on the specification sheet for your convenience. (Figure 6-1, Item 1).

6-8. FILENAME (COLUMNS 7 THROUGH 14) (REQUIRED). Enter the name of the output file to which this specification applies, as described in the File Description Specifications.

Columns 7 through 14 Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Name of printable file to which this Line Counter Specification applies.

This entry must begin in Column 7.

EXAMPLE

To designate that this specification applies to the file named PRINTX, enter that file name in the File Name Field (Figure 6-1, Item 2).

6-9. LINE NUMBER (COLUMNS 15 THROUGH 17). Enter the line number that corresponds to the first channel you wish to specify, or to the overflow line. (You will define the corresponding channel or the overflow line in the next field.)

Columns 15 through 17 Entry	Meaning
1 through 12	The line number to which the channel number in the next field, or the overflow line, refers.
Blank	If the next field describes Channel 1, Line 6 is assigned; if next field describes overflow line, Line 60 is assigned to Channel 12. If the next field describes any other channel, the line number assigned equals the channel number multiplied by 5. (For instance, Channel 3 is assigned Line 15.)


This entry must end in Column 17. Leading zeros are permitted but not required.

6-10. CHANNEL NUMBER/OL (COLUMNS 18 THROUGH 19). Enter the channel number to which the line number in the preceding field applies, or enter OL to signify that the line number in that field designates the overflow line.

Columns 18 through 19 Entry	Meaning
1 through 12	Line Number in preceding field is assigned to this channel.
OL	Line number in preceding field designates the overflow line. If no number appears in previous field, Line 60 is assigned to Channel 12.
Blank	Line 6 is assigned to Channel 1; Line 60 is assigned to Channel 12 to overflow line.

For any other channel not found on this specification, the line number assigned equals the channel number multiplied by 5.

L Line Counter Specifications

HEWLETT  PACKARD RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS Page 2 of 5

Programmer: D. DUNBAR Date: 1/2/75

Program Title: ALPHA

Punching Instructions
 Graphic:

 Punch:

Program Name:

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Seq.	Chaining File Code (C1 CB)	From Filename	To Filename	Table Array or Routine Name	Entries Per Record (1 999)	Entries Per Table Array (1 9999)	Entry Length (1 256)	Data Format (P B L R)	Decimal Positions (0 9)	Table Array Sequence (A D)	Table Array Name	Entry Length (1 256)	Data Format (P B L R)	Decimal Positions (0 9)	Table Array Sequence (A D)	Comments
0009	E	01	C	CHAINF	CHAIN													CHAINED FILE
	E	03																
	E	04																
0010	E			TABLEA	0060050010	L2							TABLEB	2R4				TABLE FILE

Line Counter Specifications

Sequence Number	Form Type (L)	Filename	Line Number	Channel Number/OL/FL	Line Number	Channel Number/OL/FL	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
0011	L	UPRINTX	0120	01	0150	03	0200	04	0630	OL								
0012	L	UPRINTY	0680	OL	0740	FL												

Figure 6-1. File Extension and Line Counter Specification Sheet

If Channel 1 is not named in this field (or in any of the remaining Channel Number Fields in Columns 20 through 74), Line 6 is assigned to Channel 1 by default. (Channel 1 is normally associated with the first line on a print-out page.) If OL or Channel 12 is not named in this field (or in any of the remaining Channel Number/OL Fields in Columns 20 through 74), Line 60 is assigned as the overflow line. The entry in this field must end in Column 19. When any channel 1 through 9 is referenced, a leading zero is permitted but not required.

6-11. LINE NUMBER AND CHANNEL NUMBER/OL/FL (COLUMNS 20 THROUGH 74). Use these fields in exactly the same way as you used the fields in Columns 15 through 19. The same rules apply. Together, Columns 15 through 74 allow you to designate line numbers for up to twelve channels. The first pair of Line Number and Channel Number Fields that is completely blank terminates your list of Line/Channel Number assignments. You can assign the channel numbers in any order on the specification sheet, and you can make the overflow line assignment in any pair of fields. Two or more Channel Numbers can be associated with the same Line Number. Any Channel Numbers left unassigned will be given default values, as described for a **Blank Entry** in Columns 18 through 19 (discussed in Paragraph 6-10). Also see Table 3-5a for a description of the initialization and default values of all carriage controls.

EXAMPLE

Relate Channel 1 to Line 12 (Item 3), Channel 3 to Line 15 (Item 4), and Channel 4 to Line 20 (Item 5), and designate Line 63 as the overflow line (Item 6) for the file PRINTX by the entries in Figure 6-1.

6-12. PROGRAM NAME (COLUMNS 75 THROUGH 80). This field contains the program name or any other information, as discussed in Paragraph 2-6.

6-13. Line Number Option Fields

For this option, the Sequence Number (Columns 1 through 5) and Program Name (Columns 75 through 80) Fields are optional and the Form Type (Column 6) and File Name (Columns 7 through 14) Fields are required. Columns 15 through 24 contain two five-position groups; one denotes the overflow line and the other indicates form length. Columns 25 through 74 are not used.

6-14. SEQUENCE NUMBER. (COLUMNS 1 THROUGH 5). This field denotes the source record sequence number, described in Paragraph 2-3.

6-15. FORM TYPE (COLUMN 6) (REQUIRED). This field denotes the type of specification, as explained in Paragraph 6-7.

6-16. FILENAME (COLUMNS 7 THROUGH 14) (REQUIRED). This field describes the name of the output file to which this specification applies, as defined in Paragraph 6-8.

L Line Counter Specifications

6-17. LINE NUMBER (COLUMNS 15 THROUGH 17). Enter either the line number of the overflow line or the form length. (You will specify whether this entry describes the overflow line or form length in the next field).

Columns 15 through 17 Entry	Meaning
1 through 112	The line number of the overflow line or form length, as determined by the next field.
Blank	None.

This entry must end in Column 17. Leading zeros are permitted but not required.

6-18. OL/FL (COLUMNS 18 THROUGH 19).

Enter OL to indicate that the number in the previous field designates the overflow line or FL to indicate that it designates form length.

Columns 18 through 19 Entry	Meaning
OL	Line number in preceding field designates the overflow line. If no number appears in previous field, Line 60 is assigned.
FL	Line number in previous field designates form length. If no number appears in previous field, Line 66 is assigned as last line.
Blank	Line 60 is assigned as Overflow Line: Line 66 is assigned for form length.

If OL and/or FL are not named in this field or in the next OL/FL Field (Columns 23 through 24), the defaults assigned are Line 60 for the overflow line and/or Line 66 for the form length.

6-19. LINE NUMBER AND CHANNEL NUMBER/OL/FL (COLUMNS 20 THROUGH 24)

Use these fields in exactly the same way as the previous Line Number and Channel Number/OL/FL Fields (Columns 15 through 19). The same rules apply. In the fields in Columns 15 through 24, you can assign overflow line and form length in any order.

EXAMPLE

Assign Line 68 as the overflow line (Item 7) and request a form length of 74 lines (Item 8) for the file PRINTY, as shown in Figure 6-1.

6-20. DEFAULT SUMMARY

If you leave the optional fields of the Line Counter Specifications blank, the default specifications shown in table 6-1 apply.

Table 6-1. Line Counter Specifications Default Values

Columns	Field	Default Specifications
1 through 5	Sequence Number	No sequence number applies.
15 through 24	Line Number and Channel Number/OL/FL	Channel Number Option: Line number 6 assigned to Channel 1; other line numbers assigned equal channel number X 5, and overflow at Line 60 (Channel 12). Line Number Option: Overflow at Line 60 and form length 66 lines.
25 through 74	Line Number and Channel Number/OL	Channel Number Option only: Line numbers assigned equal channel number X 5, and overflow at Line 60.
75 through 80	Program Name	None



INPUT SPECIFICATIONS

SECTION

VII

For every input, update, or combined file that your RPG/3000 program will read, you must enter an *Input Specification*. In this specification, you supply two general kinds of information:

1. The first kind identifies the types of records in the file and shows how they relate to one another. This information appears in the *Record Description Fields* (Columns 7 through 42).
2. The second kind describes the formats and locations of the fields in the input records, and provides directions for testing and using their contents. This information appears in the *Field Description Fields* (Columns 43 through 70).

7-1. FIELDS

The RPG/3000 Input Specifications contain the following fields:

7-2. Sequence Number (Columns 1 through 5)

This field contains the source record sequence number, described in Paragraph 2-3.

7-3. Form Type (Column 6) (Required)

This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
I	Input Specification

Because this entry is required and is always the same, it is pre-printed on the specification sheet for your convenience (Figure 7-1, Item 1).

7-4. Record Description Fields (Columns 7 through 42)

In these fields, you describe all record types in the file that are used by your program. You can begin this description on the first line on the Input Specifications Sheet. On lines containing entries in these fields, leave Columns 43 through 70 blank.

7-5. FILE NAME (COLUMNS 7 THROUGH 14) (REQUIRED). In this field, enter the name of the file whose records and fields you are describing.

**Columns 7 through 14
Entries**

Meaning

Valid file name. (File names contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.) Columns 7 through 12 may contain a Data Structure name (refer to 7-33A).

Name of input, update, or combined file containing records and fields described.

This file name must be that of an input, combined, or update file--it cannot refer to an output or display file. Begin this entry in Column 7. Be sure to spell the name exactly as it appears on your *File Description Specifications* Sheet, where it first occurs in your program. All entries on this and all following lines will apply to this file until a new file name appears in this field on the specifications sheet. Thus, if the record and field information for this file occupies more than one line, you do not need to repeat the file name on subsequent lines. You *can* repeat the file name on other lines, but whether or not you do, you must describe *all* entries for this file before describing those for another.

EXAMPLE

To show that the records you are describing belong to an input file named INP, enter this name in the File Name Field (Figure 7-1, Item 2).

7-6. GROUP SEQUENCE (COLUMN 15 THROUGH 16) (REQUIRED). Use this field to assign sequence numbers to different types of records in this file and check record sequence within groups, or to show that no group sequence applies.

**Columns 15 through 16
Entry**

Meaning

Two digits, 01 through 99.

Assign this sequence number to the type specified for this record in the *Record Identification Codes Field* (Columns 21 through 41), and verify the sequence of the record within its group when it is read.

Any two alphabetic characters.

No group sequence applies; do not sequence-check the input records.

A file may contain records of different types, with each type distinguished by an identification code in a particular field. Your program may require that the records be arranged in groups so that one type of record is processed before another within each group. For example, if each group of records includes one customer identification record and one corresponding invoice record, you may want to process each customer identification record before its corresponding invoice record. To establish this kind of sequence, you must indicate the sequence number for this record type by an entry in the *Group Sequence Field* (Columns 15 through 16). Your program will verify group sequence as it reads the records. If it finds a sequence error, it will print an appropriate message and terminate. You can only use sequence numbers to ensure that

I Input Specifications

within a group, all records of a certain type precede those of another. You cannot use them to check the order of records with the same identification codes or according to data field contents. Always assign the lowest sequence number, 01, to the first record type. Assign ascending numbers (such as 02, 03, 04, . . .) to the types remaining. The numbers following 01, however, need not be consecutive. Thus, you could specify 01, 02, 05, 52 and so forth.

If the file contains only one type of record, or records of different types that need not be grouped in order, enter any two alphabetic characters in the *Group Sequence Field*. In any case, you must NOT leave this field blank.

Within any file, you can specify alphabetic sequence entries for some record types and numeric sequence entries for others. In such cases, list all alphabetic entries first on the specification sheet. (In the file itself, however, records designated by alphabetic entries can appear in any order.) Do not enter sequence numbers or characters on specification lines containing AND or OR entries (as defined in Paragraph 7-18 and 7-19). For an OR Line, any sequence entry on the previous line applies.

Within the input, update, or combined file itself, records designated by *sequence numbers* in the *Group Sequence Field* must appear in the order specified. Records designated by *alphabetic sequence characters*, however, can appear in any order, even interspersed with those assigned *sequence numbers*.

Note: Do NOT confuse the *Group Sequence Field* with the *Sequence Field* (column 18) of the *File Description Specifications*, which specifies ascending or descending sequence for the overall file.

EXAMPLE

Assume that the file INP contains three different types of records that relate to a company's sales staff. Each type is indicated by a letter in Position 10 of the record: A (for a record showing a salesman's name), B (for one indicating his current territory), and C (for one containing his sales quota). Within each group, we want to specify that Type A appears first, Type B appears next, and Type C appears last. We enter the sequence numbers 01, 02, and 03 in the Group Sequence Field (Figure 7-1, Item 3), indicating the three types on the corresponding lines in the *Record Identification Codes Field* (Figure 7-1, Item 4). This last field is explained later in this section. To give you a general idea of how it is used, however, the first line in that field specifies that the first record in each group must contain an alphabetic character, A, in Position 10. The input records themselves are shown in Figure 7-2.

7-7. NUMBER OF RECORDS (COLUMN 17). If you have requested record group sequencing in the previous field, indicate how many records of each type can appear in each group by making an entry in the *Number of Records Field*.

Column 17 Entry	Meaning
1	Only one record of this type is allowed in a group.
N	One or more records of this type is allowed in a group.
Blank	No restrictions on types per group because sequence <i>characters</i> instead of numbers appear in the <i>Group Sequence Field</i> ; sequencing is not requested.

I Input Specifications

Use this field only when requesting sequence-checking of record groups. Do not make an entry in this field in specification lines containing AND or OR entries. For an OR line, any *Number of Records* entry in the previous line applies.

EXAMPLE

Designate that each record group in the file INP may contain only one record of Type A by entering 1 in the Number of Records Field (Figure 7-1, Item 5).

7-8. **OPTION (COLUMN 18).** If you have requested group sequence checking, use this field to indicate whether a record of this type **MUST** appear in each group or is optional.

Column 18 Entry	Meaning
O	Records of the type specified are optional and may or may not be present in each group.
U	- The Data Structure defined on this specification is the Local Data Area
Blank	At least one record of the type specified MUST be present in each group, or An alphabetic sequence entry is specified in the <i>Group Sequence Field</i> .

The entry O prevents a sequence error from occurring if a record of the designated type does not appear in a group.

Leave this field blank for specification lines containing AND or OR entries. For an OR line, any entry in this field on the previous line applies.

EXAMPLE

For the file INP, indicate that each record group must contain **AT LEAST** one record of Type B by leaving the Option Field blank on the corresponding line, and that records of Types A and C are optional by entering O in the Option Field on the lines describing those types (Figure 7-1, Item 6).

7-8A. LOCAL DATA AREA/USER DATA STRUCTURE

The **Local Data Area** is a 256-byte one record file named LDAFILE which provides a means for your program to receive data from, as well as pass data to, other programs.

To specify that your program will access the **Local Data Area**, enter a *U* in column 18 and a *DS* in columns 19 and 20. This indicates that the following data structure specification is your User Data Structure. See figure 7-2a.

The **Local Data Area** is created and initialized to BLANKS by the RPGINIT Utility. Typically this is run as a Logon UDC for all users. See the *RPG Utilities Reference Manual* for a discussion of the RPGINIT Utility.

7-9. RECORD INDICATOR/LOOK-AHEAD/TRAILER/DATA STRUCTURE (COLUMNS 19 THROUGH 20). Use this field to assign a record-identifying indicator to the record type, to indicate that the record contains look-ahead fields, to specify that the next lines define the trailer portion of a spread record, or to specify a Data Structure.

Columns 19 through 20 Entry	Meaning
A pair of digits, 01 through 99	Assign this general indicator.
F0 through F9	Assign this function key indicator.
KA through KN, KP through KY	Assign this command key indicator.
L1 through L9	Assign this control level indicator.
LR	Assign the last-record indicator.
MR	Assign the matching record indicator.

Columns 19 through 20 Entry	Meaning
OA through OG, OV	Assign this overflow indicator.
1P	Assign the first-page indicator.
H0 through H9	Assign this halt indicator.
U1 through U8	Assign this user indicator.
**	This record contains a look-ahead field.
TR	The next lines on the specification sheet define the trailer portion of a spread record.
DS	The next lines on the specification sheet define a Data Structure.
Blank	Assign no indicator, look-ahead field, or trailer portion.

7-10. Indicators. When your program processes a file containing several types of records, you may want it to perform different operations for each type. By using the *Record Indicator Field* to assign an indicator to each record type, you can control processing in just this way. The indicator is merely a software switch that can be set either ON or OFF to *condition* (request or suppress) various operations. At the beginning of each program cycle, all indicators assigned to all record types are turned off. Each time your program reads a record, it turns the indicator assigned to that record's type ON. Then, all operations conditioned by this indicator to take place are performed. (You condition such operations by entering the indicator name in the *Operation Indicator Field* (Columns 9 through 17) of the *Calculation Specifications*.)

Note: In the case of chained files, indicators for more than one record type can be ON simultaneously, since more than one record type in such files can be processed at one time. When a program is doing multiple reads from one or several demand files during the same RPG cycle, the record identifying indicators assigned to the file(s) remain on throughout the cycle if the previous READ operations were executed successfully. If you wish to ensure that these indicators are off, you must explicitly set them off in the *Calculation Specifications*.

Indicators can also be turned ON and OFF by operations in the *Calculation Specifications*, but reading a new record can then negate such settings. Because the indicator normally remains ON for the remainder of the RPG/3000 program cycle, you can use it to condition both calculation and output operations.

Use record indicators **only** when you are:

1. Processing different record types in files containing more than one type, or
2. Using update files, and want to ensure that output is not written before related input is read. (See discussion of Output Specifications in Section IX.)

You need not use indicators when you wish to process every record in a file.

I Input Specifications

Note: All indicators except the general indicators should be used with great care. HP recommends that you understand how and when the RPG/3000 program cycle logic sets and treats these indicators before you use them in a program. In particular, you should understand how heading, detail, and total records (introduced in Section IX) are processed. You can find this information in Appendixes D and E.

You can assign the indicators in any order you wish on the Input Specifications Sheet. To use the same indicator for several different record types, simply assign the same indicator to all of these types; do this with the OR operation, described in Paragraph 7-19.

Do not assign record indicators in AND lines of AND relationships. (AND relationships are described in Paragraph 7-18.)

General Indicators. These identify a specific record type in a file that contains two or more types, and are the indicators most frequently used by RPG programmers. You can assign up to 99 such indicators (designated by numbers ranging from 01 to 99), although you would rarely use all of these. Whenever your program reads a record associated with a specific indicator, it turns the indicator ON and then performs all operations conditioned by the indicator to take place.

EXAMPLE

Assign the general indicator 03 to the first record type defined in the Input Specifications (Figure 7-3) for the file READX by entering this indicator number in the Record Indicator Field (Item 1). Every time your object program reads a record of this type, it will perform the ADD operation requested in the Calculation Specifications (Item 2).

Function Key Indicators. You can use function key indicators in the same way, and for the same purpose, as general indicators.

If your program contains an RPG/VPLUS interface "WORKSTN" file, the function key indicators serve a special purpose. One of the function key indicators (F0-F9) is turned on to indicate which "Event" code is being returned to the program's Input specifications: F0 means the Enter key was pressed; F1 through F8 mean the corresponding function key was pressed; and F9 means that "Event" code 9 or greater is being returned. For a complete discussion of the RPG/VPLUS interface, refer to section XIII.

Function key indicators F1-F8 also serve a special purpose when used in conjunction with the SET, DSPLY and DSPLM operators. For a complete discussion of the use of function key indicators with SET, DSPLY and DSPLM, refer to the descriptions of those operators in Section VIII.

Command Key Indicators. You can use command key indicators in the same way, and for the same purpose, as general indicators.

If your program contains an RPG screen interface "WORKSTNR" file, the command key indicators serve a special purpose. When the "F1" function key (labelled COMMAND on the HP262X terminals) is pressed, followed by a key from the top row of the keyboard (See Figure 7-3), the corresponding command key indicator is turned on. Only keys which have been enabled in the "S" record of the screen format specifications are treated as special indicators and are turned off and on during the RPG/3000 program cycle by the

WORKSTNR interface. All command key indicators which have been disabled for the RPG screen interface can be used in the same manner as general indicators. For a complete discussion of the RPG screen interface, refer to the **RPG Utilities Reference Manual**.

Control Level Indicators. In the *Record Indicator Field*, you can use control level indicators in the same way, and for the same purpose, as general indicators. Unlike general indicators, however, control level indicators that are on at any point are turned off before the next record is read.

You need not assign control level indicators in any particular sequence, nor in contiguous order. Thus, you could assign L5, L1, and L7, to record types in that order, and ignore all other indicators.

You can associate control level indicators with individual fields as well as entire records, as discussed in Paragraph 7-34. When you do this, you can use these indicators to rank various fields according to their level of importance.

Last Record Indicator. Use this indicator (LR) to identify the last record to be read by your program. When this record is encountered, any calculations conditioned by this indicator are done, output is completed, and the program terminates.

Matching Record Indicator. This is a special indicator normally set on when matching fields occur. It only remains on during total-time calculations, and turns on or off at detail time depending on whether a matching record condition occurs.

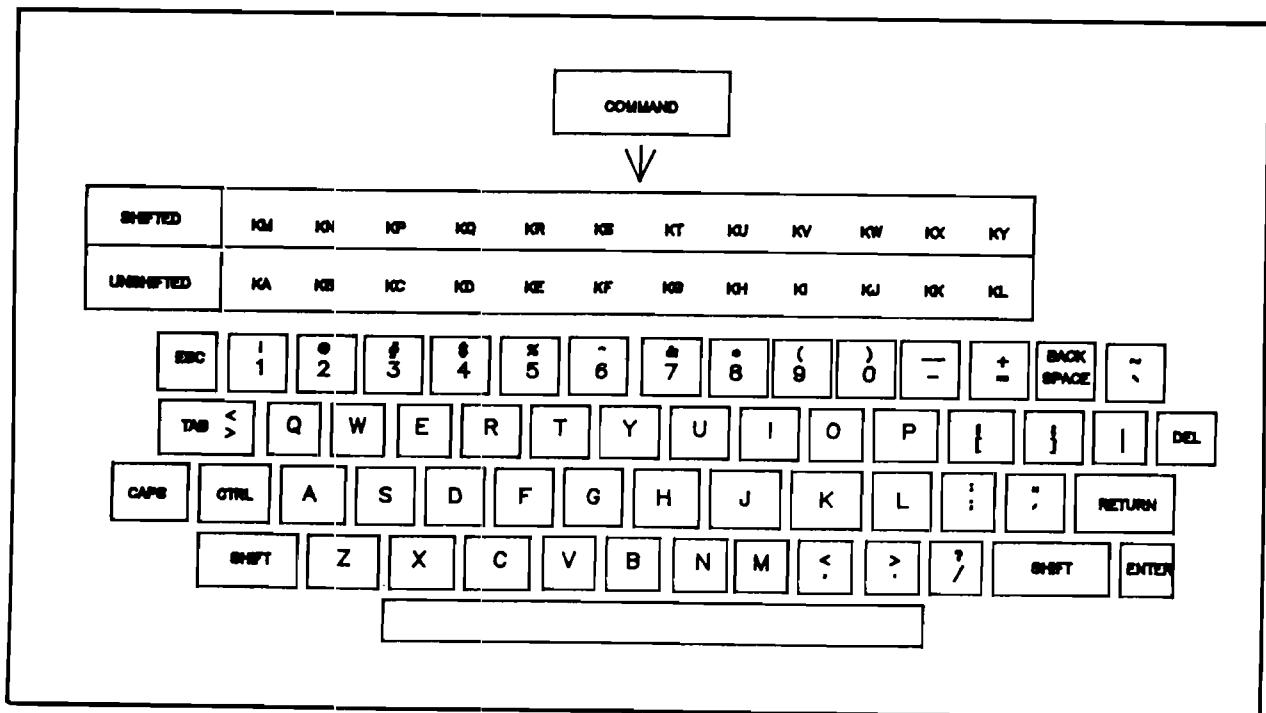


Figure 7-3. Command Keys on Keyboard

I Input Specifications

Overflow Indicators. One overflow indicator can be assigned to each printed output file named in the File Description Specifications. Assign this indicator to a record type if you want your program to turn that indicator ON when that record type is encountered. When the indicator turns ON, the overflow data for your program will be printed. You can assign any of eight overflow indicators (OA through OG, and OV). More information on overflow appears in Section IX.

First Page Indicator. You normally use this indicator in printing first-page headings. However, after these headings are printed, you can use it as you would a general indicator. Assign this indicator to a record type by entering 1P in the *Record Indicator Field*.

RPG INPUT SPECIFICATIONS

HEWLETT PACKARD

Programmer BOB BLASE Date 1/10/75

Program Title PROJECT REPT

Sequence Number	Form Type (I)	File Name	Group Sequence	Number of Records (I/N)	Record Indicator/Option (I)	Look-Ahead/Trailer	Position (1-9999)	Not (N)	Portion (C/Z/D)	Character	Position (1-9999)	Not (N)	Portion (C/Z/D)
1	0	READX	1	1	0	5	21	0	5	C	22	0	5
2	0		2	1	0	5	22	0	5	C	23	0	5
3	0		3	1	0	5	23	0	5	C	24	0	5
4	0												
5	0												
6	0												
7	0												
8	0												
9	0												
10	0												
11	0												
12	0												
13	0												
14	0												
15	0												
16	0												
17	0												
18	0												
19	0												
20	0												
21	0												
22	0												
23	0												
24	0												
25	0												
26	0												
27	0												
28	0												
29	0												
30	0												
31	0												
32	0												
33	0												
34	0												
35	0												
36	0												
37	0												
38	0												
39	0												
40	0												
41	0												
42	0												
43	0												
44	0												
45	0												
46	0												
47	0												
48	0												
49	0												
50	0												

RPG CALCULATION SPECIFICATIONS

HEWLETT PACKARD

Programmer BOB BLASE Date 1/10/75

Program Title PROJECT REPT

Sequence Number	Form Type (C)	Control Level (L/D/LP/LR/SR/AN/DOR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Initial Positions (0-9)	Final Positions (H)	Result Indicator
1	C		Not	STOREA	ADD	STOREB	RES F				
2	C		And								
3	C		Not								
4	C		Not								
5	C		Not								
6	C		Not								
7	C		Not								
8	C		Not								
9	C		Not								
10	C		Not								
11	C		Not								
12	C		Not								
13	C		Not								
14	C		Not								
15	C		Not								
16	C		Not								
17	C		Not								
18	C		Not								
19	C		Not								
20	C		Not								
21	C		Not								
22	C		Not								
23	C		Not								
24	C		Not								
25	C		Not								
26	C		Not								
27	C		Not								
28	C		Not								
29	C		Not								
30	C		Not								
31	C		Not								
32	C		Not								
33	C		Not								
34	C		Not								
35	C		Not								
36	C		Not								
37	C		Not								
38	C		Not								
39	C		Not								
40	C		Not								
41	C		Not								
42	C		Not								
43	C		Not								
44	C		Not								
45	C		Not								
46	C		Not								
47	C		Not								
48	C		Not								
49	C		Not								
50	C		Not								

Figure 7-3. Using a General Record Indicator

Halt Indicators. These indicators identify types of records that, if encountered, transmit a message to the operator and halt your program at the end of its current detail cycle. (The program turns an indicator ON when a record type to which it is assigned is encountered; the program tests the indicator at the end of the detail cycle.) The computer operator can continue the program at his option; in this case, the program resumes, turns the halt indicator off, and runs until another halt indicator is turned on and tested, or until the program is completed. You can assign ten of these indicators (H0 through H9).

User Indicators. User indicators can also be used to condition various operations, in much the same way as general indicators. You can assign any of eight indicators (U1 through U8).

7-11. Look-Ahead Fields. Sometimes, you may want your RPG/3000 program to examine information in certain fields before it actually processes that information. Such fields are called *look-ahead fields*. For an input file, a reference to a look-ahead field indicates a field on the *next* record available for processing. For an update or combined file, such a reference indicates a field on the *record* currently being processed.

You may describe one set of look-ahead fields for each input, update, or combined file; this description applies to all records in that file, regardless of type. You cannot, however, define look-ahead fields on chained files, demand files, on AND or OR lines, or on files containing spread records (described in Paragraph 7-12).

To designate a look-ahead field, enter two asterisks (**) in the *Record Indicator/Look Ahead/Trailer Field* of the Input Specifications, and then--on the NEXT line--enter its location and name in the *Field Location* and *Field Name Fields* described later in this section. You can designate as many look-ahead fields as you like by entering names on the following lines. (See Figure 7-4.) The line containing the asterisks must follow a line and/or record type with an alphabetic sequence entry in the *Group Sequence Field* (Columns 15 through 16). On this line, you must also leave Columns 17 through 18 and Columns 21 through 74 blank.

Each look-ahead field name must be unique--it must not be defined anywhere else in your program.

Your object program cannot alter the contents of look-ahead fields; it cannot use them as result fields or blank-out their current contents. To use the information in a field both before and after the record is selected for processing, you must describe this field once as a look-ahead field and then, again, as a normal field.


When your program reaches the end of a file whose records contain look-ahead fields, it automatically fills those fields with ASCII 9's (for alphanumeric fields) and packed +9's (for numeric fields).

EXAMPLE

To define a look-ahead field named LOOKHD, ten-characters long, make the entry shown as (Item 1) in Figure 7-4.

7-12. Spread Records. In some data files, records are arranged in groups consisting of a header record followed by a separate record for each item or transaction applying to that header record. For example, an inventory program for a hardware and carpentry supply house might use a file that contains a header record for each general class of item stocked (such as saws, hammers, and screwdrivers). In this file, each header record would be followed by records that show the quantity of each specific type of item on hand--one record for each type; for instance, one record might show 63 cross-cut saws, another might indicate 15 rip saws, and so forth. You can condense much information of this type by using *spread records*. These allow you to store more data on each record, and eliminate the need for individual records for each header and item/transaction.

I Input Specifications

HEWLETT  PACKARD RPG INPUT SPECIFICATIONS Page 3 of 5

Programmer D.M. YESCO Date 1/7/75

Program Title QUOTA

Punching Instructions
 Graphic

 Punch

Program Name

Sequence Number	Form Type (1)	File Name	Group Sequence	Number of Records (1/N)	Option (O)	Record Indicator/ Look Ahead/ Trailer	Record Identification Codes			Field Position		Field Name	Control Level (L1 L9)	Matching or Chaining Fields (M1 M9, C1 C9)	Field Record Relation	Field Indicators		
							1	2	3	From (1 9999)	To (1 9999)					Plus	Minus	Zero or Blank
1	I	OVERL D	AA															
2	I																	
3	I																	
4	I																	
5	I																	
6	I		AB			**												
7	I																	
8	I																	
9	I																	
10	I																	
11	I																	
12	I																	
13	I																	
14	I																	
15	I																	
16	I																	
17	I																	
18	I																	
19	I																	
20	I																	
21	I																	
22	I																	
23	I																	
24	I																	
25	I																	
26	I																	
27	I																	
28	I																	
29	I																	
30	I																	
31	I																	
32	I																	
33	I																	
34	I																	
35	I																	
36	I																	
37	I																	
38	I																	
39	I																	
40	I																	
41	I																	
42	I																	
43	I																	
44	I																	
45	I																	
46	I																	
47	I																	
48	I																	
49	I																	
50	I																	
51	I																	
52	I																	
53	I																	
54	I																	
55	I																	
56	I																	
57	I																	
58	I																	
59	I																	
60	I																	
61	I																	
62	I																	
63	I																	
64	I																	
65	I																	
66	I																	
67	I																	
68	I																	
69	I																	
70	I																	
71	I																	
72	I																	
73	I																	
74	I																	

Figure 7-4. Specifying Look-Ahead Fields

A spread record contains a header portion followed by a trailer portion for each item or transaction. Each trailer portion can include as many fields as necessary to describe the item or transaction, but the same fields must appear in *all* trailer portions.

Each spread record is restricted to a single logical record. If the number of trailer portions under a particular header require more space than that available on one logical record, you can use additional logical records for this information if you:

1. Begin each record with the same header portion.
2. Enter the header and trailer portions and their fields in the same locations on these records.

You can specify spread records for primary or secondary input files only; do not specify them for update or combined files, or for files with look-ahead fields.

When describing a spread record on the Input Specifications Sheet, follow these steps:

1. *Specify the header portion* by describing all fields within it that you plan to use in your program. The header portion includes all positions up to the first trailer portion. Describe each field in the header portion on a separate line, just as you would any other field, but following the normal file and record type entries. (Field descriptions are discussed beginning in Paragraph 7-21.) If you specify any record identification codes, control levels, or matching fields for the spread record, they must appear only in the header portion. If you enter a number rather than an alphabetic character in the *Group Sequence Field* (Columns 15 through 16) of the line containing the file and record-type entries, you must also enter *N* in the *Number of Records Field* (Column 17) of that line.
2. *Indicate the beginning of the trailer portion description* by entering TR in the Record Indicator Field (Columns 19 through 20) on one of the lines following the header description. Leave Columns 7 through 18 and 21 through 70 of this TR line blank. Enter only one TR line per spread record (but you can specify as many spread records as you like for each file).
3. *Specify the first trailer portion* by describing (after the TR line) all fields within it that your program will use. Because all trailer portions must be the same length and must contain the same fields, you need describe only the first portion. For this portion, describe each field on a separate line, in the *Field Position Field* (Columns 44 through 51) and the *Field Name Field* (Columns 53 through 58). You **MUST** describe the field or fields that contain the beginning and end position of the trailer portion. Leave columns 7 through 42, 59 through 62, and 71 through 74 of these specification lines blank. Do not allow the header portion to overlap into the trailer portion. Do not specify control levels or matching fields for the trailer portion.

You can specify as many trailer portions on each logical record as the record length allows. If a trailer portion (including the first one) is blank, or the logical record does not contain enough room for another trailer field, the program will read the next logical record in the file.

As an example, you could process a 100-character logical spread record by defining a header portion with fields in Positions 3 through 5 and 10 through 15, and a trailer portion with fields in Positions 20 through 23 and 25 through 34. Then, additional trailer portions (with non-blank data) could exist in Positions 35 through 49, 50 through 64, 65 through 79, and 80 through 94. You could also define other records for the file that may or may not include spread records.

EXAMPLE

To describe the spread records in the data file for the hardware supply house inventory program noted earlier, we might write the Input Specification shown in Figure 7-5. The data file is named CARDZ (Item 1) and multiple records are permitted (Item 2). The header portion consists of one field, TOOL (Item 3). The first trailer portion (and all subsequent ones) includes three fields--TYPE, PART, and QUANT (Item 4). In that portion of the data file shown, the data for saws appears on two records (Item 5) while that for screw-drivers appears on one (Item 6).

7-12A. Data Structures. The next Input specification lines define Data Structures. Refer to paragraph 7-33A for details.

7-13. Record Identification Codes (Columns 21 through 41). When an input, update, or combined file contains more than one type of record, and you want your program to process each type in a different way, you must use a special *record identification code* to identify each type. This code consists of a unique combination of characters that appear in certain positions in the input records. The simplest code, of course, is a single character in a particular position. For instance, you could use the letter X in Position 10 to designate a certain record type. Once you choose the code for each type, you must then describe this code and the field in which it appears by making an entry in the *Record Identification Codes Field* of the Input Specifications. This specification enables your program to tell, each time it reads a record, what type of record it is.

If the file contains only one type of record, or if all types are to be processed in the same way, you should leave this field blank.

For any record type, you can assign an identification code of up to three characters on a single specification line; you describe the first character in Columns 21 through 27, the second in Columns 28 through 34, and the third in Columns 35 through 41. By using AND entries, described in Paragraph 7-18, you can use additional lines to assign a code containing even more characters. By using OR lines, discussed in Paragraph 7-19, you can specify more than one code for any record type.


To describe each character, you use four subfields: the *Position*, *Not*, *Portion*, and *Character* Subfields. The entries in these subfields allow your program to test for the presence or absence of a full character, or the zone or digit portion of a character, in the indicated position as it reads each record.

Your program tests each record read against the identification codes you specify in the Input Specifications, continuing until it identifies the record type. Then, the program turns on the record identifying indicators associated with the type so that all corresponding operations can be executed.

When a record satisfies the requirements of more than one type, it is processed only under the first type for which it qualifies.

You can ensure that a record will be unconditionally identified for processing by leaving the *Record Identification Codes Field* for that record blank. But, when you use identification codes for *any* records in a file, you can only leave this field blank for one type, which you must describe *last*.

7-14. Position (Columns 21 through 24, 28 through 31, and 35 through 38). In this subfield, indicate the position in the input record where this character of the identification code must appear to identify the corresponding record type.

HEWLETT  PACKARD

RPG INPUT SPECIFICATIONS

Programmer D.G. SMITH Date 7/3/75

Program Title INVENTORY

Graphic					
Punch					

Program No. _____

Sequence N. in order	Form Type (I)	File Name	Group Sequence	Option (O)	Record Identification Codes			Field Position		Field Name	Control Level (L1-L9)	Matching or Chaining Fields (M1-M9; C1-C9)	Field Record Relation	Field Plus
					1	2	3	From (1-9999)	To (1-9999)					
1	I	CARDZ	01	AA				1	10	TOOL				
2	I			TR				11	20	TYPE				
3	I							21	25	PART				
4	I							26	30	QUANT				

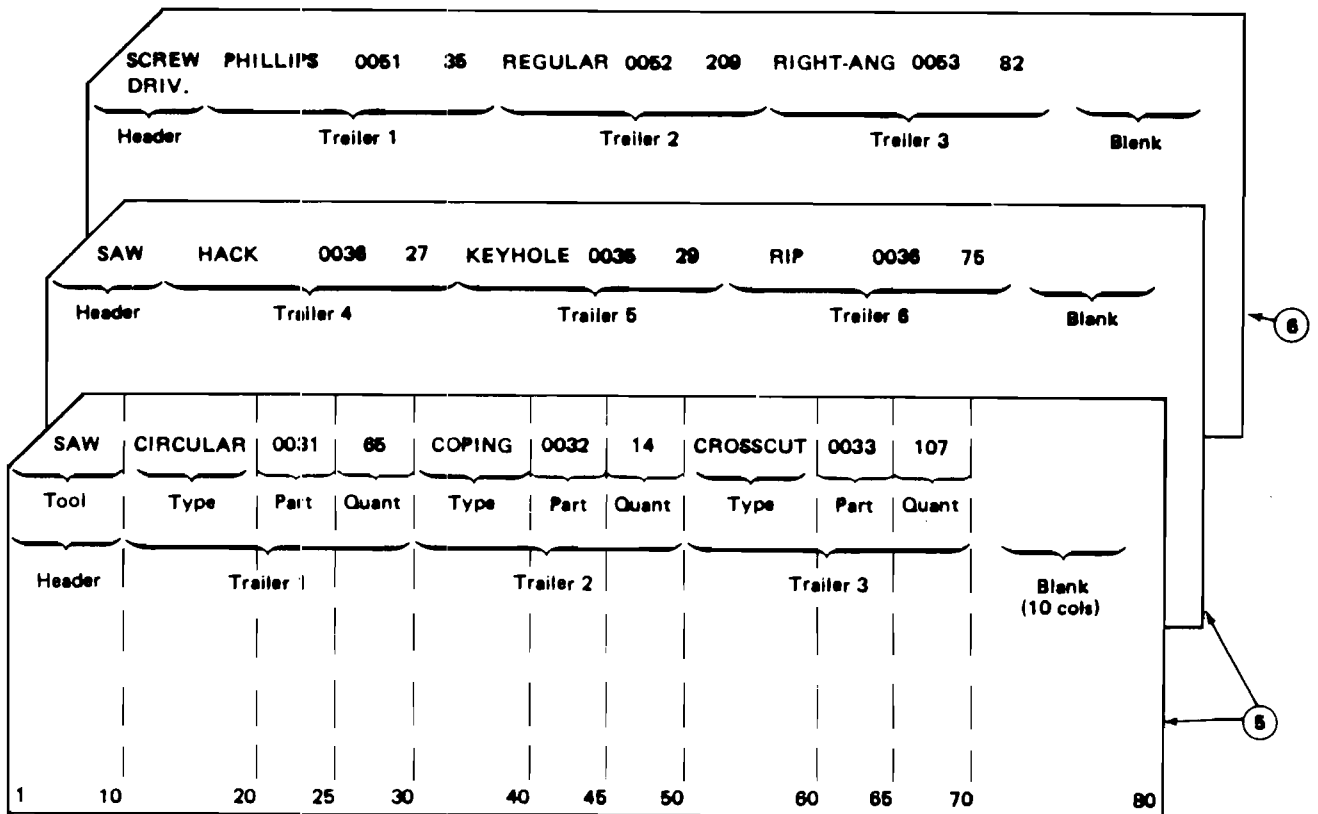


Figure 7-5. Specifying Spread Records

I Input Specifications

Columns 21 through 24, 28 through 31, 35 through 38	
Entry	Meaning
1 through 9999	Position where this identification code character appears in the input record.
Blank	A record identification code does not apply.

This entry must end in Column 24, 31, or 38. Leading zeros are permitted but not required. Be sure that you specify a position that falls within the record length defined. Describe this position in relation to the first position of the record --enter 1 for the first position, 2 for the second, and so forth.

7-15. **Not (Columns 25, 32, and 39).** This field allows your program to test a record identification code for the *absence* of a character, zone, or digit, rather than its presence.

Columns 25, 32, 39	
Entry	Meaning
N	This character, zone, or digit, must NOT be present in the position indicated by the Position subfield.
Blank	If you specify a character, zone, or digit, it must be present.

7-16. **Portion (Columns 26, 33, and 40).** This subfield indicates what portion of the character to use in the record identification code.

Columns 26, 33, 40	
Entry	Meaning
C	Use entire character (zone and digit portions).
Z	Use zone portion of character only.
D	Use digit portion of character only.
Blank	No character applies.

Your RPG/3000 program may process data input on punched cards or card images. On such cards, each character is represented by a unique combination of punches based on Hollerith code. Each combination can be divided into two portions--a zone portion and a digit portion. The zone portion encompasses the 11 and 12 Rows on the card; the digit portion includes the 0 through 9 Rows. As an example of this representation, the uppercase letter A is represented by a punch in Row 12 (zone) and another punch in Row 1 (digit). The Hollerith representations for all characters appear in Appendix G.

When you specify a character as part of a record identification code, you can request your program to use the entire character or only the zone or digit portion of that character in testing to identify a record type. If, for instance, you indicate only the zone portion, your program will compare only that portion of the character in the indicated field on each input record against the record identification code you specify. When only zone or digit portions are used in this manner, many characters--all having the same zone or digit representations--can satisfy the test.

When your program reads a character in Hollerith representation (six possible punches) into memory, that character is stored internally in the equivalent ASCII representation (eight bits). Although the character still retains zone and digit portions in the ASCII representation, the structure of either portion may not be the same as in the Hollerith representation. Since testing is based on the internal ASCII representation, you should understand how the various Hollerith equivalents are evaluated, as discussed below.

Zone Portion. During testing, your program compares the appropriate character from the input record with the character you enter in the Character Subfield (Columns 27, 34, or 41), described further in Paragraph 7-17. When you request the program to use the zone portion only of a character, testing proceeds as follows:

1. If the Character Subfield contains any character represented by a punch in Row 12 appearing alone or in combination with a punch in the digit area, ANY of the following characters on the input record satisfy the test:

The letters A through I
 { ({} is equivalent to +0)
 &

2. If the Character Subfield contains any character represented by a punch in Row 11 appearing alone or in combination with a punch in the digit area, ANY of these characters encountered in the input record satisfy the test:

The letters J through R
 } (} is equivalent to -0)
 -

3. If the Character Subfield contains any character represented by no zone punch (Row 11 or 12) with or without an accompanying digit punch, these characters on the input record satisfy the test.

Any digit, 0 through 9
 Blank

4. If the Character Subfield contains a character represented by any other combination of punches, then the upper four bits (0 through 3) of the ASCII machine representation of this character are compared to the upper four bits of the character in the input record. If these bit configurations are equal, the test is satisfied.

Note: Before using any characters in this category, you should be familiar with their ASCII representation in the HP 3000. For this information, please see Appendix G.

I Input Specifications

Digit Portion. When you request the program to use the digit portion only of a character, testing takes place in this way:

1. If the Character Subfield contains a character represented by a digit punch with or without an accompanying zone punch (in Rows 11 or 12), any character represented by the same digit punch, with or without a zone punch, satisfies the test. Thus, if you enter *A* in the Character Subfield, *A*, *J*, or *I* on the input record satisfy the test because all three characters have the same digit-portion representation.
2. If the Character Subfield contains any character other than the digits 0 through 9, +0 through +9, or -0 through -9, then the lower four bits (4 through 7) of the ASCII machine representation of that character are compared to the lower four bits of the character in the input record.

7-17. Character (Columns 27, 34, and 41). Enter a character to be used in the record identification code.

Columns 27, 34, 41 Entry	Meaning
Any letter, digit, or special character.	Test this character in the input record, in the position indicated by the Position Subfield.
Blank	No character applies in this position.

This is the character whose entire internal representation, zone portion, or digit portion is to be tested.

EXAMPLES

Three examples of record identification codes appear in the Input Specification Sheet in Figure 7-6. The first record type defined on this sheet consists of all records with the letter *A* in Position 1 (Item 1). The second record type includes all records with any character but *D* in Position 2 and the letter *F* in Position 3 (Item 2). The third record type consists of all records with a character whose zone portion matches that of the letter *T* in Position 4 (Item 3).

7-18. AND Lines. When you want to specify a record identification code that contains more than three characters, you can do so by using one or more AND Lines. Do this as follows:

1. Describe the first three characters in the code on the first line of the record type specification, in the normal way.
2. Write AND in Columns 14 through 16 of the NEXT line, and enter up to three additional character descriptions in Columns 21 through 41 of this line; leave Columns 17 through 20 blank. This is your first AND Line.
3. Continue entering as many AND Lines as you need to specify all characters in your code.

A record must contain all characters specified in the appropriate positions before its associated indicator turns on, allowing the record to be processed.

HEWLETT-PACKARD RPG INPUT SPECIFICATIONS

Programmer MAYFIELD Date 7/6/75

Program Title ANY

Graphic
 Punch

Sequence Number	Form Type (1)	File Name	Group Sequence	Number of Records (1/N)	Option (O)	Record Indicator/ Look Ahead/Trailer	Record Identification Codes									Stacker Select (1/2)	Data Format (P/B/L/R/T/9)
							1	2	3	4	5	6	7	8	9		
1	I	ANYFILE	AA	01		1 CA											
2	I		BB	02		2 WCD 3 CF											
3	I		CC	03		4 ZT											
	I																
	I																
	I																
	I																
	I																
	I																
	I																
	I																
	I																
	I																
	I																

Figure 7-6. Record Identification Codes

I Input Specifications

7-19. OR Lines. When you want to identify a record type by more than one code, enter one or more OR Lines as follows:

1. Begin the description of the first code (containing up to three characters) on the first line of the record type specification. Continue this description with as many AND lines as you need.
2. Write OR in Columns 14 through 15 of the next line, and describe up to three additional characters for the second code in Columns 21 through 41 of this line; leave Columns 16 through 18 blank. This is your first OR line.
3. Continue entering as many OR lines as you need to specify all codes that identify this type of record.

When you specify an OR line, this indicates that ANY of the codes described identifies the record. You can enter as many AND or OR Lines as you desire, in any combination.

You can also specify a record identifying indicator in Columns 19 through 20 of each OR Line. If you do not do this, the last indicator specified applies.

EXAMPLES

Examples of AND and OR lines appear in Figure 7-7. The program processes a record as the first type specified if the letter A appears in Columns 1, 2 and 3, and B appears in Column 4, of that record (Item 1). We use an AND Line to define this code because we can only specify the first three characters on the first line; the AND line indicates that the character B is part of the code on the previous line.


The program identifies a record as the second type specified if the letter A appears in Columns 1, 2, and 3 or if Z appears in Column 80 (Item 2). The OR line indicates that either AAA or Z in the appropriate position identifies the record type.

The program identifies a record as the third type specified if A appears in all five Columns 1 through 5 or if Z appears in all five Columns 76 through 80 (Item 3). This example illustrates AND and OR lines used in combination.

7-20. STACKER SELECT (COLUMN 42). If your program reads input from an 80-column combined card-reader/punch unit, you can request selection of the cards into individual stackers as they are read. This is done by making the proper entry in the *Stacker Select Field* of the record identification line.

Column 42 Entry	Meaning
1 or Blank	Select Stacker No. 1
2	Select Stacker No. 2

This entry is valid for input and combined files only. You must specify the stacker-select entry for each record identification line for records in an OR relationship unless you wish the system default (Stacker No. 1) to take effect where no stacker-select entry appears. To select different stackers for records specified in an OR relationship, enter a stacker-select code on the same line as the appropriate record identification

HEWLETT  PACKARD RPG INPUT SPECIFICATIONS

Programmer JONES Date 1/2/75

Program Title PROGRAM XX

Pu
 Graphic
 Punch

Sequence Number	Form Type (I)	File Name	Group Sequence	Option (O)	Record Indicator/ Look Ahead/Trailer	Record Identification Codes									Field Post				
						1	2	3	4	5	6	7	8	9					
1	I	MYFILE	AA	01		1	CA	2	CA	3	CA								
2	I	AND				4	CB												
3	I	BB	BB	02		1	CA	2	CA	3	CA								
4	I	OR				80	CE												
5	I	CC	CC	03		1	CA	2	CA	3	CA								
6	I	AND				4	CA	5	CA										
7	I	OR				76	CE	77	CE	78	CE								
8	I	AND				79	CE	80	CE										

Figure 7-7. AND and OR Lines

I Input Specifications

code for each record type. If the *Stacker Select Field* is blank for an OR line, Stacker No. 1 is chosen. For an AND relationship, the stacker-select entry need appear only on one line. For combined files, any stacker select entry on the Output Specifications Sheet overrides any entry on the Input Specifications Sheet.

When sorting records from more than one file into the same stacker, remember that a record from an input file is not stacked until after it is processed and the next record is read. A record from an output file is stacked right after it is punched. A record from a combined file is stacked immediately after the output operation for that record takes place, (or when the next record is read if no output operation occurred for that record).

EXAMPLES

Examples of stacker selection appear in Figure 7-8. In the first record definition for the input file named CARDFL (Item 1), which does not specify an AND/OR relationship, every record with an A in Position 1 is sent to Stacker No. 2 (Item 2). In the second record definition (Item 3), which specifies an AND relationship, every record which contains a B in the first four positions is directed to Stacker No. 1 (Item 4). In the third record definition (Item 5), which specifies an OR relationship, records with C in Position 1 are sent to Stacker No. 1 (Item 6); records with D in Position 1 are sent to Stacker No. 2 (Item 7); records with E in Position No. 1 are sent to Stacker No. 1 by default (Item 8).

A complete program illustrating stacker selection appears in Section X.

7-21. Field Description Fields (Columns 43 through 70)

The next fields (Columns 43 through 70) on the Input Specifications Sheet describe the data fields within the records specified rather than record attributes in general. (In this discussion, the term *field* applies in general to any data field, array, or array item, unless otherwise noted.) Describe each field on a separate line on the specifications sheet, beginning at least one line below the line containing the record description (Columns 7 through 42). In each line containing a field description, leave Columns 7 through 42 blank.

You must always precede the field descriptions for a record with the appropriate record description — do not enter the field descriptions alone. However, you only need to describe those fields actually used by your program.

7-22. DATA FORMAT (COLUMN 43). If the field, array, or array item described on this line contains numeric data, indicate the format in which this data is stored with one of these entries:

Column 43 Entry	Meaning
P	Packed decimal format.
B	Binary format (1 or 2 word integer).
L	Unpacked (ASCII) decimal format with leading plus or minus sign.
R	Unpacked decimal format with trailing plus or minus sign.

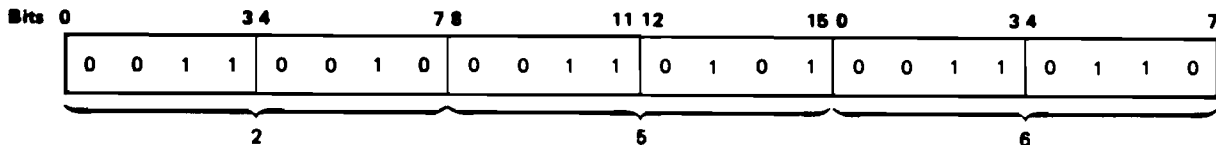
I Input Specifications

Column 43 Entry	Meaning
1-9.	Number of digits a binary field will use after being converted to packed decimal format. (See also Paragraph 7-31).
Blank	Unpacked decimal format with no leading or trailing signs, or alphanumeric format. (Unpacked decimal format is indicated by the presence of a digit in the <i>Decimal Positions Field</i> (Column 52); alphanumeric format is indicated by a blank in that field.)

Data in alphanumeric or packed decimal format is processed directly in that format. But numeric data in any other format (indicated by B, L, R, or blank) is converted to packed decimal format for internal processing; computations involving this data are also performed in packed decimal format.

7-23. Unpacked Decimal and Alphanumeric Formats. In unpacked decimal format, a field can contain any numeric characters. In alphanumeric format, a field can contain any alphabetic, numeric, or special characters. In both formats, ASCII representation is used with each character occupying an eight-bit byte of main memory. (Thus, a word of memory can hold two characters.)

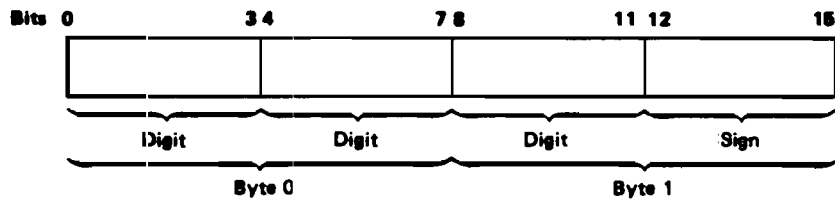
In unpacked decimal format, the number 256 would be represented as:



7-24. Unpacked Decimal Format with Leading Sign. This is similar to the unpacked decimal field just described, except that the *first* byte in the field contains the sign. For a negative value, it contains an ASCII minus sign (represented by the bit settings 00101101). For a positive value, it contains any other character. The sign character is not considered part of the numeric field. Therefore, if a field is defined on Input specs as being in positions 11 through 15, the sign is taken from position 11 and the data is taken from positions 12 through 15. The length of the field is four digits.

7-25. Unpacked Decimal Format with Trailing Sign. This is also similar to the unpacked decimal format, except that the *last* byte in the field contains the sign. For a negative value, it contains an ASCII minus sign; for a positive value, it contains any other character. The sign character is not considered part of the numeric field. Therefore, if a field is defined on Input specs as being in positions 11 through 15, the sign is taken from position 15 and the data is taken from positions 11 through 14. The length of the field is four digits.

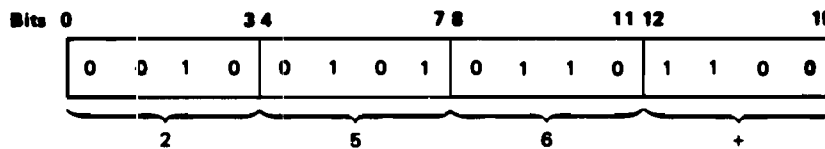
7-26. **Packed Decimal Format.** In packed decimal format, each word is divided into four digit portions, with the plus or minus sign of a numeric field contained in the rightmost (low-order) portion of the right-most byte:



In this way, packed format reduces the storage requirement to four bits per digit.

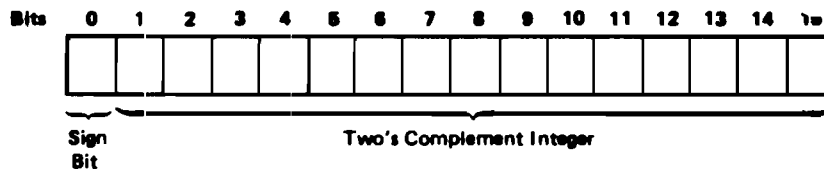
If the number is negative, the sign bits are set to 1101 (the character D in hexadecimal representation). If the number is positive, the sign bits are normally set to 1100 (the character C in hexadecimal representation); however, any sign except 1101 is recognized as positive.

In packed decimal format, the number 256 would be represented as follows:

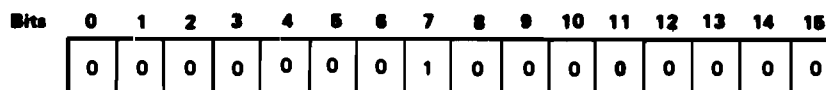


7-27. **Binary Format.** In binary format, numbers are stored in two's complement form. Each field is either two bytes (one word) or four bytes (two words) long.

The two-byte binary format, also known as single fixed-point format, permits representation of both positive and negative integers. Bit 0 is a sign bit, and the remaining bits 1 through 15 define the quantity. The range of possible integers is -32,768 through +32,767. Bit 0 is set to 0 for positive values, and to 1 for negative values.

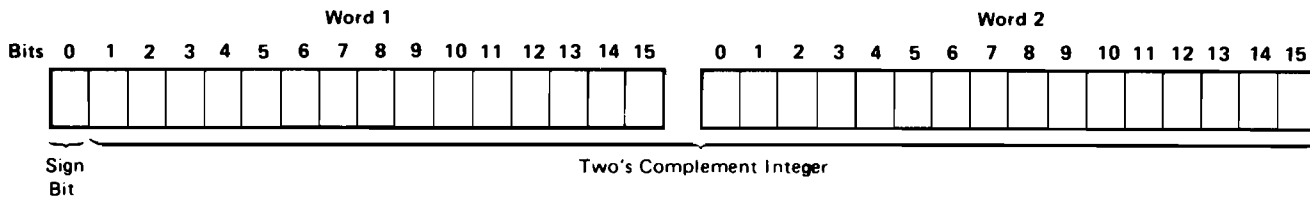


As an example, the value 256 would be represented in this format as:

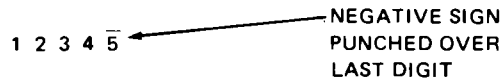


I Input Specifications

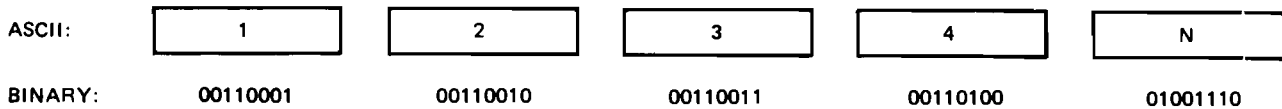
The four-byte format, also known as double fixed-point format, is the same as the two-byte format except that two words are linked together to form a 32-bit double-word quantity. Bit 0 of the most-significant word is the sign bit. The range of possible integers is from approximately -2 billion to +2 billion.



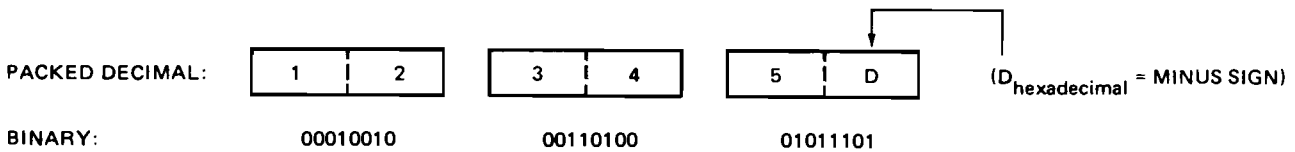
7-28. Conversion Method. When RPG reads an ASCII input field that is to be converted to a numeric field, or when RPG moves data from an alphanumeric to a numeric field, the data is converted in accordance with Table 7-1. For example, when signed data is input, the sign is indicated by a sign punch over the last digit in the field. For instance, the five-digit negative number - 1 2 3 4 5 is entered as:



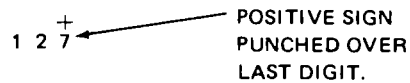
In ASCII representation, the combination of the last digit and its overpunch is translated as the character N, so that the data becomes:



Upon conversion to internal packed-decimal representation, the data becomes:



As another example, the three-digit positive number +127 is entered as:



In this case, in ASCII, the combination of the last digit and its overpunch is translated as the character G so that the data is interpreted as:

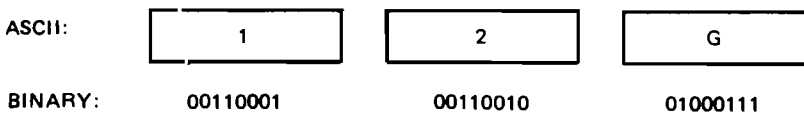
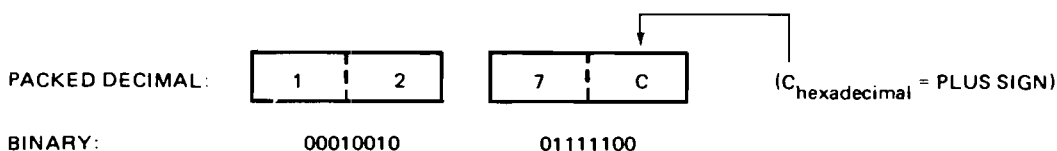


Table 7-1. Overpunch Conversion Chart For Right-most Digit

SIGN OF DATA	ASCII FORMAT										PACKED DECIMAL (HEXADECIMAL) FORMAT
	0	1	2	3	4	5	6	7	8	9	
NO SIGN (right-most digit uncharged).											F(1111 ₁₆) (assumed positive)
Positive Value (positive sign overpunched on right-most digit).	}	A	B	C	D	E	F	G	H	I	C(1100 ₁₆)
Negative Value (negative sign overpunched on right-most digit).	}	J	K	L	M	N	O	P	Q	R	D(1101 ₁₆)

Upon conversion to internal packed-decimal representation, the data becomes:



7-29. Specifying Data Formats. When you specify L or R in the Data Format Field, and the input data field contains a leading or trailing minus sign, the data is treated as a negative value; if the leading or trailing character is *not* a minus sign, the data is regarded as positive.

EXAMPLE

Suppose the input record field named DATA1 contains information in packed decimal format; you must enter P in the Data Format Field (Figure 7-1, Item 7).

7-30. FROM FIELD POSITION (COLUMNS 44 THROUGH 47) (REQUIRED). For each data field, array, or array item that your program uses, you must indicate where the field/array/item begins and ends; enter the first position in the *From Field Position Field*:

Columns 44 through 47 Entry	Meaning
A group of one to four digits, ranging from 1 to 9999.	Beginning position of field on input record.

Describe this position in relation to the leftmost position of the record (Position 1). In this entry, leading zeros are permitted but not required. The entry must end in Column 47.

EXAMPLE

Indicate that the field DATA1 begins in the second position of the input record by entering 2 in the From Field Position Field (Figure 7-1, Item 8).

I Input Specifications

7-31. TO FIELD POSITION (COLUMNS 48 THROUGH 51) (REQUIRED). Enter the last position of the data field in the *To Field Position Field*.

Columns 48 through 51 Entry	Meaning
A group of one to four digits, ranging from 1 to 9999.	Ending position of field on input record.

Describe this position in relation to Position 1 of the record. This position must be greater than, or be the same as, the beginning position; it must fall within the record length defined in the File Description Specifications.

Define a single-position field by entering the same number in both the *From Field Position* and *To Field Position Fields*.

In this entry, leading zeros are permitted but not required. The entry must end in Column 51.

To determine the *length* of an input field, subtract the value in the *From Field Position Field* from that in the *To Field Position Field*, and then add 1. The number of characters (for alphanumeric fields) or digits (for numeric fields) in the field depend upon both field length and data format (defined in the *Data Format Field*) as follows:

Format	Characters/Digits
Alphanumeric	Same number of characters as field length value. (For instance, a field beginning and ending with Position 5 has a length of 1 and can contain 1 character; a field running from Position 3 through Position 11 has a length of 9, and can contain 9 characters.)
Unpacked (ASCII) decimal with no leading or trailing signs	Same number of digits as field length value.
Unpacked (ASCII) decimal with leading or trailing sign.	Same number of digits as field length value, minus 1. (For instance, a field running from Positions 3 through 11 has a length of 9 and can contain 8 digits and an arithmetic sign.)
Packed decimal	Field length times 2, with 1 subtracted from this product.
Binary	Field length times 2-1/2. For instance, a field running from Positions 10 through 11 has a length of 2 and can hold 5 digits; a field running from Positions 20 through 23 has a length of 4 and can hold 10 digits. This can be overridden by specifying the actual digit count of a binary field in Column 43.

If you specify a portion of an array rather than an entire array or field, your program will read as many array items as it can store in a field of the length defined, beginning with the item in the first position defined for this array in the Input Specifications.

The maximum field lengths permitted are 256 for alphanumeric fields and 15 for all others.

EXAMPLE

Indicate that the field DATA1 ends in the ninth position of the input record by entering 9 in the *To Field Position Field* (Figure 7-1, Item 9). This field has a length of 8. Since it contains data recorded in packed decimal format, it can hold up to 15 digits.

7-32. DECIMAL POSITIONS (COLUMN 52). If this is a numeric field, indicate the number of positions appearing to the right of the decimal point.

Column 52 Entry	Meaning
Any digit 0 through 9.	The number of decimal positions in the field.
Blank.	The field contains alphanumeric data.

Although no decimal point actually appears within numeric data fields and arrays, implied decimal points are always associated with them. For ordinary data fields, you *must* specify the number of decimal positions by making an entry in the *Decimal Positions Field* of the Input Specifications; if the field contains no decimal positions, enter 0. For arrays, you must specify this value by making an entry in the *Decimal Positions Field* of the File Extension Specification (Paragraph 5-13); this value is optional in Input Specifications, but if you specify it here, be sure to enter the same value used in the File Extension Specifications.

Be certain that the number of decimal positions you specify does not exceed the maximum number of digits that the field can hold.

If this field contains alphanumeric data, leave the Decimal Positions Field blank.

EXAMPLE

Specify that the field DATA1 contains two decimal places by entering 2 in the *Decimal Positions Field* (Figure 7-1, Item 10).

7-33. FIELD NAME (COLUMNS 53 THROUGH 58) (REQUIRED). Enter the name of the input field, array, or array item you are describing:

Columns 53 through 58 Entry	Meaning
Field name. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters, dig-	The name of the input field.

I Input Specifications

Columns 53 through 58 Entry

Meaning

its, or @, \$, or #. Embedded blanks are not allowed.)

Array name. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.)

The name of the input array. (The name must have previously been defined as an array on File Extension specifications.)

Array name, comma, and index. The array name is as defined above. The index is either a number or the name of a field (as defined above) that contains a number. The combination of array name, comma, and index is limited to six characters.

The name of the input array item. The name must have previously been defined as an array on File Extension specifications. (See Paragraph 5-24 for further information.)

PAGE, or PAGE1 through PAGE7

The name of a field that provides the page number for output files.

*ERROR

The name of a one-character field used for run-time error codes.

You must assign a name to every field or array that your program uses. Then, whenever you reference this field or array in your program, do so by this name. By using an index, you can also reference an individual item within an array.

Within each record type, be sure that the name is unique for each field. If you assign the same name to more than one field within a type, your program will use only the last of these fields described. If you have already defined this name elsewhere in the program, be sure to associate it with the same field length, data format, and decimal positions in both instances.

You can assign the same name to fields from different record types if these fields are the same length and contain the same type of data. This is true even if the fields appear in different locations from type to type. You must describe each field individually, even though you use the same name in both instances. To eliminate duplicate coding, however, you can use OR Lines to indicate that the fields named can appear in either record type. (See the example in Figure 7-8.) Specify the first type; then, write OR in Columns 14 through 15 of the next line, and specify the second record type on that line; then, follow this OR Line with field descriptions on the next line.

Fields read from punched cards cannot exceed 80 characters. Fields used in arithmetic operations, and those to be edited or zero-suppressed, must be defined as numeric fields.

If you define an array item in the *Field Name Field*, and use a field name rather than a number for the index, you must define this index field as a numeric field with zero decimal positions, and no more than 9 digits long. If the index value is to be read from the current input record, be sure to define the index field on the record before attempting to use it—otherwise, the last index value assigned (if any) will be used. If your program establishes the index value through calculation operations, it must do so before it can process the record to which the index applies.

When an indexed array references more than one element, be sure to specify a field length that is an integral multiple of the element length.

See Paragraph 5-13 for more information on arrays and array items, and how to describe them in Input Specifications.

When your program prints a report containing more than one page, you may want to number each page. You can do this by referencing the special field PAGE in the Output Specifications whenever you want the page number to appear. (Initially, RPG/3000 pre-defines PAGE as a four-digit field initialized to zero, and adds one to this field prior to printing the page number.) Although pagination normally starts with one, you can arrange for it to start with another number by entering a value *one less* than that number in an input record field and naming that field PAGE on your Input Specifications. (For instance, to start with Page 10, enter 9 in the input record.) Each time your program reads data from the input file into this field, pagination restarts. When you define the PAGE field on the Input Specifications, always describe it as a numeric field with zero decimal positions; you can also change the length of this field if you wish. On the input record, always right-justify the page number, with or without leading zeros.

You can use up to seven additional field names--PAGE1 through PAGE7--for paginating different output files. If you use the same page-field name for more than one file, however, do so with caution to avoid confusion in your pagination system.

When an error occurs during object program execution, special error-codes are automatically transmitted to a one-character, alphanumeric field named *ERROR. Your program can check this field for information about errors that do not cause immediate termination, as noted in Section IX. If you wish, for any reason, to enter data in this field, you must define it in the *From Field Position*, *To Field Position*, and *Field Name Field* of the Input Specifications. When you do this, however, you must maintain the one-character field length.

EXAMPLES

Assign the name DATA1 to the first field in the first record type in Figure 7-1 (Item 11). This field contains packed decimal data with two decimal places, and runs from Position 2 through Position 9.

Assign the names FIELD1, FIELD2, and FIELD3 to the three ten-character fields in FILEX (described in Figure 7-9) in the record type identified by the letter A in the first position (Item 1).

Assign the names FIELDA, FIELDB, and FIELD C to the three fields in the next two record types described in FILEX (Item 2). Use the OR Line shown (Figure 7-9, Item 3) to avoid repetitious coding. This line implies that FIELDA, FIELDB, and FIELD C are each ten-character fields found in the same positions on records identified by either B or C in Position 1.

Suppose that you want to read page numbers from an input record field that is five characters long. You must then define PAGE as a five-character numeric field with zero decimal positions (Figure 7-9, Item 4).

7-33A. DATA STRUCTURES

Data structures enable you to break a field or an array into subfields which can be referenced in calculations and Output specifications. Data structures can also be used to group individual fields together into a single Data Structure field for reference.

Whenever a Data Structure field is input or modified, its subfields are automatically modified as well. Conversely, if a subfield is changed, its Data Structure is also changed.

Data structure specifications must follow all other Input specifications. When defining a Data Structure on your Input specifications, follow these steps:

1. Enter an *I* in column 6.
2. Enter an optional Data Structure name in columns 7-12. This name must follow the same naming conventions used for fields and arrays and may not exceed 6 characters. The name may be any of the following:
 - a. A new name which will be used to reference the Data Structure. In this case the length of the Data Structure will be determined by the highest end position of its subfields.
 - b. The name of an alphanumeric array which was previously defined in File Extension specifications.
 - c. The name of a field previously defined in Input specifications.
 - d. The name of a previously defined Data Structure subfield. (That is, Data Structure and subfield definitions can be "nested".)
 - e. blank

NOTE: The data structure name cannot be an array element or a table. If you are specifying the User Data Structure for your Local Data Area, the name is forced to **LDA**. Any other name you specify for the UDS will be ignored.

3. If you are specifying a Data Structure for your Local Data Area, enter a *U* in column 18. This indicates to RPG that the subfields of this Data Structure should be initialized from the Local Data Area at the very beginning of your program and used to update the LDA at Last Record time.
4. Enter *DS* in columns 19-20 to indicate that this is a Data Structure.
5. Leave all other columns blank.

Data Structure subfields are specified according to the following rules:

1. Enter an *I* in column 6.
2. Leave columns 7 through 43 blank.
3. In columns 44 through 47, enter the starting field position of the subfield, relative to the first position of the Data Structure.

I Input Specifications

4. In columns 48 through 51, enter the ending field position of the subfield, relative to the first position of the Data Structure.
5. In column 52, enter the decimal positions for a numeric subfield (0-9) or leave column 52 blank to indicate the subfield is alphanumeric.
6. In columns 53 through 58, enter the subfield name. The subfield name can be any of the following:
 - a. A new field name whose length and data type are determined by the entries in columns 44 through 52.
 - b. A field name which has been previously defined as an input field of a data record. In this case, the length (derived from columns 44–51) and decimal position (column 52) attributes must be the same as the original definition. The *from* and *to* positions refer to the position of the field within the Data Structure and not within the original data record.
 - c. An alphanumeric array name which has been specified on your File Extension specifications. The *from* and *to* positions must provide for the length of the entire array. An array element may not be specified as a Data Structure subfield.

The following rules apply to subfield definition:

- A subfield may not be an array element or a table.
- If a subfield is part of a named Data Structure whose length has been previously defined, then the subfield must be contained within that Data Structure. That is, the *to* position specified in columns 48 through 51 must not be greater than the length of the Data Structure.
- A subfield may not have the same name as a subfield in another Data Structure.
- A subfield may be later defined as a Data Structure, with its own subfields. That is, Data Structure and subfield definitions can be “nested”.
- The *from* and *to* positions for a subfield may be the same as, or overlap, other subfields within the Data Structure.

Application Note: The HEWLETT-PACKARD implementation of Data Structures does not redefine the same internal area multiple times as in other implementations of Data Structures. This enables extended features of Data Structures such as nesting and more freedom in referencing Data Structures within calculations.

EXAMPLES:

Figure 7-9a shows the use of a Data Structure to breakdown an input field (**PRODID**) into its subfields (**CATALOG**, **VENDOR** and **PRD#**). The advantage to using a Data Structure to accomplish this is that a change to a subfield will automatically change any overlapping subfields or, as in this case, the field specified as the Data Structure name. If, for example, a new value was moved to the field **VENDOR** in calculations, the field **PRODID** would immediately reflect that change.

HEWLETT-PACKARD

RPG INPUT SPECIFICATIONS

Page ____ of ____

Programmer _____ Date _____
 Program Title _____

Punching Instructions			
Graphic			
Punch			

Program Name									
--------------	--	--	--	--	--	--	--	--	--

Sequence Number	Form Type (1)	File Name	Group Sequence	Record Identification Codes			Field Position		Field Name	Control Level (L, U)	Matching or Chaining Fields (M, M, C, C)	Field Indicators		
				1	2	3	From (1-9999)	To (1-9999)				Plus	Minus	Zero or Blank
1		DATA	NS	01			1	10	PROD					
1							11	40	DDESC					
1							41	42	PRICE					
1		PRODID	OS				1	2	CATALOG					
1							5	5	VENDOR					
1							6	10	PRD					

Figure 7-9a. Breaking Down an Input Field

Figure 7-9b shows the use of a data structure to group two separate fields (PROD and DSCNT) representing a product from an order file and a discount code from a customer master file. The Data Structure name PKEY becomes a key used in chaining to a price file.

HEWLETT-PACKARD

RPG INPUT SPECIFICATIONS

Page ____ of ____

Programmer _____ Date _____
 Program Title _____

Punching Instructions			
Graphic			
Punch			

Program Name									
--------------	--	--	--	--	--	--	--	--	--

Sequence Number	Form Type (1)	File Name	Group Sequence	Record Identification Codes			Field Position		Field Name	Control Level (L, U)	Matching or Chaining Fields (M, M, C, C)	Field Indicators		
				1	2	3	From (1-9999)	To (1-9999)				Plus	Minus	Zero or Blank
1		CUSTOMR	NS	01			1	10	CUSNUM					
1							11	40	CUSNAM					
1							41	42	DSCNT					
1		ORDER	NS	02			1	8	PROD					
1							9	12	PARTY					
1		PRICE	NS	03			1	10	KEY					
1							11	17	COST					
1		PKEY	OS				1	8	PROD					
1							9	10	DSCNT					

Figure 7-9b. Grouping Two Separate Fields

I Input Specifications

7-34. CONTROL LEVEL (COLUMNS 59 THROUGH 60). If this field is not part of a chained or demand file, or is **not a look-ahead** or trailer field, you can assign to it any of nine control level indicators.

Columns 59 through 60 Entry	Meaning
L1 through L9	Assign this control level indicator to the field.
Blank	This field is not a control field.

When you assign a control-level indicator to a field, the indicator identifies it as a *control field* that your program **must check** each time it reads a record. The program compares the control field data on this record **with that on the previous record**. If the control field contains identical information, the record belongs to the **same control group** as the previous record. But, if the control field information differs, a *control break* occurs. In this case, the program turns on the indicator associated with this field--and all control-level indicators of lower rank--, and performs all calculation and output operations associated with those indicators. This sequence allows you to use control-level indicators to condition:

1. Operations to be done when the first record of a new control group is read.
2. Operations to be done when all records within a control group have been read.
3. Output of totals and sub-totals.

Among the indicator designations L1 through L9, increasing digit numbers indicate increasing importance or higher level. For instance, L9 is ranked higher than L8, L8 higher than L7, and so forth. When a control break forces your program to turn on a particular indicator, all those of lesser rank also turn on. For example, if a field has a control level indicator L3 and the contents of that field change from A243 to B1A2, then this control level indicator *and* control level indicators L2 and L1 are set on. Thus, the relative importance of data in one control field to that in others should help you determine what indicator to assign to the fields. For instance, a field that contains data on states traditionally has a higher indicator than one containing data on counties; the counties field would, in turn, have a higher indicator than a field dealing with cities.

An additional control level indicator, L0, is always on. You cannot assign it to a control field, but you can use it to condition calculation operations as discussed in Section VIII. Still another indicator, LR, turns on when your program reads the last record of the program. This is the highest level indicator available in RPG/3000. Thus, when it turns on, indicators L1 through L9 automatically turn on also.

You can direct operations on the basis of *split control fields*. Each of these fields is actually made up of more than one regular data field. You can create a split field by assigning the same indicator to two or more regular fields. RPG/3000 combines these fields in the order specified on the Input Specifications Sheet, and compares **their contents** in this order, treating them as a single control field. The lengths of the component fields can vary from record type to record type, but the total length of each control field within a record type **must be the same**. The total length for a split control field cannot exceed 256. Within a record type, the component fields need not appear adjacent to each other; however, you must not insert other control level fields **between the components** of any split control field.

Control breaks commonly occur after your program reads the first input record, because the program compares the control field contents to the contents of an empty storage area in memory. In this case, total calculations and output (defined in Sections VIII and IX) do not take place.

You need not assign control level indicators in any particular sequence, nor in contiguous order. Thus, you could assign L5, L1 and L7, in that order, and ignore all other indicators.

Control level indicators operate independently of the files to which they are assigned. Thus, if your program reads a record from FILEA having the L3 indicator assigned to one field, and then reads a record from another file having the L3 indicator assigned to a field--and the contents of the second field is different from that of the first--a control break occurs.

In comparing control field contents, your program will ignore arithmetic signs. Thus, if the control field on one record contains +5 and the same field on the next record contains -5, a control break will not occur.

Decimal positions in numeric control fields are also ignored. Thus, if the control field in one record contains 2.11 and the same field in the next record contains 211, no control break occurs.

EXAMPLES

Suppose you are writing a program that provides data on voter registration within a state. The fields on each input record specify the voter's name, street address, city, county, and precinct. (The Input Specifications for this program appear in Figure 7-10.) After the program reads each record, the program increments a count of voters in the precinct by 1. When the last record in a precinct is read, a control break occurs because the contents of the PRECNT field, associated with the L1 indicator, changes (Item 1). Now, the program performs the operations conditioned by the L1 indicator--it prints the total number of voters in the precinct, adds this value to a counter for total voters in the city, and clears the precinct counter. Then, the program continues. When the contents of the control field CITY changes, a higher level control break occurs because the L2 (Item 2) and L1 indicators both turn on. Now, the program performs operations conditioned by both indicators--it prints the number of voters in the precinct, adds this value to the city counter, prints the number of voters in the city, adds the contents of the city counter to the county counter, and then clears both the precinct and city counters. When the contents of the COUNTY field changes, the L3 (Item 3), L2, and L1 indicators all turn on, and totals for the county, in addition to those for the city and precinct are calculated and printed in a similar fashion. Finally, when the end-of-file is encountered on the file VOTERS, the program turns on the last record (LR) indicator. It then performs all operations conditioned by this indicator (and indicators L1 through L9), such as calculating and printing the total number of voters in the state as well as those for precinct, city, and county, and terminates.

7-35. MATCHING/CHAINING FIELDS (COLUMNS 61 THROUGH 62). If this field is a matching or chaining field, enter the appropriate matching field code. Otherwise, leave this entry blank.

Columns 61 through 62 Entry	Meaning
M1 through M9	This is a matching field identified by this code.
C1 through C9	This is a chaining field identified by this code.
Blank	This field is neither a matching or chaining field.

I Input Specifications

HEWLETT PACKARD

RPG INPUT SPECIFICATIONS

Page 3 of 5

Programmer TOM STANLEY Date 2/2/75

Program Title RETURNS

Punching Instructions

Graphic					
Punch					

75 76 77 78 79 80

Program Name RETURNS

Sequence Number	Form Type (1)	File Name	Group Sequence	Number of Records (1/N)	Option (O)	Record Indicator/ Look Ahead/Trailer	Record Identification Codes			Field Position		Field Name	Control Level (L1 L9)	Matching or Chaining Fields (M1-M9,C1-C9)	Field Record Relation	Field Indicators			
							Position (1-9999)	Not (N) Portion (C/Z/D) Character	Position (1-9999)	Not (N) Portion (C/Z/D) Character	From (1-9999)					To (1-9999)	Plus	Minus	Zero or Blank
1	I	VOTERS	AA	01			1	C	A	5	25	NAME							
2	I									26	30	ADDR							
3	I									31	35	CITY	L2						
4	I									36	40	COUNTY	L3						
5	I									41	45	PRECWTL	L1						

Figure 7-10. Input Specifications for Voter Count Program

7-36. Matching Fields. If your program uses two or more input, update, or combined files, you can request it to:

1. Check the records on these files for proper sequence, based on one or more specified fields, and
2. Compare the contents of these fields in records on different files, to determine if they match. (This will allow you to condition various calculation and output operations to take place only when a match occurs.)

The sequence-checking/matching-field feature applies to all files (including KSAM, INDEX, and IMAGE files) specified for sequential access.

If your program uses only one input, update, or primary file, you can request sequence checking (although record-matching does not apply).

Request sequence-checking and matching by entering a *matching field code* in the *Matching/Chaining Field* of the Input Specifications as discussed below. This entry is valid, however, only for primary or secondary files.

Multi-file Processing. When your program reads input records from more than one file, it is performing *multi-file processing*. Although the program always selects the records from one input file and one or more secondary files in such cases, the order of selection depends upon the presence or absence of matching fields on the records. When you do not specify matching fields, records are selected for processing from one file at a time: first, from the primary file and then--after all primary file records are processed--from each secondary file in the order you have described these files in the *File Description Specifications*. When you specify matching fields, however, records from all files to be read are merged on the basis of the matching fields and are then selected according to the contents of these fields; when fields on records in two or more files contain identical data, the fields match, and all matching records on the primary file are processed first, followed by matching records on the secondary files in the order specified. (The precise sequence is described below.)

By assigning matching field codes to various fields, you can request matching by one field, many fields, or entire records. In doing this, you can use up to nine individual codes (M1 through M9). Each code designates the associated field as a *matching field*. If you specify more than one matching field, your program combines these fields into a single control field, and uses this field for sequence-checking and comparing the records. (If you specify only one matching field, this field itself becomes the control field.) Whenever the data from the control field on a record in the primary file is identical to the data in the same field on a record from a secondary file, the records match. In this case, the matching record (MR) indicator turns on, the program performs the operations conditioned by that indicator, and the program reads the next record. When it encounters a record that does not match, the program turns the indicator off. An example appears later.

Note: Remember that the codes M1 through M9 are NOT indicators--they are only designators used to identify matching fields. They do, however, cause the MR indicator to turn on when a match occurs.

When you assign matching field codes for multfile processing, follow these rules.

1. In the *Input Sequence Check Field* (Column 18) of the *File Description Specifications*, be sure to specify the same order--ascending or descending--for the files whose records are to be matched with each other.

I Input Specifications

2. Assign matching field codes to at least one record type on at least two files--otherwise matching will never occur. However, you need not specify matching for all files used by the program, nor for all record types in a file.
3. Specify the same number of matching fields for all record types matched with one another, and use the same codes within these types. (You do not, however, need to assign the code to fields of the same name--field names themselves are not used in matching.)
4. Be sure that all fields with the same code are the same length and contain data in the same format (alphanumeric or numeric). Matching fields in binary format are not allowed.
5. Be certain that the combined length of all matching fields--the total control field--does not exceed 256.
6. When you use two or more codes, assign them to fields in the order you want these fields combined to form the control field. The fields are combined according to the descending sequence--M9 through M1--of the associated code. Thus, if you assign M3 to a field containing XXX, M1 to one containing YYY, and M7 to another containing ZZZ, the fields will be combined contiguously as:

ZZZ	XXX	YYY
-----	-----	-----

7. Do not assign the same matching field code (M1 through M9) to more than one matching field within a record type--matching fields cannot be split in this manner.

When your program compares matching fields, it follows these rules:

1. The program treats all matching fields with the same code as numeric, if *any* of those fields is described as numeric in the Input Specifications. Therefore, if you assign the M9 code to two fields, described as numeric for one record type and alphanumeric for another, and the field contains 03 on one record type and a blank followed by the digit 3 on another, the fields will match.
2. The program ignores arithmetic signs and decimal points in matching numeric fields. Thus, a numeric field containing 7 will match with one containing -0.07.
3. When more than one code is used, all fields in the control field must match before the MR indicator turns on. Thus, if you assign M5, M4, M3, and M2, all four fields from the primary file must match all four fields from the secondary.
4. The program ignores field names during matching--it will attempt to match fields with identical codes from different record types, whether or not these fields have the same name.
5. If you have defined an alternate collating sequence for your program, your program will match records based on this sequence.
6. All fields with the same matching field code must be the same length.

During multifile processing, your program performs the operations described below; an example appears in **Figure 7-11**.

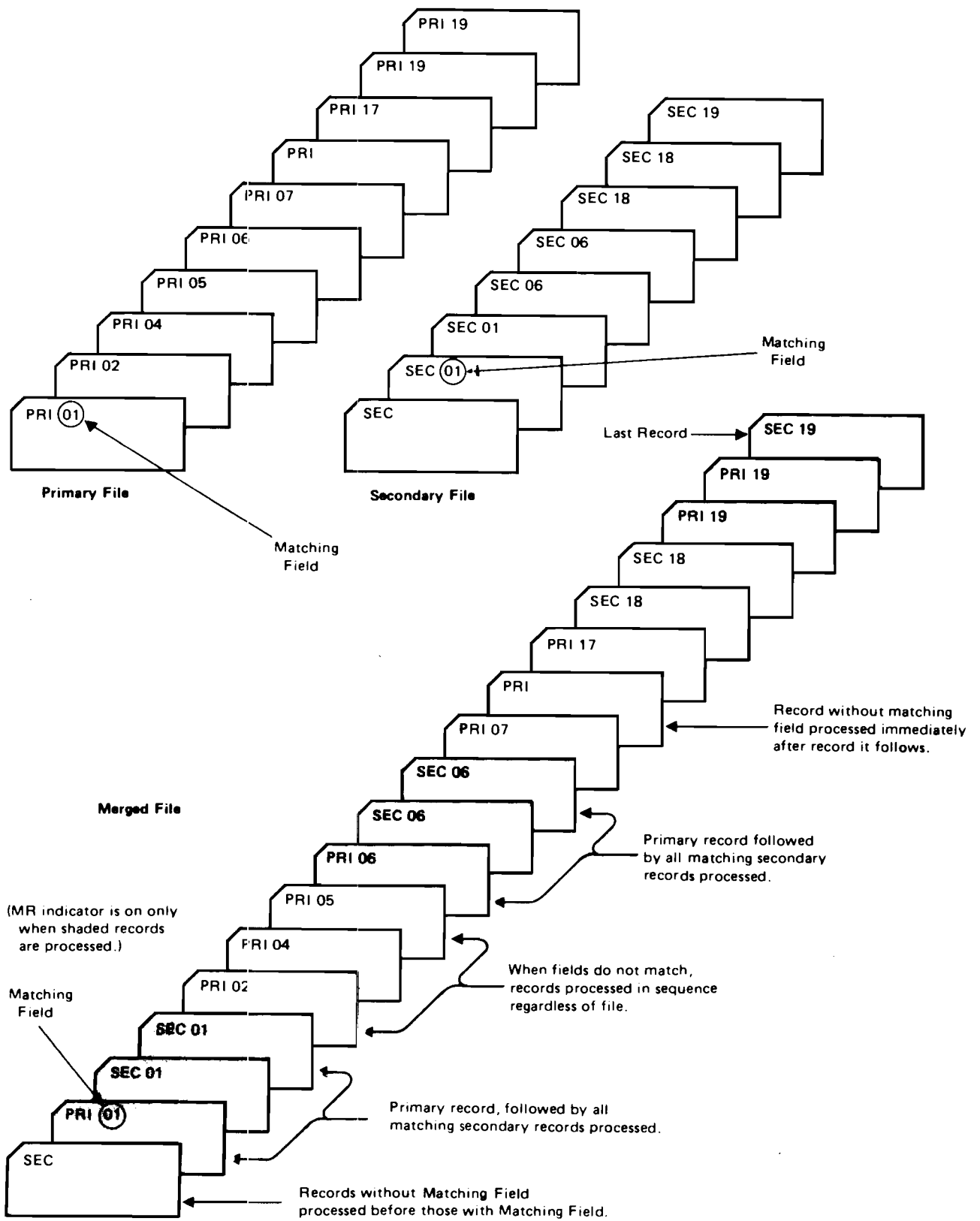


Figure 7-11. Multi-file Processing

I Input Specifications

1. The program merges the files to be compared, sequence-checking the records on the basis of the matching control field. If it detects a sequence error, the program responds according to the option you select in the *Error Response Field* of the Control Record Specification.
2. The program reads one record from every file. If the first type encountered has no matching field code, this record is processed even if it is not from the **primary** file. But, if all records have matching fields and a record from the primary file matches one from a secondary file, the program turns the MR indicator on and processes the primary file record first. Next, it processes any additional matching primary records and then the secondary file records in order of file priority (unless another file is forced, as discussed in Section VIII.)
3. After the program processes the record selected, it reads another record. At the beginning of the next program cycle, it compares this record with those not selected during the previous cycle, and chooses one for processing.
4. If the program encounters a record type that has no matching field code, it turns the MR indicator off and processes this record immediately after the previous record. If two or more records being compared have no match fields, they are processed in the order specified in the Input Specifications.
5. Whenever records from ascending files do not match, the program processes the record with the lowest match-field content first; when records from descending files do not match, the program processes the record with the highest match field content first.
6. The program always sets the MR indicator ON when one or more secondary file records match the primary record; this indicator remains on for one complete program cycle. If all primary file records match all secondary file records, the MR indicator is always ON. (The MR indicator does NOT turn on when two secondary records match with each other, but not with a primary, even if the primary record has no matching field code.) If record types for which no matching fields have been specified are read, the MR indicator turns off.

Single-File Processing. When your program reads records from only one file, you can request sequence-checking of the records within it on the basis of match fields designated by matching field codes. Do this simply by assigning any or all of the codes M1 through M9 to the desired fields. If you specify more than one field, they will be treated as a single control field. Be sure to specify the correct order--ascending or descending--in the File Description Specifications. If your program detects a sequence error, it will print an error message, and either terminate or continue (depending on the option you select in the *Error Response Field* of the Control Record Specification.)

EXAMPLES

An Input Specification for sequence-checking records from a single input file appears in Figure 7-12. The file is named INFILE, and the match fields are DEPT, DIV, and BRANCH. Three records from this file also appear in Figure 7-12. Since all fields identified are regarded as a single continuous field, they will be checked as shown below. If you specify ascending order in the File Description Specification, a sequence error will occur with the third record.

```
01005003
01006002
01004001      (Sequence Error)
```


I Input Specifications

Input Specifications for two record types used in matching multifile records appear in Figure 7-13. Both types have two matching fields and use M1 and M2 to indicate these fields. The MR indicator turns on only when both fields in the file ALPHA match both of those in the file BETA.

7-37. Chaining Fields. Chaining is a method for randomly retrieving selected records from a direct-access, IMAGE, KSAM, or INDEX file. RPG/3000 allows you to perform chaining in two different ways:

1. Using the *CHAIN operation code* in Calculation Specifications, as described in Section VIII. (Most programmers prefer this method because it is more flexible.)
2. Using *chaining field codes* in Input Specifications, as described below.

Chaining field codes designate input record fields that, in turn, reference the addresses of the records you wish to retrieve from the direct-access, IMAGE, KSAM, or INDEX file. The file whose records contain the chaining fields is called the *chaining file*; the file from which the records will be retrieved is called the *chained file*. Because the chained file is accessed randomly, it must always reside on disc.

Chaining fields contain either:

1. Numeric entries specifying the relative disc addresses of the records to be accessed on the chained file, or
2. Alphanumeric or numeric entries used as keys to find a record on the chained file. In this case, the chained file must be a KSAM, INDEX, or IMAGE file processed by key.
3. Alphanumeric or numeric entries that will be translated by a conversion routine into relative disc addresses or record keys of the records on the chained file. The conversion routine is named in the File Extension Specifications and executed in the Calculation Specifications.

The principle of chaining is illustrated in Figure 7-14. Suppose that a program reads records from a card file that shows a firm's customer transactions and the amount of money each customer owes. These records do not show the mailing addresses of the customers, but the program can obtain these addresses from a masterfile on disc simply by referencing a chaining field on the card file records. On each card, the chaining field would contain the customer name which is used as a key to obtain the record containing the corresponding customer's mailing address and other information about him. The mailing address will then be made immediately available to the program, which can then print a bill for each customer ready for mailing. The coding for these operations will be discussed in the next example.

Designate chaining fields by assigning to each one a unique chaining field code. You must enter this code in the Chaining Field Code Field on BOTH the File Extension and Input Specification Sheets.

You can identify up to nine chaining fields per record type, using the codes C1 through C9. The codes do NOT imply a numeric order for chaining or any hierarchical level — they only indicate nine possible chaining fields within one record type. The order of chaining is based on the sequence you specify in the Input Specifications. Thus, if you assign code C3 to the first field defined on the sheet, C2 to the next, and C5 to the third, the fields will be chained in this order: C3, C2, C5.

You can assign the same code to more than one field, creating a *split chaining field*. In this way, you can request your program to treat several adjoining fields, or even non-adjoining fields, in an input record as one

I Input Specifications

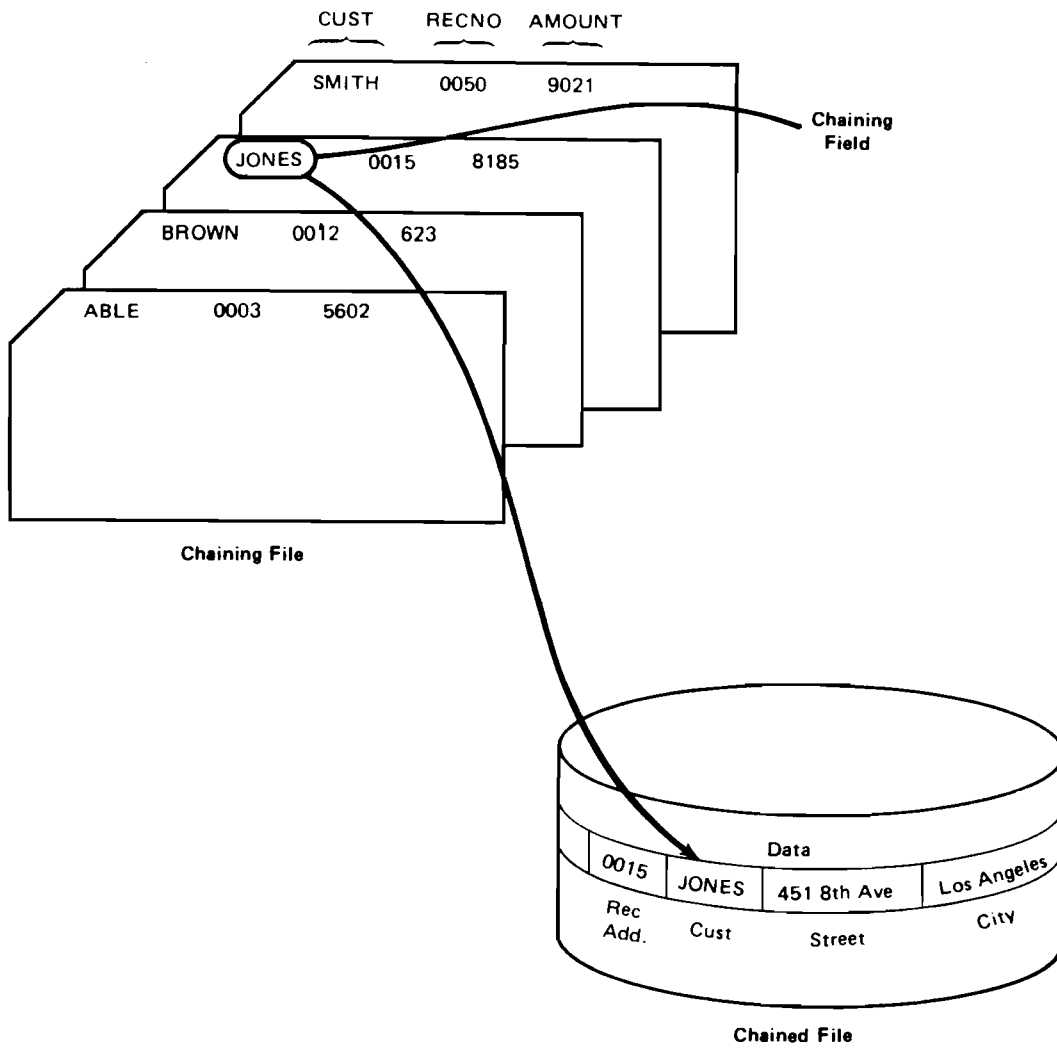


Figure 7-14. Chaining to a Direct-Access File

chaining field. The first field designated by this code on the specification sheet comprises the leftmost positions of the total chaining field; the last field assigned to the same code comprises the rightmost positions of the total chaining field.

If you do not supply a conversion routine, the chaining field will be treated as a relative record number for a direct chained file or as a record key for a KSAM, INDEX, or IMAGE chained file. For instance, if the first chaining field for a direct chained file contains 0000, this indicates the first record in the chained file; 0001 indicates the second record, and so forth.

Chaining field codes cause chaining to take place only before detail time in the RPG/3000 program cycle.

When assigning chaining field codes, follow these rules:

1. Do NOT use look-ahead fields or arrays as chaining fields.
2. Do NOT enter L or R (for leading or trailing arithmetic signs) in the *Data Format Field* (Column 43) of the Input Specifications; if you do, the object program will ignore any chaining field code specified for this record type.
3. Do NOT intersperse lines specified with split chaining fields with lines specified with other chaining fields.
4. If you use split chaining fields for relative disc addresses, you MUST use a conversion routine to convert the data in these fields to disc addresses.
5. If the chaining field for a relative disc address is NOT a numeric field, you MUST specify a conversion routine.
6. To chain the same input field to more than one disc file, define that field more than once on the Input Specifications, but assign it a different chaining field code each time.

When requesting chaining to an IMAGE/3000 file through Input Specifications, use alphanumeric or numeric keys. You may use split chaining fields, but you cannot have intervening chaining records between split chaining fields. All chainings requested from a single record will be accomplished in the order specified. If you request chaining to the same file twice, the last chaining operation only will be done. Chaining fields for IMAGE files should be an even number of bytes in length, and start on word boundaries. RPG/3000 establishes the chaining key according to the type specified in Column 31 of the File Description Specifications. (In the Input Specifications, the key field length must be the same as that defined in the File Description Specifications.)

EXAMPLE

The File Description, File Extension, and Input Specifications for the billing program mentioned earlier in this discussion appear in Figure 7-15. The Transaction records are stored on a chaining file named TRANSAC (Item 1). The customer's mailing address (and other data) resides in a chained KSAM file named MASTER (Item 2). The TRANSAC file is chained to MASTER (Item 3), and the field CUST on the records in TRANSAC is used as the chaining field (Item 4). On each record, CUST contains the name identifying the associated record on MASTER. Then, when the program reads the record for the customer JONES on TRANSAC, it will chain to the record containing JONES' mailing address on MASTER.

7-39. FIELD RECORD RELATION (COLUMNS 63 THROUGH 64). When you use OR lines to describe record types, but all fields described are not common to all types, you can use the Field Record Relation Field to relate particular input fields to specific record types.

Columns 63 through 64 Entry	Meaning
01 through 99	General indicator relating this field to a specific record type.
F0 through F9	Function Key Indicator relating this field to a specific record type.
KA through KN, KP through KY	Command Key Indicator regulating acceptance of data from a field.
L1 through L9	Control-level indicator regulating acceptance of data from a field if a control break occurs.
MR	Matching record indicator regulating acceptance of data from matching fields.
U1 through U8	User indicator conditioning use of a field.
H1 through H9	Halt indicator relating this field to a specific record type.
OA through OG, OV	Overflow indicator.
1P	First-page indicator, used as a general indicator.
LR	Last-record indicator.
Blank	This field appears in all record types covered by this OR relationship.

When two or more record types contain the same fields, you can reduce the amount of code required to describe them by using OR Lines. With this method, you need to describe each field only once even if it appears on several different record types--the field will then be associated with all record types covered by the OR Lines. But, suppose the record types you are describing share only *some, not all* fields in common. RPG/3000 allows you to distinguish those fields that are not common, relating them to specified records only, while still using OR Lines to minimize coding. You do this by entering, in the *Field Record Relation Field*, an indicator previously used in the description of the record type on which the field appears. When this indicator is on, it conditionally identifies the field as belonging to the associated record type.

Note: You should only enter an indicator that is set or re-set by RPG/3000 during the object program cycle if you fully understand why and when this setting occurs. For further information, see Appendix E.

As examples, you could enter in this field a general indicator initially used in the *Record Indicator Field* (Columns 19 through 20) of the record type on which the field is found. You could also specify a general indicator defined elsewhere in your program, which you would use to condition the movement of

I Input Specifications

data from input records into a storage area. You can also relate control fields defined in the Control Level Field (Columns 59 through 60) and matching fields defined in the *Matching/Chaining Field Field* (Columns 61 through 62) to a particular record type.

When two control fields have the same control level indicator or two matching fields have the same matching code, and you enter this control level indicator or the matching record indicator (MR) in the *Field Record Relation Field* for just one of these fields, only the field designated by the *Field Record Relation* entry will be used when the indicator is on. But, when the indicator is not on, the field without the *Field Record Relation* entry will be used if this field is described first.

When you wish to accept and use data from a particular field only when a certain condition (such as matching records or a control break) occurs, you can indicate the conditions under which you accept data from a field by using indicators L1 through L9 or MR. Data from the field named is accepted only when the indicator is on.

You may also condition a specification by a user indicator (U1 through U8). The user indicator, which you set prior to processing, determines whether a field is to be used in the program. When the indicator is on, the field is read; when the indicator is off, the field is not read.

During object program execution, the last field specified with a field record relation indicator that is on, will be the Control Level, matching, or chaining field used.

When you enter an indicator relating to control level, matching, or chaining fields in the *Field Record Relation Field*, follow these rules:

1. Specify all fields (including matching or control fields) that have no field record relation indicator before those that do.
2. Group together all fields related to one record type (that is, having the same Field Record Relation entry). Within such groups, however, you can enter these fields in any order.
3. Assign the same field record relation indicator to all portions of a split control field, and describe these portions as a group in specification lines following one another.
4. When you use a field record relation indicator with matching or control fields, it must match a record identifying indicator for this file. Group the matching or control fields according to the field record relation indicator.

EXAMPLE

Suppose that a file named CARDX contains two different types of records, as shown in Figure 7-16. You could use the general indicator 01 (Item 1) to relate the field ALPHA to the record type identified by an A in Position 1; you could also use another indicator, 02, to relate the field BETA to the type designated by a B in Position 1. This would mean that ALPHA appears only on the first record type, while BETA appears only on the second. The fields GAMMA and DELTA, however, which have no entries in the *Field Record Relation Field*, are common to both record types.

I Input Specifications

7-39. FIELD INDICATOR FIELD (COLUMNS 65 THROUGH 70). This entry allows you to test a field to determine if it contains a positive or negative value, zeros, or blanks.

Columns 65 through 70 Entry	Meaning
01 through 99	Indicator used to test data in field. (If entered in Columns 65 through 66, tests for positive data; if in Columns 67 through 68, tests for negative data; if in Columns 69 through 70, tests for zeros or blanks.)
F0 through F9 KA through KN, KP through KY	
L1 through L9	
MR	
U1 through U8	
H1 through H9	
OA through OG, OV	
1P	
LR	
blank	Do not test data in field.

You can test numeric input fields for positive, negative, or zero values, and alphanumeric fields for blank contents, by entering any of the above indicators in the appropriate subfield:

Subfield Columns	Test
65 through 66	Numeric fields for positive data.
67 through 68	Numeric fields for negative data.
69 through 70	Numeric fields for all zeros or all blanks, or alphanumeric fields for all blanks.

The indicator you name will turn on if the test is affirmative. It remains off (or turns off if already on), if the test is not affirmative. You can use different indicators for various purposes. For instance, you can use a general indicator to control calculation or output operations. You can use a halt indicator to check for an error in your data. For example, if a field should not contain a negative value, you can specify a halt indicator in Columns 67 through 68. Then, if such a value is encountered, the halt indicator turns on (when the record is fully processed) and your program will terminate or continue depending on the option you select in the *Error Response Field* of the Control Record Specification.

When your program tests a numeric field for zeros, the indicator you specify in Columns 69 through 70 turns on if the field contains either zeros or blanks (or a combination of zeros and blanks) exclusively. But, when your program tests an alphanumeric field for blanks, the indicator turns on only if the field contains blanks alone — a field comprised exclusively of zeros does not affect the indicator in this case.

When you use the indicators specified to condition calculation or output operations, you must keep in mind when these indicators are set on and off. When assigning and using the indicators, remember that:

1. Indicators for positive and negative values are off at the beginning of the program. They are not turned on until the program reads a record with a positive or negative value in the field being tested.
2. An indicator assigned to zero or blank is off at the beginning of the program, unless you set this indicator on by using the *Indicator Setting Field* (Column 42) of the Control Record Specification. It remains off until the program reads a record that contains zeros or blanks in the field being tested.
3. You can assign two or three field indicators to one input field. However, only the indicator which signals the result of the test turns on; the others are turned off.
4. If you assign the same field indicator to fields in different record types, its status is always based on the last record type selected.
5. When you assign different field indicators to fields in different record types, a field indicator turned on will remain on until another record of that type is read. Similarly, if you assign a field indicator to more than one field within a single record type, it will always reflect the status of the last field defined.
6. You can set field indicators on and off with the SETON and SETOFF operations in Calculation Specifications.

EXAMPLE

In the Input Specifications in Figure 7-17, three field indicators appear. The first one tests the field POSDAT for positive data (Item 1), the second tests the field NEGDAT for negative data (Item 2), and the third tests the field NODAT for zero or blank content (Item 3). When any of these indicators turns on, the calculations or output conditioned by them occurs.

7-40. Program Name (Columns 75 through 80)

This field contains the program name or any other information, as discussed in Paragraph 2-6.

7-41. DEFAULT SUMMARY

If you leave the optional fields of the Input Specifications blank, the default specifications shown in Table 7-2 apply:

Table 7-2. Input Specification Default Values

Columns	Field	Default Specifications
1 through 5	Sequence Number	No sequence number applies.
17	Number of Records	No restrictions on types per group; sequencing is not requested.
18	Option	At least one record of the type specified must be present in each group, or an alphabetic sequence entry is specified in the Group Sequence Field.
19 through 20	Record Indicator/Look-Ahead/Trailer/Data Structure	No record-identifying indicator, look-ahead, trailer field, or Data Structure applies.
21 through 41	Record Identification Codes	A record identification code does not apply.
43	Data Format	Unpacked decimal format with no leading or trailing signs, or field contains alphanumeric data.
44 through 47	From Field Position	No field is defined.
48 through 51	To Field Position	No field is defined.
52	Decimal Positions	The field contains alphanumeric data.
53 through 58	Field Name	No field is being described.
59 through 60	Control Level	This field is not a control field.
61 through 62	Matching/Chaining Fields	This field is neither a matching nor a chaining field.
63 through 64	Field Record Relation	This field appears in all record types covered by this OR Relationship.
65 through 70	Field Indicator Field	Do not test data in this field.
75 through 80	Program Name	None



CALCULATION SPECIFICATIONS

SECTION

VIII

The general purpose of most RPG programs is to read data, perform operations using this data, and store and print the results of these operations. The operations you can select include: addition, subtraction, multiplication, division, and square root calculations; testing, comparing, and moving data; turning indicators on and off; branching; searching tables; and others. Most RPG programmers traditionally call all these operations *calculations*, although many are not really computational or arithmetic functions. You describe these calculations on the *Calculation Specifications Sheet*, in the order in which you want them performed. On this sheet, each specification determines:

1. *Whether to perform the calculation.* You specify this by entering, in Columns 7 through 17, one or more indicators that determine the conditions under which the calculation will be done.
2. *What kind of calculation to perform on which data.* You specify the operation in Columns 28 through 32, the data in Columns 18 through 27 and/or 33 through 42, and the field where the results will be stored in Columns 43 through 51.
3. *What tests to make on the results of the calculation.* You specify these tests by entering indicators in the *Resulting Indicators Field* (Columns 54 through 59). These indicators can also be used to condition other operations later in your program.

In your program, you can request three types of calculations: detail, total, and subroutine calculations. You must request them in the order shown below:

1. *Detail Calculations* are normally performed for each individual record, subject to any conditioning indicators you specify. For instance, a program that calculates the pay earned by each employee in a company would perform detail calculations each time an employee's record was read. **If you want** your program to perform detail calculations only under specified conditions, enter general indicators in the *Indicator Field* (Columns 9 through 17). For detail calculations, always leave the *Control Level Field* (Columns 7 through 8) blank.
2. *Total Calculations* are customarily performed on data accumulated for a group of related records (a control group); they are done immediately following a control break--when the content of the control field of the current input record differs from that of the previous record. For instance, if the payroll program noted above keeps track of the total pay accrued by all employees within a department, total calculations would be made for all records within that department. Because these calculations are conditioned by control breaks, you must always associate them with control level indicators L0 through L9, or LR, entered in the *Control Level Field*.
3. *Subroutine Calculations* are special-purpose calculations within subroutines called by your RPG/3000 program. To specify them, enter SR in the *Control Level Field*.

The rules covering these types of calculations and how to specify them are described below.

C Calculation Specifications

8-1. FIELDS

The RPG/3000 Calculation Specifications contain the following fields:

8-2. Sequence Number (Columns 1 through 5)

This field contains the source record sequence number, described in Paragraph 2-3.

8-3. Form Type (Column 6) (Required)

This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
C	Calculation Specification

Because this entry is required and is always the same, it is pre-printed on the specification sheet for your convenience (Figure 8-1, Item 1).

8-4. Control Level (Columns 7 through 8)

The entry in this field determines when your program performs the calculation requested on this line:

Columns 7 through 8 Entry	Meaning
L0 through L9	A total calculation to be done when appropriate control break occurs or when the indicator is set on by a SETON operation; normally done at total time in RPG/3000 program cycle. L0 is always on.
LR	A total calculation done after the last record has been processed, or after LR indicator is set on by the program.
SR	Calculation operation defined in a subroutine.
Blank	Detail calculation, done at detail time in the RPG/3000 program cycle or a subroutine calculation if a BEGSR operation has been specified prior to or on this line.
AN	Establishes AND relation between indicators on this line and those on previous line.
OR	Establishes OR relation between indicators on this line and those on previous line or in previous AND relationship.

C Calculation Specifications

In an RPG/3000 program, calculations must be requested in the following order, in the fashion indicated:

Calculation Type	Control Level Field Entry
Detail	Blank
Total	L0 through L9, or LR
Subroutine	SR

When you specify a detail calculation by leaving this field blank, your program performs the calculation every time it reads a record, provided that the indicators appearing in the *Indicators Field* (Columns 9 through 17) permit this.

When you specify any other calculations, you must enter a control level or last record indicator, or a subroutine identifier in this field.

8-5. CONTROL LEVEL INDICATORS (L0 THROUGH L9). When a control break occurs, all operations conditioned by control level indicators are done before those that are not conditioned.

The L0 indicator is always on. You do not assign this indicator to record types or control fields, but you can still use it to condition operations. Programmers often use it when their input records have no control fields but they still want to condition calculations and total output.

The indicators L1 through L9 are set on when the content of control fields associated with them in the Input Specifications change from one record to another. In this case, all indicators of lower-level are turned on. Specific control level indicators are also turned on as a result of SETON operations, record type identification, or testing of non-control fields, but lower-level indicators are not turned on in this case. If you specify a control level indicator in the *Control Level Field* of the Calculations Specifications, the operation on the same line will be done only when that indicator is on. Thus, if you associate indicators L3, L2, and L1 with operations in your program and the program turns on L3 because of a control field change, L2 and L1 also are turned on, and all operations conditioned by these three indicators are done.

8-6. LAST RECORD INDICATOR (LR). Use this indicator to condition all operations to be done only at the end of your program. This indicator, as well as L1 through L9, are turned on after the last input record is processed. Then, the total calculations conditioned by L1 through L9 are performed, and the program ends.

You can also turn the LR indicator on by a SETON operation, but this does not affect L1 through L9. You cannot, however, use the SETOF operation to turn LR off.

8-7. SUBROUTINE IDENTIFIER (SR). Your program can call RPG/3000 subroutines that perform various detail and/or total calculations. This is done through the EXSR operation, discussed in paragraph 8-28. Write the subroutine code as the last group of entries on your Calculation Specifications Sheet. Enter SR in the *Control Level Field* of each line to indicate that it is part of the subroutine.

Entry of SR in the *Control Level Field* (columns 7 and 8) of a subroutine line is optional. Once the first BEGSR operation has been processed, the RPG compiler assumes that this subroutine line and all subsequent Calculation lines contain subroutine specifications.

8-8. AN/OR LINES. You can specify three indicators of any type for a calculation specification in the *Indicators Field* (Columns 9 through 17); these indicators might be general, halt, or overflow indicators, for instance. By using AN or OR lines, specifying AND and OR relationships respectively, you can specify even more indicators. Designate such lines by entering AN or OR in the Control Level Field, as discussed below. Then, in the *Indicators Field*, enter the appropriate indicators.

1. Write the first line of this specification, beginning with L0 through L9, LR, SR, or blanks in the *Control Level Field*, or with one to three indicators of any type in the *Indicators Field* in the normal way. Leave Columns 18 through 59 of this line blank.
2. Write AN or OR in the *Control Level Field* of the next line, and also enter any other indicators in the *Indicators Field*. This is your first AN or OR line; if it is not also your last, leave Columns 18 through 59 blank.
3. Continue entering any other AN or OR lines needed to specify the indicators you desire, up to a maximum total of seven such lines.
4. On your last AN or OR line, enter the calculation operation and data fields to which all indicators in the total relationship apply.

The indicators on each line are in an AND relationship. The type of entry (detail, total, last record, or sub-routine) for an AN or OR Line is the type specified by the first line of the relationship. For an AND relationship, the calculation is done if all indicator conditions are satisfied. For an OR relationship, the calculation is done if the conditions of any of the indicator sets separated by OR are satisfied.

You must always enter at least one indicator on an AN/OR line.

EXAMPLE

Examples of indicator entries appear in Figure 8-1. When the control level indicator L2 (**Item 2**) is on, L1 (Item 3) also turns on, and the last two calculations are performed. The first AN Line (Item 4) indicates that when 03, 04, 05, and 06 are ALL on, the contents of ETA are added to those of ZETA. The AN/OR line set (Item 5) specifies that when 01, 02, 03, and 04 are all on, or when 05 alone is on, the contents of OMEGA are subtracted from those ofOMICRON.

8-9. Indicators Field (Columns 9 through 17)

In addition to the indicators in the Control Level Field, you can enter other indicators in the *Indicators Field* to further specify when your program should perform a calculation. This field is not restricted to any particular type of indicator. On any one line, you can enter up to three indicators in this field; these appear in Columns 10 through 11, 13 through 14, and 16 through 17.

C Calculation Specifications

Columns 10 through 11, 13 through 14, 16 through 17 Entries	Meaning
01 through 99	General indicator used elsewhere in program.
F0 through F9 KA through KN, KP through KY	Function Key Indicator. Command Key Indicator.
L0 through L9	Control level indicators.
LR	Last record indicator.
MR	Matching record indicator.
H0 through H9	Halt indicators.
U1 through U8	External indicator.
OA through OG, OV	Overflow indicator.
1P	First-page indicator.
* (Column 11 only)	This calculation is conditioned by the same indicators as the previous calculation.
Blank	Operation is performed for every record read if Control Level Field does not contain L0 through L9, or SR.

You can specify that the indicator in the following subfield must be set off to condition the operation, by entering N in Columns 9, 12, and/or 15. (When you do this, you *must* enter an indicator in the following subfield.)

Columns 9, 12, 15 Entries	Meaning
N	The indicator must be OFF to condition the operation.
Blank	The indicator must be ON to condition the operation.

By using AN or OR lines, you can assign more than three indicators in these fields to condition a single calculation. You can enter up to seven AN or OR Lines in any combination.

All indicators on any one line are in an AND relationship with each other -- all must be on or all must be off to condition the operation. The indicators on one line, or in line groups, plus the Control Level Indicator in the *Control Level Field*, must be set as specified before the associated operation is done.

If you use the same indicators for several consecutive calculations, you need not repeat them in the *Indicators Field* each time. Instead, you can simply enter an asterisk (*) in Column 11 to designate that this calculation is conditioned the same as the previous one.

In general, you can use the indicators in this field in the following ways. (Before using any indicators other than general indicators, you should understand the RPG/3000 program cycle explained in Appendix D.)

1. Use any indicator previously specified in the *Record Indicator Field* (Columns 19 through 20) of the Input Specifications to condition an operation to be done only for a certain type of record.
2. Use any indicator previously specified in the *Field Indicator Field* (Columns 65 through 70) of the Input Specifications to condition an operation to be done only after the content of a field has met certain conditions.
3. Use the matching record (MR) indicator to condition an operation to be done only when matching records, specified in the *Matching/Chaining Field* (Columns 61 through 62) of the Input Specifications, occur.
4. Use any user indicators, including those specified in the *File Conditioner Field* (Columns 71 through 72) of the File Description Specifications, to condition operations.
5. Use any overflow indicators previously specified in the *Overflow Indicators Field* (Columns 33 through 34) of the *File Description Specifications* to condition operations to occur when overflow occurs.
6. Use any halt indicators previously specified in the *Field Indicator Field* (Columns 65 through 70) of the Input Specifications or the *Resulting Indicators Field* (Columns 54 through 59) of the Calculation Specifications to prevent the operation when a specified error condition occurs in the input data or in previous calculations, or to condition an operation to be done only when an error occurs.
7. Use any control level indicators specified in the *Control Level Field* (Columns 59 through 60) of the Input Specifications or in the *Resulting Indicators Field* (Columns 54 through 59) of the Calculation Specifications to perform an operation only on the *first* record of a new control group at detail time. When you do this, omit these indicators from the *Control Level Field* (Columns 7 through 8) of the Calculation Specifications.
8. Use any indicator specified in the *Resulting Indicators Field* (Columns 54 through 59) of the Calculations Specifications to condition operations on the basis of results of prior calculations.
9. Use the last record (LR) indicator to condition operations to be done at the end of your program. If this indicator is turned ON during calculations, use it by entering it in the *Indicators Field*. Otherwise, if it remains off until the last record is read, use it by entering it in the *Control Level Field* (Columns 7 through 8).

When you specify a control level indicator (L1 through L9) in the Control Level Field and MR in the *Indicators Field* in total calculations, MR indicates a matching condition for the previous record rather than the record that caused the control break. But, for detail calculations, MR indicates the matching condition of the record just read.

When you enter a control level indicator in the *Indicators Field* and not in the *Control Level Field*, the operation conditioned is done only on the record that causes a control break of this or higher level.

C Calculation Specifications

In the RPG/3000 program cycle, all operations conditioned by control level indicators in the *Control Level Field* take place before those conditioned by this type of indicator in the *Indicators Field*.

EXAMPLES

In Figure 8-2, the calculation occurs when L6 is on, if 01 and 03 are also on while 02 is off.

In Figure 8-3, the general indicator 01 is set on each time a record is read where the CLASS Field is blank. When this occurs, the ADD operation in the Calculation Specifications is not done.

8-10. Factor 1 and Factor 2 Fields (Columns 18 through 27 and 33 through 42).

Use these fields to define the data or fields you want to use in the calculation:

Columns 18 through 27 and 33 through 42 Entries	Meaning
The name of a field, table, array, array element or file.	The storage area containing the data to be used, or (in the case of a file name) the file to be used.
The name of a subroutine.	The subroutine to operate on the data.
A label.	The label for a TAG, ENDSR, or GOTO operation.
An alphanumeric, numeric or figurative literal.	The actual data to be used.
Blank	No operand is used.

The entries you use depend on the operation you select. Some calculations require entries in both the *Factor 1* and *Factor 2 Fields*, while others require an entry in only one of these fields or in none at all. The entries required and allowed are described in the discussion of each operation. Entries in *Factor 1* must begin in Column 18; those in *Factor 2* must begin in Column 33.

Field names used must be defined elsewhere in your program, unless they are the names of any of these special fields: UDATE, UMONTH, UDAY, UYEAR, PAGE, or PAGE1 through PAGE7, or *ERROR.

A *literal* is the actual data to be used in a calculation, rather than the name of a field containing that data. An *alphanumeric literal* can contain letters, digits, and/or special characters. A *numeric literal* can contain only digits, a decimal point in any position, and a leading arithmetic sign; embedded blanks are expressly forbidden.

When writing alphanumeric literals, follow these rules:

1. Use any combination of the 256 ASCII characters available, including blanks.
2. Enclose the characters comprising the literal within quotation marks. For instance, a literal comprised of the characters ALPHALIT would be entered as

"ALPHALIT"

3. Enter no more than eight characters, excluding the quotation marks.
4. To use quotation marks within a literal, represent these quotation marks by *two* quotation marks each. For example, to use a literal composed of the characters "NAME", enter:

""""NAME""""

5. Do not use alphanumeric literals in arithmetic operations.
6. For compatibility with other RPG systems, you can use apostrophes in place of quotation marks. But, when you do this, you must specify the QUOTE= parameter in the \$CONTROL compiler subsystem command (Section XII).

When using numeric literals, follow the rules below:

1. Use any combination of the digits 0 through 9, plus an optional decimal point and/or leading arithmetic sign. (An unsigned literal is treated as a positive number.) Blanks are not allowed in numeric literals.
2. Unlike alphanumeric literals, do not enclose the characters comprising a numeric literal in quotation marks. For example, the numeric literal 123.45 is written simply as:

123.45

3. Enter no more than ten characters, including sign and decimal point.

Use numeric literals in the same way you use numeric fields.

When using figurative literals (figurative constants), follow these rules:

1. Use *BLANK and *BLANKS with alphanumeric fields only.
2. Use *ZERO and *ZEROS with alphanumeric or numeric fields.
3. A figurative literal assumes the length of the other factor field. If the other factor field is blank, the result field length is assumed.
4. Figurative literals act like an elementary field when used in conjunction with an array.
5. Figurative literals can only be used in **Factor 1** or **Factor 2** of calculation specifications.

EXAMPLE

In the calculations shown in Figure 8-4, an alphanumeric literal (Item 1), a numeric literal (Item 2), and figurative literals (Items 3 and 4) appear. Other examples of the correct use of literals appear on the specification sheet.

C Calculation Specifications

HEWLETT PACKARD		RPG CALCULATION SPECIFICATIONS		Page 4 of 5											
Programmer SMITH, B.E.		Date 1/1/84		Punching Instructions											
Program Title FIN. REPT				Program Name FINREP											
Sequence Number	Form Type (C)	Control Level (L0, L1, L2, L3, SR, AN, OR)			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (0-9)	Half Adjust (H)	Resulting Indicators			Comments
		Not	And	And								Arithmetic	Plus	Minus	
1	C														
2	C				DATA X	MOVE	"AFRES"	SERV	10						
3	C					ADD	100.50	RESULT	02						
4	C														
5	C				10	MOVE	"2001"	TITLE	15						
6	C					MULT	ALPHA	PROD	02						
7	C				-1	DIV	ALPHA	QUOTA	02						
8	C														
9	C				*ZERO	MOVE	*BLANKS	SERV							
10	C					COMP	QUOTA								
11	C					MOVE	*ZEROS	RESULT							
12	C														
13	C					MOVE	*BLANK	TITLE							
14	C														
15	C					MOVE	*ZEROS	WORK	64						
16	C														
17	C														
18	C														
19	C														
20	C														
21	C														
22	C														
23	C														
24	C														
25	C														
26	C														
27	C														
28	C														
29	C														
30	C														
31	C														
32	C														
33	C														
34	C														
35	C														
36	C														
37	C														
38	C														
39	C														
40	C														
41	C														
42	C														
43	C														
44	C														
45	C														
46	C														
47	C														
48	C														
49	C														
50	C														
51	C														
52	C														
53	C														
54	C														
55	C														
56	C														
57	C														
58	C														
59	C														
60	C														
61	C														
62	C														
63	C														
64	C														
65	C														
66	C														
67	C														
68	C														
69	C														
70	C														
71	C														
72	C														
73	C														
74	C														

Figure 8-4. Literals

8-11. Operation (Columns 28 through 32) (Required)

In this field, specify the operation you want to perform. The possible operations are summarized below, and explained in detail in Paragraphs 8-21 through 8-41.

Columns 28 through 32 Entries	Meaning	Reference Page
ADD	Add <i>Factors 1</i> and <i>2</i> , and place the sum in the <i>Result Field</i> . When <i>Factor 1</i> is blank, adds <i>Factor 2</i> to the <i>Result Field</i> .	8-23
BITOF	Turn off the <i>result field</i> bits identified in <i>Factor 2</i> .	8-49
BITON	Turn on the <i>result field</i> bits identified in <i>Factor 2</i> .	8-49

Columns 28
through 32
Entries

	Meaning	Reference Page
BEGSR	Define the beginning of a subroutine.	8-38
CHAIN	Read a record from a chained field on disc during calculation operations.	8-67
CLOSE	Close any open file.	8-78
COMP	Compare <i>Factor 1</i> with <i>Factor 2</i> , and set resulting indicators on or off.	8-31
DEBUG	List indicators that are set on.	8-78
DIV	Divide <i>Factor 1</i> by <i>Factor 2</i> and place quotient in Result Field. When <i>Factor 1</i> is blank, divides Result Field by <i>Factor 2</i> and places quotient in the Result Field .	8-23
DSPLM	Print a User Message; also a field, table item, or array element and blank out or alter the field.	8-63
DSPLY	Print one or two fields, table items, literals, or array elements; also blank out or alter a field.	8-63
ENDSR	Define end of subroutine.	8-40
EXSR	Execute subroutine.	8-40
EXCPT	Write records while calculation operations are in progress.	8-56
EXIT	Transfer control from RPG program to user subroutine.	8-42
ERPGC	Indicate end of conversion routine.	8-82
EXTCV	Indicate conversion routine is external to RPG program.	8-82
FNDJW	Find a specific JCW in the JCW table.	8-87
FNUM	Obtain MPE file number for an open file.	8-88
FORCE	Identify the file from which the next record is to be selected.	8-57
GOTO	Branch to named instruction.	8-34
LOCK	Obtain exclusive access to a file.	8-74
LOKUP	Search for specific items in table or array.	8-50
MHHZO	Move zone from leftmost position of <i>Factor 2</i> to leftmost position of <i>result field</i> .	8-29
MHLZO	Move zone from leftmost position of <i>Factor 2</i> to rightmost position of <i>result field</i> .	8-30
MLLZO	Move zone from rightmost position of <i>Factor 2</i> to rightmost position of <i>result field</i> .	8-30
MLHZO	Move zone from rightmost position of <i>Factor 2</i> to leftmost position of <i>result field</i> .	8-31
MOVE	Move characters from <i>Factor 2</i> to rightmost position of <i>result field</i> .	8-26
MOVEA	Move characters from <i>Factor 2</i> array to leftmost position of <i>result field</i> .	8-28
MOVEL	Move characters from <i>Factor 2</i> to leftmost position of <i>result field</i> .	8-27
MSG	Move message specified in <i>Factor 1</i> into <i>Result Field</i> .	8-65
MULT	Multiply <i>Factor 1</i> by <i>Factor 2</i> , and place product in <i>Result Field</i> . When <i>Factor 1</i> is blank, multiplies <i>Result Field</i> by <i>Factor 2</i> and places product in <i>Result Field</i> .	8-23

C Calculation Specifications

Columns 28 through 32 Entries	Meaning	Reference Page
MVR	Move remainder from previous divide operation to <i>result field</i> .	8-24
PARM	Specify parameters for EXIT operation.	8-42
PUTJW	Put a JCW into the JCW table.	8-87
READ	Read input from demand file during current program cycle.	8-66
RLABL	Allow external subroutine to reference field internal to RPG program.	8-42
RPGCV	Indicate beginning of conversion routine.	8-80
SET	Label softkeys and enable F1 thru F8 for DSPLY and DSPLM .	8-57
SETLL	Set lower limit for a KSAM, RSAM, or IMAGE file processed sequentially between limits.	8-50
SETOF	Turn resulting indicators off.	8-48
SETON	Turn resulting indicators on.	8-48
SORTA	Sort array specified in <i>Factor 2</i> .	8-54
SQRT	Derive square root of <i>Factor 2</i> , and place it in <i>result field</i> .	8-24
SUB	Subtract <i>Factor 2</i> from <i>Factor 1</i> and place difference in <i>Result Field</i> . When <i>Factor 1</i> is blank, subtracts <i>Factor 2</i> from <i>Result Field</i> and places difference in <i>Result Field</i> .	8-23
TAG	Name instruction to which GOTO operation transfers.	8-36
TESTB	Test <i>result field</i> bits identified by <i>Factor 2</i> , and set resulting indicators on or off.	8-34
TESTN	Test alphanumeric field for presence of digits or blanks, and turn resulting indicator on or off.	8-32
TESTZ	Test zone of leftmost character in <i>result field</i> , and set resulting indicators on or off.	8-32
TIME	Return the system time, or system time and date.	8-84
TIME2	Return the formatted system date and time, and julian date.	8-85
UNLCK	Unlock a file.	8-74
XFOOT	Add array items and place sum in <i>result field</i> .	8-24
Z-ADD	Replace <i>result field</i> with <i>Factor 2</i> .	8-23
Z-SUB	Replace <i>result field</i> with negative of <i>Factor 2</i> .	8-23

The above operation codes are described in detail later in this section. When you enter one of these codes, begin the entry in Column 28. The operations will be performed in the order they appear on the *Calculation Specifications Sheet*, subject to the conditions specified by the indicators appearing in Columns 7 through 17.

EXAMPLE

In Figure 8-4, two MOVE operations, and one ADD, MULT, and DIV operation appear. Further examples appear throughout this section.

8-12. Result Field (Columns 43 through 48)

In this field, you determine where you want the program to store the result of the calculation.

Columns 43 through 48 Entries	Meaning
Name of field, table, array, or indexed array element.	Area where result is stored.
Blank	Result field does not apply to this opera- tion.

Enter the name of the field, table, array, or array element that will contain the result of the operation described in the previous fields. You can use the name of a field, table, array, or element defined elsewhere in your program. Alternatively, you can use a new field name. In this case, you must specify the length of the field in the *Field Length Field*, discussed next. For arithmetic or numeric compare operations, or for fields to be edited or zero-suppressed in Output Specifications, you must specify a numeric *Result Field*.

Begin the entry in the Result Field in Column 43.

EXAMPLE

In Figure 8-4, Result Fields with the following names appear: SERV, RESULT, TITLE, PROD, and QUOTA.

C Calculation Specifications

8-13. Field Length (Columns 49 through 51)

In this entry, define the length of the *Result Field*.

Columns 49 through 51	Meaning
1 through 256	Result field length.
Blank	This is an alphanumeric or numeric field, described elsewhere in the program.

Each field used in your program, including those used for the results of calculations, must be defined somewhere in your program. (A *field definition* specifies both field *name* and *length*.) If you enter in the *Result Field* the name of a field that is not already defined, you must specify its length in the *Field Length Field*. If you enter the name of a previously defined field, or of a table or array, you can omit the entry in the *Field Length Field* if you desire. If you do make the entry in this case, however, be certain that it specifies the same length defined elsewhere for the field.

This entry must end in Column 51. Leading zeros are permitted but not required. An alphanumeric field can range from one to 256 characters long; numeric fields can range in length from one to 15 digits.

Always be certain to make the result field long enough to hold the largest possible result obtainable from your calculation. If the field is too small, significant digits may be truncated from the result. If you specify rounding in the *Half-Adjust Field* (Column 53), the field length you enter in the *Field Length Field* applies to the result after rounding takes place.

EXAMPLE

In Figure 8-4, the result fields have lengths of 10, 8, 15, 10, and 10, respectively.

8-14. Decimal Positions (Column 52)

If you are defining a numeric field for the result, enter the number of positions to the right of the decimal point in this field.

Column 52 Entry	Meaning
0 through 9	Number of decimal positions in numeric result field.
Blank	This is an alphanumeric field, or a numeric field defined elsewhere.

If this is a numeric field defined elsewhere in your program, you can either enter the number of decimal positions in the *Decimal Positions Field* or you can leave this field blank. But, if you make an entry, be sure that it specifies the same number as that appearing in the field definition.

To indicate no decimal positions, enter 0. The number of positions must never exceed the length of the field. It may, however, be larger or smaller than the number of decimal positions resulting from the operation. If the number of decimal positions specified exceeds the number resulting from an operation, the

field is filled to the right with zeros in the unused portion. If the number of decimal positions is smaller than the number resulting from the operation, the rightmost digits are dropped from the result.

When it is placed in a *Result Field*, the numeric data is aligned on the decimal point with zero fill or truncation to the right or left as necessary to make the data fit in the field.

EXAMPLE

In Figure 8-5, we have specified that the field RESULT contains 2 decimal positions (Item 1).

8-15. Half Adjust (Column 53)

Use this field to indicate that numeric data in a result field is to be rounded (half-adjusted).

Column Entry	Meaning
H	Half adjust data.
Blank	Do not half adjust data.

Half adjusting is done by adding 5 (in the case of positive data) to the digit at the right of the last decimal position that fits in the result field, and then dropping this digit and all digits to the right of it. (In the case of negative data, -5 is added.)

You can only specify half adjusting for numeric fields. Furthermore, you cannot specify it for a MVR operation, or for a DIV operation followed by an MVR operation.

If the number of decimal positions allowed in the result field is greater than or equal to the number in the actual result, no half adjusting will be done.

EXAMPLES

In Figure 8-5, half adjusting is specified for the result to be stored in the field named RESULT (Item 2). This field is 5 digits long and contains two decimal places. Suppose that the result of the ADD calculation was 325.5769. First, the program would add 5 to the digit 6, and perform the normal carry operations, obtaining 325.5819. Then, it would truncate rightmost two digits, and store 325.58 in RESULT.

8-16. Resulting Indicators (Columns 54 through 59)

You can use this field for the following purposes:

1. To test the value in the result field after an arithmetic calculation or a LOKUP, COMP, TESTB, TESTZ, or TESTN operation.
2. To specify the indicators to be set on or off by a SETON or SETOF operation.
3. To indicate when the end of file is encountered while reading a demand file with the READ operation.
4. To indicate that a record was not found or to indicate the end-of-chain for a CHAIN operation.

The entries allowed are:

Columns 54 through 59 Entries	Meaning
01 through 99	General indicator.
F0 through F9 KA through KN, KP through KY	Function key indicator. Command key indicator.
H0 through H9	Halt indicator.
L0 through L9	Control level indicator.
LR	Last record indicator.
OA through OG, OV	Overflow indicator.
1P	First page indicator.
MR	Matching record indicator.
U1 through U8	User indicator.
Blank	No indicator is set.



You can enter these indicators in any of three subfields: *High* (Columns 54 through 55), *Low* (Columns 56 through 57), and *Equal* (Columns 58 through 59). When you enter any indicator other than a general indicator, be sure you understand how this indicator is set during the RPG/3000 Program Cycle discussed in Appendix D.

8-17. TESTING RESULTS OF OPERATIONS. After an operation is completed, you can test the contents of the result field if you have entered an indicator in the *Resulting Indicators Field*. The indicator you specify is turned off at the beginning of the operation, and turns on only if the result satisfies the condition specified by the subfield in which the indicator appears. If you use this indicator to test the result of more than one operation, the last operation performed determines the indicator setting. Normally, only general or halt indicators are used for result testing. (Because the *Resulting Indicators Field* contains three subfields, you can test for up to three conditions at the same time.) If you specify a general indicator, you can subsequently use it to condition calculation or output operations; if you specify a halt indicator, you can use it to halt your program or to suppress output operations when an unacceptable condition occurs.

Enter an indicator in the *High Subfield* to determine if:

1. The *Result Field* of an arithmetic calculation is positive (neither negative nor zero).
2. *Factor 1* is higher than *Factor 2* in a Compare operation.
3. A *Factor 2* table/array entry is found that is higher than *Factor 1* in a table or array look-up operation.
4. A record was not found during a CHAIN operation.
5. A TESTB operation reveals that the bits in a field were all set to zero.
6. A TESTZ operation reveals that the zone bits of a field are all positive, representing the ASCII characters & and A through I.

C Calculation Specifications

7. A TESTN operation reveals that an alphanumeric field contains all ASCII digits. (The low-order bits can still represent A through R.)

Enter an indicator in the *Low Subfield* to determine if:

1. The *Result Field* in an arithmetic operation is negative.
2. *Factor 1* is lower than *Factor 2* in a compare operation.
3. A *Factor 2* table/array entry is found that is lower than *Factor 1* in a table or array look-up operation.
4. A TESTB operation revealed that the bits in a field were mixed (both zeros and ones). (This evaluation is not meaningful if you test only one bit in the field.)
5. A TESTZ operation revealed that the zone bits of a field are all negative, representing the ASCII characters minus (-) and J through R.
6. A TESTN operation revealed that an alphanumeric field contains ASCII digits and leading blanks. (This field must be at least two characters long.)
7. The end-of-chain is reached for input types C and R on an IMAGE chaining file.

Enter an indicator in the *Equal Subfield* to determine if:

1. The *Result Field* in an arithmetic operation is zero.
2. *Factor 1* equals *Factor 2* in a compare operation.
3. A *Factor 2* table/array entry is found that equals *Factor 1* in a table or array look-up operation. (In these operations, only one indicator in the *Result Field* turns on; that in the *Equal Subfield* takes precedence over those in the other two subfields.)
4. A TESTB operation revealed that the bits in a field were all set to one.
5. A TESTZ operation revealed the zone bits in a field were neither all positive or negative, representing characters *other than* &, A through I, -, or J through R.
6. A TESTN operation revealed that an alphanumeric field contains all blanks.
7. The end-of-file is encountered while your program is reading a demand file with the READ operation.

EXAMPLE

The Calculation Specifications in Figure 8-6 contain a MULT calculation (Item 1) that turns the general indicator 20 on if the result is negative. The following ZSUB calculation (Item 2) is conditioned by indicator 20, and takes place only if that indicator is on. If the result of the third operation, SUB (Item 3), is positive or zero, and indicator 20 was on, it now turns off; if the result was positive or zero and indicator 20 was off, it remains off; if the result was negative and this indicator was on, it remains on; if the result was negative and the indicator was off, it turns on. In any case, at the beginning of the next program cycle, indicator 20 will be re-set to off automatically.

C Calculation Specifications

8-18. SETTING INDICATORS. In the *Resulting Indicators Field*, you can specify up to three indicators to be set on or off by a SETON or SETOF operation specified on the same line. These indicators can appear in any order in any of the three subfields.

8-19. Comments (Columns 60 through 74)

You can use the spaces in Columns 60 through 74 for comments of any kind.

8-20. Program Name (Columns 75 through 80)

This field contains the program name or any other information, as discussed in Paragraph 2-6.

8-21. OPERATIONS

The various RPG/3000 operations are discussed in detail below. Most of these operations are requested with codes that suggest or abbreviate the name of the operation, so they are easy to remember. Begin these codes in Column 28 of the Calculation Specifications Sheet. The operations themselves are grouped into the following categories.

- Arithmetic
- Move
- Compare/Test
- Branch/Exit
- Indicator/Bit Setting
- Table/Array Look-up
- File
- Debug
- Record Number Conversion
- System

8-22. Arithmetic Operations

Arithmetic operations can be requested only on *numeric* fields, literals, tables, arrays, or array elements. The *Result Field* entry must also be a *numeric* field, table, array, or array element--it cannot be a literal or alphanumeric field. When all three fields--*Factor 1*, *Factor 2*, and *Result*--are used, they may all refer to the same field or each to different fields, or any two may refer to the same field. If *Factor 1* or *Factor 2* is the name of a full array, the *Result Field* must also be the name of a full array (unless you are specifying the XFOOT operation, discussed below). If you specify more than one full array for an operation and the arrays contain different numbers of elements, the number of times that the operation is performed equals the number of elements in the smallest array. When a single field is named in one field and a full array in another, the field specified is applied to each element in the array.

The length of any field in an arithmetic operation cannot exceed 15 digits. --otherwise, the data will be truncated from either or both ends, depending upon the location of the decimal point. Automatic decimal alignment is always performed on the data in that field. *Factors 1* and *2* are not altered by any arithmetic operation, unless they are the same as the *Result Field* entry.

You can specify detection of arithmetic overflow during an operation, by leaving Column 65 of the Control Record Specification blank or entering 2, 3, 4, or 5 in this column; if overflow occurs in this case, a message is sent to the computer operator. If Column 65 contains a zero and overflow occurs, the overflow will be truncated in the result and will otherwise be ignored.

The arithmetic operations available are:

ADD

This operation adds **Factor 1** to **Factor 2**, and places the sum in the **Result Field**. If **Factor 1** is blank, this operation adds **Factor 2** to the **Result Field** placing the sum in the **Result Field**.

Z-ADD

The **Result Field** is replaced by **Factor 2**. **Factor 1** is not used. Left truncation occurs if the **Result Field** length is less than **Factor 2** length.

SUB

This operation subtracts **Factor 2** from **Factor 1**, and places the difference in the **Result Field**. If you subtract two fields of the same value, you effectively set the **Result Field** to zero. If **Factor 1** is blank, then **Factor 2** is subtracted from the **Result Field** and the difference is placed in the **Result Field**.

Z-SUB

The **Result Field** is replaced by the negative representation (complement) of **Factor 2**. Left truncation occurs if the **Result Field** length is less than **Factor 2** length.

MULT

This function multiplies **Factor 1** by **Factor 2**, and places the product in the **Result Field**. When you describe as **Factor 1** or **Factor 2** a field also described as the **Result Field**, be sure that the **Result Field** is large enough to hold the largest possible result. If **Factor 1** is blank, the **Result Field** is multiplied by **Factor 2** and the product is placed in the **Result Field**.

DIV

This divides **Factor 1** (dividend) by **Factor 2** (divisor), and places the quotient in the **Result Field**. If **Factor 1** is blank, the **Result Field** is divided by **Factor 2** and the quotient is placed in the **Result Field**. If **Factor 1** is zero, the result will be zero. If **Factor 2** is zero, an error will occur. Determine the action to be taken by entering the appropriate code in Column 66 of the Control Record Specification. If a remainder results from a **DIV** operation, it will be lost unless you move it into a **Result Field** by next specifying an **MVR** operation, discussed below. In this case, however, you should not request half-adjusting of the quotient resulting from the **DIV** operation because this also causes the remainder to be lost.

Results exceeding 15 digits are not valid. For each operation, RPG/3000 performs the following calculation as a check, and prints a warning message if a violation occurs:

$$F_1 + (D_2 - D_1 + D_r) \leq 15 \quad (14, \text{ if half-adjust is requested})$$

and

$$F_2 + (D_2 - D_1 + D_r) \leq 15$$

(In these calculations, $F_1 = \text{Factor 1}$, $F_2 = \text{Factor 2}$, and D_2 , D_1 , and D_r stand for the number of decimal positions in *Factor 1*, *Factor 2*, and the remainder, respectively.)

C Calculation Specifications

MVR

This function moves the remainder from the previous DIV operation to the *Result Field* specified in this MVR operation. Request it immediately following the DIV operation, and condition it with the same indicators. Do not use *Factors 1* and *2* nor specify half adjusting. The number of decimal positions in the result is the greater of:

1. The number of decimal positions in *Factor 1* of the DIV operation, or
2. The sum of the decimal positions in *Factor 2* and the *Result Field* of the DIV operation.

The maximum whole-number positions in the remainder is the same as the whole number positions in *Factor 2* of the DIV operation.

EXAMPLE

The DIV and MVR operations are demonstrated in Figure 8-7. Notice that both are conditioned by general indicator 26. The Quotient is stored in the field QUOTNT, and the remainder is saved in field REMAIN.

SQRT

With this operation, you can derive the square root of *Factor 2*, and place it in the *Result Field*. You can use whole arrays in this operation if you specify full array names in *Factor 2* and the *Result Field*. When you do this, the square root of each element in the *Factor 2* array will be placed in the corresponding element of the *Result Field* array.

You can request half adjusting in the SQRT operation. You should not, however, specify *Factor 1*, specify a *Result Field* length shorter than the *Decimal Positions Field* entry, or use resulting indicators. Furthermore, you must not use a negative number as *Factor 2*. (A negative number here results in an error message at the Computer Operator's console, unless you select another pre-programmed response in the Control Record Specification.)

XFOOT

The XFOOT operation adds all elements of a numeric array together, and places the sum into the *Result Field*. *Factor 2* specifies the array, which must always be numeric. The *Result Field* can name a field or an element of an array; if it names an element of the same array as *Factor 2*, the value of that element prior to the XFOOT operation is used in determining the total result. You can half-adjust the total in the *Result Field*; when you request this, rounding is done after all elements are added. You cannot specify *Factor 1* in this operation.

8-23. Move Operations

Move operations move part or all of *Factor 2* to the *Result Field*. *Factor 2* is not changed; *Factor 1* is not used, and should be left blank on the specifications sheet. Resulting indicators may not be used; half-adjusting is not allowed.

You can change a numeric field to an alphanumeric field by entering the name of the numeric field as *Factor 2* and the alphanumeric field as the *Result Field*. With this operation, packed numeric data will be unpacked in the *Result Field*.

C Calculation Specifications

You can change an alphanumeric field to a numeric field by entering the name of the alphanumeric field as **Factor 2** and the numeric field as the **Result Field**. In that case, unpacked data will be packed in the **Result Field**. The unpacked data must contain **numeric characters only except for the sign position which can contain a digit 0 through 9 or a letter A through R**.

When your program moves data into numeric fields, decimal positions are ignored; no decimal alignment is done. Thus, if your program moves data from a numeric field with two decimal positions containing 2.35 into a numeric field with one decimal position, the result will be 23.5.

MOVE

The Move operation moves characters from **Factor 2**, starting with the rightmost character, into the rightmost positions of the **Result Field**.

If **Factor 2** is longer than the **Result Field**, the program does not move the excess leftmost characters of **Factor 2**. If the **Result Field** is longer than **Factor 2**, the program does not change the characters to the left of the data moved in.

To change an alphanumeric field or constant to a numeric field, move it into a numeric field. In this operation, each alphanumeric character will correspond to a digit; the digit portion of each character is converted to the corresponding numeric character and moved into the **Result Field**. The zone portion of the rightmost alphanumeric character is converted to its corresponding sign and moved to the rightmost (sign) position of the numeric field. For the other characters, zone bits are stripped and the numeric bits are packed together. Blanks are transferred as zeros.

To change a numeric field to an alphanumeric field, move it to an alphanumeric field. For the rightmost character in **Factor 2**, the digit and zone will be transferred; for the other characters, the numeric zone bits (0011) will be added to the digit bits and the entire bit configuration will be transferred.

The effects of various MOVE operations are shown in Table 8-1.

Table 8-1. MOVE OPERATION RESULTS

Type of Move	Factor 2 Contents	Result Field Contents Before Move	Result Field Contents After Move
Alphanumeric to alphanumeric	ABC23	1234567	12ABC23
Alphanumeric to Numeric	1232C	1234567C *	1212323C *
Alphanumeric to alphanumeric	ABC23DEFG	1234567	C23DEFG
Alphanumeric to Numeric	12323456P	7654321C *	3234567D *
Alphanumeric to Alphanumeric	ABC23DE	1234567	ABC23DE

*Numeric result field shown in hexadecimal representation.

MOVE ZONE OPERATIONS

These operations allow you to move the zone bits from the high-order (leftmost) or low-order (rightmost) position of one field to the high-order or low-order position of another. These operations refer to Hollerith card zones rather than ASCII machine - representation zones. They apply specifically to the following characters:

Characters	Zone Rows Punched
A through I, +0, and &	12
J through R, -0, and -	11
S through Z	0
0 through 9	None
a through i	12-0
j through r	12-11
s through z	11-0
Any other characters	None

These bits are moved from Factor 2 into the Result Field. Factor 2 may name either a field or a literal; the Result Field must name a field. There are four move zone operations: MHHZO, MHLZO, MLLZO, and MLHZO. When any of these operations are used to move bits into an alphanumeric field, the operation may change both the zone and digit portions of a character in ASCII representation; for instance, moving the zone portion of an ASCII A (%101) to a field containing an ASCII K (%113) results in an ASCII B (%102). (In Hollerith representation, only the zone punches differ between a K and a B; the digit punches are the same.)

In the move zone operations, numeric signs are interpreted as follows:

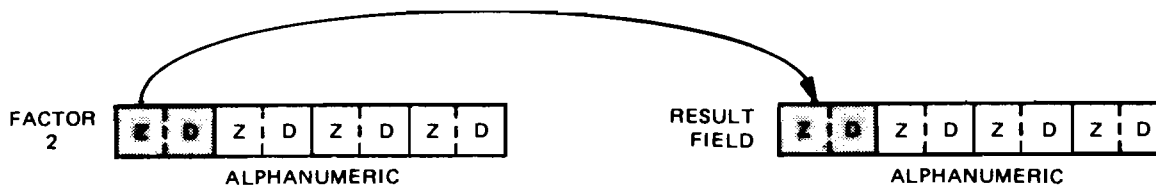
Bits	Sign	Zone Punches
1100	Plus (+)	12
1111	Plus (+)	None
1101	Minus (-)	11

Note: RPG/3000 does not recognize the literal “-0” as a valid negative quantity.

Attempts to change the sign of a field by use of this literal results in a positive sign; however, use of the literal “-” will result in a negative field.

MHHZO (Move High to High Zone)

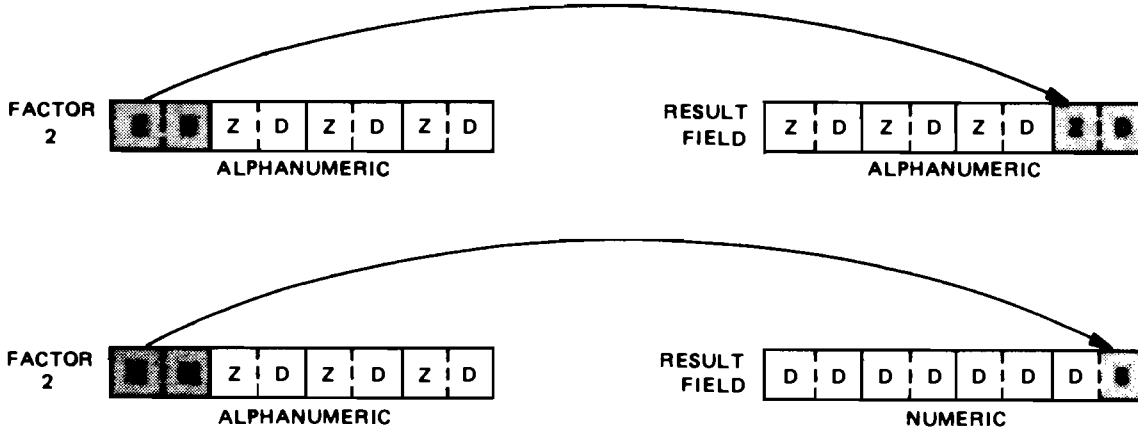
This operation moves the zone bits from the leftmost position of Factor 2 to the leftmost position of the Result Field, as noted above. Both fields must be alphanumeric. The move takes place as shown below, with Z indicating the zone portion of a character and D indicating the digit portion. The shaded area indicates the portion of the character affected.



C Calculation Specifications

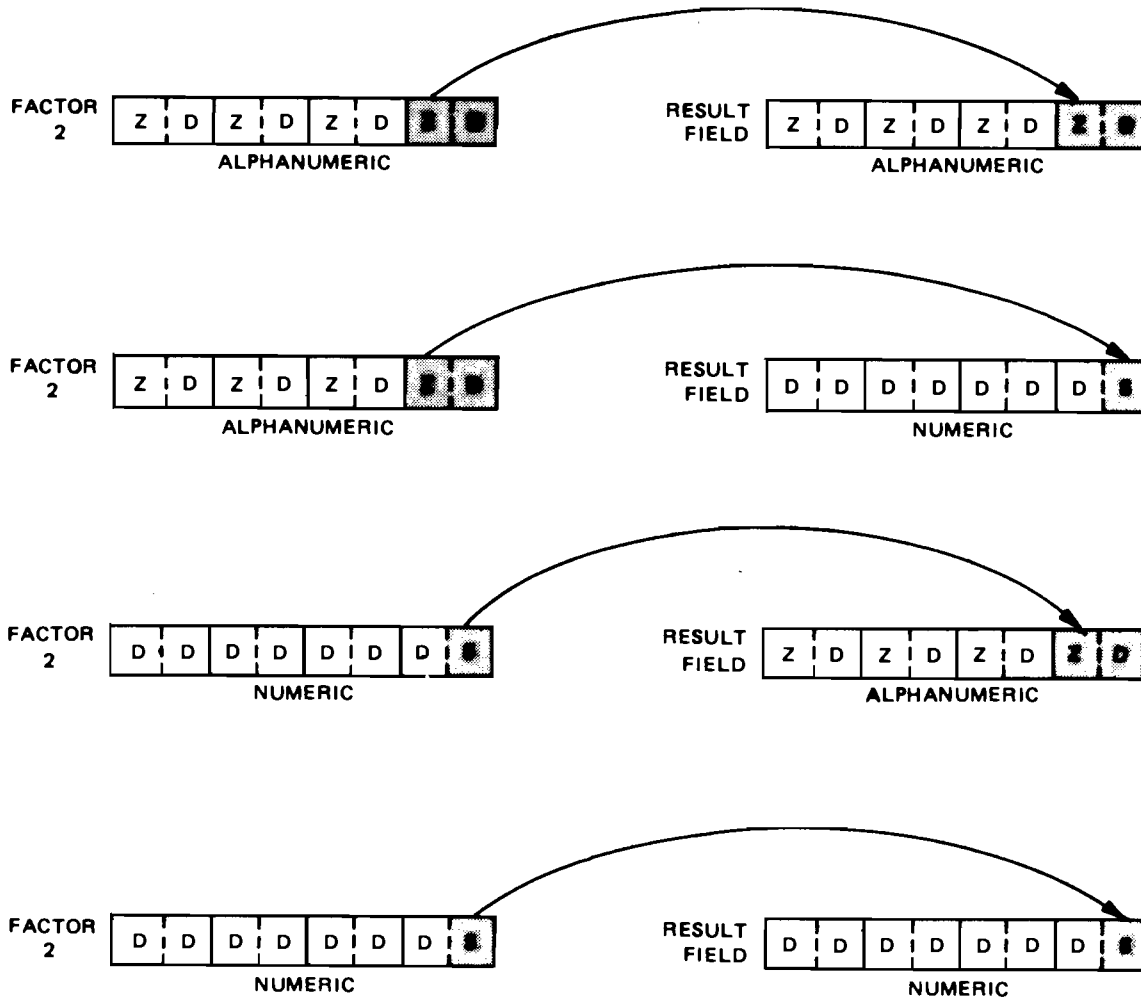
MHLZO (Move High to Low Zone)

This operation moves the sign or zone bits from the leftmost position of Factor 2 to the rightmost position of the Result Field, as noted above. Factor 2 must be alphanumeric, but the Result Field can be either alphanumeric or numeric. (In the illustration below, the letter S indicates the sign portion.)



MLLZO (Move Low to Low Zone)

This operation moves the sign or zone bits from the rightmost position of Factor 2 to the rightmost position of the Result Field, as noted above. Both fields may be either alphanumeric or numeric.



MLHZO (Move Low to High Zone)

This operation moves the sign or zone from the rightmost position of Factor 2 to the leftmost position of the Result Field, as noted above. Factor 2 can be alphanumeric or numeric, but the Result Field must be alphanumeric.



Results of various move zone operations are indicated in Table 8-2.

Table 8-2. MOVE ZONE OPERATIONS

Type of Move	Factor 2 Contents	Result Field Contents Before Move	Result Field Contents After Move Indicated			
			MLLZO	MHHZO	MLHZO	MHLZO
Alphanumeric to Alphanumeric	AH5SR	S51T	S51L	B51T	K51T	S51C
Alphanumeric to Numeric	AH5SR	12.3	12.3-	--	--	12.3+
Numeric to Numeric	852.4+	06.282-	06.282+	--	--	--
Numeric to Alphanumeric	4.7524-	␣KD	␣KM	--	-KD	--

Note: In this table, a dash (--) indicates that the move is not applicable; the symbol ␣ indicates a blank.

8-24. Compare and Test Operations

These operations test fields for certain conditions. The result of the test is shown by the status of the indicators you assign in the **Resulting Indicators Field** (Columns 54 through 59). While these operations tell you about the contents of fields, they do not actually alter those contents in any way.

COMP

This compare operation compares the contents of **Factor 1** with those of **Factor 2**. As a result of this operation, the following indicators are turned on:

Subfield	Columns	Condition
High	54-55	If Factor 1 is greater than Factor 2.
Low	56-57	If Factor 1 is less than Factor 2.
Equal	58-59	If Factor 1 equals Factor 2.

C Calculation Specifications

Factors 1 and 2 can both be either fields or literals.

Both fields are automatically aligned before they are compared. If they are alphanumeric fields, they are aligned on their leftmost characters; if one is shorter than the other, it is extended with blanks to equal size. If you defined an alternate collating sequence, alphanumeric fields are compared according to that sequence. If the fields compared are numeric, they are aligned on the implied decimal point; if one is shorter than the other, it is extended to equal size with zeros to the left or right of the decimal point. In numeric fields, a blank is equivalent to zero; maximum field length allowed is 15 digits.

EXAMPLE

Two compare operations are illustrated in Figure 8-8. In the first, if the contents of **FIELD1** are greater than the contents of **FIELD2**, the 01 indicator turns on; if they are less than those of **FIELD2**, the 02 indicator turns on; if they are equal to those of **FIELD2**, the 03 indicator turns on. In the second compare operation, the 05 indicator turns on if the field **STATUS** contains the letters **EXEMPT**.

Numeric-to-Alphanumeric Compare

You **may** compare a numeric field to an alphanumeric field. The comparison is done as if the numeric field were alphanumeric. That is, the numeric field is temporarily converted to alphanumeric and the rules for comparing two alphanumeric fields are applied.

If the numeric field contains decimal places, a physical decimal point is **not** present in the comparison. Negative signs are also ignored. For example, the numeric value **-123.45** is temporarily converted to the alphanumeric string **"12345"** for the compare operation. (You can avoid complications by using a numeric field that is **non-negative**, contains **zero decimal places**, and is the **same length** as the alphanumeric field to which it is being compared.)

To assist you in preventing inadvertent numeric-to-alphanumeric compares, the RPG compiler generates error 916W "NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE" for each such compare operation found in Calculation specs.

TESTZ

The Test Zone operation tests the zone of the leftmost character in alphanumeric **Result Fields**, and sets the indicators you specify in the **Resulting Indicators Field** to tell the results of the test. If the zone is a 12-zone (representing the characters &, +0, and A through I), the indicator in the **High Subfield** turns on. If the zone is an 11-zone (representing the characters -, -0, and J through R), the indicator in the **Low Subfield** turns on. Any other zone configurations cause the indicator in the **Equal Subfield** to turn on. Do not specify **Factors 1 and 2** in this operation.

TESTN

The Test Numeric operation determines whether an alphanumeric **Result Field** contains numeric characters and/or blanks. If all characters in the field are numeric, the indicator in the **High Subfield** turns on. (In this case, each character except the low-order character must contain an X'3' zone and a digit to be considered numeric; the low-order character is considered numeric if it contains A through R, or +0 (octal 173) or -0 (octal 175).) If the **Result Field** contains both numeric characters and leading blanks, the indicator in

the **Low Subfield** turns on. (Do not specify an indicator in this subfield when you are testing a field one character long, because the field must contain both a character and a leading blank for the test to be valid.) If the field contains blanks only, the indicator in the **Equal Subfield** turns on.

You can use the same indicator to test for more than one condition. Then, if any of these conditions exist, the indicator turns on.

To help prevent undesired results or abnormal program termination, the **TESTN** operation may be used to validate data before arithmetic operations or editing functions are performed upon it. Following this validation, you should move the data in the fields tested to numeric fields to perform the arithmetic or editing operations.

HEWLETT PACKARD

Programmer MAYFIELD Date 3/4/75

Program Title COMPARISON

RPG CALCULATION SPECIFICATIONS

Page 4 of 5

Punching Instructions

Graphic			
Punch			

Program Name 75 76 77 78 79 80
C O M P

Sequence Number	Form Type (C)	Control Level (L/O, L9, L/S, AN/DN)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (0-9)	Half Adjust (H)	Resulting Indicators			Comments
			Nor	And									Arithmetic			
				Not	Not								Plus	Minus	Zero	
1	C					FIELD1	COMP	FIELD2								
2	C					"EXEMPT"	COMP	STATUS								
3	C															
4	C															
5	C															

Figure 8-8. Compare Operations

8-33

C Calculation Specifications

TESTB

The Test Bit operation tests the bits in a one-character, alphanumeric field to determine if they are on or off. Factor 2 indicates which bits are to be tested. The Result Field specifies the field containing those bits. The results of the test are reported through the indicators you name in the Resulting Indicators Field.

For Factor 2, you can indicate up to eight specific bits, designating each by a single digit--0 for leftmost through 7 for rightmost; surround the group of digits by quotation marks. You can enter these bit designations in any order; bits not specified are not tested. As an example, to test the bits 0 and 2, enter "02". The test will indicate a positive result if the field contains these bit settings: 10100000. You can also specify the name of a one-character, alphanumeric field, a table, or array element as Factor 2. In this case, the bits that are on in Factor 2 are tested in the Result Field.

The Result Field must name a one-character alphanumeric field.

You must specify at least one resulting indicator with the TESTB operation. You can name as many as three for one operation; any two indicators can be the same for a single operation, but not any three. If you specify a field for Factor 2 that contains bits all set to zero, no resulting indicators are turned on. If each bit specified in Factor 2 is off in the Result Field, the indicator named in the High Subfield (Columns 54 through 55) is turned on. If two or more bits specified in Factor 2 are set to different status (some on, some off) in the Result Field, the indicator you name in the Low Subfield (Columns 56 through 57) is turned on. (You must be certain, in this case, that Factor 2 contains two or more bits that are on; otherwise, you must not name an indicator in the Low Subfield.) If all bits specified in Factor 2 are on in the Result Field, the indicator you name in the Equal Subfield (Columns 58 through 59) is turned on.

EXAMPLE

In the TESTB operation shown in Figure 8-9, Factor 2 is 0356 (specifying the bit-settings 10010110). This indicates that the Bits 0, 3, 5, and 6 in the Result Field are to be tested. If the Result Field contains the bit settings 01101001, indicator 01 turns on; if the Result Field contains bit settings 11110000, indicator 02 turns on; if the Result Field contains 10010110, indicator 03 turns on.

8-25. Branching and Exiting Operations

Calculations normally take place in the order you specify them on the Calculation Specifications Sheet. Occasionally, however, you may wish them to occur in a different order. For instance, you may want your program to bypass several calculations when various conditions arise, perform certain operations for selected record types only, or repeat a group of operations several times within one program cycle. You can request these steps through the branching and exiting operations discussed below.

8-26. BRANCHING OPERATIONS. RPG/3000 provides two branching operations; GOTO and TAG. The GOTO operation transfers control (branches) to another operation within your program, allowing your program to skip over calculations or to return to a previously-specified calculation. The TAG operation identifies the operation to which the GOTO operation branches.

GOTO

With the GOTO operation, you can branch to an earlier or later calculation specification in your program. Specify the name (label) that identifies that calculation in Factor 2 (Factor 1 and the Result Field are not used in the GOTO operation.) You can condition this operation with any indicators; if it is not conditioned, the GOTO operation will always be performed.

Programmer JAMES PETERS Date 3/4/75
 Program Title TESTER

Punching Instructions

Graphic			
Punch			

Program Name

T	E	S	T	R
---	---	---	---	---

Sequence Number	Form Type (CI)	Control Level (L, O, L9/ LFSR/ANVDI)			Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (0-9)	Half Adjust (H)	Resulting indicators			Comments	
		Not	And	And	Not	And	Not								Arithmetic	Plus Minus Zero	Compare		
1																			
2																			
3																			
4																			
5																			
6																			
7																			
8																			
9																			
10																			
11																			
12																			
13																			
14																			
15																			
16																			
17																			
18																			
19																			
20																			
21																			
22																			
23																			
24																			
25																			
26																			
27																			
28																			
29																			
30																			
31																			
32																			
33																			
34																			
35																			
36																			
37																			
38																			
39																			
40																			
41																			
42																			
43																			
44																			
45																			
46																			
47																			
48																			
49																			
50																			
51																			
52																			
53																			
54																			
55																			
56																			
57																			
58																			
59																			
60																			
61																			
62																			
63																			
64																			
65																			
66																			
67																			
68																			
69																			
70																			
71																			
72																			
73																			
74																			

Figure 8-9. Testing Bit Status

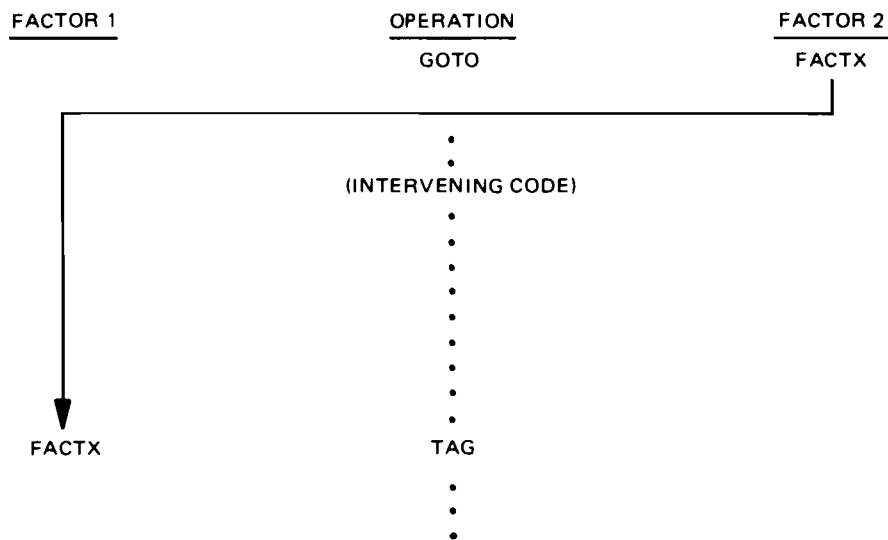
C Calculation Specifications

The name you enter in **Factor 2** must be the label of a **TAG** operation (that identifies the calculation to which you wish to branch within your program), or the label of an **ENDSR** operation (that identifies the last operation within a subroutine containing the **GOTO** specification). A **TAG** or **ENDSR** label may be the same as a field name in your program.

You can use a **GOTO** operation to branch within detail or total calculations, from detail to total calculations, or vice versa. But, when branching from detail to total calculations, you must be certain that you understand the **RPG/3000** program cycle to avoid possible mistakes.

TAG

You insert the **TAG** operation at the point in your Calculation Specifications to which you wish to branch. Enter the label as **Factor 1**; this must be the same label identified as **Factor 2** in the associated **GOTO** Statement. You cannot use the same label for more than one **TAG** operation, however, a **TAG** label may be the same as a field name which appears in your program. **Factor 2** and the **Result Field** are not used in the **TAG** operation. The branching operation affects your program in this way:



If a **TAG** operation is to occur at total time, you must specify (in the **Control Level Field**) a control level indicator. You cannot, however, specify indicators in the **Indicators Field** in any case for **TAG** operations.

EXAMPLES

An example of conditional branching appears in Figure 8-10.

If the **ADD** operation (Item 1) sets the 01 indicator on, the **GOTO CALCS** operation (Item 2) is performed and the program transfers to the **TAG** operation labeled **CALCS** (Item 3), executes the code following that statement, encounters the **GOTO START** statement (Item 4), and returns to the first **TAG** statement (Item 5).

If the **ADD** operation (Item 1) does not set the 01 indicator on, the program executes the **Multiplication** operation (Item 6) and then, because of the **GOTO** operation on the next line (Item 7), returns to the first **TAG** operation (Item 5).

8-27. EXITING OPERATIONS. Occasionally, you may write a program that must perform the same operations at several different points during its execution. Instead of specifying these operations at each point, however, you can save time and labor by writing them only once, in the form of a **subroutine**, and then transferring control (exiting) to that subroutine whenever the operations are required. This greatly simplifies your programming job. (Traditionally, any set of operations that can be repeated many times--

C Calculation Specifications

such as those in the RPG/3000 program cycle--is known as a routine: a group of operations within that program or routine that can be executed many times in one program cycle is thus called a subroutine.)

You can use two basic kinds of subroutines--internal and external. **Internal subroutines** are coded as part of your RPG/3000 program; they are useful for specifying sequences of operations required several times in that program. **External subroutines** are not coded as part of the RPG/3000 program, but as separate entities. They are frequently stored on mass-storage devices, and are used to specify sequences of operations used by many different programs.

8-28. Internal Subroutines. You code internal subroutines on the Calculation Specifications Sheet following all other calculations. These subroutines, of course, are written in the RPG/3000 language and are actually part of your program. You identify each subroutine by a label unique within your program. The label must not be the same as any other subroutine or tag label, however, it may also be used for a field name within your program without causing a conflict. You can then exit to (or call) that subroutine at any point in your program by referring to its label.

1. **The BEGSR operation**, which defines the name of the subroutine and the point at which it begins. (This is actually the first operation in the subroutine.)
2. **The subroutine body**, which consists of all calculations performed within the subroutine (except BEGSR and ENDSR).
3. **The ENDSR operation**, used to define the end of the subroutine.

The general structure of any internal subroutine is illustrated in Figure 8-11. When writing a subroutine, always describe it after the specifications for all detail and total calculations. Entry of SR in the **Control Level Field** (columns 7 and 8) of a subroutine line is optional. Once the first BEGSR operation has been processed, the RPG compiler assumes that subroutine line and all subsequent Calculation lines contain subroutine specifications.

BEGSR

Begin the subroutine with the BEGSR operation (Figure 8-11, Item 1) to define its entry point. In **Factor 1**, enter the name of the subroutine. (**Factor 2** and the **Result Field** are not used.) This name can range from one to six characters long, and must begin with a letter or the special character @, \$, or #. The remaining characters can be letters, digits, or @, \$, or # in any combination. Do not use embedded blanks. (You will also specify this same name as **Factor 2** of the EXSR operation that transfers control from your main program to this subroutine.) You must not reference the name used in **Factor 1** in any GOTO operation in or outside the subroutine. You can specify only one BEGSR operation in the subroutine.

Subroutine Body Operations

Specify the operations in the subroutine body (Figure 8-11, Item 2). You can request any operations but conversion routine functions--RPGCV, ERPGC, and EXTCV. Within a subroutine, you can condition any operations with entries in the **Indicators Field** (Columns 9 through 17). But, since SR must appear in the **Control Level Field** (Columns 7 through 8), you cannot use control level indicators in those columns to condition individual operations. (You can, however, condition the execution of an entire subroutine with a control level indicator by entering this in the **Control Level Field** or the **Indicator Field** of an EXSR operation.) If you use AN or OR relationships within a specification, you need enter SR only on the first line of that particular specification. Fields used in the subroutine can be defined inside or outside the subroutine; in either case, you can reference them in both the main program and the subroutine.

HEWLETT PACKARD RPG CALCULATION SPECIFICATIONS Page 2 of 3

Programmer: MAX SHORT Date: 3/2/75

Program Title: SUBS

Punching Instructions			
Graphic			
Punch			

Program Name:

5	8	5
---	---	---

Sequence Number	Form Type (C)	Control Level (L,O,LB/LN/SR/AN/ORN)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (D)	Half Adjust (H)	Resulting Indicators			Comments
			Not	And	And								Arithmetic	Plus Minus Zero	Compare	
1																
2																
3																
4																
5																
6	C															
7	C															
8	C															
9	C															
10	C															
11	C															
12	C															
13	C															
14	C															
15	C															
16	C															
17	C															
18																
19																
20																
21																
22																
23																
24																
25																
26																
27																
28																
29																
30																
31																
32																
33																
34																
35																
36																
37																
38																
39																
40																
41																
42																
43																
44																
45																
46																
47																
48																
49																
50																
51																
52																
53																
54																
55																
56																
57																
58																
59																
60																
61																
62																
63																
64																
65																
66																
67																
68																
69																
70																
71																
72																
73																
74																

(Main Program Code)

EXSR SUBNAM

SUBNAM BEGSR

(Subroutine Body)

ENDSR

Figure 8-11. General Subroutine Structure

C Calculation Specifications

Within a subroutine, you can use a GOTO operation during either detail or total calculations. You can use this operation to branch only to another operation in the subroutine--you cannot use it to return to the main program or to transfer to another subroutine.

ENDSR

End the subroutine with the ENDSR operation (Figure 8-11, Item 3). This operation terminates execution of the subroutine and returns control to the calling program; it transfers control to the operation following the operation (EXSR) that called the subroutine. In **Factor 1**, you can enter a label to which any GOTO operation within the subroutine can branch. (**Factor 2** and the **Result Field** are not used in the ENDSR operation.) You must not reference the name used as **Factor 1** in any GOTO operation outside the subroutine, or condition ENDSR by entries in the **Indicators Field**. You can specify only one ENDSR operation in the subroutine.

EXSR

Exit from the main program to the subroutine by specifying the EXSR operation (Figure 8-11, Item 4). You can enter this operation anywhere in your program, causing the subroutine to be executed each time. When the operations in the subroutine are done, control transfers to the operation on the line after the EXSR operation. Specify, in **Factor 2**, the name of the subroutine (defined in **Factor 1** of the BEGSR operation). (In the EXSR operation, **Factor 1** and the **Result Field** are not used.) You can condition subroutine execution by entering appropriate indicators in the **Control Level** and **Indicators Fields**.

If no indicators are used, the subroutine is always executed when this EXSR operation is encountered.

You can use as many subroutines in your program as you wish, and call them as many times as you wish. You need not define them in the same order in which you use them, but you must ensure that each one has a unique name, begins with a BEGSR operation, and ends with an ENDSR operation. Any subroutine can call another subroutine through the EXSR statement. But a subroutine should not call itself or the subroutine that called it. Furthermore, you cannot define a subroutine within a subroutine—more than one BEGSR/ENDSR operator pair within a subroutine is not allowed. You cannot use a GOTO operation outside a subroutine to branch to a statement within the subroutine.

You can use internal subroutines to produce repeatable operations common to more than one program by supplying the records containing these operations along with your main program deck. When you do this, be sure to use the correct subroutine name in the EXSR operation that executes the subroutine, and be certain that all fields common to the main program and the subroutine have the same name, length, and decimal positions.

EXAMPLE

Examples of subroutines and subroutine calls appear in Figure 8-12. Two subroutines, SUBA (Item 1) and SUBB (Item 2) are defined at the bottom of the Specifications Sheet. In the main program, two calls to SUBA appear (Items 3 and 4). When SUBA is executing, it also calls SUBB (Item 5), illustrating a nested subroutine call. SUBB contains a GOTO statement (Item 6) that branches to the end of that subroutine if indicator 01 is on.

HEWLETT PACKARD **RPG CALCULATION SPECIFICATIONS** Page 3 of 5

Programmer RICHARD I. MCKAY Date 5/5/75

Program Title EXTERN

Punching Instructions	
Graphic	
Punch	

75	76	77	78	79	80
Program Name <u>EXT</u>					

Sequence Number	Form Type (C) Control Level (L0 L9 L1/SR AN/OR)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (0-9)	Resulting Indicators	Comments
		Not	And	And								
1	C											
2	C											
3	C											
4	C											
5	C											
6	C											
7	C											
8	C											
9	C											
10	C											
11	C											
12	C											
13	C											
14	C											
15	C											
16	C											
17	C											
18	C											
19	C											
20	C											
21	C											
22	C											
23	C											
24	C											
25	C											
26	C											
27	C											
28	C											
29	C											
30	C											
31	C											
32	C											
33	C											
34	C											
35	C											
36	C											
37	C											
38	C											
39	C											
40	C											
41	C											
42	C											
43	C											
44	C											
45	C											
46	C											
47	C											
48	C											
49	C											
50	C											
51	C											
52	C											
53	C											
54	C											
55	C											
56	C											
57	C											
58	C											
59	C											
60	C											
61	C											
62	C											
63	C											
64	C											
65	C											
66	C											
67	C											
68	C											
69	C											
70	C											
71	C											
72	C											
73	C											
74	C											

Figure 8-12. Using Internal Subroutines

8-29. External Subroutines. You code these subroutines as separate entities, typically using other programming languages such as SPL/3000 or COBOL/3000. When they are compiled, prepared, and stored in the computer system, you can call them from any point in your RPG/3000 program. External subroutines offer an ideal way to supply repeatable sequences of operations used in many different RPG/3000 programs.

RPG/3000 provides three operations to permit interfacing with external subroutines: EXIT, RLABL, and PARM.

Note: To interface with COBOL subroutines, you must follow the special steps outlined under *COBOL Subroutines*, below.

- Non-COBOL Subroutines

For using the EXIT, RLABL, and PARM operations to communicate with subroutines written in languages other than COBOL, follow these directions:

C Calculation Specifications

EXIT

This operation branches from your main RPG/3000 program to the external subroutine desired. Specify the subroutine name in **Factor 2**; this name may contain a maximum of six characters. (**Factor 1** and the **Result Field** are not used.) When the subroutine completes execution, control transfers to the operation following **EXIT** in the main program.

RLABL

Every field, table, array, or indicator within your RPG/3000 program that is also used by the external subroutine must be made available to that subroutine. You can perform this function through **RLABL** operations.

You can enter these statements anywhere in your RPG/3000 program. Enter the name of the field, table, array, or numeric indicator (01 through 99 only) in the **Result Field**. (**Factors 1** and **2** are not used.) If you specify an indicator, specify it in the following format, where **xx** is the indicator:

INxx

For instance, to specify the indicator 01, enter

IN01

Regardless of how many external routines use the field, array, or indicator defined by an **RLABL** operation, you need define it only once in your program. When you define a name in **RLABL**, however, you cannot use it in any **GOTO**, **TAG**, **BEGSR**, or **ENDSR** operation.

In any **SPL** subroutine, all fields, tables, or arrays (passed by the **RLABL** operation to the routine) must be declared as external byte pointers. Indicators must be described as external integers. The example below shows how to pass these parameters in an **SPL** subroutine.

```
PROCEDURE exitname;  
  BEGIN  
  .  
  .  
  EXTERNAL BYTE POINTER rlabel;  
  .  
  .  
  EXTERNAL INTEGER INxx;  
  .  
  .
```

(**exitname** is the name of the external subroutine; **rlabel** is the field, table, or array to be passed.)

PARM

Traditionally in RPG, information is passed between an RPG program and an external subroutine through data areas identified by **RLABL** operations. However, information can also be passed through parameters which you define in a **PARM** operation immediately following the **EXIT** operation that invokes the subroutine. Write as many **PARM** operations as required--one for each parameter--just after the pertinent **EXIT** operation. Enter the parameter to be passed in the **Result Field**; you must specify only a field, table, or array name--indicators are not allowed. (**Factors 1** and **2** are not used in this operation.) The parameters are passed as byte-arrays in the order you specify them on the Calculations Specifications Sheet.

When you write an external subroutine in SPL/3000, use the following format:

```

PROCEDURE exitname (parm1,parm2, . . . parmn);
[ VALUE parm1,parm2, . . . parmn;
  BYTE POINTER parm1,parm2, . . . parmn; ] or [BYTE ARRAY parm1,parm2, . . . parmn;]
(Remainder of Procedure Head);

(Procedure Body);
.
.
.

```

(exitname is the name of the external subroutine; parm1, parm2, . . . parmn specify the parameters to be passed.)

Note: An external subroutine cannot declare "OWN" variables.

EXAMPLE

In Figure 8-13, two external subroutines, SUB1 and SUB2 (Items 1 and 2) are called. The RLABL operation (Item 3) makes general indicator 30 available to these subroutines. The PARM operation (Item 4) makes the information in the field PFIELD available to the subroutine SUB2 only.

• COBOL Subroutines

Because of differences in the way that RPG and COBOL handle parameter-passing, RPG programs cannot communicate directly with COBOL subprograms. Through the PARM operation, RPG passes parameters as byte pointers by value or as byte arrays by reference; this means that a byte-address is passed that references the DB-relative address of the RPG data-field, and that RPG can thus process data-fields aligned on *byte* boundaries. COBOL, however, can only receive parameters that are *word*-addresses of data located on *word* boundaries.

To resolve these differences and permit communication, you must perform the following two steps:

1. Ensure that the RPG data fields passed through the PARM operation to the COBOL subroutine reside on word boundaries. You can verify this by checking the RPG symbol table listing provided by the compiler. On this listing, data aligned on word boundaries is denoted by (*L*), for synchronized with the left byte of a word; data not aligned on word boundaries is denoted by (*R*), for synchronized with the right byte. Data aligned on the right byte should be changed to begin on word boundaries. To accomplish this change, you may alter the length of the preceding field or use the MOVE operation to transfer the data to a field of appropriate size.
2. Develop an SPL procedure that converts the byte addresses to word addresses and then calls the COBOL subprogram. In your RPG program, call this SPL procedure by using the EXIT operation.

Note: RPG numeric data values are processed as packed decimal data items. Your COBOL subprogram should specify them as COMP-3 data items.

EXAMPLE

Suppose you wished to call a COBOL subprogram from an RPG main program, passing to the subprogram three parameters. Two of these parameters will contain RPG data-field values. The third parameter, on return from the COBOL subprogram will contain the result of a calculation based on these two values. First, ensure that all RPG data fields are aligned on word boundaries. Then, write an SPL routine to convert byte-addresses to word addresses. The main program and symbol table listing appear in Figure 8-13A. The SPL procedure appears in Figure 8-13B. The COBOL subprogram appears in Figure 8-13C. The batch-job stream that compiles, prepares, and runs these programs appears in Figure 8-13D. The results of the program execution appear in Figure 8-13E.

0001	00010h						
0002	00720FTERMIN	IDE V	14				
0003	00730FOUTFLE	O F	39			DISC	
0004	10000ITERMIN	AA 01	1 CO				
0005	10010I					3 80FLD1	
0006	10020I					9 140FLD2	
0007	30000C			READ TERMIN			LR
0008	30010C	NLR		EXIT PRMCVT			
0009	30020C			PARM		FLD1	
0010	30030C			PARM		FLD2	
0011	30040C			PARM		RESULT 70	
0012	5000000OUTFLE	D	01				
0013	500100					5 "FLD1="	
0014	500200			FLD1		11	
0015	500300					17 "FLD2="	
0016	500400			FLD2		23	
0017	500500					31 "RESULT="	
0018	500600			RESULT		38	
0019	500700					39 "*"	

SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR
FLD1	6.0	00315(L)	FLD2	6.0	00317(L)	PRMCVT	SUBR	
						SYMBOL	SZ/TP	ADDR
						RESULT	7.0	00321(L)

Figure 8-13A. RPG Main Program and Symbol Table Listing

C Calculation Specifications

```
00000 0  $CONTROL SUBPROGRAM
00000 0  BEGIN  <<DB>>
00000 1  PROCEDURE PRMCVT(A,B,C);
00000 1  VALUE A,B,C;
00000 1  BYTE POINTER A,B,C;
00000 1  BEGIN  <<PRMCVT>>
00000 2  PROCEDURE LNKPRG;  <<COBOL SUBPROGRAM>>
00000 2  OPTION EXTERNAL;
00000 2  TOS:=@A&LSR(1);  <<CONVERT BYTE ADDRESSES>>
00002 2  TOS:=@B&LSR(1);  <<TO WORD ADDRESSES AND>>
00004 2  TOS:=@C&LSR(1);  <<LEAVE THEM ON THE STACK>>
00006 2  LNKPRG;  <<CALL THE COBOL SUBPROGRAM>>
00007 2  END;
00000 1  END.
```

Figure 8-13B. SPL Procedure for Address Conversion

```
001000$CONTROL SUBPROGRAM,USLINIT
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID. LNKPRG.
001300 ENVIRONMENT DIVISION.
001400 DATA DIVISION.
001500 LINKAGE SECTION.
001600*
001700*NUMERIC PARAMETER DATA TYPED AS COMP-3
001800*
001900 01 FLD-1 PIC S9(6) COMP-3.
002000 01 FLD-2 PIC S9(6) COMP-3.
002100 01 RESULT PIC S9(7) COMP-3.
002200 PROCEDURE DIVISION USING FLD-1 FLD-2 RESULT.
002300 START-LINK.
002400 ADD FLD-1 FLD-2 GIVING RESULT.
002500 DISPLAY "FLD1= " FLD-1 " FLD2= " FLD-2 " RESULT= " RESULT.
002600 GOBACK.
002700
```

Figure 8-13C. COBOL Subprogram Called by RPG Main Program

```
1      :JOB RPGCOB,SE.PAVONE,RPGWIZ
2      :CUBOL CUBOLS
3      :SPL SPLS
4      :PPG RPGS
5      :PREP SOLDPASS,PGM
6      :FILE TERMIN=$STDIO
7      :FILE OUTFLF=$STDLIST
8      :RUN PGM
9      0000075000010
10     :EQJ
```

Figure 8-13D.. MPE Batch-Job Stream for Compiling, Preparing, and Executing Programs

```
FLD1= 000075  FLD2= 000010  RESULT= 000008E
FLD1=000075  FLD2=000010  RESULT=0000085*
```

Figure 8-13E. Result of Program Execution

8-30. Indicator and Bit Setting Operations

With indicator setting operations, you can turn any indicators on or off. You can specify up to three indicators with each operation specification; simply enter them in the **Resulting Indicators Field**. (The subfield headings **Low**, **High**, and **Equal** are not relevant in this case.)

Whenever your program reads a new record, any record-identifying indicator associated with it is set to reflect conditions on that record. At this point, the setting from any previous indicator-setting operation does not apply.

Whenever detail operations are completed for a record, the control level indicators (L1 through L9) and the record identifying indicators are turned off regardless of any previous indicator-setting operation.

SETON

This operation turns on any indicators appearing in the **Resulting Indicators Field**. Factors 1 and 2 are not used.

SETOF

This operation turns off any indicators appearing in the **Resulting Indicators Field**. Factors 1 and 2 are not used.

With bit setting operations, you can set individual bits in a one-character, alphanumeric field. You can then use these bits as software switches in your program. Factor 2 indicates which bits are set. The **Result Field** specifies the field containing those bits. The **Resulting Indicators Field** is not used for these operations.

For Factor 2, you can specify up to eight bits, designating each by a single digit--0 for leftmost through 7 for rightmost; surround the group of digits by quotation marks. You can enter the bits in any order; bits not specified remain unchanged in the **Result Field**. As an example, to set the bits 0, 4, 5, 6, and 7 of the **Result Field**, enter "04567" as Factor 2. You can also specify the name of a one-character, alphanumeric

EXAMPLES

In Figure 8-14, the SETON operation turns on the indicators 03 and L2. The BITON operation turns on the Bits 3 and 4 (the fourth and fifth bits) in the field BITFLD. The BITOF operation turns off, in the field BITFLD, those bits that are ON in the field MYFLD.

8-31. Lower-Limit Operation

For a KSAM, RSAM, or IMAGE Demand file that is processed sequentially between limits, you can establish or alter the lower limit by requesting the SETLL operation. **Factor 1** is a literal or the name of a field denoting the lower limit to be set. However, the length of the key field specified for **Factor 1** must be the same as the key field length defined for the file. For KSAM files, the key field specified in **Factor 1** can be shorter in length than the key field length defined for the file (only if a non-blank entry is specified in columns 43 through 45 and if the key type is not “packed”). This is a “partial” key. If a partial key is used, RPG positions the file at the first record whose key (leftmost characters) matches the partial key value from **Factor 1**. The match is done according to the relational operator specified in the **Result field**. Note that this relational matching is also done if a full key, rather than a partial key, is specified in **Factor 1**.

Factor 2 is the name of the file for which the lower limit is to be set; this must be a KSAM, RSAM, or IMAGE Demand file. One of the following relational operators can be specified in columns 43 through 45 of the **Result field** to define how keys are to be matched in setting the lower limit:

Operator	Description
*EQ	to match on Equal key
*GT	to match on Greater-Than key
blank or *GE	to match on Greater-than-or-Equal key

The relational operator specifies how the key value in the file must match the value of the **Factor 1** key (which can be either a “full” or “partial” key, as described above). The record to which the file is positioned will have this relation to the **Factor 1** value.

An indicator can be specified in the **High Subfield** (columns 54 through 55). The indicator is turned on if SETLL fails to locate a record in accordance with the relational operator specified in the **Result field**. This also sets the file’s internal EOF flag, so that a subsequent READ turns on its **Equal Subfield** indicator (columns 58 through 59). No default indicator is set if no indicator is specified in columns 54 through 55 of the SETLL line.

The conditioning and control-level indicators are optional for this operation.

EXAMPLE

In Figure 8-14, the SETLL operation sets the lower limit for the file named IFILE to the value specified in the field named LOWLIM.

8-32. Table/Array Look-Up Operation

You can request your program to search a single table, a pair of related tables, or an array for a specific entry or element by specifying the LOKUP operation. For table searches, this operation retrieves the entry from the table and makes it available for use in subsequent calculations; for array searches, the operation determines if the array contains an element that matches the one you specify.

In **Factor 1**, you identify the data entry you wish to find; you can specify an alphanumeric or numeric constant, a field name, an array element, or a table name. When you enter a table name as **Factor 1**, this name

refers to the last element selected from this table during a previous LOKUP operation--not to the entire table. Factor 1 must specify data in the same format as the entry sought.

Factor 2 is the name of the table or array to be searched. During the search, the program seeks a match for the Factor 1 entry in the Factor 2 table or array.

With the LOKUP operation, you must always specify at least one indicator in the Resulting Indicators Field. These indicators serve two purposes: first, they specify the type of search you desire, and second, they show the result of that search when it is completed. Specify an indicator in the Equal Subfield to request the program to search for an entry in the table or array that equals the entry in Factor 1; if this entry is located, the indicator turns on. If several entries identical to Factor 1 are found, the first one encountered is selected. Specify an indicator in the Low Subfield to request the program to seek an entry that is nearest to, but lower in value than, Factor 1; when this entry is found, this indicator turns on. Specify an indicator in the High Subfield to request the program to find an entry that is nearest to, but higher in value than, Factor 1; when this entry is found, this indicator turns on.

Although you must specify at least one resulting indicator, you cannot specify more than two. You can use them in combination to test for Equal and High or Equal and Low. In such cases, the program searches for an entry satisfying either condition (but giving precedence to Equal); in other words, if it does not locate an entry equal to Factor 1, then it selects the next lowest or highest entry. The condition satisfied determines which indicator is set on.

HEWLETT PACKARD		RPG CALCULATION SPECIFICATIONS																				Page 2 of 3			
Programmer <u>R. D. J.</u>		Date <u>3/2/75</u>																				Punching Instructions		Program Name <u>BITS</u>	
Program Title <u>BITSET</u>																						Graphic		75 76 77 78 79 80	
																						Punch			

Sequence Number	Form Type (C)	Control Level (LD, LB, LR, SR, AN, OR)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (0-9)	Half Adjust (H)	Resulting Indicators			Comments
			Not	And	And								Arithmetic	Plus Minus Zero	Compare	
			Not	Not	Not				Look up Factor 2 is							
									High	Low	Equal					
1	C						SET ON						3	L	2	
2	C						BIT ON "34"	BIT FLD								
3	C						BIT OF MY FLD	BIT FLD								
4	C					LOW LIM	SET LL I FILE									
5	C															
6	C															
7	C															
8	C															
9	C															
10	C															
11	C															
12	C															
13	C															
14	C															
15	C															
16	C															
17	C															
18	C															
19	C															
20	C															
21	C															
22	C															
23	C															
24	C															
25	C															
26	C															
27	C															
28	C															
29	C															
30	C															
31	C															
32	C															
33	C															
34	C															
35	C															
36	C															
37	C															
38	C															
39	C															
40	C															
41	C															
42	C															
43	C															
44	C															
45	C															
46	C															
47	C															
48	C															
49	C															
50	C															
51	C															
52	C															
53	C															
54	C															
55	C															
56	C															
57	C															
58	C															
59	C															
60	C															
61	C															
62	C															
63	C															
64	C															
65	C															
66	C															
67	C															
68	C															
69	C															
70	C															
71	C															
72	C															
73	C															
74	C															

Figure 8-14. Indicator and Bit Setting Operations

C Calculation Specifications

When entering a LOKUP operation, be certain that the entry specified as **Factor 1** and the entry sought in the table or array specified as **Factor 2** are the same length; (they need not, however, have the same decimal alignment; no decimal alignment is done during the operation). You can request searching for elements that are High, Low, High and Equal, or Low and Equal only if the table or array searched is in ascending or descending sequence. The SORTA operation can be used to ensure that an array is in sequence. You cannot request a search for a High and Low element in the same LOKUP operation. If the entry sought is not found, no indicator turns on.

8-33. SEARCHING A SINGLE TABLE. When requesting a search of a single table, specify **Factor 1**, **Factor 2**, and at least one resulting indicator in the LOKUP operation. You can also condition this operation with indicators in the **Control Level** and **Indicators Fields**.

When your program locates the entry sought in the table, that entry is the one referred to by subsequent uses of the table name. But, if the program fails to find the entry sought, subsequent uses of the table name refer to the previously-found entry. If an indicator named in the **Resulting Indicators Field** turns on at the completion of the operation, then you will know that the search was successful and that the entry sought is available. To reference this entry in a subsequent calculation operation, simply use the table name in that operation. This same entry will be available for as many operations as you like, until it is changed by another LOKUP operation. If you use the table name in the **Result Field** of an operation other than LOKUP, you change the contents of the entry last found in the table.

EXAMPLE

In Figure 8-15, suppose that the field ENTRY1 in the LOKUP operation (Item 1) contains the value 300. The LOKUP operation searches the table TABLEA for this value, and if it is found, turns on the resulting indicator 10 and makes 300 available for use when the table is next referenced. When the indicator 10 is on, the ADD operation (Item 2) retrieves the value 300, adds 100 to it, and places the sum in TABLEA over-writing the previous value (300) in the table with the value 400.

8-34. SEARCHING RELATED TABLES. In a search of related tables, only one table is actually searched. Specify the name of this table as **Factor 2** and the name of the related table in the **Result Field**. Use **Factor 1** and the resulting indicators just as you would in a LOKUP operation for a single table. When the search is conducted, the table denoted by **Factor 2** will be scanned for the entry specified by **Factor 1**. If this entry is found, the appropriate resulting indicator will be turned on; both this entry and its corresponding entry in the **Result Field** table will be referenced when the **Factor 2** table name is referenced. Both tables should be the same length; otherwise, if the table searched is longer than its related table, the search will stop at the end of the shorter table.

You can reference the entry retrieved from the **Result Field** table alone by using the name of that table in calculation operations.

If you have named an array element as **Factor 1**, you cannot request a search of related tables in this LOKUP operation--you must leave the **Result Field** blank.

EXAMPLE

In the second LOKUP operation in Figure 8-15, the program searches TABLEB (Item 3) for the content of the field ENTRY2. If this search is successful, this value and its corresponding entry in TABLEC will be made available and the indicator 20 will be turned on. In this case, the following operation (Item 4) multiplies the entry retrieved from TABLEC by 20, and stores the product in the field STORA.

8-35. SEARCHING AN ARRAY. In searching an array, specify **Factors 1** and **2**, and the resulting indicators as you would in searching a table; do not, however, make an entry in the **Result Field**. If your program

C Calculation Specifications

finds the element sought, it will turn on the appropriate indicator. However, arrays differ from tables because the last element located in the array cannot be referenced by using the array name alone; it can, however, be referenced as noted below.

If you use the array name alone as **Factor 2**, the search begins with the first element in the array and the specified indicator turns on if a match with **Factor 1** is found. But, if you use the array name with an index as **Factor 2**, the search begins with the element specified by the index. In this case, if a field was used as the index and the element was found, the number of the matching array element is placed in the index field; if the element was not found, the index field is set to 1. If you used a literal as an index, you can only determine whether a search was successful by using an indicator; the content of the literal does not change in this case.

EXAMPLE

The third LOKUP operation in Figure 8-15 (Item 5), searches the array ARR1, beginning with the element specified by the index INDX, for an element whose value matches the content of the field ENTRY3. (Assume the index field INDX contains the value 50 at this point.) If a match is found, indicator 30 turns on and the number of the element containing the match is placed in INDX. (If this element was the seventy-third element in ARR1, INDX would now contain the value 73.) In this way, the matching element can still be used in a later calculation such as the MOVE operation (Item 6) which places this value in the field STORB.

The **SORTA** operation enables you to sort array elements into ascending or descending sequence.

To use this operation, specify **SORTA** in the **Operation Field**, specify the array to be sorted in the **Factor 2 Field**, enter any conditioning indicators in the **Control Level** and **Indicators Fields**, and leave all other columns blank.

If the **Sequence Field** (column 45) of the extension specification on which the array has been declared is blank or contains an "A", the array is sorted in ascending order. If the **Sequence Field** contains a "D", the array is sorted in descending order. See Figure 8-15a.

The standard ASCII collating sequence (Appendix-G) is used for the **SORTA** operation. Alternate collating sequence is not supported by the **SORTA** operation.

EXAMPLE:

In the sample program (Figure 8-15b), an array ARSEQ which has been read in from a disc file is moved to ARSEQA and sorted into ascending sequence with the **SORTA** operation (Item 1). It is then moved to ARSEQD and put into descending sequence with the **SORTA** operation (Item 2).

ARSEQ	
1	791006
2	470711
3	530820
4	640218
5	830512
6	840315

ARSEQA	
1	470711
2	530820
3	640218
4	791006
5	830512
6	840315

ARSEQD	
1	840315
2	830512
3	791006
4	640218
5	530820
6	470711

Figure 8-15a. Contents of Arrays After SORTA

HEWLETT PACKARD

RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS

Page ____ of ____

Programmer _____ Date _____
 Program Title SORT ARRAY

Punching Instructions			
Graphic			
Punch			

Program Name									
--------------	--	--	--	--	--	--	--	--	--

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Seq.	Chaining File Code (C, CN)	From Filename	To Filename	Table Array or Routine Name	Entry Per Record (1-999)	Entry Per Table Array (1-9999)	Entry Length (1-756)	Data Format (P, B, L, R)	Decimal Positions (0-9)	Table Array Sequence (A, D)	Table Array Name	Entry Length (1-756)	Data Format (P, B, L, R)	Decimal Positions (0-9)	Table Array Sequence (A, D)	Comments	
1						ARSEQ			6										
2						ARSEQA			6										
3						ARSEQD			6										



HEWLETT PACKARD

RPG INPUT SPECIFICATIONS

Page ____ of ____

Programmer _____ Date _____
 Program Title SORT ARRAY

Punching Instructions			
Graphic			
Punch			

Program Name									
--------------	--	--	--	--	--	--	--	--	--

Sequence Number	Form Type (I)	File Name	Group Sequence	Number of Records (LN)	Option (O)	Record Identification Codes			Stacke Sheet (I, IZ)	Field Position		Field Name	Control Level (L, LR)	Matching or Chaining Fields (M, LR, S, T, CN)	Field Response Relation			Field Indicators		
						1	2	3		From (1-9999)	To (1-9999)				Plus	Minus	Zero or Blank			
1		IANYFILENS									36	ARSEQ								

HEWLETT PACKARD

RPG CALCULATION SPECIFICATIONS

Page ____ of ____

Programmer _____ Date _____
 Program Title SORT ARRAY

Punching Instructions			
Graphic			
Punch			

Program Name									
--------------	--	--	--	--	--	--	--	--	--

Sequence Number	Form Type (C)	Control Level (L, LR, LRSR AND, OR)	Indicators		Factor 1	Operation	Factor 2	Result	Result Name	Result Length (1-756)	Decimal Positions (0-9)	Full Squared	Resulting Indicators			Comments			
			And	Or									Arithmetic	Plus Minus Zero	Compare				
1					MOVE	ARSEQ		ARSEQA											
2					SORTA	ARSEQA		ARSEQA											
3					MOVE	ARSEQ		ARSEQD											
4					SORTA	ARSEQD		ARSEQD											

Figure 8-15b. Using the SORTA Operation

C Calculation Specifications

8-36. File Operations

During the normal RPG/3000 program cycle, processing takes place in the following order: the program reads an input record, performs calculations upon the data just read, and writes one or more output records based upon the results of these calculations. You can, however, alter this sequence of events so that input, output or both can take place during calculation operations--that is, these functions can be interleaved with calculations. The operations that permit you to do this are: EXCPT, FORCE, DSPLY, READ, and CHAIN, all described below.

EXCPT

Normally, your RPG/3000 program outputs only the exact number of records specified in the Output Specifications during each program cycle. But, if you wish to vary the number of records output at either detail or total time during a cycle, you can use the EXCPT (Exception) operation. You could use this operation, for example, to write out several copies of a table, produce many records having the same information, or print a group of blank forms all having the same headings.

To use this operation, specify EXCPT in the **Operation Field**, enter any conditioning indicators desired in the **Control Level** and **Indicators Fields**, and leave all other columns blank. Then, in the **Type Field** of the Output Specifications, enter the letter E on each line to be written out during calculation time. Because these lines constitute an exception to the normal program cycle, they are called **exception lines**. When the EXCPT operation is performed, all exception lines whose conditioning indicators are satisfied are written out. Then, the program continues with the next calculation specification following the EXCPT operation.

You cannot specify exception lines for combined files. You cannot use an OF indicator to condition an exception line, although you can use it to condition a field on that line.

EXAMPLE

In Figure 8-16, a program reads (among other records) records containing the mailing addresses of various company customers (Item 1). When the customer's name (as specified in the NAME field) matches a name specified in the NAMEA field, indicator 50 is set on (Item 2) and the EXCPT operation (Item 3) causes a mailing address label identified by an exception line (Item 4) to be printed. The following calculations cause the label to be printed five times (Item 5) before the program continues (Item 6).

FORCE

The FORCE operation permits you to select the file from which your program takes the next record for processing. When this statement is executed, the record is actually selected at the beginning of the next program cycle. If more than one FORCE operation is specified during a single cycle, only the last is actually performed. The FORCE operation remains in effect only during the next input cycle.

With the FORCE operation, you can override the multifile processing sequence normally in effect for RPG/3000 programs. The first record read by your program, however, is always selected by the normal method--it cannot be forced; the remaining records can all be forced if you desire.

To request forcing, enter the FORCE operation in the **Operation Field** and identify the file containing the next record to be processed in **Factor 2**. Do not use **Factor 1** or the **Result Field**.

You can specify a FORCE operation at detail time in either a main program or subroutine; you cannot, however, specify it at total time. When you force a file retrieved by means of a record address file (RAF), the associated record from the RAF can also be read at this time. When a forced record is being processed, the MR indicator turns off so that the forced record is treated as a record with no matching fields. When your program encounters the end-of-file indicator on a forced file, no record will be retrieved from that file and the next record processed will be determined by the normal selection process.

EXAMPLE

Suppose that you have a program that processes records from three files--AFILE, BFILE, and CFILE, in that order--during the normal program cycle. But, suppose also that you want the next record read to be selected from CFILE if the indicator 09 is on. You could specify this as shown in Figure 8-17.

SET

The SET operation enables your program to use the function keys (f1 thru f8) of your terminal to set **Function Key Indicators** (F1 thru F8) with the DSPLY or DSPLM operation. The SET operation will also allow you to label the softkeys of Hewlett-Packard terminals and workstations which support the softkey labeling feature.

To specify the SET operation, enter SET in the **Operation Field**, then specify **Factor 1** or the **Resulting Indicators**, or both, for the functions below.

1. To label the softkeys, enter an array name as **Factor 1**. This array must be specified in your extension specifications as having eight entries with an entry length of sixteen characters each.

Each entry of this array corresponds to a softkey label. The first eight characters of an entry will appear as the top half of the key label and the following eight characters will appear as the lower half of the key label.



RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS

Programmer: _____ Date: _____
 Program Title: _____

Punching Instructions	
Graphic	
Punch	

Program Name	15	16	17	18	19	20

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Sng	Chaining File Code (C, L, S)	From Filename	To Filename	Table Array or Routine Name	Entries Per Record (1, 999)	Entries Per Table Array (1, 999)	Entry Length (1, 256)	Data Format (P, B, L, R)	Decimal Positions (0, 9)	Table Array Sequence (A, D)	Table Array Name	Entry Length (1, 256)	Data Format (P, B, L, R)	Decimal Positions (0, 9)	Table Array Sequence (A, D)	Table Array Name	Comments
1						KEYL0L	1		8										SOFTKEY LABELS



RPG CALCULATION SPECIFICATIONS

Programmer: _____ Date: _____
 Program Title: _____

Punching Instructions	
Graphic	
Punch	

Program Name	15	16	17	18	19	20

Sequence Number	Form Type (L)	Control Code (L, O, B, L, R, S, A, N, O, R)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1, 256)	Decimal Positions (0, 9)	High Adjust (H)	Resulting Indicators			Comments				
			And	And	Not								Arithmetic	Plus Minus Zero	Compare					
01						SET OF														
02						DSPLM SET OPT														
03						DSPLM SET OPT														
04						DSPLM SET OPT														
05						DSPLM SET OPT														
06						DSPLM SET OPT														
07						DSPLM SET OPT														
08						DSPLM SET OPT														
09						SET			KEY IN											

Figure 8-17a. Using SET Operation (Continued)



RPG CALCULATION SPECIFICATIONS

Programmer: _____ Date: _____
 Program Title: _____

Punching Instructions	
Graphic	
Punch	

Program Name	15	16	17	18	19	20

Sequence Number	Form Type (L)	Control Code (L, O, B, L, R, S, A, N, O, R)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1, 256)	Decimal Positions (0, 9)	High Adjust (H)	Resulting Indicators			Comments				
			And	And	Not								Arithmetic	Plus Minus Zero	Compare					
F1						SET ON														
F2						SET ON														
F3						SET ON														
F4						SET ON														
F5						SET ON														
F6						SET ON														
F7						SET ON														
F8						SET ON														

Figure 8-17a. Using SET Operation (Continued)

C Calculation Specifications

Sample message file records for example program SETU

\$SET=1	
0001	Product Sales Analysis — Press F1 — Current Month Detail
0002	— Press F2 — Current Month Summary
0003	— Press F3 — Year-to-date Detail
0004	— Press F4 — Year-to-date Summary
0005	— Press F5 — Comparative Detail
0006	— Press F6 — Comparative Summary
0007	— Press F7 — Month-end processing
0008	— Press F8 — Year-end processing
0009	— Press the Function Key(s) for the desired version(s) of this report

Figure 8-17b

Sample Array KEYLBL for Softkey Labels

Current Detail
Current Summary
Y-T-D Detail
Y-T-D Summary
Compare Detail
Compare Summary
MonthEndProcess
Year-EndProcess

Figure 8-17c

DSPLY and DSPLM

With the **DSPLY** and **DSPLM** operations, you can enable your program to print output and accept input at a terminal during calculation operations. If your program is running in a batch job, this allows you to transmit information to, and request information from, the system console. If it is running in an interactive session, it permits you to enter and receive data interactively at your terminal while your program is running.

When used in conjunction with the **SET** operation, the **DSPLY** and **DSPLM** operations can also set on **Function Key Indicators** F1 to F8. See also the **SET** operation discussion and program example for an explanation of this feature.

1. **To display data only**, use the **DSPLY** operation. Enter a field name, literal, array element, or table name as **Factor 1**, and leave the **Result Field** blank. If you enter a field name or array element, its contents will be displayed; if you enter a literal, the literal itself will be displayed; if you enter a table name, the last entry located by a **LOKUP** operation will be displayed. The display appears at the system console (in a batch job) or at your terminal (in an interactive session), after which the program continues. The data you specify by **Factor 1** cannot exceed 249 characters. (Seven characters are used for output of the message **DSPLY**, followed by two blanks.) If the data length is greater than the maximum output length defined for the terminal in use, then output is displayed on multiple lines, with the **DSPLY** heading being printed only on the first line. Output to the system console is limited to 56 characters per line.
2. **To display a User Message only**, use the **DSPLM** operation. Enter a valid message identification as **Factor 1** and leave the **Result Field** blank. The display appears at the system console (in a batch job) or at your terminal (in an interactive session), after which the program continues. The message you specify by **Factor 1** cannot exceed 249 characters. (Seven characters are used for output of the message **DSPLY**, followed by two blanks.) If the message length is greater than the maximum output length defined for the terminal in use, then output is displayed on multiple lines, with the **DSPLY** heading being printed only on the first line. Output to the system console is limited to 56 characters per line.
3. **To permit changing data, and/or setting of function key indicators**, use the **DSPLY** operation. Specify a field name, array element, or table name in the **Result Field** and leave **Factor 1** blank. The data implied by this entry will be printed at the system console (in a batch job) or your terminal (in an interactive session), and the program will pause. Then, the person at the console or terminal can press function keys to set function key indicators, then either change the data displayed (by entering new data) and direct the program to continue (by entering a carriage return), or can request the program to continue without changing the data (by entering an asterisk or carriage-return at the console, or an asterisk at the terminal).

NOTE: At the console, the operator makes these entries via the **=REPLY** console command, described in the console operator's manuals. At other terminals, however, the user makes these entries by directly typing in the data after the output produced by **DSPLY**.

The maximum length of the data specified by the **Result Field** is 256 characters for alphanumeric fields and 15 digits for numeric fields. Input from the system console is limited to 31 characters; output to the system console is limited to 56 characters; input/output at any user terminal is limited to the record-length defined for that terminal during system configuration.

C Calculation Specifications

4. **To display data in one field and permit setting of function key indicators and/or changing data in another field**, use the **DSPLY** operation. Specify both **Factor 1** and the **Result Field**. The program will print both fields on separate lines and pause. At this point the user may press function keys to set function key indicators (if the **SET** operation was specified), change the data in the **Result Field** and/or direct the program to continue, as noted above.
5. **To display a User Message and permit setting of function key indicators and/or changing data**, use the **DSPLM** operation. Specify a valid message identification in **Factor 1** and a field name, array element, or table name in the **Result Field**. The program will print the message followed on a separate line by the **Result Field** contents and pause. At this point the user may press function keys to set function key indicators (if the **SET** operation was specified), change the data in the **Result Field** and/or direct the program to continue, as noted above.

When entering data in response to a **DSPLY** or **DSPLM** operation, you need not enter leading zeros in numeric fields or trailing blanks in alphanumeric fields; your program pads these fields with those characters automatically, aligns the data to the right, and places it in the proper format before placing it in the area specified by the **Result Field**.

Refer to the discussion of the **MSG** operator for the definition of a "valid message identification".

NOTE: Specification of a filename in **Factor-2** is optional. If specified, the file named is not used because the **DSPLY** and **DSPLM** operations perform all input/output directly with your terminal (if running interactively) or with the system console (if running in a batch job). The option to allow a filename in **Factor-2** is available only for compatibility with previous versions of RPG.

EXAMPLES

The program in Figure 8-18 requests three **DSPLY** operations. Suppose that the field **MESSAGE** in the first operation (Item 1) contains the characters **RE-RUN PROGRAM WITH FILE 2**. This information will appear on the console or terminal as follows, and the program will continue without pausing:

```
DSPLY RE-RUN PROGRAM WITH FILE 2
```

Suppose that **FIELD1** in the second operation (Item 2) contains the value **0023**. This information will appear on the terminal as follows, and the program will halt:

```
0023
```

Suppose now that the operator enters **43**, followed by a carriage return; **FIELD1** will now contain **0043**, and the program will continue.

Suppose that, in the third operation (Item 3), **FIELD2** contains **ENTER NEW ID NO; PRESENT NO IS** and **FIELD3** contains **32**. The following information appears at the terminal, and the program halts:

```
DSPLY ENTER NEW ID NO;PRESENT NO. IS  
32
```

The user enters **5**, **FIELD3** now contains **05**, and the program continues; **FIELD2** is not changed.

Note: Header specification column-48 (DSPLY Options) allows you to alter the way that DSPLY operates. Specifically, you can:

- Suppress printing of the DSPLY literal on each use of DSPLY
- Force entry of new Result Field value to be on the same line as where the old value is displayed instead of on the next line, directly below the old value.

(Refer to Section III, 3-20, for a full description of DSPLY Options.)

MSG

With the MSG Operation you can immediately retrieve a message from an MPE User Message Catalog (see Appendix L for discussion of MPE User Message System) during calculation operations and move it into a field in your program.

To specify this operation, enter **MSG** in the **Operation Field**, a valid message identification in **Factor 1**, and the destination field in the **Result Field**. If you enter an indicator in the **High Subfield** and the program is not able to retrieve the message, the program will turn this indicator ON; if no indicator appears in this field, and the message is not retrieved, the **H0** indicator is turned ON.

A valid message identification is a literal or alphanumeric variable containing the message number and, optionally, the set number which identifies the message in an MPE User Message Catalog. The format for a message identification is as follows:

nnnn [:ss]

where nnnn is a message number and ss is an optional set number (1 to 62) in an MPE User Message Catalog. The colon ":" is used to separate the two parameters. The following are examples of valid message identifications:

Contents of Literal or Field	Message Number	\$SET Number
"12"	12	1 (default)
"3:"	3	1 (default)
"32:6"	32	6
"0004:02"	4	2

The **Result Field** must be an alphanumeric field or array. If the result field length is too small to contain the entire message, the message will be right-truncated. A message may be up to 256 bytes in length.

Factor 2, Decimal Positions, Half-adjust, and the Low and Equal Subfields must be blank for the **MSG** operation.

NOTE: When a **MSG** operation is used in your program it will attempt to open a User Message Catalog with the formal designator "CATALOG" in your log-on group and account. An MPE :FILE statement may be required to direct your program to the desired catalog file.

EXAMPLE: :FILE CATALOG=MYMSG.PUB.PAYROL

C Calculation Specifications

An RPG program will open only one User Message Catalog file. If you are using both the MSG Operation code and the message features of the RPG Screen Interface, you should combine your messages into a single User Message Catalog and utilize the \$SET to separate the MSG and Screen Interface messages.

Application Note: The MSG operation code can be used to allow customization of report headings and error messages for multi-company or multi-lingual (international) applications.


READ

With the READ operation, you can immediately retrieve a record from a sequentially-organized demand file during calculation operations. Unlike the FORCE operation, READ makes the record available during the present cycle instead of the next one. READ is similar to the CHAIN operation discussed later in this section, except that READ processes sequential files while CHAIN processes random (direct-access) files.

To specify this operation, enter READ in the Operation Field and the name of the demand file to be read in Factor 2. If you enter an indicator in the Equal Subfield (Columns 58 through 59), and the program encounters the end of the demand file during the READ operation, the program will turn this indicator on; if no indicator appears in this field, and the end of the demand file is encountered, the H0 indicator turns on.

Factor 1 and the Result Field are not used in this operation; neither are the High and Low Subfields (Columns 54 through 57). When your program executes a READ operation, it turns on the record-identifying indicator for the records referenced and extracts the fields specified before doing the next calculation. If the program attempts to read an unidentifiable record type, a run-time error occurs.

With the READ operation, you can only process demand files (including those residing on SPECIAL devices). You cannot use control levels, matching fields, and look-ahead fields with these files, nor can you specify sequence testing for these files on the Input Specifications Sheet.

HEWLETT  PACKARD RPG CALCULATION SPECIFICATIONS Page 3 of 5

Programmer J. BLACK Date 1/2/75

Program Title DSPLY

Punching Instructions	
Graphic	
Punch	

Program Name DISP

Sequence Number	Form Type (C)	Control Level (L, O, U, L, R, S, P, A, N, O, R)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (0-9)	Half Adjust (H)	Resulting Indicators	Comments
			Not	And	And									
1														
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														
26														
27														
28														
29														
30														
31														
32														
33														
34														
35														
36														
37														
38														
39														
40														
41														
42														
43														
44														
45														
46														
47														
48														
49														
50														
51														
52														
53														
54														
55														
56														
57														
58														
59														
60														
61														
62														
63														
64														
65														
66														
67														
68														
69														
70														
71														
72														
73														
74														
75														
76														
77														
78														
79														
80														

Figure 8-18. Using the DSPLY Operation

EXAMPLE

In the program shown in Figure 8-19, the demand file SURTAX is read only if the indicator 07 is on.

CHAIN

The CHAIN operation reads a record from a random, KSAM, INDEX, IMAGE, or direct-access disc file during calculation operations. CHAIN is similar to the READ operation discussed above, except that CHAIN reads records from randomly-organized chained files while READ processes sequential demand files. When your program encounters a CHAIN request, it immediately reads a record from the chained file, turns on any appropriate record-identifying indicators, and moves the input fields into their work areas. The record read is determined by a relative record number (direct files) or a record key (KSAM, INDEX, or IMAGE files), specified as part of the CHAIN request, that defines the position of the record in the file. The relative record number can be obtained from an input record field or can be generated by the RPG/3000 program. It defines the position of the record with respect to the first record in the file. (See also paragraph 4-12.) The record key can be obtained from an input file. You can use the CHAIN operation to read a single record at a time for use by your program, or to load (output) a record onto disc (in direct access files).

Note: The primary difference between chaining to a direct file and chaining to a KSAM, INDEX, or IMAGE file is that:

- In a direct file, you must supply the record number of the record sought in the CHAIN operation.
- In a KSAM, INDEX, or IMAGE file, you must supply the record key for the record sought in the CHAIN operation.

When your program chains to one or more files in a program cycle, record identifying indicators associated with these files stay on throughout the cycle if the operations were successful. When the program chains to the same file more than once in the same cycle, only the last record processed is updated at output time unless you associate exception lines with each CHAIN operation.

Usually when chaining with record keys, the CHAIN operation reads the record whose key value exactly matches the record key specified in **Factor 1**. Alternatively, a relational operator can be specified in the **Result field** so that, if a record with the exact key cannot be found, the record with the next higher key is read. To request chaining, enter CHAIN in the **Operation Field**. Specify **Factor 1** as the relative record number or record key that identifies the record to be retrieved; this can be a numeric literal or numeric field (for record numbers or packed keys) or alphanumeric literal, or field, array, or table name (for record keys).

However, the length of the key field specified for **Factor 1** must be the same as the key field length defined for the file. For KSAM files, a record key specified in **Factor 1** can be shorter in length than the key field length defined for the file (only if a non-blank entry is specified in columns 43 through 45 and if the key type is not "packed"). This is a "partial" key. If a partial key is used, RPG positions the file at the first record whose key (leftmost characters) matches the partial key value from **Factor 1**. The match is done according to the relational operator specified in the **Result field**. Note that this relational matching is also done if a full key, rather than a partial key, is specified in **Factor 1**. Enter the name of the chained file as **Factor 2**.

You can condition the CHAIN operation by entering indicators in the Control Level and Indicators Fields, but you must leave the following fields blank in all cases: Result (Columns 43 through 51), Decimal Positions (column 52), Half-Adjust (Column 53), and Equal Subfield (Columns 58 through 59).

When writing the File Description Specification for the chained file, always enter D (for direct-access files), I or X (for KSAM files), or M (for IMAGE files) in the File Organization Field (Column 32). Then, enter C (for chained) in the File Designation Field (Column 16). If loading this file to disc, also specify O (for output) in the File Type Field. (Column 15).

When performing a chained sequential read (Input Type C) or backward chained sequential read (Input Type R), RPG will automatically retrieve all records in a chain one at a time (a chain is a group of records having the same key value). To do this, the first time you specify a new key value in a CHAIN operation, RPG points to the first record in the chain or to the last record for mode R (using the DBFIND intrinsic) and then reads that record into your program (using the DBGET intrinsic).

NOTE: The DBFIND intrinsic locates the entry for this key value in the Master dataset in order to find the start of the chain in the Detail dataset. However, the record from the Master dataset is not read into your program and you do not specify the Master dataset on File Description specifications.

On subsequent CHAIN operations on the same key value, RPG reads the next record in the chain into your program (using the DBGET intrinsic). When the end-of-chain is reached, the indicator specified in the Low Subfield is turned on. The following table summarizes how the High and Low indicators are set.

High Subfield (Columns 54-55)	Low Subfield (Columns 56-57)	Explanation
OFF	OFF	Success: the next record in the chain is retrieved.
OFF	ON	End-of-chain for the current group of records having the same key value has been reached. (That is, DBGET has failed on the second or later CHAIN operation with the same key value.)
ON	ON	No records for this key value exist in this Detail dataset, but an entry for this key was found in the Master dataset. (That is, DBGET has failed on the first CHAIN operation with a new key value.)
ON	OFF	No records for this key value exist in this Detail dataset and no entry exists in the Master dataset. (That is, DBFIND has failed on the first CHAIN operation with a new key value.)


EXAMPLES

The File Description, Input, and Calculation Specifications for chaining records from a direct-access chained file named CHAINFL appear in Figure 8-20. This program prepares bills for various products purchased from a firm. It reads the customer's name and identification number, each product purchased, and the amount of each purchase, from records in the card file CARDZ (Items 1 and 2). The identification number (ID) is also the relative record number of a record in the chained file CHAINFL (Items 3 and 4); that record contains the customer's address. After calculating the amount of each customer's bill, the program chains to the record containing his address in CHAINFL (Item 5), and prints a bill mailed to him.

A second example, illustrating how to create a direct-access chained file on disc appears in Figure 8-21. The file is created by loading records on cards onto disc. The card file is named KARDS (Item 1) and the

chained file is named CHFILE (Item 2), and described as an output file (Item 3). On KARDS, the field RECNO (Item 4) contains the relative record number to be assigned to each record when it is copied to disc. When the CHAIN operation (Item 5) is encountered, the program first sets the five extents allocated to the chained file (Item 6) to blanks and then reads the field DATA (Item 7) from KARDS, using the relative record number defined in RECNO; it writes the entire record to the disc file CHFILE in the position indicated by RECNO. Any record locations not loaded during this process will remain set to blanks.

An example illustrating chaining to a KSAM file appears in Section X.

HEWLETT  PACKARD RPG CALCULATION SPECIFICATIONS Page 3 of 5

Programmer D. JONES Date 1/5/75

Program Title CHAINX

Punching Instructions	
Graphic	
Punch	

Program Name C H A I N X

Sequence Number	Form Type (C)	Control Level (L, O, L9/ LR, SR, AN, OR)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (0-9)	Half Adjust (H)	Resulting Indicators			Comments	
			Not	And	And								Arithmetic	Plus	Minus		Zero
1																	
2																	
3																	
4																	
5																	
6	C																
7	C																
8	C																
9	C	00019				ID		CHAINCHAINFL									
10	C																
11	C																
12	C																
13	C																
14	C																
15	C																
16	C																
17	C																
18																	
19																	
20																	
21																	
22																	
23																	
24																	
25																	
26																	
27																	
28																	
29																	
30																	
31																	
32																	
33																	
34																	
35																	
36																	
37																	
38																	
39																	
40																	
41																	
42																	
43																	
44																	
45																	
46																	
47																	
48																	
49																	
50																	
51																	
52																	
53																	
54																	
55																	
56																	
57																	
58																	
59																	
60																	
61																	
62																	
63																	
64																	
65																	
66																	
67																	
68																	
69																	
70																	
71																	
72																	
73																	
74																	

Figure 8-20. Chaining to an Input File (Continued)

LOCK and UNLCK

The LOCK and UNLCK operations allow you to perform your own conditional or unconditional locking and unlocking on IMAGE, KSAM, and MPE files. For IMAGE, you may perform locking at the data base, data set, or record level. For KSAM and MPE, you may only lock at the file level.

- **Unconditional Locking.** An unconditional lock is a lock request on a data object (IMAGE data base/set/record, or KSAM or MPE file) such that if the data object is already locked by another process, the current process is suspended until the lock request can be granted for it. The time the process is suspended may seriously degrade performance in an interactive processing environment. To overcome this, conditional locking is recommended.
- **Conditional Locking.** A conditional lock is a lock request on a data object (IMAGE data base/set/record, or KSAM or MPE file) such that if the data object is already locked by another process, the lock will fail and a **Resulting Indicator** will be turned on to inform you of the situation. The process is not suspended. Instead, it continues execution at the next program statement.
- **Enabling Locking.** In order for locking and unlocking to be allowed for a shared file, the appropriate locking facility must be enabled. For IMAGE files, you do this by entering one of the **Locking Modes** (B, S, 1, 9, R, or L*) in column 66 of the **IMAGE Continuation record** for the file. Refer to section IV, for a description of **Locking Modes**.) For KSAM and MPE files, you enable locking by specifying either a **LOCK** or a **NOLOCK* Continuation record** for the file. (Refer to section IV, for a description of the **LOCK** and **NOLOCK** continuation options.)

Note: You can perform your own LOCK and UNLCK operations in **Calculations** even if you have specified a locking mode that also cause RPG to automatically do locking and unlocking for you. (i.e. **Locking Modes** B, S, 1, 9 or R for IMAGE, or the **LOCK** continuation option for KSAM or MPE.)

Extreme caution should be observed when doing this, however, as you must ensure that your own manual locking logic is properly coordinated with RPG's automatic locking.

*Recommended entry (see NOTE above)

- **IMAGE Data Base Locking and Unlocking.** To lock or unlock an IMAGE data base, you specify **LOCK** or **UNLCK** in the **Operation Field**, a file name in **Factor 2**, a data base name in the **Result Field**, and **Resulting Indicators** which both define the type of locking to be done (conditional versus unconditional) and return status information for the operation. **Factor 1** must be blank.

The file name specified in **Factor 2** is a reference to the IMAGE data base to be locked/unlocked. It must be the name specified in columns 7-14 of the **File Description** specification for the data base.

The data base name specified in the **Result Field** must be the data base name given in the **IMAGE Continuation record** for the file named in **Factor 2**. This is the data base you wish to lock/unlock. It must be specified in the **Result Field** as a literal string, not a variable, and must not be enclosed in quotes. Also, a value between 1-256 **must be specified** in the **Result Field Length** (columns 49-51). This value is not used by RPG, but must be specified in order to prevent a compiler syntax error. An entry of 1 is recommended.

Resulting Indicators must be specified for the LOCK and UNLCK operations. The **High Indicator** is optional, but one of either the **Low** or **Equal Indicators** is required. The presence of the **High Indicator** declares conditional locking, whereas its absence declares unconditional locking. See Table 8-2b to determine how **Resulting Indicators** will be set for IMAGE data base locking and unlocking operations.

- **IMAGE Data Set Locking and Unlocking.** To lock or unlock an IMAGE data set, you specify LOCK or UNLCK in the **Operation Field**, a file name in **Factor 2**, and **Resulting Indicators** which both define the type of locking to be done (conditional versus unconditional) and return status information for the operation. **Factor 1** and the **Result Field** must be blank.

The file name specified in **Factor 2** is a reference to the IMAGE data set to be locked/unlocked. It must be the name specified in columns 7-14 of the **File Description specification** for the data set.

Resulting Indicators must be specified for the LOCK and UNLCK operations. The **High Indicator** is optional, but one of either the **Low** or **Equal Indicators** is required. The presence of the **High Indicator** declares conditional locking, whereas its absence declares unconditional locking. See Table 8-2b to determine how **Resulting Indicators** will be set for IMAGE data set locking and unlocking operations.

- **IMAGE Record Locking and Unlocking.** To lock or unlock an IMAGE record, you specify LOCK or UNLCK in the **Operation Field**, a search key in **Factor 1**, a file name in **Factor 2**, and **Resulting Indicators** which both define the type of locking to be done (conditional versus unconditional) and return status information for the operation. The **Result Field** must be blank.

The key specified in **Factor 1** is used as the search value for the record(s) which contain that key. All records with a key item equal to the search key will be locked/unlocked.

The file name specified in **Factor 2** is a reference to the IMAGE data set you wish to search for records to lock/unlock. It must be the name specified in columns 7-14 of the **File Description specification** for the IMAGE data set to which the records belong.

A **KITEM Continuation record** must be specified for the data set in order to name the IMAGE item to be used as the search key for the **Factor 1** search values. (See section IV, 4-36, for a description of the KITEM option.)

Resulting Indicators must be specified for the LOCK and UNLCK operations. The **High Indicator** is optional, but one of either the **Low** or **Equal Indicators** is required. The presence of the **High Indicator** declares conditional locking, whereas its absence declares unconditional locking. See Table 8-2b to determine how **Resulting Indicators** will be set for IMAGE record locking and unlocking operations.

- **KSAM and MPE File Locking and Unlocking.** To lock or unlock a KSAM or MPE file, you specify LOCK or UNLCK in the **Operation Field**, a file name in **Factor 2**, and **Resulting Indicators** which both define the type of locking to be done (conditional versus unconditional) and return status information for the operation. **Factor 1** and the **Result Field** must be blank.

The file name specified in **Factor 2** must be the name specified in columns 7-14 of the **File Description specification** for the file.

C Calculation Specifications

Resulting Indicators must be specified for the LOCK and UNLCK operations. The **High Indicator** is optional, but one of either the **Low** or **Equal Indicators** is required. The presence of the **High Indicator** declares conditional locking, whereas its absence declares unconditional locking. See Table 8-2b to determine how **Resulting Indicators** will be set for KSAM and MPE file locking and unlocking operations.

Table 8-2a. LOCK/UNLCK Summary

Type of Lock	Factor 1	Operation	Factor 2	Result Fld	Result Len
IMAGE Data Base	blank	LOCK/UNLCK	filename	data base	1-256
IMAGE Data Set	blank	LOCK/UNLCK	filename	blank	blank
IMAGE Record	key value	LOCK/UNLCK	filename	blank	blank
KSAM File	blank	LOCK/UNLCK	filename	blank	blank
MPE File	blank	LOCK/UNLCK	filename	blank	blank

Table 8-2b. LOCK/UNLCK Resulting Indicators

Resulting Indicator Set ON	IMAGE			KSAM and MPE		
	LOCK	UNLCK	Record	LOCK	UNLCK	
High (Conditional Locking only)	Status = 20 → Data base locked or contains locks	Status = 20 → Data base locked or contains locks 22 → Data set locked by another process 23 → Entries locked within set 24 → Item conflicts with current locks 25 → Entries already locked	Status = 20 → Data base locked or contains locks 22 → Data set locked by another process 23 → Entries locked within set 24 → Item conflicts with current locks 25 → Entries already locked	Status > 0 → Exceptional error	Condition Code > → Locked by another process	Condition Code > → Not already locked
Low	Status < 0 → File System or memory manager failure	Status = -135 → Second lock without CAP = MR	Status = -135 → Second lock without CAP = MR	Status < 0 → File system or memory manager failure	Condition Code < → Not opened with dynamic locking facility enabled or need MR capability	Condition Code < → Not opened with dynamic locking facility enabled or need MR capability
Equal	Status = 0 → Request granted	Status = 0 → Request granted	Status = 0 → Request granted	Status = 0 → Request granted	Condition Code = → Request granted	Condition Code = → Request granted
None	Status = any value other than above	Status = any value other than above	Status = any value other than above	Can never happen — will always have one Resulting Indicator ON	Can never happen — will always have one Resulting Indicator ON	Can never happen — will always have one Resulting Indicator ON

CLOSE

The CLOSE file operation allows you to dynamically close any open file. This forces normal end-of-file actions to be performed earlier than the end of program. Use this operation to allow other users to gain access to the file, release system resources used for the file, or release printer files to print.

Specify this operation by entering CLOSE in the **Operation Field**, the name of the file to be closed in **Factor-2** and an indicator in the **Low subfield** (Columns 56 through 57). The indicator will be turned on if the CLOSE operation fails.

Once a file is closed, the file cannot be reopened and cannot be used for any further input/output operations.

The CLOSE file operation is similar in function to the "Release file" operation on Output specs. Refer to Section IX and Table 9-0a for a complete description of "Release file".

8-37. Debug Operation

The DEBUG operation helps you identify errors in a program that is not working as it should. You can specify it at any point or points in your calculation specifications. Whenever your program executes this operation, it prints a list of all indicators that are on at this point in the program. If you so request, the program will also write out data that identifies this particular DEBUG operation (by distinguishing it from others in your program), and/or the contents of a particular data area as it presently exists.

To request this operation, enter DEBUG in the **Operation Field**. Then enter, in **Factor 2**, the name of the file on which the debugging information is to appear. If you request more than one DEBUG operation in your program, you do not need to specify the same output file as **Factor 2** in all of those operations.

Also, if you request more than one DEBUG operation, you may want to provide a way of telling which of these operations is associated with a particular list of indicators appearing on your output. You can do this by entering, as **Factor 1**, a literal or field whose value will appear as part of the record showing the indicator settings. The contents of this literal or field should not exceed eight characters. You may not enter an unindexed array name as **Factor 1**.

If you desire, you can use the **Result Field** to specify a field, table entry (the last obtained by a LOKUP operation), array element, or entire array whose contents will appear on records following that showing the indicator settings. The data to be printed cannot exceed 256 characters.

You can condition the **DEBUG** operation by entering indicators in the **Control Level** and **Indicators Fields**, but you must leave the fields in **Columns 48 through 59** blank in all **DEBUG** specifications.

If the **DEBUG** operation is to take effect, you must enter 1 in the **Debug Field** (**Column 15**) of the **Control Record Specification**. You can suspend **DEBUG** operations by leaving the **DEBUG Field** blank. When you do this, the program ignores the **DEBUG** requests and prints a warning message.

The first record output as a result of a **DEBUG** operation lists the indicators that are set on, plus any field or literal that you have optionally specified to identify that particular **DEBUG** operation. This record appears in the following format, and is always output:

Position	Content
1 through 8	The characters DEBUG=
9 through 12	The source statement number assigned by the compiler to this DEBUG specification; if you have not specified a field or literal in Factor 1 , this number can help you identify which DEBUG operation is associated with this output.
13 through 14	Blank
15 through 22	The contents of any field or literal that you specified as Factor 1 to identify this operation.
23	A minus sign, if the Factor 1 data is a negative value; a blank, if it is not.
24	Blank
25 through 39	The characters INDICATORS ON=
40	Blank
41 through 80	Indicators on at this point, each followed by a blank.

If so many indicators are on that they cannot all appear on one record, they will be shown on additional records; on each of these records, **Positions 1 through 40** will be blank, and the indicators will appear beginning with **Position 41**.

The next record output shows the content of any field, table entry, array, or array element that you specified in the **Result Field**. It appears in this format, if requested in the **Result Field**:

Positions	Content
1 through 13	The characters FIELD VALUE = .
14	Blank

C Calculation Specifications

Position	Content
15 through 80	The contents of the field, table, array or array element specified in the Result Field . The first 66 characters of this field appear in Positions 15 through 80 of the first record following the indicator listing. If this data exceeds 66 characters, it will be continued in Positions 15 through 80 of as many additional records as required, up to a maximum of 256 characters. If this is numeric data, it is written in unpacked format, zero-suppressed, with any sign appearing to the right of the field (a minus sign for negative values and a blank for positive values). Fields containing zeros are written to show only the rightmost zero and the sign. The program does no other editing on this data. If the Result Field contains an array name, the elements are written in order, each on an individual record.

EXAMPLE

Figure 8-22 shows a DEBUG request that lists, on the file BADFILE, all indicators that are on at the time the request is encountered (Item 1). It also prints the literal FIRST as part of this output (Item 2) and displays the contents of element 3 of ARR1 (Item 3), which is the value 370.

8-38. Record Number Conversion Operations

Your program can invoke special conversion routines that you supply to convert fields of data in RAF's or chaining files to relative record numbers; these values are then used to retrieve other records from direct-access files. The conversion routines are identified in the **Table/Array/Routine Name Field** (Columns 27 through 32) of the File Extension Specifications, and are automatically invoked whenever conversion is required.

As with subroutines, there are two kinds of conversion routines--internal and external. **Internal conversion routines** are coded in RPG/3000 language as part of your program. **External conversion routines** are typically coded in another language such as SPL/3000, and exist as separate entities, generally on mass storage devices.

8-39. INTERNAL CONVERSION ROUTINES. You code internal conversion routines on the Calculation Specifications Sheet. They can appear anywhere within your program. Each must begin with an RPGCV operation and end with an ERPGC operation.

RPGCV

The RPGCV operation is the first specification in the conversion routine; it is followed immediately by the operations comprising the body of the routine. To specify this operation, enter RPGCV in the **Operation Field**. Then, as **Factor 1**, specify the name of the conversion routine; this must be the same name entered in the **Table/Array/Routine Name Field** of the File Extension Specification. In the **Result Field**, enter the name of the numeric field that will contain the converted relative record number provided by the routine. **Factor 2** is not used.

C Calculation Specifications

When a conversion routine is used with a RAF, the RAF data is automatically made available to the routine in a field named CONTD. This field is always an alphanumeric field of the length specified in the Record Address Field Length Field (Columns 29 through 30) of the File Description Specifications.

ERPGC

This operation terminates the conversion routine. Specify it by entering ERPGC in the Operation Field. Factors 1 and 2 and the Result Field are not used in this operation.

EXAMPLE

Figure 8-23 shows an internal conversion routine named CONV (Item 1) to which your program branches when a record address file (RAF) is read. It obtains the data to be converted from the field named CONTD, converts the data, and stores the resulting relative record address in RESULT.

8-40. EXTERNAL CONVERSION ROUTINES. You code these routines separately from your program, making them available to the program on various files. Within the program, you need only one operation to specify these routines: **EXTCV**.

EXTCV

This operation, specified anywhere within your program, makes the conversion routine available whenever it is needed. To specify it, enter EXTCV in the Operation Field. Then, in Factor 1, specify the internal name (or label) of the routine (as defined in the Table/Array/Routine Name Field of the File Extension Specifications). As Factor 2, enter the name of the routine as defined externally (in the Procedure Head if it is an SPL/3000 routine). In the Result Field, enter the name of the field that is to contain the converted relative record number supplied by the routine after it is executed. (Within your program, you must define this field as a numeric field. But, because the EXTCV operation makes this field available to the external routine, you need not specify an RLABL operation to perform that function.)

In coding the subroutine as an SPL/3000 procedure, use the following format:

```
PROCEDURE exitname (result),
```

```
BYTE ARRAY result;
```

```
(Procedure Body);
```

```
.  
. .  
.
```

exitname is the name of the routine (procedure) as specified in Factor 2 of the EXTCV operation; **result** is the field containing the result of the routine, as specified in the Result Field of the EXTCV operation. In the procedure, the result data is defined as a byte array, but is received by the RPG/3000 program as a packed decimal field.

EXAMPLE

In Figure 8-23, the EXTCV operation (Item 2) makes available an external conversion routine identified externally as EXTNAM and internally (in the RPG/3000 program) as INTNAM. After the routine is executed, the converted record number is stored in the field RECNO.

C Calculation Specifications

8-41 SYSTEM OPERATIONS

Your program can utilize various functions of the MPE/3000 operating system directly from calculations. In most cases, this is done by the execution of corresponding system intrinsic functions. Additional information on the use of system intrinsics can be found in the *HP 3000 Computer Systems MPE Intrinsics Reference Manual (30009-90010)*.

TIME

The TIME operation returns the current system time of day and, optionally, the system date in the Result Field.

To specify this operation, enter TIME in the Operation Field, and in the Result Field enter the name of the numeric field which is to receive the time or time and date values. The Result Field Length must be either 6 or 12, and the Decimal Positions must be 0. Factor 1, Factor 2, Half Adjust, and Resulting Indicator Fields must be blank.

To return the time of day only, the Result Field Length must be 6 digits with 0 (zero) decimal places. To return both the time of day and the system date, the Result Field Length must be 12 digits with 0 (zero) decimal places.

The time of day is always returned in the first 6 digits of the Result Field in the format:

hhmmss

where hh represents hours, mm represents minutes, and ss represents seconds. Time is based on the 24-hour time system.

If the system date is also requested, it is returned in digits 7 to 12 of the Result Field in one of the following formats:

mmddy — Domestic Format (Header Record column 21 is blank)

ddmmyy — Foreign Format (Header Record column 21 is I, J, or D)

where mm is the month, dd is the day of the month, and yy is the year.

The date is retrieved dynamically from the system date whenever the TIME operation is executed. It is not taken from the static UDATE field which could be different than the dynamic system date. (For example, if program execution begins shortly before midnight on 12/31/80, UDATE is initialized to 123180 and never changes throughout the program run. If the program is still running after midnight, the value of UDATE is still 123180, whereas the TIME operation will return the date 010181.) The time of day is always a dynamic value.

System intrinsics used for the TIME operation are:

- CLOCK (for system time)
- CALENDAR (for system date)

EXAMPLE

The TIME operation is demonstrated in Figure 8-24 (Items 1 and 2).

TIME2

The TIME2 operation retrieves the current system date and time in a formatted 40-character string. Factor 2 and the Result Field Length are used to determine which characters from the 40-character string are to be extracted and returned in the Result Field.

To specify this operation, enter TIME2 in the Operation Field, and in the Result Field enter the name of the alphanumeric field which is to receive the extracted data. Factor 2 must specify the position in the 40-character string where the extraction is to begin, and the Result Field Length specifies the number of characters to be extracted. Factor 1, Decimal Positions, Half Adjust, and Resulting Indicator Fields must be blank.

If Factor 2 is a numeric constant, it must be an unsigned numeric value from 1 to 40, specified in columns 33 and 34. Columns 35 to 42 must be blank. If Factor 2 is a numeric field, its defined length must be 1 to 4, with 0 (zero) decimal places. The Result Field Length must be 1 to 40.

You cannot request an extraction that would overflow the 40-character string. That is, the end position of the extraction (as computed from Factor 2 and Result Field Length) cannot exceed 40.

The 40-character date and time string is formatted as follows:

day,	mon	dd,	year,	hh:mm	xM	JULIAN:nnn	←Format
↑	↑	↑	↑	↑	↑	↑	←Starting Positions
1	6	10	14	20	26	38	
↓	↓	↓	↓	↓	↓	↓	←Example
MON,	FEB	11,	1980,	9:25	AM	JULIAN:042	

where day is day of the week; mon is month name; dd is day number in the month; year is 4-digit year; hh is hours; mm is minutes; xx is AM or PM; and nnn is Julian day number in the year.

System intrinsics used for the TIME2 operation are:

- DATELINE (for 1st 27 characters of the date and time string)
- CALENDAR (for the Julian day number in the year)

EXAMPLES

The TIME2 operation is demonstrated in Figure 8-24 (Items 3 to 8). Given the date 02/11/80 and time 9:25 AM, the Result Field contents would be:

- (Item 3) MON, FEB 11, 1980, 9:25 AM JULIAN:042
- (Item 4) MON, FEB 11, 1980, 9:25 AM
- (Item 5) FEB 11, 1980
- (Item 6) 1980
- (Item 7) 9:25 AM
- (Item 8) 042

PUTJW

To put a JCW into the system JCW table, you specify **PUTJW** in the **Operation Field**, the value to be **PUT** in **Factor 1**, the name of the JCW to obtain the value in **Factor 2**, and **Resulting Indicators** which return status information on the **PUTJW** operation.

Factor 1 may be a value in the range of 0 to 65535 or it may be a variable declared from 1 to 8 digits in size with zero decimal positions. If you incorrectly try to put a value not in the range given above, RPG will halt with the runtime error "Invalid Numerical Data" and not perform the **PUT**.

Factor 2 may contain an alphanumeric variable or a literal value enclosed in quotes. The JCW name specified in **Factor 2** must begin with a letter and terminate with a blank.

At least one **Resulting Indicator** must be specified for this operation. After execution of **PUTJW**, the **High Indicator** will be on if a system table overflow occurred; the **Low Indicator** will be on if you tried to put a value in a JCW name that doesn't begin with a letter, and the **Equal Indicator** will be on if the **PUT** was successful.

If the named JCW already exists in the JCW table, its value will be replaced by the **Factor 1** value. Otherwise, a new entry will be made in the JCW table with initial value equal to the **Factor 1** value.

See Table 8-3 for a summary of the **PUTJW** operation.

System intrinsics used for the **PUTJW** operation are:

- **PUTJCW**

FNDJW

To find a JCW in the system JCW table, you specify **FNDJW** in the **Operation Field**, the name of the JCW you are finding in **Factor 2**, the variable to obtain the JCW's value in the **Result Field**, and **Resulting Indicators** which return status information on the **FNDJW** operation.

Factor 2 may contain a alphanumeric variable or a literal value enclosed in quotes. The JCW name specified in **Factor 2** must begin with a letter and terminate with a blank.

The **Result Field** must be defined as numeric with zero decimal positions.

At least one **Resulting Indicator** must be specified for this operation. After execution of **FNDJW**, the **High Indicator** will be on if the JCW was not found; the **Low Indicator** will be on if the JCW name does not begin with a letter; and the **Equal Indicator** will be on if the JCW was found and the value was returned to the variable specified in the **Result Field**.

See Table 8-3 for a summary of the **FNDJW** operation.

System intrinsics used for the **FNDJW** operation are:

- **FINDJCW**

C Calculation Specifications

Table 8-3. Summary Table for PUTJW and FINDJW

Factor 1	Operator	Factor 2	Result	Result Indicators	
value	PUTJW	JCW name	blank	>	table overflow
				<	illegal name
				=	successful
blank	FNDJW	JCW name	variable	>	JCW not found
				<	illegal name
				=	successful

FNUM

To obtain the **MPE** file number for any file used in your program, you specify **FNUM** in the **Operation Field**, a file name in **Factor 2**, and a numeric variable in the **Result Field** to which the **MPE** file number will be returned. **Factor 1** must be blank.

Factor 2 must be a file name that is specified in your **File Description** specifications.

The **Result Field** must be defined as numeric with zero decimal positions. The **MPE** file number of the file specified in **Factor 2** will be returned to this field in **Decimal** format (see the **NOTE** below).

No system **Intrinsics** are used by the **FNUM** operation.

Note: If you are passing an **MPE** file number obtained using the **FNUM** operation to an external procedure for use as a parameter in file processing **intrinsics** (or other procedures), be aware that the **Result Field** value being passed is stored internally by **RPG** as **packed-decimal** data, not **Integer** data (as required for most **intrinsics** and other procedures).

Therefore, your external procedure must convert this data field from **Decimal** to **Binary** format prior to using it in any operations that required **Binary-type** data (i.e. **Integer**, **Logical**, etc.).

8-42. DEFAULT SUMMARY

If you leave the optional fields of the Calculation Specifications blank, the default specifications shown in Table 8-4 apply:

Table 8-4. Calculation Specification Default Values

Columns	Field	Default Specifications
1 through 5	Sequence Number	No sequence number applies.
7 through 8	Control Level	Detail calculation, done at detail time.
9 through 17	Indicators	Operation done for every record if Control Level Field does not contain L0 through L9, or SR.
43 through 48	Result	No result applies.
49 through 51	Field Length	This is an alphanumeric field, or a numeric field described elsewhere in program.
52	Decimal Positions	This is an alphanumeric field, or a numeric field described elsewhere in program.
53	Half-Adjust	Do not half-adjust data.
54 through 59	Resulting Indicator	No indicator assigned.
60 through 74	Comments	No comments made.
75 through 80	Program Name	None.



OUTPUT SPECIFICATIONS

SECTION

IX

For every output, update, or combined file that your RPG/3000 program writes data upon, you must enter an Output Specification. In this specification, you define two types of information:

1. The first kind describes which records are to be written to the output file. This information appears in the Record Description Fields (Columns 7 through 31):
2. The second kind describes the formats and locations of the fields in the output records. This information appears in the Field Description Fields (Columns 23 through 70).

Note: Columns 23 through 31 overlap functionally, since they can be used for both record and field descriptions.

9-1. FIELDS

The RPG/3000 Output Specifications contain the following fields:

9-2. Sequence Number (Columns 1 through 5)

This field contains the source record sequence number, described in Paragraph 2-3.

9-3. Form Type (Column 6) (Required)

This field denotes the type of specification. It always contains the following entry:

Column 6 Entry	Meaning
0	Output Specification.

Because this entry is required and is always the same, it is pre-printed on the specifications sheet for your convenience (Figure 9-1, Item 1).

9-4. Record Description Fields (Columns 7 through 31)

Output records can be printed, written to disc or tape, or punched on cards. In Columns 7 through 31, you describe these records. You can begin these descriptions on the first line of the Output Specifications Sheet. On all lines containing entries in Columns 7 through 31, leave Columns 32 through 70 blank.

9-5. **FILE NAME (COLUMNS 7 THROUGH 14) (REQUIRED).** In this field, enter the name of the file whose records and fields you are presently describing.

Columns
7 through 14
Entry

Meaning

Valid file name. (File names contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)

Name of output, update, or combined file containing records and fields described.

This file name must be that of an output, combined, or update file — it cannot refer to an input or display file. Begin this entry in Column 7. Be sure to spell the name exactly as it appears on your File Description Specification Sheet, where it first occurs in your program. All entries on this and all following lines will apply to this file until a new file name appears in this field on the specifications sheet. Thus, if you describe more than one record for this file, you do not need to repeat the file name on subsequent record-description lines (although you may do so if you desire). If you wish to intersperse record descriptions for several files with one-another you must enter the file name at the beginning of each record description for that file.

Often you intend to direct the output file to a printer device. But, RPG does not provide a way to indicate this in your source program. Instead, the RPG compiler and library use the definitions shown in Table 9-0a to determine if the current output file is intended to be printed, and if so, what actions might be taken.

Table 9-0a. Printer File Types

Type of File	Definition	Possible Actions
List file	An output, update, or combined file for which output specs columns 17 through 22 contain Space and/or Skip entries (and columns 16 through 18 do not contain ADD or DEL). Also, the file is not specified as Indexed (I, S, or M in File Description specs in column 32).	<ul style="list-style-type: none"> • Opens the file with the CCTL (Carriage Control) option. • Performs Form Positioning if requested in Header spec column 41. • Sets “post spacing” and “auto page eject” directives in the output spoolfile. • Performs an initial page eject unless suppressed by Header spec column 47 or 41. • Adds carriage control (CCTL) directives to output records. • Performs line counting and channel operations according to Header spec column 53 (Carriage Control Type).
Print file	An output file that is either a “List file” (as defined above) or is opened at runtime with the CCTL (Carriage Control) option specified in an MPE file equation.	<ul style="list-style-type: none"> • “Release file” operations (“R” in Output specs column 16) close the output spoolfile and then open a new spoolfile to receive additional output.

EXAMPLE

To show that the records being described should be written to an output file name OUT, enter this name in the File Name Field (Figure 9-1, Item 2).

O Output Specifications

9-6. **TYPE (COLUMN 15) (REQUIRED).** In this field, you must define the record being described as one of four specific types:

Column 15 Entry	Meaning
H	Heading record.
D	Detail record.
T	Total record.
E	Exception record.

Heading records identify information on your output report or file that generally remains the same from page to page, such as the report title, column headings, and current date; they may also contain a page number that is incremented for each page. On the report output in Figure 9-2, the first line (Item 1) is a heading record containing the date, report title, and page number. The second line (Item 2) is a heading record that contains column headings.

①	DATE	10/15/75	ABSENTEE REPORT	PAGE NO. 1	
②	{	DIVISION	DEPARTMENT	EMPLOYEE	DAYS OUT
③	{	01	11	BABCOCK, J. D.	2
		01	11	JONES, F. D.	1
		01	11	MASON, M. M.	1
⑤	{	DEPARTMENT	11	TOTALS	4
④	{	01	12	MOONEY, J. P.	5
		01	12	POMEROY, G.	2
		01	12	RASKIN, K. T.	1
⑥	{	DEPARTMENT	12	TOTALS	8
⑦	{	DEPARTMENT	20	TOTALS	10
⑧	{	DIVISION	1	TOTALS	53

Figure 9-2. Output Record Types

Detail records generally contain the most basic subject matter of the report. Usually, this information is closely-related to the input records — either it is read directly from those records or is calculated by the program on the basis of the data furnished by them. On the report in Figure 9-2, the data on the detail records (Items 3 and 4) is taken directly from input records containing the division, department, name, and days absent of each employee who was away from work during the period ending on October 15, 1975.

Total records typically show sums accumulated by adding and otherwise manipulating data from one or more fields in several detail records. On the report in Figure 9-2, total records reflect the total man-days employees were absent within each department (Items 5, 6, and 7), and within an entire division (Item 8).

Exception records are written or punched during calculation time, only when the EXCPT operation (Paragraph 8-36) is encountered. They represent unusual cases and are designated by the entry E in the Type Field.

Within each type, the records are processed in the order you specify them on the **Output Specifications Sheet**. One easy way to define them is to describe the records for each file in the following order: heading, detail, total, and exception. Alternatively, you can define all heading records for all output files, then all detail records for all output files, and so forth.

On primary or secondary update files, you cannot specify exception records conditioned by control-level indicators (L0 through L9 and LR) or total records, as the results of the update will be unpredictable.

Use of control level and overflow indicators in specifying when heading and detail records are written is described in Paragraph 9-12. If you must continue a record description onto another line, you can use AND or OR Lines as noted in Paragraph 9-23 and 9-24.

EXAMPLE

The Output Specifications in Figure 9-1 define a heading record (Item 3), a detail record (Item 4), and a total record (Item 5).

9-7. RECORD ADDITION/DELETION (COLUMNS 16 THROUGH 18). You specify records to be added to or deleted from an output or update disc file in the Record Addition/Deletion Field.

Columns 16 through 18 Entry	Meaning
ADD	Add new records to an update or output file.
DEL	Delete records from an update file. This entry applies to KSAM, INDEX, or IMAGE files only.
blank	For sequential or KSAM output files, write new records to the beginning of the file. For INDEX or IMAGE output files, insert new records in the file. For update files, update the current record.

When adding records to any file (ADD entry), you must also specify A in the **File Addition Field** (Column 66) of the **File Description Specifications**. For sequential or KSAM files, new records are

O Output Specifications

appended to the file. For INDEX or IMAGE files, records are added in key sequence. If you omit the A entry for a sequential or KSAM file, records are written at the beginning of the file, overlaying any information presently there. If you specify the A entry for an output file, you must also specify ADD for all output records for that file. To determine which records are affected by these entries, see Table 4-4 in section IV.

EXAMPLE

Request the addition of new records to the file named OUTDIR by entering ADD in the appropriate record descriptions (Figure 9-1, Items 6 and 7).

9-8. STACKER SELECT (COLUMN 16). If your program performs output to an 80-column combined card-reader/punch unit, you can indicate which stacker on the unit should be used for the records described. (The cards will be directed to this stacker after they are punched.) Select the stacker by making an entry in the Stacker Select Field of the record identification line, as follows:

Column 16 Entry	Meaning
1 or blank	Select Stacker No. 1
2	Select Stacker No. 2

This entry is valid for output and combined files only. When a record is punched for a combined file, the entry in this field overrides any stacker-select entry on the Input Specifications. If you specify a stacker select code for combined files written at total time, erroneous stacker-selection may take place. If you wish to override the default stacker (Stacker No. 1) for all lines in a set of OR lines, you must explicitly specify the stacker desired on each OR line.

EXAMPLES

Examples of stacker selection appear in Figure 9-3. In the first record definition for the output file named CARDFL (Item 1), which does not describe an AND/OR relationship, a detail output card is punched and directed to Stacker No. 2 (Item 2) when the general indicator 01 is set ON. In the second record definition for the file (Item 3) which defines an AND relationship, a detail card is punched and directed to Stacker No. 1 (Item 4) when the indicators 02, 03, 04, and 05 are all ON. In the third record definition (Item 5), which defines an OR relationship, a total card is punched and transmitted to Stacker No. 2 (Item 6) when indicators L1 and 01 are both ON. The card is sent to Stacker No. 1 (Item 7) when indicators L3 and 07 are both ON. It is directed to Stacker No. 1 by default (Item 8) when the Last-Record (LR) indicator is ON.

A complete program illustrating stacker-selection appears in Section X, Paragraph 10-19.



RPG OUTPUT SPECIFICATIONS

Page _____ of _____

Programmer NORTON, S. E. Date 2/6/77

Program Title CARDOUT

Punching Instructions	
Graphic	
Punch	

75	76	77	78	79	80
Program Name					

Sequence Number	Form Type (O)	File Name	Type (H/D/T/E)	SS/Fetch (Y/Z/F)	Space	Skip	Output Indicators	Field Name	Edit Code	Blank After (B)	End Position (1-9999)	Packed/Binary (P/B/L/R/Z/A)	Constant or Edit Word	Comments
0001		CARDFL	D	2			01							
0002							01	DATA1						1
0003			D	2			01							
0004			A	ND			02 03 04							
0005			D	2			01	DATA2						3
0006			T	2			L1 01							
0007			O	R			L3 07							
0008			O	R			L3 07							
0009								DATA3						5

Figure 9-3. Stacker Selection
9-7

O Output Specifications

9-9. **FETCH OVERFLOW (COLUMN 16).** With this field, you can request output of overflow information before it is normally generated on the current printed page.

Column 16 Entry	Meaning
F	Output overflow information at this point (by fetching the RPG/3000 Overflow Routine).
Blank	Suppress overflow output at this point.

Use this field only if this is a file to be written to a printer, or a file whose output is controlled by Line Counter Specifications, to which you have assigned an overflow indicator. This indicator is assigned in the **Overflow Indicator Field** (Columns 33 through 34) of the File Description Specifications, and is turned on when your program prints a record (line) on or below the overflow line; this signals the beginning of overflow processing, described below. (If this indicator is turned on while header, detail, total, or exception records are being written, the indicator is again turned off after the header and detail records of the next cycle are written.) If you have assigned an overflow indicator to the file but never use it to condition an output record, no special processing takes place when the overflow line is reached. If you have not assigned an overflow indicator to the file, the program skips to the top of the next page (normally associated with Channel 1 of the carriage control tape) whenever the overflow line is reached.

Whenever the overflow indicator is both assigned and on, your program begins overflow processing. Normally, this occurs when the printer reaches the overflow line. During overflow processing, the following events occur:

1. The program writes all detail records (not conditioned by the overflow indicator) remaining to be output during this program cycle.
2. The program next prints all total lines (not conditioned by the overflow indicator) remaining to be output during this program cycle.
3. The program then automatically calls the **RPG/3000 Overflow Routine** to print all records conditioned by the overflow indicator. (These are called **overflow records**.)
4. If you have specified a skip to a new page, the Overflow Routine advances the printer to that page.
5. The Overflow Routine terminates, and control returns to your program.

If, however, you do not wish your program to print all of the remaining detail and total lines on the page before it prints the overflow records and advances to the next page, you can request it to call (fetch) the Overflow Routine ahead of the customary time. Request this by entering F in the **Fetch Overflow Field**. This suppresses the printing of this record and all remaining detail and total records until the overflow

records are printed at this point. The fetch will only occur if all indicators specified in Columns 23 through 31 of the Output Specifications are satisfied and an overflow has actually occurred.

Because the Overflow Routine does not automatically advance the printer to the next page, you must specify a skip operation (if you desire one) in the lines conditioned by the overflow indicator.

If you want to fetch the overflow routine for each record in an OR relation, you must specify F in each OR line.

Leave the Fetch Overflow Field blank for all output files not destined for a printer or controlled by Line Counter Specifications.

EXAMPLE

To fetch the Overflow Routine whenever a total record from the file REPORT is processed, enter F in the Fetch Overflow Field (Figure 9-4. Item 1).

9-9A. RELEASE FILE (COLUMN 16). In this field, you can request that the output file be released after writing the current output record. The release action is executed differently for each type of file, as shown in Table 9-0b.

Table 9-0b. Release File Actions

File	Action
Print File (See Table 9-0a for a definition of "Print File".)	The file is closed and then reopened, so the output spoolfile is released to print and a new spoolfile is created to receive additional output. If a Print file is redirected to disc with an MPE file equation (or by specification of "DISC" as the Device Class Name on File Description specs), the release option causes the disc file to be closed and then reopened in "append" mode. All output is concatenated into one disc file. You should create the disc file with the CCTL (Carriage Control) option so the file can later be sent to a printer (for example, using the FCOPY command) and still print as expected.
WORKSTN file	The WORKSTN file is closed and not reopened. This releases the terminal from block mode operation, which allows the program to continue terminal I/O in non-block mode. (This operation is equivalent to using the CLOSE operator in Calculations.)
Any other file	The file is closed and not reopened. This operation is equivalent to using the CLOSE operator in Calculations.

Column 16

Entry

Meaning

R Release the file (close or close and reopen the file, as shown in Table 9-0b).

Blank Do not release the file.

The Output spec record that specifies the release option is written prior to the file being released, except when no fields have been specified for the record. If no fields have been specified, the file is immediately released without writing a final blank line or record.

The Release file operation is similar in function to the CLOSE file operation on Calculation specs. Refer to Section XIII for a complete description of the CLOSE file operation.

O Output Specifications

9-10. SPACE (COLUMNS 17 THROUGH 18). You can specify line-spacing before and/or after output of this record by making entries in this field. The field is actually composed of two subfields; **Before** (Column 17) and **After** (Column 18), that allow you to determine when spacing occurs. In both fields, you can enter:

Columns 17 or 18 Entry	Meaning
blank	No spacing (If both space and skip fields (columns 17-22) are blank, then single spacing (default) occurs after printing each record.)
0	Suppress spacing.
1	Space one line.
2	Space two lines.
3	Space three lines.

Use this field only for files directed to printers or terminals, or those whose output is controlled by Line Counter Specifications. To space before the record is printed, make your entry in Column 17; to space after this record is printed, make the entry in Column 18. You can specify up to six lines between printed records by entering 3 in both Columns 17 and 18.

Because of their particular carriage-control mechanisms, terminals automatically space one line prior to printing a record. Thus, entries of 0, 1, or blank in Column 17 all result in single-spacing before a record is printed on a terminal. Entries of 2 or 3 in that subfield, however, result in double or triple spacing respectively.

Note: Overprinting (printing a second, separate output line on the same physical print line) is invalid on the HP 2607A printer. Any attempt will result in the overprinted text being printed on separate lines. Multiple overprinting (using more than one overprint on the same line) is invalid on the HP 2608A printer.

When your program spaces beyond the overflow line, but not onto a new page, and an overflow indicator is assigned to this file, that indicator is turned on and remains on until all overflow records are printed.

HEWLETT  PACKARD

RPG OUTPUT SPECIFICATIONS

Page 5 of 5

Programmer JAMES Date 3/3/75
 Program Title FISCAL REPORT

Punching Instructions

Graphic	
Punch	

Program Name FREPT

Sequence Number	Form Type (O)	File Name	Type (W/D/T/E)			Space	Skip	Output Indicators						Field Name	Edit Code	Blank/Alter (B)	End Position (1-9999)	Packed/Empty (P/B/L/R/Z/4)	Constant or Edit Word	Comments
			O	A	N			D	Before	After	Not	Not	Not							
1	0	REPORT	H																	
5	0																			
20	0		D																	
30	0																			
40	0																			
50	0		T	F																
60	0																			

Edit Codes

1	A	J	comma & zero balances	X	Remove Plus Sign
2	B	K	comma	Y	Date Field Edit
3	C	L	zero balances	Z	Zero Suppress
4	O	M			
	No Sign	CR			

Figure 9-4. Fetching Overflow

O Output Specifications

The Space Field is related to the Skip Field (Columns 19 through 22), explained in Paragraph 9-11. The Skip Field permits moving the carriage ahead by several lines at once, without proceeding line-by-line. If you request both spacing and skipping on the same line of your specification sheet, these operations will be done in this order:

1. Skip before printing.
2. Space before printing.
3. Skip after printing.
4. Space after printing.

If positions 17-22 are blank, space 1 after is assumed. However, if entries are made in position 17 (space before) and/or positions 19-22 (skip before and skip after) and no entry is made in position 18 (space after), no space occurs after printing.

EXAMPLE

To request double-spacing before each detail record is printed, enter 2 in the Before Subfield (Column 17) (Figure 9-4, Item 2). If you are using OR lines and different spacing or skipping is required, you can specify it in the space or skip fields of the OR line; if you leave these fields blank, the same spacing and skipping previously specified is used.

9-11. SKIP (COLUMNS 19 THROUGH 22). You specify line skipping before and/or following output of this record by using this field. (Like the previous field, this field is also composed of a Before Subfield (Columns 19 through 20) and an After Subfield (Columns 21 through 22) that let you determine when skipping occurs.) In both subfields, you can enter:

Columns 19 through 20 and Columns 21 through 22 Entry	Meaning
01 through 12	Skip to this line number (if L or 1 appears in the Carriage Control Type Field (Column 53) of the Control Record Specification). Skip to this Channel Number (if the Carriage Control Type Field in the Control Record Specification is blank).
13 through 99	Skip to this line number (Line 13 through 99) (if L or 1 appears in the Carriage Control Type Field of the Control Record Specification).
A0 through A9	Skip to this Line number (Lines 100 through 109) (if L or 1 appears in the Carriage Control Type Field of the Control Record Specification).
B0 through B2	Skip to this line number (Line 110 through 112) (if L or 1 appears in the Carriage Control Type Field of the Control Record Specification).
Blank	Skip no lines.

If you have specified **L** or **1** in the **Carriage Control Type Field** (Column 53) of the Control Record Specification, entries in this field indicate actual print line numbers (Lines 01 through 112); otherwise, these entries refer to logical channel numbers (01 through 12).

Normally, programmers request skipping when a new page is required or when many lines must be spanned. When skipping is based on channel numbers, you should use a Line Counter Specification (Section VI). If you do not include such a specification, RPG/3000 assumes a default value where each channel reference is equivalent to a line number that is five times the channel number. (Except for Channel 1 which defaults to line 6.) For instance, if your program requests skipping to channel 10, the printer will advance to Line 50 (5 x 10).

To request skipping, you must specify a one- or two-digit value in the **Before** or **After** Subfields, ending this entry in Column 20 or 22. If you request skipping to the line where the printer is currently positioned, the carriage does not move. If you request skipping beyond the overflow line (but not onto a new page), the overflow indicator turns on, and remains on until all overflow lines are printed. Do not request skipping to a line beyond the form-length line defined for this file on the Line Counter Specifications.

For default operations when the **Skip Field** is left blank, see Paragraph 9-10.

EXAMPLE

To print the header record at the top of the next page, enter 01 in the **Before** Subfield (Columns 19 through 20), (Figure 9-4, Item 3).

9-12. OUTPUT INDICATORS (COLUMNS 23 THROUGH 31). In this field, you can use one or more indicators to specify when the record or field described on this specification line should be output. (As you will see, this field is divided into several subfields, described below.)

O Output Specifications

Columns 24 through 25, 27 through 28, and 30 through 31 Entry	Meaning
01 through 99	Use any general indicator previously assigned to indicate the result of a calculation or to identify a record type or field.
F0 through F9	Use function key indicator previously set as a general indicator, through the RPG/VPLUS interface, or in response to a DSPLY or DSPLM operation.
KA through KN, KP through KY	Use command key indicator previously set as a general indicator or through the RPG screen interface.
L1 through L9	Use this control level indicator, previously assigned to the record or field being described.
L0	Use the Level 0 control level indicator, which is always set on.
LR	Use the last record indicator.
MR	Use the matching record indicator.
OA through OG, OV	Use this overflow indicator, normally previously assigned to this file.
1P	Use the first-page indicator.
H0 through H9	Use this halt indicator.
U1 through U8	Use this user indicator, normally set prior to program execution.
Blank	Do not use an output indicator.

When you use an indicator to condition an entire record (line of print), enter the indicator on the same specification line as the **Record Description Field** that applies to that record. When you use an indicator to condition a field within a record, place it on the same line as the **Field Description Field** for that field. For any record or field, you can assign up to three indicators on a single specification line; you enter them in Columns 24 through 25, 27 through 28, and/or 30 through 31. (By using **AND** lines, described in Paragraph 9-23, you can enter even more indicators for a record. By using **OR** lines, discussed in Paragraph 9-24, you can specify more than one set of conditions under which a record will be output.) If the conditions you specify are satisfied, the program will print the record or field described on the associated line. Do not use **AND** or **OR** lines to condition field output.

The indicators that you can specify are noted below. They are also described in Paragraph 7-10.

9-13. General Indicators. These are indicators previously set to show the result of a calculation or to identify an input record type or field. Use them as shown in the next example.

EXAMPLE

In Figure 9-5, the entire line consisting of the fields NAME, JOB, EXP, and PAY, is printed when general indicator 20 (Item 1) is on. Only the field NAME is printed as part of the record whenever general indicator 30 (Item 2) is on.

9-13A. Function Key Indicators. These are indicators previously set to show the result of a calculation, to identify an input record type or field, or to indicate the results of a **DSPLY** or **DSPLM** operation.

If your program contains an RPG/VPLUS interface "WORKSTN" file, these indicators are turned on to indicate a specific function key or the ENTER Key has been pressed, or a non-interactive "Event" has occurred, during a **READ** operation or during the input cycle.

9-13B. Command Key Indicators. These are indicators previously set to show the result of a calculation or to identify an input record type or field.

If your program contains an RPG screen interface "WORKSTNR" file, these indicators are turned on to indicate a specific command key has been pressed during a **READ** operation or during the input cycle.

9-14. Control Level Indicators. These indicators (L1 through L9) are set on when the program encounters a new control group in the input field. The Level 0 indicator, L0, is always on. You can use them to condition the output of data. If you specify a control level indicator on a record- description line for a total record (indicated by a T in Column 15), but do not use an overflow indicator, the record is written only after the last record of the control group is processed. If you enter a control level indicator for a detail record (D in Column 15), but do not use an overflow indicator, the record is written only after the first record of the new control group is processed. If you specify for any record both a control-level and an overflow indicator, and a control break occurs, the record is written after the overflow line has been sensed.

9-15. Last Record Indicator. You use this indicator (LR) to indicate when no more input records are to be read by your program. When the program reads the last record, it processes that record. Then, on the next program cycle, it turns the LR indicator on and prints all output records positively conditioned by that indicator (as long as other indicator conditions are satisfied).

9-16. Matching Record Indicator. You use this indicator to determine when two input fields match. When a match occurs, your program turns this indicator on and prints all output records positively conditioned by it (if other indicator conditions are satisfied).

9-17. Overflow Indicator. This indicator is turned on when your program output reaches the overflow line on a print file, as specified in the Line Counter Specifications. When this occurs, all output records positively conditioned by this indicator are printed (if other indicator conditions are satisfied).

Although eight overflow indicators (OA through OG, and OV) are available, you can specify only one for each print file. You should assign this indicator to the file in the File Description Specifications. If you do not assign an overflow indicator, and do not use an overflow indicator to condition an output record, the printer paper is advanced automatically to a new page when the end-of-form line is reached. If any specification line not conditioned by an overflow indicator requests skipping to a line on a new page, the overflow indicator is turned off before the printer advances.

Whenever you use an overflow indicator, you should also use a Line-Counter Specification; if no Line Counter Specification is present, a default is assumed with channel 1 equivalent to Line 6, channel 12 equivalent to Line 60, and each remaining channel equivalent to a line number that is five times the channel number. (See Section VI.)

You can use the overflow indicator in either AND or OR relationships, but you can associate only one such indicator for any one such relationship; this must be the indicator assigned to this file in the File Description Specifications.

HEWLETT PACKARD

RPG OUTPUT SPECIFICATIONS

Page 5 of 5

Programmer J. T. ROBERTS
Program Title REPT

Date 2/4/75

Punching Instructions

Graphic	
Punch	

Program Name REPT

Sequence Number	Form Type (D)	File Name	Type (H/D/E)	SS/F (H/L/2/E)	Space		Skip	Output Indicators						Field Name	Edit Code	Blank/Alter (B)	End Position (1-9999)	Packed/Binary (P/B/L/R 2/4)	Constant or Edit Word	Comments
					Before	Alter		And	And	Not	Not	Not	Not							
0		PRINTX	D	1						(1) 20			NAME			10				
0										(2) 30			JOB			20				
0													EXP			30				
0													PAY			40				
0				2						01	02	03								
0			(3) AND							04										
0			OR							N 05										
0													PAYTOT			10				

Figure 9-5. Using Output Indicators

When you use the overflow indicator in an AND relationship with another indicator that was previously used to identify an input record type, unpredictable results can occur if the record type identified by that indicator is not the type read when overflow occurs. In such a case, the indicator used to identify the input record type is not on, and thus all lines conditioned by both that indicator and the overflow indicator do not print. Therefore, if possible, use overflow and record-identifying indicators in OR relationships rather than AND relationships to condition record output.

You cannot use an overflow indicator to condition output of exception records (those specified by an E in Column 15), but you can use this indicator to condition fields within such records.

9-18. First-Page Indicator. This indicator (1P) is normally used to condition records that are to appear as headings on the first page of your report. You can also use it in conjunction with the overflow indicator to condition printing on every page; this information is usually comprised of headings or other constants that are defined in Columns 45 through 70 of the Output Specifications. Use the 1P indicator to control printing on pages other than the first only when you cannot use any other indicators for that purpose.

You can use the 1P indicator with all heading or detail records, but you cannot use it with total or exception records.

No other indicators can be specified for a record that is conditioned by the 1P indicator, except for user indicators (U1 through U8). However, if the 1P indicator is preceded by "N", other indicators preceded by "N" can also be specified.

Your program prints all lines conditioned by the 1P indicator even before it reads the first record fields from any input file. For this reason, you should not use it to condition output lines containing fields that are based on data obtained from input records, or to condition calculations.

You can use the 1P indicator to condition lines containing fields that are initialized prior to the reading of the first input record. These fields include the special fields PAGE, PAGE1 through PAGE7, UDATE, UDAY, UMONTH, UYEAR, any compile-time and pre-execution time array element(s), and any fields defined on input specifications for the User Data Structure (UDS).

9-19. Halt Indicators. These indicators (H0 through H9) are normally used to identify records that, when encountered, transmit a special message to the operator and halt your program at the end of the current detail cycle, or execute certain other pre-programmed or operator options (**Paragraph 3-19**). You can use them to condition the output of records and fields before the pre-programmed or operator option takes place.

9-20. User Indicators. These indicators (U1 through U8) can be set before program execution and are used to condition an entire file so that it can or cannot be used by the program. User indicators can also be used in the same way that general indicators are used as record, field, result, or conditioning indicators. The user indicators are set through the USWITCH source options discussed in Section III.

9-21. Error Conditions. You can use indicators to suppress or request output of records and/or fields when a program error occurs. Because you cannot use AND or OR Lines to condition fields (as noted in Paragraphs 9-23 and 9-24), you can use only three indicators for conditioning a field in the Output Specifications. However, you can use more than three indicators for this purpose by employing the SETON operation (**Paragraph 8-30**) in the Calculation Specifications in combination with the Output Specifications. For instance, suppose that various errors in your program could cause indicators 02, 04, 06, 08, and 10 to turn on. You wish to suppress output of a field in the event that these five indicators are on, but the Output Specifications allow you to specify only three indicators per field. So, in your Calculation Specifications, you condition a SETON operation to set indicator 12 on whenever indicators 02, 04, and 06 turn on. Then, in the Output Specifications, condition the output field so that it is suppressed if the indicators 12, 08, and 10 are on.

O Output Specifications

9-22. Not (Columns 23, 26, or 29). These fields allow you to condition output to take place if the indicator in the following field is off.

Columns 23, 26, or 29 Entry	Meaning
N	Output the data defined on this specification line only if the indicator in the next field is off.
Blank	Output the data defined on this line only if the indicator specified in the next field is on.

By entering N in any of the Not Fields preceding an indicator, you can specify that the record or field described on this line is output only if that indicator is off. When conditioning an entire record with such negative indicators, be sure to include at least one positive indicator (no preceding N) in the group — if you do enter all negative indicators for a heading or detail output record, it is output at the beginning of the program cycle after lines conditioned by the 1P indicator are written.

9-23. AND Lines. When you want to condition a record with more than three indicators, you can do so by using one or more AND Lines, as follows:

1. Specify the first three indicators in Columns 23 through 31 of the line containing the record description.
2. Write AND in Columns 14 through 16 of the NEXT Line, and enter up to three additional indicators in Columns 23 through 31 of this line; leave Columns 17 through 22 blank. This is your first AND line.
2. Continue writing as many AND lines as you need to specify all indicators desired.

All conditions specified by all indicators in the AND relationship must be satisfied before the record is output. For an AND relationship, stacker-select or fetch overflow must be specified in Column 16 of the first (file name) line only.

Note: You cannot use AND Lines to condition a FIELD.

9-24. OR Lines. When you want to condition a record with one or more alternative indicators, use one or more OR lines as follows:

1. Specify up to three indicators in Columns 23 through 31 of the line containing the record description. Continue this description with as many AND lines as you need.
2. Write OR in Columns 14 through 15 of the next line, and enter up to three additional indicators for the next set of conditions in Columns 23 through 31 of this line. This is your first OR line. (If different spacing or skipping is required, you can specify it in the Space or Skip Fields of the OR

Line; if you leave these fields blank, the same spacing and skipping previously specified is used. If the Fetch overflow feature is needed, you must specify it on each OR Line.)

3. Continue entering as many OR lines as you need.

When you specify an OR Line, this indicates that if either or both of the OR conditions are satisfied, the record will be output.

You can specify a maximum of 20 AND, OR, or mixed AND and OR lines in a single output operation.

Note: You cannot use OR lines to condition a **FIELD**.

EXAMPLE

An example of combined AND and OR lines appears in Figure 9-5. In this example (Item 3), the total record is output if either:

1. Indicators 01, 02, 03, and 04 are all on,

or

2. Indicator 05 is off.

9-25. Field Description Fields (Columns 23 through 70).

The fields in Columns 23 through 70 on the Output Specifications Sheet describe the data fields within the output records. Describe each field on a separate line on the specifications sheet, beginning at least one line below the line containing the record description (Columns 7 through 31). In each line containing a field description, leave Columns 7 through 22 blank.

You must always precede the field descriptions for a record with the appropriate record description — do not enter the field descriptions alone. You only need to describe those fields that your program actually outputs.

Note: The **Output Indicators Field** (Columns 23 through 31) is used for both record and field descriptions, and is described in Paragraph 9-12.

9-26. FIELD NAME (COLUMNS 32 THROUGH 37) (REQUIRED). Enter the name of the field, table, array or array element to be output.

O Output Specifications

Columns 32 through 37 Entry	Meaning	Columns 32 through 37 Entry	Meaning
Field name previously defined in the program. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.)	The name of the output field.	PAGE, or PAGE1 through PAGE7	A special RPG/3000 field that provides the number of the current page for printed output files.
Table name. (This may be a name of up to six characters, beginning with TAB. The remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.)	The name of the output table.	*PLACE	A special RPG/3000 field that repeats the output of fields defined previously within the record description.
Array name. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.)	The name of the output array.	*PRINT	A special RPG/3000 field that prints, along the top of a card, those characters punched in the card on output.
Array name, comma, and index. The array name is as defined above. The index is either a number or the name of a field (as defined above) that contains a number. The combination of array name, comma and index is limited to six characters	The name and index of the output array element.	UPDATE	A special RPG/3000 field that provides the current date.
		UDAY	A special RPG/3000 field that provides the current day of the month.
		UMONTH	A special RPG/3000 field that provides the current month.
		UYEAR	A special RPG/3000 field that provides the current year.
		*ERROR	A special RPG/3000 field used for run-time error codes.

In Output Specifications for all files but update files, you must name every data area that your program is to output. (For update files, you need *not* name a data area that is not changed on the input record.) Enter the name in the **Field Name Field**. Leave Columns 7 through 22 of this line blank. Do not, however, use this field if you specify a constant in Columns 45 through 70 of the Output Specifications, as discussed in Paragraph 9-36.

If you name one or more regular fields, tables, arrays, or array elements in the **Field Name Field**, be certain that these are areas you have previously defined in your File Description, File Extension, Input, or Calculation Specifications. (A regular field is one that is not pre-defined by RPG/3000 for a specific purpose.)

You can list these fields in any order on the specification sheet, since the sequence in which they are output is not determined by this field but by entries in the **End Position Field** (Columns 40 through 43). In most cases, however, programmers list them sequentially as a matter of personal convenience.

If a subsequent field overlaps an earlier one in the sequence, any data that is overlaid as a result is lost. In signed numeric fields, the plus or minus sign is output as part of the rightmost character; unless you request editing of the field with the **Edit Code Field** (Column 38), this signed digit appears as a letter. (For instance, -3 is printed as an L and +3 is printed as C.) A field that is unsigned when entered, and is not a result field, will be printed without a sign.

In addition to the regular fields you defined in your program, you can also use the special fields furnished by RPG/3000, described below.

PAGE

To request automatic numbering of the pages on a printed report, you can enter **PAGE** in the **Field Name Field**. This is the name of a special field provided for your convenience by RPG/3000. Since you can reference this field whenever you want the page number to appear, you can also use it to number individual records. However, it is most typically used in heading records.

At the start of your program, RPG/3000 pre-defines **PAGE** as a four-digit numeric field with no decimal positions, initialized to zero, and adds one to this field prior to printing the first page number. Then, each time your program outputs **PAGE**, it automatically increments this field by one prior to printing. When you use **PAGE** as initially pre-defined, you need not define it elsewhere in your program.

If you desire, you can re-define **PAGE** in your Input or Calculation Specifications so that it extends up to 15 positions long. When you do this, however, you must still restrict it to zero decimal positions.

Whether you use **PAGE** in its pre-defined form or re-define it, leading zeros are suppressed and no arithmetic sign appears on output unless you use an edit word or edit code to edit this field.

Although pagination normally starts with one, you can arrange for it to begin with another number by entering a value one less than that number in an input record field and naming that field **PAGE** on your Input Specifications, as described in Paragraph 7-33.

At any point in your program, you can re-start the page-numbering sequence. To do this, set the **PAGE Field** to zero prior to printing it, either by using the **Blank After** feature (Paragraph 9-30) or an output indicator set on. (Even though conditioned by an indicator, the **PAGE Field** is always printed; but, if the indicator is on, **PAGE** is set to zero and incremented by one before printing occurs.)

PAGE1 THROUGH PAGE7

You can use up to seven additional field names — PAGE1 through PAGE7 — for printing page numbers on different output files. If you use the same page-field name for more than one file, however, do so with caution to avoid confusion in your pagination system.

***PLACE**

With the special field *PLACE, you can repeat the same field or fields in several locations on an output record without the necessity of naming the individual fields and their locations each time. First, you describe the fields in the normal way. Then, enter *PLACE after the name of any field or list of fields to be output a second time on the same record. When output occurs, the fields named in the normal way in the record description are first written in the locations specified for them. Then, all those fields preceding the entry *PLACE are repeated, ending with the position specified by *PLACE.

The fields are output beginning with their first positions and with all following characters in the same relative positions. When you specify *PLACE for card output, the fields and constants above *PLACE are re-punched on each card. For each additional instance you want a field or group of fields repeated, you must enter *PLACE on a separate specification line; two consecutive *PLACE entries will repeat the initial fields four times. In each case, you must specify an end position for *PLACE, in the End Position Field (Columns 40 through 43); the value you enter must be at least twice the value of the highest end position previously specified for a field described before the *PLACE entry. (If you do not allow enough space for all fields and constants before *PLACE to be repeated, the repeated characters will overlay some or all of the previous characters on output.)

*PLACE fields can be conditioned with **Output Indicators** in columns 23 through 31.

EXAMPLE

Suppose that a program outputs four fields, with the names and contents shown below: (The character ␣ indicates a blank.)

Field Name	Field Contents
DATA1	␣␣AAA
DATA2	␣␣BBB
DATA3	␣␣CCC
DATA4	␣␣DDD

To print the contents of DATA1, DATA2, and DATA3 twice in the same output record (line), the programmer enters *PLACE after the list of those fields (Figure 9-6, Item 1). The contents of DATA4 will follow this output.

HEWLETT PACKARD RPG OUTPUT SPECIFICATIONS Page 5 of 5

Programmer B. ROBERTS Date 1/3/75

Program Title SPECIAL

Graphic	
Punch	

Program Name SPECIAL

Sequence Number	Form Type (O)	File Name	Type (IND/DEL)		Space	Skip	Output Indicators						Field Name	Edit Code	Blank After (B)	End Position (1-9999)	Packed Binary (P/B/L/R/Z/A)	Comments
			OR	AND			Before	After	Not	And	And	Not						
0		PRINTR	H	4												60	"SPECIAL REPORT"	
0			D	1														
0												DATA1				5		
0												DATA2				10		
0												DATA3				15		
0												*PLACE				30		
0												DATA4	B			35		

Edit Codes

1	A	J	commas & zero balances	X	Remove Plus Sign
2	B	K	commas	Y	Date Field Edit
3	C	L	zero balances	Z	Zero Suppress
4	D	M			

No Sign CR -

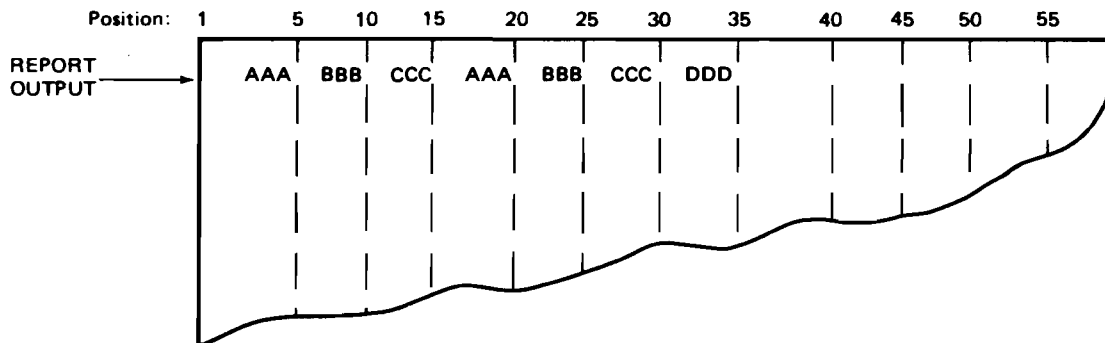


Figure 9-6. Using the *PLACE Field

O Output Specifications

***PRINT**

When punching data onto cards, you may also want to print this information along the top of the card to enable you to visually identify the card contents. You can do this by entering the special *PRINT Field name after the field or list of fields to be printed on the record being described. You can specify *PRINT only once within each record description, and use it only with card files. You can condition the *PRINT Field with indicators in the Output Indicators Field (Columns 23 through 31). Do not enter information in Columns 7 through 22 or 38 through 74 in the line specifying *PRINT.

The characters printed across the top of the card appear in the same positions as their corresponding punches on the card. To cause the printed characters to appear in different positions, use the card printing option described in Paragraph 9-33.

DATE FIELDS

You can use the following special fields to output dates to a printed report, punched cards, or other files. (The edited examples shown below are based on the Y edit code, described in Paragraph 9-28.)

Field Name	Contents	Unedited Example	Edited Example	Meaning
UUPDATE	Current Date	122575	12/25/75	December 25, 1975 (Domestic Format)
		251275	25/12/75	December 25, 1975 (United Kingdom Format)
		251275	25.12.75	December 25, 1975 (European Format)
UDAY	Current Day of Month	25	25	25th day.
UMONTH	Current Month	12	12	December
UYEAR	Current Year	75	75	1975

The UUPDATE Field can be specified as an unedited six-character field in either Domestic or United Kingdom/European Format, requested in the Control Record Specification as described in Paragraph 3-5. Alternatively by using the Y edit code described in Paragraph 9-28, you can request the insertion of slashes and decimal points for edited Domestic, United Kingdom, and European Formats, described above; the field in this case is eight characters long on output.

Your RPG/3000 program cannot change the contents of any of these special date fields.

***ERROR**

When the H0 indicator is set on during object program execution, special error codes are automatically transmitted to a special, one-character alphanumeric field named *ERROR. (The possible codes and conditions causing them appear in Appendix B.) Your program can check this field for information about errors that do not cause immediate termination. If you wish, for any reason, to enter data in this field, you must define it in the Field Name Field of the Input Specifications. When you do this, however, you must maintain the one-character field length. You can specify *ERROR in Factor 1, Factor 2, and the Result Field in Calculation Specifications. You can also reference this field with the RLABL operation in the Calculation Specifications when invoking an external subroutine, but the name is passed to the subroutine as ERROR (without the *). In the subroutine, you must always reference the field by the name ERROR.

9-27. EDIT CODE (COLUMN 38). This field allows you to edit the data your program outputs, permitting such operations as suppressing leading zeros and arithmetic signs, punctuating fields, and so forth.

Column 38 Entry	Meaning
X	Removes plus sign from positive field.
Y	Adds slashes or decimal points to numeric fields; normally used for date fields.
Z	Suppresses leading zeros.
1	Complex edit codes, used as shown below.
2	
3	
4	
A	
B	
C	
D	
J	
K	
L	
M	
Blank	Do not use an edit code.

You request editing by entering a one-character edit code in the Edit Code Field. RPG/3000 provides two types of edit codes: simple and complex. Either type can be applied to numeric fields only.

O Output Specifications

9-28. Simple Edit Codes. These edit codes remove the plus sign from positive fields, add slashes to fields, and suppress leading zeros, as follows:

Code	Function										
X	Removes plus sign from units position of field before field is written. This code does not, however, remove a minus sign nor suppress leading zeros. If you have entered an S in the Sign Process Field (Column 40) of the Control Record Specification, or have left that field blank, the plus sign will be suppressed even if you do not use the X edit code. No decimal points are printed.										
Y	Inserts slash marks into fields ranging from three to six digits long, with following results: <table><thead><tr><th>Number of Digits in Field</th><th>Format of Edited Output</th></tr></thead><tbody><tr><td>3</td><td>NN/N</td></tr><tr><td>4</td><td>NN/NN</td></tr><tr><td>5</td><td>NN/NN/N</td></tr><tr><td>6</td><td>NN/NN/NN</td></tr></tbody></table> <p>If the first digit is a zero, it is suppressed in the edited output. If you have entered I or J in the Inverted Print Field (Column 21) of the Control Record Specification, a period (decimal point) is used in place of the slash.</p>	Number of Digits in Field	Format of Edited Output	3	NN/N	4	NN/NN	5	NN/NN/N	6	NN/NN/NN
Number of Digits in Field	Format of Edited Output										
3	NN/N										
4	NN/NN										
5	NN/NN/N										
6	NN/NN/NN										
Z	Suppresses leading zeros. This edit code also ignores (does not print) any decimal point specified for the field, and removes any arithmetic sign from the units position.										

9-29. Complex Edit Codes. These edit codes insert various punctuation marks into the data fields specified. Their effect is summarized in Table 9-1. This table indicates, for instance, that a negative field, subject to the A edit code, would be output with the characters CR (for credit) appended; the same field, subject to the J edit code, would be output with a minus sign (for debit) appended.

All complex edit codes suppress leading zeros in the edited data, unless the letter J appears in the Inverted Print Field (Column 21) of the Control Record Specification; that entry always overrides any edit code that otherwise would suppress leading zeros. On the Control Record Specification, J is not an edit code, and should not be confused with the J edit code described below. On the Control Record Specification, J is used to request the output of data in European Format with leading zeros; in response to it, all zero balances with zero values to the left of the decimal comma are written with one leading zero (such as 0,00 or 0,04).

The decimal positions of the source field determine if and where a decimal point appears in the edited data. If decimal positions are specified for the source field and the zero balance is to be suppressed, the decimal point is printed as long as the value of the field is not zero. If J is not specified in the Inverted Print Field of the Control Record Specification, all digits to the left of a decimal point are zero-suppressed. If a zero balance is not to be printed, a field that contains only zeros will be output as blanks.

Table 9-1. Complex Edit Code Summary

FORMAT OF EDITED DATA	NONE	CR	—
Prints with commas, prints zero balance	1	A	J
Prints with commas, zero balance suppressed	2	B	K
Prints without commas, prints zero balance	3	C	L
Prints without commas, zero balance suppressed	4	D	M

When a zero balance is to be printed and the value of the field is zero, the output depends on the presence or absence of decimal points, as follows:

1. A decimal point followed by a number of zeros is printed, if the field contains decimal places; the number of zeros equals the number of decimal places in the field.

or

2. A zero appears in the units position of the field, if the field does not contain decimal places.

When you determine the end position of an edited field to be output, be sure to consider whatever characters (such as slashes or decimal points) are added by the edit operation. The amount of punctuation will increase the total length of the output field. If you do not consider these extra characters, the output fields generated may overlap.

You can use a floating dollar sign or asterisk protection with any complex edit code. Request the floating dollar sign by entering "\$" in the Constant/Edit Word Field (Columns 45 through 47), along with an edit code for the field to be edited in the Edit Code Field. On output, the dollar sign will appear to the left of the first significant (non-zero) digit. The dollar sign will not print on a zero balance when you use an edit code that suppresses zero balance. Request asterisk protection by entering "*" in the Constant/Edit Word Field, along with the appropriate edit code for the field; this prints an asterisk in the field for each zero that is suppressed, and is useful for protection of paychecks and other documents against unwanted post-printing entries. With asterisk-protection, a complete field of asterisks is printed on a zero-balance field.

When you have used an edit code in a request to print an entire array, two blanks precede each array element in the printed output.

If you specify J in the Inverted Print Field of the Control Record Specification, the use of commas and decimal points is reversed. Zero-suppression will always leave at least one zero to the left of the decimal point.

A complete list of all possible edit codes and their effect, shown by example, appears in Table 9-2. The floating dollar sign and asterisk protection can be used with all complex codes, but not with simple codes (X, Y, and Z). The Y edit code suppresses the leftmost zero only. In Table 9-2, the character \emptyset represents a blank. The right brace, $\}$, represents a zero with a negative-overpunch.

O Output Specifications

Table 9-2. Effects of Edit Codes

Edit Codes	Positive Number- Two Decimal Positions	Positive Number- No Decimal Positions	Negative Number- Three Decimal Positions#	Negative Number- No Decimal Positions#	Zero Balance- Two Decimal Positions	Zero Balance No Decimal Positions
Unedited	1234567	1234567	00012 }	00012 }	000000	000000
1	12,345.67	1,234,567	.120	120	.00	0
2	12,345.67	1,234,567	.120	120		
3	12345.67	1234567	.120	120	.00	0
4	12345.67	1234567	.120	120		
A	12,345.67 bb	1,234,567 bb	.120CR	120CR	.00 bb	0 bb
B	12,345.67 bb	1,234,567 bb	.120CR	120CR		
C	12345.67 bb	1234567 bb	.120CR	120CR	.00 bb	0 bb
D	12345.67 bb	1,234,567 bb	.120CR	120CR		
J	12,345.67 b	1,234,567 b	.120-	120-	.00 b	0 b
K	12,345.67 b	1,234,567 b	.120-	120-		
L	12345.67 b	1234567 b	.120-	120-	.00 b	0 b
M	12345.67 b	1234567 b	.120-	120-		
X	1234567	1234567	00012 }	00012 }	000000	000000
Y	(3)* 12/3	(4)* 12/34	(5)* 12/34/5	(6)* 12/34/56		
Z	1234567	1234567	120	120		

* The number in parentheses represents the number of field digits.

9-30 BLANK AFTER (COLUMN 39). You can use this field to re-set the contents of a numeric field to zero, or those of an alphanumeric field to blanks.

Column 39 Entry	Meaning
B	Re-set the contents of the field after output.
Blank	Do not re-set the contents of the field.

You typically use this entry when accumulating totals for several individual control groups. After finding and printing the totals for one group, you re-set the total field to zero. Then, when totals are accumulated for the next control group, the new totals will not be added to those of the previous group. The field is re-set immediately after it is printed. A numeric field will be re-set to zero; an alphanumeric field will be re-set to blanks.

You cannot use the Blank-After feature in descriptions of constants, Look-Ahead fields, or the special date fields UDATE, UDAY, UMONTH, or UYEAR.

When you specify the Blank-After feature, any indicators used to test the field for zeros or blanks (during processing of Input or Calculation Specifications) are set on to show that the field is now blank. Only the first indicator used in such a test is set on.

If you want to use a field more than once, as when both punching and printing, be sure to enter **B** on the last line for the field appearing in the Output Specifications. Otherwise, the field will be re-set to blanks before all output is complete.

If you use the Blank-After feature in the description of a table, the last entry found in the Table as a result of a LOKUP operation will be re-set. If no LOKUP has been performed for the table, the first entry in the table is re-set.

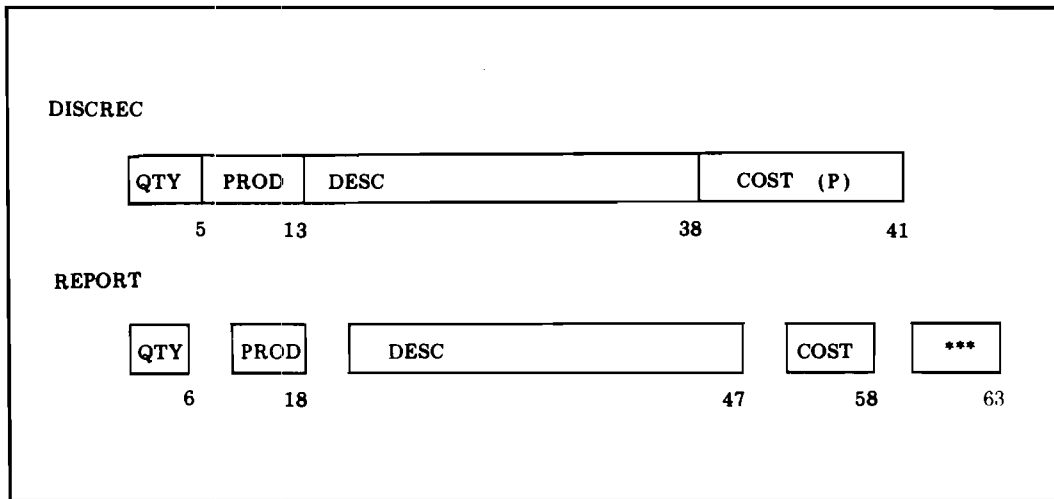
EXAMPLE

Request re-setting of the field named DATA4 to zero after its contents are printed, by entering **B** in Column 39 of the Output Specifications. (Figure 9-6.)

9-31. END POSITION (COLUMNS 40 THROUGH 43). In this field, you indicate where, on the output record, you wish the output field or constant to appear.

Columns 40 through 43 Entry	Meaning
1 through 9999	The number of the position where the rightmost character of the data is to appear on the output record.
Blank or ±nnn	Relative end position.

The actual record layouts for the files defined by these specifications would appear as follows:



9-31A. RPG Screen Interface Form Name Specification. (Column 42 through 43)

To indicate that this specification line contains the name of the form to be displayed on the WORKSTN device enter a "K" in column 42. In column 43 enter the character length of the form name.

The form name itself must appear on the same line as either a constant (Columns 45 through 70) or as the contents of a variable field name specified in columns 32 through 37).

EXAMPLE:

In the example, the form name "FM01" will be displayed when indicator 61 is on. When indicator 62 is on, whatever form name is present in the field named FORM will be displayed.

The form is titled 'RPG OUTPUT SPECIFICATIONS' and includes fields for Program Number, Date, Program Title, and Program Name. The main table has columns for Sequence Number, File Name, Type, and various indicators. Handwritten entries include 'SCREEN' in column 13, 'E' in column 14, '61' in column 21, 'DATE FLD1 FLD2' in column 32, 'K4' in column 42, '19' in column 43, and 'FM01' in column 45. A second entry shows 'FORM DATE FLD3 FLD4' in column 32, 'K8' in column 42, '16' in column 43, and '1' in column 45. A legend for Edit Codes is also present.

Figure 9-6A. RSI Form Name Specification

O Output Specifications

9-32. Output for Printed Reports, Punched Cards, or Disc. To indicate where a field or constant is to appear on these output media, simply enter, in the **End Position Field**, the number of the printing or punching position of the rightmost character of this data. This entry must end in Column 43. For printer output, the maximum value you can enter depends on the number of print positions available on the printer. For punched card output, the maximum value is 80. For disc output, the maximum value equals the size of the records written. The end position of this field minus the end position of the preceding field (or zero, if this is the first field on the record) determines the amount of space allowed for the data on the output record. Be careful, especially in the case of edited fields, to ensure that this space is sufficient for all characters.

When you specify the ***PLACE** entry for printed output in the **Field Name Field**, your entry in the **End Position Field** indicates the last character position of the last field of the group to be printed. Be certain to allow enough space for all fields covered by the ***PLACE** entry.

9-33. Output for Card Reader Punch. When you specify the ***PRINT** option in the **Field Name Field** (Columns 32 through 37) and direct output to a combined card reader punch, you can print along the top of the cards output the same information that is punched in them; in such cases, each character printed appears in the same position as the punches corresponding to that character. You can also arrange for the printed characters to appear in different positions than their corresponding punches, by following these directions:

1. Name the output field to be printed in the **Field Name Field** (Columns 32 through 37).
2. Enter an asterisk (*) in Column 40.
3. Specify the end position for the printed field in Columns 41 through 43; the maximum value allowed is 80. This entry must end in Column 43.

Enter all lines with an asterisk in Column 40 after those lines that specify punching only or ***PRINT** in each record description. After your program punches the fields required, it then prints those fields specified for printing across the top of the card in the position you specify.

If you use the **Blank-After** feature (specified by **B** in Column 39) for a field to be punched and printed, enter **B** on the last line that specifies printing for that field. (Because punching occurs before printing on a card, be careful not to blank the field after punching but before printing.) If the last line specifying printing for a field contains a ***PRINT** entry in the **Field Name Field**, enter **B** in Column 39 of the last punching specification line for that field. If an asterisk appears in Column 40 to request printing after punching, enter **B** on the last print specification line for that field.

EXAMPLE

The Output Specification in Figure 9-7 requests that the contents of the fields **ADATA**, **BDATA**, and **CDATA** are to be punched on the card file **CRDPNCH**, ending in Positions 10, 15, and 20, respectively, (Item 1). They also require that the contents of **ADATA** be printed on this file, ending in Column 25 (Item 2), as indicated by the asterisk in Column 40.

O Output Specifications

9-34. PACKED/BINARY (COLUMN 44). When a numeric field is to be written with a leading or trailing sign, or in packed decimal or binary format, without editing, an entry is required in this field.

Column 44 Entry	Meaning
P	Write the field in packed decimal format.
B	Write the field in binary format. A field of five digits or fewer is written as two bytes; a field of six to ten digits is written as four bytes.
L	Write the field in unpacked format with an arithmetic sign (plus or minus) preceding the field.
R	Write the field in unpacked format with an arithmetic sign following the field.
2	Write the field in binary format, two bytes long.
4	Write the field in binary format, four bytes long.
Blank	This is an unpacked numeric or alphanumeric field; a numeric field containing data to be edited; or a constant.

You can only make an entry in the Packed/Binary Field when describing a numeric field that is not to be edited. The L or R specify an alternate format for leading or trailing arithmetic signs that can be used in place of the standard format (where the sign appears over the low-order position in the field). Be sure that you allow enough room for any signs when you specify the entry in the Ending Position Field (Columns 40 through 43).

EXAMPLE

Specify that the output field MDATA is to contain packed decimal information by entering P in Column 44 (Figure 9-7, Item 3).

9-35. CONSTANT/EDIT WORD (COLUMNS 45 THROUGH 70). Use this field to specify either a constant to be output or an edit word for punctuating an output field.

Columns 45 through 70 Entry	Meaning
From one to 24 characters (any letters, digits, or special characters) surrounded by quotation marks.	Constant (if no field name appears on this line), or edit word (if field name appears on this line). If edit code is specified in the Edit Code Field (Column 38), the Constant/Edit Word Field may contain a \$ (for floating dollar sign) or * (for asterisk fill).
Blank	No constant is output or no edit word applies.

9-36. Constants. A constant is a group of characters, entered by a specification, that remains the same each time it is output. Frequently, constants appear as report titles, page or column headings, or record-identifying information. You will often use them in heading records.

Specify a constant in this way:

1. Leave the **Field Name Field** (Columns 32 through 37) blank on the line containing the constant.
2. Enter a quotation mark in Column 45 to denote the beginning of the constant.
3. Enter the constant itself. This may be any combination of letters, digits, or special characters from the ASCII character set. You can enter up to 24 characters on a single line, with the last character appearing in any column up through 69. (To output a constant containing more characters, you must actually specify the additional characters as another constant on the next line; use as many separate constant lines as you need for this purpose.)
4. Enter a quotation mark following the constant; this may appear in any position through Column 70, and identifies the end of the constant.
5. Because quotation marks are used to delimit the beginning and end of a constant, you must specify quotation marks within a constant in some other way; RPG/3000 requires that you use two adjacent quotation marks for this purpose. For example, to output **A PROGRAM NAMED "ALPHA"**, specify:

"A PROGRAM NAMED ""ALPHA""" .

6. For each constant line, define where the constant appears in the output record by an entry in the **End Position Field** (Columns 40 through 43).

O Output Specifications

EXAMPLE

Several constants appear in the Output Specifications shown in Figure 9-8. Notice the quotation marks within the first constant, and that the last two constants are output as a single group of characters, allowing the programmer to use a group that exceeds 24 characters in length.

9-37. Edit Words. By using edit words, you can obtain even greater flexibility in punctuating numeric fields than you can with edit codes. Edit words allow you to directly insert commas, decimal points, dollar signs, and other punctuation into numeric fields, output a minus sign or credit notation (CR) to denote negative data, provide asterisk protection, and print constants appended to the edited field. With edit words, you can explicitly lay out the output field you wish, character by character.

Specify an edit word in the Output Specifications on the same specification line as the field to be edited, as follows:

1. Enter the name of the field to be edited in the **Field Name Field** (Columns 32 through 37). This must be a **numeric** field.
2. Leave the **Edit Code Field** (Column 38) of this specification line blank.
3. Define where the edited field is to appear on the output record by an entry in the **End Position Field** (Columns 40 through 43).
4. Enter a quotation mark in Column 45 to denote the beginning of the edit word.
5. Enter the edit word itself, as defined below, beginning in Column 46. This may contain a maximum of 24 letters, digits, and/or special characters, with the last character appearing in any column up through 69.
6. Enter a quotation mark following the edit word; this may appear in any column up through 70, and identifies the end of the edit word.

The edit word itself consists of three portions; the body, status, and expansion portions, arranged in this order:

Column
46

Body	Status	Expansion
------	--------	-----------

Body Portion

The **Body Portion** is used to specify the punctuation to be superimposed upon the field to be edited. It contains both constant and replaceable characters. The constant characters, in general, comprise the punctuation marks to be inserted in the edited field. The replaceable characters are special characters that have unique meanings with respect to edit words. They will be replaced by digits transferred from the **source data field** (field being edited) to the **edited field** on the record output; they are used to provide space for these digits. As an example, an edit word body might contain two constants (a fixed dollar sign and a decimal point) and four replaceable characters (all blanks, indicated in this discussion by the symbol \emptyset):

\$ $\emptyset\emptyset$. $\emptyset\emptyset$



HEWLETT PACKARD Page 5 of 6

RPG OUTPUT SPECIFICATIONS

Programmer TOM HALL Date 3/7/75

Program Title PROSPECTS

Punching Instructions			
Graphic			
Punch			

35 76 77 78 79 80
Program Name <u>PROSP</u>

Sequence Number	Form Type (O)	File Name	Type (M/D/E)	Space Before	Space After	Skip	Output Indicators			Field Name	Edit Code	Blank Alter (B)	End Pkcs (1999)	Packed/Binary (P/B/L/R/2/4)	Constant or Edit Word																									Comments
							And	And							1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	
1		PRINT	H	3					1					50	"REPORT ON " "PROSPECTS" ""																									
2			H	5				2					9	"LAST NAME"																										
3													20	"FIRST NAME"																										
4													25	"AGE"																										
5													35	"OCCUPATION"																										
6													50	"PERSONNEL EVALUATION BAS"																										
7													72	"ED ON BACKGROUND CHECK"																										

Edit Codes

1 A J	commas & zero balances	X-	Remove Plus Sign
2 B K	commas	Y-	Date Field Edit
3 C L	zero balances	Z-	Zero Suppress
4 D M			

No Sign CR -

Figure 9-8. Specifying Constants

O Output Specifications

Suppose the source data field was four characters long and contained the value 1043. The resulting edited field would be printed as:

\$10.43

The body portion begins at the leftmost character of the edit word. The total number of replaceable characters it contains can be less than, equal to, or greater than, the number of digits in the source data field, but it must never exceed 15. The body portion ends with the rightmost replaceable character. The replaceable characters are:

Replaceable Character	Meaning
Blank	Replace this blank with a digit from the corresponding position of the source data field.
0	Replace leading zeros with blanks, up to and including this position.
*	Replace leading zeros with asterisks up to and including this position (asterisk protection).
\$	Place a floating dollar sign before the most significant digit in the field, (if this dollar sign appears to the immediate left of a zero).

EXAMPLE

Suppose the following edit word body were used:

00,000,000

If the source data field contained 21200450, the edited output would be:

21,200,450

Use an **ampersand (&)** to cause a blank to appear in the edited field; the ampersand in the body will be replaced by a blank in the edited field.

Normally, all leading zeros appearing before the first non-zero digit in the source data are automatically suppressed on output. But, you can use a **zero** as a replaceable character in the edit word to halt zero suppression at a particular character; place this zero in the rightmost position where you want suppression to cease. This zero will be replaced by the character from the corresponding position of the source data field, unless that character is also zero (in which case a blank will be substituted). Any zeros in the source data field that appear to the right of the character where suppression is to stop, will be printed. If any non-zero digit appears to the left of the character where suppression is to stop, suppression will instead stop with that non-zero digit. All suppressed zeros are replaced by blanks.

EXAMPLES

Consider the following edit word body:

##0###

If this edit word were applied to the data fields shown below, the edited fields indicated would result.

Source Data Field	Edited Field
0000123	##0123
1234567	1234567
0146000	##146000

If you desire leading zeros in an edited field, ensure that the edit word body contains at least one character more than the data field, and enter a zero in the leftmost position of the edit word body.

If there are more digit positions in an edit word than there are digits in the data field, leading zeros are actually added to the data field before editing takes place, and are then stripped out if so required by the edit word.

You can use an asterisk for halting zero suppression as well as for asterisk protection. To halt zero-suppression, place the asterisk in the rightmost position of the edit word body where zero-suppression is to stop. Each zero suppressed is replaced by an asterisk. If an asterisk precedes a zero in an edit word, it is interpreted as a request for asterisk protection; in this case, the zero is printed as a constant.

EXAMPLE

The following edit word body is used for asterisk protection.

###*#.##

When a field containing 0002388 is edited, it appears on output as:

*****23.88**

You can use **floating dollar signs** in edit words. To do this, enter a dollar sign immediately to the left of a character that halts zero-suppression. As a result, a dollar sign will be inserted in the first position to the left of the first significant digit in the output field. You cannot use the floating dollar sign with asterisk protection, however. The sum of the number of blanks plus the character that halts zero suppression in the edit word body must be equal to or greater than the number of characters in the data field to be edited. (Do not count the dollar sign in the edit word body in making this calculation.)

O Output Specifications

EXAMPLE

The floating dollar sign appears in this edit word body:

#####\$0.##

When a data field containing 00012345 is edited, the following appears in the output record:

###\$123.45

A dollar sign in the leftmost position of an edit word is a **fixed dollar sign** — it always appears in the same location in printed output, as the first character in the field.

When you use decimal points or commas in an edit word body, they appear in the same relative positions when they are placed in the edited field unless they are placed to the left of significant digits; in that case, they are replaced by blanks or (if you request asterisk protection) by asterisks.

To represent a **quotation mark** in an edit word body, use two adjacent quotation marks.

If any characters other than those discussed above appear to the right of significant digits in the data field, they are printed. But, if they appear to the left of the high-order significant digit, they are replaced by blanks or asterisks (in the case of asterisk protection).

Constants that follow a significant digit (1 through 9) or a zero-suppression stop character (* or 0) and immediately precede a -, or CR in the status portion (discussed below) are always printed. A field of negative zeros is edited as a field of positive zeros, and the status portion is not printed in that case.

Status Portion

The status portion of an edit word is used to display the positive or negative status of the data in the edited field. This portion is optional. To use it, enter either CR (for credit) or - (for minus) next to the rightmost character of the edit word body. If the data in the edited field is negative, then the symbol you specify will print. Otherwise, it will not. If the edit word body is not followed by CR or -, the edit word has no status portion. To include a blank in the status portion, use an ampersand; the ampersand will be replaced by a blank in edited output.

EXAMPLE

The following edit word contains the CR notation in its status portion:

#####.##CR

If this edit word is used to edit the value -12333, the output record will contain:

##123.33CR

Expansion Portion

The expansion portion is used to append a constant to the preceding portions of the edit word. It begins after the last character of the status portion (or body portion, if no status portion is included), and ends with the last character in the edit word. It can include any character but a blank; to cause blanks to appear in the expansion portion in the edited output, enter ampersands; these will be replaced by blanks in the edited output. The expansion portion is optional; but when it is included, it is always printed.

EXAMPLES

The following is a complete edit word, with body, status, and expansion portions:

\$###.##CR&NET

When it is used to edit a field containing - 31262, the following output appears:

\$312.62CR#NET

Many additional examples showing the effect of edit words upon data fields appear in Figure 9-9. In this chart, the first column shows the complete edit word as it would appear in Columns 45 through 70 of the Output Specifications. The next column shows the source data in the fields being edited; a plus or minus sign following the data indicates whether it is positive or negative when used with the edit word. The last column shows how the edited data appears in the output record.

9-38. Comments (Columns 71 through 74)

You can use the spaces in Columns 71 through 74 for comments of any kind.

9-39. Program Name (Columns 75 through 80)

This field contains the program name or any other information, as discussed in Paragraph 2-6.

O Output Specifications

45	EDIT WORD										70	SOURCE DATA	APPEARS IN OUTPUT RECORD AS:	
"		,											0000000005 -	bbbbbbbbbb.05bCR
"		,											00000000	bbbbbbbbbb0bONbHAND
"		,											0000000005 +	bbbbbbbbbb\$0.05b*
"		,											0034567890 -	bbb\$345,678.90CR**
"	\$,											0000000000	\$bbbbbbbbbb.00
"	\$	&	,										1234567890 -	\$b12,345,678.90b-bGROSS
"		,											0000000123 -	bbbbbbbbbb1.23-
"		,											0000000000	bbbbbbbbbbbbbb
"		,											0000135792 -	*****1,357.92b -
													0000135678	0000135678
													0000135678 +	0000135678
													0000135678 -	000013567Q
"													0000000000	bbbbbbbbbb
"													0000135678 +	bbb135678
"													0000135678 -	bbb135678
"													0000135678 -	bbb135678
"													0000135678 +	b000135678
"													0000135678 +	bbb135678bbbNET
"													0000135678 -	bbb135678bCRbNET
"													0000135678 -	bbb135678b-bbNET
"													0000135678	bbb135678bNETbbb
"													0000135678 -	bbb135678bNETbCR
"													0000135678	bbb135678bPROFIT
"	\$												0000135678 +	\$bbb135678bbbNET
"	\$												0000135678 -	\$bbb135678b-bNET
"	\$												0000135678	\$b000135678bNET
"													0000135678 -	bbb135678bCR
"													1234567809 -	\$1234567809bCR
"													0000000000 -	bbbbbbbbbbbWTOTAL
"													0000000000 -	bbbbbbbbbbbWTOTAL
"	\$												0000000000 -	\$bbbbbb00bbbGROSS
"													0000000000 -	bbbbbb135678bGROSS
"													0000000000 -	bbbbbb135678bGROSS
"													0000000000 -	bbbbbb135678bGROSS
"													0000000000 -	*****b
"													0000000000 -	*****00b
"	*												0000135678 -	*000135678
"	*												1234567890 +	1234567890
"													0000135678 -	****135678
"		,											0000135678 -	bbb1,356.78bCRbNET

Figure 9-9. Examples of Edit Words

O Output Specifications

9-40. DEFAULT SUMMARY

If you leave the optional fields of the Output Specifications blank, the default specifications shown in Table 9-3 apply:

Table 9-3. Output Specifications Default Values

COLUMNS	FIELD	DEFAULT SPECIFICATIONS
1 through 5	Sequence number.	No sequence number applies.
16 through 18	Record Addition/Deletion.	For sequential or KSAM output files, write new records to beginning of the file. For RSAM or IMAGE output files, insert new records in the file. For update files, update the current record.
16	Fetch Overflow.	Do not fetch overflow.
16	Stacker Select.	Select Stacker No. 1.
17 through 22	Space and Skip.	Space one line after printing each record.
24 through 25, 27 through 28, and 30 through 31	Output Indicators.	Do not assign an indicator.
23, 26, or 29	Not.	Output the data defined on this line only if the indicator specified in the next field is on.
38	Edit Code.	Do not use an edit code.
39	Blank After.	Do not re-set the contents of the field to blanks.
44	Packed/Binary	This is an unpacked numeric or alphanumeric field; a numeric field containing data to be edited; or a constant.
45 through 70	Constant/Edit Word.	No constant is output or no edit word applies.
71 through 74	Comments.	No comments are to appear.
75 through 80	Program Name.	None.

9-41. PRINTER SPACING CHART

Typically, when you code an RPG/3000 source program that produces a report or other printed material, you will want to keep in mind a clear idea of what your program output will look like. To help you to determine the exact appearance of this output, you can lay out your report on a special form called a **Printer Spacing Chart**. You can do this before you begin any of the coding, wait until you are ready to write the Output Specifications, or ignore this step altogether, according to your own preference. This chart provides a grid that allows you to plan exactly where each character will appear in each output record (line) produced by the program, in terms of vertical and horizontal alignment on the report. It also contains an area that helps you visualize the effect of references to carriage-control tape channels upon your output. (Punches in this tape are simulated by using Line Counter Specifications to equate channel numbers to line numbers, as noted in Section VI.)

EXAMPLE

To indicate that you plan to permit skipping to Lines 4, 6, and 8 by simulated references to Tape Channels 1, 2, and 3, respectively, mark these channels as indicated in Figure 9-10 (Items 1, 2, and 3).

The Printer Spacing Chart allows for 60 lines of output on each page, with each line containing 134 print positions. On this chart, you can show heading, total, and detail records (lines) as they will appear on the printed page output by your program. To indicate constants on the chart, write each constant exactly as it will appear on output. To indicate fields, mark off the variable content of each field by using characters such as X's, Y's, Z's, or special symbols, and then write the field name in parentheses under these characters. (You can actually use any method you wish in describing a field — the method presented here is one of those most commonly employed among RPG programmers.) As an example, you could use X's to indicate the contents of alphanumeric fields, Z's to show the contents of numeric fields, and 0's to indicate where zero suppression is to cease in edited numeric fields. You might insert periods and commas into these fields just as they will appear on the printed output. You can also indicate special editing symbols, such as CR (for credit). Be sure that each field description contains the maximum number of characters that can appear in the field, and ends in the same column that you will specify in the **End Position Field** on the Output Specification Sheet.

HEADING LINES. Heading lines contain information written across the top of the printed page to describe the data that appears below. Generally, they contain constants, except for the date and page numbers obtained from special RPG fields. Denote each heading line by writing H in the **REC TYPE** Column on the Printer Spacing Chart.

EXAMPLE

The chart in Figure 9-10 contains three heading lines. The first of these contains two constants -- **DATE** and **PAGE** (Items 4 and 5). It also contains two fields — **UDATE** and **PAGE1** (Items 6 and 7), whose names refer to special RPG/3000 fields. The first of these is eight characters long and ends in Column 22.

DETAIL LINES. Detail lines may appear many times on printed output, but we need describe them only once on the Printer Spacing Chart. They contain only fields, each of which you must show and name on the chart. Denote each detail line by placing a **D** in the **REC TYPE** Column.

EXAMPLE

The Printer Spacing Chart in Figure 9-10 contains one detail line, (Item 8) with four fields — **DEPNO**, **EMPNAM**, **INT**, and **IDNO**. When the report is printed, these fields will contain, respectively, the department number, last name, initials, and identification number of each employee on the roster.

TOTAL LINES. Like detail lines, total lines may appear several times on a printed report but need be described only once. They can contain both constants and data fields, and show information typically derived by adding the contents of the fields in detail lines. Indicate them by entering **T** in the **REC TYPE** Column. For lines output as a result of control breaks, you can use designations such as **TL1** (for a level 1 control break), **TLR** (for a last record break), and so forth.

EXAMPLE

The chart in Figure 9-10 shows two total lines (Items 9 and 10). The first contains the constant **TOTAL EMP IN DEPT =** and the numeric field named **TOTNO**; the second contains the constant **TOTAL EMP IN DIV =** and the numeric field **DTOTNO**. The first is output as a result of a level 1 control break; the second, when the last record indicator is on.



To illustrate some of the RPG/3000 features that are commonly used in various programming applications, various examples appear in this section.

10-1. A SIMPLE RPG/3000 PROGRAM

The following example makes use of all RPG/3000 Specification Sheets, a pre-execution time table, and the following conditioning indicators: 15, 1P, OF, L1, L2, and LR. It outputs the heading, detail, and total records typically produced by most RPG programs.

10-2. Analyzing the Application

Suppose that a programmer wants to produce a weekly sales report for a book store at a local community college. This report is to be produced by a program that is run once each week at the college's data center. The program will read its input from punched cards that show, for each book:

- Stock number of the book.
- General subject area (corresponding to the college division under which the subject is taught).
- Specific subject (corresponding to the college department in which the subject is taught).
- Author
- Title
- Edition
- Price per copy.
- Number of copies shipped during past week.

The program also calculates the following totals for the past week:

- Income from all copies sold of a particular book.
- Income from all books on a particular subject (under a particular department).
- Income from all books in a particular subject area (under a particular division).
- Income from all books sold.

The program will also update a pre-execution time table that contains the total number of copies of each book on hand at any given moment. (This table will be used later by another program that prints an inventory report for the book store).

Programming Applications

The program will print a report that shows:

- The title TEXTBOOK SALES.
- The current date and page number.
- Headings for the following items, followed by the items themselves (for each book):
 - Stock number
 - Author
 - Title
 - Edition
 - Number sold during week
 - Price per copy
 - Income from sales of this book during past week.
- The following totals for the past week:
 - Income from all books under the particular department.
 - Income from all books under the particular division.
 - Income from all books sold by the book store.

10-3. Writing the Program

To handle this application, the programmer might write the RPG/3000 program shown on the specification sheets in Figures 10-1 through 10-7. (The numbered items on these sheets illustrate significant points and are keyed to the discussion appearing prior to the sheets.)

10-4. CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS (FIGURE 10-1). In the Control Record Specification, the programmer:

- ① Requests output of a Cross-Reference Listing.
- ② Declares that the skip requests (that will appear in the Output Specifications) refer to actual line numbers and not carriage-control tape channel numbers.
- ③ Requests the compiler to check the source program records for proper sequence.

In the File Description Specifications, the programmer specifies:

- ④ Two input files, both containing 80-character, fixed-length records. The primary file is named CARDS and is read from punched cards; the other input file is named TABFILE and contains a table permanently residing on disc. Because he must also use a File Extension Specification for further describing this file, the programmer enters E in Column 39.
- ⑤ One output file named REPORT, to contain 132-character, fixed-length records directed to a line printer. This file is associated with a OF overflow indicator. Because the programmer must further describe the file in a Line Counter Specification, he enters L in Column 39.

Programming Applications

10-5. FILE EXTENSION AND LINE COUNTER SPECIFICATIONS (FIGURE 10-2). With the File Extension Specifications, the programmer:

1. Specifies that the input file TABFILE contains two alternating tables, TABA and TABB, each containing a maximum of 160 entries; each entry is five characters long; each record contains eight entries for each table. When it reads the tables, the program is to check them for proper ascending sequence.

With the Line Counter Specifications, the programmer:

2. Specifies a form length of 66 lines and equates the overflow line to Line 55.

10-6. INPUT SPECIFICATIONS (FIGURE 10-3). In the Input Specifications, the programmer needs to describe only the input file named CARDS. (He has already furnished the descriptive information for the other input file, TABFILE, in the File Extension Specifications; that file already resides in the system, entered through the HP 3000 FCOPY subsystem.) The programmer:

1. Includes a record description specifying one type of record only; thus, he need assign no group sequence number. Each time a record is read, the general indicator 15 is turned on.
2. Describes a field for each of the input items discussed under Paragraph 10-2. When the contents of the GSAREA field (showing general subject area (division)) changes, indicators L2 and L1 are turned on; when the contents of the SUBJ field (showing the specific subject (department)) changes, the L1 indicator is turned on. (This illustrates the use of control-level fields.)

The data file described by these Input Specifications is shown in Figure 10-4.

10-7. CALCULATION SPECIFICATIONS (FIGURE 10-5). In the Calculation Specifications, the programmer requests several operations:

1. Multiplying the price per copy of each book (PPCOPY) by the number of copies sold (NUMSHP) to obtain the income received for all copies sold (BKSL).
2. Adding the income received for all copies sold (BKSL) to obtain totals for sales per subject/department (SUBSL), per general area/division (GARSL), and for the entire college (TOTSL).
3. Looking up the stock number of each book (in TABA), finding the corresponding number of copies on hand at the end of the previous week (in TABB), and setting General Indicator 10 on if this search is successful.
4. If the table search is successful (Indicator 10 is on), subtracting the number sold (NUMSHP) during the past week from the number previously on hand and entering the new amount in TABB.

HEWLETT  PACKARD

RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS

Page 2 of 6

Programmer B. FRANKLIN Date 1/2/75
 Program Title TEXTBOOK SALES

Punching Instructions	
Graphic	
Punch	

Program Name TEXTSL

File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec Seq	Chaining File Code (C1 CB)	From Filename	To Filename	Table Array or Routine Name	Entries Per Record (1 999)	Entries Per Table Array (1 9999)	Entry Length (1 256)	Data Format (P B L R)	Decimal Positions (0 9)	Table Array Sequence (A D)	Table Array Name	Entry Length (1 256)	Data Format (P B L R)	Decimal Positions (0 9)	Table Array Sequence (A D)	Comments
0050	E			TABFILE		TAB A	0	160	5				ATABB	5				ALTERNATING TABLE
	E																	
	E																	
	E																	
	E																	
	E																	
	E																	
	E																	
	E																	

Line Counter Specifications

Sequence Number	Form Type (L)	Filename	1		7		3		4		5		6		7		8		9		10		11		12		
			Line Number	Channel Number/OL/FL	Line Number	Channel Number/OL/FL	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number
0060	L	REPORT	66	FL	550	L																					
	L																										
	L																										
	L																										

Figure 10-2. Sample Program File Extension and Line Counter Specifications Sheet

HEWLETT PACKARD

Programmer B. FRANKLIN Date 1/2/75

Program Title TEXTBOOK SALES

RPG INPUT SPECIFICATIONS

Punching Instructions

Graphic				
Punch				

Page 3 of 6

Program Name TEXTSL

Sequence Number	Form Type (1)	File Name	Group Sequence	Number of Records (1.N)	Record Indicator	Record Identification Codes									Field Position				Control Level (L1, L9)	Matching or Chaining Fields (M1, M9, C1, C9)	Field Record Relation	Field Indicators															
						Position (1 9999)	Not (N)	Position (C/Z/D) Character	Position (1 9999)	Not (N)	Position (C/Z/D) Character	Position (1 9999)	Not (N)	Position (C/Z/D) Character	From (1 9999)	To (1 9999)	Decimal Positions (D9)	Field Name				Plus	Minus	Zero or Blank													
0070	I	CARDS	AA	15												1	5	STOKNO																			
0080	I															6	5	GSARFAL																			
0090	I															1	6	25	SUBJ																		
0100	I															2	6	35	AUTH																		
0300	I															3	6	55	TITLE																		
0400	I															5	6	58	EDITN																		
0500	I															5	9	63	ZPPCOPY																		
0600	I															6	4	68	NUMSHP																		


Figure 10-3. Sample Program Input Specification Sheet

00001	BUSINESS	ACCOUNTING	WAKEFIELD	PRINCIPLES OF ACCT	1	00857	00010
-------	----------	------------	-----------	--------------------	---	-------	-------

1 5 6 15 16 25 26 35 36 55 56 58 59 63 64 68

STOKNO GSAREA SUBJ AUTH TITLE EDITN PPCOPY NUMSHP

Figure 10-4. Sample Program Input Data Deck



RPG CALCULATION SPECIFICATIONS

Page 4 of 6

Programmer B. FRANKLIN Date 1/2/75

Program Title TEXTBOOK SALES

Punching Instructions

Graphic				
Punch				

Program Name TEXTSL

Sequence Number	Form Type (C)	Control Level (L/O-LB/LR/SR/AN/OR)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-256)	Decimal Positions (D)	Half Adjust (H)	Resulting Indicators			Comments
			Not	And									Arithmetic	Plus Minus Zero	Compare	
				Not	Not											
0808	C					PPCOPY	MULT	NUMSHP	BKSL	72					SALES/BOOK	1
0908	C					BKSL	ADD	SUBSL	SUBSL	92					SALES/SUBJECT	2
1008	C					BKSL	ADD	GARSL	GARSL	102					SALES/GEN AREA	3
1108	C					BKSL	ADD	TOTSL	TOTSL	122					TOTAL SALES	4
1208	C					STOKNO	LOOKUP	TABA	TABB				10			
1308	C		10			TABB	SUB	NUMSHP	TABB						UPDATE INV TAB	
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															
	C															

Figure 10-5. Sample Program Calculation Specifications Sheet

10-8. PRINTER SPACING CHART (FIGURE 10-6) AND OUTPUT SPECIFICATIONS (FIGURE 10-7).

Next, the programmer plans the general layout of the sales report by using a Printer Spacing Chart. Then, using this chart as an aid, he writes the Output Specifications directing the records comprising the report to the printer file named REPORT:

1. The first record is a heading record, containing the heading TEXTBOOK SALES, plus the current date and page number (preceded by the word PAGE). Because it is conditioned by the 1P indicator, the record appears on the first page only.
2. The second record is a heading record consisting of the word PAGE followed by the current page number. Whenever the overflow line is encountered, the printer skips to Line 6 of a new page (as indicated in Column 20) and prints this record; thus, it appears at the top of each page other than the first.
3. The third record is also a heading record that provides headings to appear over the data fields output. This record appears as the third line on the first page (as conditioned by the 1P indicator) or as the second record on each succeeding page (as conditioned by the OF indicator).
4. The fourth record described is a detail record containing the output fields described in Paragraph 10-2. Each time an input record is read (indicated when the General Indicator 15 is set on), one detail record is printed.
5. The next three records are total records showing sales totals for each department (output when the L1 indicator is on), for each division (output when the L2 indicator is on), and for the entire college (output when the last record from CARDS is read). In each record, the totals will be preceded by a floating dollar sign and followed by one, two, or three asterisks (as directed in the Constant or Edit Word Field); these asterisks denote totals for departments (*), divisions (**), and the entire college (***) .

10-9. Compiling, Preparing, and Executing the Program

Next, the programmer submits his program for compilation, preparation, and execution in a batch job. The complete job output appears in Figure 10-8, with significant items keyed to the discussion below. Many of these items pertain to the MPE/3000 Operating System and are described more fully in Section XI.

1. The MPE/3000 :JOB command that initiates the batch job.
2. The MPE/3000 log-on message output by the operating system, showing job number, present date and time, software part number for the operating system, and welcome record.
3. Two MPE/3000 :FILE commands, used to specify that the input file CARDS will be read from the standard input device (a card reader) and that the output file REPORT will be printed on the standard listing device (a line printer).
4. The MPE/3000 command :RPGGO that compiles, prepares, and executes the RPG/3000 program.

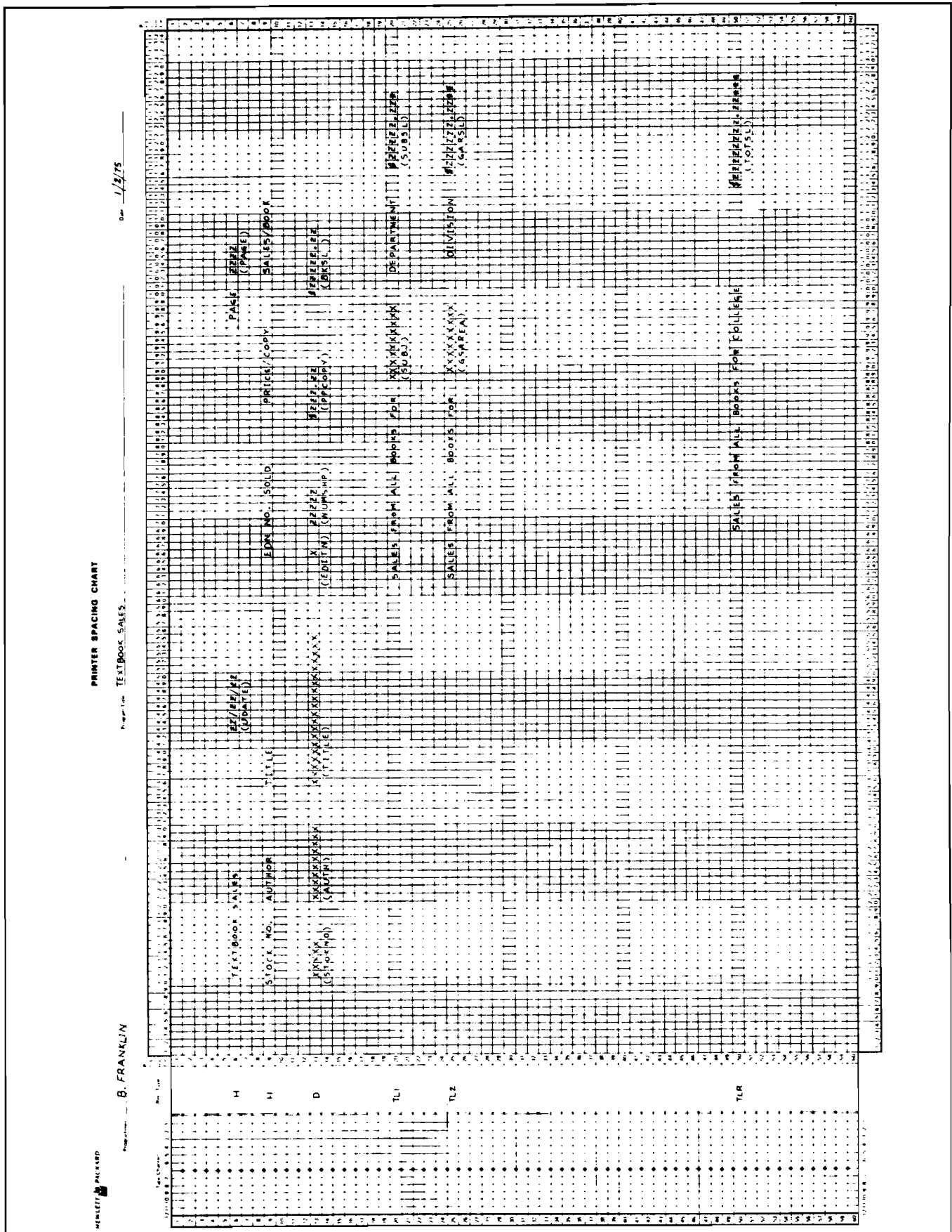



Figure 10-6. Sample Program Printer Spacing Chart

HEWLETT  PACKARD

RPG OUTPUT SPECIFICATIONS

Page 5 of 6

Programmer B. FRANKLIN Date 1/2/75

Program Title TEXTBOOK SALES

Punching Instructions

Graphic					
Punch					

Program Name TEXTSL

Sequence Number	Form Type (O)	File Name	Type (H/D/E)		Space		Skip		Output Indicators						Field Name	Edit Code	Blank Alter (B)	End Position (1-9999)	Packed/Binary (P/B/L/R/Z)	Constant or Edit Word	Comments
			Before	Alter	Before	Alter	And	And	Not	Not	Not	Not	Not								
1040	O	REPORT	H																		
1050	O																24	"TEXTBOOK SALES"			
1060	O																50				1
1070	O																100	"PAGE"			
1080	O																106				
1090	O		H				6														
1100	O																				
1110	O																				
1120	O		H				33														
1130	O		OR																		
1140	O																18	"STOCK NO."			
1150	O																26	"AUTHOR"			
1160	O																40	"TITLE"			
1170	O																68	"EDN"			
1180	O																77	"NO. SOLD"			
1190	O																95	"PRICE/COPY"			
1200	O																112	"SALES/BOOK"			
1210	O		D																		
1220	O																				
1230	O																				
1240	O																				
1250	O																				
1260	O																				
1270	O																				
1280	O																				

Figure 10-7. Sample Program Output Specifications Sheet



RPG OUTPUT SPECIFICATIONS

Programmer B. FRANKLIN Date 1/2/75
 Program Title TEXT BOOK SALES

Punching Instructions

Graphic	
Punch	

Program Name TEXTSL

Sequence Number	Form Type (O)	File Name	Type (H/D/O/E)		Space		Skip		Output Indicators			Field Name	Edit Code	Blank Alter (B)	End Position (1-9999)	Packed Binary (B B L R Z 4)	Constant or Edit Word	Comments
			A	D	O	E	Before	Alter	Not	And	And							
1290	0														86		"SALES FROM ALL BOOKS FOR"	
1300	0														98			
1310	0														112		"DEPARTMENT"	
1320	0														125		"\$"	
1330	0														126		"*"	
1335	0														86		"SALES FROM ALL BOOKS FOR"	
1340	0														98			(5)
1350	0														12		"DIVISION"	
1360	0														125		"\$"	
1370	0														127		"**"	
1380	0														92		"SALES FROM ALL BOOKS FOR"	
1385	0														100		"COLLEGE"	
1390	0														125		"\$"	
1400	0														128		"***"	
1420	0																	
1425	0																	

Figure 10-7. Sample Program Output Specifications Sheet (Continued)

```

1
IJOB LESLIE,RPQ,LANG, RQGIMA
PRI= CS: INPRI= B I TIME= ?
JOB NUMBER = #J128
THU, JAN 2, 1975, 4:39 PM
HP3200C,F0,18
} 2
** H=Y **
HAPPY NEW YEAR, ACCT, MINISOFT WILL BE PURGED
FROM THE SYSTEMS ON FRI, 1/3/75, PLEASE TRANSFER
YOUR FILES.
IFILE CARDS=SSDIN } 3
IFILE REPORT=SSDLIST
IRPGG } 4

```

```

PAGE 0001 HEWLETT PACKARD 32104A,P2,03 RPQ/3000 THU, JAN 2, 1975, 4:39 PM } 5
0001 0010 H XLS TEXTSL
0002 0020 FCARDS IP F 80 CARD
0003 0030 FTABFILE IT F 80 FDISC
0004 0040 FREPORT O F 132 OF LLP
0005 0050 E TABFILE TABA B 160 5 ATABB 5 0 ALTERNATING TABLE
0006 0060 LREPORT 66F. 550L
0007 0070 ICARDS AA 15
0008 0080 I 1 5 STOKNO
0009 0090 I 6 15 OSAREAL2
0010 0100 I 16 25 SUBJ L1
0011 0200 I 26 35 AUTH
0012 0300 I 36 55 TITLE
0013 0400 I 56 58 EDITN
0014 0500 I 59 632PPCOPY
0015 0600 I 64 680NUMSHP
0016 0800 C PPCOPY MULT NUMSHP RKSL 72 SALES/BOOK
0017 0900 C BKSL ADD SUBSL SURSL 92 SALES/SUBJECT
0018 1000 C BKSL ADD GARSL GARSL 102 SALES/GEN AREA
0019 1010 C BKSL ADD TOTSL TOTSL 122 TOTAL SALES
0020 1020 C STOKNO LOKUPTABA TABB 10
0021 1030 C 10 TABB SUB NUMSHP TABB UPDATE INV TAB
0022 1040 OREPORT H 1P
0023 1050 O UDATE Y 24 "TEXTBOOK SALES"
0024 1060 O PAGE 50
0025 1070 O PAGE 100 "PAGE"
0026 1080 O PAGE 106
0027 1090 O X 6 OF PAGE 100 "PAGE"
0028 1100 O PAGE 106
0029 1110 O
0030 1120 O H 33 1P
0031 1130 O OP OF
0032 1140 O 18 "STOCK NO,"
0033 1150 O 26 "AUTHOR"
0034 1160 O 40 "TITLE"
0035 1170 O 68 "EDCN"
0036 1180 O 77 "NO. SOLD"
0037 1190 O 95 "PRICE/COPY"
0038 1200 O 112 "SALES/BOOK"
0039 1210 O O 1 15

```

Figure 10-8. Batch Job Output

Programming Applications

PAGE 0002 TEXTS.

0040	1220	C				STOKNO	15	
0041	1230	O				AUTH	30	
0042	1240	O				TITLE	55	
0043	1250	O				EDITN	66	
0044	1260	O				NUMSHP	74	
0045	1270	O				PPCOPY1	90	"S"
0046	1280	O				BKSL 1	108	"S"
0047	1290	O	T	3	L1			
0048	1300	O					86	"SALES FROM ALL BOOKS FOR"
0049	1310	O				SUBJ	98	
0050	1320	O					112	"DEPARTMENT"
0051	1330	O				SUBSL 1R	125	"S"
0052	1335	O					126	"S"
0053	1340	O	T	3	L2			
0054	1350	O					86	"SALES FROM ALL BOOKS FOR"
0055	1360	O				GSAREA	98	
0056	1370	O					112	"DIVISION"
0057	1380	O				GARSL 1R	125	"S"
0058	1385	O					127	"S"
0059	1390	O	T		LR			
0060	1400	O					92	"SALES FROM ALL BOOKS FOR"
0061	1410	O					100	"COLLEGE"
0062	1420	O				TOTSL 1	125	"S"
0063	1425	O					128	"S"

6

SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR
AUTH	10	01777(L)	GSAREA	10	01765(L)	STOKNO	5	01762(R)	YABR	5,0	00544(L)T
BKSL	7,2	01127(L)	NUMSHP	5,0	01125(R)	SUBJ	10	01772(L)	TITLE	20	02004(L)
EDITN	3	02016(L)	PAGE	4,0	00523(L)	SUBSL	9,2	01131(L)	TOTSL	12,2	01136(R)
GARSL	10,2	01133(R)	PPCOPY	5,2	01124(L)	TARA	5	01142(R)T	UPDATE	6,0	00537(L)

7

Figure 10-8. Batch Job Output (Continued)

PAGE 0003	TEXTSL	INDICATORS USED
	INDICATOR 10 DEFINED	0020
	REFERENCED	0021
	INDICATOR 15 DEFINED	0007
	REFERENCED	0039
	INDICATOR L1 DEFINED	0010
	REFERENCED	0047
	INDICATOR L2 DEFINED	0009
	REFERENCED	0053
	INDICATOR LR NOT DEFINED	
	REFERENCED	0059
	INDICATOR OF DEFINED	0004
	REFERENCED	0027 0031
	INDICATOR 1P NOT DEFINED	
	REFERENCED	0022 0030

8

Figure 10-8. Batch Job Output (Continued)

Programming Applications

PAGE 0004	TEXTSL	FIELD NAMES USED
AUTH	(FIELD) 10	0011
	REFERENCED	0041
BKSL	(FIELD) 7.2	0016
	REFERENCED	0017 0018 0019 0046
EDITN	(FIELD) 3	0013
	REFERENCED	0043
GARSL	(FIELD) 10.2	0018
	REFERENCED	0057
OSAREA	(FIELD) 10	0009
	REFERENCED	0055
NUMSHP	(FIELD) 5.0	0015
	REFERENCED	0016 0021 0044
PAGE	NOT DEFINED	
	REFERENCED	0026 0029
PPCOPY	(FIELD) 5.2	0014
	REFERENCED	0016 0045
STOKNO	(FIELD) 5	0008
	REFERENCED	0020 0040
SUBJ	(FIELD) 10	0010
	REFERENCED	0049
SUBSL	(FIELD) 9.2	0017
	REFERENCED	0051
TABA	(TABLE) 5	0005

8

Figure 10-8. Batch Job Output (Continued)

PAGE 0005	TEXTSL		FIELD NAMES USED	
	REFERENCED	0020		}
TABB	(TABLE)	5.0	0005	
	REFERENCED	0020	0021	
TITLE	(FIELD)	20	0012	
	REFERENCED		0042	
TOTSL	(FIELD)	12.2	0019	
	REFERENCED		0062	
UDATE	NOT DEFINED			
	REFERENCED		0024	

PAGE 0006	TEXTSL		FILE NAMES USED		
CARDS		0002		}	
	REFERENCED	0007			
REPORT		0004			
	REFERENCED	0006	0022		
TABFILE		0003			
	REFERENCED	0005			
					(8)

PAGE 0007	TEXTSL		FILE NAMES USED		
NO. SERIOUS ERRORS 000	NO. WARNINGS 000			}	
PROCESSOR TIME#01001131	ELAPSED TIME#0100129				
END OF COMPILE					
END OF PREPARE					
					(8)
					(9)
					(10)

Figure 10-8. Batch Job Output (Continued)

Programming Applications

TEXTBOOK SALES		1/02/75	PAGE 0001			
STOCK NO.	AUTHOR	TITLE	EDN NO.	SOLD	PRICE/CDPY	SALES/BOOK
00001	WAKEFIELD	PRINCIPLES OF ACCT	1	00010	\$8.75	\$87.50
00002	SMITH	ACCOUNTING FOR MGT	1	00005	\$5.75	\$28.75
SALES FROM ALL BOOKS FOR ACCOUNTING DEPARTMENT						\$116.25*
00003	WOODRY	MANAGEMENT BY OBJECT	3	00007	\$9.00	\$63.00
SALES FROM ALL BOOKS FOR ADMINISTRATION DEPARTMENT						\$63.00*
00007	SIGMA	APPLIED STAT FOR BUS	2	00010	\$8.50	\$85.00
00008	OLCOTT	BAYESIAN STATISTICS	1	00011	\$8.75	\$96.25
SALES FROM ALL BOOKS FOR STATISTICS DEPARTMENT						\$181.25*
SALES FROM ALL BOOKS FOR BUSINESS DIVISION						\$360.50**
00020	RAPHAEL	ROMANTICISM IN ART	1	00015	\$15.00	\$225.00
00021	RAND	FIGURE DRAWING	1	00021	\$11.00	\$231.00
SALES FROM ALL BOOKS FOR ART DEPARTMENT						\$456.00*
00030	DAWDLER	THE BAROQUE PERIOD	2	00003	\$7.00	\$21.00
00032	LENNON	ROCK AS AN ART FORM	3	00004	\$4.50	\$18.00
SALES FROM ALL BOOKS FOR MUSIC DEPARTMENT						\$39.00*
00040	LESLIE	THE PROGRAMMING MIND	1	00030	\$8.75	\$262.50
SALES FROM ALL BOOKS FOR PHILOSOPHY DEPARTMENT						\$262.50*
SALES FROM ALL BOOKS FOR HUMANITIES DIVISION						\$757.50**

11

Figure 10-8. Batch Job Output (Continued)

						PAGE 0002
STOCK NO.	AUTHOR	TITLE	EDN NO.	SOLD	PRICE/COPY	SALES/BOOK
00051	MESSICK	CIRCUITS	2	00015	\$11.00	\$165.00
00052	MESSICK	BASIC ELECTRONICS	1	00020	\$6.00	\$120.00
SALES FROM ALL BOOKS FOR FLEC ENG						DEPARTMENT \$285.00*
00060	GRUNDY	MODERN MATH IDEAS	2	00003	\$8.00	\$24.00
SALES FROM ALL BOOKS FOR MATH						DEPARTMENT \$24.00*
00070	BROWN	MODERN PHYSICS	1	00018	\$15.00	\$270.00
00072	DARWELL	NUCLEAR PHYSICS	1	00005	\$7.50	\$37.50
SALES FROM ALL BOOKS FOR PHYSICS						DEPARTMENT \$307.50*
SALES FROM ALL BOOKS FOR SCI & TECH						DIVISION \$616.50**
SALES FROM ALL BOOKS FOR COLLEGE						\$1,734.50***

11

END OF PROGRAM
IEOJ

CPU (SEC) = 30
ELAPSED (MIN) = 2
TMU, JAN 2, 1975, 4:40 PM
END OF JOB

12

13

14

Figure 10-8. Batch Job Output (Continued)

Programming Applications

5. The standard first-page heading output by the compiler during compilation, showing the software part number for RPG/3000, the product name, and the current date and time.
6. The Source Program Listing, showing all records included in the source program, plus (in the first four columns) the sequence number automatically assigned to each record by the compiler. (The sequence number assigned by the programmer for each record appears in the next group of columns.)
7. The Symbol Table Listing output by the compiler, showing the name, length, type, and beginning address of each field, table, and array used by the program. (See Section XI for further information.)
8. The Cross Reference Listing output by the compiler, showing the names and characteristics of indicators, fields, tables, arrays, and files used by the program, and where in the program they are referenced. (See Section XI for further details.)
9. The standard message generated at the end of compilation, indicating no serious errors or warnings, and the duration that the compiler used the central processor plus the total duration from beginning to end of compilation (both in hours/minutes/seconds).
10. The standard message output by the MPE/3000 Segmenter Subsystem, indicating that the object program was successfully prepared (segmented) by MPE/3000 and is linked to all software modules required for its support.
11. The report produced by the object program, showing all items described in the Output Specifications.
12. The object program terminations message, generated by MPE/3000.
13. The MPE/3000 :EOJ command that terminates the job.
14. The standard MPE/3000 job termination message, showing central processor time used by the entire job, total time elapsed since the job was initiated, the current date and time, and the END OF JOB record.

10-10. KSAM FILE APPLICATIONS

As noted in Section IV, the Keyed Sequential Access Method (KSAM) allows you to create files that can be processed both sequentially and randomly. For sequential access of the entire file or between limits, and for random access, each record in a KSAM file has a key through which that record can be accessed. In this section, several applications of KSAM files are demonstrated.

10-10A. Creating KSAM Files

A KSAM file can be created using the KSAMUTIL program or by your own RPG program.

If you specify a KSAM file as an Output file on your file description specifications ("O" in column 15), and it is **not** a file addition ("A" in column 66), your program will automatically build a new KSAM file if it does not already exist on disc.

The key information you provide in your file specification record and your **KEYFL** continuation record are used in defining the new file.

RPG will build your file with a default number of records of 1023. You can override this using the "DISC=" keyword parameter on the MPE file statement.

The sample program in Figure 10-9A shows a **KSAM** file being created using data from the batch transaction file shown in Figure 10-9. The file is created with the **KSAM Firstrec = 1** option, allowing duplicate keys to be added in random order.

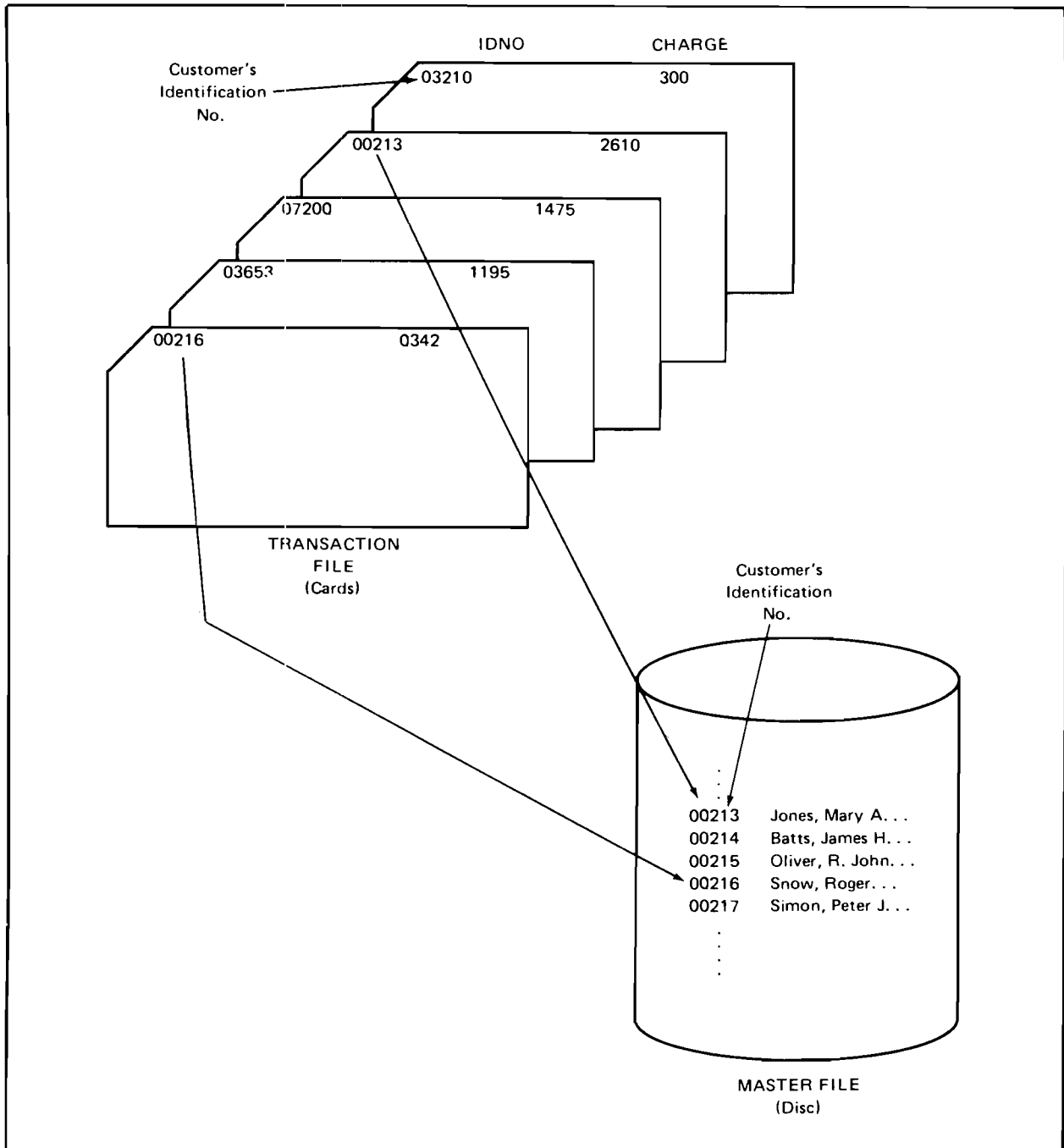


Figure 10-9. Chaining to a KSAM File

Programming Applications

HEWLETT  PACKARD

RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS

Page _____ of _____

Programmer _____
Program Title _____

Date _____

Punching Instructions			
Graphic			
Punch			

Program Name

--	--	--	--	--	--	--	--	--	--

Control Record Specification

Sequence Number	Form Type (F)	Debug	Inverted Print (I)	First Record No. (1)	All Control Seq. (S O E)	Processing Mode (R L)	Key Field Record Address	Record Address (A P K L)	File Org. Add. (D I X S M D 7/7)	Table Area Look Up (B)	Sign Process (S N L O B)	Form Positioning (I)	Indicator Setting (S/B/T)	File Translation (T O)	Blank Sign Substitute	Sign Suppress (S)	Cost Reference Listing (X)	Control Sequence Check (S N)	Error Response (1 2 3 4 5)
1	H																		

File Description Specifications

Sequence Number	Form Type (F)	File Name	File Type (I O D C)	Group Sequence	Record Format (F V)	Block Length	Logic Record Length	Processing Mode (R L)	Key Field Record Address	Record Address (A P K L)	File Org. Add. (D I X S M D 7/7)	Table Area Look Up (B)	Key Field Starting Location (0 9999)	Extension Code (E L)	Device Class Name	Dir. Label Cont. (S E R 2 9)	Name of Label	Ext. Option Type (E B C D)	to ASCII File	Option Target	File Addition (A)	Extent (1 2)	File Cond. (U I U B)
1	F	TRANSFL	D		F		80						015C										
2	F	MASTFL	D		F		256						015C										
3	F	KEYFL	D		F																		

HEWLETT  PACKARD

RPG INPUT SPECIFICATIONS

Page _____ of _____

Programmer _____ Date _____
Program Title _____

Punching Instructions			
Graphic			
Punch			

Program Name

--	--	--	--	--	--	--	--	--	--

Sequence Number	Form Type (I)	File Name	Group Sequence	Number of Records (I/N)	Option (O)	Record Identification Codes			Field Position		Field Name	Control Level (L 1 L 9)	Matching or Changing Fields (M) MB C (C)	Field Record Relation			Field Indicators		
						1	2	3	From (1 9999)	To (1 9999)				Plus	Minus	Zero or Blank			
1	F	TRANSFL	NS																
2	F	MASTFL	NS																
3	F	KEYFL	NS																

HEWLETT  PACKARD

RPG OUTPUT SPECIFICATIONS

Page _____ of _____

Programmer _____ Date _____
Program Title _____

Punching Instructions			
Graphic			
Punch			

Program Name

--	--	--	--	--	--	--	--	--	--

Sequence Number	Form Type (O)	File Name	Type (M O T E)	Skip	Output Indicators				File Name	Edit Code	Blank After (B)	End Position (1 9999)	Packed (B) or (P) B L R 7 8	Constant or Edit Word	Comments
					And	And	And	And							
1	F	MASTFL	D												
2	F	KEYFL	D												
3	F	TRANSFL	D												

Figure 10-9A. Creating a new KSAM file

10-11. KSAM Files and CHAIN Operation

The chaining features of RPG/3000 can be used to randomly retrieve selected records from a KSAM file via its record keys. In such applications, the KSAM file serves as the chained file. As an example, suppose that a large department store is processing a sequential input file on cards; this file reflects recent customer transactions with the store, and hence is called a transaction file. In this file each record contains two fields, as shown in Figure 10-9: IDNO, reflecting the customer's five-digit identification code, and CHARGE, reflecting the amount charged (but not yet paid) by the customer. The programmer wishes to use this transaction file to update billing data on a master file that contains information about all of the store's regular customers — their names, addresses, current balances, and so forth. Specifically, he wants to add the amount charged each customer to the customer's current balance (the CURBAL field on the customer's record in the master file). The master file is a KSAM file that is to be processed randomly. In this file, the key field on each record (ID) contains the customer identification number that is also reflected in the IDNO field of the records in the transaction file. The identification number on each record in the transaction file can be used to chain to the corresponding record in the master file. The object program reads this number from a transaction file record, locates the record with the identical key in the masterfile, and makes that entire record available for processing. The programmer accomplishes all this by requesting the CHAIN operation.

HEWLETT-PACKARD RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS Page 1 of 4

Programmer: **AMOS, NED** Date: **3/2/77** Punching Instructions

Program Title: **CHAINJOB** Punch

Control Record Specification																								
Error Numbers																								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

File Description Specifications																								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0002	F	TRANSFL	I	P																				
0003	F	MASTFL	U	C																				
0004	F	MSGFL	O	F																				
	F																							
	F																							
	F																							
	F																							
	F																							
	F																							

Figure 10-10. Specifying Chaining to KSAM File

In the Calculation Specifications, the CHAIN operation (Item 6) appears, naming IDNO as the chaining field and MASTFL as the chained file. If the program cannot locate a record in MASTFL that corresponds to the key from IDNO in TRANSFL, it turns Indicator 04 on. But if the record is found, the input fields described for MASTFL are moved into work areas for processing. Then, with the ADD operation (Item 7), the amount charged during this transaction (CHARGE) is added to the current BALANCE (CURBAL).

HEWLETT PACKARD **RPG CALCULATION SPECIFICATIONS** Page 3 of 4

Programmer AMOS, NED Date 3/2/77

Program Title CHAINJOB

Punching Instructions
 Graphic Punch

Program Name


Sequence Number	Form Type (C) Control Level (LD, LR) LP, SR, AN, OR	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Field Length (1-255)	Decimal Positions (99)	Half Adjust (H)	Resulting Indicators			Comments
		Not	And	And								Arithmetic	Plus Minus Zero	Compare	
00210	C				IDNO	CHAIN	MASTFL					04			(6)
00211	C	02			CURBAL	ADD	CHARGE	CURBAL							(7)
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														
	C														

Figure 10-10. Specifying Chaining to KSAM File (Continued)

Programming Applications

The File Extension Specifications show the entry AA in the Chaining File Record Sequence Field (Item 2). This same entry also appears in the Group Sequence Field of the Input Specifications (Item 3). The chaining code C1 appears in the Chaining File Code Field of the File Extension Specifications (Item 4). This same code also appears in the Matching or Chaining Fields Field of the Input Specifications (Item 5), identifying the field named ID in TFILE as the chaining field. Notice that in the File Description Specifications, MFILE is identified as a chained file (Item 6).

In the Output Specifications, both valid output (Item 7) and error messages (Item 8) are written to PFILE. This last output is similar to the error-handling procedures used in the previous example.

HEWLETT  PACKARD RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS Page 2 of 4

Programmer MEDLEY, R. A. Date 3/5/77

Program Title CHAINCODE

Punching Instructions	
Graphic	
Punch	

Program Name KICHAIN


File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec Seq	Chaining File Code (C1 C9)	From Filename	To Filename	Table Array or Routine Name	Entries Per Record (1 999)	Entries Per Table Array (1 9999)	Entry Length (1 256)	Data Format (P B L R)	Decimal Positions (0 9)	Table Array Sequence (A D)	Table Array Name	Entry Length (1 256)	Data Format (P B L R)	Decimal Positions (0 9)	Table Array Sequence (A D)	Comments
0030	E	AA	C1	TFILE	MFILE													
		(2)	(4)															

Line Counter Specifications

Sequence Number	Form Type (L)	Filename	1		2		3		4		5		6		7		8		9		10		11		12	
			Line Number	Channel Number (O/U/F/L)	Line Number	Channel Number (O/U/F/L)	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
L																										
L																										
L																										

Figure 10-11. Specifying Chaining Codes for a KSAM File (Continued)

HEWLETT  PACKARD RPG INPUT SPECIFICATIONS Page 3 of 4

Programmer MEDLEY, R.A. Date 3/5/77 Punching Instructions

Graphic					
Punch					

Program Title CHAINCODE Program Name KCHAIN

Sequence Number	Form Type (I)	File Name	Group Sequence # (I)	Number of Records (I/N)	Option (O)	Record Indicator/ Look Ahead/Trailer	Record Identification Codes						Field Position			Control Level (L/LG)	Matching or Chaining Fields (M/B/C/L/CS)	Field Record Relation	Field Indicators																				
							1	2	3	4	5	6	From (1-9999)	To (1-9999)	Decimal Position (0-9)				Plus	Minus	Zero or Blank																		
0040	I	TFILE	AA																																				
0041	I		(3)											15	20																								
0042	I	MFILE	BB																																				
	I		OR																																				
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						
	I																																						

Figure 10-11. Specifying Chaining Codes for a KSAM File (Continued)

HEWLETT-PACKARD **RPG OUTPUT SPECIFICATIONS** Page 4 of 4

Programmer: MEDLEY, R. A. Date: 3/5/77

Program Title: CHAIN CODE Punching Instructions

Graphic

Punch

Program Name **KCHAIN** 75 76 77 78 79 80

Sequence Number	Form Type (OI)	File Name	Type (H/O/T/E)				Space	Skip	Output Indicators						Field Name	Edit Code	Blank After (B)	End Position (1-9999)	Packed Binary (P.B.L.R./2/4)	Constant or Edit Word	Comments
			O	A	Z	I			Before	After	Not	And	And	Not							
0050	0	PROFILE					2				01	01		ID		10					
0051	0												NAME		20					7	
0052	0																				
0053	0						1				01	03	ID		10					8	
	0														30	DELETED INFORMATION					
	0																				
	0																				
	0																				
	0																				
	0																				
	0																				
	0																				
	0																				

Edit Codes

1	A	J	commas & zero balances	X	Remove Plus Sign
2	B	K	commas	Y	Date Field Edit
3	C	L	zero balances	Z	Zero Suppress
4	D	M			
No Sign	CR	.			

Figure 10-11. Specifying Chaining Codes for a KSAM File (Continued)

10-13. KSAM Files, RAFS, and Processing Between Limits

KSAM files can be accessed sequentially between limits by means of record address files (RAFS). A RAF used for this purpose must contain two entries for each record sought on the KSAM file. The RAF supplies the high and low limits that denote the portion of the KSAM file to be processed. Programmers must observe these rules when creating the RAF:

1. Each record in the RAF must contain only two entries; one entry specifies the lower limit of the KSAM file, where processing begins; the other entry specifies the upper limit, where processing ends. (The program begins processing at the lower limit and continues until the upper limit or the end-of-data is encountered.)
2. The first entry must begin in Position 1 of the RAF record. The second entry must begin in the position immediately following the end of the first entry. No intervening spaces are permitted between the two entries.
3. The record key entries must be specified in packed decimal format if the keys in the KSAM file are also written in that format.

The RPG program reads the first record in the RAF and begins processing the KSAM file at the lower limit specified in that record, continuing sequentially until the upper limit specified in the RAF record or the end of data on the KSAM file is reached. Then the program reads the next record from the RAF, and processes the KSAM file between the limits specified on that record. When the RAF reaches the end of file, processing ends. If two limits supplied by a single record in the RAF are equal, only the record with a matching key in the KSAM file is retrieved.

The limits specified in the RAF need not reflect valid record keys in the KSAM file — processing begins when the program finds a key that is greater than or equal to the lower limit; processing terminates when the program finds a key that is greater than the upper limit or when the end of the KSAM file is reached. This feature allows a single KSAM file to be processed completely many times when the program specifies limits that are both lower and higher than the keys on the KSAM file.

As an example of file-access between limits, consider a program that processes a KSAM file containing employee records (EMPFIL) using input from a RAF (LIMFIL). EMPFIL is organized in sequence by four-digit numbers that identify the company departments to which the employees belong. The programmer wishes to print a report that shows the names of employees in Departments 0006 through 0010, 0016 through 0020, 0026 through 0030, and 0036 through 0040. To do this, he sets up a RAF containing the limits shown in Figure 10-12.

In the File Description Specification for his program, the programmer specifies LIMFIL as a RAF (Figure 10-13, Item 1). He also indicates that the length of each field (limits) on the records in the RAF is 4 positions (Item 2). He describes EMPFIL as a KSAM file (Item 3) to be processed sequentially between limits (Item 4). He identifies a third file, REPFIL, to contain the report written to the line printer (Item 5).

In the File Extension Specifications, the programmer declares that LIMFIL (Item 6) is used to define the records processed in EMPFIL (Item 7).

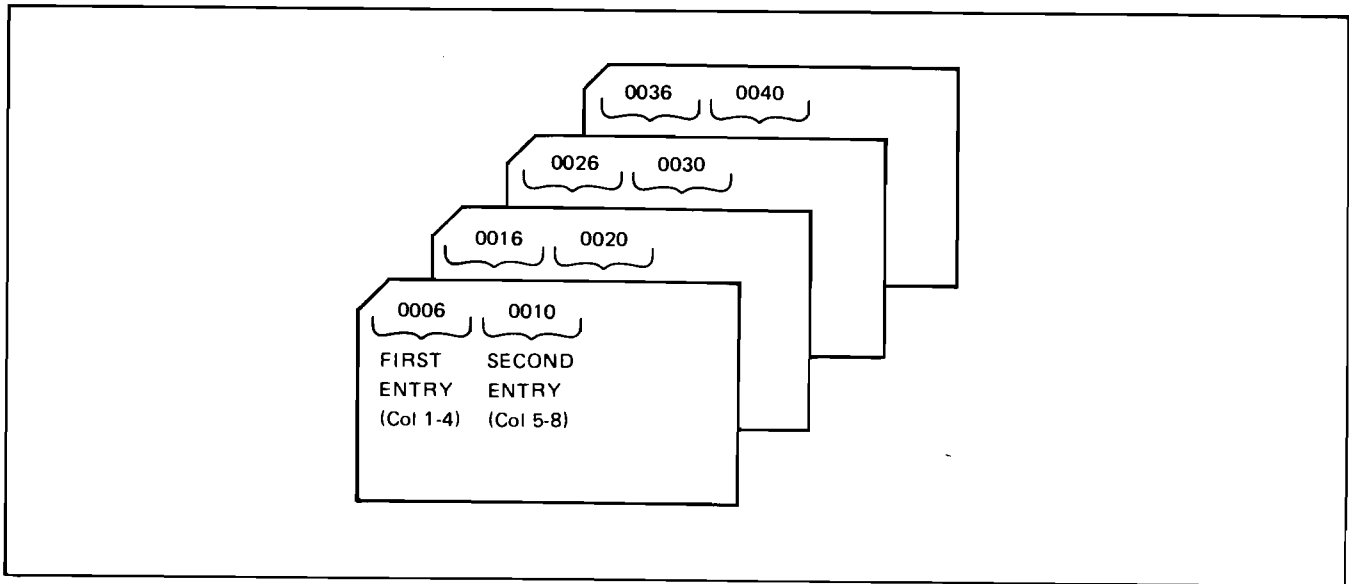


Figure 10-12. RAF File Entries for Processing Between Limits

HEWLETT ^{HP} ~~PACARD~~ **RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS** Page 1 of 4

Programmer **JOHNSON, RONALD K.** Date **4/8/77** Graphics Pictor Program Name **RAFJOB**

Control Record Specification

Sequence Number		Form Name		Form Type		Form Length		Form Code		Form Title		Form Description		Form Security		Form Control		Form Other		Form Total	
1		2		3		4		5		6		7		8		9		10		11	
0020		IR		F		8		4		2		E		C		A		R		D	
0021		IP		F		24		4		1		D		I		S		C		L	
0022		IO		F		132		4		0		F		L		P		L		P	

File Description Specifications

Sequence Number		File Name		File Type		File Length		File Code		File Title		File Description		File Security		File Control		File Other		File Total	
1		2		3		4		5		6		7		8		9		10		11	
0020		LIMFILE		I		8		4		2		E		C		A		R		D	
0021		EMPFILE		I		24		4		1		D		I		S		C		L	
0022		REPFIL		O		132		4		0		F		L		P		L		P	

Figure 10-13. KSAM File Processing by RAF, Between Limits

HEWLETT PACKARD RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS Page 2 of 4

Programmer JOHNSON, RONALD K. Date 4/8/77

Program Title LIMITJOB

Punching instructions
 Graphics _____
 Paper _____

Program Name **RAFJOB**


File Extension Specifications

Sequence Number	Form Type (E)	Chaining File Rec. Seq	Chaining File Code (C) (9)	From Filename	To Filename	Table Array in Rec. File Name	Entries Per Record (1-999)	Entries Per Table Array (1-999)	Entries Length (1-256)	Table Extension (P B C H)	Decimal Positions (0-9)	Table Array Sequence (A-D)	Table Array Name	Entries Length (1-256)	Data Extension (P B C H)	Decimal Positions (0-9)	Table Array Sequence (A-D)
0030	E			LIMFILE	EMPFILE												
	E																
	E																
	E																
	E																
	E																
	E																
	E																

Line Counter Specifications

Sequence Number	Form Type (L)	Filename	Line Number	Character Number (0-255)	Line Number	Character Number (0-255)	Line Number	Character Number	Line Number	Character Number	Line Number	Character Number	Line Number	Character Number	Line Number	Character Number	Line Number	Character Number
	L																	
	L																	
	L																	

Figure 10-13. KSAM File Processing by RAF, Between Limits (Continued)

HEWLETT  PACKARD RPG INPUT SPECIFICATIONS

Page 3 of 4

Programmer JOHNSON, RONALD K Date 4/8/77


Program Title LIMITJOB

Punching restrictions
 Graphic
 Punched

Program Name RAFIJOB

Sequence Number	Field	Field Name	Field Length	Field Position	Record identifier codes			Field Position		Field Name	Control Level (1-9)	Maximum in Changing Length (M1-M9, L1-L9)	Field indicators							
					1	2	3	From (1-9999)	To (1-9999)				Plus	Minus	Zero or Blank					
					Position (1-9999)	Next (N)	Position (1-9999)	Position (1-9999)	Position (1-9999)				Character	Position (1-9999)	Next (N)	Position (1-9999)	Character	Position (1-9999)	Next (N)	Position (1-9999)
0040	EMPFILE	AA	01	02	01	02	CA													
	OR						CI													
									10	DEPT										
										NAME										

Figure 10-13. KSAM File Processing by RAF, Between Limits (Continued)

HEWLETT  PACKARD RPG OUTPUT SPECIFICATIONS Page 4 of 4

Programmer: JOHNSON, RONALD K. Date 4/8/77 Punching Instructions
Graphic:

--

Punch:

--

Program Title: LIMITJOB Program Name:

RAFJOB

Sequence Number	Form Type	File Name	Type		Stack	Skip	Output Indicators					Field Name	Edit Code	Blank Alter. Bl.	End Position (1-9999)	Packed Binary (P B L R Z 4)	Constant or Edit Word	Comments
			Before	After			And	And	And	And	And							
0050	0	REPFILE	H	4	1													
0051	0		OR															
0052	0																	
0053	0																	
	0		D	2														
	0																	
	0																	
	0																	
	0																	
	0																	
	0																	

1 A J commas & zero balances X Remove Plus Sign
2 B K commas zero balances Y Date Field Edit
3 C L zero balances Z Zero Suppress
4 D M
No Sign CR

Constant or Edit Word

Figure 10-13. KSAM File Processing by RAF, Between Limits (Continued)

In the Input Specifications, the two input fields to be retrieved from EMPFILE are described (Item 8). Indicator 01 is used to indicate active records (flagged by an A in position 80) (Item 9). Indicator 02 indicates inactive, deleted records (flagged by an I in position 80) (Item 10).

In the Output Specifications, a heading line is specified (Item 11) and detail records are printed when Indicator 01 is set *on* (Item 12).

10-14. KSAM Files, RAFS, and Random Processing

KSAM files can also be accessed randomly via RAFs. In such cases, the RAF must contain the keys for the records to be processed; each time a key is encountered in the RAF, a record with the same key is selected from the KSAM file. Programmers must follow these rules when creating the RAF:

1. Each record in the RAF must contain one or more record keys for records to be processed in the KSAM file.
2. The first key on a RAF record must begin in Position 1 of that record, and any additional keys on this record must be continued without blank spaces between key fields.
3. The length of the key fields on all records must be the same, but the number of fields per record can vary from record to record. (A blank field the same length as the record key causes the RPG program to skip to the next record in the RAF.)
4. The key field entries on the RAF and the key fields in the KSAM file all must be the same length and be written in the same format.

When processing begins, the RPG program reads the first record from the RAF and uses the first key on that record to locate a record on the KSAM file. After this KSAM record is processed, the program uses the next key encountered in the RAF record to select another record from the KSAM file. When the program encounters a blank field or the end of the record in the RAF, it reads another record from the RAF. When the program encounters the end of the RAF, processing terminates.


To illustrate random-access of a KSAM file with a RAF, suppose that a programmer wishes to retrieve records from a KSAM file named KSAMFILE via a RAF named RAFFILE. In the File Description Specifications, he describes KSAMFILE as his primary input file (Figure 10-14, Item 1) to be processed randomly (Item 2) using KSAM (Item 3). He describes RAFFILE as a RAF (Item 4) to be checked during processing for the end-of-file condition (Item 5). The record key is specified as six characters long (Item 6).

In the File Extension Specifications, the programmer relates the RAF (Item 7) to the KSAM file (Item 8). (Once the records from the KSAM file are accessed, additional coding is needed to process them.)

10-15. IMAGE FILE APPLICATIONS

As noted in Section IV, IMAGE/3000 files can be used with RPG/3000 programs. The following restrictions, however, apply:

1. Data is added and retrieved as complete entries; in other words, you cannot read or modify single data items through RPG. Therefore, if the RPG program is to read an entry from a data set, the specified password must correspond to a user class number that allows read access to all data items

HEWLETT  PACKARD

RPJ CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS

Page **1** of **2**

Programmer: **ROBBINS, JUDITH** Date: **7/7/77**

Program Title: **RANDOMJOB**

Printing Instructions
 Copy: _____
 Punch: _____

Program Name: **RANJOB**

Control Record Specification

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
														1. Sequence Number 2. File Type (R) 3. File Type (D) 4. Description (R/C/D) 5. Start of File (R) 6. End of File (R) 7. File Sequence Check (A/D) 8. File Sequence (R) 9. Block Length 10. Record Length 11. Retrieval Method (R/L) 12. Key Field Record Address 13. Field Length 14. Record Address Type (A/R/L) 15. File Org. Addr. (D/LX/S/M/D/F/T) 16. Overflow (00/0A/0B/0C) 17. Key Field Starting Location (0-9999) 18. Extension Code (1-9) 19. Device Code (M) 20. Name of Unit 21. Operator's Name (R/L) 22. Unit Target 23. File Address A 24. Extension (3) 25. File Name (1-18)																																																											
														26. File Type (R) 27. File Type (D) 28. File Type (R) 29. File Type (D) 30. File Type (R) 31. File Type (D) 32. File Type (R) 33. File Type (D) 34. File Type (R) 35. File Type (D) 36. File Type (R) 37. File Type (D) 38. File Type (R) 39. File Type (D) 40. File Type (R) 41. File Type (D) 42. File Type (R) 43. File Type (D) 44. File Type (R) 45. File Type (D) 46. File Type (R) 47. File Type (D) 48. File Type (R) 49. File Type (D) 50. File Type (R) 51. File Type (D) 52. File Type (R) 53. File Type (D) 54. File Type (R) 55. File Type (D) 56. File Type (R) 57. File Type (D) 58. File Type (R) 59. File Type (D) 60. File Type (R) 61. File Type (D) 62. File Type (R) 63. File Type (D) 64. File Type (R) 65. File Type (D) 66. File Type (R) 67. File Type (D) 68. File Type (R) 69. File Type (D) 70. File Type (R) 71. File Type (D) 72. File Type (R) 73. File Type (D) 74. File Type (R)																																																											

File Description Specifications

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
														1. Sequence Number 2. File Type (R) 3. File Type (D) 4. Description (R/C/D) 5. Start of File (R) 6. End of File (R) 7. File Sequence Check (A/D) 8. File Sequence (R) 9. Block Length 10. Record Length 11. Retrieval Method (R/L) 12. Key Field Record Address 13. Field Length 14. Record Address Type (A/R/L) 15. File Org. Addr. (D/LX/S/M/D/F/T) 16. Overflow (00/0A/0B/0C) 17. Key Field Starting Location (0-9999) 18. Extension Code (1-9) 19. Device Code (M) 20. Name of Unit 21. Operator's Name (R/L) 22. Unit Target 23. File Address A 24. Extension (3) 25. File Name (1-18)																																																											
														26. File Type (R) 27. File Type (D) 28. File Type (R) 29. File Type (D) 30. File Type (R) 31. File Type (D) 32. File Type (R) 33. File Type (D) 34. File Type (R) 35. File Type (D) 36. File Type (R) 37. File Type (D) 38. File Type (R) 39. File Type (D) 40. File Type (R) 41. File Type (D) 42. File Type (R) 43. File Type (D) 44. File Type (R) 45. File Type (D) 46. File Type (R) 47. File Type (D) 48. File Type (R) 49. File Type (D) 50. File Type (R) 51. File Type (D) 52. File Type (R) 53. File Type (D) 54. File Type (R) 55. File Type (D) 56. File Type (R) 57. File Type (D) 58. File Type (R) 59. File Type (D) 60. File Type (R) 61. File Type (D) 62. File Type (R) 63. File Type (D) 64. File Type (R) 65. File Type (D) 66. File Type (R) 67. File Type (D) 68. File Type (R) 69. File Type (D) 70. File Type (R) 71. File Type (D) 72. File Type (R) 73. File Type (D) 74. File Type (R)																																																											


Handwritten entries in the File Description Specifications table:

- Row 1: 0020 KSAMFILEP 256R 6AX 10 DISCARD
- Row 2: 0021 RAFFILEIRE 804 6

Diagrammatic annotations:

- Circle 1 points to File Sequence (7)
- Circle 2 points to Record Length (10)
- Circle 3 points to Key Field Starting Location (10)
- Circle 4 points to Start of File (1)
- Circle 5 points to End of File (1)
- Circle 6 points to Field Length (6)

Figure 10-14. KSAM File Processing by RAF, Random Access

HEWLETT  PACKARD RPG FILE EXTENSION AND LINE COUNTER SPECIFICATIONS Page 2 of 2

Programmer ROBBINS, JUDITH Date 7/7/77

Program Title RANDOMJOB

Punching Instructions

Graphic	
Punch	

Program Name RANJOB

File Extension Specifications

Sequence Number	Entry Type (E)	Channeling File Rec. Seq	Channeling File Code (C1-C9)	Front Filename	Trs Filename	Table Array or Routine Name	Entries Per Record (1-999)	Entries Per Table Array (1-9999)	Entry Length (1-256)	Data Format (P B L R)	Decimal Positions (0-9)	Table Array Sequence (A, D)	Table Array Name	Entry Length (1-256)	Data Format (P B L R)	Decimal Positions (0-9)	Table Array Sequence (A, D)	Comments
0030	E			RAFFILE	KSAMFILE													
	m			7	8													
	m																	
	m																	
	m																	
	m																	

Line Counter Specifications

Sequence Number	Entry Type (E)	Filename	Line Number	Channel Number (C1-C9)	Line Number	Channel Number (C1-C9)	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
	L																	
	L																	
	L																	

Figure 10-14. KSAM File Processing by RAF, Random Access (Continued)

in the entry. If the RPG program is to write an entry to a data set, the password must correspond to a user class number that allows write access to all data items in the entry.

Since entries are handled in this way, data sets to be used with RPG programs are sometimes defined with one-item entries. However, if you intend to use QUERY with the data set, you may wish to define more items.

2. Only one search item can be used to reference a data set in a program unless the data set is defined as more than one file, or you are doing an ISAM simulation and processing between limits. (Consult Appendix H of this manual for more information about ISAM simulation.)
3. RPG supports all of the DBGET procedure input modes except reread. It provides two additional modes:
 - Simulated indexed sequential read, forward and backward (Appendix H).
 - Read up or down chain until key changes.

To use an IMAGE data base through RPG application programs you must describe the data base with file description specifications. A data set may be described by more than one file description specification to allow you to access it in more than one way, for example, performing both serial and chained reads or using two different search items (keys). The file description specification and its continuation records specify:

- an IMAGE file by naming both the data base and a data set within it
- a search item name
- an access mode (1 through 8)
- a password
- an input/output mode for the file.

In addition, you can add and delete entries by using special RPG output specifications.

Further information about IMAGE data bases appears in *IMAGE/3000 Data Base Management Subsystem Reference Manual* (part No. 30000-90041). To illustrate the use of IMAGE files by RPG programs, the following examples are presented.

10-16. Generating a Report Through IMAGE.

The sample RPG program in Figure 10-15 reads the SALES entries associated with a particular stock number and prints the contents in a report. The file description specifications include:

- Line 0003 — a description of the SALES data set as a chained input file with fixed length records 38 bytes long. The processing mode used for the data set is random. The key field is 8 bytes long and contains alphanumeric data. The file organization code M signifies an IMAGE file.
- Line 0004 — a data base name record specifying the STORE data base, an access (open) mode of 3, and input/output mode C (chained sequential read).
- Line 0005 — an item name record specifying the STOCK# search item as the key.

```

0001      SCONTROL USLINIT
0002      H                                     X

0003      FSALES  IC  F      38R 8AM
0004      F                                     KIMAGE STORE 3C
0005      F                                     KITEM  STOCK#
0006      F                                     KLEVEL DO=ALL
0007      F                                     KDSNAMESALES
0008      FINPUT  ID  F      8
0009      FPRINT  O  V      80

0010      ISALES  AA  01
0011      I      B  1  40ACCT
0012      I      5  12 STOCK#
0013      I      B 13 140QTY
0014      I      B 15 1R2PRICE
0015      I      R 19 222TAX
0016      I      B 23 262TOTAL
0017      I      27 32 PDATE
0018      I      33 3R DDATE
0019      IINPUT  RB
0020      I      1  8  Istock

0021      C      SETOF      12
0022      C      SETON      15
0023      C      EXCPT
0024      C      READ INPUT      LR
0025      C      SETOF      15
0026      C      SETON      16
0027      C  NLR      EXCPT
0028      C      SETOF      16
0029      C      LOOP      TAG
0030      C  NLR      Istock CHAINSALES      1211
0031      C  N11NLR      FXCPT
0032      C  M11N12NLR      GOIU LOOP

0033      OPRINT  E 2      16
0034      O      10 "ACCOUNT"
0035      O      19 "STOCK#"
0036      O      28 "QUANTITY"
0037      O      36 "PRICE"
0038      O      43 "TAX"
0039      O      52 "TOTAL"
0040      O      62 "PURCHASED"
0041      O      72 "DELIVERED"

```

Figure 10-15. Printing a Report

0042	0	E 1	01		
0043	0			ACCT Z	10
0044	0			STOCK#	19
0045	0			TOTAL J	52
0046	0			TAX J	44
0047	0			PRICE J	36
0048	0			QTY J	26
0049	0			PDATE	62
0050	0			DDATE	72
0051	0	E 22	12		
0052	0				15 "NO SUCH STOCK#"
0053	0	E 21	15		
0054	0				20 "ENTER STOCK# OR IEOD"

Figure 10-15. Printing a Report (Continued)

- Line 0006 — a level identification record specifying the DO-ALL password.
- Line 0007 — a data set name record specifying the SALES data set which overrides the file name.
- Line 0008 — a description of the INPUT file as a demand file with fixed length records 8 bytes long.
- Line 0009 — a description of the PRINT file as an output file of variable length records which are at most 80 characters long.

The input specifications describe:

- Lines 0010 through 0018 — a SALES data entry with five binary and three character (ASCII) data items.
- Lines 0019 through 0020 — an INPUT record of 8 bytes with a field named ISTOCK.

The calculation specifications, Lines 0021 through 0032, request input to the INPUT file which can be equated to \$STDIN by using the MPE :FILE command before executing the program. They also read the SALES entries with values equal to the stock number entered, print the information, and, when the end of chain is encountered, request another stock number.

The output specifications, Lines 0033 through 0050, describe a report with column headings for each item and one-line records for each entry. The ACCT item is edited with a Z edit specification and the QTY, PRICE, TAX, and TOTAL items with a J edit specification.

The last output specifications, Lines 0051 through 0054, describe the message to be printed if there is no entry with the requested stock number value and the message which prompts for the stock number.

10-17. Creating an IMAGE Data Base

If you have not yet created the data-base file you plan to use with your RPG program, you must design a schema for the data base and then use this schema to produce the data base. An example of an IMAGE

schema, created on a file named CUSTDBI using the Editor subsystem (EDIT/3000), appears in Figure 10-16. An example of a job stream that uses this schema as input to produce the data-base on a file called CMASTR appears in Figure 10-17. For a further explanation of IMAGE schema entries, see *IMAGE/3000 Data-Base Management Subsystem Reference Manual*. For details about the MPE commands used in the job stream, see the MPE Reference Manuals, the IMAGE/3000 Reference Manual, and Section XI of this manual.

```

$TITLE "          CUSTDBI:  SCHEMA RETAIL CUSTOMER DATA BASE"
$PAGE
BEGIN

DATA BASE CMASTR:
LEVEL:

ITEMS:
ITEMNO,   X21   << 1 >>
CUSTNO,   X61   << 2 >>
STATE,    X21   << 3 >>
CO,       X21   << 4 >>
CITY,     X41   << 5 *** >>
NAMF,     X201  << 6 >>
ADDR,     X201  << 7 >>
CITST,    X201  << 8 >>
CRLMST,   X21   << 9 *** >>
TRANS,    X21   << 10 >>
CHARGE,   X61   << 11 >>
PAY,      X61   << 12 >>
CREDIT,   X61   << 13 >>
BAL,      X61   << 14 >>
YTDLS,    X61   << 15 >>
YTDNO,    X61   << 16 >>
FILLER,   X101  << 17 >>
DELFTE,   X21   << 18 *** >>

SETS:
NAMF:     CDIR,A:
ENTRY:    CUSTNO(1):
CAPACITY: 100:

NAMF:     CUSTF1,D:
ENTRY:
ITEMNO,
CUSTNO(ICDIR),
STATE,
CO,
CITY,
NAME,
ADDR,
CITST,
CRLMST,
TRANS,
CHARGE,
PAY,
CREDIT,
BAL,
YTDLS,
YTDNO,
FTLLER,
DELFTE:
CAPACITY: 100:
END.

```

Figure 10-16. Data-Base Schema

```

!JOB SCHEMA,JOF,DIETZGEN,PPG3
!COMMENT SRSCHM1
!FILE DRSTEXT=CUSTDBI
!RUN DBSCHEMA,PUB,SYS1PARM=1
!RUN DBUTIL,PUB,SYS,CREATF
CMASTR
!EOJ

```

Figure 10-17. Data-Base Create Job Stream

10-18. Updating a Data Set

When your data-base file has been created, it is ready for use with your RPG program. As an example, the RPG program listed in Figure 10-18 demonstrates how to use direct-access methods to update a detail data set in an IMAGE transaction file. (This transaction file is CMSTR, produced by the job in the previous example.) When a transaction record is read, the value of the key field (in this case, CUSTNO) is used to chain to the detail record in the IMAGE data set. After the record is updated, a write is performed to place the updated record into the data set in the original location.

```

10* PR0G4
20* UP1
30* EXAMPLE OF UPDATING AN INDEXED FILE
50*CONTROL MAP
60*CONTROL QUOTE='
70*TITLE "          LP1. UPDATING AN INDEXED FILE"
80*PAGE
90H
100FTRANS4 IP AF H0 80          DISC          XLS 11          CX4
102FMASTER UC F 256 128R06AT  DISC
104F
106F          <IMAGE CMASR35
          <ITEM CUSTNO
          <LEVEL
          <DSNAMECUSTF1
          <STATUSXT
120FLISTING 0 F 96 96          DV          LP          CX4
130FTRANS4 AA 01 1 CD
140F          2 7 CUSTNOL1
150F          AB 02 1 C3
160F          2 7 CUSTNOL1
170F          12 172CHG          CX3
180F          18 232PAYMT
190F          24 292CROT
200FMASTER NS 03 1 CC 2 CM 128NCD
210F          OR 04 1 CC 2 CM 128 CD          CX3
220F          3 8 CUSTNO
230F          17 36 NAME
240F          77 77 CRLIM
250F          81 862CHARGE
260F          87 922PAY
270F          93 982CREDIT
280F          99 10423AL
290F          128 128 DELETE
300C L1          SETOF          111213
310C L1          SETOF          142021
C 01
COR 02          CUSTNO          CHAINMASTER          50H0
C 01N50          SETON          21          SAVE INDICATO
C 02N50          TOTCHG          ADD CHG          TOTCHG 62
C 02N50          TOTPAY          ADD PAYMT          TOTPAY 62
C 02N50          TOTCRD          ADD CROT          TOTCRD 62
C 03N50          CHARGE          ADD TOTCHG          CHARGE
C 03N50          PAY          ADD TOTPAY          PAY
C 03N50          CREDIT          ADD TOTCRD          CREDIT
C 03N50          BAL          ADD TOTCHG          BAL
C 03N50          BAL          SUB TOTPAY          BAL
C 03N50          BAL          SUB TOTCRD          BAL
390C 03N21          DELETE          COMP '0'          21
400C 03          CRLIM          COMP '2'          1112
410C 03N11N12CRLIM          COMP '4'          1314
420C 03 11          BAL          COMP 250.00          20          CX1
430C 03 12          BAL          COMP 500.00          20          CX1
440C 03 13          BAL          COMP 1000.00          20          CX1
450C 03 14          BAL          COMP 2000.00          20          CX1
490CL1          TOTCHG          SUR TOTCHG          TOTCHG
500CL1          TOTPAY          SUR TOTPAY          TOTPAY
510CL1          TOTCRD          SUR TOTCRD          TOTCRD
520OLISTING H          1010 1P
530O          OR          OV
540O          53 'CUSTOMER FILE LISTING'

```

Figure 10-18. Indexed Update Program

```

5500      H 1      1P
5600      OR      0V
5700
5800      8 'CUSTOMER'
5900      24 'CUSTOMER'
6000      68 'NEW'
6100      80 'CREDIT'
6200      86 'OVER'
6300
6400      H 2      1P
6500      OR      0V
6600
6700      7 'NUMBER'
6800      22 'NAME'
6900      39 'CHARGES'
7000      50 'CREDITS'
7100      60 'PAYMENTS'
7200      70 'BALANCE'
7300      79 'LIMIT'
7400      87 'LIMIT'
7500
7600      T 1      L1
7700
7800      CUSTNO 7
7900      NAME 30
8000      CHARGEK 40
8100      CREDITK 50
8200      PAY K 60
8300      BAL A 72
8400
8500      11 79 '250'
8600      12 79 '500'
8700      13 79 '1,000'
8800      14 79 '2,000'
8900      20 A6 '****'
9000      21 95 'DELETED'
9100
9200MASTER D 03
9300      21 128 'D'
9400
9500      CHARGE 86
9600      PAY 92
9700      CREDIT 98
9800      BAL 104

```

Figure 10-18. Indexed Update Program (Continued)

Note the following aspects of the source code used to perform this function. (On the listing in Figure 10-18, these points are high-lighted by boxes around the pertinent entries.)

Line(s)

- 102 Detail Data Set File Specification:
 - Column 15: U—Update File.
 - Column 16: C—Chained File.
 - Column 28: R—Random Access.
 - Columns 29-30: 06—Key field length.
 - Column 31: A—Key values are alphanumeric.
 - Column 32: I—IMAGE/3000 data set.
- 104 Data Base Name Record:
 - Column 66: 3—Exclusive Access.
 - Column 67: 5—Chained Read processing.
- 106 Data Item Name Record (specifies key field name).

10-19. CARD-READER/PUNCH APPLICATIONS

RPG/3000 allows you to read input from, and direct output to, an 80-column combined card-reader/punch. The program shown in Figure 10-19 illustrates such an application. This program checks the first column of

each card and sorts the cards into Stacker No. 1 when the column contains a "B" or "C", or into Stacker No. 2 when the column contains an "A". At the same time, the cards are both printed and punched. In the listing, the entries in the output specifications that select the particular stackers are high-lighted with a box. The listing also shows the MPE commands used to control the job (each preceded by a colon) and the data cards read by the program (appearing between the :EOD and :EOJ job control commands).

Note that the MPE :FILE command for the card-reader/punch specifies NOBUF; to use the card-reader/punch as a combined file or with the stacker-select feature, always use the NOBUF parameter.

```

:JOB FIELD.SUPPORT, HP32104
:FILE INPUT;DEV=CRPUNCH;NOBUF
:RPGGO
*
*****
*
* THIS RPG PROGRAM READS AN 80 COLUMN CARD FROM THE
* CARD READER PUNCH. THE CARD HAS AN "A", "B", OR "C"
* IN COLUMN 1.
*
* THE CARD WILL BE PUT IN STACKER 1 IF COLUMN 1 CONTAINS
* A "B" OR "C". IF COLUMN 1 CONTAINS AN "A", THE CARD
* WILL BE PUT IN STACKER 2.
*
* THE STACKER RECEIVING THE CARD WILL BE PRINTED AND
* PUNCHED ON THE CARD. ALSO, "PRINT / NO PUNCH" WILL BE
* PRINTED (BUT NOT PUNCHED) ON EACH CARD IN COLUMNS 65-80.
*
*****
*
$CONTROL USLIMIT
H
FINPUT CP F 80 CRP
IINPUT AA 01 1 CA
I OR 02 1 CB
I OR 03 1 CC
OINPUT D2 01
O OR1 02
O OR1 03
O 60 "STACKER "
O 61 "1"
O 61 "2"
O 01
O *PRINT
O * 80 "PRINT / NO PUNCH"

:EOD
A
B
C
C
B
B
A
:EOJ

```

Figure 10-19. Card-Reader/Punch Application



OPERATING THE RPG/3000 COMPILER

SECTION

XI

Once you have written and desk-checked your RPG/3000 program, you can prepare it for compilation and execution. If you plan to compile the source program from cards, submit the program for key-punching and check the resulting card deck to be sure that the cards (records) are in proper sequence, as shown in Figure 11-1. As part of the program, you must include:

1. One (and only one) Control Record Specification record.
2. At least one File Description Specification record.
3. At least one Input or Output Specification record (Unless all files are Display type).

All other specification records are optional.

You can also include with your program special instructions to the compiler, called **compiler subsystem commands**. These commands request special compilation options and are discussed in Section XII.

Now, you are ready to submit your program to the HP 3000 computer, either in a batch-processing job or in an interactive session. In either case, basically you follow these steps:

1. Access the computer by initiating the job or session.
2. Compile your source program into an object program. (If errors are detected during compilation, debug and re-compile the program.)
3. Prepare the object program for execution by linking it to all subroutines and procedures required for its support.
4. Execute the object program.
5. Terminate the job or session.

You accomplish these steps through commands directed to the Multiprogramming Executive Operating System (MPE/3000). The specifications for the commands you will need to perform these functions are described in this section, along with examples showing how they are used. In these specifications, the command formats, including all parameters, appear. For more explanatory information about these and other MPE commands, please see the MPE reference manuals.

You can enter commands through any standard input device, typically a terminal (for sessions) or a card reader (for jobs). Each command is accepted by the MPE Command Interpreter, which passes it to the appropriate system procedure for execution. Following this execution, control returns to the Command Interpreter, which is now ready for another command.

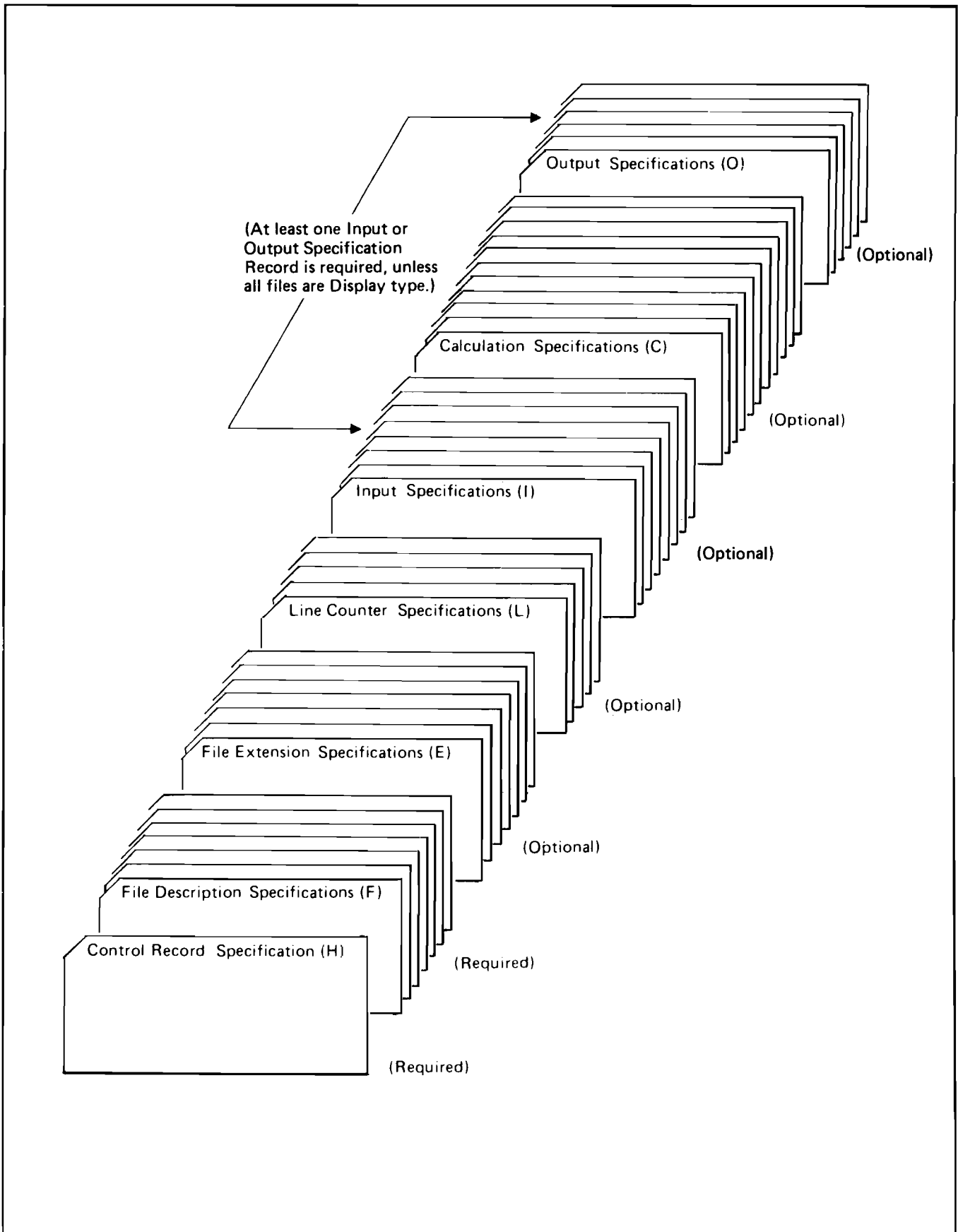


Figure 11-1. RPG/3000 Source Program Input Records

11-1. COMMAND ELEMENTS

Each MPE command consists of:

- A colon (required in all cases as an MPE command identifier).
- A command name (required in all cases).
- A parameter list (used in most cases).

A typical command including all three elements appears as follows:

:RUN PROG, ENTRYX

The *colon* identifies a statement as an MPE command. In an interactive session, MPE prints the colon on the terminal whenever it is ready to accept a command; you respond by entering the remainder of the command after the colon. In a batch job, however, *you* must enter the colon, placing it in column 1 of the source card (or card image) on which the command is to appear.

The *command name*, which you enter immediately after the colon, requests a specific operation. MPE prohibits embedded blanks within the name, and rejects the command if they appear. MPE interprets the next non-alphabetic character encountered as the end of the command name; typically this character is a blank.

The *parameter list* contains one or more parameters that specify options for the command. It is required in some commands, but is optional or prohibited in others. Parameter lists can include positional parameters and/or keyword parameter groups (defined below), separated from each other by delimiters such as commas, semicolons, equal signs, or other punctuation marks.

Normally, you must separate the parameter list from the command name by one or more blanks. (But when you omit the first optional parameter in a positional list, you can begin the list, starting with the comma that normally follows the first parameter, immediately after the command name. The comma serves in place of blanks as a delimiter, as noted under Paragraph 11-2.) Within the parameter list, any delimiter can be surrounded by any number of blanks, permitting a free and flexible command format.

MPE permits both decimal and octal numbers as command parameters. You distinguish between the two by preceding the octal numbers with a percent sign (%).

The end of each command is indicated by the end of the record on which it appears — for example, a *carriage return* for terminal input or the end of the card containing the command for card input. But if the last non-blank character of the record is a continuation character (as defined in Paragraph 11-4), the command is continued onto the next record.

Note: If you are running programs in batch job mode, bear in mind that MPE scans all 80 columns on each card image, and thus *no characters are ignored*.

11-2. Positional Parameters

With positional parameters, the meaning of a parameter depends upon its position in the parameter list. For example, in the :RPG command, issued to compile an RPG program, the parameter list specifies the input file containing the source program, the output file to which the object program is written, and the output file to which the source listing is transmitted, *always in that order*. In the following :RPG command, for instance, the variable names INP, OUT, and *LST indicate the source, object, and list files, respectively.

```
:RPG INP,OUT, *LST
```

Note: In *LST, the asterisk(*) is not a delimiter but a special character denoting a back-reference to a previously-defined file, as described in the MPE reference manuals; MPE regards the asterisk as part of the parameter.

Positional parameters are separated from one another by commas. Note that in the above example, the second delimiting comma in the parameter list is followed by an optional blank. In future examples, for clarity, delimiters will always be followed by blanks.

When you omit an optional positional parameter from within a list, you must still include the delimiter that would normally follow that parameter. Thus, on a listing, two adjacent delimiters indicate a missing optional parameter. When you omit a positional parameter that would otherwise immediately follow a command name, indicate this by entering its delimiter as the first character in the parameter list. When you omit positional parameters from the end of the list, however, you need not include delimiters to signify this — the terminating *return* or end-of-card is sufficient. The following examples demonstrate how to properly omit parameters from a command:

```
:RPG , USLFL, *LISTFL, MFL, NFL           First parameter omitted.
:RPG *SOURCEFL, , *LISTFL, MFL, NFL      Second (embedded) parameter omitted.
:RPG *SOURCEFL, USLFL, *LISTFL          Last two (trailing) parameters omitted.
:RPG                                     All parameters omitted.
```

11-3. Keyword Parameters

When a parameter list is so long that use of positional parameters becomes difficult, MPE provides keyword parameter groups. The meaning of such a group is independent of its position in the list — thus, you can enter keyword groups in any order with respect to each other. A keyword group consists of a keyword that denotes the group's meaning, sometimes followed by an equal sign and one or more sub-parameters. Each keyword group is preceded by a semicolon. When more than one sub-parameter appears in a group, they are usually separated from each other by commas. All delimiters can be optionally preceded or followed by blanks. The following example shows a :PREP command containing both positional and keyword parameters. INPT and OUTP are the variable names of the positional parameters. DL and CAP are keywords

that designate the keyword parameter groups. PH, DS, and MR are sub-parameters of the keyword group designated by CAP.

```
:PREP INPT, OUTP; DL=500; CAP=PH, DS, MR
```

When both keyword groups and positional parameters form a list, the positional parameters always occur before the keyword groups. When you omit trailing parameters from the positional group in this list, you need not include their delimiters since the occurrence of the first keyword indicates the omission. When you omit optional sub-parameters from a keyword group, simply follow the same rules that apply to positional parameters.

11-4. Continuation Characters

When the length of a command exceeds one record (for instance, one entry-line or source card), you may enter an ampersand (&) as the last non-blank character of the record and continue the command on the next record. This next record must begin with a colon (supplied automatically by MPE in interactive processing, but entered by yourself in batch processing). Optionally, you can embed blanks between the colon that begins the continuation record and the first non-blank character in the continuation record. In the example below, the command contains a continuation character at the end of the first line and an embedded blank at the beginning of the second.

```
:RUN PROGB; NOPRIV; LMAP; STACK=500; PARM=5; &  
: DL=600; LIB=G
```

You can continue commands up to 255 characters; prompting colons and continuation ampersands are not counted as part of this total.

When continuing a command onto another line, you must not divide a command name, keyword, positional parameter, or keyword sub-parameter — MPE does not permit any such element to span more than one line.

MPE does not begin interpretation of a command until the last record of the command is read.

11-5. COMMAND ERRORS

If you make an error while entering a command in an interactive session, MPE suppresses execution of that command, displays an error number indicating the type of error (described in the MPE reference manuals), and returns control to your terminal. If you desire a full explanation of the error, you can request it by entering any character other than a carriage-return immediately after the error number. In response, MPE displays the appropriate error message on the next line. If you do not require an explanation of the error, enter a carriage-return directly after the error number. MPE then prints a colon, prompting you for another command.

If you enter an erroneous command in a batch job (and do not precede this command with a :CONTINUE command, as discussed in Paragraph 11-6), MPE suppresses execution of the command, prints both the appropriate error number and message on your standard list device, ignores all subsequent commands in the job, and aborts the job.

11-6. REFERENCE SPECIFICATIONS

The reference specifications for the MPE commands you are likely to require appear on the following pages. For easy reference, they are presented alphabetically by command name; should you prefer to reference them by function, simply remove these pages and re-arrange them according to the *Functional List of Commands* shown in Table 11-1. For each command, the reference specifications show the following information:

- Syntax
- Parameter definitions (including meaning, constraints, and defaults).
- When legal (in sessions or jobs, during break, or programmatically).
- Whether interruptable (with BREAK key).
- Operation or functional description.
- Examples

In the reference specifications, the indication *Available In Break* means: the command can be entered and executed if the *current* operation is suspended by pressing the BREAK key on the terminal (or calling the CAUSEBREAK intrinsic). The notation *Available Programmatically* means: the command is available through the COMMAND intrinsic. CAUSEBREAK and COMMAND are explained in the *MPE reference manuals*. A summary of the rules for entering commands, plus notes on the syntax conventions used in this manual, appear in Table 11-2.

NOTE: IF YOU HAVE NO PRIOR EXPERIENCE WITH MPE, YOU SHOULD READ THE MPE REFERENCE MANUALS DEALING WITH USER COMMANDS BEFORE ATTEMPTING TO USE THE SPECIFICATIONS ON THE FOLLOWING PAGES.

Table 11-1. Functional List of Commands

Command	Function	Page Where Described
:RPG	Compiles a source program.	11-9
:RPGPREP	Compiles and prepares a program.	11-13
:RPGGO	Compiles, prepares, and executes a program.	11-11

Table 11-2. Reference Notes for Command Definitions

ELEMENTS OF COMMAND FORMAT										
• Leading Colon	is the prompt/command identifier character in interactive sessions. is the command identifier character in batch jobs.									
• Command Name	is shown in CAPITAL LETTERS IN REGULAR (ROMAN) TYPE, contains no blanks, is delimited by a non-alphabetic character (usually a blank)									
• Parameters	are shown in CAPITAL LETTERS IN REGULAR TYPE when they are literal information that you always enter exactly as shown. are shown in <i>lower-case italics</i> when they are variable parameters to be replaced by information that you must supply									
• Positional Parameters:	have significance implied by positional order after command name; use adjacent commas (or semicolons where required) to indicate omitted parameter(s) as follows. <table style="margin-left: 40px;"> <tr> <td>.COMMANDNAME</td> <td>p1,..p3</td> <td>(from middle of list)</td> </tr> <tr> <td>.COMMANDNAME</td> <td>.p2,p3</td> <td>(from beginning of list)</td> </tr> <tr> <td>COMMANDNAME</td> <td>p1</td> <td>(from end of list)</td> </tr> </table>	.COMMANDNAME	p1,..p3	(from middle of list)	.COMMANDNAME	.p2,p3	(from beginning of list)	COMMANDNAME	p1	(from end of list)
.COMMANDNAME	p1,..p3	(from middle of list)								
.COMMANDNAME	.p2,p3	(from beginning of list)								
COMMANDNAME	p1	(from end of list)								
• Keyword Parameters:	are separated by semicolons and can appear in any order.									
• Mixed Parameters:	positional parameters are given first; first keyword indicates end of positional list.									
• Optional Parameters	<p>[A] means "A" may be included.</p> <p>$\left[\begin{array}{c} A \\ B \end{array} \right]$ means "A" or "B" may be included.</p> <p>$\left\{ \begin{array}{c} A \\ B \end{array} \right\}$ means "A" or "B" must be included.</p> <p>$\left[\begin{array}{c} A \\ B \end{array} \right]$ means "A" and/or "B" may be included in any order.</p>									
USER/SYSTEM DIALOGUE										
• User input is <u>underlined</u> where necessary for clarity:	NEW NAME? <u>ALPHA1</u>									

:RPG

Compiles an RPG program.

SYNTAX

```
:RPG [textfile] [, [uslfile] [, [listfile] [, [masterfile] [, [newfile]]]]
```

PARAMETERS

<i>textfile</i>	Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is RPGTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
<i>uslfile</i>	<p>Actual designator of user subprogram library (USL) file on which object program is written. Can be any binary output file with file code of USL (or 1024). (Compiler's formal designator for this file is RPGUSL.) If you enter this parameter, it must indicate a file created in one of four ways:</p> <ol style="list-style-type: none">1. By saving a USL file (through the :SAVE command) created by a previous compilation where the default value was used for the <i>uslfile</i> parameter.2. By specifying a non-existent <i>uslfile</i> parameter, thereby creating a permanent file of the correct size and type.3. By creating a new USL file (through the :BUILD command, with the mnemonic USL or decimal code of 1024 used for the <i>filecode</i> parameter).4. By building the USL (through the MPE Segmenter command -BUILDUSL, discussed in <i>MPE Segmenter Reference Manual</i>). <p>If you omit <i>uslfile</i> parameter, MPE assigns as default either \$OLDPASS (if a passed file currently exists in session/job) or \$NEWPASS (if no passed file exists). (Optional parameter.)</p>
<i>listfile</i>	Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is RPGLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
<i>masterfile</i>	Actual designator of file to be optionally merged with <i>textfile</i> and optionally written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is RPGMAST.) If omitted, no merging takes place. (Optional parameter.)
<i>newfile</i>	Actual designator of file on which (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is RPGNEW.) If omitted, no text is written to <i>newfile</i> . (Optional parameter.)

NOTE

The formal file designators used in this command (RPGTEXT, RPGUSL, RPGLIST, RPGMAST, RPGNEW) cannot be back-referenced as actual file designators in the command parameter list. (For further information, see the discussion of implicit :FILE commands on page 7-16 of the MPE Commands Reference Manual.)



USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Compiles source-language program written in RPG onto a USL file on disc. If you do not specify a source text file, MPE expects input from your standard session/job input device. If you create the USL prior to compilation, you must assign it a *filecode* of USL (or 1024). If you do not specify a list file, MPE transmits the program listing to your standard session/job listing device. On the USL, each program unit exists as a relocatable binary module (RBM) that contains object code plus header information that labels and describes this code. You can compile RBMs onto the same USL during several compilations.

EXAMPLE

To compile an RPG program that you enter from your session/job input device into an object program in the USL file \$NEWPASS, and write the listing to your standard listing device, enter the following command. (If the next command is one to prepare an object program, \$NEWPASS can be passed to that command as an input file named \$OLDPASS.)

```
:RPG
```

To compile an RPG program residing on the disc file SOURCE into an object program on the USL file OBJECT, with a program listing generated on the disc file LISTFL, enter:

```
:BUILD OBJFCT; CODE=USL
```

Creates USL File OBJECT in special format.

```
:RPG SOURCE, OBJECT, LISTFL
```

Compiles program from SOURCE onto OBJECT, creating LISTFL and writing listing to it.

:RPGGO

Compiles, prepares, and executes an RPG program.

SYNTAX

```
:RPGGO [textfile] [, [listfile] [, [masterfile] [, [newfile]]]
```

PARAMETERS

<i>textfile</i>	Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is RPGTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
<i>listfile</i>	Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is RPGLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
<i>masterfile</i>	Actual designator of file to be optionally merged with <i>textfile</i> and optionally written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is RPGMAST.) If omitted, no merging takes place. (Optional parameter.)
<i>newfile</i>	Actual designator of file on which (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is RPGNEW.) If omitted, no text is written to <i>newfile</i> . (Optional parameter.)

NOTE

The formal file designators used in this command (RPGTEXT, RPGLIST, RPGMAST, RPGNEW) cannot be back-referenced as actual file designators in the command parameter list. (For further information, see the discussion of implicit :FILE commands on page 7-16 of the MPE Commands Reference Manual.)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?	YES (Suspended.)	

OPERATION

Compiles, prepares, and allocates/executes an RPG program. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you do not specify a listing file, MPE writes your listing to your standard session/job listing device. This command creates a temporary user subprogram library (USL) file (\$NEWPASS) that you *cannot* access, and a temporary program file that you *can* access under the name \$OLDPASS.

EXAMPLE

To compile, prepare, and execute an RPG program entered from your standard input device, with the program listing transmitted to your standard listing device, enter:

```
:RPGGO
```

To compile, prepare, and execute an RPG program read from the disc file SOURCE and transmit the resulting program listing to the disc file LISTFL, enter:

```
:RPGGO SOURCE, LISTFL
```

:RPGPREP

Compiles and prepares an RPG program.

SYNTAX

```
:RPGPREP [ textfile] [, [progfile] [, [listfile] [, [masterfile] [, [newfile]]]]
```

PARAMETERS

<i>textfile</i>	Actual designator of input file from which source program is read. Can be any ASCII input file. (Compiler's formal designator for this file is RPGTEXT.) If omitted, MPE assigns default designator \$STDIN (current input device). (Optional parameter.)
<i>progfile</i>	Actual designator of program file on which prepared program segments are written. Can be any binary output file with file code of PROG (or 1029). If you enter this parameter, it must indicate a file created in one of two ways: <ol style="list-style-type: none">1. By creating a new program file (through the :BUILD command, with the mnemonic PROG or decimal code of 1029 used for the <i>filecode</i> parameter).
	NOTE
	A program file is limited to only one disc extent. Thus, the <i>numextents</i> parameter of the :BUILD command must be one.
	<ol style="list-style-type: none">2. By specifying a non-existent file in the <i>progfile</i> parameter, in which case a session/job temporary file of the correct size and type is created.
	If omitted, the default file \$NEWPASS is assigned. (Optional parameter.)
<i>listfile</i>	Actual designator of file on which program listing is written. This can be any ASCII output file. (Compiler's formal designator for this file is RPGLIST.) If omitted, MPE assigns default designator \$STDLIST (typically the terminal in a session or the printer in a batch job). (Optional parameter.)
<i>masterfile</i>	Actual designator of file to be optionally merged with <i>textfile</i> and optionally written onto a file named <i>newfile</i> , as discussed in the manuals covering compilers. Can be any ASCII input file. (Compiler's formal designator for this file is RPGMAST.) If omitted, no merging takes place. (Optional parameter.)
<i>newfile</i>	Actual designator of file on which (re-sequenced) records from <i>textfile</i> (and <i>masterfile</i>) are placed. Can be any ASCII output file. (Compiler's formal designator for this file is RPGNEW.) If omitted, no text is written to <i>newfile</i> . (Optional parameter.)

NOTE

The formal file designators used in this command (RPGTEXT, RPG-PROG, RPGLIST, RPGMAST, RPGNEW) cannot be back-referenced as actual file designators in the command parameter list. (For further information, see the discussion of implicit :FILE commands on page 7-16 of the MPE Commands Reference Manual.)

:RPGPREP

(Continued)

USE

Available	In Session?	YES
	In Job?	YES
	In Break?	NO
	Programmatically?	NO
Breakable?		YES (Suspended.)

OPERATION

Compiles and prepares an RPG program onto a program file on disc. If you do not specify a source file, MPE expects your input from your standard session/job input device. If you create the program file prior to compilation, you must assign it a *filecode* of PROG (or 1029). If you do not specify a list file, MPE transmits the listing output to your session/job listing device. The user subprogram library (USL) file created during compilation is a temporary file passed directly to the preparation mechanism; you can access it under the name \$OLDPASS *only if* the program file is not \$NEWPASS. The program file is also opened as a temporary file. For both files, the Segmenter searches first for a file of the proper name in the session/job temporary file domain; if such a file cannot be found, it then searches for a permanent file of that name. If no program file of the referenced name exists, the Segmenter creates a new program file that is saved in the session/job temporary file domain and prepares into this.

EXAMPLES

To compile and prepare an RPG program entered through the session/job input device, onto the file \$NEWPASS, with the listing printed on the session/job listing device, enter the following command. (If the next command is one to execute the program, the file \$NEWPASS is referenced in the execute command under the name \$OLDPASS.)

```
:RPGPREP
```

To compile and prepare an RPG source program input from a source file named SFILE into a program file named MYPROG, with the resulting listing generated on the session/job listing device, enter the following command:

```
:RPGPREP SFILE, MYPROG
```

11-7. SAMPLE JOBS AND SESSIONS.

The following examples illustrate typical jobs and sessions. (The standard system log-on and log-off messages are not shown in these examples.)

EXAMPLES

A programmer wants to compile a source program residing on a disc file named SRC. He wishes to list this program on a disc file named LST, and transmit the compiled object code to a magnetic tape unit. The entire job could appear as the following sequence of commands:

:JOB JONES.ACT3	Initiates job.
:FILE OBJ; DEV=TAPE; SAVE; REC=, ,U	Equates OBJ File with magnetic tape unit.
:RPG SRC, *OBJ, LST	Compiles program from SRC onto OBJ (back-referencing tape unit specification), and transmits listing to LST.
:EOJ	Terminates job.

A user wants only to print a listing of his program on the standard list device (default file designator \$STDLIST). The source program is read from the standard input device (job input stream, default file designator \$STDIN); object output is transmitted to a temporary file (\$NEWPASS) that ceases to exist when the job ends. The entire job is input as follows:

:JOB SMITH.ACCT2	Initiates job.
:RPG	Compiles program, prints listing.
:	
:	
:	
(source program)	
:	
:	
:EOD	Terminates source deck.
:EOJ	Terminates job.

A user wants to merge/edit and compile program input in the disc files EDIT and OLD, print a listing on the standard list device (the default file designator \$STDLIST), save the object code in a new file named OBJ, and save the merged source code in the file NEWTXT. In session mode, he could enter:

<u>:HELLO SMITH.ACT2</u>	Initiates session.
<u>:FILE RPGTEXT=EDIT</u>	Equates RPGTEXT File with disc file EDIT.
<u>:FILE RPGMAST=OLD</u>	Equates RPGMAST File with disc file OLD.
<u>:FILE RPGNEW=NEWTXT,NEW</u>	Equates RPGNEW File with disc file NEWTXT.
<u>:FILE RPGOBJ=OBJ,NEW;SAVE</u>	Equates RPGOBJ File with disc file OBJ, and saves the file.
<u>:RPG</u>	Merges input files, and compiles program.
<u>:BYE</u>	Terminates job.

Operating the RPG/3000 Compiler

The user could also accomplish the above with the following job.

:JOB	SMITH.ACT2	Initiates job.
:FILE	RPGTEXT=EDIT	Equates RPGTEXT File with disc file EDIT.
:FILE	RPGMAST=OLD	Equates RPGMAST File with disc file OLD.
:FILE	RPGNEW=NEWTXT,NEW	Equates RPGNEW File with disc file NEWTXT.
:RPG		Merges input files and compiles program.
:SAVE	\$OLDPASS,OBJ	Saves RPGOBJ File, now re-named OBJ. (This command is explained in MPE/3000 Operating System Reference Manual.)
:EOJ		Terminates job.

11-8. COMPILER LISTING OUTPUT

The RPG/3000 Compiler outputs the following listings. When all are requested, they appear in this order:

1. Program Listing (At your option, only erroneous statements appear.)
2. Symbol Table Listing (Optional.)
3. Error Messages (If errors are detected.)
4. Cross-Reference Listing (Optional.)

Each page of printed output contains 60 lines (for printer output) or 32,767 lines (for terminal output), unless you request otherwise through the \$CONTROL compiler subsystem command discussed in Section XII. On line-printer output, the format of the heading on the first page is:

PAGE 0001 HEWLETT-PACKARD pppppv.uu.ff RPG/3000 day-of-week, month and day, year, time am/pm

In this heading, ppppp is the HP software product number assigned to the compiler; v is the version letter; uu is the present update level of the compiler; and ff is the fix-level number.

On terminal output, the heading appears in an abbreviated format, where the letters shown have the same meanings noted above:

PAGE 0001 HP pppppv.uu.ff

After the first page, the remaining pages are headed by information in the following format (on both printers and terminals):

PAGE xxxx program name title

In this format, xxxx is the page number, program name is the name of the program as specified in the Program Name Field of the Control Record Specification, and title is the title of the program, as specified in the \$TITLE or \$PAGE compiler subsystem command.

The complete listing output terminates with the following messages:

NO.SERIOUS ERRORS sss

NO. WARNINGS www

PROCESSOR TIME = h:mm:ss

ELAPSED TIME = h:mm:ss

In these messages, sss indicates the number of fatal errors detected in the program; this will be 000 if none occur. Next, www indicates the number of non-fatal errors detected; h:mm:ss indicates (in the first case) the central processor time used by your program and (in the second case) the time elapsed between the beginning and end of the program.

11-9. Program Listing

The Program Listing shows the following information:

Position	Content
1 through 4	Sequence numbers assigned automatically to all RPG/3000 specification lines and compiler subsystem commands that you enter.
5 through 8	Blank
9 through 88	Specification or comment lines, beginning with sequence numbers read from the Sequence Number Field (Columns 1 through 5) of the specification records.
89 through 92	Blank
93 through 96, 98 through 101, 103 through 106, continuing through 128 through 131	Message number, followed by W for warning message or T for terminating (fatal) error message, if the compiler detected any errors.

If errors were detected, explanatory error messages (referencing the above message numbers) will appear at the end of the Program Listing. If all indicators named in the program were not referenced and/or defined by statements, a list of the unreferenced and undefined indicators will appear.

If you are programming in an interactive session, a greater-than sign (>) will be printed before each statement as a prompt character; following this character, you enter the statement.

An example of a Program Listing appears in Figure 11-2.

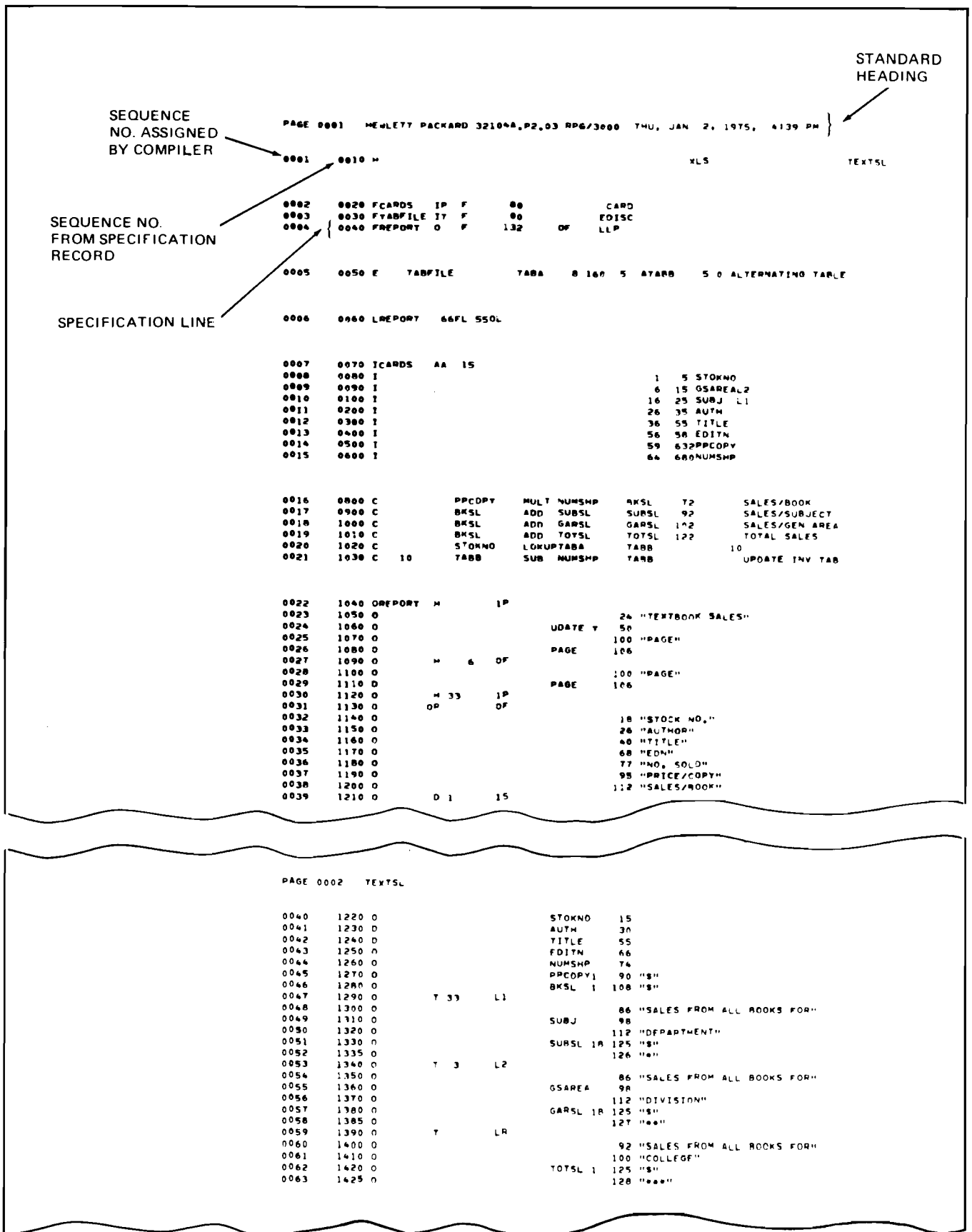


Figure 11-2. Program Listing

11-10. Symbol Table Listing

The Symbol Table Listing shows information about the various areas used by your program for data storage:

Position	Content
1 through 6	The name of the data field, table item, array element, subroutine, or TAG operation label being described.
8	Blank
9 through 12	If Columns 1 through 6 contain the name of a field, table, or array element, Columns 9 through 12 show its length: for numeric fields, the length includes a decimal point, showing the number of digits in the field, followed by the number of decimal positions; for instance, 5.2 is a five-digit field with two decimal positions. For alphanumeric fields, the length includes no decimal point. If Columns 1 through 6 name a label of a TAG operation, Columns 9 through 12 contain TAG. If Columns 1 through 6 contain a subroutine name, Columns 9 through 12 contain SUBR.
13 through 17	If Columns 1 through 6 contain the name of a field, table, or array element, Columns 13 through 17 show the address of the word in memory where the data is stored. Otherwise, Columns 13 through 17 are blank.
18 through 20	If Columns 1 through 6 contain the name of a field, table, or array element, Columns 18 through 20 show whether the data begins in the left or right byte of the word appearing in Columns 13 through 17. (L) indicates the left byte; (R) indicates the right byte. Otherwise, Columns 18 through 20 are blank.
21	The letter T (to indicate a table) or A (to indicate an array); otherwise, this field is blank.

On the listing, the symbols appear in alphabetical order. Listings that contain many symbols may show them in several groups of columns, written across the page. A sample Symbol Table Listing appears in Figure 11-3.

The diagram shows a table listing symbols with their sizes, types, and addresses. An arrow labeled 'SYMBOL NAME' points to the first column. Another arrow labeled 'TABLE NAME' points to the 'TARA' entry in the third column.

SYMBOL NAME	SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR	SYMBOL	SZ/TP	ADDR
	AUTH	10	01777(L)	GSAREA	10	01765(L)	STOKNO	5	01762(R)	TABB	5,0	00544(L)T
	BKSL	7,2	01127(L)	NUMSHP	5,0	01125(R)	SUBJ	10	01772(L)	TITLE	20	02004(L)
	EDITN	3	02016(L)	PAGE	4,0	00523(L)	SUBSL	9,2	01131(L)	TOTSL	12,2	01136(R)
	GARSL	10,2	01133(R)	PPCOPY	5,2	01124(L)	TARA	5	01142(R)T	UDATE	6,0	00537(L)

Figure 11-3. Symbol Table Listing

11-11. Error Messages

Error messages are discussed in Appendix B.

11-12. Cross-Reference Listing.

The Cross-Reference Listing shows the indicators, data areas, TAG operation labels, subroutine names, and files used by your program, and describes what statements referenced them. The listing is divided into three portions for these purposes: The Indicators Used, Field Names Used, and File Names Used Portions. Following the heading that defines each portion, a group of records is written for each item (indicator, field name, or file name) being described. The first record in the group tells where the item is defined. It appears in this format:

Position	Content
1 through 20	<p>For the Indicators Used Portion, INDICATOR nn DEFINED; nn is the indicator defined.</p> <p>For the Field Names Used Portion, the name of the field, table, array element, TAG operation label, or subroutine defined, followed by one of these descriptors: (FIELD), (TABLE), (ARRAY), (TAG), or (SUBR). The unused portion of this field remains blank.</p> <p>For the File Names Used Portion, the name of the file defined. The unused portion of this field remains blank.</p>
21 through 27	Field Length. For numeric fields, this includes a decimal point showing the number of digits in the field followed by the number of decimal positions. For alphanumeric fields, no decimal point appears.
28 through nn	One or more groups of four digits; each group is the sequence number of a specification that defines the indicator, field, table, array element, TAG label, subroutine, or file.

If your program references the item being described, one or more lines follow that show which specifications contained these references. They appear in this format.

Position	Content
1 through 8	Blank
9 through 18	R.REFERENCED
19 through nn	The sequence numbers of the specifications that reference the above indicator, field name, table, array, TAG label, subroutine, or file.

An example of a Cross-Reference Listing, showing all three portions, appears in Figures 11-4, 11-5, and 11-6.

PAGE 0003	TEXTSL	INDICATORS USED
INDICATOR 10 DEFINED	0020	
REFERENCED	0021	
INDICATOR 15 DEFINED	0007	
REFERENCED	0039	
INDICATOR L1 DEFINED	0010	
REFERENCED	0047	
INDICATOR L2 DEFINED	0009	
REFERENCED	0053	
INDICATOR LR NOT DEFINED		
REFERENCED	0059	
INDICATOR OF DEFINED	0004	
REFERENCED	0027 0031	
INDICATOR 1P NOT DEFINED		
REFERENCED	0022 0030	

Figure 11-4. Cross-Reference Listing (Indicators Used Portion)

PAGE 0004		TEXTSL	FIELD NAMES USED	
AUTH	(FIELD)	10	0011	
	REFERENCED		0041	
BKSL	(FIELD)	7,2	0016	
	REFERENCED		0017 0018 0019 0046	
EDITH	(FIELD)	3	0013	
	REFERENCED		0043	
GARSL	(FIELD)	10,2	0018	
	REFERENCED		0057	
GSAREA	(FIELD)	10	0009	
	REFERENCED		0055	
NUMSWP	(FIELD)	5,0	0015	
	REFERENCED		0016 0021 0044	
PAGE	NOT DEFINED			
	REFERENCED		0028 0029	
PPCOPY	(FIELD)	5,2	0014	
	REFERENCED		0016 0045	
STOKND	(FIELD)	5	0008	
	REFERENCED		0020 0040	
SUBJ	(FIELD)	10	0010	
	REFERENCED		0049	
SUBSL	(FIELD)	9,2	0017	
	REFERENCED		0051	
TABA	(TABLE)	5	0005	

Figure 11-5. Cross-Reference Listing (Field Names Used Portion)

PAGE 0006	TEXTSL	FILE NAMES USED
CARDS		0002
	REFERENCED	0007
REPORT		0004
	REFERENCED	0004 0022
TABFILE		0003
	REFERENCED	0005

Figure 11-6. Cross-Reference Listing (File Names Used Portion)

11-13. COMPILER INPUT/OUTPUT FILE CHARACTERISTICS

This section discusses the characteristics assigned to the input/output files used by the RPG/3000 Compiler when they are opened, and the disposition and security provisions used when they are closed. Many of these parameters can be overridden by using the MPE/3000 :FILE command, discussed previously.

11-14. Input Files

The input files used by RPG/3000 are **textfile** and **masterfile**, discussed below.

11-15. TEXTFILE. The textfile can be used in either of two ways:

1. If you do not specify a **masterfile** in a command to compile a source program, **textfile** is the primary input file that supplies both source code to be processed and compiler subsystem commands to direct the processing.
2. If you specify a **masterfile**, then **masterfile** becomes the primary input file and **textfile** is used for editing. In that case, the source code and commands from **textfile** supercede or supplement those from **masterfile**.

Operating the RPG/3000 Compiler

Textfile is always present, even if empty, but sequence checking is done only when a **masterfile** is specified; thus, if you want to perform sequence checking, you can use either **textfile** or **masterfile** for input and equate the unused file to **\$NULL** when editing is not required. For example, the following command requests sequence-checking while compiling only from the single file **SOURCE**:

```
:RPG SOURCE, , $NULL
```

Note: **\$NULL** is a non-existent “ghost” file that is always treated as an empty file. When referenced as an input file by a program, that program receives only an end-of-data indication upon first access. When referenced as an output file, the associated write request is accepted by MPE but no physical output is actually performed. Thus, **\$NULL** can be used to discard unneeded output from a running program.

Complete information on merging and editing appears in Section XII.

If the **textfile** and **listfile** are interactive with each other, the compiler prompts you for **textfile** input by printing a greater-than sign (>) on **listfile**. If these two files are duplicative, the lines input from **textfile** are counted against the page size for **listfile**.

If **textfile** input is entered through a terminal, lines exceeding 80 characters are truncated to 80, and lines less than 80 characters are padded with blanks to 80 characters.

Textfile is recognized by the formal file designator **RPGTEXT**, and is assigned the characteristics shown in Table 11-3. In this table, these characteristics are described in terms of the **FOPEN** intrinsic that opens the file and the **FCLOSE** intrinsic that closes it (in the compiler code). You can find further information about these intrinsics in the MPE reference manuals.

Table 11-3. RPGTEXT Characteristics

<u>FOPEN</u>	
Formal File Designator:	RPGTEXT
Foptions: (%2044)	
Domain:	New file (00) if not specified; old user file (11) if specified.
ASCII/BINARY:	ASCII (1)
Default File Designator:	\$STDIN (100) if specified; formaldesignator (000) if unspecified.
Record Format:	Fixed length (00)
Carriage Control:	No (0)
Label Option:	Standard label processing (0)
Disallow File Equation:	Disallow (1) if unspecified; Allow (0) if specified.
Aoptions:	
Access Type:	Input only (0000)
Multirecord Access:	No (0)
Dynamic Locking:	No (0)
Exclusive:	Default (00)
Inhibit Buffering:	No (0)
Record Size:	80 bytes.
Device Name:	MPE/3000 default.
Forms Message:	None
User labels:	None
Block Factor:	MPE/3000 default calculation.
Number of Buffers:	2
File Size:	1023
Number of Extents:	8
Initial Allocation of Extents:	1
File Code:	0
<u>FCLOSE</u>	
Disposition	No change.
Security Code:	Normal (0)

Operating the RPG/3000 Compiler

11-16. MASTERFILE. If you specify **masterfile** in your compilation command, it becomes the primary input file and is merged with the contents of **textfile** (if any) to produce the complete source code used by the compiler. If you specify this file (even when equating it to \$NULL), sequence checking of merged input records always occurs. Merging is discussed further in Section XII.

If **masterfile** and **listfile** are interactive with each other, the compiler prompts you for **masterfile** input by printing a greater-than sign (>) on the **listfile**. If these two files are duplicative, the lines input from **textfile** are counted against the page size for **listfile**, even if the lines were voided or replaced during merging/editing.

Input for **masterfile** should be entered as 80-character, fixed-length records.

The **masterfile** is recognized by the formal file designator **RPGMAST** and has the characteristics defined in Table 11-4.

Table 11-4. RPGMAST Characteristics

<u>FOPEN</u>	RPGMAST
Formal File Designator:	Old Job/User file (11).
Options (%007)	ASCII (1)
Domain:	formaldesignator (000)
ASCII/BINARY:	Fixed-length (00)
Default File Designator:	No (0)
Record Format:	Standard label processing (0)
Carriage Control:	No (0)
Label Option:	No (0)
Disallow :FILE equation:	(0)
Aoptions:	Input only (0000)
Access Type:	No (0)
Multirecord Access:	No (0)
Dynamic Locking:	Default (00)
Exclusive:	No (0)
Inhibit Buffering:	Device record size.
Record Size:	MPE/3000 default.
Device Name:	None
Forms Message:	None checked.
User labels:	MPE/3000 default calculation.
Blockfactor:	2
Number of Buffers:	1023
File Size:	8
Number of Extents:	1
Initial Allocation of Extents:	0
File Code:	
<u>FCLOSE</u>	
Disposition:	No change (0)
Security Code	Normal (0)

11-17. Output Files

The output files created by the compiler are **newfile**, **uslfile**, and **listfile**.

11-18. NEWFILE. If you specify **newfile** in a compilation command, this file will contain source program records merged and edited from **textfile** and **masterfile**, ready as a new source program for compilation. You should specify **newfile** and **listfile** as separate files, unless you equate both to the actual designator **\$NULL**; otherwise, unintelligible results will occur.

The **newfile** is recognized by the formal file designator **RPGNEW**, and has the characteristics discussed in Table 11-5.

Table 11-5. RPGNEW Characteristics

<u>FOPEN</u>	RPGNEW
Formal File Designator:	New (00)
Foptions (%404)	ASCII: (1)
Domain:	formaldesignator (000)
ASCII/BINARY:	Fixed Length (00)
Default File Designator:	No (0)
Record Format:	Standard label processing.
Carriage Control	No (0)
Label Option:	No (0)
Disallow :FILE Equation:	
Aoptions: (%001)	Output only (0001)
Access Type:	No (0)
Multirecord Access:	No (0)
Dynamic Locking:	Default (00)
Exclusive:	No (0)
Inhibit Buffering:	80 characters (bytes) (-80)
Record Size:	MPE/3000 default.
Device Name:	None
Forms Message:	None written.
User Labels:	MPE/3000 default calculations.
Block Factor:	2
Number of Buffers:	5000 records.
File Size:	8
Number of Extents:	1
Initial Allocation of Extents:	0
File Code:	
<u>FCLOSE</u>	
Disposition:	Permanent File (1)
Security Code:	Normal (0)

11-19. USLFILE. The uslfile contains the object program produced by the compiler. It is recognized by the formal file designator RPGUSL, with the characteristics described in Table 11-6.

Table 11-6. RPGUSL Characteristics

<u>FOPEN</u>	
Formal File Designator:	RPGUSL
Options:	
Domain:	If an existing file is in passed state, RPGUSL is an old temporary file in job file domain (10); if no existing file is in passed state, RPGUSL is an old file in system file domain (11).
ASCII/Binary:	Binary (0)
Default File Designator:	If an existing file is in passed state, \$OLDPASS. If no existing file is in passed state, \$NEWPASS.
Record Format:	Fixed Length (00)
Carriage Control:	No (0)
Label Option:	Standard label processing.
Disallow :FILE Equation:	If no file is specified, disallow (1). If file is specified, allow: (0)
Aoptions:	
Access Type:	Input/Output (0100)
Multirecord Access:	No (0)
Dynamic Locking:	No (0)
Exclusive:	Exclusive (0)
Inhibit Buffering:	No (0)
Record Size:	128 words.
Device Name:	MPE/3000 default.
Forms Message:	None.
User Labels:	None written.
Blockfactor:	MPE/3000 default calculations.
Number of Buffers:	2
File Size:	400 (if \$NEWPASS) or 1023 (otherwise).
Number of Extents:	2 (if \$NEWPASS) or 8 (otherwise).
Initial Allocation:	1 extent.
File Code:	1024
<u>FCLOSE</u>	
Disposition:	Permanent file (1) if \$NEWPASS. Temporary job file (2) if \$OLDPASS. No change (0) otherwise.
Security Code:	Normal (0)

11-20. LISTFILE. The compiler writes all listing output to the listfile. This output includes the program, symbol table, and cross-reference listings, input prompts (for interactive sessions), and diagnostic messages. If you assign listfile to an actual file designator other than \$STDLIST (the standard listing device), the following output is also transmitted to \$STDLIST:

1. Upon initiation of the compiler, the HP product number, version letter, update and fix levels of the compiler.
2. Upon termination of the compiler, any terminating diagnostic messages, number of error and warning messages, central processor time, and time elapsed since compiler initiation.

The listfile is recognized by the formal file designator RPGLIST. When it opens or closes this file, the compiler assigns it the following characteristics noted in Table 11-7.

Table 11-7. RPGLIST Characteristics

<u>FOPEN</u>	RPGLIST
Formal File Designator:	
Foptions: (%2514 if unspecified; %504 if specified).	
Domain:	If specified, old job/user file (11) If not specified, new file (00)
ASCII/Binary:	ASCII (1)
Default File Designator:	\$STDLIST (001) if unspecified; formaldesignator (000) if specified.
Record Format:	Variable-length (01)
Carriage Control:	Yes (1)
Label Option:	Standard label processing (0)
Disallow :FILE Equation:	Yes (1) if unspecified. No (0) if specified.
Aoptions (%001)	
Access Type:	Output only (0001)
Multirecord Access	No (0)
Dynamic Locking:	No (0)
Exclusive:	Default (00)
Inhibit Buffering:	No (0)
Record Size:	Device record size.
Device Name:	MPE/3000 default.
Forms Message:	None.
User Labels:	None written.
Blockfactor:	MPE/3000 default calculation.
Number of Buffers	2
File Size:	5000 (if device is disc).
Number of Extents:	8
Initial Allocation of Extents:	1 extent.
File Code:	0
<u>FCLOSE</u>	
Disposition:	Permanent File (1)
Security Code:	Normal (0)

11-21. USER FILE CHARACTERISTICS

The characteristics assigned to user files when they are opened by RPG-generated object programs, and the disposition and security provisions assigned when these files are closed, are summarized in Table 11-8. Some of these file characteristics can be overridden by the MPE :FILE command.

Table 11-8. User File Characteristics

<u>FOPEN</u>	
Formal File Designator:	File name or DSNAME supplied by user.
Foptions:	
Domain:	First, program attempts to open OLD (11) file; if this fails for an output file, program attempts to open it as a NEW file.
ASCII/BINARY:	ASCII (1)
Default File Designator:	Same as formal designator (000).
Record Format:	Fixed length (00) or variable length (01) as requested.
Carriage Control:	NOCCTL (0) if no carriage-control used. CCTL (1) if carriage control used.
Label Option:	Standard label processing (0)
Disallow File Equation:	Allow (0)
Aoptions:	
Access Type:	Input Files: 0 Output Files: Regular: 1 Display: 1 Chain: 2 Append Only: 3 Combined: 4 Update: 5
Multirecord Access:	No (0)
Dynamic Locking:	No (0) if no KLOCK File Description Continuation Record; otherwise, yes (1).
Exclusive:	Default (00) if no KLOCK File Description Continuation Record; otherwise, share (11).
Inhibit Buffering:	No (0)
Record Size:	As specified in File Description Specifications (Columns 24-27). Default: 80 bytes.
Device Name:	As specified in File Description Specifications (Columns 40-46). Default: DISC.
Forms Message:	If specified, RPG points to a buffer containing blanks, causing standard alignment line to be printed.
User Labels:	As specified in File Description Specifications (Column 53). Default: none.

Table 11-8. User File Characteristics (Continued)

Block Factor:	As specified in File Description Specifications (Columns 20-23)/(Columns 24-27). Default: 1.
Number of Buffers:	Default (2).
File Size:	Default (1023).
Number of Extents:	As specified in File Description Specifications (Columns 68-69). Default: 8.
Initial Allocation of Extents:	1
File Code:	0
<u>FCLOSE</u>	
Disposition	No change.
Security Code:	Normal (0)

11-22. COMPILER SECONDARY ENTRY POINTS

The compiler contains secondary entry points that provide access to optional utility procedures. These utilities are accessed by using the MPE command

```
:RUN RPG.PUB.SYS, entry-point
```

where `entry-point` is "VERSION" or "ERRORS".

The VERSION entry point lists the version number designations for the currently installed RPG compiler and run-time library. The format is as follows:

```
:RUN RPG.PUB.SYS, VERSION [ ;LIB=G  
                           P ]
```

The LIB= parameter is optional. If not specified, the RPG run-time library located in the system SL (Segmented Library in PUB.SYS) is referenced.

Specify LIB=G to reference the RPG run-time library in a group SL; specify LIB=P to reference the run-time library in an account SL (in the PUB group of the account).

The ERRORS entry point lists the compiler error messages for the currently installed compiler. The command

```
:RUN RPG.PUB.SYS, ERRORS
```

prints the errors to the default \$STDLIST device. To print errors to another device, the format is as follows:

```
:FILE RPGLIST; DEV=printerdeviceclass  
:RUN RPG.PUB.SYS, ERRORS; PARM=2
```

Refer to the MPE commands manual for discussion of DEV = on file equations.



COMPILER SUBSYSTEM COMMANDS

SECTION

XII

In general, compiler options such as source input merging, listing format specification, or warning message suppression are determined by default settings assigned by the compiler. However, you can override these settings and select different options by issuing compiler subsystem commands. These commands take effect only after access to the compiler is established. They are directed only to the compiler and are not effective during object program execution.

Compiler subsystem commands differ in both function and format from compiler language source statements — they are not RPG/3000 specifications. (However, the compiler subsystem commands accepted by RPG/3000 conform to the general formats for all other HP 3000 language translators. For each function performed by more than one translator, the same command name is used; in most cases, the same command parameters also apply. This feature helps programmers familiar with one translator subsystem to use another.)

12-1. SYNTAX AND FORMAT

In describing the syntax and format of compiler subsystem commands, these conventions are used for consistency and clarity:

1. Entries that always appear exactly as shown are designated by UPPERCASE CHARACTERS.
2. Variable entries (such as class names) are indicated by lowercase characters.
3. Optional information is indicated by [brackets]. (However, you do not enter these brackets as part of the commands.) Where one member in a group of entries may be selected and entered, the entire group is surrounded by [brackets].
4. Where one member in a group of entries must be selected and entered, the group is surrounded by {braces}.
5. An item or group of items, within brackets or braces, that may be repeated an indefinite number of times is followed by ellipses (. . .).

The general form of a compiler subsystem command is:

`[$]commandname [parameterlist]`

The first dollar-sign (\$) is required, and identifies the command as a compiler subsystem command. It must be the first character in the text portion of the record containing the command. For commands directed to RPG/3000, this dollar sign must appear in Position 6.

The second \$, optional, suppresses transmittal of the command to **newfile** (if a **newfile** is created during compilation). (For commands directed to RPG/3000, this second \$ must appear in Position 7.)

The **commandname** specifies the function requested. It follows the first \$ (or second \$, if present), with zero or more intervening spaces or comments. However, the **commandname** must be entirely contained on this record, and not on a continuation record.

Compiler Subsystem Commands

The optional **parameterlist**, if present, specifies various command options. The list is separated from the **commandname** by one or more spaces. Within the list, parameters are separated from each other by commas, optionally followed and/or preceded by spaces. The **parameterlist** may continue through Position 72 of the source record on which it appears.

The sequence field (Positions 1 through 5) of a record containing a compiler subsystem command is not part of the command; it may, however, be used for sequence-checking the record during editing and merging operations, as described later.

Note: Use only upper-case letters, and numbers and special characters in entering compiler subsystem commands; when lower-case letters are input as part of a command, the compiler interprets them as their upper-case equivalents (except when the lower-case letters are contained within character strings as defined below.)

12-2. Parameters

Within the **parameterlist**, a parameter may be any of the following four items:

- A character string.
- A symbolic name.
- A keyword.
- A keyword with a subparameter.

A **character string** consists of a quotation mark (") that denotes its beginning, optionally followed by one or more alphanumeric characters, followed by another quotation mark that terminates the string. You can include blank characters (spaces) in the string. Write quotation marks within the string as double quotation marks (two adjacent quotation marks, "") to distinguish them from the quotation marks that begin and end the string. A character string consisting only of beginning and terminating quote marks is called an **empty string**. (Note that quotation-mark delimiters in **subsystem commands** are not changed by the QUOTE=' parameter in the \$CONTROL subsystem command, discussed later — the QUOTE=' parameter only affects delimiters in RPG/3000 source-language specifications.)

A **keyword** is a reserved word (with respect to a given command) that consists of a letter followed by one or more letters and/or digits.

A **keyword with subparameter** is a keyword followed by an equal sign, followed by a subparameter consisting of a character string, a symbolic name, or a number. (A **number** consists of one or more decimal digits.) You can enter one or more spaces before and/or after the equal sign. The general format is:

keyword=subparameter

12-3. Comments

Within any command, you can also include **comments**. You generally use comments to document the purpose of coding or to make notations about program logic. A **comment** is not interpreted as part of the command, and has no effect upon compilation. It is syntactically treated as a space, and can appear in any of these locations:

- Following the **commandname**, separated from it by at least one space.
- Preceding or following any parameter in the **parameterlist**.

You cannot embed a **comment within** a parameter; for instance, you cannot enter it within a keyword or within a quoted string. Furthermore, you cannot continue a **comment** from one record to the next.

Within a comment, you can include any characters from the ASCII character set. You must begin the **comment** with two adjacent less-than signs (<<) and terminate it with two adjacent greater-than signs (>>) as delimiters. (Since adjacent greater-than signs terminate a **comment**, they cannot appear within the **comment** itself.) You can continue the **comment** through Position 72 of the record on which it appears. (This **comment** feature is provided in addition to the **comment** features of the RPG/3000 language, and applies only to subsystem command records.)

EXAMPLES

The following examples illustrate various ways in which you can include comments in compiler subsystem commands.

1. Following the command name (\$PAGE) in a command with no parameterlist.

```
$PAGE<<PAGE EJECT,NO TITLE CHANGE.>>
```

2. Following the last parameter in a parameterlist (where the comment effectively appears as a separate field).

```
$SET X1=ON,X2=ON,X3=ON <<SWITCHES 1-3 ON.>>
```

3. Embedded within the parameterlist (preceding the last parameter):

```
$SET X1=ON,X2=ON, <<LAST SW OFF.>> X3=OFF
```

12-4. Continuation Records

When the length of a command exceeds one physical record (source card or entry line), you can enter an ampersand (&) as the last non-blank character of this record and continue the command on the next record (called a **continuation record**). You must begin the text portion of the continuation record, in turn, with a dollar sign in Position 6. (Even when a command begins with the double dollar sign (\$\$), its continuation records still begin only with a single dollar sign.)

Note: A subsystem command record must never be separated from its continuation record by an RPG/3000 source record (specification); nor must an RPG/3000 source record be separated from its continuation record by a subsystem command record.

Compiler Subsystem Commands

In continuing a command onto another record, you cannot divide a primary command element (a command name, keyword, subparameter (including quoted strings), or comment) — no primary element is allowed to span more than one line.

When the compiler encounters a command containing one or more continuation records, it concatenates each continuation record (beginning with the character following the \$) to the preceding record; it also replaces each \$ and continuation ampersand by a space.

EXAMPLE

The programmer continues the following \$CONTROL command onto a second record:

```
$CONTROL LIST, SOURCE, WARN, MAP, &  
$CODE, LINES=36.
```

The compiler interprets these two records as:

```
$CONTROL LIST, SOURCE, WARN, MAP, CODE, LINES=36
```

Even though you cannot divide a comment over more than one line, you can enter extensive commentary text requiring several lines by enclosing it within separate comments that each occupy one line.

EXAMPLE

The following \$CONTROL command includes commentary text spread over three lines.

```
$CONTROL NOWARN <<WARNING MESSAGES ON TRIVIAL ERRORS>>&  
$ <<WILL NOT BE LISTED. BUT MESSAGES ON>>&  
$ <<FATAL ERRORS WILL APPEAR.>>
```

12-5. Effects of Commands

A command does not take effect until all of its parameters have been interpreted. Thus, a command that suppresses source listing output will not affect the listing of any continuation records within the command itself. Parameters are interpreted from left to right. In some cases, parameters may be redundant or supercede previous parameters within the same command. In other cases, certain parameters are allowed only once within a command.

EXAMPLE

In the following \$CONTROL command, the redundant parameters LIST and NOLIST each appear twice:

```
$CONTROL LIST, NOLIST, NOLIST, LIST
```

Because the final redundant parameter in any \$CONTROL command always takes effect, the above command is equivalent to:

```
$CONTROL LIST
```

12-6. COMMAND SUMMARY

A summary of the compiler subsystem commands for RPG/3000 appears in Table 12-1. (Only the commandnames are shown; the parameterlists are described later.)

Table 12-1. Compiler Subsystem Command Summary

COMMAND	PURPOSE
\$CONTROL	Restricts access to listfile; suppresses source text, object code, cross-reference listing of defined indicators, fields, tables, arrays, and files; suppresses warning messages; sets maximum number of lines listed per page; sets maximum number of severe errors allowed; establishes delimiting character for non-numeric literals; limits segment size allowed for object program.
\$COPY	Invokes preprocessing of source code to expand \$INCLUDE statements.
\$INCLUDE	Merges lines from an external file into the current source program.
\$IF	Interrogates software switches for conditional compilation.
\$SET	Sets software switches for conditional compilation.
\$TITLE	Establishes or changes page title on listing.
\$PAGE	Establishes or changes page title, and ejects page.
\$EDIT	Specifies editing options during merging (omitting sections of old source program and/or re-numbering sequence fields).

12-7. LISTING AND COMPILATION OPTIONS (\$CONTROL COMMAND)

When you invoke the compiler without specifying compiler subsystem commands, several listing control, error message, and object-file formulation options take effect by default:

1. The compiler is given unrestricted access to listfile.
2. All source records are listed unless the listfile and primary input file (normally the textfile) are assigned to the same terminal.
3. Warning messages are listed.
4. Cross-reference listing of defined indicators, fields, tables, arrays, and files is suppressed.
5. Listing of the object code generated is suppressed.
6. The number of lines appearing on each printed page (output to listfile) is a maximum of 60.
7. The maximum number of severe errors allowed before compilation is terminated is 100.
8. The quotation mark (") is recognized as the character used to delimit non-numeric literals in RPG/3000 specifications.
9. The segment size allowed for the object program is limited to 4096 words.

Compiler Subsystem Commands

You can override the above default options by entering the `$CONTROL` compiler subsystem command. This command allows you to restrict the compiler's access to the `listfile`; suppress source record listings; produce object code and symbol table listings; re-specify the maximum number of lines per printed page; and otherwise alter the normal compiler control options. The format of the `$CONTROL` command is:

```
[$]CONTROL parameterlist
```

Each parameter in the `parameterlist` specifies a different option: the options are described below. Unless otherwise noted, each parameter can appear in a `$CONTROL` command placed anywhere in the source input. Each parameter remains in effect until explicitly cancelled by an opposing parameter (for example, `NOLIST` cancelling `LIST`), or until access to the compiler terminates. In any `$CONTROL` command, the `parameterlist` (containing at least one parameter) is always required. Within the `parameterlist`, the parameters can appear in any order. In the descriptions below, default parameters appear in boldface type.

Parameter	Option Requested
<code>LIST</code>	Allows the compiler unrestricted access to <code>listfile</code> , permitting the <code>SOURCE</code> , <code>MAP</code> , <code>CODE</code> , and <code>LINES</code> listing parameters described below to take effect when issued. The <code>LIST</code> parameter remains in effect until a <code>\$CONTROL</code> command specifying the <code>NOLIST</code> parameter (described below) is encountered. When neither <code>LIST</code> nor <code>NOLIST</code> is specified at the beginning of the compilation, <code>LIST</code> takes effect by default.
<code>NOLIST</code>	Allows only source records that contain errors, appropriate error messages, and subsystem initiation and completion messages to be written to the <code>listfile</code> . <code>NOLIST</code> remains in effect until a <code>\$CONTROL</code> command specifying <code>LIST</code> appears.
<code>SOURCE</code>	Requests listing of all source records input (as edited by the compiler's editor) while <code>LIST</code> is also in effect. When you invoke the compiler with <code>listfile</code> and the primary input file assigned to the same terminal, <code>NOSOURCE</code> is initially the default. In all other cases, <code>SOURCE</code> is initially the default.
<code>NOSOURCE</code>	Suppresses the listing of source text, cancelling the effect of any previous <code>SOURCE</code> parameter. <code>NOSOURCE</code> remains in effect until <code>SOURCE</code> is subsequently encountered.
<code>WARN</code>	Permits the reporting of doubtful minor error conditions in the source input. These reports are transmitted to <code>listfile</code> in the form of warning messages. The <code>WARN</code> parameter remains in effect until a <code>\$CONTROL</code> command specifying the <code>NOWARN</code> parameter (described below) is encountered. When you specify neither <code>WARN</code> nor <code>NOWARN</code> , <code>WARN</code> takes effect by default.
	Note: <code>NOLIST</code> does not suppress warning messages — they are suppressed solely by <code>NOWARN</code> .
<code>NOWARN</code>	Suppresses warning messages; cancels the effect of any previous <code>WARN</code> parameter. The <code>NOWARN</code> parameter remains in effect until a <code>\$CONTROL</code> command specifying <code>WARN</code> appears.

Parameter	Option Requested
MAP	Requests printing of cross-reference listing of defined indicators, fields, tables, arrays and files following the listing of the source text (if LIST is in effect). The MAP parameter remains in effect until the NOMAP PARAMETER (described below) is encountered. When you specify neither MAP nor NOMAP at the beginning of the compilation, NOMAP is assumed by default.
NOMAP	Suppresses printing of cross-reference listing of defined indicators, fields, tables, arrays, and files, cancelling effect of any previous MAP parameter. The NOMAP parameter remains in effect until MAP is again encountered.
CODE	Requests listing of object code generated following the listing of the source text (if LIST is in effect). The CODE parameter remains in effect until the NOCODE parameter (described below) is encountered. When you specify neither CODE nor NOCODE, NOCODE is assumed by default.
NOCODE	Suppresses listing of object code, cancelling effect of any previous CODE parameter. The NOCODE parameter remains in effect until a CODE parameter is again encountered.
LINES=nnnn	Limits lines printed on listfile to nnnn lines per page. Whenever the next line sent to listfile would overflow the line count (nnnn), the page is ejected and the standard page heading and two blank lines are printed at the top of the next page, followed by the line to be transmitted. (A page heading and its following two blank lines are counted against the total line count, nnnn.) The subparameter nnnn is an integer ranging from 10 to 9999. The LINES=nnnn parameter remains in effect until another LINES=nnnn parameter appears. If you omit this parameter, the default value assigned is: <p style="margin-left: 40px;">60 lines per page, for listing output through devices other than terminals.</p> <p style="margin-left: 40px;">32767 lines per page, for listings output through terminals.</p>
ERRORS=nnn	Sets the maximum number of severe errors allowed during compilation to nnn; if this limit is exceeded, compilation terminates and the uslfile is unchanged. (If the limit specified has already been exceeded when this ERRORS=nnn parameter is encountered, compilation terminates at this point.) If you omit the ERRORS=nnn parameter, nnn is set to 100 by default.
USLINIT	Initializes the uslfile to empty status prior to generation of object code. If you do not specify a uslfile (and it is thus supplied by the compiler through default), and if \$OLDPASS is not a uslfile , or if you supply a uslfile whose contents are obviously incorrect, the compiler automatically initializes the uslfile to empty status whether or not USLINIT is specified.

Parameter	Option Requested
QUOTE={“;}	Defines the character to be used for delimiting non-numeric literals in the following source text. This character may be either a quotation mark (“) or an apostrophe ('). If you do not specify the QUOTE parameter, the quotation mark is assumed. You can enter the QUOTE parameter anywhere in the source input; typically however, you enter it near the beginning of the program.
SEG=n	Limits the segment size allowed for the object program to n x 1024 words. For instance, SEG=3 limits generated segment size to 3,072 words. Set n to 1, 2, 3, or 4 only. If this parameter is not used or is incorrect, n is set to 4.
Note:	The SEGMENT= parameter of the \$CONTROL command, used by some compilers to assign different segment names to program units, is not supported by RPG because of its limited application with this compiler — most RPG programmers do not interface with the system at the segment level because it is difficult to relate object code generated to original source statements.
NAME=source	Assigns the “Source” program name, superceding any entry made in the Header spec columns 75-80. “Source” must be a valid RPG program name as defined in paragraph 2-6.
RSPACE=n	Adds additional spaces between output fields for which relative end positions have been specified. The default RSPACE value is zero.

EXAMPLES

The following \$CONTROL command requests unrestricted access to listfile; listing of all source text, symbol table information, and object code; suppression of warning messages (but not error messages); and use of the apostrophe as a delimiter for non-numeric literals. By default, the maximum number of lines per printed page is limited to 60, the maximum number of errors allowed is 100, the usfile supplied is not initialized to empty status, and the size of segments for the object program is limited to 4096 words.

```
$CONTROL LIST, SOURCE, MAP, CODE, NOWARN, QUOTE='
```

The following \$CONTROL command illustrates the default values for the command parameters; it produces the same effect as if no \$CONTROL command were entered:

```
$CONTROL LIST, SOURCE, WARN, NOMAP, NOCODE, LINES=60, ERRORS=100, QUOTE='
```

12-7A. COPYLIB PREPROCESSING (\$COPY COMMAND)

Before normal execution of the RPG compiler, you might want to temporarily modify your source program with records retrieved from other files (called *copylib* files). These files are named in your source program on \$INCLUDE statements, and require “preprocessing” in order to be included. This preprocessing step must be explicitly requested by a \$COPY command.

The format of the \$COPY command is:

```
*COPY
```

where the “*” is specified in column 6.

Refer to Section XIV for a complete discussion of the \$COPY and \$INCLUDE preprocessing commands.

12-7B. COPYLIB FILE NAMING (\$INCLUDE COMMAND)

Sometimes you might need a set of RPG source specifications (such as the Input specs for a specific file) in several programs. Instead of entering or copying the same specs in all of the programs, you can keep the specs in a single *copylib* file and instruct the RPG compiler to merge that file into each program. This is done by naming the copylib file on a \$INCLUDE command in each program requiring the specs.

The format of the \$INCLUDE command is:

```
$INCLUDE filename
```

where the "\$" is specified in column 6 and *filename* is the copylib file name.

Refer to Section XIV for a complete discussion of the \$INCLUDE preprocessing command.

12-8. CONDITIONAL COMPILATION (\$IF COMMAND)

Generally, when you submit a program to the compiler, you want to compile the entire program. but, you may occasionally wish to compile only one or more portions of the program. You can request such **conditional compilation** by delimiting the source code to be compiled (or omitted from compilation) with a series of \$IF compiler subsystem commands. These \$IF commands interrogate any of ten compiler toggle switches, named X0 through X9, inclusive. (These switches are set on or off by the \$SET compiler subsystem command described later.) Thus, a \$IF command can direct the compiler to compile or ignore all source code between this \$IF command and the next \$IF command encountered, if a particular relation is true or false.

The format of the \$IF command is:

$$\$[\$]IF [X_n = \left\{ \begin{array}{l} OFF \\ ON \end{array} \right\}]$$

The parameters and their resultant options are:

Parameter	Option
X _n	The letter X, followed by a digit (n) from 0 to 9 that specifies the name of the switch to be tested — for example, X3. Spaces between X and n are not allowed.
OFF } ON }	The state that the switch is to be tested for in determining if the statements following the \$IF command are to be compiled. If the relation is false (the switch is not in the state specified by this parameter), the following source records (except \$EDIT, \$PAGE and \$TITLE commands) are ignored until another \$IF command is encountered. If the relation is true , succeeding source records are compiled normally.

You can enter a \$IF command anywhere in the source text. The appearance of a \$IF command always terminates the influence of any preceding \$IF command. When you enter a \$IF command but do not include a parameter list, the following text is compiled in the normal way but the effect of any previous \$IF command is cancelled. Regardless of whether conditional compilation occurs or is suspended as a result of a \$IF command,

1. \$EDIT, \$PAGE, \$TITLE and \$IF commands within the range of this \$IF command are interpreted and executed.
2. Normally-listable source text (regardless of whether or not it is compiled) is listed if the \$CONTROL command options LIST and SOURCE are in effect.

The \$IF command does not affect the textfile-masterfile merging operation and transmission of merged/edited text to the newfile. (Merging and editing are described in the discussion of the \$EDIT command.)

An example illustrating use of the \$IF command appears in the discussion of the \$SET command below.

12-9. SOFTWARE SWITCHES FOR CONDITIONAL COMPILATION (\$SET COMMAND)

When the compiler is invoked, all ten software toggle switches are initially turned off. You can turn them on (and off again) by using the \$SET command.

$$\$[\$]SET [X_n=\begin{Bmatrix} OFF \\ ON \end{Bmatrix} [, X_n=\begin{Bmatrix} OFF \\ ON \end{Bmatrix}] \dots]$$

The parameters and options are:

Parameter	Option
X _n	The letter X followed by a digit (n) from 0 to 9 that specifies the name of the switch to be set.
$\begin{Bmatrix} OFF \\ ON \end{Bmatrix}$	The state to which the switch is to be set. Specify either OFF or ON (but not both) for each X _n parameter.

You can enter a \$SET command anywhere in the source text. If the compiler encounters a \$SET command that contains no parameter list, all ten switches are turned off. If more than one parameter appears in a \$SET command, each parameter must be separated from its predecessor by a comma.

EXAMPLE

In the following source text, switches X4 and X5 are set on and interrogated, with the results indicated by the comments:

```

      .
      .
$SET  X4=ON, X5=ON  <<SETS SWITCHES X4 AND X5 ON.>>
      .
      .
$IF   X4=ON        <<REQUESTS COMPILATION OF SOURCE BLOCK 1.>>
      .
      .
      (SOURCE BLOCK 1)
$IF   X5=OFF      <<REQUESTS THAT SOURCE BLOCK 2 BE IGNORED>>&
$     .           <<BY CANCELLING PREVIOUS $IF COMMAND.>>
      .
      .
      (SOURCE BLOCK 2)
      .
      .
$IF   .           <<CANCELS PREVIOUS $IF COMMANDS SO THAT>>&
$     .           <<SOURCE BLOCK 3 IS COMPILED.>>
      .
      .
      (SOURCE BLOCK 3)

```

12-10. PAGE TITLE IN STANDARD LISTING (\$TITLE COMMAND)

On each page of output listed during compilation, a standard heading appears. Positions 29 through 132 of this heading are reserved for a title (usually describing the page's content), optionally specified with the \$TITLE compiler subsystem command.

```
$[$] TITLE [string [,string] . . . ]
```

Each string parameter is a character string (bounded by quotation marks) that is combined with any other strings specified to form the title. In forming the title, the compiler strips the strings of their delimiting quotation marks, and then concatenates them from left to right. The entire parameter list can specify up to 104 characters, including spaces within the string but excluding delimiters and spaces between the strings

If the title contains fewer than 104 characters, the unused portion is filled to the right with spaces. If no string parameters are present in the \$TITLE command, or if you enter no \$TITLE command (or \$PAGE command with title specification, discussed below), the title portion of the heading is blank. When a new \$TITLE command is encountered, it supersedes any previously specified title from that point on.

When the compiler reads a \$TITLE command and the NOLIST parameter of the \$CONTROL command is in effect, title specification or replacement occurs even when the \$TITLE command appears within the range of an \$IF command whose relation is evaluated as false.

EXAMPLE

Consider the following \$TITLE command, occupying two lines of code:

```
$TITLE "THE PROGRAM TITLE IS", &  
$"RUN III."
```

This command results in the following entry in the title field of the standard page heading:

```
THE PROGRAM TITLE IS RUN III.
```

12-11. PAGE TITLE AND EJECTION (\$PAGE COMMAND)

You can specify a program title (as with the \$TITLE command), together with page ejection by entering the \$PAGE command. This allows varied listing formats. For example, you can list individual sections of a program starting on a new page, and assign to each section its own descriptive title. The \$PAGE command format is:

```
[$]PAGE [string[,string] . . . ]
```

Each string parameter has the same format, meaning, result, and constraints as in the \$TITLE command. If no parameter is present in the \$PAGE command, the current title (assigned by a previous \$TITLE or \$PAGE command) or a blank title (if no previous \$TITLE or \$PAGE command occurred) remains in effect.

Following title specification or replacement, if the LIST parameter of the \$CONTROL command is in effect, this action occurs:

1. The current page is ejected.
2. The standard page heading (including the new title) is printed, followed by two blank lines.

If no string was specified in the \$PAGE command and LIST is in effect, the current page is ejected and the standard page heading (including the old title) is printed, followed by two blank lines.

If LIST is not in effect, specified title replacement occurs, but no printing or page eject takes place. Any new title will appear, however, after LIST is requested.

The \$PAGE command itself is never listed.

12-12. SOURCE TEXT MERGING AND EDITING (\$EDIT COMMAND)

You can request the following merging and editing operations:

1. Merge corrections or additional source text (on **textfile**) with an existing source program and commands (on **masterfile**) to produce a new source program and commands. (Figure 12-1). This new input is compiled and optionally copied to **newfile**, which can be saved for re-use through an MPE/3000 :FILE command.
2. Check source-record sequence numbers for ascending order.
3. Omit sections of the old source program during merging.
4. Re-number the sequence fields of the records in the new, merged source program.

The editing done by the compiler is limited to linear source text modification. Extensive or more sophisticated editing is possible with the HP 3000 Text Editor, EDIT/3000.

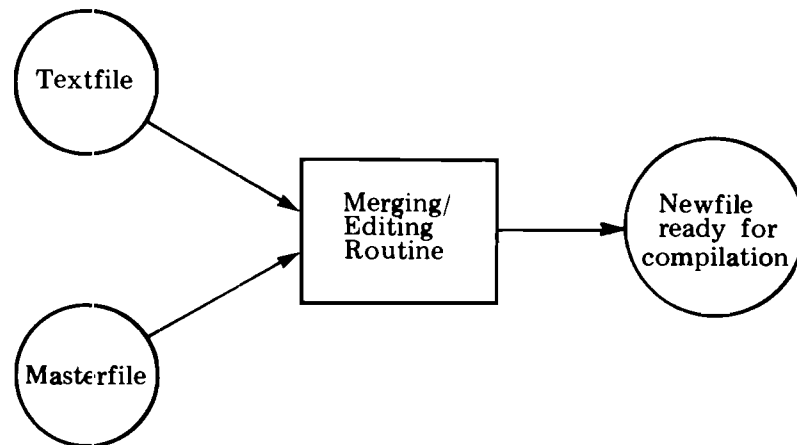


Figure 12-1. Creating a New Source File By Merging and Editing

12-13. Merging

To request merging, simply equate actual file names to the **textfile**, **masterfile**, and (optionally) **newfile** formal designators (RPGTEXT, RPGMAST, and RPGNEW, respectively). Do this by using the MPE/3000 :RPG command when the compiler is invoked.

An example appears on the following page.

EXAMPLE

To specify merging of a textfile TFILE with a masterfile MFILE, you could enter the following :RPG command. The merged source text is copied to the newfile NFILE, with the object code and listing output written to the default files \$NEWPASS and \$STDLIST, respectively.

```
:RPG TFILE, , MFILE, NFILE
```

Prior to merging, the records in both textfile and masterfile must be arranged in ascending order as dictated by their sequence fields — that is, the value of the sequence field on any record — or their sequence fields must be blank. (The order of sequencing is based on the ASCII Collating Sequence.) There are no restrictions regarding blank sequence fields — the sequence fields of some or all of the records in either textfile, masterfile, or both files can be blank, and such records can appear anywhere in either file.

The merging operation is also based on ascending order of sequence fields, according to the ASCII Collating Sequence. During merging, the compiler checks the sequence fields of the records in both files for ascending order. If their order is improper, the compiler skips the offending records during merging and transmits appropriate diagnostic messages to listfile. During each comparison step in merging, the compiler reads one record from each file, and compares these records, with one of three results:

1. If the values of the sequence fields of the masterfile and textfile records are equal, then the textfile record is compiled and (optionally) passed to newfile; the masterfile record is ignored; and one more record is read from each file for the next comparison.
2. If the value of the sequence-field of the masterfile record is less than that of the textfile record, the masterfile record is compiled and (optionally) passed to newfile; the textfile record is retained for comparison with the next masterfile record.
3. If the value of the sequence field of the textfile record is less than that of the masterfile record, the textfile record is compiled and (optionally) passed to newfile; the masterfile record is retained for comparison with the next textfile record.

During merging, the compiler assumes that a record with a blank sequence field has the same sequence field as that of the last record with a non-blank sequence field read from the same file (or a null sequence field, if no record with a non-blank sequence field has yet been encountered in the file). Thus, a group of one or more records with blank sequence fields residing on masterfile are never replaced by records from textfile; they can only be deleted through use of the \$EDIT command, as explained later.

Records from masterfile that are replaced during merging and thus neither compiled nor sent to newfile are not listed during compilation.

When the compiler encounters the end-of-file on either textfile or masterfile, merging terminates (except for the continuing influence of an unterminated VOID parameter in an \$EDIT command, as discussed later). At this point, the compiler checks the subsequent records on the remaining file for proper sequence, compiles them, and (optionally) passes them to newfile. (However, masterfile records within the range of a VOID parameter are neither compiled nor sent to newfile.)

The merging operation does not normally change the sequence field values of records transmitted to newfile. However, you can request the assignment of new sequence characters by using the \$EDIT command.

12-14. Checking Sequence Fields

The presence of a masterfile during compilation implicitly requests checking of source records for proper sequence. Thus, when you specify both **textfile** and **masterfile** as input files for the compiler, or when you specify **masterfile** alone, sequence-checking is done automatically on both files. But when you specify **textfile** as the only input file, sequence checking does not occur. Therefore, when you want source input sequence-checked but do not require merging of two input files, you can transmit the input from either the **textfile** or the **masterfile** and equate the unused file to \$NULL.

EXAMPLE

You could compile a program from the **textfile** SOURCE (with no **masterfile** input) and implicitly request sequence checking with the following command:

```
:RPG SOURCE, , , $NULL
```

12-15. Editing

Editing operations during merging enable you to omit sections of the old source program (residing on **masterfile**) and/or re-number the sequence fields of the new, merged source program (residing on **newfile**). Both of these operations are requested through the \$EDIT command, written in the following format:

```
[$]EDIT [VOID=sequencevalue]
        [ ,SEQNUM=sequencenumber ]
        [ ,NOSEQ ]
        [ ,INC=incnumber ]
```

The parameters and options are as follows; you can specify the parameters in any order.

Parameter	Option
VOID=sequencevalue	<p>Requests the compiler to bypass (during merging) all records on masterfile whose sequence fields contain a value less than or equal to sequencevalue, plus any subsequent records with blank sequence fields. This request remains in effect until a masterfile record with a sequence-field value higher than sequencevalue is encountered. The VOID request is initially disabled when the compiler is invoked.</p> <p>The sequencevalue subparameter can be either a legal sequence number or a character string. If sequencevalue is less than five characters, RPG/3000 left-fills sequence numbers with ASCII zeros and sequence character strings with spaces.</p>

Parameter	Option
SEQNUM=sequencenumber	<p>Requests re-numbering of the merged source records on newfile, beginning with the value specified by the sequencenumber subparameter; this value replaces the sequence number of the next record sent to newfile. The sequence number of each succeeding record is incremented according to the value specified by the INC parameter (or its default), described below. If the SEQNUM=sequencenumber parameter is present but newfile does not exist, the compiler ignores the re-numbering request. If this parameter is present, and newfile exists, the re-numbering request remains in effect until an \$EDIT command with the NOSEQ parameter is encountered. When the merged output is listed, records actually transmitted to newfile appear with the new sequence numbers but records not sent to newfile appear with blank sequence fields. The re-sequencing request is initially disabled when the compiler is invoked.</p> <p>The sequencenumber subparameter can be a legal sequence number of one to five digits. If less than five digits, RPG/3000 left-fills this value with ASCII zeros.</p>
NOSEQ	<p>Suspend re-numbering of merged records on newfile; current sequence numbers are retained. If you specify neither SEQNUM nor NOSEQ, then, NOSEQ takes effect by default until superseded by SEQNUM.</p>
INC=incnumber	<p>Sets increment by which records sent to newfile are re-numbered if SEQNUM is in effect. The increment is specified by incnumber, which can be a value ranging from 1 through 99999. Notice, however, that very large increments are of limited value, since they may cause the five-digit sequence number to overflow. Re-numbering only occurs if SEQNUM is specified (or if the last parameter is not overridden by a NOSEQ parameter) and a newfile exists. If SEQNUM is specified but INC is not, the sequence number is incremented by the default value of 10 for each succeeding record; this default value applies until the compiler encounters an INC parameter specifying a new value.</p>

You normally enter **\$EDIT** commands from **textfile**. (Their input from **masterfile** is allowed, but is not recommended since any **\$EDIT** command containing a **VOID** parameter on **masterfile** could void its own continuation records.) **\$EDIT** commands themselves are never sent to **newfile**; thus, the command form **\$\$EDIT**, while permitted, is redundant.

While sequence fields are allowed (and usually necessary) on records containing **\$EDIT** commands, continuation records for such commands should have blank sequence fields.

During merging, a group of one or more **masterfile** records with blank sequence fields are never replaced by lines from **textfile**; you can only delete them by using an **\$EDIT** command with a **VOID=sequencevalue** parameter at least as great as the last non-blank sequence field preceding the group. In this case, the compiler deletes the entire group of **masterfile** records with blank sequence number fields.

Since voided records are never passed to the **uslfile** or **newfile**, their sequence is never checked, and they never generate an out-of-sequence diagnostic message.

A **VOID** parameter does not affect records in **textfile**.

Any **masterfile** record replaced by a **textfile** record is treated as if voided, except that following records with blank sequence fields are not also voided. If a replaced record would have been out-of-sequence, the **textfile** record that replaces it produces an out-of-sequence diagnostic message.

In general, whenever a record sent to **newfile** has a non-blank sequence field lower in value than that of the last record with a non-blank sequence field, the compiler prints a diagnostic message.

EXAMPLE

Suppose a programmer wants to merge text input from the standard input device (default value for **textfile**, designated by **\$STDIN**) with an old program on the file **OLDPROG**, creating new source input on the file **NEWPROG**. He wants to re-number the merged source records on **NEWPROG** beginning with the value 50, incrementing the sequence number of each subsequent record by 10. After logging on, he enters:

```

:RPG  , , , OLDPROG, NEWPROG
.
.
.
$EDIT SEQNUM=50, INC=10  <<SPECIFIES EDITING PARAMETERS.>>
.
.
.
(New text or corrections to be merged with old program.)
.
.
.

```



OVERVIEW

The RPG interface to V/3000 provides functions similar to those provided by the V/3000 procedures for other languages. Since RPG does not make it easy to call the V/3000 procedures directly, the interface is designed to operate in a manner familiar to RPG programmers. Using the interface, the RPG programmer can:

- Get a form from the forms file and display it at the terminal.
- Display a message in window area of the terminal screen.
- Display initial values specified through FORMSPEC in fields displayed on the screen.
- Accept input from the terminal.
- Determine if the input contains errors and, if so, flag the fields in error and display an error message.
- Write data accepted from the terminal to the user program or to a batch file.
- Transfer data from the user program or a batch file to the terminal screen.
- Transfer data between the user program and a batch file.

These features provide a complete interface between a user program, the terminal, a forms file, the entered data, and, for data entry, the batch file to which entered data is written.

Note that appendix A of the VPLUS/3000 Reference Manual contains a listing of the RPG version of program ENTRY. Since program ENTRY controls standard data entry using V/3000, the RPG interface described in this section need be used only to modify ENTRY to suit particular applications, or to provide an interface between an existing application and the facilities provided by FORMSPEC.

OPERATION OF THE V/3000 INTERFACE

So that the RPG interface to V/3000 appears natural to RPG programmers, all input to, output from, and commands that control the data entry facility are treated as input/output to a file. This file is identified by the special device class name:

WORKSTN

In effect, you request the interface to V/3000 by specifying "WORKSTN" in the device field of a File Description Specification. "WORKSTN" is *not* a system configured name.

WORKSTN FILE

The file assigned to this device can have any legal file name, but must be specified as an Update or Combined file. (WORKSTNC can be **any** file type.) The file is usually designated as a demand file, but may be a primary file. RPG executes update primary files within the normal RPG cycle. For example, an update primary file might be

V/3000 Interface

used when an application is limited to a single cycle such as showing a form, reading data entered on the form, and writing the data to a file. For most applications, however, the WORKSTN file is specified as update demand. For update demand files, the program stays within the calculation section, and uses demand reads and exception output.

There may be only one file per program assigned to the device file WORKSTN. This file must be defined with variable length records.

OTHER FILES

Three other files are specified as continuation files in the same File Description Specification. These three files are:

- Forms file — contains the forms designed through program FORMSPEC. (File is required.)
- Batch file — file to which data entered on form is written during standard V/3000 data entry. (May be omitted.)
- Trace file — file that contains a record of every action and event during execution of RPG interface, and on which runtime errors are recorded. (May be omitted.)

ACTIONS AND EVENTS

The operation of the RPG V/3000 interface is specified in terms of “actions” and “events”. Actions are output records written to the file; they carry out all the V/3000 functions. Events are input records read from the file; they specify input from the terminal, or from the V/3000 buffer.

Actions include, but are not restricted to:

- Get next form to display
- Display form and data
- Read input from terminal
- Write data buffer to batch file

Events include:

- ENTER key was pressed
- Function key was pressed
- V/3000 editing error occurred

RUNTIME ERRORS

RPG allows the programmer to specify the action to take in case of a runtime error. The action can be one of six options specified on the Control Record Specification. If no option is specified, the system asks the operator to specify an option by means of a function key.

A stack dump is provided as one of the RPG error options. Since the WORKSTN programs are run as sessions, the program can redirect the dump to a file. If no file is specified, the default file is \$STDLIST.

A further aid is the trace file. When runtime problems occur, the trace file can be used to determine where the problem occurred.

Error messages are displayed at the terminal unless suppressed. Each message is displayed for three seconds unless this interval is increased or decreased.

BREAK KEY DISABLED

On many 264x terminals, the break key is physically positioned near the ENTER key. As a result, it is easy for the terminal operator to press the BREAK key by accident. Since it is difficult to recover from a break, the BREAK key is disabled when a WORKSTN file is used.

If you feel you need the BREAK key, you can enable it with a notation on the File Description Specification for the WORKSTN file. (Refer to the discussion of "BREAK KEY ENABLE" on page 13-14 for details.)

FORMS DOWNLOADING

For terminals that allow local forms storage (such as the HP2424B and HP2626A/W), you can request that VPLUS forms be downloaded to the terminal memory to achieve faster screen displays. Do this by specifying a Continuation record for the WORKSTN File Description spec, entering "FORMDL" in columns 54-59, and the number of forms to download (1-255) **right justified** in columns 60-62.

USING THE V/3000 INTERFACE

All actions and events have a two-digit code. These codes are integers in the range 00 through 99. Codes from zero through 49 denote events; codes from 50 through 99 denote actions. Events codes are specified in columns 1-2 of each record.

Any numeric action code can be replaced by a six-character mnemonic in columns 1-6 of the output record. The mnemonic differs from the numeric code only in that it is easier to remember. On the other hand, performance may be improved by using numeric codes rather than the action names.

Events and actions are treated programmatically as input and output records. Events can be identified by record indicators on Input Specifications. Actions are specified as fields on Output Specifications.

DATA ENTRY THROUGH RPG

Data Entry using V/3000 is essentially a three step process:

1. Enter data at the terminal.
2. Transfer the data to the V/3000 buffer; perform FORMSPEC edits.
3. Transfer data from the V/3000 buffer to a batch file or to a user program buffer.

Output to the terminal reverses these steps.

If data editing is specified by FORMSPEC, this editing is performed on the data in the V/3000 buffer. Since many RPG applications may not use the FORMSPEC editing features, the RPG interface is designed so that the application can skip step 2, and write data directly to a user program, or vice versa.

EVENTS

The user program requests the return of an event by performing a read operation on the WORKSTN file. Event codes indicate the type of input passed to WORKSTN. The input is either received from the terminal or is in response to a particular action code. Input data is transferred to the user program according to the WORKSTN file input specifications.

Events 00 through 08 indicate which terminal key was pressed by the operator. If ENTER was pressed (event 00), then data has been entered and can be read. If any function key was pressed (events 01-08), the RPG programmer must specify the action to be performed in response.

The remaining event codes respond to actions specified in the RPG program. These actions involve the V/3000 data buffer, so event codes greater than 08 will probably be used only by those applications that take full advantage of the V/3000 data handling features.

All event records contain the name of the current form in columns 3-17. This allows the program to associate the data with the form on which it was entered.

Refer to Table 13-1 for the complete list of the event codes, the functions they perform, and the actions to which there may be a response.

Table 13-1. Event Codes

Code	Function	Response to Action Code
00	Terminal operator pressed the ENTER key. Data is included in the record.	54 (RDTERM)
01	Terminal operator pressed f1 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
02	Terminal operator pressed f2 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
03	Terminal operator pressed f3 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
04	Terminal operator pressed f4 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
05	Terminal operator pressed f5 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
06	Terminal operator pressed f6 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
07	Terminal operator pressed f7 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
08	Terminal operator pressed f8 key; no data. (unless "F" on RDTERM)*	54 (RDTERM)
09	Read number of fields that failed V/3000 or user edits.	59 (EDITS) or 61 (NUMERR)
10	Read data from V/3000 data buffer. Data is included in the record.	64 (GETDTA)
11	Return record number of current batch record, mode of operation (0 = collect, 1 = browse), repeat/append and freeze/append status, and next form name.	Any action (or no action) except 54, 59, 61, 64 or 74.
12	Return length and contents of a particular field in the V/3000 data buffer.	74 (GETFLD)

* Refer to RDTERM in Table 13-2 for description of this option.

ACTIONS

For simple applications that bypass V/3000 editing, RPG programmers need use only actions numbered 50 through 57. These actions are designed to allow direct interaction between a terminal and a user program. Any editing must be specified in the user program rather than by FORMSPEC.

For more complex applications the RPG programmer can use the full set of actions that provide access to all the V/3000 capabilities.

Refer to Table 13-2 for a list of the action codes with their functions.

Table 13-2. Action Codes

Code	Mnemonic	Function	Corresponding V/3000 Procedure
50	CHGNXT	Specify next form name and whether form is repeat/append, freeze/append, or normal.	—
51	GETNXT	Get the next form from the forms file, and set repeat/append, and freeze/append status. If repeat mode set, the form is not actually retrieved. Follow this action by action 53 to display form at terminal.	VGETNEXTFORM
52	PUTMSG	Specify a message, with any enhancements to be displayed on the error/status line on the terminal screen (the "window"). The message is not displayed until action 53 is executed.	VPUTWINDOW
53	SHOW	Display current form, any initial data, and any message on the terminal screen.	VSHOWFORM
54	RDTERM	Read input from terminal to the V/3000 data buffer. This action must be followed by a read to transfer the data to the user program. The record read is an event type 00-08. Normally, screen data is not included on records returned for event types 01-08 (operator presses function key F1-F8). Entry of "F" in column 7 of the Action 54 (RDTERM) Output record causes screen data to be included on an event 01-08 record whenever a function key is pressed. (Data is always included when the ENTER key is pressed.)	VREADFIELDS VGETBUFFER
55	SHOMSG	Display message specified in user program with any enhancements, in error/status line of terminal screen (the "window"). (Action combines actions 52 and 53.) If data in the V/3000 buffer has changed, the new data is displayed.	VPUTWINDOW VSHOWFORM
56	CORERR	Indicate fields that failed user edits. The output record contains the number of the bad field, and a message with its length and any enhancements. The current form and the message are displayed on the screen and the user's response is read. Fields with errors are enhanced if enhancements were specified. Follow this action with a read of the user response (event type 00-08). This action combines actions 62, 53, and 54.)	VSETERROR VSHOWFORM VREADFIELDS VGETBUFFER
57	SHODTA	Display data from the user program buffer to fields on ** screen. (This action combines actions 63 and 53.)	VPUTBUFFER VSHOWFORM
The actions to this point can all be used to transfer data between a user program and the terminal according to forms specified by FORMSPEC. The remaining actions are intended for users who want to make use of the full V/3000 capability.			
58	INIT	Initialize fields in the current form according to initialization specified for the field through FORMSPEC. If errors, move message to window.	VINITFORM VERRMSG VPUTWINDOW
59	EDITS	Perform edits specified for the fields in the current form; display message describing the first error in the terminal window. This action should be followed by a read (event type 09); the record read contains the number of fields that failed the edit.	VFIELDEDITS VERRMSG VPUTWINDOW
60	PRINT	Print current form with its data on line printer. Form must have been read from the forms file by a prior call to action 51.	VPRINTFORM
61	NUMERR	Ask for error status. Follow output of this action by a read (event type 09); the record read indicates the number of fields that failed edits.	—

Table 13-2. Action Codes (Continued)

Code	Mnemonic	Function	Corresponding V/3000 Procedure
62	BADFLD	Indicate that field failed a user edit. The output record contains the number of the bad field and a message for the window.	VSETERROR
63	PUTDTA	Replace the data in the V/3000 data buffer with values specified in the user program buffer.** The output record contains data for the V/3000 data buffer and the length of the data.	VPUTBUFFER
64	GETDTA	Write the data in the V/3000 data buffer to the user program buffer. This action should be followed by a read (event type 10); the record read contains the data from the V/3000 buffer and its length.	VGETBUFFER
65	FINISH	Perform all the final processing that is specified as part of the "finish" phase for the form. If errors, move message to window.	VFINISHFORM VERRMSG VPUTWINDOW
66	WRTBAT	Write the contents of the V/3000 data buffer to the batch record corresponding to the current record number. If in collect mode, the current record number is then advanced by one. If in browse mode, the current record number is unaffected.	VWRITEBATCH
67	PREV	Read data from the previous batch record to the V/3000 data buffer.*** If not in browse mode, place program in browse mode. Save the current location in the batch file and the current form name.	VREADBATCH
68	REREAD	Reread data from current batch record into V/3000 data buffer. Program must be in browse mode (requested by prior call to action 67).	VREADBATCH
69	NEXT	Read data from next batch record into V/3000 data buffer.*** Program must be in browse mode (requested by prior call to action 67).	VREADBATCH
70	RESUME	Return to collect mode from browse mode. Program must be in browse mode (prior call to action 67). Next form name and location in batch file is restored.	-
71	DELETE	Delete current batch record. Program must be in browse mode (prior call to action 67).	-
72	RDBTNU	Read batch record identified by its record number.	-
73	CLRMSG	Clear the message window. If CLRMSG is followed by the letter "I", clear the message from both the terminal screen and from the window buffer in memory; otherwise, clear only the window buffer.	VPUTWINDOW VSHOWFORM
74	GETFLD	Locate contents of specified field in V/3000 data buffer. Follow this action by a read (event 12) to transfer the data and its length to the user program.	VGETFIELD
75	PUTFLD	Transfer data from user program buffer to specified field in V/3000 data buffer.	VPUTFIELD
<p>* Browse mode is the mode of operation in which existing data in the batch file may be examined and modified. New batch records cannot be added during browse mode.</p> <p>** The user program buffer may contain old data from a previous action or event. To clear the buffer before updating, specify "ADD" in columns 16 to 18 of the first output specification for this action (you must also place an "A" in column 66 of the WORKSTN File Description Specification).</p> <p>*** Any attempt to read past either end of the batch file will cause the current record number to be set to the out-of-bounds value (-1 for PREV, and End-of-Batch +1 for NEXT), although no read will actually be attempted. No other indication is given to the user program that the batch file is out-of-bounds.</p>			

V/3000 Interface

Refer to figure 13-1 for an overview of the relation between the terminal, the forms file, the batch file, the user program and the areas of memory used by V/3000. The figure illustrates how the RPG action codes are used to transfer information between these areas.

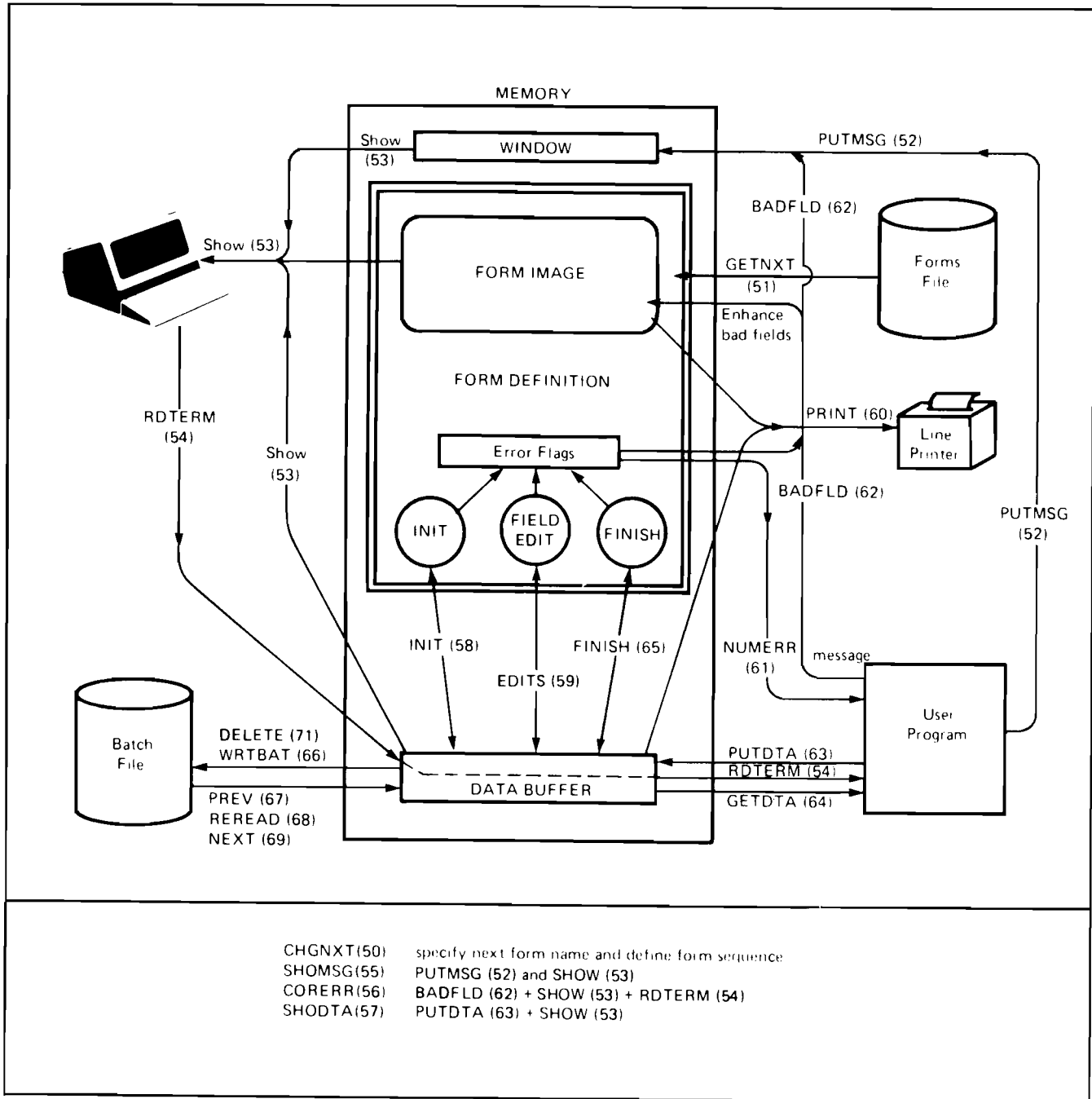


Figure 13-1. Relation of RPG Action Codes of V/3000

RECORD TYPES

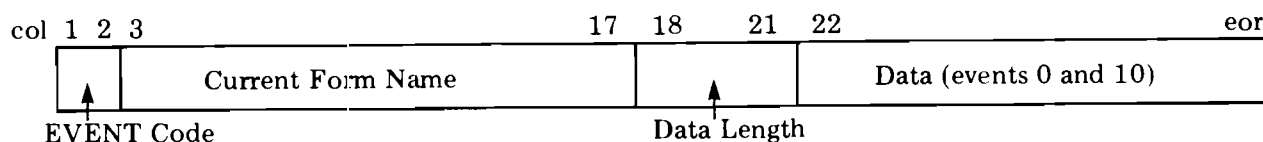
All records used for the RPG interface to V/3000 must be variable length ("V" in column 19 of the File Description Specification for WORKSTN.) Input records contain the EVENT codes plus forms information. Output records contain the ACTION codes or mnemonics, plus other information that depends on the particular action.

INPUT RECORD FORMATS

Depending on the EVENT code, input records have one of four possible formats. All input records contain the EVENT code in the first two locations, and the current form name in the 15 characters immediately following.

EVENTS 00-08, AND 10

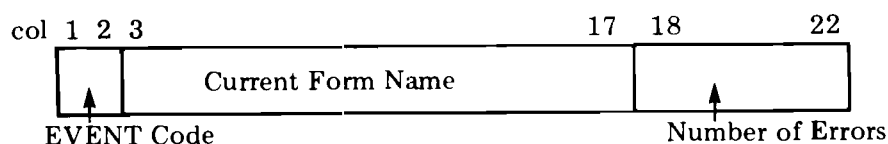
uses the input record format:



The data starting in location 22 is separated into fields according to the definition of the current form. The Data length is the total number of characters required by all the fields in the current form. Only events 0 and 10 actually have data.

EVENT 09

uses the input record format:

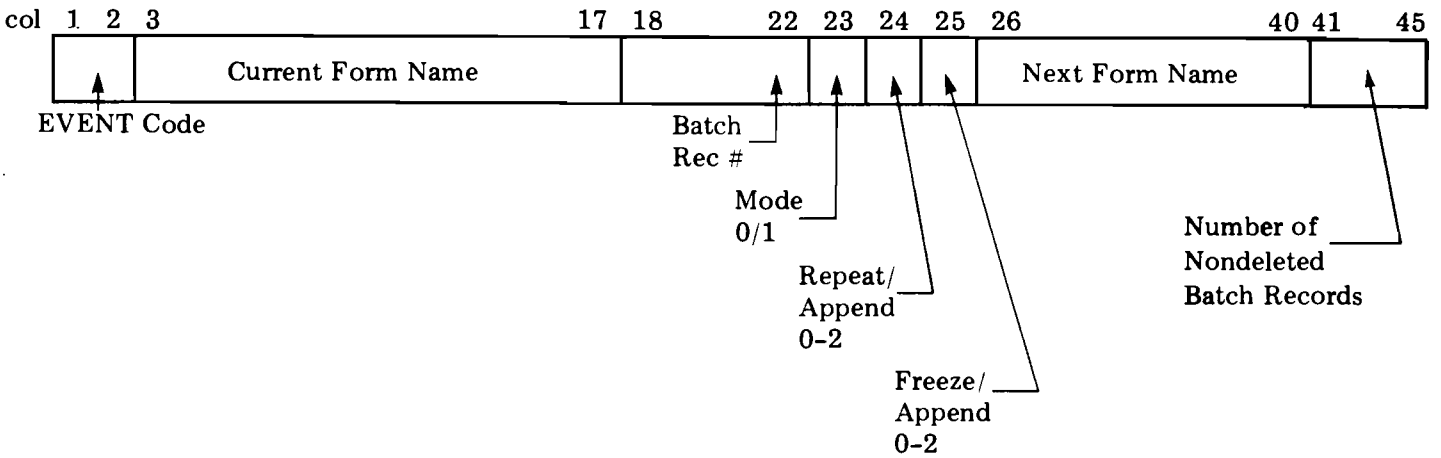


The number of errors specifies the number of fields in which errors were detected, either by the FORM-SPEC edit checks or by user program editing.

V/3000 Interface

EVENT 11

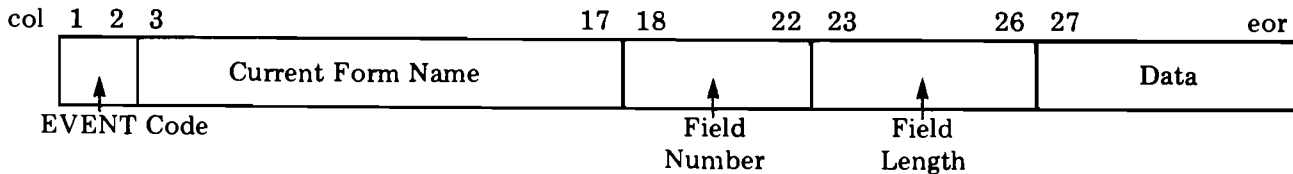
uses the input record format:



Event 11 is a general status event which is available after any action that does not return another event. Event 11 is also available at the beginning of execution when it returns the number of the next batch record (i.e. End-of-batch +1). This is particularly useful in determining the batch file upper bounds when an existing batch file is used.

EVENT 12

uses the input record format:



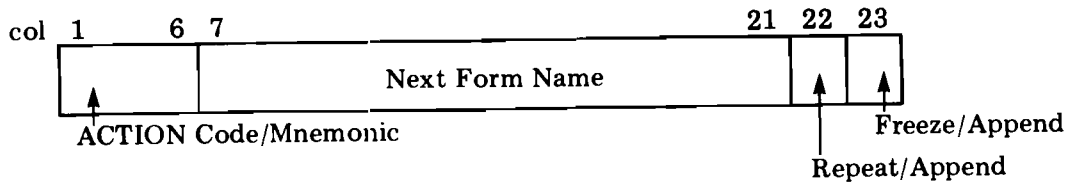
The field number is a unique identifier for the field being retrieved; it does not change if the position of the field in the form changes. The data in the specified field is returned in this record.

OUTPUT RECORD FORMATS

Output records are in one of eight formats depending on the ACTION code or mnemonic. All output records contain an ACTION code in location 1 and 2, or an ACTION mnemonic in locations 1 through 6.

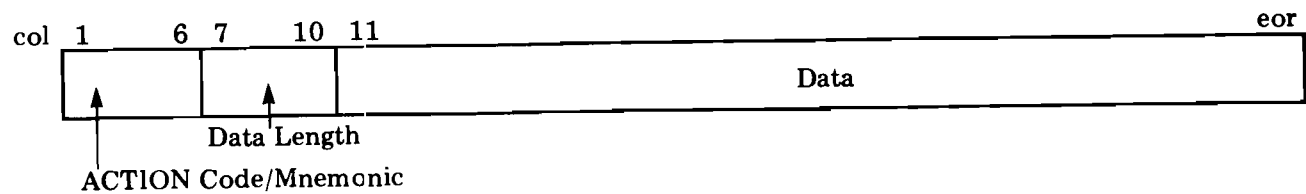
ACTION 50 (CHGNXT)

uses the output record format:



ACTIONS 57 (SHODTA) AND 63 (PUTDTA)

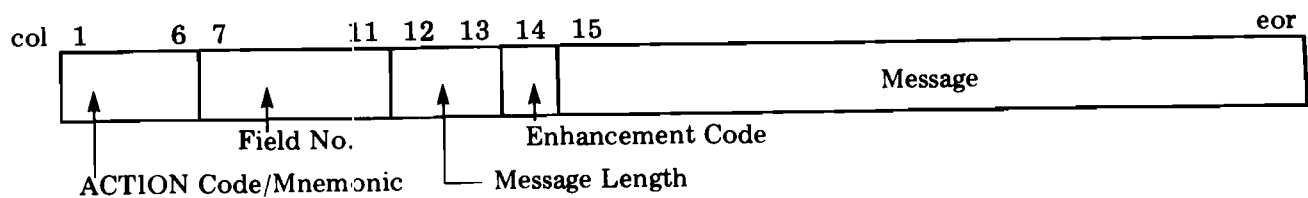
use the output record format:



The data starting in location 11 is separated into fields according to the field definition of the current form. The data length is the total number of characters for the data in this form.

ACTIONS 56 (CORERR) AND 62 (BADFLD)

use the output record format:



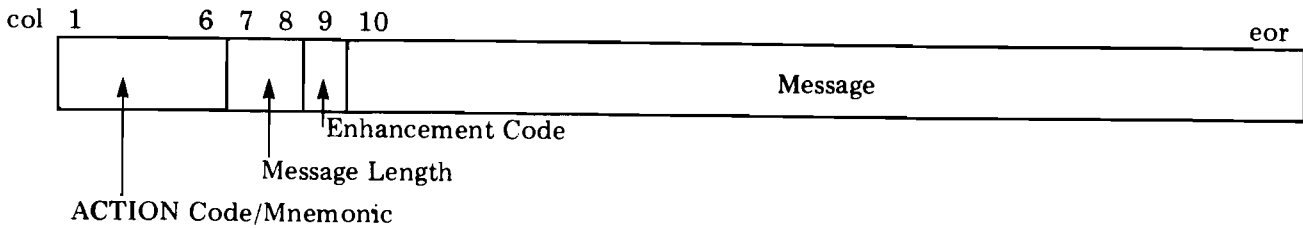
The field number identifies the field in which an error was detected so that this field can be enhanced at the terminal. The message length specifies the number of characters in the message to be sent to the window for subsequent display. For best results, the message should have no more than 79 displayable characters. The enhancement code may be one of the following:

- @,A-0 Specify particular enhancement (refer to WINDOWENH description, section VI in the VPLUS/3000 Reference Manual)
- zero No enhancement
- blank Do not change existing enhancement

V/3000 Interface

ACTIONS 52 (PUTMSG) AND 55 (SHOMSG)

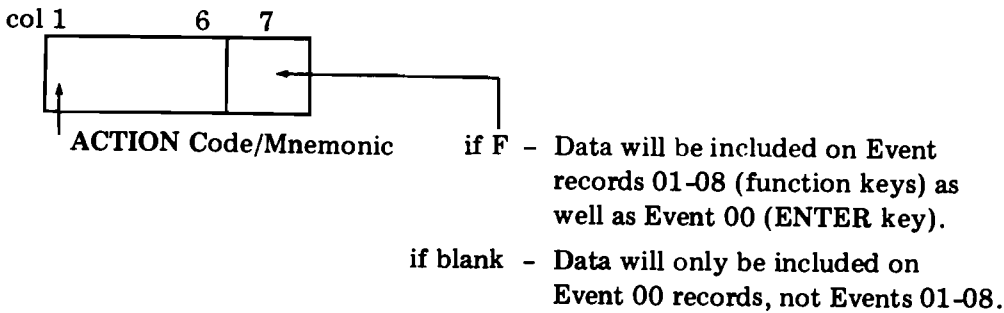
use the output record format:



The message length specifies the number of characters in the message, and the enhancement code determines the message enhancement (see list of codes above for actions 56 and 62).

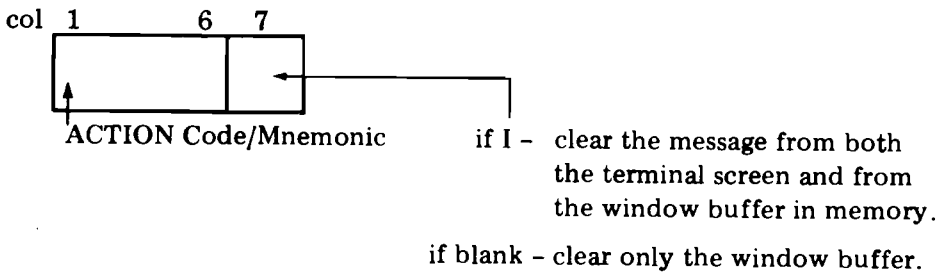
ACTION 54 (RDTERM)

use the output record format:



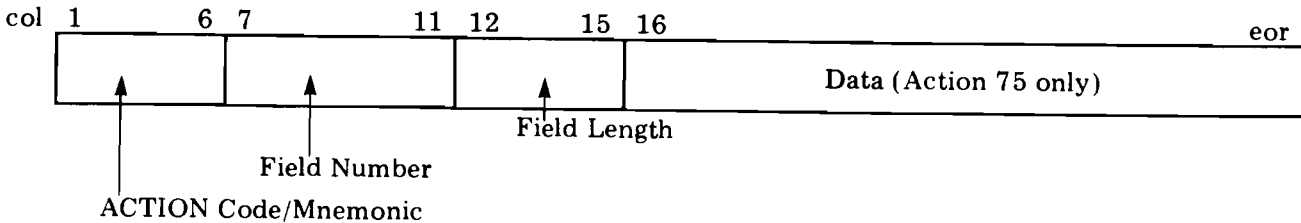
ACTION 73 (CLRMSG)

use the output record format:



ACTIONS 74 (GETFLD) AND 75 (PUTFLD)

use the output record format:



This record contains the field number of the selected field in the V/3000 data buffer, the number of characters in the field, and for PUTFLD, the data to be written to the V/3000 buffer. Note that each field in a form is assigned a number by FORMSPEC that does not change regardless of any changes to the position of the field in the form or to its description.

ACTIONS 51 (GETNXT), 53 (SHOW), 58-61 (INIT,EDITS,PRINT,NUMERR) AND 64-72 (GETDTA,FINISH,WRTBAT,PREV,REREAD,NEXT,RESUME,DELETE,RDBTNU)

use the output record format:

col 1 6
 ┌───────────┐
 ↑
 ACTION Code/Mnemonic

Note that this output record contains only the ACTION code. It is used by all actions that do not use the seven preceding output formats.

HANDLING RUNTIME ERRORS

RPG allows the programmer to specify one of six error response options. The specified option determines what RPG does in case an error occurs during program execution. The selected option is entered on a Control Record specification (H spec). When an error occurs for which no option is specified, the system asks the terminal operator to specify an option.

In a data entry operation with a form displayed on the terminal screen and the terminal operating in block mode, the request cannot take the usual form. Instead, the RPG interface displays a message describing the error in the screen message window, and waits for the terminal operator to enter an error response by pressing a function key.

Function keys f1 through f6 are used for this purpose. The terminal operator should press one of the following keys to select an error response:

- f1 Continue execution
- f2 Skip the input record containing the error and continue execution.
- f3 Terminate the program by executing the normal termination code.
- f4 Terminate the program immediately.
- f5 Terminate normally, and print an error dump.
- f6 Terminate immediately, and print an error dump.

ERROR DUMP

The RPG error dump is normally written to the standard list device, (\$STDLIST), which in a session, is the terminal. Since an error dump to a terminal is of limited use, particularly in a data entry environment, the dump can be redirected to a file. The name of this file is specified in columns 7-14 of the Control Record Specification (H spec). If omitted, \$STDLIST is used and the dump is sent to the terminal.

ERROR MESSAGE DISPLAY

Some error and warning messages are displayed on the terminal screen. Each message is displayed for three seconds unless this interval is changed by entering a digit between 0 and 9 in column 51 of the File Description specification. For example, to increase the time during which the message appears on the screen to 5 seconds, enter "5" in column 51 of the File Description spec. If the interval is set to zero, messages are not displayed at the terminal. (This display interval applies only to WORKSTN-generated errors, not to RGP-generated errors, V/3000 edit-checking errors, or to messages placed directly in the window by your program. See "Error Message Interval" on page 13-14 for details.)

BREAK KEY

The **BREAK** key is placed close to the **ENTER** key on some terminals. As a result, it is easy for the terminal operator to press **BREAK** rather than **ENTER**. To avoid this possibility, the **BREAK** key is disabled when a **WORKSTN** file is in use. You can override this default setting and enable the **BREAK** key with a notation on the File Description Specification for the **WORKSTN** file. (See “Break Key Enable” below for details.)

If **BREAK** is enabled, the terminal operator should be told how to recover from an accidental break by typing **RESUME**. If the operator was entering data, he or she should then press function key 4 (**REFRESH** under **ENTRY** program control) to clear previously typed data from the form. The operator can then retype the data in the form and press **ENTER** to enter the data.

TRACE FILE

The trace file is a tool to help find program errors. This file is specified as a continuation line on the File Description Specification. If a trace file is specified, every action and event causes at least one record to be written to the file. Then, if the program does not operate as expected, you can examine the trace file to isolate the error. If a runtime error occurs, the message describing the error is written to the trace file as well as displayed at the terminal. If the trace file becomes full, tracing stops, but program execution continues.

SPECIAL WORKSTN FIELDS

Columns 48-52 of the **WORKSTN** File Description Specification are reserved for entry of special options that are unique to the **WORKSTN** file.

FUNCTION KEY LABELS ENABLE (COLUMN 50)

In a formfile, you can define labels that are to be displayed for the eight function keys on certain types of terminals. In order for the labels to be displayed at runtime, they must be “enabled” prior to opening the formfile. In **RPG**, you do this by entering an “L” in the “Function Key Labels Enable” field.

ERROR MESSAGE INTERVAL (COLUMN 51)

For **WORKSTN**-generated errors, you can specify the number of seconds during which the error message will be displayed in the terminal’s message window. Do this by entering the number of seconds desired (0 through 9) in the “Error Message Interval” field. The default interval is 3 seconds. An entry of zero suppresses display of error messages.

This display interval does not apply to **RPG**-generated errors, **V/3000** edit-checking errors, or to messages placed directly in the window by your program with **PUTMSG** or **SHOMSG** actions.

BREAK KEY ENABLE (COLUMN 52)

Entry of a “B” in this field causes the keyboard **BREAK** key to be “enabled”. See “Break Key” for further discussion about **BREAK** key usage.

EXAMPLE USING THE V/3000 INTERFACE

The following simple example uses the V/3000 interface to control data entry through forms design by the FORMSPEC program of V/3000. (For a complete example using all the capabilities of V/3000 to control data entry, refer to the RPG program in appendix A.)

Assume the program is to perform the following actions:

- Display an initialized form.
- Read data entered at the terminal.
- Perform V/3000 and user edits.
- Write the edited data to the batch file.

The first step is to use the V/3000 FORMSPEC program to define the forms and the editing associated with them. The next step is to write the RPG program. A general plan of the program is shown below. Note that all output is exception output, and all reads are demand reads.

Note that in this example, the function keys f1 through f7 are treated as if they were f8 (EXIT). Thus, if any function key is pressed, the program terminates.

PROGRAM PLAN

- *1. Get the next form from the forms file using ACTION 51 (GETNXT). (If the form is a repeating form, the next form is not actually fetched.)
2. Move any initial values to this form using ACTION 58 (INIT).
- *3. Display the form with the initial values and message window contents using ACTION 53 (SHOW).
4. Indicate that data entered on form is to be read with ACTION 54 (RDTERM).
5. Read entered data from the WORKSTN file (demand read). The data is input as an EVENT of one of the types 00 through 08:

If EVENT is type 01 through 08 (EXIT), terminate.
If EVENT is type 00 (ENTER), continue.

(For simplicity in this example, events 01 through 07 are treated as if they were event 08.)

6. Perform all V/3000 edits specified for the fields on the form using ACTION 59 (EDITS).
7. Read the WORKSTN file. The record returned is an EVENT type 09 and specifies the number of errors found in the entered data.

*If errors are found, a message describing the first error found is moved to the window buffer, and all the fields with errors are enhanced. Return to step 3 to display the form with the error message and enhancements. Continue through steps 3-7 until no errors are found.

If no errors are found, continue.

8. Transfer the data from the V/3000 data buffer to the user program with ACTION 64 (GETDTA).
 9. Read the WORKSTN file. The record returned is an EVENT type 10, containing the data from the V/3000 buffer.
 - *10. Perform any editing specified in the user program. If errors are found, do step 11; otherwise, go on to step 12.
 11. Use ACTION 56 (CORERR) to indicate which fields failed the user edit, display the form with fields in error enhanced, and if the program buffer contains a message for the window, display this message with the form. Follow ACTION 56 by a demand read. The record read should be an EVENT type 00 indicating ENTER was pressed and new data has been entered.
- *Repeat steps 10 and 11 until all data has passed the user edits.
12. Transfer the edited data to the V/3000 buffer using ACTION 63 (PUTDTA).
 13. Write the data in the V/3000 buffer to the batch file with ACTION 66 (WRTBAT).
 14. If the next form name or the repeat/append or repeat/freeze status is to be changed, use ACTION 50 (CHGNXT) to make these changes.

*Return to step 1.

During execution of this program, all data entered by an operator at the terminal into forms designed through FORMSPEC will be written to the batch file, one record per form. You can use the reformatter (described in section V) to insure that the data is in the exact form needed by your application.

Before running the program, all files must be described. These files include the file allocated to the device WORKSTN, the forms file on which the forms are stored, the batch file to which the data is written, a trace file to trace the actions and events, and a dump file to which a program dump is sent in case of runtime errors.

These files are entered on the Control Record and File Description Specification sheets. (Refer to figure 13-2).

HEWLETT PACKARD
RPG CONTROL RECORD AND FILE DESCRIPTION SPECIFICATIONS
Page 1 of 7

Programmer: _____ Date: _____

Program Title: **DATA ENTRY EXAMPLE**

Punching Instructions
 Graphic:
 Punch:

Program Name: **RPG0BJ**

Control Record Specification

Sequence Number	Error Type	Priority	Operator	Error Record No.	Error Message	Error Code	Error Location	Error Date	Error Time	Error User	Error Device	Error Status	Error Action	Error Comment
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

→ **DUMPFILE**

File Description Specifications

File Name	File Type	File Format	File Length	File Positioning	File Organization	File Access Method	File Record Length	File Record Format	File Record Key	File Record Indicator	File Record Substitution	File Record Suppression	File Record Control	File Record Check	File Record Error	File Record Action	File Record Comment
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
TRANSFILE	U	V	30														

File Names

File Name	File Type	File Format	File Length	File Positioning	File Organization	File Access Method	File Record Length	File Record Format	File Record Key	File Record Indicator	File Record Substitution	File Record Suppression	File Record Control	File Record Check	File Record Error	File Record Action	File Record Comment
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
WORKSTN																	

Annotations:

- If omitted, dump is sent to terminal
- Update Demand File
- Records must be variable length
- # of data characters in longest form plus 20
- 5-second message interval
- File names

Figure 13-2. File Description Specification for Sample Program

CONTROL RECORD SPECIFICATION

Columns 7 - 14 "DUMPFIL" - Name of file to which dump is sent in case of a runtime error.
(If omitted, dump is directed to the terminal.)

FILE DESCRIPTION SPECIFICATION

Columns 7 - 14 "TRANSFIL" - Name of RPG file assigned to device class WORKSTN.

Column 15 "U" - Indicates file is type update. No other type is allowed for interface with
V/3000.

Column 16 "D" - Indicates file is demand. For this application, file must be demand file;
for other applications that use a simple RPG cycle, P for primary may be
specified.

Column 19 "V" - Indicates record length is variable. All records in WORKSTN file must
be variable length.

Columns 24 - 27 Record length is dependent on the data buffer length needed to hold all the
data from the longest form, plus 20 characters for control information.
(Assume the longest form for this application requires 60 characters of data,
so the record length is specified as 80.)

Columns 40 - 46 "WORKSTN" - Device class name must be WORKSTN for the V/3000
interface.

Column 50 "L" - Enables Function key labels for display on the terminal.

Column 51 "0" through "9" indicates number of seconds during which message is
displayed at terminal (default = 3). If zero, message is suppressed.

Column 52 "B" - Enables BREAK key; leaving this column blank or entering any other
value leaves the BREAK key disabled.

Column 53 "K" - Indicates file description continuation, one for each additional file.

Columns 54 - 59 "FORMS" - Specifies that a forms file is used. This file is required.

 "BATCH" - Specifies that a batch file is used. If this keyword is not
specified, no batch file is used.

 "TRACE" - Specifies that a trace file is used. If this keyword is not specified,
no trace file is used.

Columns 60 - 74 Name of file whose use is indicated in columns 54 - 59; specified as an MPE
file name. If more than 15 characters are needed for the name, a :FILE
command should be used.



INPUT SPECIFICATIONS

The input specifications in figure 13-3 are for events 00 (ENTER key), 01 through 08 (function keys f1 - f8), and 09 (number of fields with errors). Note that events 01 through 07 are treated by this program as if they were event 08. No data is read by events 01 through 08; in this program, they signal that a key has been pressed at the terminal, and are treated as EXIT.

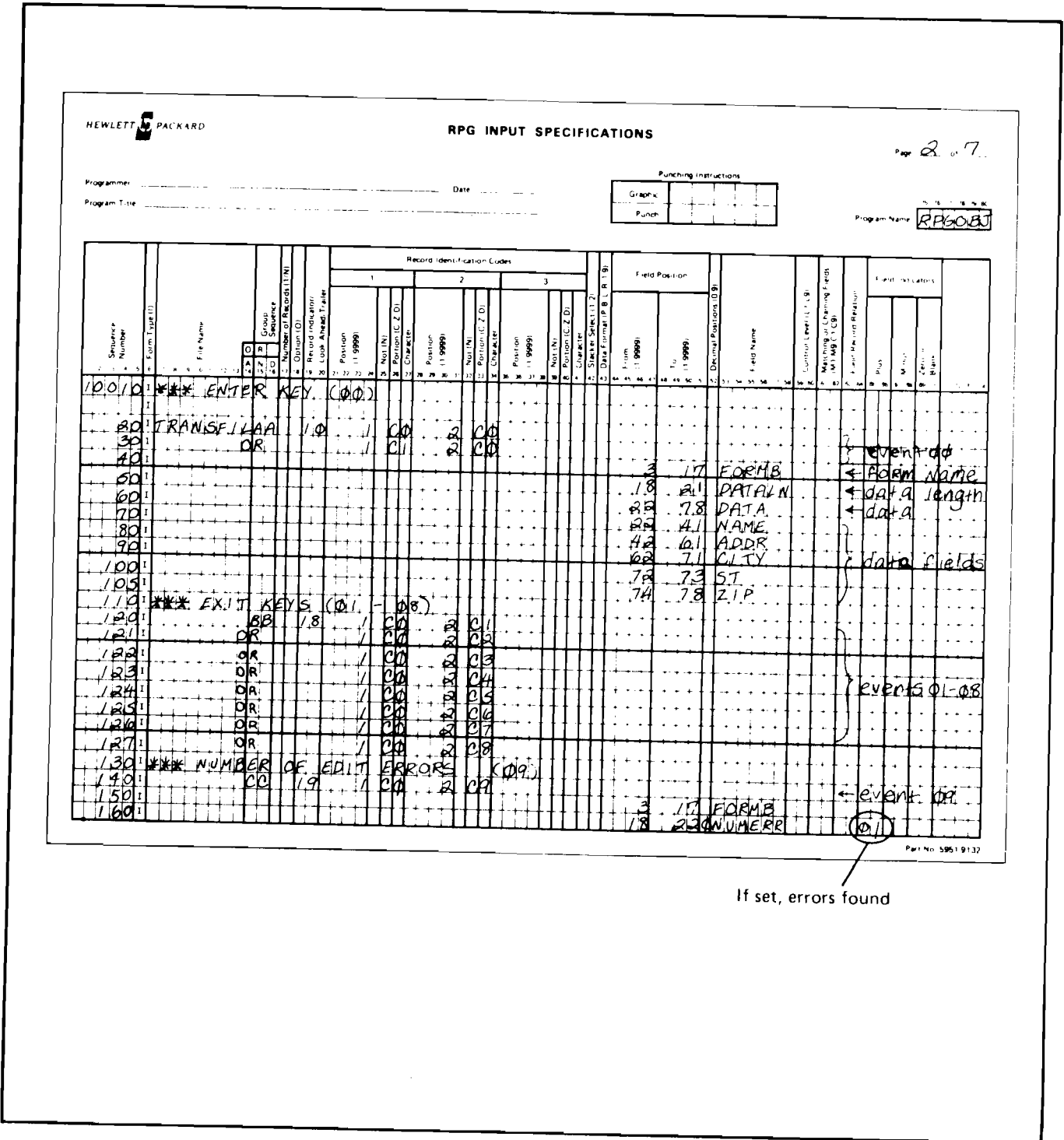


Figure 13-3. Input Specifications for Sample Program

CALCULATION SPECIFICATIONS

Since the file TRANSFIL assigned to device file WORKSTN is specified as an update demand file, the program control remains in the calculation specifications. The program terminates when the f8 key (event 08) is pressed. Refer to figure 13-4 for the calculation specifications used by the program. Note that each numbered step corresponds to the steps described above under the heading "Program Plan".

HEWLETT PACKARD		RPG CALCULATION SPECIFICATIONS																				Page 3 of 7	
Program: _____ Date: _____		Punching Instructions																				Program Name: RP6000	
Program Title: _____																						Punch: _____	
Sequence Number	Form Type (1)	Control Level (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)	Indicators			Factor 1	Operation	Factor 2	Result Field Name	Result Length (1-256)	Decimal Position (0-9)	Initial Adjust (1)	Resulting Indicator				Comments						
			And	And	And								Arithmetic	Plus Minus Zero	Compare	1-2		1-2	1-2				
			Not	Not	Not								High	Low	Equal								
30000	*	1	*			GET NEXT FORM START.	TAG																
10							MOVEL "GETNXT"	ACTION	60									GETNXT - 61					
20							SETDM																
30							EXCPT																
40																							
50	*	2	*			DET. INITIAL VALUES	MOVEL "INIT."	ACTION										INIT - 58					
60							SETDM																
70							EXCPT																
80																							
90	*	3	*			DISPLAY FORM REPEAT.	TAG																
100							SETOP																
101							MOVEL "SHOW"	ACTION										SHOW - 53					
110							SETDM																
120							EXCPT																
130							SETOP																
140	*	4	*			READ FROM TERMINAL	MOVEL "RDTERM"	ACTION										RDTERM - 64					
150							SETDM																
160							EXCPT																
170							SETOP																
180							SETOP																
190	*	5	*			READ RECORD	READ TRANSFIL																
200							ELSE CONTINUE																
210	***					IF FI - FB EXIT.	SETDM											EXIT IF FI - FB					
220							EXCPT																
230							SETDM																
240							EXIT																
30240	*	6	*			IF ENTER - EDIT	DATA																
250							MOVEL "EDITS"	ACTION										EDITS - 59					
260							SETDM																
270							EXCPT																
280	*	7	*			DETERMINE NO. OF ERRORS	READ TRANSFIL																
290							IF ERRORS - RETURN TO STEP 3																
300	***						GOTO REPEAT.																
310							IF NO ERRORS CONTINUE																
320	***																						
330	*	8	*			TRANSFER DATA TO PROGRAM	MOVEL "GETDTA"	ACTION										GETDTA - 64					
340							SETDM																
350							EXCPT																
360																							
370	*	9	*			READ DATA FROM TRANSFIL	READ TRANSFIL																
380																							
390	*	10	*			PERFORM USER EDITS	MORERR	TAG															
400																							
410	*																						
420	***					SUPPLY USER EDITS HERE																	
430	*																						

Figure 13-4. Calculation Specifications for Sample Program

HEWLETT PACKARD

RPG CALCULATION SPECIFICATIONS

Page 5 of 7

Program _____ Date _____
 Program Title _____

Punching Instructions	
Graphic	
Punch	

Program Name **RIF6000**

Sequence Number	Form Type (C)	Control Level (L, O, L, P, U, R, S, R, A, N, O, H)	Indicators			Factor 1	Operation	Factor 2	Reg'd Field Name	Reg'd Length	Decimal Positions (D, S)	Reg'd Adjust (A)	Resulting Indicators				Comments		
			Not	And	And								Not	Arithmetic	Plus Minus Zero	Compare		Lowest Order 2's	
30440	C		1	1															
450	C																		
451	C																		
452	C																		
453	C																		
454	C																		
460	C																		
470	C																		
480	C																		
490	C																		
500	C																		
510	C																		
520	C																		
530	C																		
540	C																		
550	C																		
560	C																		
570	C																		
580	C																		
590	C																		

Part No. 5188-9133

6 of 7

RIF6000

60600	C																			
610	C																			
620	C																			
630	C																			
640	C																			
650	C																			
660	C																			
670	C																			
680	C																			
690	C																			
700	C																			
710	C																			
720	C																			
730	C																			
740	C																			

Part No. 5188-9133

Figure 13-4. Calculation Specifications for Sample Program (Cont.)

OUTPUT SPECIFICATONS

Four types of output record are established for this program. They are illustrated in figure 13-5. Refer to "Output Record Formats" in the preceding text for a description of the formats assigned to each action code.

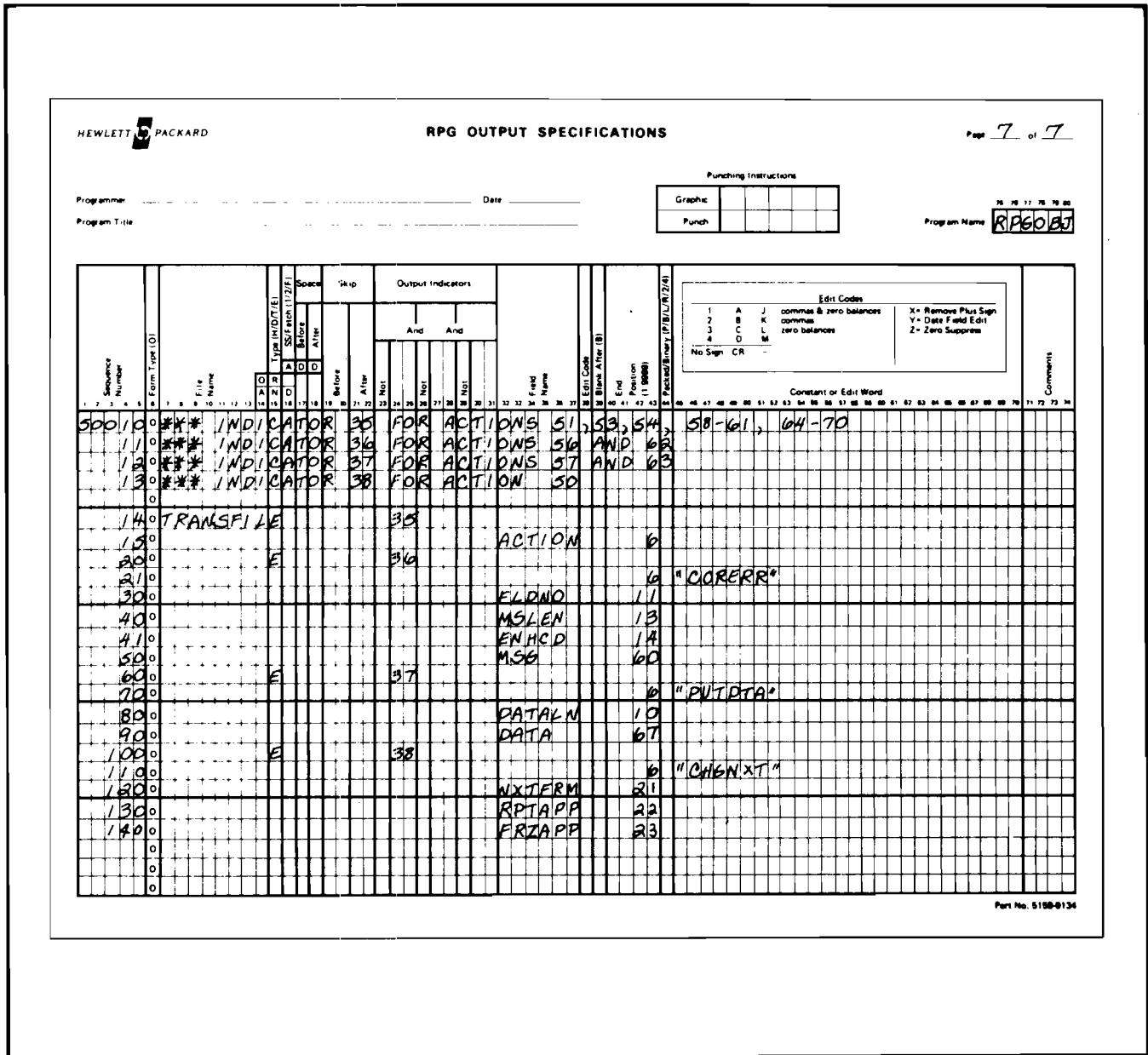


Figure 13-5. Output Specifications for Sample Program

COMPILED VERSION

The following pages contain a listing of the compiled version of the sample program illustrated in figures 13-2 through 13-5.

```

$CONTROL LIST,SOURCE,NOWARN,USLIMIT,QUOTE="
H
F TRANSFILUD V      80          WORKSTN
F
F
F
I*** PROCESS ENTER KEY (00)

I TRANSFILAA 10    1 C0    2 C0
I      UR      1 C1    2 C0
I
I      3    17 FORMB
I      18    21 DATALN
I      22    78 DATA
I      22    41 NAME
I      42    61 ADDR
I      62    71 CITY
I      72    73 STATE
I      74    78 ZIP
I*** EXIT KEYS (01 - 08)
I      BB 18    1 C0    2 C1
I      UR      1 C0    2 C2
I      UR      1 C0    2 C3
I      UR      1 C0    2 C4
I      UR      1 C0    2 C5
I      UR      1 C0    2 C6
I      UR      1 C0    2 C7
I      UR      1 C0    2 C8
I*** NUMBER OF EDIT ERRORS (09)
I      CC 19    1 C0    2 C9
I
I      3    17 FORMB
I      18    22 NUMERR      01
C* 1 * GET NEXT FORM

C          STAP1      TAG
C          MOVEL"GETNXT" ACTION 6
C          SETON      35      GEINXT - 51
C          EXCPT
C* 2 * SET INITIAL VALUES
C          MOVEL"INIT  " ACTION
C          SETON      35      INIT - 58
C          EXCPT
C* 3 * DISPLAY FORM
C          REPEAT      TAG
C          SETOF      01
C          MOVEL"SHOW  " ACTION
C          SETON      35      SHOW - 53
C          EXCPT
C* 4 * READ FROM TERMINAL
C          MOVFL"RDTERM" ACTION
C          SETON      35      RDTERM - 54
    
```



```

C          EXCPI
C          SETOF          35
C* 5 * READ RECORD
C          READ TRANSFIL          H0
C*** IF F1 - F6 THEN EXIT ELSE CONTINUE
C 18          SETON          LP
C LR          GO10 EXIT          EXIT IF F1
C* 6 * IF ENTER, EDIT DATA
C 10          MOVE1"EDITS " ACTION
C 10          SETON          35 EDITS - 59
C 10          EXCPI
C* 7 * DETERMINE NUMBER OF EDIT ERRORS
C 10          READ TRANSFIL          H0
C*** IF ERRORS, RETURN TO STEP 3
C 01          GO10 REPEAT
C*** IF NO ERRORS - CONTINUE
C* 8 * TRANSFER DATA TO PROGRAM
C          MOVE1"GE1DIA" ACTION
C          SETON          35 GE1DIA - 64
C          EXCPI
C* 9 * READ DATA FROM TRANSFIL
C          READ TRANSFIL          H0
C* 10 * PERFORM USER EDITS
C          MOVERR IAG
C*
C*** SUPPLY EDIT ROUTINES HERE ***
C*
C* 11 * FIND ERRORS, ENHANCE FIELDS, DISPLAY MESSAGE
C M20          GO10 M0ERRS
C          MOVE " " FLDNO 5
C          MOVE "00" MSLEN 2
C          MOVE " " ENHCD 1
C          MOVE " " MSG 47
C          SETON          36 COREPR - 56
C          EXCPI
C          READ TRANSFIL
C          GO10 M0ERRR
C          MOERRS IAG
C          SETOF          36
C* 12 * MOVE DATA FROM PROGRAM TO BUFFER
C          SETON          37 PUIDTA - 63
C          EXCPI
C          SETOF          37
C* 13 * WRITE DATA FROM BUFFER TO BATCH FILE
C          MOVE1"WRTBAT" ACTION
C          SETON          35 WRTBAT - 66
C          EXCPI
C          SETOF          35
C* 14 * CHECK NEXT FORM FOR STATUS CHANGE
C          MOVE " " NXTFRM 15
C          MOVE1" " NXTFRM
C          MOVE1FURMB NXTFRM
C          MOVE "0" RPTAPP 1
C          MOVE "0" FRZAPP 1
C          SETON          38 CHGNXT - 50

```

```

C          EXCPT
C          SETOF          38
C* 15 *   RETURN TO STEP 1
C          SETOF          011018
C          SETOF          19
C          GO TO START
C* 16 *   END OF PROCESSING
C          EXIT          TAG
U*** IND 35 FOR ACTIONS 51,53,54,58-61,64-70
    
```

```

UTRANSETLE      35
U
U      F      36
U
U          ACTION      6
U          FLDNO      11
U          MSLEN      13
U          ENHCD      14
U          MSG        60
U
U      E      37
U
U          DATA LN    6
U          DATA      10
U          DATA      67
U
U      E      38
U
U          CHGNXT      6
U          WXIFRM      21
U          RPTAPP      22
U          FRZAPP      23
    
```

RPG Copylib Preprocessor (RPGCOPY)

SECTION

XIV

RPGCOPY.PUB.SYS is an optional preprocessor utility used to include RPG source records from other files into the source code being compiled. You must specifically request that the RPG compiler invoke the preprocessor by specifying a \$COPY subsystem command as the first line of your source program. For example:

```
0001 $COPY
0002 $CONTROL USLINIT,MAP,NAME=MYPROG
0003 H          1 L X
0004 FINPUT   IP F . . .
```

For both the \$COPY and \$INCLUDE commands, the "\$" must be specified in column 6.

14-1. INSERTING COPYLIB RECORDS

To add source lines from an external Copylib file into your program, specify \$INCLUDE subsystem commands at the necessary locations. The format of the \$INCLUDE command is:

```
$INCLUDE filename
```

where filename is an MPE or KSAM file.

When encountering this line, the preprocessor opens the file and inserts its RPG source records into the appropriate section or sections of the current program being preprocessed. Note that an external Copylib file can contain records for more than one section of the RPG program (such as File specs and Inputs specs). The rules for placement of Copylib records into the current program are as follows:

- If the Copylib record is an F-spec, E-spec, L-spec, I-spec, Data Structure I-spec, C-spec, O-spec, or Table/Array data, the record is placed at the end of its respective section of the program (after any records already in place).
- Any other Copylib record (H-spec, A-spec, \$-spec, comment using "*" in column 6, blank line, or invalid spec type) is placed in the section where the previous source line from original source or a Copylib file was placed. Typically this section is where the \$INCLUDE is specified. However, for example, if a Copylib file contains a File Description spec followed by a comment ("*" in column 6), and the \$INCLUDE line that names that Copylib file is specified in Output specs, the comment line is placed after the File Description spec, not after Output specs. Note that the \$COPY and \$INCLUDE lines are placed according to the same rules although they are changed to comment lines (by putting "*" in column 6) before being sent to the compiler.

All \$COPY and \$INCLUDE lines are printed to \$STDLIST (the standard list device) as they are encountered so you can track the progress of the preprocessor.

On the compile listing, all lines inserted from an external Copylib file are shown with a "C" in Column 5. If any of those lines are later modified (as described in Paragraph 14-2), the "C" in Column 5 is changed to "M".

NOTE: Do not place \$INCLUDE records within any compile-time table or array data specified at the end of your source program (after the first "*" separator record). Any \$INCLUDE records will be ignored as compiler commands and the characters "\$INCLUDE..." will be treated as table or array data.

14-2. MODIFYING COPYLIB RECORDS

In general, it is not appropriate to include program-specific items such as Level-break or Matching record indicators on external Copylib records. However, RPGCOPY allows you to omit such items from the Copylib records and specify them as modifications (or customizations) in individual programs. This is done by first specifying the \$INCLUDE line, and following the line with any modification lines to be applied to the records retrieved from the Copylib file named by the \$INCLUDE.

Modifications are limited to all fields in columns 15 through 74 of File Description specs and columns 43 through 52 and 59 through 70 of Input specs. The term "field" refers to a set of columns that define a unique entry on the specification, such as columns 59 through 60 (Control Level) on Input specs or columns 40 through 46 (Device Class Name) on File Description specs.

The modification record for a File Description specification must contain the same file name in columns 7 through 14 as was specified on the record to be modified. Likewise, the modification record for an Input specification must contain the Field name found on the Input record to be modified. To change a field, enter the new contents in the field's columns on the modification record. To blank out a field, enter an ampersand ("&") in the first column of the field. If a field is not to be changed, leave the field blank on the modification record.

For example, if an RPG source program contains the following

```

0001 $COPY ←----- Invokes RPGCOPY
0002 $CONTROL USLINIT,NAME=ENH017,RSPACE=2
0003 $INCLUDE RPGHEADR.RPG ←----- Copylib file
0004 FINPUT  IP  F      72          DISC
0005 IINPUT   NS
0006 $INCLUDE ENH017C1 ←----- Copylib file
0007 I                               NOWA  L1 ←----- Modification record
0008 I      NS  **
0009 I                               1   5  NXTA
0010 I                               6  10NXTN
0011 INXTN      DS
0012 I                               1   20NXTNN2
0013 I                               3   5  NXTNA3
0014 INOWNXT    DS
0015 I                               1   5  NOWA
0016 I                               6  10  NXTA
0017 OOUTPUT  H  201  1P
0018 O                               "TEST LOOKAHEAD DS"
0019 O                               + 1 "FIELDS, ETC."
0020 $INCLUDE ENH017C2 ←----- Copylib file
0021 FOUTPUT      &  $STDLST ←----- Modification record
0022 O                               L1  + 2 "*"
    
```

and the Copylib file RPGHEADR.RPG contains

```

HRPGDUMP  JF      1  L  X      B  E  B  N      N  P  1  1
    
```

and the Copylib file ENH017C1 contains

```

I                               1   5  NOWA
I                               6  10NOWN
    
```

and the Copylib file ENH017C2 contains

```

      FOUTPUT  O  F      132      OV      PP
      O*
201  $CONTROL RSPACE=1
202  OOUTPUT  D          N1P
203  O
204  O
205  O
206  O
207  O
208  O
209  O
      NOWA
      NOWN
      NXTA
      NXTN
      NXTNN2
      NXTNA3
      NOWNXT
  
```

then the resultant RPG compilation will show:

RPG/3000 COPYLIB PREPROCESSOR (RPGCOPY) - MON, MAY 13, 1985, 1:13 PM

```

0001 $COPY
0003 $INCLUDE RPGHEADR.RPG
0006 $INCLUDE ENH017C1
0020 $INCLUDE ENH017C2
  
```

} Progress report from preprocessor

PAGE 0001 HP32104A.06.05 RPG/3000 (C) HEWLETT-PACKARD CO. 1985.

```

0001 *COPY
0002 $CONTROL USLINIT,NAME=ENH017,RSPACE=2
0003 *INCLUDE RPGHEADR.RPG
      CHRPGDUMP JF      1  L  X      B  E  B  N  N  P  1  1 ← From RPGHEADR.RPG
  
```

```

0004 FINPUT  IP  F      72      DISC
      MFOUTPUT O  F      132      $STDLST ← From ENH017C2,
                                                then modified
  
```

```

0005 IINPUT  NS
0006 *INCLUDE ENH017C1
      MI
      CI
0008 I      NS  **
0009 I
0010 I
0011 INXTN   DS
0012 I
0013 I
0014 INOWNXT DS
0015 I
0016 I
      1  5  NOWA  L1 ← From ENH017C1, then
      6 10 NOWN  modified
      1  5  NXTA
      6 10 NXTN ← From ENH017C1
      1  2  NXTNN2
      3  5  NXTNA3
      1  5  NOWA
      6 10  NXTA
  
```

RPG Copylib Preprocessor (RPGCOPY)

```

0017 OOUTPUT H 201 1P
0018 O          17 "TEST LOOKAHEAD DS"      R
0019 O          30 "FIELDS, ETC."          +001
0020 *INCLUDE ENH017C2
      CO*
201 C$CONTROL RSPACE=1
202 COOUTPUT D          N1P
203 CO          NOWA      5          R
204 CO          NOWN      11         R
205 CO          NXTA      17         R
206 CO          NXTN      23         R
207 CO          NXTNN2    26         R
208 CO          NXTNA3    30         R
209 CO          NOWNXT    41         R
0022 O          L1        44 "*"        +002
    
```

} From ENH017C2

Table 14-1 describes some of the lines of the source program.

Table 14-1. Explanation of Program

Line Number	Description
0001	Invokes the copylib preprocessor.
0003	Inserts the Header spec from the RPGHEADR.RPG file.
0006	Inserts the two Input specs from the ENH017C1 file.
0007	Modifies one of the inserted Input specs by adding "L1" in columns 59 through 60.
0020	Inserts specifications from the ENH017C2 file. These include a File Description spec that is inserted after line 0004, and several Output specifications that are inserted in place after line 0020. Note that the "O*" line (second line of ENH017C2) is needed for the next line (\$CONTROL RSPACE=1) to be placed in Output specs rather than after the File Description spec. Refer to "Inserting Copylib Records" in this section for more details.
0021	Modifies the File Description spec just inserted by blanking out the Overflow Indicator field in columns 33 through 34. Blanking is done by the "&" in column 33 (the first column of the field), and by changing the Device Class Name field to "\$STD LST". Although the source line appears out of sequence (an F-spec included with the O's), the line is in sequence because it is only used to modify an existing line in File Descriptions; the line is <i>not</i> placed with the Output specs.
0018, 0019, 203 through 209, and 0022	Uses Relative End Positions, including some "+/-" variations. The compiler listing shows the RPG-computed ending positions in columns 40 through 43 and notes the use and type of Relative End positions in comments in columns 71 through 74.

COMPATIBILITY WITH OTHER RPG SYSTEMS

APPENDIX

A

Of the various RPG languages used today, RPG/3000 most closely resembles IBM System/3 RPG II, IBM System/360 Disk Operating System (DOS) RPG II, and IBM System/360 Model 20 RPG I. RPG/3000 does differ in some respects from these IBM languages. The following paragraphs describe the major differences.

A-1. FUNCTIONAL DIFFERENCES REQUIRING CONVERSION

Complete compatibility of a source program written for another RPG compiler with the RPG/3000 compiler may require some source code conversion, depending on the RPG features that the source program uses. The features that necessitate conversion are described below:

A-2. Printer Files

If your RPG programs reference any carriage control tape channel other than Channel 1 for output files directed to a line printer, you should include Line Counter Specifications in that program to equate each channel referenced to a particular line. If you do not, the compiler will equate Channel 1 to Line 6, and Channels 2 through 12 to the product obtained by multiplying the channel number by 5; it will also equate the overflow line to Line 60.

A-3. Card Reader/Punch/Interpreter

RPG/3000 supports a combined card reader/punch/interpreter that provides two input hoppers and two output stackers. Because the compiler treats this unit as one logical device (or file), however, the program can read input from only one hopper. (It can, however, transmit output to either stacker.) Your program can use this one file for input, output, interpretation or any combination of these three functions.

A-4. Edit Words

In the cases where handling of edit words by RPG/3000 differs from other systems, RPG/3000 does the following:

- Treats all blanks in edit words as replaceable characters, and requires that you use an ampersand (&) in the edit word to produce a blank in the edit field.
- Allows constants at the right of an edit word.
- Does not use the floating dollar sign as a replaceable character.
- Treats all fields containing only zeros as positive values.
- Permits the field being edited to contain more or fewer digits than there are replaceable characters in the edit word; the field will either be truncated or leading zeros will be added to it, accordingly.
- Prints all constants that follow a significant digit (except for a minus (-) or credit (CR) sign that follows a positive number).

Compatibility With Other RPG Systems

A-5. Differences in Character Codes

The HP 3000 Computer accepts source code based upon the American Standard Code for Information Interchange (ASCII) Character Set/Collating Sequence. For HP 3000 systems configured to support devices equipped with Katakana characters, this source code is the Japanese Industrial Standard (JIS). However, the IBM System/3 and System/360 use Extended Binary Coded Decimal Interchange Code (EBCDIC, and in Japan, EBCDIK). This difference results in:

- Alphabetic characters having a higher place in the collating sequence than numeric characters. This could affect compare and matching field operations. However, with a Control Record Specification option, you can request RPG to automatically generate an EBCDIC or EBCDIK alternate collating sequence table. Furthermore, with a File Description Specification option, you can request RPG to automatically generate file translation tables for EBCDIC-to-ASCII (EBCDIK to JIS) or ASCII-to-EBCDIC (JIS to EBCDIK) conversion.
- The need to convert all existing translation tables (including alternate collating sequence and file translation tables) to ASCII (JIS) equivalents.
- The possibility that Move Zone operations specified in an RPG/3000 program would yield different results (for special characters) than the same operations specified in an RPG program for an EBCDIC or EBCDIK machine.

A-6. Device Class Names

Because device class names can be arbitrarily defined during system generation on the HP 3000, RPG/3000 accepts **any** device class names rather than the specific names required on IBM's System/3 and System/360. When specifying a device class name on the File Description Specification Sheet, be sure to use the same name defined during system generation for the device you wish to reference. For existing programs written for other machines, this requirement could necessitate converting the device class name specified in the source code for use on the HP 3000.

A-7. Rewind Operations

RPG/3000 does not support all tape-rewinding operations offered by other RPG's because, on the HP 3000, rewinding is controlled by MPE/3000 according to the device type referenced.

A-8. Quotation Marks

Most other RPG's accept only single quotation marks as delimiters for constants and edit words. But, because other HP 3000 language processors permit use of double quotation marks for these entries, RPG/3000 allows use of either double or single quotation marks in source programs. When compiling programs using single quotation marks, however, you must specify the single-quote option by entering a \$CONTROL compiler subsystem command containing the parameter QUOTE='. (Compiler subsystem commands are discussed in Section XII.)

A-9. File and Program Names

RPG/3000 recognizes only file and program names that begin with a letter (A-Z) followed by letters (A-Z) or digits (0 through 9). File names may contain up to eight characters. Fully-qualified file names reference a file, a group, and an account, in this format: FILENAME.GROUPNAME.ACCOUNTNAME (as described in the MPE reference manuals). Program names may contain up to six characters.

A-10. NON-SUPPORTED FEATURES

RPG/3000 does not support the following features:

A-11. Sterling Notation

This feature would allow Sterling currency (pounds, shillings, pence) notation on input and output. Its absence affects entries in the Control Record, Input, and Output Specifications.

A-12. Telecommunications

This feature would allow input/output through telecommunications facilities. However, RPG/3000 does not currently support telecommunications specifications.

A-13. ULABL Operation

RPG/3000 does not support the ULABL calculation operation. (That operation would allow a field defined in an external subroutine to be referenced in an RPG/3000 program.) RPG/3000, however, does support the RLABL operation code that makes fields, tables, arrays, and indicators used by RPG/3000 programs available to external subroutines, and a PARM operation that allows passing of parameters to and from external subroutines.

A-14. Control Record Operations

Some specifications recognized on the RPG Control Record by other RPG's are not supported by RPG/3000, because they do not apply to this system. These include:

- Core-memory sizes required to compile and execute programs.
- Destination of object program compiled.
- Inquiry request option (to allow/disallow interruption and roll-out of a running program, followed by roll-in of a new program)
- Bypass of the normal halt when the object program attempts to transmit an unrecognizable character to an output device.
- Sharing of a single output area by all of the program's disc files.

A-15. RPG/3000 EXTENSIONS

RPG/3000 extends the features found in IBM's System/3 and System/360 DOS RPG II to include:

A-16. External Subroutine Call Parameters

RPG/3000 allows you to specify parameters after an EXIT (external subroutine call) operation. This simplifies interfacing with COBOL/3000, SPL/3000, BASIC/3000, or FORTRAN/3000 subroutines.

Compatibility With Other RPG Systems

A-17. Run-Time Error Options

RPG/3000 provides three methods for handling run-time errors:

1. Specifying on the Control Record Specification, at compile time, whether the run-time error should be ignored or the program terminated.
2. Letting the operator determine, at run time, how to handle the error.
3. Testing an error code in RPG/3000 calculations and determining within the program how to handle the error.

A-18. Cross-Reference Listing Option

You can request a Cross Reference Listing that shows all references to file names, indicators, and field names.

A-19. Automatic Program Segmentation

RPG/3000 automatically segments code generated for an RPG/3000 program into segments of 1K, 2K, 3K, or 4K words (K = 1024). Since all segments need not exist in memory at the same time, this results in virtually unlimited sizes for object programs.

A-20. EBCDIC/ASCII Translation

You can request RPG/3000 to generate, automatically, file translation tables for EBCDIC-to-ASCII, EBCDIK to JIS or ASCII-to-EBCDIC, JIS to EBCDIK conversions, or to use an EBCDIC or EBCDIK alternate collating sequence.

A-21. Partial Field Translation

You can request RPG/3000 to translate only alphanumeric or unpacked numeric fields, leaving packed numeric or binary fields untranslated.

A-22. Combined Input/Output (Terminal) File

RPG/3000 allows a program to treat a terminal as a single file, and to conduct both read and write operations on that terminal.

A-23. Calculation Indicator Repetition

RPG/3000 does not require you to repeat, line for line, duplicate conditioning indicators in Calculation Specifications.

A-24 Compile-Time Tables/Arrays On Separate Disc Files

If you wish tables/arrays compiled into your program to be the same tables/arrays used for many other programs, you need specify them only once and then reference each with a Table/Array File Name (A) Record in each program.



DIAGNOSTIC MESSAGES

APPENDIX

B

During compilation, preparation, and execution of an RPG/3000 program, three types of messages may be issued to help you diagnose various types of errors:

1. Compile-Time Messages
2. Run (Execution)-Time Messages
3. Operating System Messages

Compile-Time and Run-Time messages are discussed in this appendix. Operating System Messages, generated by MPE/3000, are discussed in the MPE reference manuals.

B-1. COMPILE-TIME MESSAGES.

Whenever RPG/3000 detects an error during compilation, it prints an appropriate error number on the source program listing to the right of the record (line) that contains the error. The number will be followed by the suffix W (for a non-fatal warning) or T (for a fatal error that terminates the program). Then, following the Symbol Table (and optional Cross Reference Listing), RPG/3000 prints an error summary showing, for each error encountered:

1. The error number,
2. A message describing the error, and
3. The number of the line in which the error appears.

EXAMPLE

An example of an error indication appears in Line 34 of the program listing shown in Figure B-1. This error is identified by the error number 527W (Item 1), where the suffix W shows that the error is not fatal. The error is described further in a detailed message (Item 2), **NO FIELD DESCRIPTIONS FOR PREVIOUS RECORD.**

For a discussion of any particular message, please see Tables B-1 through B-10. These tables show the error numbers, their type (T or W), their related messages, their probable causes, and what you should do in response to the messages. In these tables, the error messages are listed in ascending order according to their corresponding error message numbers. The error message numbers, in turn, are grouped according to the type of specification or record to which they pertain, as follows:

Specification/Record Type	Error Number	Table Number
Common (Any Specification or Record)	0 through 99	B-1
Control Record (H)	100 through 199	B-2
File Description (F)	200 through 299	B-3

Diagnostic Messages

```

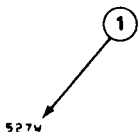
PAGE 0001  HEWLETT PACKARD 3210A.1P.03  RPG/3000  TUE, NOV 19, 1974,  4:11 PM

0001      %CONTROL USLIMIT, CODE
0002      00010%CONTROL QUOTE=
0003      00020H                                X  S

0004      00030FLETIN  IPF F  80  80          READ          FPT004
0005      00040FRPGFILE20  F 120 120        PRINTER        FPT004
0006      00050FRPGFILE30  F  80  80          PUNCH          FPT004

0007      00060F                LN          64  1          FPT004
0008      00070F                LT          2400 1          FPT004
0009      00080F                IA          15  1          FPT004
0010      00090F                IB          15  1          FPT004
0011      00100F                IC          15  1          FPT004
0012      00110F                ID          15  1          FPT004
0013      00120F                IF          15  1          FPT004
0014      00130F                PT          75  1          FPT004
0015      00140F                IN          15  1          FPT004

0016      00150IFLETIN  AA  01  1  CD          2  20  DATE          FPT004
0017      00160I                21  25  PUNCH          FPT004
0018      00170I                RB  02  1  CL          2  2  SPACE          FPT004
0019      00180I                3  6  MATLEVLIM1          53          FPT004
0020      00190I                3  6A  LN          FPT004
0021      00200I                CA  06  1  C%  2  CA          5  19  IA          FPT004
0022      00210I                CB  07  1  C%  2  CB          5  19  IA          FPT004
0023      00220I                CC  08  1  C%  2  CC          5  19  IA          FPT004
0024      00230I                CD  09  1  C%  2  CD          5  19  IA          FPT004
0025      00240I                CE  10  1  C%  2  CE          5  19  IA          FPT004
0026      00250I                XX  LR  1  C/  2  C%  3  C          5  19  IF          FPT004
0027      00260I                DD  05  1  CE  2  CD  3  CI          FPT004
0028      00270I                XX  LR  1  C/  2  C%  3  C          FPT004
0029      00280I                DD  05  1  CE  2  CD  3  CI          FPT004
0030      00290I                XX  LR  1  C/  2  C%  3  C          FPT004
0031      00300I                DD  05  1  CE  2  CD  3  CI          FPT004
0032      00310I                XX  LR  1  C/  2  C%  3  C          FPT004
0033      00320I                DD  05  1  CE  2  CD  3  CI          FPT004
0034      00330C. CHECK FOR DATE CARD          FPT004
0035      00340C. CHECK FOR DATE CARD          FPT004
0036      00350C. CHECK FOR DATE CARD          FPT004
0037      00350C. CHECK FOR DATE CARD          FPT004
    
```



```

PAGE 0017  RPG0RJ          ERRORS DURING COMPILATION

527W  NO FIELD DESCRIPTIONS FOR PREVIOUS RECORD.
646W  DEBUG OPTION NOT SPECIFIED IN COL 15 OF CONTROL CARD
721W  OUTPUT INDICATORS IN COL 23-31 MISSING OR ALL NEGATIVE.
901W  INDICATOR DEFINED BUT NOT REFERENCED

NO. SERIOUS ERRORS 000      NO. WARNINGS 004
PROCESSOR TIME=0100129:    ELAPSED TIME=0102129
    
```



Figure B-1. Compile-Time Error Messages

DIAGNOSTIC MESSAGES

APPENDIX

B

During compilation, preparation, and execution of an RPG/3000 program, three types of messages may be issued to help you diagnose various types of errors:

1. Compile-Time Messages
2. Run (Execution)-Time Messages
3. Operating System Messages

Compile-Time and Run-Time messages are discussed in this appendix. Operating System Messages, generated by MPE/3000, are discussed in the MPE reference manuals.

B-1. COMPILE-TIME MESSAGES.

Whenever RPG/3000 detects an error during compilation, it prints an appropriate error number on the source program listing to the right of the record (line) that contains the error. The number will be followed by the suffix W (for a non-fatal warning) or T (for a fatal error that terminates the program). Then, following the Symbol Table (and optional Cross Reference Listing), RPG/3000 prints an error summary showing, for each error encountered:

1. The error number,
2. A message describing the error, and
3. The number of the line in which the error appears.

EXAMPLE

An example of an error indication appears in Line 34 of the program listing shown in Figure B-1. This error is identified by the error number 527W (Item 1), where the suffix W shows that the error is not fatal. The error is described further in a detailed message (Item 2), **NO FIELD DESCRIPTIONS FOR PREVIOUS RECORD.**

For a discussion of any particular message, please see Tables B-1 through B-10. These tables show the error numbers, their type (T or W), their related messages, their probable causes, and what you should do in response to the messages. In these tables, the error messages are listed in ascending order according to their corresponding error message numbers. The error message numbers, in turn, are grouped according to the type of specification or record to which they pertain, as follows:

Specification/Record Type	Error Number	Table Number
Common (Any Specification or Record)	0 through 99	B-1
Control Record (H)	100 through 199	B-2
File Description (F)	200 through 299	B-3

Diagnostic Messages

```

PAGE 0001  HEWLETT PACKARD 3210AA,IP,03 RPG/3000  TUE, NOV 19, 1974,  4:11 PM

0001      CONTROL USLIMIT, CODE
0002      00010%CONTROL QUOTE=
0003      00020H                                X  S

0004      00030FLETIN  IPF F  80  80          READ          FPT004
0005      00040FPPGFIL20 F 120 120          PRINTER        FPT004
0006      00050FPPGFIL30 F  80  80          PUNCH           FPT004

0007      00060F                LN          64  1          FPT004
0008      00070F                LY          2400 1          FPT004
0009      00080F                IA           15  1          FPT004
0010      00090F                IB           15  1          FPT004
0011      00100F                IC           15  1          FPT004
0012      00110F                IO           15  1          FPT004
0013      00120F                IF           15  1          FPT004
0014      00130F                PT           75  1          FPT004
0015      00140F                IN           15  1          FPT004

0016      00150FLETIN  AA  01  1 CD          2  20 DATE        FPT004
0017      00160F                21  25 PUNCH        FPT004
0018      00170F                RB  02  1 CL          2  2 SPACE        FPT004
0019      00180F                3  3 MATLEVLM1      53          FPT004
0020      00190F                3  66 LN          FPT004
0021      00200F                CA  06  1 C%  2 CA      5  19 IA          FPT004
0022      00210F                CB  07  1 C%  2 CB      5  19 IA          FPT004
0023      00220F                CC  08  1 C%  2 CC      5  19 IC          FPT004
0024      00230F                CD  09  1 C%  2 CD      5  19 ID          FPT004
0025      00240F                CE  10  1 C%  2 CE      5  19 IF          FPT004
0026      00250F                DO  05  1 CE  2 CO  3 CI      5  19 IF          FPT004
0027      00260F                XX LR  1 C/  2 C*  3 C          FPT004
0028      00270F                NO. SERIOUS ERRORS 000          FPT004
0029      00280F                NO. WARNINGS 004           FPT004
0030      00290F                PROCESSOR TIME*01001291          FPT004
0031      00300F                ELAPSED TIME*0102129          FPT004
0032      00310F                527W
0033      00320F                646W
0034      00330F                781W
0035      00340F                901W
0036      00350F                CHECK FOR DATE CARD
0037      00360F

```

1

527W

```

PAGE 0017  RPG00J          ERRORS DURING COMPILATION

527W  NO FIELD DESCRIPTIONS FOR PREVIOUS RECORD,          0034
646W  DEBUG OPTION NOT SPECIFIED IN COL 15 OF CONTROL CARD  0105
781W  OUTPUT INDICATORS IN COL 23-31 MISSING OR ALL NEGATIVE, 0235
901W  INDICATOR DEFINED BUT NOT REFERENCED                 0241

NO. SERIOUS ERRORS 000      NO. WARNINGS 004
PROCESSOR TIME*01001291    ELAPSED TIME*0102129

```

2

Figure B-1. Compile-Time Error Messages

Specification/Record Type	Error Number	Table Number
File Extension (E)	300 through 399	B-4
Line Counter (L)	400 through 499	B-5
Input (I)	500 through 599	B-6
Calculation (C)	600 through 699	B-7
Output (O)	700 through 799	B-8
Compiler Subsystem Command (\$)	800 through 899	B-9
Other Errors (not listed at the right of the pertinent statement)	900 through 999	B-10

Table B-1. Common Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
01	W	EXPECTS \$ CONTINUATION, RECORD IGNORED	You did not enter a complete compiler subsystem command on this record (or card image).	The compiler ignores this record. If necessary, correct the record.
02	T	SEQUENCE NUMBER ERROR	You did not enter the records making up the source program in the proper sequence (as indicated by the Sequence Number Field, Cols. 1-5).	Arrange the source program properly, and re-compile it.
03	T	NON-EXISTENT SPEC TYPE	You specified an entry in the Form Type Field (column 6) that indicates a type of specification not existing in RPG.	Enter the correct type and re-compile.
04	T	SPEC TYPE OUT OF ORDER	You submitted a source program with at least one specification record out of proper sequence (according to type). For example, with a File Description Specification before the Control Record Specification.	Re-arrange and re-compile the source program.

Common Errors

Table B-1. Common Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
06	T	ENTRY OR HEADER-SPACE OVERFLOW IN USL FILE	You specified a user subprogram library that was not large enough to contain all program entries or headers.	Define a larger USL file and re-compile. If you used the default value, you must explicitly specify a USL file when you re-compile (:RPGGO or :RPGPREP).
07	T	CODE GENERATION ERROR	Compiler error.	Inform HP Customer Engineering.
08	T	USL GENERATION ERROR	Compiler error.	Inform HP Customer Engineering.
09	T	RPG CODE GENER- ATION ERROR	Compiler error.	Inform HP Customer Engineering.
10	T	SYMBOL TABLE AREA (DB TO DL) OVERFLOW	You specified, in your source program, too many symbols for the RPG Symbol Table to hold.	Define a larger DB-DL area. If DB-DL is presently 32K, try to reduce number of symbols in your program.
11	T	MORE THAN 1085 ERRORS AND WARNINGS FOUND, END COMPILE	Statements missing or out of order.	Correct program and re-compile.



Table B-2. Control Record (H) Specification Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
101	W	MORE THAN ONE HEADER SPEC, SPECIFICATION DROPPED.	You included more than one Control Record Specification in your source program.	The compiler ignores any Control Record Specification after the first one. Remove the extra specifications.
102	W	INVALID ERROR DUMP FILENAME IN COLUMNS 7-14, ASSUME BLANK.	You entered an invalid filename in the Error Dump Filename Field (Cols. 7-14).	The compiler ignores your entry and assumes you left these columns blank.
103	W	COLUMN 15 (DEBUG OPTION) NOT BLANK OR 1, ASSUME 1.	You specified a character other than 1 in the Debug Field (Col. 15).	The compiler ignores your entry and assumes you entered a 1, specifying execution of DEBUG operations. If you do not want DEBUG operations, correct this record.
104	W	COLUMNS 18-19 NOT BLANK, ASSUME BLANK.	You entered data in column 18-19, (not used in RPG/3000).	The compiler ignores your entry and assumes you left these columns blank.
105	W	COLUMN 21 (INVERTED PRINT) NOT BLANK, I, J OR D, ASSUME BLANK.	You entered a character other than an I, J, or D in the Inverted Print Field (Col. 21).	The compiler ignores your entry and assumes you left it blank, specifying Domestic Format. If you do not want Domestic Format, change this entry to I, J, or blank.
106	W	COLUMNS 23-24 NOT BLANK, ASSUME BLANK.	You entered data in Columns 23-24, (not used by RPG/3000).	The compiler ignores your entry and assumes you left these columns blank.
107	W	COLUMN 26 (ALTERNATE COLLATING SEQUENCE) NOT BLANK, S OR O, ASSUME BLANK.	You specified a character other than an S or O, in the Alternate Collating Sequence Field (Col. 26).	The compiler ignores your entry and assumes you left the field blank, specifying no alternate sequence. If you wish an alternate collating sequence, change the entry to S or O.

H Control Record Specification Errors

Table B-2. Control Record (H) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
108	W	COLUMNS 29-33 NOT BLANK, ASSUME BLANK.	You entered data in Columns 29-33, (not used by RPG/3000).	The compiler ignores your entry and assumes you left these columns blank.
109	W	COLUMN 34 (BINARY SEARCH OPTION) NOT BLANK OR B, ASSUME B.	You specified a character other than B in the Table/Array Look-Up Field (Col. 34).	The compiler ignores your entry and assumes you entered a B for Binary Look-Up. If you do not wish a binary look-up, change this entry to blank.
110	W	COLUMNS 35-38 NOT BLANK, ASSUME BLANK.	You entered data in Columns 35-38, (not used by RPG/3000).	The compiler ignores your entry and assumes you left these columns blank.
111	W	COLUMN 40 (SIGN CHECK) NOT BLANK, B, S, I, O OR N, ASSUME BLANK.	You entered a character other than a blank or B, S, I, O OR N in the Sign Process Field (Col. 40).	The compiler ignores your entry and assumes you left the field blank, specifying sign-forcing on output. If you wish other sign-forcing or no sign-forcing, specify S, I, O OR N.
112	W	COLUMN 41 (1P FORMS POSITIONING) NOT BLANK OR 1, ASSUME 1.	You entered a character other than a 1 in the Form Positioning Field (Col. 41).	The compiler ignores your entry and assumes you entered a 1, specifying form position verification. If you do not wish form positioning, correct the entry.
113	W	COLUMN 42 (INDICATOR INITIALIZATION) NOT BLANK OR S, ASSUME S.	You entered a character other than S in the Indicator Setting Field (Col. 42).	The compiler ignores your entry and assumes you entered an S, specifying that the IP, L0, and all field indicators be set on. Correct the entry, if necessary.
114	W	COLUMN 43 (FILE TRANSLATION) NOT BLANK, F OR O, ASSUME BLANK.	You entered a character other than an F or O, in the File Translation Field (Col. 43).	The compiler ignores your entry and assumes you left the field blank, specifying no translation. If you wish file translation, correct the entry to F or O.

Table B-2. Control Record (H) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
115	W	COLUMN 45 NOT BLANK, ASSUME BLANK.	You entered data in Column 45, (not used by RPG/3000).	The compiler ignores your entry and assumes you left this column blank.
116	W	COLUMN 47 (SKIP TO CHANNEL 1 SUPPRESS) NOT BLANK OR S, ASSUME S.	You entered a character other than an S, in the Skip-Suppress field (Col. 47).	The compiler ignores your entry and assumes you entered an S, specifying skip suppress. If you do not wish skip suppress, change the entry to blank.
117	W	COLUMN 48 (DSPLY OPTIONS) NOT BLANK, B, D, N OR S, ASSUME BLANK.	You specified an incorrect entry for Column 48.	The compiler assumes that the entry was blank.
118	W	COLUMN 52 (CROSS REFERENCE) NOT BLANK OR X, ASSUME X.	You entered a character other than an X, in the Cross-Reference Listing Field (Col. 52).	The compiler ignores your entry and assumes you entered an X, for printing a Cross-Reference Listing. If you do not want a Cross-Reference Listing, enter a blank in Col. 52.
119	W	COLUMN 53 (CARRIAGE CONTROL TYPE) NOT BLANK OR L, ASSUME L.	You entered a character other than L in the Carriage Control Type Field (Col. 53).	The compiler ignores your entry and assumes you entered an L, so that skip requests refer to line numbers. If your skip requests refer to channel numbers, leave this column blank.
120	W	COLUMN 54 (TEXT SEQUENCE CHECK) NOT BLANK, N OR S, ASSUME BLANK.	You entered a character other than an N or S in the Textfile Sequence Check Field (Col. 54).	The compiler ignores your entry and assumes you left the field blank for no sequence checking. If you wish sequence-checking, enter S in Col. 54.

H Control Record Specification Errors

Table B-2. Control Record (H) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
121	W	COLUMN 55 (RUN TIME ERROR CONTROL) NOT BLANK, N OR S, ASSUME S.	You entered a character, other than an N or S in the Error Log Field (Col. 55).	The compiler ignores your entry and assumes you entered an S, for transmitting error messages to the operator and providing a dump.
122	W	COLUMN 56 (RUN TIME ERROR OPTION) NOT BLANK OR 1-5, ASSUME 5.	You entered a character other than 1-5, in the Error Response Field (Col. 56).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
123	W	COLUMN 57 (RUN TIME ERROR OPTION) NOT BLANK OR 1-5, ASSUME 5.	You entered a character other than 1-5, in the Error Response Field (Col. 57).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
124	W	COLUMN 58 (RUN TIME ERROR OPTION) NOT BLANK OR 1-5. ASSUME 5.	You entered a character other than 1-5, in the Error Response Field (Col. 58).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
125	W	COLUMN 59 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5. ASSUME 5.	You entered a character other than 0 or 2-5, in the Error Response Field (Col. 59).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
126	W	COLUMN 60 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5. ASSUME 5.	You entered a character other than 0 or 2-5, in the Error Response Field (Col. 60).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.

Table B-2. Control Record (H) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
127	W	COLUMN 61 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5. ASSUME 5.	You entered a character other than 0 or 2-5, in the Error Response Field (Col. 61).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
128	W	COLUMN 62 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5, ASSUME 5.	You entered a character other than 0-5, in the Error Response Field (Col. 62).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
129	W	COLUMN 63 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5, ASSUME 5.	You entered a character other than 0-5, in the Error Response Field (Col. 63).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
130	W	COLUMN 64 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5. ASSUME 5.	You entered a character other than 0-5, in the Error Response Field (Col. 64).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
131	W	COLUMN 65 (RUN TIME ERROR OPTION) NOT BLANK, 0, OR 2-5. ASSUME 5.	You entered a character other than 0 or 2-5, in the Error Response Field (Col. 65).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
132	W	COLUMN 66 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5. ASSUME 5.	You entered a character other than 0 or 2-5, in the Error Response Field (Col. 66).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.

H Control Record Specification Errors

Table B-2. Control Record (H) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
133	W	COLUMN 67 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5, ASSUME 5.	You entered a character other than Ø or 2-5, in the Error Response Field (Col. 67).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
134	W	COLUMN 68 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5, ASSUME 5.	You entered a character, other than Ø or 2-5, in the Error Response Field (Col. 68).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
135	W	COLUMN 69 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5, ASSUME 5.	You entered a character, other than Ø or 2-5, in the Error Response Field (Col. 69).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
136	W	COLUMN 70 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5, ASSUME 5.	You entered a character, other than Ø or 2-5, in the Error Response Field (Col. 70).	The compiler ignores your entry and assumes you entered a 5, for re-directing or suppressing error messages, printing dump, and ending program.
137	W	COLUMN 71 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5, ASSUME 5	You entered a character other than Ø or 2-5 in the Error Response Field (Col. 71).	The compiler ignores your entry and assumes you left this column blank.
138	W	COLUMNS 75-80 (PROGRAM NAME) INVALID, ASSUME RPGOBJ.	Your program name did not begin with an alphabetic character (A-Z), and/or the remaining characters were not alphanumeric (A-Z, 0-9), and/or they contained embedded blanks.	The compiler ignores your entry and assumes you assigned the name RPGOBJ. Correct the entry if necessary.

Table B-2. Control Record (H) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
139	W	NO HEADER SPEC FOR RPG PROGRAM, ASSUME BLANK HEADER SPEC.	You left the Control Record Specification out of your RPG program.	The compiler assumes you had a blank Control Record Specification. Correct the specification if necessary.
140	W	COLUMNS 56-70 NOT BLANK WHEN COLUMN 55 CONTAINS S, ASSUME BLANK.	You entered data in the Error Response Field (Col. 56-70) when the Error Log Field (Col. 55) contained an S.	The compiler ignores your entry and assumes you left Cols. 56-70 blank.
141	W	COLUMNS 72-74 NOT BLANK, ASSUME BLANK.	You entered data in Columns 72-74, (not used by RPG/3000).	The compiler ignores your entry and assumes you left these columns blank.
142	W	COLUMN 16 (U-SWITCH SOURCE) NOT BLANK, J OR F, ASSUME BLANK.	You specified an incorrect entry for Column 16.	The compiler assumes that the entry was blank.
143	W	COLUMN 17 (UPDATE SOURCE) NOT BLANK OR F, ASSUME BLANK.	You specified an incorrect entry for Column 17.	The compiler assumes that the entry was blank.
144	W	COLUMN 27 NOT BLANK, ASSUME BLANK.	You entered a character other than a blank in Column 27 (not used by RPG/3000).	The compiler assumes that the entry was blank.
145	W	COLUMN 28 (BUFCHK DE-FAULTS) NOT BLANK, C, N, B, U OR X, ASSUME BLANK.	You specified an incorrect entry for Column 28.	The compiler assumes that the entry was blank.
146	W	COLUMN 25 (NAME LOGGING) NOT BLANK OR L, ASSUME BLANK.	You specified an incorrect entry for Column 25.	The compiler assumes that the entry was blank.
147	W	COLUMN 39 (EBCDIC ZONE/DIGIT TESTS) NOT BLANK OR E, ASSUME BLANK.	You specified an incorrect entry for Column 39.	The compiler assumes that the entry was blank.

H Control Record Specification Errors

Table B-2. Control Record (H) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
148	W	COLUMN 44 (NON-NUMERIC DIGITS) NOT BLANK OR N, ASSUME BLANK.	You specified an incorrect entry for Column 44.	The compiler assumes that the entry was blank.
149	W	COLUMN 22 (RECORD NUMBER ADJUST) NOT BLANK, 1, + OR 0, ASSUME BLANK.	You specified an incorrect entry for Column 22.	The compiler assumes that the entry was blank.

Table B-3. File Description (F) Specification Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
201	T	FILE NAME PREVIOUSLY DEFINED IN COL. 7-14, SPEC IS DROPPED.	You assigned the file name used in the Filename Field (Col. 7-14) in a previous File Description record.	The compiler ignores the specification. Correct the file name and re-compile.
202	T	INVALID FILETYPE ENTRY IN COL. 15, ASSUME I.	You entered an invalid filetype entry in the File Type Field (Col. 15).	The compiler ignores your filetype entry and assumes you entered an "I" for Input File. If you do not intend an input file, change the entry to O, U, D or C.
203	W	FILE DESIGNATION IN COL. 16 IS INVALID FOR FILETYPE, ASSUME SECONDARY.	You specified a file designation in the File Designation Field (Col. 16) that is invalid for the filetype.	The compiler ignores your file designation entry and assumes you designated a secondary file. If you do not intend a secondary file, change the entry to P, R, C, T, D, or blank.
204	W	INVALID END OF FILE ENTRY IN COL. 17, ASSUME BLANK.	You entered an invalid end-of-file entry in the End-of-File Field (Col. 17).	The compiler ignores your end-of-file entry and assumes you left it blank.
205	W	INVALID SEQUENCE ENTRY IN COL. 18, ASSUME BLANK.	You entered an invalid sequence entry in the Input Sequence Check Field (Col. 18).	The compiler ignores your sequence entry and assumes you left it blank, for sequence check of matching file in ascending order. If you wish sequence checking in descending order, change this entry to D.
206	W	FILE FORMAT IN COL. 19 IS INVALID, ASSUME F.	You entered an invalid file format in the Record Format Field (Col. 19).	The compiler ignores your entry and assumes the records in the file are Fixed-length.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
207	W	INVALID BLOCK LENGTH IN COL. 20-23, ASSUME EQUAL TO RECORD LENGTH.	You entered a block length in the Block Length Field (Col. 20-23) other than 1 through 9999, or blank.	The compiler ignores your entry and assumes the block length and record length are equal. Correct the block length if necessary.
208	W	INVALID RECORD LENGTH IN COL. 24-27, ASSUME RECORD LENGTH OF 80.	You entered a record length in the Logical Record Length Field (Col. 24-27) other than 1 through 9999.	The compiler ignores your entry and assumes the record length is 80 characters. Correct the record length.
209	T	INVALID MODE OF PROCESSING ENTRY IN COL. 28.	You entered a character other than an R, in the Processing Mode Field (Col. 28).	The compiler ignores your entry and assumes sequential processing. Correct the processing mode to L or R.
210	T	INVALID LENGTH OF KEY FIELD OR RECORD ADDRESS FIELD IN COLUMNS 29-30.	You entered a key length other than 1 through 99 or blank in the Key or Record Address Field (Cols. 29-30).	The compiler ignores your entry and assumes that the length is equal to 3.
211	W	INVALID RECORD ADDRESS TYPE ENTRY IN COLUMN 31, ASSUME BLANK.	You entered a character other than an I, in the Record Address Type Field (Column 31).	The compiler ignores your entry and assumes you left it blank, for direct-access file not processed by chaining file or RAF. To specify processing with a chaining file or RAF, change the entry to I.
212	W	INVALID ENTRY IN COL. 32, ASSUME BLANK.	You entered a character other than a T, D or 1 through 7, in the File Organization/Additional I/O Area Field (Col. 32).	The compiler ignores your entry and assumes you left it blank, requesting two buffers.
213	W	INVALID OVERFLOW INDICATOR IN COL. 33-34, ASSUME BLANK.	You entered an invalid overflow indicator in the Overflow Indicator Field (Col. 33-34).	The compiler ignores your entry and assumes you left this field blank (no indicator assigned). To specify an indicator, enter OA-OG, or OV.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
214	W	INVALID KEY FIELD LOCATION IN COLS 35-38.	You entered an invalid Key Field Starting Location Columns 35-38.	The compiler ignores your entry and assumes a Key Field Starting Location of 0.
215	T	IF YOU WISH TO PROCESS YOUR OWN LABELS, COLUMN 53 MUST BE E OR 2-9.	You entered a character other than the letter E or digits 2-9, to process your own labels.	The compiler assumes no user label processing. To process labels, enter E or 2-9.
216	W	COLUMNS 48-49 ARE NOT BLANK.	You entered invalid data in Columns 48-49, (not used by RPG/3000).	The compiler ignores your entry and assumes you left these columns blank.
217	W	INVALID ENTRY IN COL. 53.	You entered a character, other than an S, E or 2 through 9, in the Disc Labels Field (Column 53).	The compiler assumes that you left the entry blank.
218	W	COLUMNS 60-65 ARE NOT BLANK.	You entered characters in these fields but did not enter K (for continuation record) in Column 53.	The compiler ignores your entry and assumes you left these columns blank.
219	W	COLUMN 66 IS NOT 'A' OR BLANK, ASSUME BLANK.	You entered a character, other than an A in the File Addition Field (Col. 66).	The compiler ignores your entry and assumes you left this field blank. To specify appending new records, change this entry to A.
220	W	COLUMN 67 IS NOT BLANK.	You entered a character other than a blank in this column.	The compiler assumes that you entered a blank, requesting 8 disc extents. To request more or less extents, enter the number desired.
221	W	COLUMN 70 IS NOT BLANK.	You entered a character other than a blank in this column.	The compiler assumes that you entered a blank.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
222	W	COLUMN 73-74 ARE NOT BLANK.	You entered a character other than a blank in this column.	The compiler assumes that you entered a blank.
223	W	INCALID FILE CONDITIONING INDICATOR IN COL. 71-72, ASSUME BLANK.	You entered an invalid file conditioning indicator in the File Conditioner Field (Col. 71-72).	The compiler ignores your entry and assumes you left this field blank.
224	W	MULTIPLE PRIMARY FILE DEFINED IN COL. 16, ASSUME SECONDARY.	You defined more than one primary input file in the File Designation Field (Col. 16).	The compiler assumes any primary file defined after the first primary file is a secondary file.
225	T	IF 'SPECIAL' IN COLUMN 40-46, A ROUTINE NAME MUST BE IN COLUMN 54-59.	You didn't enter a subroutine name in the Name of Label Exit Field (Col. 54-59) to process user labels or files on SPECIAL devices.	Enter a routine name in Col. 54-59, or remove SPECIAL from Col. 40-46 and re-compile.
226	W	INVALID NUMBER OF EXTENTS IN COL. 68-69, SYSTEM DEFAULT VALUE IS USED.	You entered a number of extents less than 1 or greater than 16 in the Extents Field (Cols. 68-69).	The compiler ignores your entry and by system default assigns eight extents.
227	T	FILE DESIGNATION ENTRY IN COL. 16 INVALID FOR OUTPUT OR DISPLAY FILE, ASSUME BLANK.	You specified a file designation in the File Designation Field (Col. 16) that is invalid for the output or display file.	The compiler ignores your file designation entry and assumes you left it blank. Set the File Description entry to blank or change File Type to I, U, or C.
228	W	END OF FILE ENTRY IN COL. 17 INVALID FOR FILE TYPE.	You entered a character other than E or blank in the End-of-File (Col. 17) for the file type specified.	The compiler assumes you entered a blank.
229	W	ENTRY IN COL. 18 INVALID FOR TYPE OF FILE OR MODE OF PROCESSING, ASSUME BLANK.	You entered a letter or digit, other than an A or D, in the Input Sequence Check Field (Col. 18) that was invalid for the type of file or mode of processing.	The compiler assumes you entered a blank.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
230	W	OVERFLOW INDICATOR IN COL. 33-34 PREVIOUSLY ASSIGNED, ASSUME BLANK.	You previously assigned an overflow indicator in the Overflow Indicator Field (Col. 33-34).	The compiler ignores your entry and assumes you left this field blank. If you wish overflow processing, enter an unused overflow indicator.
231	W	INVALID OR BLANK EXTENSION CODE ENTRY IN COL. 39 FOR TABLE OR RECORD ADDRESS FILE, ASSUME E.	You entered a character, other than an E or L, in the Extension Code Field (Col. 39) for a Table or Record Address File.	The compiler ignores your entry and assumes you entered an E for a Table File in the File Extension Specification.
232	W	EXTENSION CODE IN COL. 39 IS INVALID, ASSUME BLANK.	You entered a letter or digit, other than an E or L, in the Extension Code Field (Col. 39).	The compiler ignores your entry and assumes you left it blank.
233	W	RECORD LENGTH IS LARGER THAN BLOCK LENGTH, ASSUME BLOCK LENGTH EQUAL TO RECORD LENGTH.	You specified the record length to be larger than the block length.	The compiler ignores your entry and assumes the block length is equal to the record length.
234	W	BLOCK LENGTH IS NOT A MULTIPLE OF RECORD LENGTH, ASSUME UNBLOCK.	You specified that the block length is a multiple of the record length.	The compiler ignores your entry and assumes there is no blocking (the physical and logical record lengths are identical).
235	T	NO FILE DESCRIPTION SPECIFICATION FOUND.	You did not include any File Description Specification with your source program.	Re-compile the program with an appropriate File Description Specification.
236	W	NO PRIMARY FILE SPECIFIED IN COL. 16, ASSUME FIRST SECONDARY FILE AS PRIMARY.	You didn't specify a primary file in the File Designation Field (Col. 16).	The compiler assumes that the first secondary file is the primary file.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
237	W	COLUMNS WHICH SHOULD BE BLANK FOR CONTINUATION LINE ARE NOT BLANK, ASSUME BLANK.	You made an entry in these columns on a File Description Continuation Record.	The compiler assumes these columns are blank.
238	W	CONTINUATION LINE IS NOT ALLOWED AT THIS POINT.	Your first line in the File Description Specifications is a continuation line.	The compiler ignores this continuation line.
239	W	CONTINUATION ENTRY IN COL. 54-59 IS REPEATED FOR A FILE, SECOND ENTRY IGNORED.	You repeated a continuation entry; for example, you specified RDEXIT in two continuation records for the same file.	The compiler ignores the second entry.
240	W	ENTRY IN COL. 54-59 OF A CONTINUATION RECORD IS INVALID OR MISSING.	You entered invalid information in the Option Type Field (Cols. 54-59) or left this field blank.	The compiler ignores this continuation record.
241	W	INVALID FILENAME IN COL. 7-14, SPEC IS DROPPED.	You entered an invalid filename in the File Name Field (Col 7-14).	The compiler drops the specification; re-define the filename and re-compile.
242	T	ONLY ONE RECORD ADDRESS FILE PER PROGRAM IS ALLOWED, ASSUME SECONDARY.	You already specified one record address file in the program.	The compiler assumes the second file designated is a secondary file.
243	W	A RECORD ADDRESS FILE CAN NOT BE A 'SPECIAL' FILE.	You specified a record address file as a SPECIAL file.	The compiler assumes this file is not a SPECIAL file.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
244	W	NO PRIMARY OR SECONDARY FILE SPECIFICATION IN COLUMN 16.	In a program that uses multi-file input, you defined no primary or secondary input files.	Make the appropriate file definitions, and re-compile the program.
245	W	THIS CONTINUATION RECORD IS NOT ALLOWED FOR A NON-IMAGE FILE, SPEC IS DROPPED.	You entered a File Description Continuation record that is only permitted for an IMAGE/3000 file.	The compiler ignores this record.
246	W	ILLEGAL DATA BASE NAME IN COLUMNS 60-65.	You entered an improper IMAGE data base name.	Enter a correct name (an asterisk, followed by a letter, followed by letters or numbers).
247	W	ILLEGAL OPEN MODE IN COLUMN 66, ASSUME 2.	You entered a character other than 1, 2,, 3, or blank in the Open Mode Field (Col. 66).	The compiler assumes Mode 2 — Update Shared. Enter the correct mode, if you wish other than Mode 2.
248	W	ILLEGAL INPUT/OUTPUT MODE IN COLUMN 67.	You entered a character other than 2 through 8, S, B, C, or R, in the Input/Output Mode Field (Col. 67).	The compiler assumes Mode 2. Enter the correct mode, if you wish other than Mode 2.
249	W	RECORD ADDRESS TYPE IN COLUMN 31 MUST BE 'A', 'P', 'K', OR 'I' FOR IMAGE FILE, ASSUME 'A'.	You entered an improper character in the Record Address Type Field (Col. 31).	The compiler assumes you entered A (Alphanumeric Keys).
250	W	RECORD ADDRESS TYPE IN COLUMN 31 MUST BE 'I' FOR DIRECT FILE.	The IMAGE File is a direct-access file, but you specified a record address type other than I (retrieve by record number).	The compiler assumes Mode I. Enter the correct mode, if necessary.
251	T	MODE OF PROCESSING IN COLUMN 28 MUST BE 'R' FOR DIRECT FILE, ASSUME R.	The IMAGE File is a direct-access file, but you specified a processing mode other than R (random).	The compiler assumes you entered R. Enter the correct mode if necessary.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
252	T	RECORD ADDRESS TYPE IN COLUMN 31 MUST BE BLANK FOR SEQUENTIAL OR TAG FILE, ASSUME BLANK.	The IMAGE File is a sequential or TAG file, but you specified a record address type other than blank (sequential).	The compiler assumes you left this field blank. Enter the correct record address type.
254	T	L IN COLUMN 28 CAN ONLY BE USED WITH S, B, C OR R IN COLUMN 67 FOR PREVIOUS FILE.	You specified L (for sequential processing between limits for this file) but a conflicting Input/Output Mode for the previous file.	The compiler assumes column 28 is blank. Enter the correct input/output mode in Column 28.
255	T	ITEM AND/OR ITEMXX NAME MISSING FOR 'L' IN COLUMN 28 FOR PREVIOUS FILE.	You did not supply the required ITEM and ITEMXX continuation records.	The compiler terminates your job at the end of compilation; supply the ITEM and ITEMXX records.
256	T	ITEM NAME MISSING FOR INPUT/OUTPUT MODE 5, 6, C OR R FOR PREVIOUS FILE.	You did not provide the corresponding ITEM continuation record for this chained file.	The compiler terminates your job at the end of compilation; provide the ITEM continuation record and re-compile.
257	T	ITEMXX MISSING FOR INPUT/OUTPUT MODE S OR B FOR PREVIOUS FILE.	You did not provide the corresponding ITEMXX continuation record for this sequential read operation.	The compiler terminates your job at the end of compilation; provide the ITEMXX continuation record and re-compile.
258	T	DATA BASE NAME MISSING FOR PREVIOUS FILE.	You did not provide a Data Base Name Record for this file.	The compiler terminates your job at the end of compilation; provide the Data Base Name Record and re-compile.
259	T	FORMS, BATCH AND TRACE FILES MAY ONLY BE SPECIFIED FOR WORKSTN FILE	You included a FORMS BATCH or TRACE continuation card for a non-WORKSTN File.	Correct the program and re-compile.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
260	T	TRACE FILE NAME MUST BE NON-BLANK AND BEGIN IN COL. 60	Your TRACE file name was missing or did not begin in column 60.	Add the name beginning in column 60.
261	W	WORKSTN FILE MUST BE UPDATE OR COMBINED. ASSUME UPDATE	A file on device WORKSTN was not declared update or combined.	The compiler assumes the WORKSTN file is an update file.
262	W	WORKSTN FILE MUST BE DEMAND OR PRIMARY, ASSUME PRIMARY	A file on device WORKSTN was not declared demand or primary.	The compiler assumes WORKSTN file is primary.
263	W	WORKSTN FILE MUST HAVE VARIABLE LENGTH RECORDS, ASSUME VARIABLE	A file on device WORKSTN was declared with fixed length records.	The compiler assumes the WORKSTN file has variable length records.
264	W	COLUMN 52 MUST BE BLANK FOR FILES NOT ON DEVICE WORKSTN	Column 52 was not blank for a non-WORKSTN file.	The compiler assumes column 52 is blank.
265	W	INVALID ENTRY IN COLUMN 52, ASSUME BLANK	Column 52 contains an invalid entry for a WORKSTN file.	The compiler assumes column 52 is blank.
266	W	COLUMN 51 MUST BE BLANK FOR FILES NOT ON DEVICE WORKSTN	Column 51 was not blank for a non-WORKSTN file.	The compiler assumes column 51 is blank.
267	W	INVALID ENTRY IN COLUMN 51, ASSUME BLANK	Column 51 contains an invalid entry for a WORKSTN file.	The compiler assumes column 51 is blank.
268	W	FILE ON DEVICE \$STDIN MUST BE INPUT, ASSUME INPUT	A file declared on device \$STDIN was not an input file.	The compiler assumes the \$STDIN file is an input file.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
269	W	FILE ON DEVICE \$STDLST MUST BE OUTPUT OR DISPLAY, ASSUME OUTPUT	A file declared on device \$STDLST was not an output or display file.	The compiler assumes the \$STDLST file is an output file.
270	W	KEY FIELD/RECORD ADDRESS FIELD LENGTH MUST BE ENTERED FOR THIS FILE	No key field length was entered for a chaining type file.	Correct the program and re-compile.
271	W	KEY FIELD STARTING LOCATION MUST BE ENTERED FOR THIS FILE	No key field starting location was entered for a chaining file.	Correct the program and re-compile.
272	W	DEVICE NAME NOT LEFT JUSTIFIED IN FIELD, ASSUME JUSTIFIED.	You specified a device name that was not left justified in the device name field. (Columns 40 through 46).	The compiler left justifies the entry.
273	W	FILE SPECIFIED AS CHAINED ACCESS MUST BE RANDOM PROCESSED, ASSUMED R IN COLUMN 28.	You specified a file as chained access, but did not specify random processing.	The compiler assumes you entered R in Column 28.
274	W	INPUT/OUTPUT MODE MUST BE 5, 6, 7, 8, C OR R FOR RECORD LEVEL LOCKING, ASSUME 1 IN COLUMN 66.	You specified an incorrect entry in Column 66.	The compiler assumes you entered 1 in Column 66.
275	W	LOCKING MODE INCONSISTENT FOR DATA BASE, FIRST MODE IS B, MODES AFTER ARE DIFFERENT, ASSUME B.	You specified different locking modes following an initial locking mode of B. This sequence is invalid.	The compiler assumes the mode B for all accesses.
276	W	LOCKING MODE INCONSISTENT FOR DATA SET, FIRST MODE IS S, MODES AFTER ARE DIFFERENT, ASSUME S.	You specified different locking modes following an initial locking mode of S. This sequence is invalid.	The compiler assumes the mode S for all accesses.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
277	W	LOCKING MODE INCONSISTENT, MODE FOLLOWING FIRST LOCK MODE ARE DIFFERENT, ASSUME FIRST MODE	You specified different locking modes following an initial locking mode. This sequence is invalid.	Correct your program for locking mode consistency and re-compile.
278	T	LOCKING MODE INCONSISTENT, FIRST MODE IS S, THE FOLLOWING MODE IS B.	You specified different locking modes following an initial locking mode.	Correct program for locking mode consistency and re-compile.
279	T	LOCKING MODE INCONSISTENT, MODES FOLLOWING FIRST LOCK MODE ARE OF HIGHER PRECEDENCE.	You specified different locking modes following an initial locking mode.	Correct your program for locking mode consistency and re-compile.
280	T	FILE DESCRIBED AS AN OUTPUT CHAIN FILE MUST BE A DIRECT FILE, NOT A KSAM, IMAGE OR INDEXED FILE.	You specified a file as OC in Columns 15-16; therefore, Column 32 must be specified as a direct file.	Correct your program to specify blank, D or 1 through 7 in Column 32, and re-compile.
281	W	FOR UPDATE-ADD FILE W/OUT 'UPDATE-PROTECT CHECK', ENSURE THAT NO UPDATE IS INTERRUPTED BY AN ADD.	You specified a file as update without specifying the update-protect check option in H spec Column 28.	Add the update protect check option (U or X in Column 28) or ensure that your program does not attempt to add a record while an update operation is in progress.
282	W	COLUMN 50 MUST BE BLANK FOR FILES NOT ON DEVICE WORKSTN.	You specified an incorrect entry for Column 50.	The compiler assumes blank.
283	W	INVALID ENTRY IN COLUMN 50, ASSUME BLANK.	You specified an incorrect entry for Column 50.	The compiler assumes blank.
284	W	COLUMN 32 FOR WORKSTN FILE MUST BE BLANK, ASSUME BLANK.	You specified an incorrect entry for Column 32.	The compiler assumes blank.

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
285	W	BUFCHK SPEC ONLY ALLOWED FOR SEQUENTIAL, DIRECT, AND INDEXED FILES, SPEC IS DROPPED.	You specified a BUFCHK continuation record for a file that is not a sequential, direct or indexed file.	The compiler ignores the BUFCHK continuation record.
286	W	BUFCHK SPEC COLUMN 60 (CURRENT DATA CHECK) MUST BE BLANK OR N, ASSUME BLANK.	You specified an incorrect entry for Column 60.	The compiler assumes blank.
287	W	BUFCHK SPEC COLUMN 61 (NO-READ CHECK) MUST BE BLANK OR N, ASSUME BLANK.	You specified an incorrect entry for Column 61.	The compiler assumes blank.
288	W	BUFCHK 'NO-READ CHECK' DISABLED FOR NON-UPDATE FILE.	You specified the NO-READ CHECK option in H spec Column 28, and a BUFCHK continuation record with a blank in Column 61 for a non-update file. This sequence is invalid.	The compiler disables the NO-READ CHECK option for this file.
289	W	BUFCHK SPEC COLUMN 62 (UPDATE-PROTECT CHECK) MUST BE BLANK OR N, ASSUME BLANK.	You specified an incorrect entry for Column 62.	The compiler assumes blank.
290	W	BUFCHK 'UPDATE-PROTECT CHECK' DISABLED FOR NON-UPDATE FILE.	You specified the UPDATE-PROTECT CHECK option in H spec Column 28, and a BUFCHK continuation record with a blank in Column 62 for a non-update file. This sequence is invalid.	The compiler disables the UPDATE-PROTECT CHECK option for this file.
291	W	BUFCHK 'CURRENT DATA CHECK' DISABLED FOR ALL NON-LOCKING FILES HAVING A 'BUFCHK' CONTINUATION REC.	You specified the CURRENT DATA CHECK option in H spec Column 28, and a BUFCHK continuation record for file(s) for which locking is not specified.	The compiler disables the CURRENT DATA CHECK option for the file(s).

Table B-3. File Description (F) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
293	W	FORMS DOWN-LOAD VALUE IN COLUMNS 60-62 MUST BE 1-255, ASSUME 1.	You specified an incorrect entry for Columns 60 through 62. You might not have right-justified your entry.	The compiler assumes 1.
294	W	BUFCHK 'CURRENT DATA CHECK' DISABLED FOR IMAGE FILE.	You specified a BUFCHK continuation record for an IMAGE file. This sequence is invalid.	The compiler disables the CURRENT DATA CHECK option for this file specification.
295	W	KEYFL SPEC CONTAINS INVALID PARAMETER(S), OR IS SPECIFIED FOR A NON-KSAM FILE.	Columns 68 through 70 contained invalid parameter information or you specified the KEYFL continuation record for a non-KSAM file.	The compiler disregards your entry. Correct your program and re-compile.
296	W	WORKSTN INTERFACE TYPE NOT BLANK, R OR C, ASSUME BLANK.	You specified an incorrect entry for Column 47 of F spec.	The compiler assumes blank.

Table B-4. File Extension (E) Specification Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
301	T	TO FILENAME (COLUMNS 19-26) INVALID OR UNDEFINED - RECORD IGNORED.	You entered an invalid or undefined filename in the To Filename Field (Col. 19-26). (An undefined file name is one that does not appear in the File Description Specifications.)	Re-define the To Filename and re-compile.
302	T	FROM FILENAME (COLUMNS 11-18) INVALID OR UNDEFINED - RECORD IGNORED.	You entered an invalid or undefined filename in the From Filename Field (Col. 11-18). (An undefined file name is one that does not appear in the File Description Specifications.)	Re-define the From Filename and re-compile.
303	W	E NOT SPECIFIED ON FILE DESCRIPTIONS FOR FROM FILE.	You didn't specify an E on the File Description Specification (Col. 39) for the From File (Col. 11-18).	Enter an E in Column 39 of the File Description Specification for the From File.
304	T	FROM FILE IS NOT AN INPUT FILE - RECORD IGNORED.	You specified a non-input file in the From Filename Field (Col. 11-18).	Re-specify the correct type of file in the From Filename Field and re-compile.
305	W	CHAINING OR RECORD ADDRESS FILE AND COLS. 33-57 NON-BLANK - BLANK ASSUMED.	You entered data in Cols. 33-57, which should remain blank for chaining or Record-Address Files.	Remove the data from Cols. 37-57.
306	T	CHAINING FIELD NUMBER (COLUMN 10) NOT 1-9 - RECORD IGNORED.	You entered a character other than one of the digits 0 through 9 in Column 10.	Specify the correct digit and re-compile.
307	W	CHAINING AND C NOT SPECIFIED IN COLUMN 9 - ASSUME C.	You didn't specify a C in Column 9 of the Chaining Field Code (Col. 9-10).	The compiler assumes you entered a C in Col. 9, and combines this with the digit in Col. 10 to form the Chaining Code. Correct Col. 9.

Table B-4. File Extension (E) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
308	T	CHAINING CODE AND FROM FILE PREVIOUSLY SPECIFIED - RECORD IGNORED.	You previously specified the chaining code in the Chaining Field Code (Col. 9-10) and the from file in the From Filename Field (Col. 11-18).	The compiler ignores your record. Remove the duplicate chaining description.
309	T	TO FILE NOT A CHAINED OR RANDOM FILE - RECORD IGNORED.	You did not specify a chained or direct-access file in the To Filename Field.	Specify the correct type of record, and re-compile.
310	T	CONVERSION ROUTINE NAME (COLUMNS 27-32) INVALID - BLANK ASSUMED.	You entered an invalid conversion routine name in the Table, Array, or Routine Name Field (Col. 27-32).	The compiler ignores your entry and assumes you left this field blank. Enter a valid conversion routine name or leave this field blank.
311	T	CONVERSION ROUTINE (COLUMNS 27-32) PREVIOUSLY DEFINED AS A FIELD NAME - BLANK ASSUMED.	You previously defined a conversion routine in the Table, Array, or Routine Name Field (Col. 27-32) as a data field.	The compiler ignores your entry and assumes you left Cols. 27-32 blank. Re-name either the conversion routine name or the field name.
313	T	RECORD ADDRESS FILE PREVIOUSLY SPECIFIED ON EXTENSION SPECIFICATION - RECORD IGNORED.	You specified a record address file in a previous File Extension Specification.	Set up program so that there is only one record address file.
314	T	TO FILENAME (COLUMNS 19-26) MISSING FOR CHAINING OR RAF - SPECIFICATION IGNORED.	You didn't specify a filename, in the To Filename Field (Col. 19-26), for the chaining file or the record address file you specified.	Specify the filename and re-compile.

Table B-4. File Extension (E) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
315	T	TABLE/ARRAY NAME (COLUMNS 27-32) PREVIOUSLY DEFINED - RECORD IGNORED.	You previously defined the Table/Array Name in the Table, Array, or Routine Name Field (Col. 27-32).	Re-define the name and re-compile.
316	T	ENTRIES PER RECORD (COLUMNS 33-35) INVALID - ASSUME 1.	You specified an invalid number of entries in one record of the Entries Per Record Field (Col. 33-35). (The number of entries must be from 1 to 999.)	The compiler ignores your entry and assumes you entered a 1. Enter valid numeric characters, right-adjusted without embedded blanks, and re-compile.
317	T	ENTRIES PER TABLE/ARRAY (COLUMNS 36-39) INVALID - ASSUME 1.	You entered an invalid number of entries in the Entries Per Table/Array Field (Col. 36-39).	The compiler ignores your entry and assumes you entered a 1. Enter valid numeric characters, right-adjusted, with no embedded blanks, and re-compile.
318	W	P/B/L/R FIELD (COLUMN 43) INVALID - BLANK ASSUMED.	You entered a letter or digit, other than a P, B, L, or R, in the Data Format Field (Column 43).	The compiler ignores your entry and assumes you left it blank. Enter the correct type of numeric field.
319	W	FIELD SIZE OF 1 NOT ALLOWED WITH L OR R SPECIFIED IN COLUMN 43 - 2 ASSUMED.	You specified a field size of 1 in the Entry Length Field (Col. 40-42) with an L or R format in the Data Format Field (Col. 43).	The compiler ignores your entry and assumes you specified a field size of 2 characters. Specify a field size of at least 2 characters.
320	T	FIELD SIZE (COLUMNS 40-42) BLANK OR INVALID - 1 ASSUMED	You entered an invalid field size or left the Entry Length Field (Col. 40-42) blank.	The compiler ignores your entry and assumes you specified a field size of 1 character. Enter valid numbers, right-adjusted, with no embedded blanks.

Table B-4. File Extension (E) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
321	W	DECIMAL POSITIONS (COLUMN 44) INVALID - ASSUME NUMERIC WITH ZERO DECIMAL POSITIONS.	You entered an invalid number of digit positions to the right of the decimal point in the Decimal Positions Field (Col. 44).	The compiler ignores your entry and assumes a numeric table with zero decimal positions. Enter a digit 0-9, in this field.
322	W	P, B, L OR R SPECIFIED (COLUMN 43) AND BLANK DECIMAL POSITIONS- ASSUME NUMERIC W/ZERO DECIMALS.	You entered a P, B, L, or R in the Data Format Field (Column 43) and left the Decimal Positions Field (Column 44) blank.	The compiler ignores your entry and assumes a numeric table with zero decimal positions. Either enter a blank in Column 43 or a digit, 0-9, in Column 44.
323	T	DECIMAL POSITIONS (COLUMN 44) GREATER THAN NUMBER OF DIGITS - ASSUME EQUAL.	You specified more digits to the right of the decimal in the Decimal Positions Field (Col. 44) than there are positions.	The compiler ignores your entry and assumes that the number of decimal positions equals the number of digit positions. Change either the decimal positions or the number of digits, and re-compile.
324	W	SEQUENCE (COLUMN 45) NOT BLANK, A OR D - A ASSUMED.	You entered a letter or digit, other than an A or D, in the Table/Array Sequence Field (Column 45).	The compiler ignores your entry and assumes you entered an A (for ascending sequence).
325	T	FIELD NAME (COLUMNS 27-32) BLANK OR INVALID - ENTRY IGNORED.	You entered an invalid Table, Array, or Routine Name or left it blank in the Table, Array or Routine Field (Col. 27-32).	The compiler ignores your entry. Enter a valid field name and re-compile.
326	T	BINARY FIELD NOT LENGTH 5 OR 10.	You specified a binary field which isn't either 5 or 10 digits long.	Re-define the field to be either 5 or 10 digits long.
327	T	TO FILE (COLUMNS 19-26) NOT AN OUTPUT FILE - TO FILE IGNORED.	You specified the To File as an Output File in the To Filename Field (Columns 19-26).	The compiler ignores To Filename entry. Enter a valid output file name or blanks, and re-compile.

E File Extension Specification Errors

Table B-4. File Extension (E) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
328	W	P/B/L/R FIELD (COLUMN 55) INVALID - BLANK ASSUMED.	You entered a letter or digit, other than a P, B, L, or R, in the Data Format Field (Column 55).	The compiler ignores your entry and assumes you left the field blank. Correct this field to the proper field format (P, B, L, or R).
329	W	FIELD SIZE OF 1 NOT ALLOWED WITH L OR R SPECIFIED IN COLUMN 55 - 2 ASSUMED.	You specified a field size of 1 in the Entry Length Field (Col. 52-54) with an L or R format in the Data Format Field (Col. 55).	The compiler ignores your entry and assumes you specified a field size of 2 characters. Enter a field size of at least two positions.
330	T	FIELD SIZE (COLUMNS 52-54) BLANK OR INVALID - 1 ASSUMED.	You entered an invalid field size or left the Entry Length Field (Col. 52-54) blank.	The compiler ignores your entry and assumes you specified a field size of 1 character. Enter valid numeric characters, right-adjusted, with no embedded blanks.
331	T	ALPHA FIELD SIZE GREATER THAN 256 - ASSUME 256.	You entered an alphabetic field size in the Entry Length Field (Col. 40-42 or Col. 52-54) greater than 256 characters.	The compiler ignores your entry and assumes you entered 256 characters. Enter a field size of 256 or less.
332	T	NUMERIC FIELD SIZE GREATER THAN 15 - ASSUME 15.	You entered a numeric field size in the Entry Length Field (Col. 40-42 or Col. 52-54) greater than 15 digits.	The compiler ignores your entry and assumes you entered 15 digits. Either define this field as an alphanumeric field, or change the number of digits to 15 or less.
333	W	DECIMAL POSITIONS (COLUMN 56) INVALID - ASSUME NUMERIC WITH ZERO DECIMAL POSITIONS.	You specified an invalid number of positions to the right of the decimal point in the Decimal Positions Field (Col. 56).	The compiler ignores your entry and assumes numeric data with zero decimal positions. Enter the correct number of decimal positions (0-9, or blank).

Table B-4. File Extension (E) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
334	W	P, B, L OR R SPECIFIED (COLUMN 55) AND BLANK DECIMAL POSITIONS - ASSUME NUMERIC W/ZERO DECIMALS.	You specified a P, B, L or R in the Data Format Field (Col. 55) and left the Decimal Positions Format Field (Col. 56) blank.	The compiler ignores your entry and assumes a numeric table with zero decimal positions. Either enter a blank in Col. 55 or 0-9 in Col. 56.
335	T	DECIMAL POSITIONS (COLUMN 56) GREATER THAN NUMBER OF DIGITS - ASSUME EQUAL.	You specified more positions to the right of the decimal in the Decimal Positions Field (Col. 56) than there are digits.	The compiler ignores your entry and assumes the number of decimal positions equals the number of digits. Change either the number of decimal positions or the field length.
336	W	SEQUENCE (COLUMN 57) NOT BLANK, A, OR D - A ASSUMED.	You entered a letter or digit, other than an A or D, in the Table/Array Sequence Field (Column 57).	The compiler ignores your entry and assumes you entered an A (for ascending sequence).
337	T	FIELD NAME (COLUMNS 46-51) BLANK OR INVALID - ENTRY IGNORED.	You entered an invalid Table, Array, or Routine Name or left it blank in the Table, Array or Routine Field (Col. 46-51).	Enter the correct name and re-compile.
338	T	TABLE/ARRAY NAME (COLUMNS 46-51) PREVIOUSLY DEFINED - RECORD IGNORED.	You previously used this Table or Array name in the Table/Array Field (Col. 46-51).	The compiler ignores the record entry; change the name and re-compile.
339	W	NO EXTENSION SPECIFICATION FOR FILE WITH E IN COLUMN 39 OF FILE SPECIFICATION.	Your program uses a table, array, chaining file, or RAF, but you omitted the File Extension Specification needed to describe it.	Insert the Extension Specification and re-compile the program.

Table B-4. File Extension (E) Specification Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
340	W	ENTRIES PER RECORD BLANK AND FILENAME SPECIFIED - ASSUME 1.	You didn't specify the number of entries in one record, yet you specified the filename in the Table, Array, or Routine Name Field (Col. 27-32).	The compiler ignores your entry and assumes you specified 1 entry per record. Enter valid numeric characters, right-adjusted, with no embedded blanks.
341	W	EXTENSION SPECIFICATION BLANK - IGNORE RECORD.	You didn't enter filenames in the From Filename Field (Col. 11-18) and the To Filename Field (Col. 19-26).	The compiler ignores the record; specify a From Filename and To Filename.
342	W	ENTRIES PER RECORD GREATER THAN TABLE SIZE - ASSUME ENTRIES REVERSED.	You specified a greater number of entries in one record of the table (Entries Per Record Field, Col. 33-35) than the number of entries in the entire table (Entries Per Table/Array Field, Col. 36-39).	The compiler ignores your entry and assumes the entries are reversed (the table size is greater than the entries per record). Enter the correct number of entries per record and table size, and re-compile.
343	T	FROM FILE RECORD LENGTH NOT LARGE ENOUGH TO HOLD ENTRIES PER RECORD.	You specified a number of entries per record (Col. 33-35) that exceeds the space allotted by the length of the records in the From File.	Re-specify the number of entries per record, and re-compile.
344	T	TO FILE RECORD LENGTH NOT LARGE ENOUGH TO HOLD ENTRIES PER RECORD.	You specified a number of entries per record (Col. 33-35) that exceeds the space allotted by the length of the records in the To File.	Re-specify the number of entries per record, and re-compile.
345	T	TABLE FILE CANNOT BE A SPECIAL FILE.	You specified that a table resides on a SPECIAL file.	Specify a file of a type other than SPECIAL for the table.
346	T	RECORD LENGTH FOR PRE-EXECUTION TIME TABLE/ARRAY FILE CANNOT BE LESS THAN 3.	You specified an incorrect entry for Columns 24-27 of F spec.	Correct entry and re-compile your program.

Table B-5. Line Counter (L) Specification Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
401	W	INVALID, MISSING, OR UNDEFINED FILENAME IN COL. 7-14, SPEC IS DROPPED.	You specified a filename in the Filename Field (Col. 7-14) that was invalid or undefined, or you omitted this filename completely.	Include the correct filename and re-compile.
402	W	FILENAME IN COL. 7-14 DOES NOT REFER TO 'L' EXTENSION CODE.	You specified a filename in the Filename Field (Col. 7-14) that is not defined in the File Description Specifications, or you specified it in the File Description but failed to specify L in Col. 39.	Define the filename properly, and re-compile.
403	W	FORM LENGTH ENTRY IN COL. 15-17 INVALID OR GREATER THAN 112.	You specified an invalid Line number or a number greater than 112 in the Line Number Field (Col. 15-17).	The compiler ignores your entry and assumes Line Number 66.
404	W	INVALID OR MISSING ENTRY IN COL. 18-19, ASSUME FL.	You entered characters other than OL or FL, or the digits 01 through 12 in the channel Number/OL/FL Field (Col. 18-19), or left this field blank.	The compiler ignores your entry and assumes you entered FL (for Form Length).
405	W	OVERFLOW LINE ENTRY IN COL. 20-22 INVALID OR GREATER THAN 112.	You entered an invalid overflow line or specified a line number greater than 112 in the Channel Number /OL/FL Field (Col. 20-24).	The compiler assumes Line Number 66. Correct the entry and re-compile.
406	W	INVALID OR MISSING OL ENTRY IN COL. 23-24, ASSUME OL.	You entered characters other than OL or FL, or the digits 01 through 12 in the Channel Number/OL/FL Field (Col. 23-24), or left this field blank.	The compiler assumes OL. Correct the entry and re-compile.
407	W	OVERFLOW LINE IN COL. 20-22 EXCEEDS FORM LENGTH IN COL. 15-17, ASSUME FORM LENGTH.	You specified an overflow line in the Line Number Field (Col. 20-22) that exceeds the form length in the previous Line Number Field (Col. 15-17).	The compiler assumes that the form length line and overflow line are equal.

Table B-5. Line Counter (L) Specification Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
408	W	LINE NUMBER ENTRY IS INVALID, DROP LINE AND CHANNEL ENTRY.	You specified an erroneous line number.	The compiler ignores your entry.
409	W	CHANNEL NUMBER ENTRY IS INVALID, DROP LINE AND CHANNEL ENTRY.	You specified an erroneous channel number.	The compiler ignores your entry.
410	W	CHANNEL 01 AND/OR CHANNEL 12 IS NOT DEFINED.	You didn't specify channel 01 and/or channel 12.	The compiler assumes channel 1 indicates Line 6 and channel 12 (overflow line) indicates Line 60.
411	W	MULTIPLE LINE COUNTER SPECIFICATION FOR SAME FILE IS DROPPED.	You included more than one line counter specification for the same channel, form length, or overflow line.	The compiler ignores the second line counter specification.
412	W	COLUMNS 25-74 SHOULD BE BLANK.	You entered data in Columns 25-74 when using the Line Number Option of the Line Counter Specifications.	The compiler ignores the entries in Col. 25-74, and assumes you left these columns blank.
413	W	NO LINE SPECIFICATION FOR FILE WITH L IN COLUMN 39 OF FILE SPECIFICATION.	You specified L in Column 39 of F spec with no associated 'L' specification.	Correct entry and re-compile your program.

Table B-6. Input Specification (I) Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
501	T	INVALID OR UNDEFINED FILE-NAME - DISC FILE ASSUMED.	You entered, in the Filename Field (Col. 7-14), an improper filename or a filename that was not defined in the File Description Specifications.	Specify the correct filename or include a File Description Specification record for the file.
502	T	FILE IS NOT AN INPUT, UPDATE, OR COMBINED FILE - DISC INPUT ASSUMED.	You entered a filename that does not identify an input, update, or combined file.	Correct the File Description Record to indicate an input, update, or combined file.
503	T	AND/OR LINE CANNOT FOLLOW TRAILER OR LOOK-AHEAD.	You specified an AND/OR Line in Col. 14-16 after specifying a Trailer or Look-Ahead record in the Record Indicator/Look-Ahead/Trailer Field (Col. 19-20).	Remove the AND/OR Line.
504	T	AND/OR LINE DOES NOT FOLLOW RECORD IDENTIFICATION LINE.	You did not specify a record to which this AND/OR Line applies.	Enter the first line of the record identification for this record.
505	W	AND/OR LINE FOLLOWS LINE WITHOUT RECORD IDENTIFICATION CODES.	You did not include record identification codes in Cols. 21-41 of the previous line.	Enter the record identification codes (C, Z, D) in the previous line.
506	W	AND/OR LINE AND NUMBER/OPTION (COLUMNS 17-18) NOT BLANK.	You entered an AND/OR Line in Col. 14-16 and on the same specification line you entered a Number or Option in the Number of Records Field (Col. 17) or Option Field (Col. 18).	Enter the number of records or option field on a separate specification line from the AND/OR line, then re-compile.
507	W	AND LINE AND RECORD INDICATOR (COLUMNS 19-20) NOT BLANK.	You entered an AND line in Col. 14-16 and on the same specification line you entered a record indicator in the Record Indicator/Look-Ahead/Trailer Field (Col. 19-20).	Enter a record indicator on a separate specification line from the AND line, then re-compile.

I Input Specifications Errors

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
508	W	AND LINE AND STACKER SELECT (COL. 42) NOT BLANK.	You entered an AND Line in Cols. 14-16, and on the same line, you entered a stacker number in Col. 42.	Remove the stacker number from the AND Line and place it on the first record identification line of the AND group.
509	T	TRAILER HEADER DOES NOT CONTAIN ALPHA SEQUENCE OR NUMERIC WITH N.	You specified a trailer record and the previous record identification specification contained a numeric sequence entry, but did not contain N in Col. 17.	On the previous record identification specification, enter N in Col. 17 or specify an alphabetic sequence entry.
510	W	COLS. 7-14, 17, 18, 21-70 FOR LOOK-AHEAD OR COLS. 7-18, 21-70 FOR TR MUST BE BLANK.	You entered data in improper fields in the description of a spread or look-ahead record, or entered the file name and/or group sequence in a spread record description.	Remove the data from all columns that should be blank.
511	T	PREVIOUS LOOK-AHEAD FOR FILE.	You defined more than one Look Ahead record for a file, but only one is allowed.	Remove one of the look-ahead records (consolidate them).
512	W	LOOK AHEAD RECORD HAS NON-ALPHA SEQUENCE NUMBER (COLUMNS 15-16).	You specified a numeric sequence for a look ahead record.	Change the sequence characters (in Cols. 15-16) to alphabetic characters.
513	W	ALPHA SEQUENCE (COLUMNS 15-16) FOLLOWS NUMERIC — ASSUME NUMERIC.	You listed the numeric sequence entries before the alphabetic entries in the Group Sequence Field (Col. 15-16).	The compiler ignores your sequence entry and assumes a numeric sequence. Move the alphabetic sequence specifications so that they come before the numeric sequence specifications.
514	W	OPTION/NUMBER FIELDS (COLUMNS 17-18) NOT BLANK FOR ALPHA SEQUENCE — ASSUME BLANK.	You entered an option in the Option Field (Col. 17) or indicated number of records in the Number of Records Field (Col. 18) and also specified an alphabetic Sequence in the Group Sequence Field (Col. 15-16).	The compiler ignores your entry and assumes you left the Number of Records Field blank. Either change the alphabetic sequence characters (Cols. 25-26) to numeric characters, or set Cols. 17-18 to blank.

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
515	W	INVALID SEQUENCE NUMBER (COLUMNS 15-16) - ASSUME ALPHA SEQUENCE.	You entered an invalid sequence number in the Group Sequence Field (Col. 15-16), and the previous sequence characters were alphabetic.	The compiler ignores your sequence number and assumes you entered an alphabetic sequence. Correct the sequence number.
516	W	INVALID SEQUENCE NUMBER (COLUMNS 15-16) - ASSUME NUMERIC SEQUENCE.	You entered an invalid sequence number in the Group Sequence Field (Col. 15-16), and the previous sequence characters were numeric.	The compiler ignores your sequence number and assumes you specified a numeric sequence. Correct the sequence number.
517	T	SEQUENCE (COLUMNS 15-16) NOT ASCENDING - ASSUME ASCENDING SEQUENCE.	Your numeric sequence character group was not higher than the previous sequence number.	The compiler ignores your sequence and assumes ascending sequence. Correct the numeric sequence character group to fall in proper ascending sequence.
518	W	FIRST NUMERIC SEQUENCE NOT 1 - ASSUME 1.	You didn't assign the first sequence as 01 in the Group Sequence Number Field (Col. 15-16).	The compiler ignores your sequence number and assumes you entered 01. Correct the first numeric sequence character group.
519	T	NUMBER FIELD (COLUMN 17) NOT 1 OR N WITH NUMERIC SEQUENCE - ASSUME N.	You entered a character other than an N or 1, in the Number of Records Field (Col. 17) and specified numeric sequencing.	The compiler ignores your entry and assumes you entered an N. Correct the Number of Records entry.
520	W	OPTION FIELD (COLUMN 18) NOT BLANK OR O - ASSUME O.	You entered a character, other than an O, in the Option Field (Col. 18).	The compiler ignores your entry and assumes you entered an O. Correct the Option Field.
521	T	INVALID RECORD IDENTIFICATION INDICATOR.	You used an improper indicator in the Record Indicator Field (Col. 19-20).	Enter in Cols. 19-20, one of the indicators 01-99, H0-H9, L0-L9, LR, MR, OA-OG, OV, 1P.

I Input Specifications Errors

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
522	W	STACKER SELECT (COLUMN 42) NOT BLANK, 1 OR 2 - ASSUME 1.	You entered an improper character in the Stacker Select Field (Col. 42).	The compiler assumes you entered 1. Correct the Stacker Select Field.
523	T	INVALID RECORD IDENTIFICATION CODE POSITION FIELD (COLUMNS 21-24, 28-31, OR 35-38) - ASSUME 1.	You specified invalid numbers or embedded blanks in the Record Identification Code position Field (Col. 21-24, 28-31, or 35-38).	Correct the field to include valid numeric characters, right-adjusted.
524	T	RECORD ID CODE POSITION (COLUMNS 21-24, 28-31, OR 35-38) NOT WITHIN RECORD LENGTH - ASSUME 1.	You specified a record identification code in the Record Identification Code Position Field (Col. 21-24, 28-31, or 35-38) that doesn't fall within the record length defined.	The compiler ignores your entry and assumes you entered a 1 in the Position Field. Correct the Position Field so that it falls within the record length.
525	W	NOT FIELD (COLUMNS 25, 32, OR 39) NOT BLANK OR N - ASSUME N.	You specified a character other than N in the Not Field.	Correct the Not Field to contain Nor blank.
526	W	C/Z/D FIELD (COLUMNS 26, 33, OR 40) NOT C, Z OR D - ASSUME C.	You entered a letter or digit, other than a C, Z, or D, in the C/Z/D Field (Col. 26, 33, or 40).	The compiler ignores your entry and assumes you entered a C. Change the entry to C, Z, or D.
527	W	NO FIELD DESCRIPTIONS FOR PREVIOUS RECORD.	You followed one record description specification with another record description specification, without one or more intervening field description specifications.	This entry is probably valid. However, the compiler issues a warning just in case this may be an AND/OR line or in case you forgot to include field descriptions for the previous record.
528	W	COLUMNS 43-70 NON-BLANK FOR RECORD DESCRIPTION.	You entered data in Columns 43-70 in a record description specification.	The compiler ignores the data in Cols. 43-70. Remove the data from these columns.

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
529	T	COLUMN 43 NOT P, B, L, R, 1-9, OR BLANK - ASSUME BLANK.	You entered a character other than a P, B, L, R, or 1-9, in the Data Format Field (Col. 43).	The compiler ignores your entry and assumes you left the field blank. Enter P, B, L, R, 1-9, or blank in Col. 43.
530	T	FROM FIELD LOCATION INVALID - ASSUME 1.	You specified invalid numeric characters or embedded blanks in the From Field Position (Col. 44-47).	The compiler ignores your entry and assumes you entered a 1. Correct the From Field Location to valid numbers, right-adjusted.
531	T	TO FIELD LOCATION INVALID - ASSUME 1.	You specified invalid numeric characters or embedded blanks in the To Field Position (Col. 48-51).	The compiler ignores your entry and assumes you entered a 1. Correct the To Field Location to valid numbers, right-adjusted.
532	T	'FROM' GREATER THAN 'TO' FIELD LOCATION - ASSUME FIELD LENGTH OF 1.	You specified that the From Field in the From Field Position (Col. 44-47) is greater than the To Field in the To Field Position Field (Col. 48-51).	The compiler ignores your entry and assumes the From Field length is 1. Correct the From and/or To Field Locations so that the To Field Location is greater than or equal to the From Field Location.
533	T	FIELD LOCATION NOT WITHIN RECORD LENGTH.	You specified a To Field Location that does not fall within the record length defined for the field.	Either correct the record length for the file, or change the field location so that it falls within the record length.
534	T	DECIMAL POSITIONS (COLUMN 52) NOT BLANK OR 0-9 - ASSUME BLANK.	You entered a character, other than 0-9, in the Decimal Positions Field (Col. 52).	The compiler ignores your entry and assumes you left the field blank. Correct the field to reflect the number of decimal positions desired.
535	T	FIELDNAME (COLUMNS 53-58) BLANK OR INVALID - SPECIFICATION DROPPED.	You entered an invalid field name or left the field blank in the Field Name Field (Col. 53-58).	The compiler ignores this specification. Correct the field name.

I Input Specifications Errors

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
536	T	INVALID INDEX FOR FIELD NAME (COLUMNS 53-58) ASSUME INDEX OF 1.	You entered invalid numeric characters or embedded blanks for index of array named in the Field Name Field (Col. 53-58).	The compiler ignores your index and assumes you entered a 1. Correct the index to a proper numeric character or variable.
537	T	VARIABLE ARRAY INDEX (COLUMNS 53-58) IS NOT NUMERIC VARIABLE.	You specified a value other than a numeric variable for the variable array index in the Field Name Field (Col. 53-58).	Specify a numeric variable with zero decimal positions as the array index.
538	W	VARIABLE ARRAY INDEX (COLUMNS 53-58) DOES NOT HAVE ZERO DECIMAL POSITIONS - ACCEPTED AS IF IT DID.	You specified a variable array index without specifying zero decimal positions for it when you defined it.	The compiler accepts your entry as if you specified zero decimal positions for the index. Re-define the index to have zero decimal positions.
539	T	INDEX CANNOT BE AN ARRAY NAME.	You used an array name as an array index.	Use either a numeric constant or a numeric variable with zero decimal positions as the array index.
540	T	FIELD NAME (COLUMNS 53-58) IS A NON-ALTERABLE FIELD - RECORD IGNORED.	The field named is a previously-defined look-ahead field, or UDAY, UMONTH, UYEAR, or UDATE. These cannot be used on Input Specifications.	Specify a different field name.
541	T	PAGE INVALIDLY REDEFINED - MUST BE NUMERIC WITH ZERO DECIMAL POSITIONS.	You re-defined the PAGE Field (Field Name Field, Col. 53-58) without entering zero in the Decimal Positions Field (Col. 52).	Enter a zero in the Decimal Positions Field (Col. 52) for the field name PAGE, and re-compile.

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
542	T	DECIMAL POSITIONS (COLUMN 52) DIFFERS FROM PREVIOUS SPECIFICATION - ASSUME PREVIOUS DEFINITION.	You entered the same field name that was previously defined with different decimal positions in the field (Col. 52).	The compiler ignores your entry and assumes the decimal position is the same as in the previous definition of this field name. Correct one or the other entry so that both are the same.
543	T	FIELD SIZE IS NOT A MULTIPLE OF THE NUMBER OF ELEMENTS - NEXT LOWEST MULTIPLE ASSUMED.	You specified an array name, and the field size allows for several array elements plus a fraction of an array element.	Correct the field size to contain an integral number of array elements.
544	T	ARRAY SIZE EXCEEDED - ASSUME TO END OF ARRAY.	You specified an array name and the field size allows for more array elements than the array was defined to hold.	Correct either the array definition or the number of array elements defined by the field length.
545	T	FIELD SIZE NOT SAME AS PREVIOUSLY DEFINED - ASSUME PREVIOUS DEFINITION.	You specified a field size that wasn't the same as previously defined for this field.	The compiler ignores your entry and assumes the field size is the same as the one previously defined for this field. Correct the size of one or the other fields so that both are the same size.
546	T	FIELD PREVIOUSLY DEFINED AS ALPHA - ASSUME ALPHA.	You defined a field as numeric, but this field was previously defined as alphanumeric.	The compiler assumes that this is an alphanumeric field. Correct the type of one or the other so that both fields are the same type.
547	T	FIELD PREVIOUSLY DEFINED AS NUMERIC - ASSUME NUMERIC.	You defined a field as alphanumeric, but this field was previously-defined as numeric.	Correct the definition of one field or the other so that both fields are the same type.

I Input Specifications Errors

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
548	T	ALPHA FIELD SIZE GREATER THAN 256 - ASSUME 256.	You entered an alphanumeric field size greater than 256 characters; the To Field Location minus the From Field Location plus 1 is greater than 256.	The compiler ignores your entry and assumes you specified 256 characters. Correct the From Field and/or To Field Location to reflect a field length of 256 or less.
549	T	DECIMAL FIELD LENGTH EXCEEDS 15 DIGITS - ASSUME 15.	You entered a decimal field length that exceeds 15 digits.	The compiler ignores your entry and assumes you specified 15 digits. Correct the From Field and/or To Field Location to reflect a digit length of 15 or less.
550	T	DECIMAL POSITIONS EXCEED NUMBER OR DIGITS - ASSUME EQUAL TO DIGITS.	You specified, in the Decimal Positions Field (Cols. 52), a value greater than the total number of digits in the field.	The compiler ignores your entry and assumes the decimal positions are equal to the number of digits. Correct the number of decimal positions or the digit size of the field.
551	T	FIELD RECORD RELATION (COLUMNS 63-64) INVALID INDICATOR - BLANK ASSUMED.	You specified an invalid record-identifying indicator in the Field Record Relation Field (Col. 63-64).	The compiler ignores your entry and assumes you left this field blank. Correct the field record relation entry to reflect a valid indicator (01-99, H0-H9, L0-L9, LR, MR, OA-OG, OV, IP).
552	T	CONTROL LEVEL (COLUMNS 59-60) NOT BLANK OR L1-L9 - ASSUME BLANK.	You specified invalid characters, other than the control level indicators L1-L9, in the Control Level Field (Col. 59-60).	The compiler ignores your entry and assumes you left this field blank. Correct the Control Level Field to contain L1-L9 or blank.

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
553	T	CONTROL LEVEL LENGTH DIFFERS FROM PREVIOUS DEFINITION - ASSUME PREVIOUS DEFINITION.	You specified a different control level field length than the length previously defined for this field.	The compiler ignores your entry and assumes you specified the same control level field length as you previously defined. Correct one length or the other so that both are the same.
554	T	CONTROL LEVEL (COL. 59-60) OR MATCHING FIELD (COL. 61-62) USED W/CHAINED OR DEMAND FILE - ASSUME BLANK.	You specified a control level indicator in the Control Level Field (Col. 59-60) or a matching-record indicator in the Matching/Chaining Field (Col. 61-62) with a chained or demand file.	The compiler ignores your entry and assumes you omitted the control level indicator or matching record indicator. Change the file designation or leave Cols. 59-62 blank in the Input Specification.
555	T	CONTROL OR MATCHING FIELD (COLUMNS 59-62) USED FOR TRAILER OR LOOKAHEAD RECORD - ASSUME BLANK.	You assigned a control level or matching field indicator to a spread record trailer or a look-ahead record.	The compiler ignores your entry and assumes that you left this field blank. Change these columns to blank.
556	T	INVALID MATCHING OR CHAINING ENTRY - ASSUME BLANK.	You assigned a code other than M1 through M9, C1 through C9, or blank to the Matching/Chaining Code Field (Columns 61-62).	The compiler ignores your entry and assumes that you left this field blank. Change the entry to M1-M9, C1-C9, or blank.
557	T	ARRAY USED FOR MATCHING FIELD - ASSUME NO MATCHING FIELD.	You used an array rather than a data field to test for matching fields.	The compiler assumes that no match occurred. Remove the matching field entry for this record. If this field must be used as a matching field, re-define it with another name and assign M1-M9 to the new field name.

I Input Specifications Errors

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
558	T	FIELD RECORD RELATION PREVIOUSLY USED WITH THIS MATCHING OR CONTROL LEVEL - ASSUME PREVIOUS.	You used the same field record relation previously in this record with matching or control level fields.	Either remove these matching/control level indicators, change the field record relation, or move the specification records to a new place.
559	T	MATCHING LEVEL ALREADY SPECIFIED FOR THIS FIELD RECORD RELATION.	The same matching level (M1-M9) has already been used in this record specification with the same field record relation (may be a blank).	Change the matching level or the field record relation.
560	T	MATCHING LEVEL LENGTH DIFFERS FROM PREVIOUS DEFINITION - ASSUME PREVIOUS.	The matching field length specified differs from the length previously defined for this field.	The compiler assumes the previous length for this field. Change one field length or the other so that both are the same.
561	W	COMPLETE SET OF MATCHING FIELDS NOT DEFINED FOR LAST GROUP.	When you previously specified matching fields in this program, you specified more levels of matching fields than the current definition.	Either delete the extra matching levels specified earlier or add levels to the current definition.
562	T	MATCHING OR CHAINING LEVEL (COLUMNS 61-62) NOT VALID - ASSUME BLANK.	You entered characters other than C1-C9, M1-M9, or blank, in the Matching/Chaining Code Field (Columns 61-62).	The compiler ignores entry and assumes you entered blanks. Correct the entry in Cols. 61-62 to be C1-C9, M1-M9, or blanks.
563	T	CHAINING (COLUMNS 61-62) SPECIFIED WITH LOOK-AHEAD FIELD - CHAINING IGNORED.	You requested chaining with a look-ahead record.	The compiler ignores your entry and assumes a request for look-ahead. Delete the chaining request from the look-ahead field. Include it in a regular field definition if it is needed.

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
564	T	L OR R (COLUMN 43) SPECIFIED WITH CHAINING (COLUMNS 61-62) - CHAINING IGNORED.	You specified a field containing unpacked decimal data with leading or trailing sign for a chaining field, but such a field cannot be used for this purpose.	The compiler ignores your request for chaining. Remove the chaining specification or the L or R. The field may be re-defined as a regular field (with From Location one larger for L or To Location one smaller for R) with no L or R for the field.
565	W	ARRAY SPECIFIED AS CHAINING FIELD - IGNORE CHAINING.	You specified an array rather than a data field as a chaining field.	The compiler ignores your request for chaining. Remove the chaining entry from this record. The field may be re-defined as a regular field (same location but different name) with chaining
566	T	SPLIT CHAINING USED WITH NO CONVERSION ROUTINE.	You did not supply a conversion routine to process a split chaining field for a relative file.	If you wish to use split chaining, supply a conversion routine.
567	T	CHAINING LEVEL (COLUMNS 61-62) ALREADY SPECIFIED FOR THIS FIELD RECORD RELATION.	The same chaining level has already been used in this record specification with the same field record relation.	Delete the chaining level (C1-C9) from this record.
568	T	FIELD RECORD RELATION PREVIOUSLY USED IN THIS RECORD FOR CHAINING.	All chaining fields for one field record relation were not together.	Move the record so that it is among the others of the same field record relation or delete the chaining entry (C1-C9) from this record.
569	W	INVALID PLUS RESULT INDICATOR (COLUMNS 65-66) - ASSUME BLANK.	You entered an improper indicator or illegal characters in the High subfield (Col. 65-66).	The compiler assumes that you left this subfield blank. Enter blanks or a valid indicator (01-99, H0-H9, L0-L9, LR, MR, OA-OG, OV, 1P) in these columns (65-66).

I Input Specifications Errors

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
570	W	INVALID MINUS RESULT INDICATOR (COLUMNS 67-68) - ASSUME BLANK.	You entered an improper indicator or illegal characters in the Low subfield (Col. 67-68).	The compiler assumes that you left this subfield blank. Enter blanks or a valid indicator (01-99, H0-H9, L0-L9, LR, MR, OA-OG, OV, 1P) in these columns (67-68).
571	W	INVALID BLANK/ZERO RESULT INDICATOR (COLUMNS 69-70) ASSUME BLANK.	You entered an improper indicator or illegal characters in the Equal subfield (Col. 69-70).	The compiler assumes that you left this subfield blank. Enter blanks or a valid indicator (01-91, H0-H9, L0-L9, LR, MR, OA-OG, OV, 1P) in these columns 69-70).
572	W	FIELD USED PREVIOUSLY AS INDEX NOT DEFINED AS NUMERIC WITH ZERO DECIMAL POSITIONS.	You used an alphanumeric field or a numeric field containing one or more decimal positions as an array element index.	Re-define the index field to be numeric with zero decimal positions.
573	T	TABLE USED AS INPUT FIELD.	You specified a table name in the Field Name Field (Col. 53-58).	Delete the reference to the table from the Input Specification.
574	T	MATCHING OR CONTROL LEVEL WITH BLANK FIELD RELATION MUST BE SPECIFIED FIRST.	You specified matching or control-level fields with a field record relation indicator before the current one (which has a blank field record relation).	Move the blank field record relation specification so that it comes before those that specify an indicator.
575	T	FIRST INPUT SPECIFICATION DOES NOT HAVE FILE NAME.	You omitted the file name from the Filename Field (Cols. 7-14) of your first Input Specification Record, or entered a field description with no preceding record description.	Include a file name with the first input specification record.



Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
576	T	FILENAME ALREADY USED FOR INPUT SPECIFICATIONS.	You specified a file name that was previously used in a record description in the Input Specifications and included an intervening specification for a different file.	Keep all Input specifications for a file together, without intervening specifications for another file.
577	W	NUMERIC SEQUENCE (COLUMNS 15-16) SPECIFIED FOR CHAINED FILE - ASSUME BLANK.	You specified a numeric group sequence for this chained file, in the Group Sequence Field, where an alphanumeric sequence is required.	The compiler assumes that you specified an alphanumeric sequence. Change the sequence number to alphanumeric characters.
578	W	BLANK RECORD - RECORD IGNORED.	You submitted a blank record within your Input Specification Records.	The compiler ignores this record. Remove this record.
579	T	LOOK-AHEAD OR TRAILER RECORD NOT IN A PRIMARY OR SECONDARY FILE.	You included a look-ahead record or spread record trailer in a file other than a primary or secondary file.	Either change the file designation or remove the look-ahead or trailer records.
580	T	P, B, L, OR R (COLUMN 43) SPECIFIED FOR ALPHA FIELD - ASSUME BLANK.	You specified in the Data Format Field, a numeric format for a field previously defined as alphanumeric.	The compiler assumes that you left the Data Format field blank in this specification. Either change Col. 43 to blank or define this as a numeric field (by entering a digit in the Decimal Positions Field).
581	T	BINARY FIELD LENGTH IS NOT 2 OR 4 - ASSUME UNPACKED NUMERIC FIELD.	You defined an input field length other than 2 or 4 for a binary field; (2 is one word and 4 is two words).	Either correct the field length or change the type of numeric field.
582	T	INDEX SPECIFIED FOR NON-ARRAY - IGNORE INDEX.	You specified an index for a data field that is not an array.	The compiler ignores the index. Either include an array definition (E specification) for this field name, or delete the index.

I Input Specifications Errors

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
583	T	MATCHING RECORD FILE NOT IN SAME SEQUENCE AS LAST.	A file containing matching records is in a different sequence than the last matching record file previously specified.	Change the file specification records (Col. 18) to be all A or all D for files with matching fields.
584	T	INPUT, UPDATE, OR COMBINED FILE HAS NO INPUT SPECIFICATIONS.	You named an input, update, or combined file but failed to specify the required record characteristics.	Include the specifications for the input, update, or combined file or change the file type.
585	T	DIGIT LENGTH OF FIELD IS ZERO - ASSUME 1.	You have specified a numeric field containing no digits, but such fields must contain at least one digit.	The compiler assumes that you specified a one-digit field. Re-define the field to be at least one digit long; (fields with L or R in Col 43 must be at least 2 bytes long to have 1 digit).
586	T	CHAINING CODE AND FILE NOT SPECIFIED ON EXTENSION SPECS - IGNORE CHAINING.	You failed to specify a chaining field code (in Cols. 9-10) and/or chaining file name (in Cols. 11-18) of the File Extension Specifications, even though you have defined such a file in the Input Specifications.	Either correct the File Extension Specifications or delete the chaining entries from the Input Specifications.
587	W	ARRAY USED FOR LEVEL FIELD - ASSUME NOT A LEVEL FIELD.	You specified an array name in the Control Level Field (Cols. 59-60), but a field name is required.	The compiler assumes that you left the Control Level Field blank. Change the Control Level Field to blanks. If this field must be a control field, re-define it with a new field name and include the chaining level.
588	T	LOOK-AHEAD FIELD PREVIOUSLY DEFINED - RECORD IGNORED.	You defined a look-ahead field that has already been defined.	The compiler ignores the record. Re-name this field or delete the record.

Table B-6. Input Specification (I) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
589	W	CHAINING FIELD NUMERIC BUT ALPHA KEYS SPECIFIED ON FILE SPEC.	You specified a numeric chaining field in conflict with alphanumeric searching keys on the IMAGE File.	Either change the field type or the record address type (Col. 31) of the File Description Specification.
590	W	CHAINING FIELD LENGTH NOT SAME AS KEY FIELD LENGTH FROM FILE SPEC.	You specified different lengths for the Chaining and key fields; these must be the same.	Either re-define the chaining field length, the key length, or include a conversion routine.
591	W	THE PRECEDING CONTROL LEVEL LENGTH DIFFERS FROM ITS PREVIOUS DEFINITION.	You specified a control field for which the total length differs from the total length of the same control level defined for a previous record.	Change your program to specify the same total length for all definitions of the same control level, and re-compile.
592	W	PRECEDING CHAINING FIELD LENGTH NOT SAME AS KEY FIELD LENGTH FROM FILE SPEC.	You specified a chaining field for which the total length differs from the Key Field Length in File Description spec Columns 29 through 30.	Change your program to correct the length, and re-compile.
593	T	DATA STRUCTURE (DS) SPECIFICATIONS CANNOT BE FOLLOWED BY NON-DATA-STRUCTURE SPECS.	You placed some non-data structure input specification entries following a Data Structure.	Rearrange input specifications to place all data structures at end, and re-compile.
594	W	RESERVED NAME FOR UDS IS 'LDA' - NAME SPECIFIED IN COLS. 7-12 IS REPLACED BY 'LDA' .	You specified UDS (User Data Structure) in Columns 18 through 20 and either: (1) specified a name other than LDA or blank in Columns 7 through 12, or (2) specified LDA in Columns 7 through 12, but had already defined LDA as a field on a prior Input Specification.	Correct your program and re-compile.

Table B-7. Calculation Specification (C) Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
601	T	INVALID CONTROL LEVEL IN COL. 7-8, ASSUME LATEST LEVEL.	You entered an indicator other than L0-L9, or LR in the Control Level Field (Cols. 7-8).	The compiler assumes that you intended the latest level indicator. Correct the control level and re-compile.
602	W	AN/OR LINE OUT OF ORDER OR INVALID, IGNORE AN/OR LINE.	You entered an AN and/or OR Line, and the preceding line has an entry in the Operation Field or the preceding line did not have an entry in the Indicator Field.	The compiler ignores this line. Put AN/OR Line in proper sequence.
603	T	MORE THAN 7 AN/OR LINES. IGNORE ENTIRE INDICATOR GROUP.	You entered a total of more than seven AN and/or OR lines in one grouping, which is not permitted.	The compiler ignores the entire grouping. Change program so that there are no more than seven AN/OR Lines.
604	W	'AN' OR 'OR' NOT SPECIFIED IN COL. 7-8, IGNORE GROUP.	The statement before this AN/OR record had conditioning indicators, but no operation was specified.	The compiler ignores the entire group. Put operator in previous statement or include AN/OR on this record.
605	T	OPERATION NOT RECOGNIZED.	You entered an illegal or undefined operation code in the Operation Field (Cols. 28-32).	The compiler ignores this operation request. Correct the operation code and re-compile.
606	W	'NOT' ENTRY IN COL. 9, 12 OR 15 NOT N OR BLANK, ASSUME N.	You entered a character other than N or blank in the Not Subfield (Col. 9, 12, or 15).	The compiler assumes that you entered N.
607	W	INVALID OPERATION INDICATOR, IGNORE INDICATOR GROUP.	You entered an invalid operation indicator in the Indicators Field (Cols. 9-17).	The compiler ignores the indicators and performs the operation in any event. Correct the indicator and re-compile.
608	W	INDICATOR MISSING FOR 'NOT' IN COL. 9, 12 OR 15, IGNORE NOT.	You entered an N (for Not) in Columns 9, 12, or 15, but failed to specify an operation indicator in the following column.	The compiler ignores the N entry.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
609	W	INVALID RESULT- ING INDICATOR, IGNORE INDICATOR GROUP.	You entered an improper in- dicator in the Resulting Indicators Field (Cols. 54-59).	The compiler ignores this indicator and all others in the group to which it belongs. Correct the in- dicator and re-compile.
610	W	INVALID RESULT FIELD ENTRY IN COL. 43-48.	You entered an improper name in the Result Field (Cols. 43- 48).	The compiler ignores the operation associated with this entry. Correct the Result Field name so that it begins with a letter (A-Z) or #, \$, or @ followed by alphanumeric characters or #, \$, or @.
611	W	INVALID FACTOR 1 ENTRY IN COL. 18-27.	You entered an improper name, label, or literal in the Factor 1 Field (Cols. 18-27).	The compiler ignores this operation. Correct Factor 1 so that it reflects a proper field name (see Error Message No. 610) or a literal.
612	T	INVALID FACTOR 2 ENTRY IN COL. 33-42.	You enter an improper name, label, or literal in the Factor 2 Field (Cols. 33-42).	The compiler ignores this operation. Correct Factor 2 so that it reflects a proper field name (see Error Message No. 610) or a literal.
613	W	INVALID INDEX IN RESULT FIELD (COL. 43-48).	You specified an indexed array in the Result Field, but failed to use a proper index.	The compiler assumes the index value to be 1. Correct the index so that it is a valid field name (see Error Message No. 610) that re- flects a numeric field with zero decimal positions, or a numeric unsigned literal with no decimal point.
614	W	INVALID INDEX IN FACTOR 1 (COL. 18-27).	You specified an indexed array in the Factor 1 Field, but failed to use a proper index.	The compiler assumes the index value to be 1. Correct the error as in Error Message No. 613.
615	W	INVALID INDEX IN FACTOR 2 (COL. 33-42).	You specified an indexed array in the Factor 2 Field, but failed to use a proper index.	The compiler assumes the index value to be 1. Correct the error as in Error Message No. 613.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
616	T	INVALID FILE NAME ENTRY FOR FACTOR 2 (COL. 33-42).	You entered an improper file name in the Factor 2 Field.	The compiler ignores the operation. Correct the file name so that it appears as defined in the File Description Specifications.
617	W	INVALID RESULT FIELD LENGTH ENTRY IN COL. 49-51.	You entered a value other than 1 through 256 in the Field Length Field.	The compiler ignores the field length entry. Enter valid numeric characters for the Result Field length, right-adjusted, without embedded blanks.
618	W	INVALID DECIMAL POSITIONS ENTRY IN COL. 52.	You entered a value other than 0 through 9 in the Decimal Positions Field.	The compiler assumes zero decimal positions. Enter a digit 0-9 in Col. 52.
619	W	RESULT FIELD LENGTH MISSING BUT DEC POS SPECIFIED. IGNORE DEC POS.	You specified the appropriate number of decimal positions for the Result Field, but neglected to specify the field length.	The compiler ignores the decimal positions entry, as if you specified no decimal positions. Enter a valid field length, or leave the Decimal Positions Field blank.
620	W	HALF ADJUST ENTRY IN COL. 53 NOT H OR BLANK, ASSUME H.	You entered a character other than H or a blank in the Half-Adjust Field, which is illegal.	The compiler assumes that you entered H, specifying Half-Adjust.
621	W	FIELD ATTRIBUTE NOT SAME AS PREV DEFN, USE PREV DEFN.	You assigned to a field a different length or number of decimal positions than that previously assigned to the field.	The compiler uses the previous definition for the field. Correct the field attributes of the incorrect statement.
622	W	RESULT FIELD HAS INDEX BUT NOT ARRAY NAME, IGNORE INDEX.	You entered an array index in the Result Field but did not specify an array name.	The compiler ignores the array index. Remove the index, define the field name as an array on a File Extension Specification or use a proper array name.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
623	T	FILE NAME IN FACTOR 2 NOT DEFINED BY FILE DESC SPECS.	You entered a file name in the Factor 2 Field (Cols 33-42) that refers to a file not described in the File Description Specifications.	The compiler ignores the operation. Either define this file name on a File Description Specification, or use a proper file name.
625	W	RESULT FIELD IS MISSING (COL. 43-48).	You omitted the Result Field from an operation that requires this field.	The compiler ignores the operation. Enter the Result Field Wanted, and re-compile.
626	W	FACTOR 1 IS MISSING (COL. 18-27).	You omitted Factor 1 from an operation that requires it.	The compiler ignores the operation. Enter Factor 1 and re-compile the job.
627	W	FACTOR 2 IS MISSING (COL. 33-42).	You omitted Factor 2 from an operation that requires it.	The compiler ignores the operation. Enter Factor 2 and re-compile the job.
628	W	A RESULTING INDICATOR MUST BE SPECIFIED FOR THIS OPER.	You failed to specify a Resulting Indicator (in Cols. 54-59) for an operation that requires this indicator.	The compiler ignores the operation. Specify a Resulting Indicator and re-submit the job.
629	W	MISSING AN/OR LINE, IGNORE OPERATION INDICATOR GROUP.	You did not specify the required AN or OR Line for a group of more than three indicators that applies to this operation.	The compiler ignores the indicator group and performs the operation under any condition. Either give the previous statement an operator or include an AN/OR record.
630	W	MISSING ENDSR STATEMENT, ASSUME ENDSR.	You omitted the ENDSR operation from an internal subroutine.	The compiler assumes that an ENDSR statement follows the last statement with SR in the Control Level Field.
631	W	OPERATION INDICATORS MISSING ON AN/OR LINE.	You entered AN or OR in Cols. 7-8, but did not specify any indicators in the Indicators Field (Cols. 9-17).	The compiler uses the indicators specified in the preceding lines.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
632	W	INDICATORS IN COL. 9-17 NOT ALLOWED FOR THIS OPERATION. ASSUME BLANK.	You entered one or more indicators in Cols. 9-17, but this operation must not be conditioned with indicators.	The compiler assumes that you left Cols. 9-17 blank, and performs the operation under all conditions.
633	W	INDICATORS IN COL. 54-59 NOT ALLOWED FOR THIS OPERATION. ASSUME BLANK.	You entered one or more indicators in the Resulting Indicators Field, but such indicators are not permitted for this operation.	The compiler assumes that you left the Resulting Indicators Field Blank.
634	W	RESULT FIELD SHOULD BE BLANK FOR THIS OPER. ASSUME BLANK.	You entered a field name in the Result Field, which should remain blank for this operation.	The compiler assumes that you left the Result Field blank.
635	W	FACTOR 1 SHOULD BE BLANK FOR THIS OPER. ASSUME BLANK.	You specified Factor 1 for an operation that prohibits this operand.	The compiler assumes that you left the Factor 1 Field blank.
636	W	FACTOR 2 SHOULD BE BLANK FOR THIS OPER. ASSUME BLANK.	You specified Factor 2 for an operation that prohibits this operand.	The compiler assumes that you left the Factor 2 Field blank.
637	W	FIELD USED AS RESULT IS NON ALTERABLE. IGNORE SPEC.	You specified a look-ahead field, or UDAY, UMONTH, UYEAR, or UDATE as a result field.	The compiler ignores this operation. Specify a different field name and re-compile.
638	W	PAGE FIELD USED AS RESULT FIELD IS ALREADY RE-DEFINED. USE PREV DEFN.	You specified a pre-defined PAGE Field as the Result Field, but attempted to re-define the PAGE field.	The compiler will use the previous definition for the PAGE field.
639	W	NAME ALREADY HAS ANOTHER USE	The same name was used for two of the following: a field name, a subroutine name, and a tag.	Correct program and re-compile.
640	T	GOTO OPERATION MAY NOT BRANCH INTO OR OUT OF SUBROUTINE.	You specified a GOTO operation within a subroutine that transfers to an operation outside the subroutine, or vice-versa.	The compiler ignores the operation. Correct and re-compile.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
641	W	DEC POS ENTRY GREATER THAN FIELD LENGTH. ASSUME ZERO POS.	You defined a field where the number of decimal positions exceeds the length of the field.	The compiler assumes that the field contains no decimal positions. Correct either the number of decimal positions or the entry length, and re-compile.
642	T	NAME REFERENCED SHOULD BE TAG NAME.	You entered a GOTO request that does not reference a TAG operation label. (Fields, tags, and subroutines may not have the same name.)	The compiler ignores the operation. Either re-name the tag or re-name the field or subroutine.
643	T	A SUBROUTINE MAY NOT CALL ITSELF.	You included, within an internal subroutine, an operation that attempts to invoke that subroutine; this type of recursion is illegal.	The compiler ignores the operation. Correct and re-compile.
644	T	BEGSR STATEMENT MISSING. ASSUME BEGSR.	You did not begin an internal subroutine with the BEGSR operation.	The compiler assumes that the first operation in the subroutine is BEGSR. Include a BEGSR operation and re-compile.
645	T	BEGSR OUT OF SEQUENCE. IGNORE BEGSR LINE.	You included a BEGSR operation within the body of an internal subroutine, after the first operation.	The compiler ignores this BEGSR request. Place BEGSR record in correct order and re-compile.
646	W	DEBUG OPTION NOT SPECIFIED IN COL. 15 OF CONTROL RECORD.	You included one or more DEBUG requests in your program but did not specify the Debug option in the Debug Field (Col. 15) of your Control Record Specification.	The compiler does not execute the DEBUG requests.
647	W	FIELD LENGTH AND DEC POS SHOULD BE BLANK. ASSUME BLANK.	You entered the name of a previously-defined table or array in the Result Field, and also specified entries in the Field Length and/or Decimal Positions Field.	The compiler assumes that the Field Length and Decimal Positions Fields are blank.

C Calculation Specification Errors

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
648	W	DEC POS ENTRY (COL 52) SHOULD BE BLANK. ASSUME BLANK.	You specified a decimal position that should not be specified.	The compiler ignores this specification.
649	W	HALF ADJUST ENTRY (COL 53) SHOULD BE BLANK. ASSUME BLANK.	You specified a half-adjust that should not be specified.	The compiler ignores this specification.
650	W	BEGSR STATEMENT SHOULD HAVE SR OR BLANK IN COL. 7-8; ASSUME SR.	You specified a BEGSR operation that did not include SR or a blank in Cols. 7-8.	The compiler assumes that you entered SR in these columns.
651	W	FIELD LENGTH OVER 256 FOR ALPHANUMERIC OR OVER 15 FOR NUMERIC.	You specified a field length that exceeds the maximum length allowed for the Result Field (Cols. 29-51).	The specification is ignored. Re-define the field length and re-compile.
652	W	OPERATION NOT IMPLEMENTED ON THIS VERSION OF RPG.	You entered an operation code (in Cols. 28-32) that cannot be executed with this version of the compiler.	The compiler ignores this specification.
653	W	PRECEDING OPERATOR WAS NOT A DIV.	You specified an MVR (move remainder) operation that was not immediately preceded by a DIV (divide) operation.	The compiler ignores this MVR operation. Enter MVR immediately following the DIV operation, and re-compile.
654	T	DUPLICATE TAG NAME.	You assigned the same label to two or more TAG operations.	The compiler ignores the second TAG specification. Change one of the TAG labels and re-compile.
655	T	DUPLICATE SUBROUTINE NAME.	You assigned the same name to two or more subroutines.	The rest of the subroutine is compiled. Change one of the subroutine names and re-compile.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
656	T	SYMBOL NOT DEFINED.	You referenced the name of a data area or label of an operation that is not defined in your program.	Define the symbol by naming it in a result field with a defined length, by specifying it as a field name on Input Specifications, or by defining it as an array. Re-compile the program.
657	W	RESULT FIELD SHOULD BE ALPHANUMERIC FOR THIS OPERATOR.	You used a numeric field for the Result Field (Cols. 43-48), which should be alphanumeric for this operator.	The compiler uses the field as specified. Change the Result Field type and re-compile.
658	W	FIELD 1 SHOULD BE ALPHANUMERIC FOR THIS OPERATOR.	You used a numeric field for Factor 1 (Cols. 18-27), which should be alphanumeric for this operator.	The compiler uses the field as specified. Change the Factor 1 type and re-compile.
659	W	FIELD 2 SHOULD BE ALPHANUMERIC FOR THIS OPERATOR.	You used a numeric field for Factor 2 (Cols. 33-42), which should be alphanumeric for this operator.	The compiler uses the field as specified. Change the Factor 2 type and re-compile.
660	W	FIELD 2 MUST BE THE SAME TYPE AS FIELD 1.	You specified a different data format for Factor 1 than for Factor 2, which is illegal for this operation.	The compiler uses the field as specified. Change either the Factor 1 or Factor 2 type and re-compile.
661	W	RESULT FIELD SHOULD BE NUMERIC FOR THIS OPERATOR.	You used an alphanumeric field for the Result Field (Cols. 43-48), when a numeric field was required.	The compiler uses the field as specified. Change the Result Field type and re-compile.
662	W	FIELD 1 SHOULD BE NUMERIC FOR THIS OPERATOR.	You specified an alphanumeric field for Factor 1 (Cols. 18-27), when a numeric field was required.	The compiler uses the field as specified. Change the Factor 1 type and re-compile.
663	W	FIELD 2 SHOULD BE NUMERIC FOR THIS OPERATOR.	You specified an alphanumeric field for Factor 2 (Cols. 33-42), when a numeric field was required.	The compiler uses the field as specified. Change the Factor 2 type and re-compile.
664	W	RESULTING INDICATOR REQUIRED IN COL 54-55; ASSUME H0.	You did not enter the required indicator in the Resulting Indicator Field (Cols. 54-55).	The compiler assumes that you entered the H0 indicator.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
665	W	RESULTING INDICATOR REQUIRED IN COL 58-59; ASSUME H0.	You did not enter the required indicator in the Resulting Indicator Field (Cols. 58-59).	The compiler assumes that you entered the H0 indicator.
666	W	RESULTING INDICATOR REQUIRED IN COL 56-57; ASSUME H0.	You did not enter the required indicator in the Resulting Indicator Field (Cols. 56-57).	The compiler assumes that you entered the H0 indicator.
667	W	ILLEGAL I/O MODE IN IMAGE FILE.	You specified an operation that is incompatible with the Input/Output mode.	The compiler ignores the operation. Correct and re-compile.
668	W	INCOMPATIBLE FIELD1 LENGTH.	You specified a field length that was different from the Key Length specified in the File Description Specifications. (If packed decimal keys are used, the key field length is the number of digits divided by 2 plus 1.)	The length specified in the File Description Specifications is used. Change either the key length in the File Description Specification, or the Factor 1 field length.
669	T	FILE SPECIFIED IN FACTOR 2 NOT AN INDEXED DEMAND FILE.	You requested a SETLL operation on a file that is not a Demand file.	Change the file type to a Demand file.
670	W	FILE BEING READ IS NOT A DEMAND FILE.	You used the READ command for a file that was not a Demand file.	If you wish to ignore this warning, Demand reads are intermixed with all other accesses of the file. Otherwise change the specification to reflect a Demand file.
671	T	CURRENT FIELD USE INCONSISTENT WITH PREVIOUS USE.	You either used an alphanumeric field as a numeric field, or vice-versa.	Change specifications to make all references consistent.
672	W	FILE SPECIFIED IN FACTOR 2 NOT A CHAINING FILE.	A chain operation was attempted to a non-chaining file.	Correct the program and re-compile.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
673	T	CAN'T FORCE READ A FILE DESIGNATED BY BLANK IN COL 16 OF FILE SPEC.	You specified a FORCE operation code for a file which contained a blank in Column 16 of the File specification.	Correct your program and re-compile.
674	T	PARM OPERATIONS MUST IMMEDIATELY FOLLOW AN EXIT OPERATION.	You specified a PARM operation which does not follow an EXIT or another PARM operation.	Correct your program and re-compile.
675	T	RESULT FIELD LENGTH IN COLUMN 49-51 OF C SPEC MUST BE 6 OR 12 FOR TIME OPERATION.	You specified an incorrect entry for Columns 49 through 51.	Correct your program and re-compile.
676	T	DECIMAL POSITIONS FOR FIELD USED MUST BE 0 FOR THIS OPERATION.	The number of decimal positions defined for the field used in this operation is not zero.	Correct your program and re-compile.
677	T	DIGIT LENGTH FOR VARIABLE IN FACTOR 2 MUST BE 1 TO 4 DIGITS FOR TIME2 OPERATION.	You specified an incorrect digit length.	Correct your program and re-compile.
678	T	FACTOR 2 OR RESULT FIELD LENGTH MUST BE A NUMERIC VALUE FROM 1 TO 40 FOR TIME2.	You specified an incorrect FACTOR 2 or RESULT FIELD length.	Correct your program and re-compile.
679	T	START POSITION SPECIFIED IN FACTOR 2 PLUS RESULT FIELD LENGTH IS BEYOND 40 FOR TIME2.	You specified an incorrect combination of entries for the start position and the RESULT FIELD length.	Correct your program and re-compile.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
680	T	FIELD SIZE MUST BE 1 TO 8 DIGITS FOR THIS OPERATION.	The size defined for the field used in this operation is not within 1 to 8 digits.	Correct your program and re-compile.
681	T	LOW OR EQUAL RESULT INDICATOR MUST BE SPECIFIED FOR THIS OPERATION.	You must specify a result indicator in the low subfield (Columns 56 through 57) and/or the equal subfield (Columns 58 through 59).	Correct your program and re-compile.
682	T	IMAGE FILE LOCKING MODE IS NOT L, B, S, 1, 9 OR R TO ENABLE LOCKING.	You specified an incorrect locking mode entry to use the LOCK/UNLCK operators. Locking mode is specified in Column 66 of the file specification, IMAGE continuation record.	Correct your program and re-compile.
683	T	FILE NOT DESCRIBED WITH NOLOCK OR LOCK CONTINUATION RECORD TO ENABLE LOCKING.	You are using LOCK/UNLCK operators, but did not specify NOLOCK/LOCK on F spec to enable locking.	Correct your program and re-compile.
684	T	DBNAME IN RESULT FIELD MUST BE SAME NAME SPECIFIED IN IMAGE SPEC FOR FILE IN FACTOR 2.	Data base name specified in result field does not match name specified on the IMAGE continuation record for FACTOR 2 file.	Correct your program and re-compile.
685	T	LENGTH OF A VARIABLE USED TO INDEX AN ARRAY MUST BE 1 TO 9 DIGITS.	You specified an incorrect entry for the length of a variable.	Correct your program and re-compile.

Table B-7. Calculation Specification (C) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
687	W	NO INDICATOR IN COL 58-59. H0 WILL BE SET ON FOR ERRORS NOT FLAGGED BY INDICATORS IN COL 54-57.	You did not specify an equal indicator, so there is no way to inform you that an error other than those covered by the high and low indicators has occurred.	Specify and use the equal indicator for this operation.
688	W	SORTA OPERATOR ONLY ALLOWED ON NON-ALTERNATING ARRAY.	The SORTA operator is not allowed for your array.	Correct your program and re-compile.
689	T	GENERIC KEY ACCESS ONLY ALLOWED WITH KSAM FILES.	You specified *EQ, *GT, or *GE in the result field for a CHAIN or SETLL operation on a non-KSAM file.	Correct your program and re-compile.
690	T	GENERIC KEY ACCESS ONLY ALLOWED WITH ALPHANUMERIC OR PACKED KEYS.	You specified *EQ, *GT, or *GE in the RESULT field for a CHAIN or SETLL operation on a file for which File Specification Column 31 (Record Address Type) does not contain A or P.	Correct your program and re-compile.
691	T	PACKED GENERIC KEY MUST BE EXACT SIZE DEFINED IN FILE DESC. SPEC.	You specified *EQ, *GT, or *GE in the RESULT field and the generic key size differs from the size defined in Columns 29 through 30 (Key field length) of the F spec.	Correct your program and re-compile.

Table B-8. Output Specification (O) Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
701	T	INVALID FILENAME IN COL 7-14, SPEC IS DROPPED.	You referenced the name of an undefined file in the Filename Field (Cols 7-14).	Define the file name in the File Description Specifications, or use a different, valid file name in the Output Specifications, and re-compile.
702	T	'AND' OR 'OR' LINE NOT PRECEDED BY RECORD DESCRIPTION.	You entered an AND or OR Line that was not preceded by an entry that describes the record to which that line should apply.	Enter a record description specification before this AND/OR Line, and re-compile.
703	T	NO RECORD DESCRIPTION ENTRIES DEFINED YET.	You did not specify a record description as the first item in your Output Specification.	Define a valid record description and re-compile.
704	W	INVALID ENTRIES IN COL 32-74 FOR A RECORD DESCRIPTION SPECIFICATION, ASSUME BLANK.	You entered field description information in a record-description specification.	The compiler assumes Col. 32-74 are blank. Correct the record description specification and re-compile.
705	W	INVALID ENTRIES IN COL 7-22 FOR A FIELD DESCRIPTION SPECIFICATION, ASSUME BLANK.	You entered record description information in a field description specification.	Correct the field description specification and re-compile.
706	T	FILE AND RECORD ENTRIES IN COL 7-31 AND FIELD ENTRIES IN COLS 32-74 ON SAME LINE.	You entered both a record description and a field description on the same line.	Correct this error and re-compile.
707	W	NO FIELD DESCRIPTION ENTRIES FOR PREVIOUS RECORD.	You entered a record description specification that was not followed by any field description specifications.	Supply a field description entry and re-compile.
708	T	NO RECORD DESCRIPTION ENTRIES FOR THIS FIELD DESCRIPTION ENTRY, SPEC IS DROPPED.	You entered a field description specification that was not preceded by any record description specification.	Insert the record description entry, and re-compile.

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
709	T	MORE THAN 20 AND/OR LINES, SPEC IS DROPPED.	You exceeded the limit of 20 AND and/or OR lines in an AND/OR grouping.	Delete the appropriate number of AND/OR Lines and re-compile.
710	W	EDIT WORD IN COL 45-70 IS NOT ENCLOSED IN QUOTES, ASSUME BLANK.	You failed to place bounding quotation marks around an edit word.	Add quotation marks to the edit word and re-compile.
711	W	THE NUMBER OF REPLACEABLE CHARACTERS IN THE EDIT WORD IS NOT EQUAL TO THE FIELD TO BE EDITED.	You did not specify an equal number of replaceable characters in the edit word.	The compiler makes the number of characters in the source data equal to the number of replaceable characters in the edit word, either by truncation or padding with zeros. If you desire this result, ignore this warning message.
712	W	INVALID EDIT CODE IN COL. 38.	You entered a character other than X, Y, Z, 1, 2, 3, 4, A, B, C, D, J, K, L, M, or blank in the Edit Code Field (Col. 38).	Enter a correct character and re-compile.
713	W	EDIT CODE INVALID WITH CONSTANT OTHER THAN '*' OR '\$'.	You paired the wrong edit code with the constant that appears in Cols. 45-70.	Correct this error and re-compile.
714	W	INVALID FIELD NAME IN COL 32-37.	You used an improper or undefined field name in the Field Name Field (Cols 32-37).	Correct the field name and re-compile.
715	W	INVALID LINE TYPE IN COL 15, ASSUME D.	You used a character other than H, D, T, or E in the Type Field (Col. 15).	Enter the proper character and re-compile.
716	W	INVALID FETCH-OVERFLOW OR RELEASE-FILE IN COL 16, ASSUME BLANK.	You made an improper entry in the Fetch Overflow or Release-File Field (Col. 16).	Correct this error and re-compile.
717	W	INVALID SPACE ENTRIES IN COL 17-16, ASSUME BLANK.	You made an entry other than 0, 1, 2, 3, or blank in the Space Field (Cols. 17-18).	Correct this entry and re-compile.

O Output Specification Errors

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
718	W	INVALID SKIP ENTRIES IN COL 19-22 OR GREATER THAN FORM LENGTH SPECIFIED, ASSUME BLANK.	You made entries other than 01-99, A0-A9, B0-B2, or blank in the Skip Field, (Cols. 19-22) or the entry in this field exceeded the form length specified.	Correct this error and re-compile.
719	W	INVALID SKIP ENTRIES IN COL 19-22 OR CHANNEL NUMBER GREATER THAN 12, ASSUME BLANK.	You made invalid skip entries, or referred to a channel number greater than 12 in the Skip Field (Cols. 19-22).	Enter the correct skip requests or channel number and re-compile.
720	W	INVALID NOT ENTRY IN COL. 23, 26, OR 29; ASSUME N.	You entered a character other than N or blank in the Not Subfield, (Cols. 23, 26, or 29).	Enter N or blank, and re-compile.
722	W	INVALID OUTPUT INDICATOR IN COL. 24-25, 27-28, 30-31; ASSUME BLANK.	You entered an improper indicator in the Output Indicator Field (Cols. 24-25, 27-28, or 30-31).	Enter the proper indicator and re-compile.
723	W	INVALID FIELD LENGTH FOR Y EDIT CODE.	You specified a field less than 3 or greater than 6 digits long for the Y edit code.	Re-define the field length to 3-6 digits.
724	W	END POSITION NOT LARGE ENOUGH TO CONTAIN *PLACE FIELDS, OR IS INVALID.	You specified, in the End Position Field (Cols. 40-43), a value too small to allow output of the *PLACE data without overlapping the previous field, or you made another invalid entry in the End Position Field.	Correct this error and re-compile.
727	W	MISSING OR INCORRECTLY SPECIFIED END POSITION IN COL. 40-43; ASSUME END POSITION 1.	You omitted an entry from the End Position Field (Cols. 40-43), or included an entry other than 1 through 9999 in this field.	Make the proper entry in Cols. 40-43 and re-compile.
728	W	OUTPUT FILE DESCRIBED AS 'ADD' TYPE, EACH RECORD LINE MUST HAVE 'ADD' IN COL. 16-18, ASSUME 'ADD'.	You did not include ADD in Cols. 16-18 of each record line for an output file to which new records are to be appended at the end.	Correct this error and re-compile.

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
729	W	INVALID BLANK AFTER IN COL. 39, ASSUME BLANK.	You attempted to use the Blank After feature in a description of a constant, look-ahead field, or special DATE field.	Remove this entry from the constant or field description, and re-compile.
730	W	INVALID CONSTANT IN COLUMNS 45-70.	You entered an improperly-defined constant in the Constant or Edit Word Field (Cols. 45-70).	Define the constant properly and re-compile.
731	W	INVALID PACKED/BINARY OUTPUT ENTRY IN COL 44, ASSUME BLANK.	You made an entry in this field applying to alphanumeric data but such an entry properly applies only to a numeric data, or you entered a character other than P, B, L, R, 2, or 4.	Correct this entry and re-compile.
732	W	OVERFLOW INDICATOR INVALID FOR AN EXCPT RECORD.	You entered an overflow indicator in the Output Indicators Field (Cols. 23-31), in the description of an Exception Record.	Delete the overflow indicator and re-compile.
733	W	FETCH OVERFLOW INVALID WITH OVERFLOW INDICATOR ENTERED IN COL. 23-31; ASSUME NO FETCH.	You entered a request for Fetch Overflow in Col. 16 of a line conditioned by an overflow indicator.	Correct this error and re-compile.

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
734	W	OVERFLOW INDICATOR USED IS NOT ASSIGNED TO THIS FILE.	You attempted to use an overflow indicator in the Output Indicators Field (Cols. 23-31), but this indicator is not assigned to this file.	The compiler assumes that you assigned this indicator to this file. If this is what you intend, ignore this message; otherwise, correct the record and re-compile.
735	W	1P INDICATOR INVALID WITH TOTAL OR EXCPT RECORDS.	You attempted to use the First Page Indicator to condition a total or exception record.	Correct the record description and re-compile.
736	W	FETCH OVERFLOW INVALID WITH 1P INDICATOR, ASSUME NO FETCH OVERFLOW.	You entered a Fetch Overflow request (Col. 16) for a record conditioned with a First Page Indicator.	Correct the error and re-compile.
737	W	1P INDICATOR INVALID FOR A COMBINED FILE.	You specified a First-Page (1P) Indicator for a combined file.	Correct the error and re-compile.
738	W	INVALID INDICATORS USED IN AN 'AND' RELATIONSHIP WITH 1P.	You used an indicator with 1P in an AND relationship.	Correct this error and re-compile.
739	T	END POSITION ENTRY IN COL. 40-43 FOR CONSTANT, EDIT WORD, FIELD, OR ARRAY EXCEEDS RECORD LENGTH.	You specified an end position for an output record that exceeds the maximum length possible for that record — for instance, Position 150 for a record directed to a printer with lines of 132 characters.	Correct the end position to conform with the maximum length possible, and re-compile.
740	W	CONSTANTS IN COL. 45-70 INVALID FOR X, Y, OR Z EDIT CODES.	You did not leave Cols. 45-70 blank for an X, Y, or Z edit code.	Remove the characters in Cols. 45-70 and re-compile.
741	W	DECIMAL POSITIONS INVALID FOR FIELD EDITED BY Y CODE.	You applied the Y edit code to a non-integer numeric field.	Correct this error and re-compile.

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
742	W	NAME OF FIELD TO BE EDITED, BY CODE SPECIFIED IN COL 38, MISSING.	You specified an edit code in Col. 38 but did not name an edit field name in Cols. 32-37.	Enter a field name in Cols. 32-37, and re-compile.
743	T	INVALID FILE TYPE FOR OUTPUT RECORD.	You directed output to an input or display file.	Correct this error and re-compile.
744	W	T OR E ENTRY IN COL. 15 INVALID FOR COMBINED FILE.	You directed total or exception records to a combined file.	Correct this error and re-compile.
745	W	BLANK AFTER ENTRY IN COL 39 INVALID WITH RESERVED WORD OTHER THAN PAGE; ASSUME BLANK.	You attempted to use the Blank-After feature with a field named *PLACE, *PRINT, UDATE, UMONTH, UYEAR, UDAY, or *ERROR.	Delete the blank-after request from Col. 39.
746	W	BLANK AFTER SPECIFIED FOR A CONSTANT.	You requested the blanking out of a constant instead of a data field.	Correct this error and re-compile.
747	W	FORMS POSITIONING SPECIFIED ON CONTROL RECORD, BUT MISSING A RECORD CONDITIONED BY 1P.	You requested Form Positioning in Header spec Column 41, but did not specify an output record conditioned by the first page indicator (1P).	This is an informational message only, no recovery is necessary.
748	W	'ADD' IN COL. 16-18 NOT ALLOWED ON AND/OR LINES, ASSUME BLANK.	You entered an ADD request on an AND or OR Line.	Remove the ADD request and re-compile.
749	W	INVALID ENTRY IN COL. 17-22 FOR 'AND' LINE, ASSUME BLANK.	You entered one or more characters in Cols. 17-22 on an AND Line.	Remove the characters from these columns and re-compile.
750	W	INVALID INDEX IN COLS. 32-37.	You used an improper array index in the Field Name Field.	Correct the invalid index and re-compile.

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
751	W	*PRINT MAY BE USED ONLY ONCE FOR EACH RECORD.	You used the *PRINT option more than once in a record description.	Correct this record and re-compile.
752	W	COL 7-22 AND 38-74 MAY NOT BE USED FOR *PRINT SPECIFICATION.	You placed entries in these columns, which is illegal for the *PRINT option.	Remove these entries or change the *PRINT request, and re-compile.
753	W	ALPHANUMERIC DATA CANNOT BE EDITED.	You attempted to apply an edit code or edit word to alphanumeric data, but these can only be used with numeric information.	Correct this record and re-compile.
754	W	PACKED/BINARY IN COLUMN 44 CANNOT APPEAR TOGETHER WITH EDIT CODE OR EDIT WORD.	You attempted to apply a Packed/Binary field to an edit code or edit word.	Correct this error and re-compile.
755	W	END POSITION SPECIFIED IN COLUMNS 40-43 CANNOT CONTAIN THE WHOLE DATA ITEM.	You did not specify an edit field long enough to hold the entire item to be edited.	Specify the proper field length and re-compile.
756	W	PACKED/BINARY IN COLUMN 44 IS NOT BLANK FOR ALPHANUMERIC DATA, ASSUME BLANK.	You entered a character specifying numeric data (Col. 44) but previously defined the field as alphanumeric.	Correct this error and re-compile.
757	W	TARGET LENGTH SPECIFIED BY PACKED/BINARY IN COLUMN 44 MAY BE TOO SMALL TO CONTAIN THE DATA.	You may have specified an output field too short to hold the data to be output.	Re-specify the output field length and re-compile.
758	W	THE SKIP IN COLUMNS 19-22 MUST NOT BE BEYOND THE FORM LENGTH DEFINED ON THE LINE COUNTER SHEET.	You directed the printer carriage to skip to a line beyond the form length but still on the current page.	Correct the skip request and re-compile.

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
759	W	INVALID ENTRY IN COL 7-13 FOR 'AND/OR' LINE, ASSUME BLANK.	You entered characters in these columns, but none are allowed for an AND or OR Line.	Remove the characters from Cols. 7-13 and re-compile.
760	T	'DEL' IN COLS. 16-18 ONLY ALLOWED FOR AN IMAGE/KSAM/INDEX UPDATE FILE.	You specified DEL in these columns for the wrong type of file.	Correct this error and re-compile.
761	W	'ADD' IN COLS. 16-18, BUT FILE IS NOT 'ADD' TYPE, BLANK IS ASSUMED.	You specified ADD in Cols. 16-18 of the Output Specifications but did not also specify A in the File Addition Field (Col. 66) of the File Description Specifications.	Remove ADD from Cols. 16-18 of the Output Specifications, or specify A in Col. 66 of the File Description Specifications.
762	W	*PRINT FILE NOT 80 BYTES LONG, ASSUME 80.	You specified *PRINT for a file that is not a card or card-image oriented file.	Remove the *PRINT specification, or use a proper card or card-image file.
763	T	IMAGE/3000 OPEN MODE 2 NOT ALLOWED FOR DATABASE SPECIFIED AS OUTPUT, UPDATE-ADD, OR UPDATE-DEL.	You cannot specify Open Mode 2 for this database.	Correct your program and re-compile.
764	W	OUTPUT, UPDATE, OR COMBINED FILE HAS NO OUTPUT SPECIFICATIONS.	You have not entered Output Specifications for an Output, Update or Combined file.	This is an informational message only, no recovery is necessary.

Table B-8. Output Specification (O) Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
765	W	FETCH OVERFLOW INVALID IF NO OVERFLOW INDICATOR ASSIGNED TO FILE. ASSUME NO FETCH OVERFLOW.	You specified F in Column 16 of Output Specifications with no overflow indicator defined in Columns 33 through 34 of the File Specification.	Change your program and re-compile.
766	W	SPACE BEFORE/ AFTER INVALID FOR INDEXED FILE, ASSUME BLANK.	You specified spacing in Columns 17 through 18 for an indexed file.	The compiler assumes that the entry was blank.
767	W	COL 16-18 NOT BLANK, 'ADD' OR 'DEL' FOR UPDATE FILE, ASSUME UPDATE OPERATIONS - NOT ADD.	You did not specify a blank, ADD, or DEL in Columns 16 through 18 for an update file.	The compiler perform update operations on the file, not add operations. Correct your program and re-compile.

Table B-9. Compiler Subsystem Command Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
801	W	ERROR IN COMMENT	The set of broken brackets is not correct.	The compiler ignores the command image from the comment on. Correct and re-compile.
802	W	MISSING SYMBOL	The compiler expects a symbol after the \$ in Col. 6, but none was present.	The compile ignores the rest of the command image, including any incomplete option present. Correct and re-compile.
803	W	ILLEGAL SYMBOL	The compiler encountered an unexpected symbol.	The compiler ignores the symbol. Correct and re-compile.
804	W	MISSING OPTION OR INVALID COMMAND	On a \$CONTROL command, the compiler either did not encounter an option following the command or it encountered an invalid option.	The compiler ignores the command image. Correct and re-compile.
805	W	MISSING COMMA	You did not include a comma expected as a separator.	The compiler assumes that a comma is present. Correct and re-compile.
806	W	MISSING EQUAL SIGN	You did not include an equal sign expected after the keyword.	The compiler assumes that an equal sign is present. Correct and re-compile.
807	W	ILLEGAL NUMBER	You specified a alphanumeric entry where a numeric entry was required.	The compiler ignores the option currently being scanned. Correct and re-compile.
808	W	NUMBER OUT OF RANGE	A numeric entry is out of range.	The compiler ignores the option current being scanned. Correct and re-compile.
809	W	STRING TOO LARGE	The concatenation of character substrings exceeds a total of 103 characters.	The string is truncated at 103 characters. Correct and re-compile.
810	W	ILLEGAL STRING	You failed to terminate a character substring with a delimiting quotation mark.	The compiler ignores the string. Correct and re-compile.

Table B-9. Compiler Subsystem Command Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
811	W	SEQUENCE ERROR IN VOID RECORD	A record to be voided by an \$EDIT command is out of proper sequence.	The compiler ignores the \$EDIT command. Correct and re-compile.
812	W	MAXIMUM ERRORS EXCEEDED.	The maximum number of errors allowed by the ERRORS= parameter of the \$CONTROL command (or the default value of 100) has been exceeded.	(Not applicable.)
813	W	DUPLICATE ENTRY - IGNORED.	You specified a \$command or option more than once.	Remove duplicate entry.
814	W	ILLEGAL NAME	You specified an incorrect value in \$CONTROL NAME= option.	Correct the option and re-compile.
815	W	\$COPY NOT FIRST LINE OF PROGRAM OR COPYLIB PRE- PROCESSING NOT NOT DONE - SPECIFICATION DROPPED.	You either did not specify a \$COPY command as the first line, or you did specify the \$COPY command but the copylib preprocessor did not execute properly.	Specify the \$COPY command properly; make sure RPGCOPY.PUB.SYS is available and can be executed; then re-compile.

Table B-10. Other Errors

Error Number	Type	Message	Probable Cause	Recovery Procedure
901	W	INDICATOR DEFINED BUT NOT REFERENCED.	Your program defined an indicator that was not referenced in any specification.	If you intended to reference this indicator, enter a correct reference and re-compile.
902	W	INDICATOR REFERENCED BUT NOT DEFINED.	Your program referenced an indicator that was not defined in any specification.	Correct the error and re-compile.
903	T	INVALID FILENAME IN COLUMNS 7-14, SPEC DROPPED.	You used an improper file name in the File Name Field for a Table/Array File Name (A) Record.	The compiler ignores this specification.
904	T	CANNOT OPEN FILE SPECIFIED BY 'A' SPEC.	The system could not open the file named on a Table/Array File Name (A) Record.	Check to determine the reason for this file error, correct the error, and re-compile.
905	T	CANNOT OPEN TEMPORARY FILE TO PROCESS COMPILE-TIME TABLE.	The compiler cannot open the temporary file for processing a compile-time table.	Check to determine the reason for this file error, correct the error, and re-compile.
906	W	THE FILE DEFINED BY 'A' SPEC IS EMPTY.	The file named on the Table/Array File Name Record does not contain table or array records.	Supply the compiler with the right file and re-compile your program.
907	T	I/O ERROR OCCURRED DURING READING COMPILE-TIME TABLE.	The system could not read the compile-time table.	Check to determine the reason for this file error, correct it, and re-compile.
908	T	I/O ERROR OCCURRED DURING WRITING COMPILE-TIME TABLE.	An input/output error occurred while the compiler was copying your table/array files to a temporary file; the most probable cause was insufficient disc space remaining.	Check to determine the reason for this file error, correct it, and re-compile.

Table B-10. Other Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
909	W	ALTERNATE COLLATING SEQUENCE, COMPILE-TIME TABLE/ARRAY OR TRANSLATION TABLE NOT FOUND.	You did not define an alternate collating sequence, table or array, or file translation table in the file named on the required Table/Array File Name (A) Record.	Supply a Table/Array File Name Record (after the last Output Specification), and re-compile.
910	W	THE FILES SPECIFIED BY 'A' SPEC ARE NOT ENOUGH TO INITIALIZE COMPILE-TIME TABLES.	The files named on the Table/Array File Name (A) Records do not contain enough entries to fill the Tables/Arrays.	Supply the compiler with a proper file (or files) and re-compile.
911	W	ILLEGAL HEXADECIMAL DIGIT IN COMPILE-TIME TABLE.	You entered an improper hexadecimal value (other than 0-9 or A-F) in a compile-time table - this value does not represent an ASCII character.	The compiler assumes that this character is zero. Correct and re-compile.
912	W	ILLEGAL OCTAL DIGIT IN COMPILE-TIME TABLE.	You entered an improper octal value (other than 0-7) in a compile-time table - this value does not represent an ASCII character.	The compiler assumes that this character is zero. Correct and re-compile.
913	W	THE NAMES IN COLUMNS 1-8 OF A FILE TRANSLATION TABLE ARE NOT THE SAME.	You specified more than one file name in Columns 1-8 of a group of File Translation Records supposedly belonging to the same file.	The compiler assumes that the file names are the same.
914	W	FIELD NAME(S) DEFINED BUT NOT REFERENCED.	You did not reference the field name(s) you defined.	Correct your program and re-compile.
915	W	FILE NAME(S) DEFINED BUT NOT REFERENCED.	You did not reference file name(s) you defined.	Correct your program and re-compile.

Table B-10. Other Errors (Continued)

Error Number	Type	Message	Probable Cause	Recovery Procedure
916	W	NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE.	You are attempting to compare an alphanumeric field to a numeric field.	RPG temporarily converts the numeric field to alphanumeric to do the comparison. This is an informational message only.
917	T	AUTOMATIC RECORD-LEVEL LOCKING FOR IMAGE OUTPUT REQUIRES ENTRY IN FILE DESC. SPEC COLUMNS 35-38.	You specified 'R' in Column 66 of the IMAGE continuation record without entry in Columns 35 through 38 of the File Specification.	Correct your program and re-compile.
918	W	COMPILE-TIME TABLE ERROR - SEQUENCE ERROR.	You specified an incorrect order for the compile-time table.	Correct your program and re-compile.
919	W	COMPILE-TIME TABLE ERROR - BAD DATA (INVALID DIGIT).	You specified invalid digits in a numeric table/array.	Correct your program and re-compile.
920	W	COMPILE-TIME TABLE ERROR - TOO MANY ITEMS FOR TABLE/ARRAY.	You specified too many items in the compile-time table.	Correct your program and re-compile.
921	T	FOR LOKUP OPERATION, STORAGE LENGTHS OF FACTOR 1 AND FACTOR 2 ARE NOT THE SAME.	You specified different lengths for FACTOR 1 (search item of LOKUP) and for FACTOR 2 (table/array being searched).	Correct your program and re-compile.
922	W	ALTSEQ WILL NOT BE APPLIED TO THIS OPERATION.	You declared an alternate collating sequence; however, the compiler does not use the alternate sequence for this operation.	The compiler does not use alternate collating sequence for this operation.

B-2. RUN (EXECUTION) TIME ERRORS

Run-time errors are those encountered during object-program execution. They fall into three general classes:

1. RPG-detected errors.
2. IMAGE-detected errors.
3. USWITCH command errors.

Normally, when an RPG/3000 object program encounters any such errors, a run-time message describing the error appears on the operator's console (in a batch job) or terminal (in an interactive session), and the program pauses. (RPG-detected errors can also be handled in other ways, as discussed below.) The operator or terminal user can take any of three options:

1. Continue processing in-line,
2. skip the erroneous record and continue processing, or
3. terminate.

The three types of run-time messages are described below.

B-3. RPG-Detected Errors

RPG-detected errors are detected and reported by code supplied automatically by the RPG/3000 compiler. RPG/3000 allows you (at compile-time) to direct the way in which RPG-detected run-time errors are handled. You do this by:

1. Re-directing or suppressing transmission of error messages, by making an entry in the **Error Log Field** (Column 55) of the Control Record Specification (discussed in **paragraph 3-18**), and
2. Specifying a pre-selected response to errors, by making an entry in the **Error Response Field** (Columns 56 through 71) of the Control Record Specification Sheet (**discussed in paragraph 3-19**).

Note: You can re-direct or suppress transmission of RPG-detected messages only if you also specify a pre-selected response.

The seventeen possible RPG-detected error messages are listed in Table B-11. This table shows:

- The error number (1 through 17).
- The message content.
- The probable cause of the error.
- The recovery procedure suggested.
- The column number (56 through 71) in the **Error Response Field** that corresponds to this error. (On the specification sheet, you make an entry (0 through 5) in this column to determine how you want the error handled; see **paragraph 3-19**.)

- The *ERROR Code. (This is a letter written into the one-byte, alphanumeric field named *ERROR. Whenever this error occurs, and the continue (0) or bypass (1) option is chosen in the Error Response Field, the *ERROR field can be tested by using Calculation Specifications.)
- The entries (0-5) allowed in the Error Response Field. (Not all pre-selected responses apply to all RPG-detected errors.)

Note: Error No. 1, which is a terminal error on a file, is always treated under Error Response No. 5, automatically. There is no provision for specifically requesting a pre-selected response for this error in the Error Response Field.



Table B-11. RPG-Detected Errors

Error Number	Message	Probable Cause	Recovery Procedure	Error Response Column	*Error Code	Error Response Entries
1	FATAL FILE ERROR	Check MPE error in File Information Display or IMAGE error message.	Determine cause of file error and correct the program.	NONE	F	NONE
2	UNIDENTIFIED RECORD, FILE-NAME = xxxx, RECORD NUMBER = nnn	Record number nnn on the input file named xxxx was not identifiable.	Choose one of the error response options. Either include this record type on Input Specifications or delete record from the file.	56	A	0,1,2,3,4,5
3	MATCHING RECORD SEQUENCE ERROR, FILENAME = xxxx, RECORD NUMBER = nnn	The matching record with record number = nnn and filename = xxxx was not in sequence.	Choose one of the error response options and correct the record sequence.	57	B	1,2,3,4,5
4	INPUT SEQUENCE ERROR, FILENAME = xxxx, RECORD NUMBER = nnn	Record number nnn on the input file named xxxx was not in proper sequence.	Rearrange the record in the proper sequence and execute the program again.	58	C	1,2,3,4,5
5	INDEX INVALID, LINE NUMBER = nnn, INDEX = nnn	The array index is 0 or greater than the number of array elements.	Choose one of the error response options and correct the program.	59	D	0,2,3,4,5
6	NEGATIVE SQUARE ROOT, LINE NUMBER = nnn	Your program calculated a square root that was a negative value.	Choose one of the error response options and correct the data.	60	E	0,2,3,4,5
7	TABLE/ARRAY SEQUENCE BAD DATA ERROR, OVERFULL. FILE = xxxx, LINE NUMBER = nnn, VALUE = xxx	The Table or array on file xxxx with a line number = nnn and value = xxx had a sequence, bad data, or overfull error.	Choose one of the error response options and correct the table or array.	61	H	0,2,3,4,5

Table B-11. RPG-Detected Errors (Continued)

Error Number	Message	Probable Cause	Recovery Procedure	Error Response Column	*Error Code	Error Response Entries
8	RECORD NOT FOUND, FILENAME = xxx, STATUS = xxx	The record sought was not found. (STATUS portion appears only for IMAGE file).	Choose one of the error response options and make the appropriate changes.	62	U	1,2,3,4,5
9	SPECIAL FILE ERROR, FILENAME = xxxx, STATUS = xxxx	The special file processing routine returned an error.	Check the error status and make the appropriate changes.	63	J	0,1,2,3,4,5
10	RDEXIT ERROR, FILENAME = xxxx, STATUS = xxxx	The Read Exit file processing routine returned an error.	Choose one of the error response options and make the appropriate changes.	64	J	0,1,2,3,4,5
11	ARITHMETIC OVERFLOW, LINE NUMBER = nnn, VALUE = nnn	The arithmetic result exceeded the number of digits the field could hold.	Correct the program (make the field size larger).	65	N	0,2,3,4,5
12	DIVIDE BY ZERO, LINE NUMBER = nnn	The number was divided by zero.	Choose one of the error response options and correct the program.	66	O	0,2,3,4,5
13	INVALID NUMERICAL DATA, LINE NUMBER = [nnn, VALUE = nnn, RECORD NUMBER = nnn, COLUMN NUMBER = nnn], or [VAL at DB + xxx ($\frac{L}{R}$)]	Either an input field or an alphanumeric field, being moved to a numeric field, contained characters other than numeric characters or embedded blanks.	Choose one of the error response options and correct the program.	67	P	0,2,3,4,5
14	BINARY CONVERSION OVERFLOW, LINE NUMBER = nnn, VALUE = nnn	The binary output field was not large enough to hold the value.	Define a larger binary field.	68	Q	0,2,3,4,5
15	INDICATOR H X IS ON (H0-H9)	These Halt indicators were either set on programmatically or the RPG program has set on H0 because of a file error.	Choose one of the error response options and correct the program.	69	NONE	0,2,3,4,5
16	REL REC# INVALID, FILE NAME = xxxx	An attempt was made to read or write a relative record number past the end of the file or before the beginning of the file.	Choose one of the error response options and correct the program.	70	M	0,2,3,4,5

Table B-11. RPG-Detected Errors (Continued)

Error Number	Message	Probable Cause	Recovery Procedure	Error Response Column	*Error Code	Error Response Entries
17	DUPLICATE KEY, FILENAME = xxxx, KEY = kkkk	The program attempted to write a record having the same key as a record already in the file but duplicate keys are not allowed.	Redefine the file to permit duplicate keys, or delete one of the records.	71	Y	0,2,3,4,5

B-4. IMAGE-Detected Errors

IMAGE-detected errors are detected by the IMAGE/3000 Data Base Management Subsystem. The corresponding error messages are summarized in the appendix of the IMAGE/3000 Reference Manual.

B-5. USWITCH Command Errors

In batch mode, various error messages may follow the attempted execution of a USWITCH command (Section 4-33). The messages are summarized in Table B-12. (In an interactive session, no messages are output for the USWITCH command.)

Table B-12. USWITCH Command Errors

Message	Probable Cause	Recovery Procedure
I/O ERROR ON \$STDLIST	An attempt to write output to the standard list device failed (for unknown reasons).	The job or session terminates. Determine the reason if possible, and correct and re-run.
I/O ERROR ON \$STDIN	An attempt to read input from the standard input device failed (for unknown reasons).	The job or session terminates. Determine the reason, and correct and re-run.
INVALID INPUT DATA	You included invalid information in a <i>resp</i> subparameter of the USWITCH command.	The batch job terminates. Correct and re-run. (This cannot occur in a session.)
UNEXPECTED END OF FILE	An end-of-file indication was unexpectedly encountered in the input stream during interpretation of a USWITCH command.	The batch job terminates. (A session assumes that undefined indicators are off and continues.) Correct and re-run.



RPG/3000 provides several tools to help you detect and resolve errors in your programs. For instance, you can usually debug compile-time errors most easily by examining the Compile-Time Error Messages (discussed in Appendix B) together with the listings produced during compilation — the Source Program, Symbol Table, and Cross Reference Listings (Section XI). You can debug execution (run) time errors by consulting the run-time error messages output (Appendix B); by using the DEBUG operation to list indicators that are on and data field contents at various points in your program (Sections III and VIII); or by using an Error Dump, as described below.

You request an Error Dump by entering 4 or 5 in the Error Response Field (Columns 56 through 71) of the Control Record Specifications. Alternatively, if a file error occurs, RPG/3000 assumes an entry of 5 and provides the dump automatically. In either case, when an error is encountered, the following output appears, in the order indicated:

1. The Run-Time Error Message describing the error.
2. In the case of an IMAGE/3000 file error, a status message.
3. In the case of a file error on a file not used by IMAGE/3000, a File Information Display output by the operating system. (This display is explained in the MPE reference manuals.)
4. The Error Dump, showing the data in storage belonging to your program at the time the program was terminated. (The executable program code is maintained by MPE/3000 in a different portion of memory, and does not appear in an Error Dump.) In the Error Dump, the data appears in two portions — the Formatted Portion and the Unformatted Portion, as explained below and illustrated in Figure C-1. In this figure, the dump begins immediately after the Run-Time Error Message (Item 1).

C-1. FORMATTED PORTION

The first set of information in the Formatted Portion of the Error Dump shows the addresses of the pointers to various tables and working storage areas (Item 2) used by the program. Each pointer, in turn, contains the address of the first word of the table or storage area.

The next set of information (Item 3) shows current information from the File Table for each file used by the program. For instance, the file named LETIN (Item 4) is the file that was being accessed when the dump was taken (Item 5) and is known to the operating system by the file number 1 (Item 6). It resides on a device designated by the device type READ (Item 7), which is a card reader (Item 8). It is a primary (Item 9) input (Item 10) file. Within the file, each record contains 80 bytes (characters) (Item 11) and records are blocked one record per block (Item 12). The records are fixed-length (Item 13), arranged for sequential access (Item 14). The contents of the input buffer for LETIN next appears, in ASCII (Item 15), followed by the contents of the File Table in octal (Item 16), in turn followed by the contents of the control-level (Item 17) and matching (Item 18) fields for the file.

The third set of information shows the setting of each fixed indicator (Item 19) and general indicator (Item 20) used by the program.

Debugging Techniques

The contents of the master level break (Item 21) and matching (Item 22) fields appear next. This is followed by the addresses and contents of all numeric fields used by the program, shown in hexadecimal (Item 23); (this is the only hexadecimal output in the entire Error Dump).

Next, the addresses and contents of all alphanumeric fields appear (Item 24). Note that in the example, most of these fields contain blanks.

The last area of the formatted portion shows the description and current contents of tables and arrays used by the program (Item 25). For instance, the array defined on Line 7 of the source program is a run-time array (Item 26), not arranged in ascending or descending sequence (Item 27), containing alphanumeric data (Item 28), not related to an alternating table (Item 29). Each entry is one character long (Item 30). The pointer to the array is located in the right byte of the word at address %000620 (Item 31). The array contains 64 entries (Item 32). An octal listing of the Control Information for the array from the Table/Array Control Table (Item 33) and an ASCII list of the array contents (Item 34) follow.

C-2. UNFORMATTED PORTION

The Unformatted Portion of the Error Dump shows the current contents of the entire Z-DB area of the data stack used by the program in main memory, beginning with the address DB+0. (The Z-DB area and other portions of the stack, are explained in MPE/3000 reference manuals.) The dump is written in ten columns; the first column shows the address (in octal) of every eighth word in the stack; the next eight columns show the contents of the next eight words in the stack, beginning with the word whose address appears on the same line in the first column; the tenth column shows the ASCII equivalent of the contents of those words, if applicable.

In the Unformatted Portion, the following information appears in the order discussed below, (also keyed to the example in Figure C-1):

- **General Pointers and Working Storage (from DB+0 to DB+66).** The first location, DB+0 (Item 35) contains the pointer 003325 (Item 36); the next location, DB+1 (Item 37) contains 000166.
- **Fixed Indicators (from DB+67 to DB+125),** each occupying a word, where 0 indicates that the indicator is off and -1 (%177777) indicates that it is on. The indicator settings are listed in this order:

- H0 through H9
- L0 through L9
- LR
- MR
- 0A through 0G
- 0V
- 1P

The first indicator shown, H0 (Item 38), is set **off**.

- **General Indicators Used (from DB+126 to DB+165),** listed in numeric order, and their settings. Only those indicators specified in your program are shown. The first general indicator in the example (Item 39) is set off.
- **File Translation Tables** (not used in the example).
- **Alternate Collating Sequence Table** (not used in the example).

- **File Tables Used.** Each contains 37 words, in the format shown in Tables C-1 and C-2. The example contains three tables, beginning at the points indicated by Items 40, 41, and 42.
- **Record Buffer Areas,** showing the contents of all input/output record buffers. The contents of the first buffer appears as Item 43.
- **Contents of Numeric Fields Used.** In the example these fields begin at Item 44.
- **Contents of Alphanumeric Fields Used.** In the example, these fields begin at Item 45.
- **Level Break Table,** showing the contents of fields that caused control level breaks. In the example, this table begins at Item 46.
- **Matching Field Table,** showing the contents of fields that match. This table begins at Item 47.
- **Chaining Field Table,** showing the contents of all chaining fields used. The format of this table appears in Table C-3. (It is not represented in this example.)
- **Table/Array Control Table.** The structure of this table appears in Table C-4. In the example, the table begins at Item 48.
- **Record Identification Indicator Pointers.** In the example, this listing begins at Item 49.
- **Bit Mask of All Indicators Used,** in this order:

```

00 through 99
H0 through H9
L0 through L9
LR
MR
0A through 0G
0V
KA through KN
KP through KY
(6 bits reserved for future use)
F0 through F9
(8 bits reserved for future use)
1P
U1 through U8
(5 bits reserved for other use)

```

In the example, the mask for the first indicator appears as Item 50.

The dump terminates with a run-time error message from the operating system (Item 51). The format of this message is described in the MPE reference manuals.

Debugging Techniques

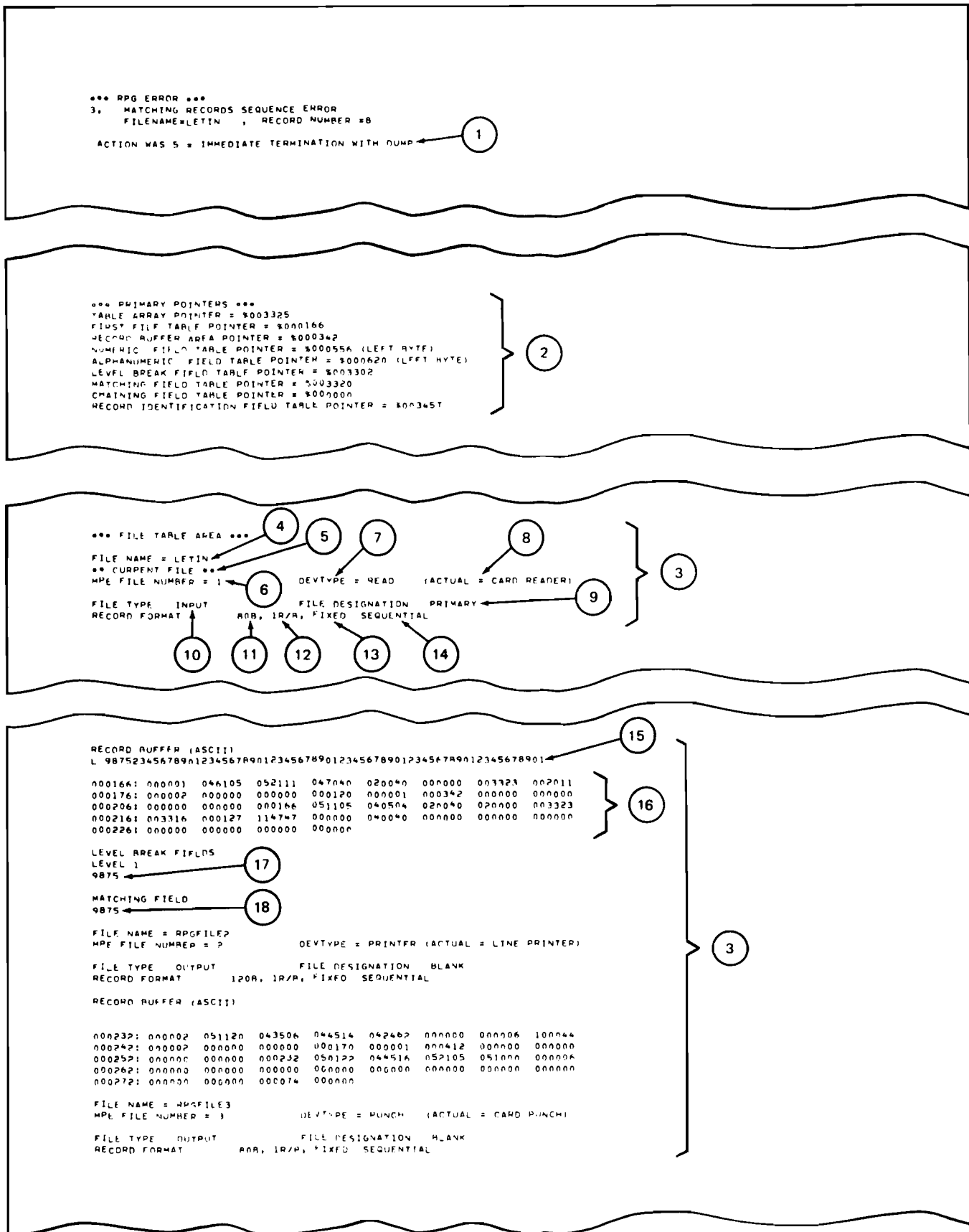


Figure C-1. Error Dump

RECORD BUFFER (ASCII)

```
000276: 000003 051120 043506 044514 042463 000000 000000 000044
000306: 000002 000000 000000 000120 000001 000506 000000 000000
000316: 000000 000000 000276 050125 047103 044040 020000 000000
000326: 000000 000000 000000 000000 000000 000000 000000 000000
000336: 000000 000000 000000 000000
```

3

*** LEVEL INDICATORS ***

```
L0 = ON    L1 = OFF    L2 = OFF    L3 = OFF    L4 = OFF
L5 = OFF   L6 = OFF   L7 = OFF   L8 = OFF   L9 = OFF
```

*** HALT INDICATORS ***

```
H0 = OFF   H1 = OFF   H2 = OFF   H3 = OFF   H4 = OFF
H5 = OFF   H6 = OFF   H7 = OFF   H8 = OFF   H9 = OFF
```

19

*** OVERFLOW INDICATORS ***

```
OA = OFF   OB = OFF   OC = OFF   OD = OFF
OE = OFF   OF = OFF   OG = OFF   OH = OFF
LI = OFF   HI = OFF   IP = OFF
```

*** OTHER INDICATORS ***

```
1 = OFF   2 = OFF   3 = OFF   4 = OFF   5 = OFF   6 = OFF
9 = OFF  10 = OFF  11 = OFF  12 = ON   13 = OFF  14 = OFF
15 = OFF  16 = OFF  17 = OFF  18 = OFF  19 = OFF  20 = OFF
23 = OFF  24 = OFF  25 = OFF  26 = OFF  27 = OFF  28 = ON
49 = OFF  50 = OFF  51 = OFF  52 = OFF  53 = ON   54 = OFF
98 = OFF  99 = ON
```

20

*** MASTER LEVEL BREAK FIELD ***

LEVEL 1
9876 ← 21

*** MASTER MATCHING FIELD ***

9876 ← 22

*** NUMERIC FIELDS ***

```
000556: 0000 0000 0000 0000 0000 0000 0000 0000
000566: 0000 0000 0000 0000 0111 9740 0190 0110
000576: 0740 001C 0006 *C06 5C06 *C00 1C00 1C00
000606: 1C00 1C00 1C00 0000 0000 0000 0000 0000
000616: 0000 0000
```

23

Figure C-1. Error Dump (Continued)

Debugging Techniques

```
*** ALPHANUMERIC FIELDS ***  
000A20: H98762145678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234  
000715: 5678901  
001012:  
001107:  
001204:  
001301:  
00137A:  
001473:  
001570:  
001665:  
001762:  
002057:  
002154:  
002251:  
002346:  
002443:  
002540:  
002635:  
002732:  
003027:  
003124:  
003221:  
  
          ABCDEFGHIJKLMNOPQRSTUVWXYZ,.,-+_*!@#$%^&'()*+,-./:;=<=>?[]\|_`{|}~" '() *+,-./:;=<=>?[]\|_`{|}~" '()  
          A                                         JANUARY 30, 1975                                         987A  
  
*** TABLES AND ARRAYS ***  
ARRAY DEFINED ON LINE NUMBER = 7
```

24

```
RUN TIME ARRAY, NO SEQUENCE, ALPHANUMERIC, NOT ALTERNATING  
ENTRY LENGTH = 1          POINTER TO ARRAY = 8000620 (RIGHT BYTE) NUMBER OF ENTRIES = 64
```

26 27 28 29 30 31 32 25

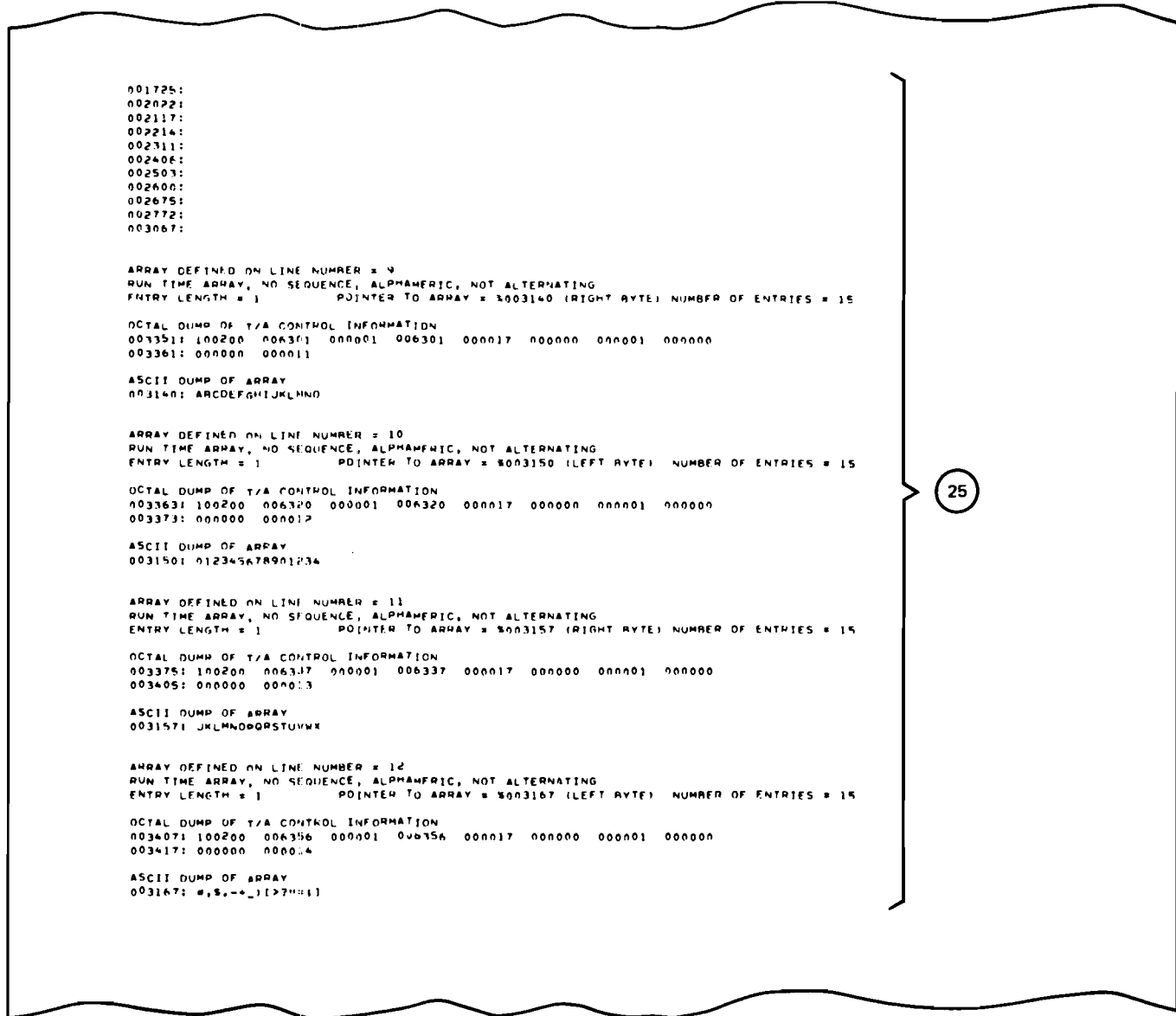
```
OCTAL DUMP OF T/A CONTROL INFORMATION  
003325: 100200 001441 000001 001441 000100 000000 000001 000000  
003335: 000000 000007  
ASCII DUMP OF ARRAY  
000A20: 9876214567890123456789012345678901234567890123456789012345678901
```

33 34

```
ARRAY DEFINED ON LINE NUMBER = 8  
RUN TIME ARRAY, NO SEQUENCE, ALPHANUMERIC, NOT ALTERNATING  
ENTRY LENGTH = 1          POINTER TO ARRAY = 8000660 (RIGHT BYTE) NUMBER OF ENTRIES = 2400  
OCTAL DUMP OF T/A CONTROL INFORMATION  
003337: 100200 001541 000001 001541 004540 000000 000001 000000  
003347: 000000 000010  
ASCII DUMP OF ARRAY  
000660: 9876214567890123456789012345678901234567890123456789012345678901  
000754:  
001052:  
001147:  
001244:  
001341:  
001436:  
001533:  
001630:
```

25

Figure C-1. Error Dump (Continued)



25

Figure C-1. Error Dump (Continued)

Debugging Techniques

```
ARRAY DEFINED ON LINE NUMBER = 13
RUN TIME ARRAY, NO SEQUENCE, ALPHAMERIC, NOT ALTERNATING
ENTRY LENGTH = 1          POINTER TO ARRAY = %003176 (RIGHT BYTE) NUMBER OF ENTRIES = 15

OCTAL DUMP OF T/A CONTROL INFORMATION
003421: 100200 006375 000001 006375 000017 000000 000001 000000
003431: 000000 000015

ASCII DUMP OF ARRAY
003176: ABCDEFGHIJKLMNO

ARRAY DEFINED ON LINE NUMBER = 14
RUN TIME ARRAY, NO SEQUENCE, ALPHAMERIC, NOT ALTERNATING
ENTRY LENGTH = 1          POINTER TO ARRAY = %003206 (LEFT BYTE) NUMBER OF ENTRIES = 75

OCTAL DUMP OF T/A CONTROL INFORMATION
003433: 10 200 006414 000001 006414 000013 000000 000001 000000
003443: 000000 000016

ASCII DUMP OF ARRAY
003206:

ARRAY DEFINED ON LINE NUMBER = 15
RUN TIME ARRAY, NO SEQUENCE, ALPHAMERIC, NOT ALTERNATING
ENTRY LENGTH = 1          POINTER TO ARRAY = %003253 (RIGHT BYTE) NUMBER OF ENTRIES = 15

OCTAL DUMP OF T/A CONTROL INFORMATION
003445: 100200 006527 000001 006527 000017 000000 000001 000000
003455: 000000 000017

ASCII DUMP OF ARRAY
003253: A
```

25

Figure C-1. Error Dump (Continued)

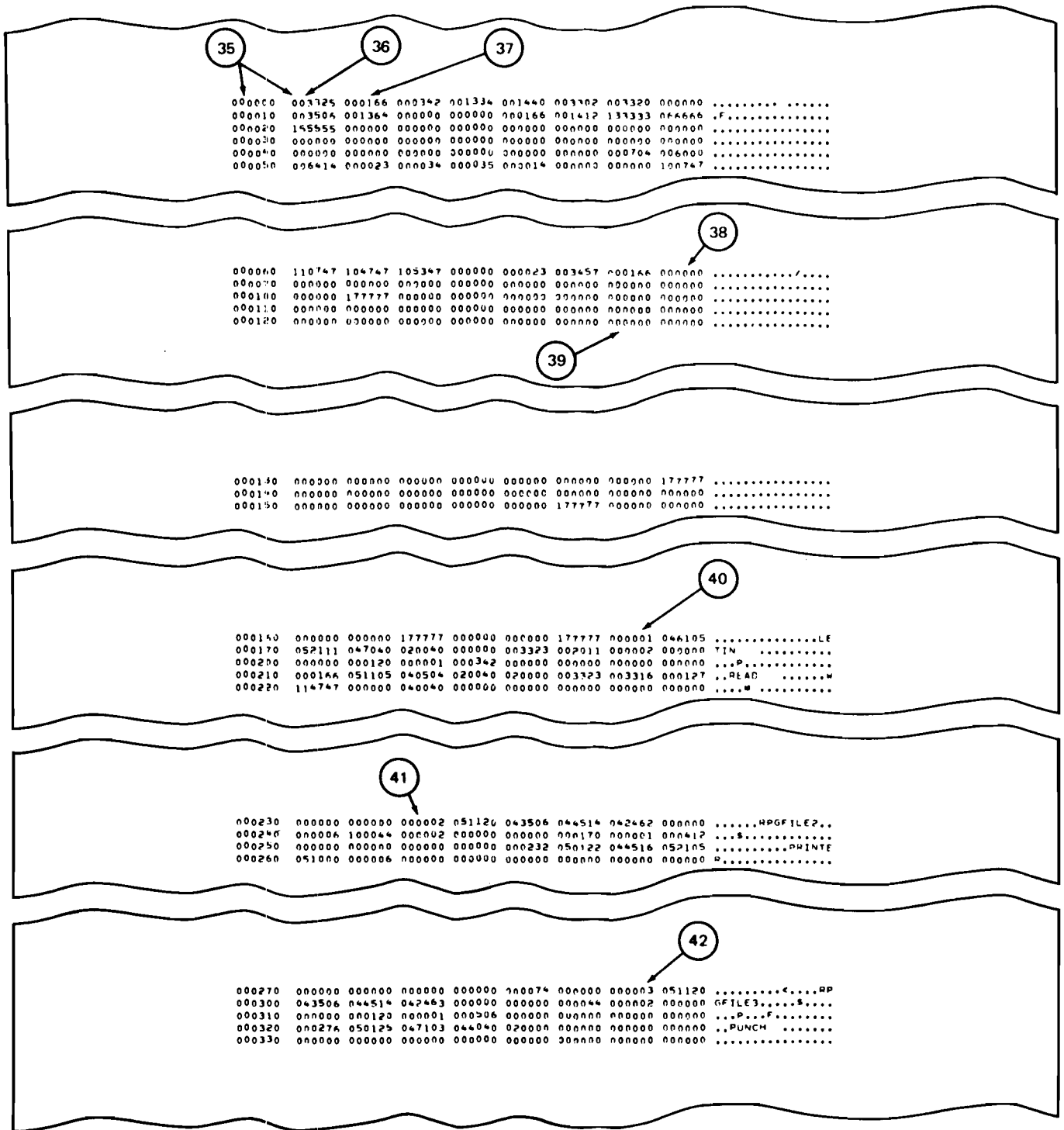


Figure C-1. Error Dump (Continued)


```

002640 020040 020040 020040 020040 020040 020040 020040 020040 020040
002650 020040 020040 020040 020040 020040 020040 020040 020040 020040
002660 020040 020040 020040 020040 020040 020040 020040 020040 020040
002670 020040 020040 020040 020040 020040 020040 020040 020040 020040
002700 020040 020040 020040 020040 020040 020040 020040 020040 020040
002710 020040 020040 020040 020040 020040 020040 020040 020040 020040
002720 020040 020040 020040 020040 020040 020040 020040 020040 020040
002730 020040 020040 020040 020040 020040 020040 020040 020040 020040
002740 020040 020040 020040 020040 020040 020040 020040 020040 020040
002750 020040 020040 020040 020040 020040 020040 020040 020040 020040
002760 020040 020040 020040 020040 020040 020040 020040 020040 020040
002770 020040 020040 020040 020040 020040 020040 020040 020040 020040
003000 020040 020040 020040 020040 020040 020040 020040 020040 020040
003010 020040 020040 020040 020040 020040 020040 020040 020040 020040
003020 020040 020040 020040 020040 020040 020040 020040 020040 020040
003030 020040 020040 020040 020040 020040 020040 020040 020040 020040
003040 020040 020040 020040 020040 020040 020040 020040 020040 020040
003050 020040 020040 020040 020040 020040 020040 020040 020040 020040
003060 020040 020040 020040 020040 020040 020040 020040 020040 020040
003070 020040 020040 020040 020040 020040 020040 020040 020040 020040
003100 020040 020040 020040 020040 020040 020040 020040 020040 020040
003110 020040 020040 020040 020040 020040 020040 020040 020040 020040
003120 020040 020040 020040 020040 020040 020040 020040 020040 020040
003130 020040 020040 020040 020040 020040 020040 020040 020040 020040
003140 020101 041103 042105 043107 044111 045113 046115 047117 048119 049121
003150 030061 031063 032065 033067 034071 035073 036075 037077 038079 039081
003160 045514 046516 047520 048522 049524 050526 051528 052530 053532 054534
003170 022056 026453 057451 058453 059455 060457 061459 062461 063463 064465
003200 042105 043107 044111 045113 046115 047117 020040 020040 020040
003210 020040 020040 020040 020040 020040 020040 020040 020040 020040
003220 020040 020040 020040 020040 020040 020040 020040 020040 020040
003230 020040 020040 020040 020040 020040 020040 020040 020040 020040
003240 020040 020040 020040 020040 020040 020040 020040 020040 020040
003250 020040 020040 020040 020040 020101 020040 020040 020040 020040
003260 020040 020040 020040 020112 040516 052501 051131 020063
003270 030054 020061 034467 032440 020040 020040 020040 020040 020071 0, 1975

```

ARCDEFGHIJKLMNO
012345678901234J
KLMNOPQRSTUUVWXY,
\$,-+.)(>?!"@|AR
C
A
JANUARY 3
9

46

```

003300 034067 033040 000000 000004 000000 000000 000000 000000 000000 000000 876 .....
003310 000000 000000 000000 000000 000000 034470 033466 034470 033465 .....98769875

```

47

48

```

003320 000004 034470 033466 034470 033465 100200 001441 000001 ..98769875...!..
003330 001441 000100 000000 000001 000000 000000 000007 100200 !.#.
003340 001541 000001 001541 000000 000000 000001 000000 000000 .....
003350 000010 100200 006301 000001 006301 000017 000000 000001 .....
003360 000000 000000 000011 100200 006320 000001 006320 000017 .....
003370 000000 000001 000000 000000 000012 100200 006337 000001 .....
003400 006337 000017 000000 000001 000000 000000 000013 100200 .....
003410 00635A 000001 00635B 000017 000000 000001 000000 000000 .....
003420 000014 100200 006375 000001 006375 000017 000000 000001 .....
003430 000000 000000 000015 100200 006414 000001 006414 000013 .....
003440 000000 000001 000000 000000 000016 100200 006527 000001 .....
003450 006527 000017 000000 000001 000000 000000 000017 000011 .....

```

49

50

```

003460 00012A 000127 000130 000131 000132 000133 000134 000135 .V.W.X.Y.Z.(.\)
003470 000113 063777 163762 000000 076400 000000 000000 032001 .K.....
003500 000200 000000 000000 000000 000134 140074 000004 000000 .....
003510 000023 060346 000004 000000 000000 000000 000001 000000 .....
003520 000345 000731 020346 000000 000126 000007 001311 060345 .....
003530 000016 000165 006446 006042 003532 001412 000000 000010 .....
003540 000127 060311 000012 000007 000000 177777 177777 000010 .....
003550 000000 000010 003563 003614 003645 007430 007512 003676 .....
003560 007610 000053 000002 020101 041524 044517 047040 053501 ..... ACTION WA
003570 051440 032440 036440 044015 046505 042111 040024 042440 S 5 = IMMEDIATE

```

Figure C-1. Error Dump (Continued)

```

003600 052105 051115 044516 040924 044517 047040 053511 052110 TERMINATION WITH
003610 020104 052515 050040 020040 020040 020040 020106 044514 DUMP      FIL
003620 042516 040515 042475 046105 052111 047040 020040 026040 ENAME=LEFTIN
003630 020122 042503 047522 042040 047125 046502 042522 020075 RECORD NUMBER =
003640 034040 020040 020040 020040 020040 000005 001154 000001 8
003650 010014 000016 010176 000000 000000 010016 000014 000003 .....
003660 015434 000002 131640 020666 000013 000002 000005 000317 .....!.....
003670 14 063 000024 005147 077777 000132 010000 170000 133014 .....3.....Z.....
003700 072631 177777 100000 010016 022123 052104 046111 051524 .....%STDLIST
003710 020040 000050 005021 061325 000152 000000 000006 000114 .....L

```

ABORT :%OLDPASS...%0.%221GHSL.%4.%50271 PROCFS QUIT P=3

51



Figure C-1. Error Dump (Continued)

Table C-1. File Table Format

Words	Bits	Meaning
0	0:15	File number returned by MPE/3000 when file is opened.
1-4	0:15	File name.
5	0:15	Pointer to File Translation Table (if any).
6	0:15	Current line number for printer output file.
7	0:1	Carriage control bit. (If on, carriage control (skip/space before or after printing) used for output.)
7	1:1	Run-time end-of-file indicator (set on if end-of-file encountered).
7	2:1	Look-Ahead File Indicator (set on if this is a look-ahead file).
7	3:5	(Unused by RPG/3000 at run-time.)
7	8:3	File Type: 0 = Input File 1 = Output File 2 = Update File 3 = Display File 4 = Combined File
7	11:3	File Designation 0 = Secondary input file 1 = Blank 2 = Primary input file 3 = Record address file 4 = Chaining File 5 = Table or Array File 6 = Demand File
7	14:1	KSAM Flag 0 = File opened at run-time is not a KSAM file 1 = File opened at run-time is a KSAM file
7	15:1	End of File Termination 0 = The End of File Field for the File Description Specifications in blank. 1 = The End-of-File Field for the File Description Specifications contains E; the program cannot end until it reads all records from this file.
8	0:2	Sequence of File 0 = The file is not to be sequence-checked. 1 = The file is to be sequence-checked for ascending order. 2 = The file is to be sequence-checked for descending order.
8	2:1	File Format 0 = Fixed Length records. 1 = Variable Length records.
8	3:1	Unused

Table C-1. File Table Format (Continued)

Words	Bits	Meaning
8	4:2	<p>Record Address Type</p> <p>0 = This file is a direct access file not processed through chaining or by a RAF, or is a sequential file.</p> <p>1 = This file is processed through chaining or under direction of a RAF.</p> <p>2 = This is a KSAM, RSAM, or IMAGE file processed by alphanumeric key.</p> <p>3 = This is a KSAM, RSAM, or IMAGE file processed by packed numeric key.</p>
8	6:2	<p>File Extension/Line Counter Specification</p> <p>0 = No File Extension or Line Counter Specifications are defined for this file.</p> <p>1 = File Extension Specifications are defined for this file.</p> <p>2 = Line Counter Specifications are defined for this file.</p>
8	8:4	Number of Disc Extents for the file.
8	12:4	<p>File Organization</p> <p>00 = This is neither an ADDROUT or other direct-access file.</p> <p>10 = This is an ADDROUT file.</p> <p>11 = This is a direct-access file.</p> <p>13 = This is a KSAM file</p> <p>14 = This is an RSAM file.</p> <p>15 = This is an IMAGE file.</p>
9	0:7	Length of key for RAF or for IMAGE/3000 file, in bytes.
9	7:4	Overflow Indicator for the File (if this is a printer file).
9	11:1	<p>File Addition</p> <p>1 = Add new information to the end of the file.</p> <p>0 = Add new information beginning at the start of the file.</p>
9	12:2	<p>Processing Mode</p> <p>0 = This is a sequential file.</p> <p>1 = This is a random-access file.</p> <p>2 = This is an indexed file processed sequentially between limits.</p>
9	15:1	<p>Special File Flag</p> <p>0 = This is not a SPECIAL file.</p> <p>1 = This is a SPECIAL file.</p>
10	0:8	<p>File Conditioner</p> <p>0 = This file can be used unconditionally.</p> <p>Other settings = The user indicator represented conditions the use of this file.</p>

Table C-1. File Table Format (Continued)

Words	Bits	Meaning
10	8:4	Not used at run time.
10	12:4	Number of User Labels Processed 0 = Standard Label only. 1 - 9 = Standard Label plus from one to nine user labels.
11	0:15	Record Length (in bytes).
12	0:15	Blocking Factor (Logical Records per block).
13	0:15	File Buffer Address (First word of buffer).
14	0:15	Pointer to Label Exit Name or SPECIAL file processing routine (used by compiler for calling subroutines).
15	0:15	Pointer to Routine for handling errors (used by compiler).
16	0:15	Pointer to Bypass Field Name (used for the Bypass option on the File Description Specification continuation record; points to numeric counter of number of file errors for the file.)
17	0:15	Pointer to RDEXIT Routine Name.
18	0:15	Unused.
19 thru 22	0:15	Device Type Name
23 thru 34	0:15	(Depends upon type of file, as indicated below.)

For Printer File Defined by Line Number Option of Line Counter Specification

Words	Bits	Meaning
23	0:15	Form Length
24	0:15	Overflow Line Number

For Printer File Defined by Channel Number Option of Line Counter Specification

Words	Bits	Meaning
23 thru 34	0:15	Line Number associated with each channel number. (Word 23 contains Line Number of Channel 1, Word 24 contains Line Number of Channel 2, and so forth.)

Table C-1. File Table Format (Continued)

For Input File with Matching Field

Words	Bits	Meaning
23	0:15	Matching Field Table Pointer.

For Input File with Control Field

Words	Bits	Meaning
24	0:15	Control Field Table Pointer.

For All Other Files

Words	Bits	Meaning
25	0:15	Record Identifying Indicator Table
26	0:15	(Pointer used internally by compiler for moving input fields); or Pointer to RAF Continuation Field.
27	0:15	Displacement (in bytes) to Trailer Field on Spread Record.
28	0:15	Matching or Control Level Field in Current Record For Control Level: Bit 1 = 1 means Level 1 is present. Bit 2 = 1 means Level 2 is present. Bit 3 = 1 means Level 3 is present. Bit 4 = 1 means Level 4 is present. Bit 5 = 1 means Level 5 is present. Bit 6 = 1 means Level 6 is present. Bit 7 = 1 means Level 7 is present. Bit 8 = 1 means Level 8 is present. Bit 9 = 1 means Level 9 is present. For Matching Field: Bit 10 = 1 means Match Fields are present.
29	0:15	Sequence Number of Current Record
30	0:15	Trailer Index for Current Spread Record
31	0:15	Trailer Length for Current Spread Record
32	0:15	Pointer to RAF Chaining Table
33-34	0:15	Searching Key for Random-Access File

Table C-1. File Table Format (Continued)

Words	Bits	Meaning
35	0:15	Pointer to IMAGE/3000 Table (see Figure C-2).
36	0:15	Word for Controlling Card Reader/Punch/Interpreter, or Pointer to Skip/Space Options.

Table C-2. IMAGE/3000 Table Format

Words	Bits	Meaning
0-3	0:15	IMAGE/3000 Data Base Name
4	0:8	Open Mode 1 = Read/Write Shared mode. 2 = Update Shared mode. 3 = Exclusive mode.
4	8:8	I/O Mode 2 = Serial read. 3 = Backward serial read. 4 = Directed read. 5 = Chained read. 6 = Backward chained read. 7 = Associative read. 8 = Primary associative read. 9 = Sequential read. 10 = Backward sequential read. 11 = Chained sequential read. 12 = Backward chained sequential read.
5-8	0:15	Password (Level) Identification (8 bytes)
9	0:15	Initial Chain Key (ITEMXX key)
10-17	0:15	First Item (Key) Name (16 bytes)
18-25	0:15	Second Item (Key) Name (16 bytes) (ITEMXX Name)
26-33	0:15	Data Set Name (16 bytes)
34-43	0:15	IMAGE/3000 Communication Area (10 Status Words)
44	0:15	User Status Array Pointer. (Points to a six-element array where IMAGE/3000 status is stored.)
45	0:15	IMAGE/3000 Key

Table C-3. Chaining Table Format

Words	Bits	Meaning
0	0:15	TO FILE Control Table Address
1	0:15	Chaining Field Byte Address
2	0:15	Chaining Field Length (in bytes)
3	0:15	(Used by compiler for branching to conversion routine.)
4	0:15	Number of Line where Chaining Table Defined

Table C-4. Table/Array Control Table Format

Words	Bits	Meaning
0	0:2	Table/Array Descriptor 2 = Array 3 = Table
0	2:0	Table/Array Sequence 0 = No sequence applies. 1 = Ascending sequence. 2 = Descending sequence.
0	4:3	Data Format (on Input and/or Ouput) 0 = Alphanumeric Data. 1 = Packed Decimal. 2 = Binary. 3 = ASCII format with leading sign. 4 = ASCII format with trailing sign. 5 = Unpacked Decimal.
0	7:1	Alternating Table Flag 0 = No alternating table associated with this table. 1 = Alternating table associated with this table.
0	8:2	Table/Array Type 0 = Compile Time Table/Array 1 = Pre-Execution Time Table/Array 2 = Execution Time Array
1	0:15	Pointer to Last Item Found in LOKUP Operation.

Table C-4. Table/Array Control Table Format (Continued)

Word	Bits	Meaning
2	0:15	Length of Entry in Table/Array (in bytes or digits).
3	0:15	Pointer to Table/Array
4	0:15	Number of Entries in Table/Array
5	0:15	Number of Entries per Record
6	0:15	Length of Each Entry on Input or Output (in bytes).
7	0:15	Pointer to Input File Table
8	0:15	Pointer to Output File Table
9	0:15	Number of Line where Table/Array is Defined.

OBJECT PROGRAM OPERATING CYCLE

APPENDIX

D

Every RPG/3000 object program executes the same general sequence of operations each time it processes a record. This sequence is called the *RPG/3000 Object Program Operating Cycle*. Although you do not actually observe this cycle, knowledge of its details will help you organize source code and debug programs more effectively. The cycle is summarized in the discussion below, and shown in detail in the *Object Program Flow Chart* presented in Appendix E.

Before the object program begins the cycle, it performs various initialization and housekeeping operations (described under Step 1 in the discussion below). It then enters the cycle (Steps 2 through 8), repeating it once for every record processed. This process is illustrated in Figure D-1.

Step	Action
1	<p><i>Pre-Cycle Initialization and Housekeeping.</i> The object program performs certain preliminary functions that are done only once, before the cycle actually begins. During these operations, the program opens (prepares for processing) all files to be used and reads all pre-execution time tables and arrays into memory. It aligns the first page for any printer files and writes all first-page headings (those records conditioned by the IP indicator). The program next reads a record from each file used for input and identifies its type. Now, the program is ready to begin the cycle.</p>
2	<p><i>Heading and Detail-Time Output.</i> The program begins the cycle. It outputs all other heading and detail records whose conditions are satisfied. (Heading Records are indicated by an H in the <i>Type Field</i> of the Output Specifications; detail records are indicated by a D in the <i>Type Field</i>.) The program tests the status of all halt indicators (and halts if any are on), and turns off all record-identifying and control-level indicators that are on at this point. It turns off any overflow indicators that were on before the last detail calculation (done in Step 8). It also tests the LR indicator for end-of-program status (and transfers to Step 6 if this indicator is on).</p> <p>NOTE: The term <i>Detail-Time Operations</i> used throughout this manual collectively refers to detail-time output (occurring in this step) and detail-time calculations (occurring in Step 8).</p>
3	<p><i>Input Record Operation.</i> If this is not the first pass through the cycle, the program reads the next input record from the last file processed, and identifies its type, checking it for an end-of-file indication and proper sequence. (The program skips this step on the first pass through the cycle.)</p>
4	<p><i>Record Selection.</i> If the program uses only one file for input, then the next record from that file is selected for processing. But, if the program uses more than one file for input, then the record to be processed is selected as follows:</p> <ol style="list-style-type: none">If a file is forced, the next record from that file is selected.If one or more records without matching fields have been read, the highest-priority record among them is chosen; primary file records have greatest priority, followed by secondary file records in the order that they were specified in the RPG/3000 source program.If all input records read have matching fields, then the lowest-sequenced record is chosen (if the merged file is in ascending sequence) or the highest-sequenced record is used (if the merged file is in descending sequence).
5	<p><i>Control Break Test.</i> The program tests for a control break, and turns on the appropriate control-level indicator (and all lower-level indicators) if a break occurs.</p>

Object Program Operating Cycle

Step	Action
6	<p><i>Total Time Operations.</i> If this is the first record with control-level fields, the program skips total-time operations. But, if this is a second or later record with control-level fields, the program performs total time calculations and output (including total-time overflow output). (Total-time operations are all those calculations with L0-L9 or LR indicator entries in the <i>Control Level Field</i> (Columns 7 through 8) and output operations of record type T.) These operations are done <i>after</i> each control break occurs or the last input record is read, but <i>before</i> the information on the input record selected in Step 4 is actually made available for processing. The program also sets all resulting-indicators as specified. If the last-record indicator is on, the program terminates.</p> <p><i>Data Movement.</i> The program moves the data from the record selected into work areas for processing. If this is a matching record, it sets the matching record indicator. If this is a look-ahead record, the program reads the look-ahead fields and moves them into work areas.</p>
8	<p><i>Detail-Time Calculations.</i> The program does detail-time calculations for the record specified, and returns to Step 2. (Detail-time calculations are those calculations <i>not</i> conditioned by control-level indicators in the <i>Control Level Field</i> (Columns 7 through 8).) Notice that these calculations and the detail-time output described in Step 2 are both done <i>after</i> information from the record selected in Step 4 becomes available; they are usually based on information from this record. At this point, the program also turns on resulting indicators influenced by these calculations.</p>

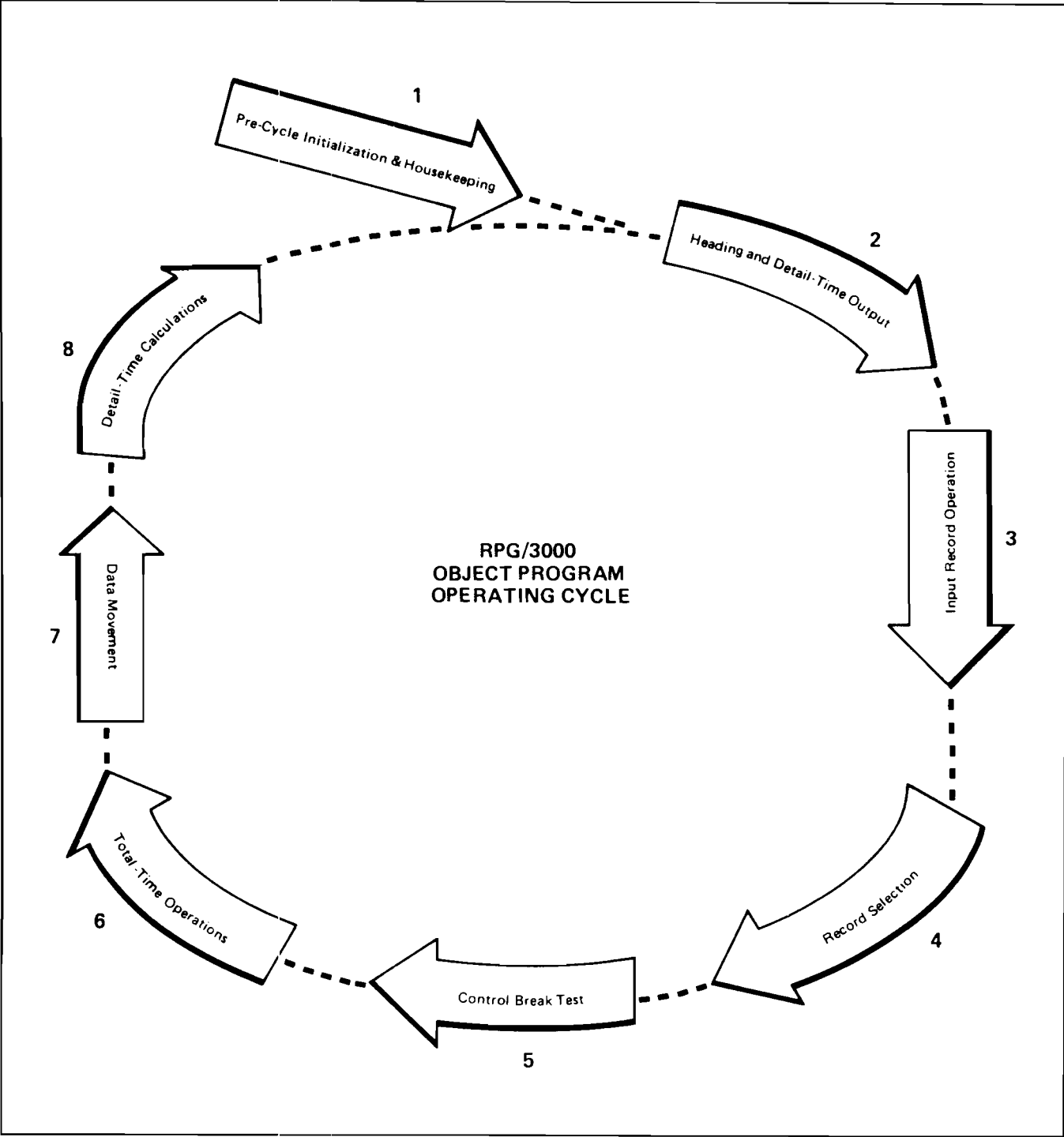


Figure D-1. RPG/3000 Object Program Operating Cycle



OBJECT PROGRAM LOGIC FLOW CHART

APPENDIX

E

The specific details of the RPG/3000 object program logic are discussed in this appendix and illustrated in the Object Program Logic Flow Chart (Figure E-1). The step numbers that appear in the text below are keyed to those in the flow chart and to the generalized discussion in the preceding appendix.

Step	Action
1	The object program initializes all data areas for execution and performs certain house-keeping functions. It sets the external user indicators as specified, initializes the user date (UPDATE) fields, opens all files, reads pre-execution time tables and arrays into memory, aligns the first page for print files, and writes first-page heading records to these files. The program then reads a record from each file to be used for input, and identifies the record type and moves the look-ahead field for each. These operations are performed only once for every program, and are not part of the actual cycle.
2A	The object program now begins the program cycle, writing out all other heading lines and detail lines whose conditions are satisfied. If fetch overflow is specified and the overflow indicator is on, the program writes overflow lines. If the first-page indicator (1P) is on, the program turns it off.
2B	The program checks the status of the halt indicators (H0 through H9).
2C	If any halt indicators are on, the program acts as directed in Column 69 of the Control Record Specification if a pre-programmed response is taken; otherwise, a message is sent to the computer operator, who determines whether or not to continue. (These halt indicators can be set at any time during the program; you must set them off before this step if you do not want your program to halt.) If no halt indicators are on, the program transfers to Step 2E.
2D	The program either continues or terminates at this point (at the discretion of yourself or the operator). If you have requested termination at this point, it terminates. Otherwise, it sets the halt indicator off and returns to Step 2B to determine if other halt indicators are on.
2E	The program sets off all recording-identifying, first-page, and control-level indicators L1 through L9. It also sets off overflow indicators unless they were set on during preceding detail-time calculations or output. It also sets the L0 indicator on. ALL OTHER INDICATORS PRESENTLY ON REMAIN ON.
2F	The program tests the status of the last-record (LR) indicator.
2G	If the LR indicator is on, the program also turns on the lower-level indicators (L1 through L9) and transfers to Step 6A. Otherwise, the program proceeds to Step 3A.

Object Program Logic Flow Chart

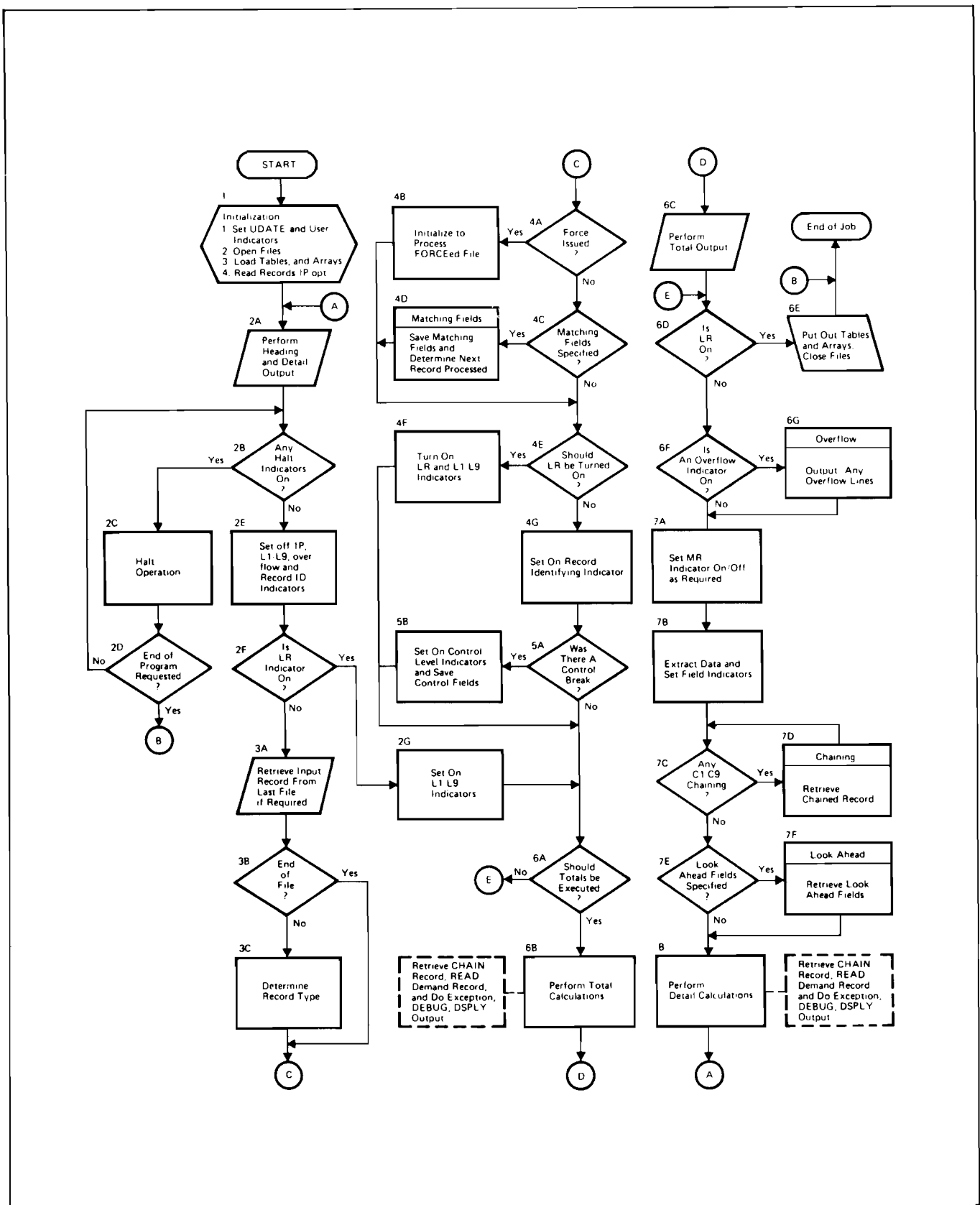


Figure E-1. Object Program Logic Flow Chart

Step	Action
3A	The program reads a record from the last file processed. If this file is retrieved by a RAF, the data in the RAF defines the record to be selected. If look-ahead fields were specified in the last record processed, the record is already in storage and no read is required at this time. If the last record was a spread record, the program does not necessarily read a new record; instead, the next trailer portion of the spread record is combined with the header portion to form a new spread record, unless all trailer portions of the current record have been processed. (If all trailer portions have been processed, however, the program reads a new record.)
3B	The program checks to determine whether an end-of-file indication occurred on the file just read. If so, the program skips to Step 4A.
3C	If a record was read from the file (Step 3A), the program determines the record type and checks for the correct record sequence (as specified in the Group Sequence Field of the Input Specifications.)
4A	The program tests to determine whether a FORCE operation was executed on the previous pass through the cycle. If it was not, the program skips to Step 4C.
4B	If a FORCE operation was encountered on the previous cycle, the file is selected for processing. If the record contained matching fields, they are ignored. If the program is at the end of the forced file, it uses the normal multi-file logic to select the next record for processing. The program transfers to Step 4E.
4C	The program tests to determine if matching fields are specified. If so, the program proceeds to Step 4D. Otherwise, it transfers to Step 4E.
4D	If matching fields were specified, the program extracts the matching fields and performs sequence checking. If these fields are not in sequence, the program either requests the operator to determine how the error should be handled, or takes any pre-programmed option requested. If Option 2, 3, 4, or 5 is selected, the program terminates; if Option 1 is selected, the program transfers to Step 2B.
4E	The program checks to determine if the LR indicator should be turned on. (This indicator should be on when all records from all files described with an E in the End-of-File Field of the File Description Specifications, and all matching secondary records, have been processed.) If the indicator should not be on, the program skips to Step 4G.
4F	The program turns on the LR indicator (and sets on indicators L1 through L9), and skips to Step 6A. These indicators can only be used for conditioning if they have been defined in another specification.
4G	The program sets on the record-identifying indicator for the record selected for processing.
5A	The program tests to determine if the record selected caused a control break. (A control break occurs when the value in the control fields of the record being processed differs from the value of the control fields of the last record processed.) If no control break occurred, the program skips to Step 6A.
5B	When a control break occurs, the program sets the appropriate control level indicator (and all lower-level indicators) on.

Object Program Logic Flow Chart

Step	Action
6A	If this is the first record with control level fields to be processed, the program continues to Step 6D; otherwise, the program transfers to Step 6B.
6B	Total calculations are performed at this time. Resulting indicators are set on or off at this point.
6C	Total output lines whose conditioning indicators are satisfied are written out at this point. If fetch overflow is specified and the overflow indicator is on, the program writes the overflow lines.
6D	The program checks the status of the LR indicator. If this indicator is not on, the program transfers to Step 6F.
6E	If the LR indicator is on, the program writes any table or array specified for output; closes all open files; passes the status of the user indicators (U1 through U8) to the job control word (which may be used in the next program within the job); and terminates.
6F	The program tests to determine if an overflow indicator is on. If none are on, the program skips to Step 7A.
6G	The program outputs any overflow lines conditioned by overflow indicators that have not already been executed with fetch overflow logic.
7A	If matching records are used, the program sets the MR indicator on; when the records match, the MR indicator remains on for the complete cycle during which the matching record is processed. The Matching Record Routine invoked at this point proceeds as follows:
7A-1	The routine tests to determine if multfile processing is in progress. If only one file is being input, the routine skips to Step 7A-3.
7A-2	The values of the match fields presently in the matching field hold area are tested to determine which file is to be processed next. (The program finds the file with the lowest (ascending sequence) or highest (descending sequence) matching field.)
7A-3	The routine tests to determine whether the matching fields are in the correct sequence.
7A-4	If the sequence is incorrect, the routine requests the operator to determine whether to bypass the current record or terminate the program, or takes any pre-programmed response requested.
7A-5	The match fields are moved to the master hold area for the file being processed.
7B	The program extracts the data fields from the record selected for processing, and moves them into work areas for processing.
7C	The program checks to determine if any fields in the record being processed are specified with the C1 through C9 chaining codes. If any of these codes are specified, the program proceeds to Step 7D. Otherwise, it skips to Step 7E.

Step	Action
7D	If chaining codes were specified, the program retrieves the selected record, sets the appropriate record-identifying indicator, extracts the input fields, and sets the corresponding field indicators. The program then returns to Step 7C to test for additional chaining code fields, continuing on when no more are found.
7E	The program tests for look-ahead fields. If they are present, it proceeds to Step 7F; otherwise, it branches to Step 8.
7F	The program retrieves the look-ahead record and extracts from it the look-ahead fields. (For combined and update files, the look-ahead fields are extracted from the record currently being processed; for all other files, the look-ahead fields are extracted from the next record in the file currently being processed.)
8	The program executes all conditioned detail calculations and subroutines, turns on any appropriate resulting indicators, and returns to Step 2A.



SPECIFICATION SHEET SUMMARIES

APPENDIX

F

If you are already familiar with RPC/3000, this appendix can provide you with a quick reference to the entries allowed in each field on each specification sheet. For a detailed discussion of the entries, please see Section II through IX. (In the discussion below, the notation (REQUIRED) denotes a required entry.)

CONTROL RECORD SPECIFICATION

The Control Record Specification entries are:

SEQUENCE NUMBER (COLUMNS 1 THROUGH 5)

Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

FORM TYPE (COLUMN 6) (REQUIRED)

Entry	Meaning
H	Control Record (Header) Specification.

DEBUG (COLUMN 15)

Entry	Meaning
1	Execute all DEBUG operations, as explained below.
blank	Suppress all DEBUG operations; the compiler will treat records requesting these operations as comment records.

LINE NUMBER OPTION (COLUMN 20)

Entry	Meaning
N	Disable line number option.
blank	Enable line number option.

INVERTED PRINT (COLUMN 21)

Entry	Meaning
blank	Domestic Format.
I	European Format.
J	European Format with leading zero.
D	United Kingdom Format.

ALTERNATE COLLATING SEQUENCE (COLUMN 26)

Entry	Meaning
S	Alternate collating sequence, specified in hexadecimal format, applies.
O	Alternate collating sequence, specified in octal format, applies.
E	EBCDIC collating sequence applies.
K	EBCDIK collating sequence applies.
blank	Normal ASCII Collating Sequence applies.

TABLE/ARRAY LOOK-UP (COLUMN 34)

Entry	Meaning
blank	Sequential look-up applies.
B	Binary look-up applies.

CONTROL RECORD SPECIFICATION

SIGN PROCESS (COLUMN 40)

Entry	Meaning
N, B, or blank	Sign-forcing does not take place during input of unpacked numeric data or during output of any data.
S	Standard sign-forcing occurs. Thus, signs are forced for all numeric input (including tables and arrays); all alphanumeric fields moved to numeric fields with the MOVE, MOVEL, MLLZO, MHHZO, MLHZO, or MHLZO operations (discussed in Section VIII); and all packed numeric output fields. When moving a numeric field to an alphanumeric field, or unpacking a field on output, the object program changes all plus signs to an ASCII zone representation (the binary value 0011).
I	Sign-forcing does not occur on input of unpacked numeric data.
O	Sign-forcing does not occur on output of any data.

FORM POSITIONING (COLUMN 41)

Entry	Meaning
1	Program repeats a forms-alignment record on the printer until operator verifies correct form position.
blank	Form position is not verified.

INDICATOR SETTING (COLUMN 42)

Entry	Meaning
S	Blank/Zero Indicators are set on at program initialization and following Blank/After operations.
T	Blank/Zero Indicators are set on at program initialization, but not following Blank/After operations.
B	Blank/Zero Indicators are not set on at program initialization nor following Blank/After operations.

blank Blank/Zero Indicators are not set on at program initialization; they are set on, however, following Blank/After operations.

FILE TRANSLATION (COLUMN 43)

Entry	Meaning
F	Translate characters according to file translation table specified in hexadecimal format.
O	Translate characters according to file translation table specified in octal format.
blank	Translation is not required.

DOLLAR-SIGN SUBSTITUTE (COLUMN 46)

Entry	Meaning
Any special character (not a letter or digit).	Treat this character as a dollar sign when encountered in edit words and codes.
blank	Do not substitute any character for the dollar sign.

SKIP-SUPPRESS (COLUMN 47)

Entry	Meaning
S	Suppress skip so that output begins at current position of paper in printer.
blank	Skip to a new page.

CROSS-REFERENCE LISTING (COLUMN 52)

Entry	Meaning
X	Print a Cross-Reference Listing.
blank	Do not print a Cross-Reference Listing.

**CARRIAGE CONTROL TYPE
(COLUMN 53)**

Entry	Meaning
blank	Skip requests refer to channel numbers on carriage-control tape, with channel alignment specified in Line Counter Specifications.
1	Skip requests refer to channel numbers on carriage-control tape, with channel-alignment simulating IBM System/3 alignment: carriage-control tape-channel 1 is equated with line 1 on the printed page, and line counting begins at line 1.
L	Skip requests refer to actual line numbers.

**TESTFILE SEQUENCE CHECK
(COLUMN 54)**

Entry	Meaning
S	Check all records in textfile for proper sequence, based on the sequence numbers appearing in Columns 1-5 of these records. Print a message if a sequence error occurs.
N or blank	Do not perform sequence checking.

**ERROR LOG
(COLUMN 55)**

Entry	Meaning
blank	When an error occurs and a pre-selected response is specified in the Error Response Field, (described next), send the message to the standard list device. (This device is typically a printer for a batch job, or a terminal for a session.)
N	When an error occurs and a pre-selected response is specified in the Error Response Field, do not output an error message.
S	When an error occurs, whether or not a pre-selected response is specified in the Error Response Field, send an error message to the operator's console (in a job) or terminal (in a session), print an error dump (described in Section XII), and terminate the program. Ignore the Error Response Field.

**ERROR RESPONSE
(COLUMNS 56 THROUGH 71)**

Entry	Meaning
0	Re-direct or suppress error message (in accordance with the Error Log Field) and continue program execution in-line.
1	Re-direct or suppress error message, skip the erroneous input record, and continue execution.
2	Re-direct or suppress error message, and terminate program (by executing normal termination code). This procedure turns on all level indicators (L1 through L9, and LR, as defined in Section VI), calculates all totals, prints all totals and table output, and closes all files.
3	Re-direct or suppress error message, close all files, and terminate program immediately. (Do not turn on level indicators, calculate totals, or print total and table output.)
4	Re-direct or suppress error message, print an error dump, and terminate program. This includes turning on all level indicators, calculating all totals, printing all totals, and closing all files.
5	Re-direct or suppress error message, print an error dump, and terminate program immediately. (Do not turn on level indicators, calculate totals, or print total and table output.)
blank	Transmit error message to operator—no pre-selected response is requested.

**PROGRAM NAME
(COLUMNS 75 THROUGH 80)**

Entry	Meaning
Valid program name. (Program names can contain up to six characters, beginning with one of the letters A through Z. The remaining characters can be letters, any of the digits 0 through 9 or an apostrophe ('). Embedded blanks are not allowed in the name.)	Name assigned to object program, also appearing on Source Program Listing.
blank	Name RPGOBJ is assigned to object program, and also appears on Source Program Listing.

FILE DESCRIPTION SPECIFICATIONS

FILE DESCRIPTION SPECIFICATIONS

The File Description Specifications entries are:

SEQUENCE NUMBER (COLUMNS 1 THROUGH 5)

Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

FORM TYPE (COLUMN 6) (REQUIRED)

Entry	Meaning
F	File Description Specification.

FILE NAME (COLUMNS 7 THROUGH 14) (REQUIRED)

Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.) For an IMAGE file containing a data set, this entry indicates a data set name.	File name.

FILE TYPE (COLUMN 15) (REQUIRED)

Entry	Meaning
I	Input file.
O	Output file.
U	Update file.
D	Display file.
C	Combined file.

FILE DESIGNATION (COLUMN 16)

Entry	Meaning
P	Primary file.
S	Secondary file.
R	Record address file (input only).
C	Chained file.
T	Pre-execution table/array file (input only).
D	Demand file.
blank	Sequential output or display file.

END OF FILE (COLUMN 17)

Entry	Meaning
E	The program cannot end until it reads all records from this file.
blank	If this field contains E for any other files, the program can end whether or not it reads all records from this file. If this field is blank for all input, update, or combined files, the program cannot end until it reads all records from all of these files.

F-27. INPUT SEQUENCE CHECK (COLUMN 18)

Entry	Meaning
A or blank	If this is a matching file, check the records for proper ascending order.
D	If this is a matching file, check the records for proper descending order.

F-28. RECORD FORMAT (COLUMN 19) (REQUIRED)

Entry	Meaning
F	Fixed-length record format.
V	Variable-length record format.

**BLOCK LENGTH
(COLUMNS 20 THROUGH 23)**

Entry	Meaning
1 through 9999 (ending in Column 23).	The length of the block (physical record) in characters.
blank	No blocking; block and logical record lengths are identical.

**LOGICAL RECORD LENGTH
(COLUMNS 24 THROUGH 27) (REQUIRED)**

Entry	Meaning
1 through 9999 (ending in Column 27).	For files containing fixed-length records, the length of each record. For files with variable-length records, the length of the longest record in the file.

**PROCESSING MODE
(COLUMN 28)**

Entry	Meaning
blank	Sequential processing for entire file.
R	Random processing. (by record number or key).
L	Sequential processing between limits.

**RECORD ADDRESS FIELD LENGTH
(COLUMNS 29 THROUGH 30)**

Entry	Meaning
1 through 99, (ending in Column 30).	For an RAF, length (in characters) of the relative record number or key fields in the RAF. For ADDR. OUT files, this is always 4. For a KSAM, INDEX, or IMAGE file, length of key fields; maximum key length is 99.
blank	This is not a RAF, KSAM, INDEX, or IMAGE File.

**RECORD ADDRESS TYPE
(COLUMN 31)**

Entry	Meaning
A	This file is a KSAM, INDEX, or IMAGE file processed by alphanumeric keys. (When a numeric field is used as the key, the field is unpacked prior to chaining.)

P This file is a KSAM, INDEX, or IMAGE file processed by packed decimal numeric keys. Split chaining (described in Section VII) is not allowed with such keys.

Note:
For an IMAGE file, the sign of all numeric input fields is forced to C or D. All numeric result fields specified by Calculation Specifications have a C or D sign. Thus, if the sign of the key field in the IMAGE file is not C or D, a record-not-found error occurs when chaining to that field is specified.

K This file is a direct- (random) access file processed by using record keys. (If K is specified for a KSAM, INDEX, or IMAGE file, the file is processed using packed decimal numeric keys; the system assumes an entry of P.)

I This file is a direct-access KSAM, INDEX, or IMAGE file processed by relative record number through chaining or under direction of an RAF.

blank This file is a sequential file; or if Column 28 contains R, this is a random file processed by relative record number.

**FILE ORGANIZATION/ADDITIONAL I/O AREA
(COLUMN 32)**

Entry	Meaning
I	This is a KSAM file.
X	
S	This program will use a special interface to user-supplied routines for handling indexed-sequential access-method files, or to INDEX software optionally supplied by HP.
T	This is an ADDR. OUT file with two input/output buffers.
1 through 7	This is a non-ADDR. OUT random-access or sequential file; assign the specified number of buffers (a through 7) for the file. Accepted (but not used) for IMAGE files.
blank	This is a non-ADDR. OUT random-access or sequential file; assign two buffers for the file.
D	This is a random-access (direct) file; assign two buffers for the file. (This entry is not required to specify direct files, and is available only for compatibility with other systems.)
C	This is a KSAM input only file. The records are read chronologically.

FILE DESCRIPTION SPECIFICATIONS

OVERFLOW INDICATOR (COLUMNS 33 THROUGH 34)

Entry	Meaning
OA	Assign the indicator named.
OB	
OC	
OD	
OE	
OF	
OG	
OV	
blank	Assign no indicator.

KEY FIELD STARTING LOCATION (COLUMN 38)

Entry	Meaning
1 through 9999	Starting position of key field.
blank	No key field applies.
	Note: This entry is not required for IMAGE or INDEX files; if present, it is accepted but not used.

EXTENSION CODE (COLUMN 39)

Entry	Meaning
E	File Extension Specifications further describe the file.
L	Line Counter Specifications further describe the file.
blank	Neither File Extension nor Line Counter Specifications apply to this file.

DEVICE CLASS NAME (COLUMNS 40 THROUGH 46)

Device identifier. (This can consist of up to seven letters, digits, and/or special characters.)	Class name or logical device number of device on which the file resides.

SPECIAL This is a file on a device that requires a user-defined external subroutine to handle input/output. You supply the name of this routine in the Name of Label Exit Field (Columns 54 through 59), and also furnish the routine itself.

\$STDIN A special device class; if an input file is declared on \$STDIN, it will be opened as \$STDIN.

\$STDLIST A special device class; if an output or display file is declared on \$STDLIST, it will be opened as \$STDLIST.

WORKSTN A special device class file that provides an interface between RPG and VPLUS or the RPG Screen Interface.

INTERFACE TYPE (COLUMN 47)

Entry	Meaning
R	This is an RPG Screen Interface WORKSTN file.
C	This is an RPG Screen Interface console file.
blank	This is an RPG/VPLUS Interface WORKSTN file.
V	(Reserved for future use — do not use this option.)

INTERFACE CONTROL (COLUMNS 48 THROUGH 52)

These fields are reserved for entry of special options that are unique to the WORKSTN Interface in use.

DISC LABELS/CONTINUATION CODE (COLUMN 53)

Entry	Meaning
S or blank	Process standard labels.
E	Process standard labels and either call a user-supplied subroutine to process one user label, or bypass this label if it exists.
2 through 9	Process standard labels and either call a user-supplied subroutine to process the number of user labels specified by this entry or bypass this many labels if they exist.
K	Continuation record.

OPTION TYPE
(COLUMNS 54 THROUGH 59)

Entry	Meaning
Subroutine name. (This may be a name of up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	The name of the external subroutine that will process the labels or SPECIAL files.
blank	The program processes no user labels or SPECIAL files.
ERROR	When an input/output error occurs, transfer control to a user-defined external routine that handles the error. (The name of the routine must appear in the Option Target Field.)
BYPASS	When an input/output error occurs, bypass the current logical record and increment a special counter by one. (The counter is a five-digit numeric field containing no decimal positions; its name must appear in the Option Target Field.)
RDEXIT	Whenever the object program reads a record, pass control to an external routine that determines whether the record is to be bypassed. (The name of this routine must appear in the Option Target Field.)

FILE DESCRIPTION SPECIFICATIONS

ASCII The file is in ASCII format. (RPG/3000 does not require this entry for ASCII files but allows it so that programs written for other RPG compilers that do use it can be compiled on the HP 3000.)

EBCDIC The file is in EBCDIC format. If it is an input file, the object program reads the input and RPG/3000 translates it to ASCII code prior to processing. If it is an output file, the object program creates the output and RPG/3000 translates the output to EBCDIC code before it is written to the file.

EBCDIK The file is in EBCDIK format. If it is an input file, the object program reads the input and RPG/3000 translates it to JIS code prior to processing. If the file is an output file, the object program creates the output and RPG/3000 translates the output to EBCDIK code before it is written to the file.

PARTR When you are entering a user-defined file translation table that contains packed decimal or binary fields, those fields are not translated - a partial file translation occurs. For EBCDIC file input, use the EBCDIC entry above.

LOCK Lock this file during an input or output operation. (Does not apply to INDEX or IMAGE files.)

NOLOCK Opens this file so that other concurrent users of the file can specify LOCK; however, this option will not dynamically lock and unlock the file.

WARNING

Use of the NOLOCK feature may invalidate the integrity of data read by your RPG program.

DSNAME This is a DSNAME record for file-sharing.
Note:
For IMAGE files, other entries may appear in this field. See Section IV.

STATUS This record specifies the RPG Screen Interface or IMAGE Data Base status array name.

FORMS This record specifies the RPG Screen Interface or RPG/VPLUS Interface form library file name.

BATCH This record specifies the VPLUS batch file name.

TRACE This record specifies the RPG/VPLUS Interface trace file name.

TRMID This record specifies the name of the two-character field to contain the terminal identification of the RPG Screen Interface WORKSTN file.

START This record specifies the name of the two-digit field which will contain the starting line number for RPG Screen Interface forms which have a variable starting line number.

blank No options are selected.

OPTION TARGET (COLUMNS 60 THROUGH 65)

Entry	Meaning
Subroutine name. (This may be a name of up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	If ERROR appears in the Option Type Field, the Option Target Field contains the name of the subroutine that handles the error processing. If RDEXIT appears in the Option Type Field, the Option Target Field contains the name of the subroutine that determines if the input record is to be bypassed.
Field name. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters or digits.)	If BYPASS appears in the Option Type Field, the Option Target Field contains the name of the field to be used for the error counter.
P (in Column 60)	If EBCDIC appears in the Option Type Field, a partial field translation is accomplished - packed decimal and binary fields on input and/or output are not translated.
blank	No option applies.
Qualified file name. (This can contain from one to 15 characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	If DSNAME appears in the Option Type Field, override the file name specified in Columns 7 through 14 for the preceding file specification when opening the file; share the file number (assigned by MPE) with all files with the same DSNAME file name. Thus, all references to this file number, by whatever file name, access the same logical file. Note: For IMAGE files, other entries may appear in this field. See Section IV.

**FILE ADDITION
(COLUMN 66)**

Entry	Meaning
A	Append new records at end of sequential file, or add new records to KSAM, INDEX, or IMAGE file.
blank	Write new records at beginning of sequential file; do not add records to KSAM, INDEX, or IMAGE input or update file.

**EXTENTS
(COLUMNS 68 THROUGH 69)**

Entry	Meaning
1 through 32.	Allow the specified number of disc extents.
blank	Allow eight disc extents.

**FILE CONDITIONER
(COLUMNS 71 THROUGH 72)**

Entry	Meaning
U1 through U8	Use this file only when the indicator named is set on.
blank	Use this file unconditionally.

FILE EXTENSION SPECIFICATIONS

**PROGRAM NAME
(COLUMNS 75 THROUGH 80)**

Entry	Meaning
Any characters.	Name identifying program or part of program on Source Program Listing, or any other meaning desired.
blank	None.

FILE EXTENSION SPECIFICATIONS

The File Extension Specifications entries are:

**SEQUENCE NUMBER
(COLUMNS 1 THROUGH 5)**

Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

**FORM TYPE
(COLUMN 6) (REQUIRED)**

Entry	Meaning
E	File Extension Specification

**CHAINING FILE RECORD SEQUENCE
(COLUMNS 7 THROUGH 8)**

Entries	Meaning
Two digits (00 through 99) or two letters (A through Z).	The same record sequence specified for the chaining file in the Input Specifications (Section VII).
Blank	None.

**CHAINING FIELD IDENTIFIER
(COLUMNS 9 THROUGH 10)**

Entry	Meaning
C1 through C9	The code number identifying chaining field, as defined in the Matching or Chaining Field entry in the Input Specifications.
Blank	The file is not a chaining file.

**FROM FILENAME
(COLUMNS 11 THROUGH 18)**

Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Name of chaining file used in Input Specification with chaining numbers in Matching or Chaining Field, or Name of file from which pre-execution time table or array is read. or Name of record address file (RAF) containing relative record numbers or record keys to be processed.
Blank.	This is a compilation-time table or array if an entry appears in the Entries Per Record Field (Columns 33-35); it is an execution-time array if no entry appears in the Number of Entries Per Record Field.

**TO FILENAME
(COLUMNS 19 THROUGH 26)**

Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	The name of the direct-access chained file processed by the chaining file noted in the From Filename Field, or The name of the KSAM, INDEX, IMAGE or direct-access input or update file

(Continued)

Entry	Meaning
	to be processed by the RAF named in the From Filename Field, or The name of a sequential output file to which the compile-time or pre-execution time table or array is named in the Table, Array or Routine Name Field (Columns 27 through 32) is copied when your program terminates.
Blank.	None. Tables and arrays, if any, are not copied.

**TABLE, ARRAY, OR ROUTINE NAME
(COLUMNS 27 THROUGH 32)**

Entry	Meaning
Table, array, or routine name. (Table names can contain from three to six characters, beginning with TAB; array and routine names can contain from one to six characters, beginning with a letter or one of the special characters @, \$, or #. In both table and array names, the remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.	The name of an array or table searched by your program, or The name of a conversion routine used to calculate relative record numbers retrieved from a RAF or chaining file.
Blank.	None.

**ENTRIES PER RECORD
(COLUMNS 33 THROUGH 35)**

Entry	Meaning
1 through 999	The number of table/array entries in each table or array input record.
Blank.	This is an execution-time array.

**ENTRIES PER TABLE/ARRAY
(COLUMNS 36 THROUGH 39)**

Entry	Meaning
1 through 9999	The number of entries reserved by the compiler for table or array.
Blank.	This is not a table or array.

**ENTRY LENGTH
(COLUMNS 40 THROUGH 42)**

Entries	Meaning
1 through 256	Entry length (space allowed for each item entered).
Blank.	This is not a table or array entry.

**DATA FORMAT
(COLUMN 43)**

Entry	Meaning
P	Packed decimal format.
B	Binary format.
L	ASCII format with preceding arithmetic (plus or minus) sign.
R	ASCII format with trailing arithmetic sign.
Blank.	ASCII format with no arithmetic sign, or alphanumeric table/array, or execution time array.

**DECIMAL POSITIONS
(COLUMN 44)**

Entry	Meaning
0 through 9	This is a numeric table or array with the specified number of decimal positions in each field.
Blank.	This is an alphanumeric table or array.

LINE COUNTER SPECIFICATIONS

TABLE/ARRAY SEQUENCE (COLUMN 45)

Entry	Meaning
A	Sequence check the table or array for ascending order.
D	Sequence check the table or array for descending order.
Blank	Do not sequence check the table or array; high or low LOKUP operations prohibited for unsequenced tables or arrays, and LOKUPs will be sequential.

TABLE/ARRAY NAME (COLUMNS 46 THROUGH 51)

Entry	Meaning
Table or array name, as defined in Paragraph 5-8.	The name of a second alternating table or array.
Blank.	The name in Columns 27 through 32 does not apply to an alternating table or array.

ENTRY LENGTH (COLUMNS 52 THROUGH 54)

Entry	Meaning
1 through 256	Entry length (space allowed for each item entered).
Blank	No entry is specified.

DATA FORMAT (COLUMN 55)

Entry	Meaning
P	Packed decimal format.
B	Binary format.
L	ASCII format with preceding arithmetic (plus or minus) sign.
R	ASCII format with trailing arithmetic sign.
Blank	ASCII format with no arithmetic sign, or alphanumeric table/array, or no alternating table/array specified.

DECIMAL POSITIONS (COLUMN 56)

Entry	Meaning
0 through 9	This is a numeric table or array with the specified number of decimal positions in each field.
Blank	This is an alphanumeric table or array.

TABLE/ARRAY SEQUENCE (COLUMN 57)

Entry	Meaning
A	Sequence check the table or array for ascending order.
D	Sequence check the table or array for descending order.
Blank	Do not sequence check the table or array.

COMMENTS (COLUMNS 58 THROUGH 74)

Entry	Meaning
Any characters.	Comments.

PROGRAM NAME (COLUMNS 75 THROUGH 80)

Entry	Meaning
Any characters.	Name identifying program or part of program on Source Program Listing, or any other meaning desired.
blank	None.

LINE COUNTER SPECIFICATIONS

The Line Counter Specifications entries are:

SEQUENCE NUMBER (COLUMNS 1 THROUGH 5)

Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

LINE COUNTER SPECIFICATIONS

**FORM TYPE
(COLUMN 6) (REQUIRED)**

Entry	Meaning
L	Line Counter Specification.

**FILE NAME
(COLUMNS 7 THROUGH 14) (REQUIRED)**

Entry	Meaning
Valid file name. (File names can contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Name of printable file to which this Line Counter Specification applies.

**LINE NUMBER
(COLUMNS 15 THROUGH 17)**

CHANNEL NUMBER OPTION

Entry	Meaning
1 through 112	The line number to which the channel number in the next field, or the overflow line, refers.
Blank	If the next field describes Channel 1, Line 6 is assigned; if next field describes overflow line or Channel 12, Line 60 is assigned. If the next field describes any other channel, the line number assigned equals the channel number multiplied by 5. (For instance, Channel 3 is assigned Line 15.)

LINE NUMBER OPTION

Entry	Meaning
1 through 112	The line number of the overflow line or form length, as determined by the next field.
Blank	None.

**CHANNEL NUMBER/OL/FL
(COLUMNS 18 THROUGH 19)**

CHANNEL NUMBER OPTION

Entry	Meaning
1 through 12	Line Number in preceding field is assigned to this channel.
OL	Line number in preceding channel designates the overflow line. If no number appears in previous field, Line 60 is assigned.
Blank	Line 6 is assigned to Channel 1; Line 60 is assigned to overflow line. For any other channel, the line number assigned equals the channel number multiplied by 5.

LINE NUMBER OPTION

Entry	Meaning
OL	Line number in preceding field designates the overflow line. If no number appears in previous field, Line 60 is assigned.
FL	Line number in previous field designates form length. If no number appears in previous field, Line 66 is assigned as last line.
Blank	Line 60 is assigned as Overflow Line; Line 66 is assigned for form length.

**LINE NUMBER/CHANNEL NUMBER/OL/FL
(COLUMNS 20 THROUGH 74)**

CHANNEL NUMBER OPTION

(Use these fields in exactly the same way as Columns 15 through 19).

**LINE NUMBER OPTION
(Columns 20 through 24 only.)**

(Use these fields in exactly the same way as Columns 15 through 19).

INPUT SPECIFICATIONS

PROGRAM NAME (COLUMNS 75 THROUGH 80)

Entry	Meaning
Any characters.	Name identifying program or part of program on Source Program Listing, or any other meaning desired.
blank	None.

INPUT SPECIFICATIONS

The Input Specifications entries are:

SEQUENCE NUMBER (COLUMNS 1 THROUGH 5)

Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

FORM TYPE (COLUMN 6) (REQUIRED)

Entry	Meaning
I	Input Specification

FILE NAME (COLUMNS 7 THROUGH 14) (REQUIRED)

Entries	Meaning
Valid file name. (File names contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Name of input, update, or combined file containing records and fields described.

GROUP SEQUENCE (COLUMNS 15 THROUGH 16) (REQUIRED)

Entry	Meaning
Two digits, 01 through 99.	Assign this sequence number to the type specified for this record in the <i>Record Identification Codes Field</i> (Columns 21 through 41), and verify the sequence of the record within its group when it is read.
Any two alphabetic characters.	No group sequence applies; do not sequence-check the input records.

NUMBER OF RECORDS (COLUMN 17)

Entry	Meaning
1	Only one record of this type is allowed in a group.
N	One or more records of this type is allowed in a group.
Blank	No restrictions on types per group because sequence <i>characters</i> instead of numbers appear in the <i>Group Sequence Field</i> ; sequencing is not requested.

OPTION/UDS (COLUMN 18)

Entry	Meaning
O	Records of the type specified are optional and may or may not be present in each group.
U	User Data Structure initialized from Local Data Area
Blank	At least one record of the type specified MUST be present in each group, or An alphabetic sequence entry is specified in the <i>Group Sequence Field</i> .

**RECORD INDICATOR/LOOK-AHEAD/
TRAILER
(COLUMNS 19 THROUGH 20)**

Entry	Meaning
A pair of digits, 01 through 99	Assign this general indicator.
F0 through F9 KA through KN, KP through KY	Assign this function key indicator. Assign this command key indicator.
L1 through L9	Assign this control level indicator.
LR	Assign the last-record indicator.
MR	Assign the matching record indicator.
OA through OG, OV	Assign this overflow indicator.
1P	Assign the first-page indicator.
H0 through H9	Assign this halt indicator.
U1 through U8	Assign this user indicator.
**	This record contains a look-ahead field.
TR	The next lines on the specification sheet define the trailer portion of a spread record.
DS	The next lines on the specification sheet define a Data Structure.
Blank	Assign no indicator, look-ahead field, or trailer portion.

**POSITION
(COLUMNS 21 THROUGH 24, 28 THROUGH 31,
35 THROUGH 38)**

Entry	Meaning
1 through 9999	Position where this identification code character appears in the input record.
Blank	A record identification code does not apply.

**NOT
(COLUMNS 25, 32, 39)**

Entry	Meaning
N	This character, zone, or digit, must NOT be present in the position indicated by the Position subfield.
Blank	If you specify a character, zone, or digit, it must be present.

**PORTION
(COLUMNS 26, 33, 40)**

Entry	Meaning
C	Use entire character (zone and digit portions).
Z	Use zone portion of character only.
D	Use digit portion of character only.
Blank	No character applies.

**CHARACTER
(COLUMNS 27, 34, 41)**

Entry	Meaning
Any letter, digit, or special character.	Test this character in the input record, in the position indicated by the Position Subfield.
Blank	No character applies in this position.

**STACKER SELECT
(COLUMN 42)**

Entry	Meaning
1 or Blank	Select Stacker No. 1
2	Select Stacker No. 2

INPUT SPECIFICATIONS

DATA FORMAT (COLUMN 43)

Entry	Meaning
P	Packed decimal format.
B	Binary format (1 or 2 word integer).
L	Unpacked (ASCII) decimal format with leading plus or minus sign.
R	Unpacked decimal format with trailing plus or minus sign.
1 through 9	Number of digits a binary field will use after conversion to packed decimal format.
Blank	Unpacked decimal format with no leading or trailing signs, or alphanumeric format. (Unpacked decimal format is indicated by the presence of a digit in the <i>Decimal Positions Field</i> (Column 52); alphanumeric format is indicated by a blank in that field.)

FROM FIELD POSITION (COLUMNS 44 THROUGH 47) (REQUIRED)

Entry	Meaning
A group of one to four digits, ranging from 1 to 9999.	Beginning position of field on input record.

TO FIELD POSITION (COLUMNS 48 THROUGH 51) (REQUIRED)

Entry	Meaning
A group of one to four digits, ranging from 1 to 9999.	Ending position of field on input record.

DECIMAL POSITIONS (COLUMN 52)

Entry	Meaning
Any digit 0 through 9.	The number of decimal positions in the field.
Blank.	The field contains alphanumeric data.

FIELD NAME (COLUMNS 53 THROUGH 58) (REQUIRED)

Entry	Meaning
Field name. (This may be a name of up to six characters, beginning with a letter or @, \$, or =; the remaining characters can be letters, digits, or @, \$, or =. Embedded blanks are not allowed.)	The name of the input field.
Array name. (This may be a name of up to six characters, beginning with a letter or @, \$, or =; the remaining characters can be letters, digits, or @, \$, or =. Embedded blanks are not allowed.)	The name of the input array.
Array name, comma, and index. The array name is as defined above. The index is either a number or the name of a field (as defined above) that contains a number. The combination of array name, comma, and index is limited to six characters.	The name of the input array item. (See Paragraph 5-39 for further information.)
PAGE, or PAGE1 through PAGE7	The name of a field that provides the page number for output files.
*ERROR	The name of a one-character field used for run-time error codes.

CONTROL LEVEL (COLUMNS 59 THROUGH 60)

Entry	Meaning
L1 through L9	Assign this control level indicator to the field.
Blank	This field is not a control field.

CALCULATION SPECIFICATIONS

**MATCHING/CHAINING FIELDS
(COLUMNS 61 THROUGH 62)**

Entry	Meaning
M1 through M9	This is a matching field identified by this code.
C1 through C9	This is a chaining field identified by this code.
Blank	This field is neither a matching or chaining field.

**FIELD RECORD RELATION
(COLUMNS 63 THROUGH 64)**

Entry	Meaning
01 through 99	General indicator relating this field to a specific record type.
F0 through F9	Function key indicator relating this field to a specific record type.
KA through KN KP through KY	Command key indicator regulating acceptance of data from a field.
L1 through L9	Control-level indicator regulating acceptance of data from a field if a control break occurs.
MR	Matching record indicator regulating acceptance of data from matching fields.
U1 through U8	User indicator conditioning use of a field.
H1 through H9	Halt indicator relating this field to a specific record type.
OA through OG, OV	Overflow indicator.
1P	First-page indicator, used as a general indicator.
LR	Last-record indicator.
Blank	This field appears in all record types covered by this OR relationship.

**FIELD INDICATOR FIELD
(COLUMNS 65 THROUGH 70)**

Entry	Meaning
01 through 99	Indicator used to test data in field. (If entered in Columns 65 through 66, tests for positive data; if in

Entry	Meaning
F0 through F9 KA through KN, KP through KY	Columns 67 through 68, tests for negative data; if in Columns 69 through 70, tests for zeros or blanks.)
L1 through L9	
MR	
U1 through U8	
H1 through H9	
OA through OG, OV	
1P	
LR	
Blank	Do not test data in field.

**PROGRAM NAME
(COLUMNS 75 THROUGH 80)**

Entry	Meaning
Any characters.	Name identifying program or part of program on Source Program Listing, or any other meaning desired.
blank	None.

CALCULATION SPECIFICATIONS

The Calculation Specification entries are:

**SEQUENCE NUMBER
(COLUMNS 1 THROUGH 5)**

Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

**FORM TYPE
(COLUMN 6) (REQUIRED)**

Entry	Meaning
C	Calculation Specification

CALCULATION SPECIFICATIONS

CONTROL LEVEL (COLUMNS 7 THROUGH 8)

Entry	Meaning
L0 through L9	A total calculation to be done when appropriate control break occurs or when the indicator is set on by a SETON operation; normally done at total time in RPG/3000 program cycle. L0 is always on.
LR	A total calculation done after the last record has been processed, or after LR indicator is set on by the program.
SR	Calculation operation defined in a subroutine.
Blank	Detail calculation, done at detail time in the RPG/3000 program cycle or a subroutine calculation if a BEGSR operation has been specified prior to or on this line.
AN	Establishes AND relation between indicators on this line and those on previous line.
OR	Establishes OR relation between indicators on this line and those on previous line or in previous AND relationship.

INDICATORS FIELD (COLUMNS 9 THROUGH 17)

Columns 9, 12, 15 Entries	Meaning
N	The indicator must be OFF to condition the operation.
Blank	The indicator must be ON to condition the operation.

Columns 10 through 11, 13 through 14, 16 through 17 Entries

Entries	Meaning
01 through 99	General indicator used elsewhere in program.
F0 through F9 KA through KN, KP through KY	Function key indicator. Command key indicator.
L0 through L9	Control level indicators.

LR	Last record indicator.
MR	Matching record indicator.
H0 through H9	Halt indicators.
U1 through U8	External indicator.
OA through OG, OV	Overflow indicator.
1P	First-page indicator.
*(Column 11 only)	This calculation is conditioned by the same indicators as the previous calculation.
Blank	Operation is performed for every record read if Control Level Field does not contain L0 through L9, or SR.

FACTOR1 AND FACTOR 2 FIELDS (COLUMNS 18 THROUGH 27 AND 33 THROUGH 42)

Entries	Meaning
The name of a field, table, array, array element or file.	The storage area containing the data to be used, or (in the case of a file name) the file to be used.
The name of a subroutine.	The subroutine to operate on the data.
A label.	The label for a TAG, ENDSR, or GOTO operation.
An alphanumeric or numeric literal.	The actual data to be used.
Blank	No operand is used.

OPERATION (COLUMNS 28 THROUGH 32) (REQUIRED)

Entries	Meaning
ADD	Add <i>Factors 1</i> and <i>2</i> , and place the sum in the result field.
BITOF	Turn off the result field bits identified in <i>Factor 2</i> .
BITON	Turn on the result field bits identified in <i>Factor 2</i> .
BEGSR	Define beginning of a subroutine.
CHAIN	Read a record from a chained file on disc during calculation operations.

Columns 28 through 32 Entries	Meaning
COMP	Compare <i>Factor 1</i> with <i>Factor 2</i> , and set resulting indicators on or off.
DEBUG	List indicators that are set on.
DIV	Divide <i>Factor 1</i> by <i>Factor 2</i> , and place quotient in result field.
DSPLM	Print a User Message; also a field, table item, or array element and blank out or alter the field.
DSPLY	Print one or two fields, table items, literals, or array elements; also blank out or alter a field.
ENDSR	Define end of subroutine.
EXSR	Execute subroutine.
EXCPT	Write records while calculation operations are in progress.
EXIT	Transfer control from RPG program to user subroutine.
ERPGC	Indicate end of conversion routine.
EXTCV	Indicate conversion routine is external to RPG program.
FORCE	Identify the file from which the next record is to be selected.
GOTO	Branch to named instruction.
LOKUP	Search for specific items in table or array.
MHHZO	Move zone from leftmost position of <i>Factor 2</i> to leftmost position of <i>result field</i> .
MHLZO	Move zone from leftmost position of <i>Factor 2</i> to rightmost position of <i>result field</i> .
MLLZO	Move zone from rightmost position of <i>Factor 2</i> to rightmost position of <i>result field</i> .
MLHZO	Move zone from rightmost position of <i>Factor 2</i> to leftmost position of <i>result field</i> .
MOVE	Move characters from <i>Factor 2</i> to rightmost position of <i>result field</i> .
MOVEA	Move characters from <i>Factor 2</i> array to leftmost position of <i>result field</i> .

MOVEL	Move characters from <i>Factor 2</i> to leftmost position of <i>result field</i> .
MSG	Move message specified in <i>Factor 1</i> into <i>result field</i> .
MULT	Multiply <i>Factor 1</i> by <i>Factor 2</i> , and place produce in <i>result field</i> .
MVR	Move remainder from previous divide operation to <i>result field</i> .
PARM	Specify parameters for EXIT operation.
READ	Read input from demand file during current program cycle.
RLABEL	Allow external subroutine to reference field internal to RPG program.
RPGCV	Indicate beginning of conversion routine.
SET	Label softkeys and enable F1 thru F8 for DSPLY and DSPLM.
SETLL	Set lower limit for KSAM, INDEX, or IMAGE file processed sequentially between limits.
SETOF	Turn resulting indicators off.
SETON	Turn resulting indicators on.
SORTA	Sort array specified in <i>Factor 2</i> into ascending or descending order.
SQRT	Derive square root of <i>Factor 2</i> , and place it in <i>result field</i> .
SUB	Subtract <i>Factor 2</i> from <i>Factor 1</i> and place difference in <i>result field</i> .
TAG	Name instruction to which GOTO operation transfers.
TESTB	Test result field bits identified by <i>Factor 2</i> , and set resulting indicators on or off.
TESTN	Test alphanumeric field for presence of digits or blanks, and turn resulting indicator on or off.
TESTZ	Test zone of leftmost character in result field, and set resulting indicators on or off.
XFOOT	Add array items and place sum in result field.
Z-ADD	Replace <i>result field</i> with <i>Factor 2</i> .
Z-SUB	Replace <i>result field</i> with negative of <i>Factor 2</i> .

OUTPUT SPECIFICATIONS

RESULT FIELD (COLUMNS 43 THROUGH 48)

Entries	Meaning
Name of field, table, array, or indexed array element.	Area where result is stored.
Blank	Result field does not apply to this operation.

FIELD LENGTH (COLUMNS 49 THROUGH 51)

Entry	Meaning
1 through 256	Result field length.
Blank	This is an alphanumeric or numeric field, described elsewhere in the program.

DECIMAL POSITIONS (COLUMN 52)

Entry	Meaning
0 through 9	Number of decimal positions in numeric result field.
Blank	This is an alphanumeric field, or a numeric field defined elsewhere.

HALF-ADJUST (COLUMN 53)

Entry	Meaning
H	Half adjust data.
Blank	Do not half adjust data.

RESULTING INDICATORS (COLUMNS 54 THROUGH 59)

Entry	Meaning
01 through 99	General indicator.
F0 through F9	Function key indicator.
KA through KN, KP through KY	Command key indicator.
H0 through H9	Halt indicator.
L0 through L9	Control level indicator.
LR	Last record indicator.

OA through OG, OV	Overflow indicator.
1P	First page indicator.
MR	Matching record indicator.
U1 through U8	User indicator.
Blank	No indicator set.

COMMENTS (COLUMNS 60 THROUGH 74)

Entry	Meaning
Any characters.	Comment.

PROGRAM NAME (COLUMNS 75 THROUGH 80)

Entry	Meaning
Any characters.	Name identifying program or part of program on Source Program Listing, or any other meaning desired.
Blank	None.

OUTPUT SPECIFICATIONS

The Output Specifications entries are:

SEQUENCE NUMBER (COLUMNS 1 THROUGH 5)

Entry	Meaning
00000 through 99999	Sequence number.
blank	None. (Record not sequence-checked.)

FORM TYPE (COLUMN 6) (REQUIRED)

Entry	Meaning
O	Output Specifications.

**FILE NAME
(COLUMNS 7 THROUGH 14) (REQUIRED)**

Entry	Meaning
Valid file name. (File names contain from one to eight characters, beginning with a letter. The remaining characters can be letters or digits. Embedded blanks are not allowed.)	Name of output, update, or combined file containing records and fields described.

**TYPE
(COLUMN 15) (REQUIRED)**

Entry	Meaning
H	Heading record.
D	Detail record.
T	Total record.
E	Exception record.

**RECORD ADDITION/DELETION
(COLUMNS 16 THROUGH 18)**

Entry	Meaning
ADD	Add new records to the file.
DEL	Delete records from an updated or chained input or output file. This entry applies to KSAM, INDEX, or IMAGE files only.
Blank	For sequential, non-indexed output files, write new records to the beginning of the file. For KSAM, INDEX, and IMAGE output files, write new records to the file in key sequence. For update files, update the current record.

**STACKER SELECT
(COLUMN 16)**

Entry	Meaning
1 or blank	Select Stacker No. 1
2	Select Stacker No. 2

Entry	Meaning
F	Output overflow information at this point (by fetching the RPG/3000 Overflow Routine).
Blank	Suppress overflow output at this point.

**RELEASE FILE
(COLUMN 16)**

Entry	Meaning
R	Release the file (close or close and re-open).
blank	Do not release the file.

**SPACE
(COLUMNS 17 THROUGH 18)**

Entry	Meaning
0	Suppress spacing.
1 or blank	Space one line unless entry in 17 and/or 19-22.
2	Space two lines.
3	Space three lines.

**SKIP
(COLUMNS 19 THROUGH 22)**

Entry	Meaning
01 through 12	Skip to this line number (if L appears in the Carriage Control Type Field (Column 53) of the Control Record Specification). Skip to this Channel Number (if the Carriage Control Type Field in the Control Record Specification is blank).
13 through 99	Skip to this line number (Line 13 through 99) (if L appears in the Carriage Control Type Field of the Control Record Specification).
A0 through A9	Skip to this Line number (Lines 100 through 109) (if L appears in the Carriage Control Type Field of the Control Record Specification).
B0 through B2	Skip to this line number (Line 110 through 112) (if L appears in the Carriage Control Type Field of the Control Record Specification).
Blank	Skip no lines.

**OUTPUT INDICATORS
(COLUMNS 23 THROUGH 31)**

Columns 23, 26, or 29 Entry	Meaning
N	Output the data defined on this specification line only if the indicator in the next field is off.
Blank	Output the data defined on this line only if the indicator specified in the next field is on.

OUTPUT SPECIFICATIONS

Columns 24 through 25, 27 through 28, and 30 through 31 Entry	Meaning
01 through 99	Use any general indicator previously assigned to indicate the result of a calculation or to identify a record type or field.
F0 through F9	Use function key indicator previously set as a general indicator, through the RPG/VPLUS interface, or in response to a DSPLY or DSPLM operation.
KA through KN, KP through KY	Use command key indicator previously set as a general indicator or through the RPG Screen Interface.
L1 through L9	Use this control level indicator, previously assigned to the record or field being described.
L0	Use the Level 0 control level indicator, which is always set on.
LR	Use the last record indicator.
MR	Use the matching record indicator.
OA through OG, OV	Use this overflow indicator, normally previously assigned to this file.
1P	Use the first-page indicator.
H0 through H9	Use this halt indicator.
U1 through U8	Use this user indicator, normally set prior to program execution.
Blank	Do not use an output indicator.

FIELD NAME (COLUMNS 32 THROUGH 37)

Entry	Meaning
Field name previously defined in the program. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.)	The name of the output field.
Table name. (This may be a name of up to six characters, beginning with TAB. The remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.)	The name of the output table.

Entry	Meaning
Array name. (This may be a name of up to six characters, beginning with a letter or @, \$, or #; the remaining characters can be letters, digits, or @, \$, or #. Embedded blanks are not allowed.)	The name of the output array.
Array name, comma, and index. The array name is as defined above. The index is either a number or the name of a field (as defined above) that contains a number. The combination of array name, comma and index is limited to six characters	The name and index of the output array element.
PAGE, or PAGE1 through PAGE7	A special RPG/3000 field that provides the number of the current page for printed output files.
*PLACE	A special RPG/3000 field that repeats the output of fields defined previously within the record description.
*PRINT	A special RPG/3000 field that prints, along the top of a card, those characters punched in the card on output.
UPDATE	A special RPG/3000 field that provides the current date.
UDAY	A special RPG/3000 field that provides the current day of the month.
UMONTH	A special RPG/3000 field that provides the current month.
UYEAR	A special RPG/3000 field that provides the current year.
*ERROR	A special RPG/3000 field used for run-time error codes.

**EDIT CODE
(COLUMN 38)**

Entry	Meaning	
X	Removes plus sign from positive field.	
Y	Adds slashes or decimal points to numeric fields; normally used for date fields.	
Z	Suppresses leading zeros.	
1 2 3 4 A B C D J K L M	Complex edit codes, used as shown in Section IX.	
Blank		Do not use an edit code.

**BLANK AFTER
(COLUMN 39)**

Entry	Meaning
B	Re-set the contents of the field after output.
Blank	Do not re-set the contents of the field.

**END POSITION*
(COLUMNS 40 THROUGH 43)**

Entry	Meaning
1 through 9999	The number of the position where the rightmost character of the data is to appear on the output record.
Blank or ±nnn	Relative end position.

**PACKED/BINARY
(COLUMN 44)**

Entry	Meaning
P	Write the field in packed decimal format.
B	Write the field in binary format. A field of five digits or fewer is written as two bytes; a field of six to ten digits is written as four bytes.
L	Write the field in unpacked format with an arithmetic sign (plus or minus) preceding the field.
R	Write the field in unpacked format with an arithmetic sign following the field.
2	Write the field in binary format, two bytes long.
4	Write the field in binary format, four bytes long.
Blank	This is an unpacked numeric or alphanumeric field; a numeric field containing data to be edited; or a constant.

**CONSTANT/EDIT WORD
(COLUMNS 45 THROUGH 70)**

Entry	Meaning
From one to 24 characters (any letters, digits, or special characters) surrounded by quotation marks.	Constant (if no field name appears on this line), or edit word (if field name appears on this line). If edit code is specified in the Edit Code Field (Column 38), the Constant/Edit Word Field may contain a \$ (for floating dollar sign) or * (for asterisk fill).
Blank	No constant is output or no edit word applies.

OUTPUT SPECIFICATIONS

COMMENTS (COLUMNS 71 THROUGH 74)

Entries	Meaning
Any characters.	Comments.

PROGRAM NAME (COLUMNS 75 THROUGH 80)

Entry	Meaning
Any characters.	Name identifying program or part of program on Source Program Listing, or any other meaning desired.
blank	None.

ASCII CHARACTER SET AND COLLATING SEQUENCE

APPENDIX

G

The American Standard Code for Information Interchange (ASCII) character set and collating sequence is presented in this appendix. In this sequence, the characters appear in order of the ASCII bit patterns by which they are represented internally.

ASCII CHARACTER	MEANING	B8	B7	B6	B5	B4	B3	B2	B1	HP 3000 OCTAL VALUE (LEFT BYTE)	OCTAL VALUE (RIGHT BYTE)	HEXADECIMAL VALUE	HOLLERITH CARD ROWS PUNCHED
NUL	Null (Normally ignored by MPE/3000.)	0	0	0	0	0	0	0	0	000000	000000	00	12-0-1-8-9
SOH	Start of header	0	0	0	0	0	0	0	1	000001	000000	01	12-1-9
STX	Start of text	0	0	0	0	0	0	1	0	001000	000002	02	12-2-9
ETX	End of text	0	0	0	0	0	0	0	1	000000	000003	03	12-3-9
EOT	End of transmission	0	0	0	0	0	0	1	0	002000	000004	04	7-9
ACK	Acknowledge	0	0	0	0	0	1	1	0	003000	000006	06	0-6-8-9
DEL	Delete	0	0	0	0	0	1	1	1	003400	000007	07	0-7-8-9
BS	Backspace	0	0	0	0	1	0	0	0	004000	000010	08	11-6-9
HT	Horizontal tabulation	0	0	0	0	1	0	0	1	004400	000011	09	12-5-9
LF	Line Feed (Normally ignored by MPE/3000.)	0	0	0	0	1	0	1	0	005000	000012	0A	0-5-9
VT	Vertical tabulation	0	0	0	0	1	0	1	1	005400	000013	0B	12-3-8-9
FF	Form feed	0	0	0	0	1	1	0	0	006000	000014	0C	12-4-8-9
CR	Carriage return	0	0	0	0	1	1	0	1	006400	000015	0D	12-5-8-9
SO	Shift out	0	0	0	0	1	1	1	0	007000	000016	0E	12-6-8-9
SI	Shift in	0	0	0	0	1	1	1	1	007400	000017	0F	12-7-8-9
DLE	Data Link escape	0	0	0	1	0	0	0	0	010000	000020	10	12-11-1-8-9
DC1	Device control 1 (Normally ignored by MPE/3000.)	0	0	0	1	0	0	0	1	010400	000021	11	11-1-9
DC2	Device control 2 (Normally ignored by MPE/3000.)	0	0	0	1	0	0	1	0	011000	000022	12	11-2-9
DC3	Device control 3 (Normally ignored by MPE/3000.)	0	0	0	1	0	0	1	1	011400	000023	13	11-3-9

ASCII Character Set

ASCII CHARACTER	MEANING	B8	B7	B6	B5	B4	B3	B2	B1	HP 3000 OCTAL VALUE (LEFT BYTE)	OCTAL VALUE (RIGHT BYTE)	HEXADECIMAL VALUE	HOLLERITH CARD ROWS PUNCHED
.	Period (Decimal)	0	0	1	0	1	1	1	0	027000	000056	2E	12-3-8
/	Slant	0	0	1	0	1	1	1	1	027400	000057	2F	0-1
0	Zero	0	0	1	1	0	0	0	0	030000	000060	30	0
1	One	0	0	1	1	0	0	0	1	030400	000061	31	1
2	Two	0	0	1	1	0	0	1	0	031000	000062	32	2
3	Three	0	0	1	1	0	0	1	1	031400	000063	33	3
4	Four	0	0	1	1	0	1	0	0	032000	000064	34	4
5	Five	0	0	1	1	0	1	0	1	032400	000065	35	5
6	Six	0	0	1	1	0	1	1	0	033000	000066	36	6
7	Seven	0	0	1	1	0	1	1	1	033400	000067	37	7
8	Eight	0	0	1	1	1	0	0	0	034000	000070	38	8
:	Colon	0	0	1	1	1	0	1	0	035000	000072	3A	2-8
;	Semi-colon	0	0	1	1	1	0	1	1	035400	000073	3B	11-6-8
<	Less than	0	0	1	1	1	1	0	0	036000	000074	3C	12-4-8
>	Greater than	0	0	1	1	1	1	1	0	037000	000076	3E	0-6-8
@	Commercial at	0	1	0	0	0	0	0	0	040000	000100	40	4-8
B	Uppercase B	0	1	0	0	0	0	1	0	041000	000102	42	12-2
D	Uppercase D	0	1	0	0	0	1	0	0	042000	000104	44	12-4
F	Uppercase F	0	1	0	0	0	1	1	0	043000	000106	46	12-6

ASCII CHARACTER	MEANING	ASCII BIT PATTERN								HP 3000 OCTAL VALUE (LEFT BYTE)	HEXADECIMAL VALUE	HOLLERITH CARD ROWS PUNCHED	
		B8	B7	B6	B5	B4	B3	B2	B1				
d	Lowercase d	0	1	1	0	0	1	0	0	062000	000144	64	12-0-4
e	Lowercase e	0	1	1	0	0	1	0	0	062000	000144	64	12-0-4
f	Lowercase f	0	1	1	0	0	1	1	0	063000	000146	66	12-0-6
g	Lowercase g	0	1	1	0	0	1	1	0	063000	000146	66	12-0-6
h	Lowercase h	0	1	1	0	1	0	0	0	064000	000150	68	12-0-8
i	Lowercase i	0	1	1	0	1	0	1	0	064000	000150	68	12-0-8
j	Lowercase j	0	1	1	0	1	0	1	0	065000	000152	6A	12-11-1
k	Lowercase k	0	1	1	0	1	1	0	0	065000	000152	6A	12-11-1
l	Lowercase l	0	1	1	0	1	1	0	0	066000	000154	6C	12-11-3
m	Lowercase m	0	1	1	0	1	1	1	0	066000	000154	6C	12-11-3
n	Lowercase n	0	1	1	0	1	1	1	0	067000	000156	6E	12-11-5
o	Lowercase o	0	1	1	0	1	1	1	1	067000	000156	6E	12-11-5
p	Lowercase p	0	1	1	1	0	0	0	0	070000	000160	70	12-11-7
q	Lowercase q	0	1	1	1	0	0	1	1	070000	000160	70	12-11-7
r	Lowercase r	0	1	1	1	0	0	1	0	071000	000162	72	12-11-9
s	Lowercase s	0	1	1	1	0	1	1	1	071000	000162	72	12-11-9
t	Lowercase t	0	1	1	1	0	1	0	0	072000	000164	74	11-0-3
u	Lowercase u	0	1	1	1	0	1	0	1	072400	000165	75	11-0-4
v	Lowercase v	0	1	1	1	0	1	1	0	073000	000166	76	11-0-5
w	Lowercase w	0	1	1	1	0	1	1	1	073400	000167	77	11-0-6
x	Lowercase x	0	1	1	1	1	0	0	0	074000	000170	78	11-0-7
y	Lowercase y	0	1	1	1	1	0	0	1	074400	000171	79	11-0-8
z	Lowercase z	0	1	1	1	1	0	1	0	075000	000172	7A	11-0-9
{	Opening brace	0	1	1	1	1	0	1	1	075400	000173	7B	12-0
	Verticle line	0	1	1	1	1	1	0	0	076000	000174	7C	12-11
}	Closing brace	0	1	1	1	1	1	0	1	076400	000175	7D	11-0



~	Tilde	0	1	1	1	1	1	1	1	0	077000	000176	7E	11-0-1
K0	(The following characters are all reserved for future use.)	1	0	0	0	0	0	0	0	0	100000	000200	80	11-0-1-8-9
K2		1	0	0	0	0	0	1	0	0	101000	000202	82	0-2-9
K4		1	0	0	0	0	1	0	0	0	102000	000204	84	0-4-9
K6		1	0	0	0	0	1	1	0	0	103000	000206	86	12-6-9
K8		1	0	0	0	1	0	0	0	0	104000	000210	88	0-8-9
K10		1	0	0	0	1	0	0	1	0	105000	000212	8A	0-2-8-9
K12		1	0	0	0	1	1	0	0	0	106000	000214	8C	0-4-8-9
K14		1	0	0	0	1	1	1	0	0	107000	000216	8E	12-2-8-9
K16		1	0	0	1	0	0	0	0	0	110000	000220	90	12-11-0-1-8-9
K18		1	0	0	1	0	0	1	0	0	111000	000222	92	11-2-8-9
K20		1	0	0	1	0	1	0	0	0	112000	000224	94	4-9
K22		1	0	0	1	0	1	0	1	0	113000	000226	96	6-9
K24		1	0	0	1	0	1	1	0	0	114000	000230	98	8-9
K26		1	0	0	1	1	0	0	1	0	114400	000231	99	1-8-9

ASCII Character Set

ASCII CHARACTER	MEANING	ASCII BIT PATTERN								HP 3000 OCTAL VALUE (LEFT BYTE)	OCTAL VALUE (RIGHT BYTE)	HEXADECIMAL VALUE	HOLLERITH CARD ROWS PUNCHED
		B8	B7	B6	B5	B4	B3	B2	B1				
K26		1	0	0	1	1	0	1	0	115000	000232	9A	2-8-9
K27		1	0	0	1	1	0	1	1	115400	000233	9B	3-8-9
K28		1	0	0	1	1	1	0	0	116000	000234	9C	12-4-9
K29		1	0	0	1	1	1	0	1	116400	000235	9D	11-4-9
K30		1	0	0	1	1	1	1	0	117000	000236	9E	6-8-9
K31		1	0	0	1	1	1	1	1	117400	000237	9F	11-0-1-9
N0		1	0	1	0	0	0	0	0	120000	000240	A0	12-0-1-9
N1		1	0	1	0	0	0	0	1	120400	000241	A1	12-0-3-9
N2		1	0	1	0	0	0	1	0	121000	000242	A2	12-0-3-9
N3		1	0	1	0	0	0	1	1	121400	000243	A3	12-0-3-9
N4		1	0	1	0	0	1	0	0	122000	000244	A4	12-0-5-9
N5		1	0	1	0	0	1	0	1	122400	000245	A5	12-0-5-9
N6		1	0	1	0	0	1	1	0	123000	000246	A6	12-0-7-9
N7		1	0	1	0	0	1	1	1	123400	000247	A7	12-0-7-9
N8		1	0	1	0	1	0	0	0	124000	000250	A8	12-1-8
N9		1	0	1	0	1	0	0	1	124400	000251	A9	12-1-8
N10		1	0	1	0	1	0	1	0	125000	000252	AA	12-11-2-9
N11		1	0	1	0	1	0	1	1	125400	000253	AB	12-11-2-9
N12		1	0	1	0	1	1	0	0	126000	000254	AC	12-11-4-9
N13		1	0	1	0	1	1	0	1	126400	000255	AD	12-11-4-9
N14		1	0	1	0	1	1	1	0	127000	000256	AE	12-11-6-9
N15		1	0	1	0	1	1	1	1	127400	000257	AF	12-11-6-9
N16		1	0	1	1	0	0	0	0	130000	000260	B0	12-11-8-9
N17		1	0	1	1	0	0	0	1	130400	000261	B1	12-11-8-9
N18		1	0	1	1	0	0	1	0	131000	000262	B2	11-0-2-9
N19		1	0	1	1	0	0	1	1	131400	000263	B3	11-0-2-9

N20	1 0 1 1 0 1 0 0	132000	000264	B4	11-0-4-9
N21	1 0 1 1 0 1 0 1	132400	000265	B5	11-0-5-9
N22	1 0 1 1 0 1 1 0	133000	000266	B6	11-0-6-9
N23	1 0 1 1 0 1 1 1	133400	000267	B7	11-0-7-9
N24	1 0 1 1 1 0 0 0	134000	000270	B8	11-0-8-9
N25	1 0 1 1 1 0 0 1	134400	000271	B9	0-1-8
N26	1 0 1 1 1 0 1 0	135000	000272	BA	12-11-0
N27	1 0 1 1 1 0 1 1	135400	000273	B8	12-11-0-1-9
N28	1 0 1 1 1 1 0 0	136000	000274	BC	12-11-0-2-9
N29	1 0 1 1 1 1 0 1	136400	000275	BD	12-11-0-3-9
N30	1 0 1 1 1 1 1 0	137000	000276	BE	12-11-0-4-9
N31	1 0 1 1 1 1 1 1	137400	000277	BF	12-11-0-5-9
N32	1 1 0 0 0 0 0 0	140000	000300	C0	12-11-0-6-9
N33	1 1 0 0 0 0 0 1	140400	000301	C1	12-11-0-7-9
N34	1 1 0 0 0 0 1 0	141000	000302	C2	12-11-0-8-9
N35	1 1 0 0 0 0 1 1	141400	000303	C3	12-0-1-8
N36	1 1 0 0 0 1 0 0	142000	000304	C4	12-0-2-8
N37	1 1 0 0 0 1 0 1	142400	000305	C5	12-0-3-8
N38	1 1 0 0 0 1 1 0	143000	000306	C6	12-0-4-8
N39	1 1 0 0 0 1 1 1	143400	000307	C7	12-0-5-8
N40	1 1 0 0 1 0 0 0	144000	000310	C8	12-0-6-8
N41	1 1 0 0 1 0 0 1	144400	000311	C9	12-0-7-8
N42	1 1 0 0 1 0 1 0	145000	000312	CA	12-11-1-8
N43	1 1 0 0 1 0 1 1	145400	000313	CB	12-11-2-8
N44	1 1 0 0 1 1 0 0	146000	000314	CC	12-11-3-8
N45	1 1 0 0 1 1 0 1	146400	000315	CD	12-11-4-8
N46	1 1 0 0 1 1 1 0	147000	000316	CE	12-11-5-8
N47	1 1 0 0 1 1 1 1	147400	000317	CF	12-11-6-8

ASCII Character Set

ASCII CHARACTER	MEANING	ASCII BIT PATTERN								HP 3000 OCTAL VALUE (LEFT BYTE)	OCTAL VALUE (RIGHT BYTE)	HEXADECIMAL VALUE	HOLLERITH CARD ROWS PUNCHED
		B8	B7	B6	B5	B4	B3	B2	B1				
N48		1	1	0	1	0	0	0	0	15000	000320	D0	12-11-7-8
N49		1	1	0	1	0	0	1	150400	000321	D1	11-0-1-8	
N50		1	1	0	1	0	0	1	151000	000322	D2	11-0-2-8	
N51		1	1	0	1	0	0	1	151400	000323	D3	11-0-3-8	
N52		1	1	0	1	0	1	0	152000	000324	D4	11-0-4-8	
N53		1	1	0	1	0	1	0	152400	000325	D5	11-0-5-8	
N54		1	1	0	1	0	1	0	153000	000326	D6	11-0-6-8	
N55		1	1	0	1	0	1	1	153400	000327	D7	11-0-7-8	
N56		1	1	0	1	1	0	0	154000	000330	D8	12-11-0-1-8	
N57		1	1	0	1	1	0	1	154400	000331	D9	12-11-0-1	
N58		1	1	0	1	1	0	1	155000	000332	DA	12-11-0-2	
N59		1	1	0	1	1	0	1	156400	000333	DB	12-11-0-3	
N60		1	1	0	1	1	1	0	156000	000334	DC	12-11-0-4	
N61		1	1	0	1	1	1	0	156400	000335	DD	12-11-0-5	
N62		1	1	0	1	1	1	1	157000	000336	DE	12-11-0-6	
N63		1	1	0	1	1	1	1	157400	000337	DF	12-11-0-7	
G0		1	1	1	0	0	0	0	160000	000340	E0	12-11-0-8	
G1		1	1	1	0	0	0	1	160400	000341	E1	12-11-0-9	
G2		1	1	1	0	0	0	1	161000	000342	E2	12-11-0-2-8	
G3		1	1	1	0	0	0	1	161400	000343	E3	12-11-0-3-8	
G4		1	1	1	0	0	1	0	162000	000344	E4	12-11-0-4-8	
G5		1	1	1	0	0	1	0	162400	000345	E5	12-11-0-5-8	
G6		1	1	1	0	0	1	1	163000	000346	E6	12-11-0-6-8	
G7		1	1	1	0	0	1	1	163400	000347	E7	12-11-0-7-8	
G8		1	1	1	0	1	0	0	164000	000350	E8	12-0-2-8-9	
G9		1	1	1	0	1	0	1	164400	000351	E9	12-0-3-8-9	

G10	1 1 1 0 1 0 1 0	165000	000352	EA	12-0-4-8-9
G11	1 1 1 0 1 0 1 1	165400	000353	EB	12-0-5-8-9
G12	1 1 1 0 1 1 0 0	166000	000354	EC	12-0-6-8-9
G13	1 1 1 0 1 1 0 1	166400	000355	ED	12-0-7-8-9
G14	1 1 1 0 1 1 1 0	167000	000356	EE	12-11-2-8-9
G15	1 1 1 0 1 1 1 1	167400	000357	EF	12-11-3-8-9
G16	1 1 1 1 0 0 0 0	170000	000360	F0	12-11-4-8-9
G17	1 1 1 1 0 0 0 1	170400	000361	F1	12-11-5-8-9
G18	1 1 1 1 0 0 1 0	171000	000362	F2	12-11-6-8-9
G19	1 1 1 1 0 0 1 1	171400	000363	F3	12-11-7-8-9
G20	1 1 1 1 0 1 0 0	172000	000364	F4	11-0-2-8-9
G21	1 1 1 1 0 1 0 1	172400	000365	F5	11-0-3-8-9
G22	1 1 1 1 0 1 1 0	173000	000366	F6	11-0-4-8-9
G23	1 1 1 1 0 1 1 1	173400	000367	F7	11-0-5-8-9
G24	1 1 1 1 1 0 0 0	174000	000370	F8	11-0-6-8-9
G25	1 1 1 1 1 0 0 1	174400	000371	F9	11-0-7-8-9
G26	1 1 1 1 1 0 1 0	175000	000372	FA	12-11-0-2-8-9
G27	1 1 1 1 1 0 1 1	175400	000373	FB	12-11-0-3-8-9
G28	1 1 1 1 1 1 0 0	176000	000374	FC	12-11-0-4-8-9
G29	1 1 1 1 1 1 0 1	176400	000375	FD	12-11-0-5-8-9
G30	1 1 1 1 1 1 1 0	177000	000376	FE	12-11-0-6-8-9
E0	1 1 1 1 1 1 1 1	177400	000377	FF	12-11-0-7-8-9



SIMULATING ISAM FUNCTIONS WITH IMAGE/3000

APPENDIX

H

You may use IMAGE/3000 in conjunction with RPG/3000 to simulate the forward and backward read functions of the Indexed Sequential Access Method (ISAM) feature available with certain RPG compilers.

Note: As a general rule, however, you obtain faster processing by using KSAM/3000 for this purpose.

When using IMAGE/3000, you specify the IMAGE files in your RPG programs using the same general specifications that you would use for ISAM or KSAM files. However, specific differences arise from the fact that you must supply more information to read an IMAGE file than an ISAM/KSAM file. These differences are noted wherever IMAGE is mentioned throughout this manual, and are summarized below. Specifically, IMAGE requires that you:

- Specify an IMAGE/3000 file by naming both the data base and the data set within that data base.
- Specify the item name. (This is the name assigned to the key field when the data base is defined.)
- Specify an open mode (shared, shared with locking, or exclusive) for the file.
- Specify an input/output mode for the file.

These additional specifications are provided for in the File Description Specification continuation record. The standard part of the File Description record resembles an RSAM or KSAM specification. The Input, Calculation, and Output Specifications are the same as when using the regular, RSAM, or KSAM files. (The Calculation Specifications include an end-of-chain indicator.)

H-1. PERFORMANCE

RPG/3000 provides general support of IMAGE/3000 and provides an excellent tool for producing reports from IMAGE/3000 data bases. The ISAM simulation is less efficient than using KSAM for indexed-sequential operations.

In sequential mode, if the records are loaded in sequence, they should be processed as rapidly as any sequential disc file.

If records are being added to the file, the time required depends on the size of the file and the eventual position of the records in the file. If records with keys that fall at the end of the file are added, processing will be done quickly; however, the closer to the beginning of the file that a record belongs, the longer it takes to process that record. In general, adding records to an IMAGE file is slower than adding them to an ISAM or KSAM file.

H-2. ISAM SIMULATION RULES

To simulate the use of ISAM with RPG/3000 by using IMAGE/3000, follow these basic steps:

1. Create an IMAGE/3000 detail data set with a sort key to sequence it. Two master data sets must reference the detail data set; one contains a single entry — a two-byte dummy key pointing to the

Simulating ISAM Functions With IMAGE/3000

beginning of the detail set (which is in sequence by the sort key); the other master data set has one entry for each record in the detail data set. When the file is set up in this manner, it can be accessed sequentially or randomly by key.

2. IMAGE/3000 requires that all fields begin and end on a word boundary. If you are creating the file for the first time, this presents no problem. But if you are converting an existing ISAM file, two possible ways to approach the problem are:

Method 1

When describing the data base (with the DBSCHEMA operation in IMAGE/3000), define the entire record as one field. Add the regular record key and the dummy key to the end of the record. (If the key in the record is on a word boundary and contains an even number of bytes, it can be used and defined as is in the record.) The dummy key has the same value in all records.

In the RPG/3000 programs that reference this file, the record length specified in the File Description Specifications must reflect any changes made to the actual record length. Input Specifications remain the same, but you must change Output Specifications for records being added to the file to include the new key fields at the end of these records.

The disadvantage of this method is that you cannot fully utilize QUERY/3000 (IMAGE's Data Base Inquiry Facility) because all fields have not been described in the data base.

Method 2

Re-define the fields in the record so that they begin on a word boundary and have an even byte length. Then, add a dummy key (two bytes long) to the record. In existing RPG programs that reference this file, you must change Input and Output Specifications to reflect the new record format. (If the field names are the same in all RPG programs that reference this file, you may easily make this change by using the text editor, EDIT/3000.)

3. If you are using RPG programs to process the simulated ISAM file, then File Description Specification continuation records must be added to the programs. Modify the record length to include the dummy key or any other changes to the record length. If you move the key in the record so that it falls on a word boundary and is an even number of bytes long, then adjust the Input Specifications accordingly (if you wish to use the key as an input field).
4. Add the appropriate File Description Specification continuation records to the RPG/3000 programs that reference the file:
 - a. Include a **Data-Base Name Record** to specify:
 - Data-base name.
 - Open Mode. (All users simultaneously accessing a data base must use same open mode.)
 - Input/Output mode.
 - Initial chain key. (Only necessary if processing records sequentially or between limits.)
 - b. Add an **Item Name Record**. (ITEMXX if processing sequentially, ITEM if processing randomly, or both if processing between limits.)
 - c. Include a **Password (Level) Identification Record** if you have defined passwords for the data base.

- d. You may use a Data Set Name Record if the name in the Field Name Field (columns 7 through 14) of the File Description Specification record is not the data set name. This allows the same data set to be defined as two or more separate files.
 - e. You may use a Status Record if you wish to do your own error processing, and/or make use of other IMAGE status information (for instance, current, next, and previous record number).
5. For indexed sequential files of more than 65, 535 records, a somewhat different approach is required:
- a. To process a file of any size randomly, you need make no additional changes.
 - b. To process a large file sequentially, you must make additional modifications. Because IMAGE/3000 allows a maximum of 65,535 records on a sorted chain, there must be at least one dummy key for each of the 65,535 records. To allow you to process the entire file sequentially, the records with the second dummy key must logically follow all records with the first dummy key. For example, you might set up a file with 90,000 records as follows. To allow room for additions to the file, divide it into three parts so that all records whose sort key begins with A through H belong to the dummy key group "AH"; those with sort key I through Q belong to "IQ"; and the remainder belong to "RZ". Therefore, if the first dummy key group is processed first, the second is processed second, and so forth, the entire file is processed in sequence. Once the file is set up, you must modify all programs that process the file sequentially. Because RPG/3000 allows only one dummy key per file, you must make the dummy key available in a different way. You can do this through a record address file or by including a CHAIN calculation operation. An Input/Output Mode of C on the File Description Specification Record allows you to process all records in one chain. You can request processing of the records in physical sequence (Input/Output Mode 2) without special chains or RAF files. However, the physical sequence of the file may not correspond with the key sequence.
 - c. If you are processing the file between limits, you must use only one dummy key.

H-3. CODING SPECIFICATIONS FOR ISAM SIMULATION

RPG specifications for simulated ISAM operations through IMAGE resemble other RPG specifications for interfacing with IMAGE files; in general they contain the same entries, with the following exceptions:

H-4. File Description Specifications

To process an IMAGE file sequentially between limits, enter L in the Processing Mode Field (Column 28) of the File Description Specifications. (When you select this mode, you may use more than one key field to reference a data set from your program.) Supply the upper and lower limits of the file by using an RAF; these limits should be valid (exact-match) keys on the IMAGE file — otherwise, finding the beginning record could consume significant time. You must also supply both ITEM and ITEMXX continuation records.

H-5. File Description Specifications Continuation Records

ISAM simulation operations require special entries in two types of File Description Specifications Continuation Records:

- Data Base Name Records
- Item Name Records

H-6. DATA BASE NAME RECORDS. You may select either forward or backward simulated ISAM reads. This is done by making the appropriate entry in the Input/Output Mode Field (Column 67) of the Data Base Name Record:

- To select a forward sequential read, enter **S** in this field. There must be a master data set with a dummy key to which each record of the detail data set is chained. A DBFIND operation is done initially on the dummy key and the records are then read sequentially. A dummy key must be provided in Columns 68 through 69 of this record. An ITEMXX continuation record is required for this mode.
- To select a backward sequential read, enter **B** in this field. This option has the same requirements and produces the same effect as described immediately above, except that the records are read in reverse order.

For simulated ISAM reads, you must specify the initial chain key in the Initial Chain Key Field (Columns 68 through 69) of the Data Base Name Record. The records of a detail data set must all be chained together with this initial chain and a record in a master data set must point to the beginning of this chain. After the file is opened, a DBFIND operation is done on this initial chain, with sequential reads thereafter.

An additional continuation record is required which shows the item name of this dummy chaining key (ITEMXX) in Columns 54 through 59.

H-7. ITEM NAME RECORDS. You must enter the name of the dummy chaining key (ITEMXX) in the Record Type Field (Columns 54 through 59) of the Item Name Record. Then, in the Item Name Field (Columns 60 through 74) of this same record, you must enter the name of the key used to reference the file; this is the same name used to define the data base, but need not be the same name used in the RPG CHAIN operation.

H-8. APPLICATIONS

The following examples illustrate various applications of ISAM simulation with IMAGE files:

H-9. Creating and Listing an Indexed File

To illustrate how an RPG program interfaces with IMAGE to create a simulated indexed file, the program in Figure H-1 is presented. This program assumes that the data base has already been created (as shown in Figure 10-16, Section X). Fields that are particularly pertinent appear in boxes. The listing produced by this program appears in Figure H-2.

H-10. Processing An Indexed File Between Limits

The program in Figure H-3 illustrates the processing of records whose key values fall between the limit values read from a Record Address File (RAF). Each record on the RAF contains a starting value and an ending value. Processing starts with the first record which has a key field value that falls within the specified limits. (In this case, customer number is the key field.) When all the records in the specified range have been processed, another set of limits is read from the RAF and processing proceeds as before. Processing terminates when an end-of-file is reached on the RAF.

Note the following aspects of the source code to process between limits:

Line(s)

```

82      RAF File Specification (RAFLIMS1)
           Column 16:      R      (Record Address File designation)
           Columns 29-30:  06      (Key field length)
           Column 39:      E      (File Extension Specification is used)

84      Data File Specification (SEQDISK)
           Column 16:      P      (Primary File)
           Column 28:      L      (Limits — processing mode)

86      Data Base Name Record
           Column 66:      3      (Exclusive Access)
           Column 67:      S      (Sequential Read — processing mode)
           Columns 68-69:  CM      (Value of the dummy key)

88-90   Item Name Records
           CODE is the dummy key field name
           CUSTNO is the actual key field name
           (Both records are required for processing between limits)

105     RAF File Extension Specification
           Columns 11-18:  From File (RAF file)
           Columns 19-26:  To File (Data file)
    
```

The name of the data file as specified in the File Specification need not be the same as that used in the data base schema even though they both refer to the same file. The file name in columns 7-14 of the File Specification must conform to RPG/3000 file name requirements (a maximum of 8 characters; special characters are not allowed), whereas the data set name (columns 60-74 of the Data Set Name Record) is a legal IMAGE/3000 data set name (a maximum of 15 characters; special characters are allowed). In this

Simulating ISAM Functions With IMAGE/3000

example, SEQDISK is the RPG/3000 file name and CUSTF1 is the IMAGE/3000 data set name for the customer file. A listing showing the contents of the RAF appears in Figure H-4. Output from the program appears in Figure H-5.

```

10* PROG1
20* CREATE3
30* CREATE SEQUENTIAL CUSTOMER FILE (WITH ERROR LOGGING)
60%CONTROL MAP
70%CONTROL QUOTE='
80$TITLE "          CREATE3.  CREATE A SEQUENTIAL CUSTOMER FILE"
90$PAGE
100H
110FCARDIN IP F      80
FSEQDISK 0  F 256 128 06A1      2 DISC          U          CREAT1
F          <IMAGE CMASTR35CM          IMX1
F          <ITEM CUSTNO          IMX1
F          <LEVEL          IMX1
F          <DSNAMECUSTF1          IMX1
F          <STATUSXT          IMX1
140FCUSTLISTO F      132      OV      LP
150FCARDIN 011      1 D2          CX2
160I          2  7 CUSTNO
170I          15 34 NAME
180I          35 54 ADDR
190I          55 74 CITST
200I          8  74 CARD2
I          8  14 CDX1
I          15 74 CDX2
210I          021 30  1 D3
220I          2  7 CUSTNO
230I          8  47 CARD3
2400SEQDISK D      30
2500          2 'CM'
2600          CUSTNO      8
2800          CDX1      15
0          CDX2      76
2900          CARD3     116
3000
3200CUSTLISTH 210  1P          CX?
3300      OR      OV
3400          8 'CUSTOMER'
3500          22 'NAME'
3600          56 'ADDRESS'
3700      D  1      30
3800          CUSTNO      7
3900          NAME      30
4000          ADDR      52
4100          CITST     74

```

Figure H-1. ISAM File Creation Program (Source Code)

CUSTOMER	NAME	ADDRESS
136778	JONE'S VARIETY	14 S MAIN BEDROCK, TX
140001	MILPITAS GOODIES	111 N MAIN MILPITAS, CALIFORNIA
140002	KINGPIN HARDWARE	870 W SAN CARLOS SAN JOSE, CALIFORNIA
140003	ALUM ROCK 5 AND 10	2241 ALUM ROCK AVE SAN JOSE, CALIFORNIA
140004	ANGELS WHAT NICKS	4775 STORY RD SAN JOSE, CALIFORNIA
140005	BEAVER'S HARDWARE	9220 MOORPARK AV SAN JOSE, CALIFORNIA
140006	BIGGIES' AUTO SUPPLY	3047 EL CAMINO RL SANTA CLARA, CA
140007	RUILDERS SUPPLY	7365 PROSPECT RD SAN JOSE, CALIFORNIA
140008	CENTRAL SUPPLY	29082 STEVENS CREEK CUPERTINO, CA
140010	COAST TO COAST	1761 HOLLENBECK AVE SUNNYVALE, CA
140011	COUNTRY CLUB RUILDER	2469 ALMADEN RD SAN JOSE, CALIFORNIA
140012	DE ANZA HARDWARE	108R9 DE ANZA BLVD CUPERTINO, CA
140013	CAMPBELL HAROWARE	943 CAMDEN AVE CAMPBELL, CALIFORNIA
301628	JIM'S 5 AND 10	1103 FRANKLIN ST GLENCOE, MN
795246	SCHMIDT HARDWARE	100 1ST ST NW HILL CITY, MD

Figure H-2. ISAM File Creation Program (Output)

```

20* LIMITS1
30* LIST CUSTOMER FILE
50%CONTROL QUOTE='
60%TITLE "          LIMITS1, LIST CUSTOMER FILE"
70%PAGE
80H
82FRAFLIMS1IR F      12 06      EDISC
84FSEODISK IP F      128L06AI    DISC
86F
88F
90F
92F
94F
100FCLIST1 O V      132      OV      LP
105F RAFLIMS1SEODISK
110FSEODISK AA 10 1 CC 2 CM
120I
130I
140I
150I
160I
170I
180I
190I
200I
210I
220I
230I
240I
250I
260I
270I
280I
310I
320FCLIST1 M 1010 1P
330I
340I
350I
360I
370I
380I
410I
420I
430I
440I
450I
460I
470I
480I
490I
510I
520I
530I
540I
XLS
CLIST
<IMAGE CMASTR3SCM
<ITEMXXCODE
<ITEM CUSTNO
<LEVEL
<DSNAMECUSTF1
3 8 CUSTNO
9 10 STATE
11 12 CO
13 16 CITY
17 36 NAME
37 56 ADDR
57 76 CITST
77 78 CRLMST
79 800TRANS
81 862CHARGE
87 922PAY
93 982CREDIT
99 1042BAL
105 1102YTDLS
111 1162YTDNO
117 126 SLACK
127 128 DELETE
1 128 REC
64 'LIST THE CUSTOMER FILE'
65 'PROCESS BETWEEN LIMITS'
81 'WITH A TAG FILE'
CUSTNO 10
STATE 14
CO 18
CITY 23
NAME 45
ADDR 67
CITST 89
CRLMST 93
TRANS Z 97
CHARGEZ 105
PAY Z 113
CREDITZ 121
BAL Z 129
    
```

Figure H-3. Limits Program (Source Code)

```

30162A795246
13672A136728
140001140013
    
```

Figure H-4. Limits Program (Record Address File)

Simulating ISAM Functions With IMAGE/3000

H-11. Setting the Lower Limit with the SETLL Operation.

The program in Figure H-6 is the same as that in Figure H-3, except that instead of processing a data file between limits (as read from a Record Address File), the SETLL operation is used to specify a lower limit and processing continues to the end of the file. Note the following changes from Figure H-3.

Line(s)

- 84 Data File Specification
Column 16: Changed file designation from P (Primary) to D (Demand)
- 85, 105 Record Address File Specification and Extension were deleted
- 312 SETLL Calculation Specification
Columns 9-11: N20 — Set lower limit when indicator 20 is not on
Columns 18-27: 140005 — value of lower limit for key field CUSTNO
Columns 28-32: SETLL — operation
Columns 33-42: SEQDISK — name of data file
- 314 SETON Calculation Specification
Columns 9-11: N20 — Set indicator on if indicator 20 is not on
Columns 28-32: SETON — operation
Columns 54-55: 20 — indicator to be set on
- 316 READ Calculation Specification
Columns 10-11: 20 — Perform read when indicator 20 is on
Columns 28-32: READ — operation
Columns 33-42: SEQDISK — data file name
Columns 58-59: LR — Set Last Record indicator at the end of the file

On the first pass through the Calculation Specifications, the lower limit is set and the first read accesses the record with that key value. Each successive pass simply reads the next record until the end of file is reached.

The output from this program appears in Figure H-7.

LIST THE CJSTOMER FILE PROCESS BETWEEN LIMITS WITH A TAG FILE							
301620	MN	SC	GLE	JIM'S 5 AND 10	1103 FRANKLIN ST	GLENCOE, MN	15
795246	MD	SC	HIL	SCHMIDT HARDWARE	100 1ST ST NW	HILL CITY, MD	45
136720	TX	SC	RED	JONE'S VARIETY	14 S MAIN	BEDROCK, TX	35
140001	CA	SC	MIL	MILPITAS GOODIES	111 N MAIN	MILPITAS, CALIFORNIA	25
140002	CA	SC	SAN	KINGPIN HARDWARE	820 W SAN CARLOS	SAN JOSE, CALIFORNIA	45
140003	CA	SC	SAN	ALUM ROCK 5 AND 10	2241 ALUM ROCK AVE	SAN JOSE, CALIFORNIA	25
140004	CA	SC	SAN	ANGELS WHAT NICKS	4775 STORY RD	SAN JOSE, CALIFORNIA	25
140005	CA	SC	SAN	BEAVER'S HARDWARE	9220 MOORPARK AV	SAN JOSE, CALIFORNIA	15
140006	CA	SC	SAN	BIGGIES' AUTO SUPPLY	3047 EL CAMINO RL	SANTA CLARA, CA	45
140007	CA	SC	SAN	BUILDERS SUPPLY	7365 PROSPECT RD	SAN JOSE, CALIFORNIA	25
140008	CA	SC	CUP	CENTRAL SUPPLY	29082 STEVENS CREEK	CUPERTINO, CA	25
140010	CA	SC	SUN	COAST TO COAST	1761 HOLLENBECK AVE	SUNNYVALE, CA	35
140011	CA	SC	SAN	COUNTRY CLUB BUILDER	2469 ALMADEN RD	SAN JOSE, CALIFORNIA	35
140012	CA	SC	CUP	DE ANZA HARDWARE	10889 DE ANZA BLVD	CUPERTINO, CA	35
140013	CA	SC	CAM	CAMPRELL HARDWARE	943 CAMDEN AVE	CAMPBELL, CALIFORNIA	35

Figure H-5. Limits Program (Output)


```

10* PROG7
20* TSEILL
30* LIST CUSTOMER FILE
50*CONTROL QUOTE='
60*TITLE "          SFTL1, LIST CUSTOMER FILE"
70*PAGE
R0H
84FSEQDISK 10 F 128L06AI DISC XLS CLIST
86F
88F <IMAGE CMASTR3SCH
90F <ITEMXXCODE
92F <ITEM CUSTNO
94F <LEVEL
100FCLIST1 0 1 132 OV LP <DSNAMECUSTF1
110TSEQDISK AA 10 1 CC 2 CM
120T 3 8 CUSTNO
130T 9 10 STATE
140T 11 12 CO
150T 13 16 CITY
160T 17 36 NAME
170T 37 56 ADDR
180T 57 76 CITST
190T 77 78 CRLMST
200T 79 800TRANS
210T 81 862CHARGE
220T 87 922PAY
230T 93 982CREDIT
240T 99 10423AL
250T 105 1102YDLS
260T 111 1162YDNO
270T 117 126 SLACK
280T 127 128 DELETE
310T 1 128 REC
312C N20 140005 SETLLSFQDISK
314C N20 SETON 20
316C 20 READ SFQDISK LR
3200CLIST1 M 1010 LP
3300 64 'LIST THE CUSTOMER FILE'
3400 H 13 1P
3500 OH OV
3700 62 'USE SETLL OPERATION'
3800 D 10
4100 CUSTNO 10
4200 STATE 14
4300 CO 18
4400 CITY 23
4500 NAME 45
4600 ADDR 67
4700 CITST 89
4800 CRLMST 93
4900 TRANS Z 97
5100 CHARGEZ 105
5200 PAY Z 113
5300 CRFDITZ 121
5400 HAL Z 129

```

Figure H-6. SETLL Program (Source Code)

```

LIST THE CUSTOMER FILE
USE SETLL OPERATION

140005 CA SC SAN BEAVER'S HARDWARE 9220 MOORPARK AV SAN JOSE, CALIFORNIA 15
140006 CA SC SAN HIGGIES' AUTO SUPPLY 3047 EL CAMINO RL SANTA CLARA, CA 45
140007 CA SC SAN BUILDERS SUPPLY 7365 PROSPECT RD SAN JOSE, CALIFORNIA 25
140008 CA SC CUP CENTRAL SUPPLY 29082 STEVENS CREEK CUPERTINO, CA 25
140010 CA SC SUN COAST TO COAST 1761 HOLLENBECK AVE SUNNYVALE, CA 35
140011 CA SC SAN COUNTRY CLUB BUILDER 2469 ALMADEN RD SAN JOSE, CALIFORNIA 35
140012 CA SC CUP DE ANZA HARDWARE 10889 DE ANZA BLVD CUPERTINO, CA 35
140013 CA SC CAM CAMPBELL HARDWARE 943 CAMDEN AVE CAMPBELL, CALIFORNIA 35
301628 MN SC GLE JIM'S 5 AND 10 1103 FRANKLIN ST GLENCOE, MN 15
795246 MU SC HIL SCHMIDT HARDWARE 100 1ST ST NW HILL CITY, MO 45

```

Figure H-7. SETLL Program (Output)



If you wish to provide your own procedure for handling Indexed-Sequential Access Method (ISAM) files, as discussed in Section IV, paragraph 4-15, you must name this procedure R'ISAM. You must also link it to your RPG object program by substituting it for the R'ISAM procedure located in the RPG Segmented Library (SL) Segment named RPG'ISAM. Replacement must be made in the SL file that will be referenced at runtime — the MPE system library (SSL), the public library of your account (PSL), or your group library (GSL).

In order to reference the account or group library, you must specify the Keyword parameter LIB=P (for account) or LIB=G (for group) on the MPE :RUN command used to execute your program.

When you write the procedure R'ISAM, use the following format:

```
INTEGER PROCEDURE R'ISAM (fnum, type, farray);
VALUE type;
INTEGER type, fnum;
INTEGER ARRAY farray;
      :
      :
(Procedure Body)
      :
```

In this procedure, the *fnum* parameter indicates the file number returned when the file is opened by the FOPEN MPE intrinsic. (See *MPE Intrinsic Reference Manual*.) The *type* parameter indicates the operation to be performed with the ISAM file, as described in Table 4-1. The *farray* parameter specifies the name of an array that contains information used in support of the operation. The contents of this array depends on the contents of *type*, as shown in Table I-1. For instance, when *type* specifies that the file is to be closed, the array named in *farray* specifies:

- The file number that identifies the file (word 0).
- The disposition of the file after it is closed (word 1).
- The file security code (word 2).

Note: While *fnum* is set by the system, *type* and *farray* must be initialized by your program.

When the R'ISAM procedure is executed, it returns one of the values shown in Table I-2 to your program; this indicates the result of the procedure.

Table I-1. R'ISAM Type and Farray Parameter Values

Type		Farray	
Value of Parameter Set by User	Meaning	Array Word	Value/Meaning of Word Set by User
0	Open an R'ISAM file	0	Byte pointer to formal designator (RPG file name).
		1	FOPTIONS.
		2	AOPTIONS.
		3	Record size. (Negative = Byte Size).
		4	Byte pointer to device name (as specified in RPG File Description Specifications.)
		5	Forms message pointer (not used).
		6	Number of user labels.
		7	Blocking Factor.
		8	Number of buffers.
		9-10	File size.
		11	Maximum number of extents.
		12	Initial number of extents.
		13	File code.
		14	Key size (byte length).
		15	Key type (0 = Byte key, 1 = Packed Decimal Key).
			Correspond to FOPEN intrinsic parameters. (See MPE <i>Intrinsic Reference Manual</i> .)
			Special R'ISAM parameters.
1	Close an R'ISAM file	0	File number.
		1	Disposition (always zero).
		2	Security code (always zero).
2	Read the next record in the key sequence.	0	File number.
		1	Target address (word).
		2	Record length (bytes, negative, maximum).
3	Read the next physical record.	0	File number.
		1	Target address (word).
		2	Record length (bytes, negative, maximum).
4	Read by relative record number.	0	File number.
		1	Target address (word).
		2	Record length (bytes, negative, maximum).
		3-4	Record number.
5	Read by record key.	0	File number.
		1	Target address (word).
		2	Record length (bytes, negative, maximum).
		3	Key pointer (byte pointer).
		4	Generic key length (should be zero or same as in open)

Table I-1. R'ISAM Type and Farray Parameter Values (Continued)

Type		Farray	
Value of Parameter Set by User	Meaning	Array Word	Value/Meaning of Word Set by User
6	Write a record to the file; key is contained in record.	0	File number.
		1	Target address (word).
		2	Record length (byte, negative).
		3	Control (always zero).
7	(Not used)	(Not applicable)	(Not applicable.)
8	Write a record to the file; key is passed as parameter.	0	File number.
		1	Target address (word).
		2	Record length (byte, negative).
		3	Pointer to key (byte).
9	Write previously-read record; key-field cannot be modified.	0	File number.
		1	Target address (word).
		2	Record length (byte, negative).
10	Seek relative record number.	0	File number.
		1-2	Record number.
11	Seek record by key	0	File number.
		1	Key pointer (byte).
		2	Key length (zero, not currently used).
12	Get current record number.	0	File number
		1-2	Returned record number (used by RPG to save the current location of a sequential update file to which a record is being added).
13	Get value of current key.	0	File number.
		1	Hold area byte pointer; the key of the current record should be moved to this location.
14	Delete current record.	0	File number.

Table I-2. Result Values Returned By R'ISAM Procedure

Value Returned	Meaning
0	Normal processing; no error occurred.
+1	End-of-file encountered.
+2	Record not found.
+3	Duplicate record.
+4	No ISAM file processor present.
<0	Fatal error occurred. (RPG prints the error number and an error dump.)



RPG CODING AND PLANNING FORMS

APPENDIX

J

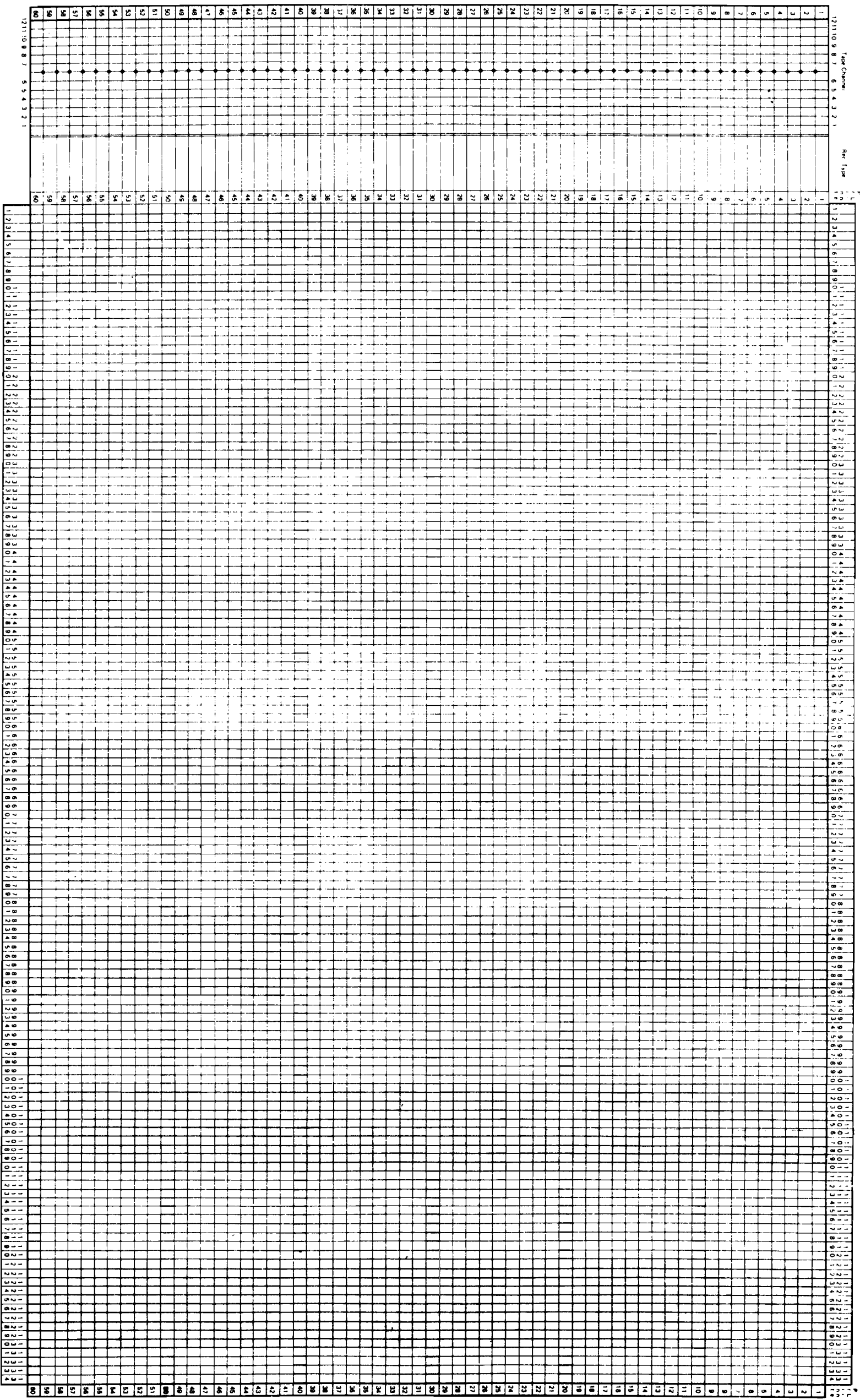
Blank copies of the RPG Specification Sheets, Translation Table and Alternate Collating Sequence Coding Sheet, and Printer Spacing Chart, are provided in this appendix.



Programmer

Program Title

Date





HEWLETT
PACKARD

RPG TRANSLATION TABLE AND ALTERNATE
COLLATING SEQUENCE CODING SHEET

Code	Character	Hexa- decimal Entry	Octal Entry	Replaces/ Replaced By
00000000	NUL	00	000	
00000001	SOH	01	001	
00000010	STX	02	002	
00000011	ETX	03	003	
00000100	EOT	04	004	
00000101	END	05	005	
00000110	ACK	06	006	
00000111	BEL	07	007	
00001000	BS	08	010	
00001001	HT	09	011	
00001010	LF	0A	012	
00001011	VT	0B	013	
00001100	FF	0C	014	
00001101	CR	0D	015	
00001110	SO	0E	016	
00001111	SI	0F	017	
00010000	DLE	10	020	
00010001	DC1	11	021	
00010010	DC2	12	022	
00010011	DC3	13	023	
00010100	DC4	14	024	
00010101	NAK	15	025	
00010110	SYN	16	026	
00010111	ETB	17	027	
00011000	CAN	18	030	
00011001	EM	19	031	
00011010	SUR	1A	032	
00011011	ES	1B	033	
00011100	FS	1C	034	
00011101	GS	1D	035	
00011110	RS	1E	036	
00011111	US	1F	037	
00100000	SP	20	040	
00100001		21	041	
00100010		22	042	
00100011		23	043	
00100100	S	24	044	
00100101		25	045	
00100110	R	26	046	
00100111		27	047	
00101000		28	050	
00101001		29	051	
00101010		2A	052	
00101011		2B	053	
00101100		2C	054	
00101101		2D	055	
00101110		2E	056	
00101111		2F	057	
00110000	0	30	060	
00110001	1	31	061	
00110010	2	32	062	
00110011	3	33	063	
00110100	4	34	064	
00110101	5	35	065	
00110110	6	36	066	
00110111	7	37	067	
00111000	8	38	070	
00111001	9	39	071	
00111010		3A	072	
00111011		3B	073	
00111100		3C	074	
00111101		3D	075	
00111110		3E	076	
00111111		3F	077	

Code	Character	Hexa- decimal Entry	Octal Entry	Replaces/ Replaced By
01010000	a	40	100	
01010001	A	41	101	
01010010	B	42	102	
01010011	C	43	103	
01010100	D	44	104	
01010101	E	45	105	
01010110	F	46	106	
01010111	G	47	107	
01011000	H	48	110	
01011001	I	49	111	
01011010	J	4A	112	
01011011	K	4B	113	
01011100	L	4C	114	
01011101	M	4D	115	
01011110	N	4E	116	
01011111	O	4F	117	
01100000	P	50	120	
01100001	Q	51	121	
01100010	R	52	122	
01100011	S	53	123	
01100100	T	54	124	
01100101	U	55	125	
01100110	V	56	126	
01100111	W	57	127	
01101000	X	58	130	
01101001	Y	59	131	
01101010	Z	5A	132	
01101011		5B	133	
01101100		5C	134	
01101101		5D	135	
01101110		5E	136	
01101111		5F	137	
01100000		60	140	
01100001		61	141	
01100010		62	142	
01100011		63	143	
01100100		64	144	
01100101		65	145	
01100110		66	146	
01100111		67	147	
01101000		68	150	
01101001		69	151	
01101010		6A	152	
01101011		6B	153	
01101100		6C	154	
01101101		6D	155	
01101110		6E	156	
01101111		6F	157	
01110000		70	160	
01110001		71	161	
01110010		72	162	
01110011		73	163	
01110100		74	164	
01110101		75	165	
01110110		76	166	
01110111		77	167	
01111000		78	170	
01111001		79	171	
01111010		7A	172	
01111011		7B	173	
01111100		7C	174	
01111101		7D	175	
01111110		7E	176	
01111111		7F	177	

Code	Character	Hexa- decimal Entry	Octal Entry	Replaces/ Replaced By
10000000	K0	80	200	
10000001	K1	81	201	
10000010	K2	82	202	
10000011	K3	83	203	
10000100	K4	84	204	
10000101	K5	85	205	
10000110	K6	86	206	
10000111	K7	87	207	
10001000	K8	88	210	
10001001	K9	89	211	
10001010	K10	8A	212	
10001011	K11	8B	213	
10001100	K12	8C	214	
10001101	K13	8D	215	
10001110	K14	8E	216	
10001111	K15	8F	217	
10010000	K16	90	220	
10010001	K17	91	221	
10010010	K18	92	222	
10010011	K19	93	223	
10010100	K20	94	224	
10010101	K21	95	225	
10010110	K22	96	226	
10010111	K23	97	227	
10011000	K24	98	230	
10011001	K25	99	231	
10011010	K26	9A	232	
10011011	K27	9B	233	
10011100	K28	9C	234	
10011101	K29	9D	235	
10011110	K30	9E	236	
10011111	K31	9F	237	
10100000	N0	A0	240	
10100001	N1	A1	241	
10100010	N2	A2	242	
10100011	N3	A3	243	
10100100	N4	A4	244	
10100101	N5	A5	245	
10100110	N6	A6	246	
10100111	N7	A7	247	
10101000	N8	A8	250	
10101001	N9	A9	251	
10101010	N10	AA	252	
10101011	N11	AB	253	
10101100	N12	AC	254	
10101101	N13	AD	255	
10101110	N14	AE	256	
10101111	N15	AF	257	
10110000	N16	80	260	
10110001	N17	81	261	
10110010	N18	82	262	
10110011	N19	83	263	
10110100	N20	84	264	
10110101	N21	85	265	
10110110	N22	86	266	
10110111	N23	87	267	
10111000	N24	88	270	
10111001	N25	89	271	
10111010	N26	8A	272	
10111011	N27	8B	273	
10111100	N28	8C	274	
10111101	N29	8D	275	
10111110	N30	8E	276	
10111111	N31	8F	277	

Code	Character	Hexa- decimal Entry	Octal Entry	Replaces/ Replaced By
11000000	N32	C0	300	
11000001	N33	C1	301	
11000010	N34	C2	302	
11000011	N35	C3	303	
11000100	N36	C4	304	
11000101	N37	C5	305	
11000110	N38	C6	306	
11000111	N39	C7	307	
11001000	N40	C8	310	
11001001	N41	C9	311	
11001010	N42	CA	312	
11001011	N43	CB	313	
11001100	N44	CC	314	
11001101	N45	CD	315	
11001110	N46	CE	316	
11001111	N47	CF	317	
11010000	N48	D0	320	
11010001	N49	D1	321	
11010010	N50	D2	322	
11010011	N51	D3	323	
11010100	N52	D4	324	
11010101	N53	D5	325	
11010110	N54	D6	326	
11010111	N55	D7	327	
11011000	N56	D8	330	
11011001	N57	D9	331	
11011010	N58	DA	332	
11011011	N59	DB	333	
11011100	N60	DC	334	
11011101	N61	DD	335	
11011110	N62	DE	336	
11011111	N63	DF	337	
11100000	G0	E0	340	
11100001	G1	E1	341	
11100010	G2	E2	342	
11100011	G3	E3	343	
11100100	G4	E4	344	
11100101	G5	E5	345	
11100110	G6	E6	346	
11100111	G7	E7	347	
11101000	G8	E8	350	
11101001	G9	E9	351	
11101010	G10	EA	352	
11101011	G11	EB	353	
11101100	G12	EC	354	
11101101	G13	ED	355	
11101110	G14	EE	356	
11101111	G15	EF	357	
11110000	G16	F0	360	
11110001	G17	F1	361	
11110010	G18	F2	362	
11110011	G19	F3	363	
11110100	G20	F4	364	
11110101	G21	F5	365	
11110110	G22	F6	366	
11110111	G23	F7	367	
11111000	G24	F8	370	
11111001	G25	F9	371	
11111010	G26	FA	372	
11111011	G27	FB	373	
11111100	G28	FC	374	
11111101	G29	FD	375	
11111110	G30	FE	376	
11111111	G0	FF	377	

RPG Screen Interface

SECTION

K

RPG Screen Interface Command Key Indicator Usage

Function Key F1 followed by:	Sets on Command Key Indicator:
1 (one)	KA
2 (two)	KB
3 (three)	KC
4 (four)	KD
5 (five)	KE
6 (six)	KF
7 (seven)	KG
8 (eight)	KH
9 (nine)	KI
0 (zero)	KJ
— (underscore)	KK
= (equals)	KL
! (exclamation point)	KM
@ (commercial at)	KN
# (number sign)	KP
\$ (dollar sign)	KQ
% (percent sign)	KR
^ (circumflex)	KS
& (ampersand)	KT
* (asterisk)	KU
((opening parenthesis)	KV
) (closing parenthesis)	KW
- (hyphen/minus)	KX
+ (plus)	KY



MPE USER MESSAGE SYSTEM

APPENDIX

L

MPE USER MESSAGE SYSTEM

The MPE User Message System consists of user message catalogs and a program (MAKECAT.PUB.SYS) for building message catalogs. Within the RPG Subsystem, the user message catalogs are accessed by the **MSG** and **DSPLM** operation codes and the RPG Screen Interface.

MESSAGE CATALOG

A message catalog **must be a standard editor-type file** containing sets of messages. That is, a numbered file which contains 80 byte records in a fixed record format. The sets serve to break the catalog into functional sections. After a message file is created, the MAKECAT.PUB.SYS program is used to build a catalog that is readable by the message system. This catalog file can still be texted into the editor, but it now contains a directory (written as a user label by MAKECAT.PUB.SYS).

Continuation of a message is indicated by “%” or “&” at the end of a line. The “%” symbol indicates that the message is continued and that a carriage return, line feed will be embedded. The “&” symbol indicates that the message is continued on the same line with no carriage return, line feed. For use in RPG applications, the following restrictions apply to the length of the message:

- MSG Operation** — The length of the message should not exceed 256 characters (the maximum field length of an alphanumeric field).
- DSPLM Operation** — The length of the message should not exceed 249 characters (the message is preceded by **DSPLY** and two blanks).

Message sets are indicated by “\$SET=n” starting in column 1 (the rest of the line is treated as a comment). The maximum value for n is 62. Comments can be inserted in the catalog by placing “\$” in column 1 of a line. Message numbers are positive integers, need not be contiguous, but must be in ascending order. After processing by the program MAKECAT.PUB.SYS, the catalog file contains records of 80 bytes, blocked 16, in 32 extents.

A sample user message catalog follows.

```
$SET=1
0001 Product Sales Analysis — Press F1 — Current Month Detail
0002 — Press F2 — Current Month Summary
0003 — Press F3 — Year-to-date Detail
0004 — Press F4 — Year-to-date Summary
0005 — Press F5 — Comparative Detail
0006 — Press F6 — Comparative Summary
0007 — Press F7 — Month-end processing
0008 — Press F8 — Year-end processing
0009 — Press the Function Key(s) for the
      desired version(s) of this report
```

\$SET=2

\$ THIS MESSAGE SET IS USED FOR THE REPORT HEADINGS

0001 PRODUCT SALES ANALYSIS

0004 DETAIL PRODUCT REPORT — MONTH OF:

```
0006 PRODUCT NUMBER   DESCRIPTION   QUANTITY SOLD   &
TOTAL SALES-CURRENT MONTH   TOTAL SALES-YEAR-TO-DATE
```

\$SET=3

\$ ERROR MESSAGES

0001 DISCONTINUED PRODUCT

0002 PRODUCT NOT ON FILE

0003 NO SALES CURRENT MONTH

0004 NEW PRODUCT — NO HISTORY

NOTE: In the example, message 6 of \$SET=2 is continued onto a second line.

MAKECAT.PUB.SYS PROGRAM

The program MAKECAT.PUB.SYS is used to build message catalogs. The program's input file has the formal designator INPUT. The program reads from the input file and builds a temporary file (format designator CATALOG). It also renames any old temporary CATALOG, CATnn, using an archival numbering scheme (i.e., CAT1,CAT2, etc.).

NOTE: MAKECAT.PUB.SYS has entry points (BUILD, DIR, HELP) which should not be used without full understanding of the MPE Message Catalog System. Refer to the MPE Intrinsic Reference Manual for a full discussion of the MAKECAT.PUB.SYS program.

EXAMPLE: To use MAKECAT.PUB.SYS to build a user message catalog.

```
:FILE INPUT=MYMSG.S.PAYROL (Note: MYMSG was previously built using the editor)
```

```
:RUN MAKECAT.PUB.SYS
```

```
** VALID MESSAGE CATALOG (Printed if no errors in MYMSG)
```

```
:SAVE CATALOG
```

```
:RENAME CATALOG,MYMSG.PUB.PAYROL
```

A

A records, 5-24-5-26
 Action codes, V/3000 interface
 definition, 13-6, 13-7
 relation to V/3000, 13-8
 Actions, V/3000 interface
 description, 13-5
 overview, 13-2, 13-4
 ADD operation, 8-12, 8-23
 ADDROUT files, 4-13, 4-16
 Alphanumeric format, 7-26
 Alphanumeric literal, 8-8, 8-11
 Alphanumeric/numeric field conversion, 7-28-7-29
 Alternate Collating Sequence Field (of Control Record Specification), 3-14-3-18, 3-44, F-1
 Alternate Collating Sequence Table, C-2
 Alternating tables, 5-10-5-12, 5-17
 ALTSEQ records, 3-14-3-16
 American Standard Code for Information Interchange (ASCII) Collating Sequence, 3-14-3-18, 3-44, A-2, A-4, G-1-G-11
 AN lines, 8-4-8-5
 AND lines, 7-20-7-21, 9-15, 9-17, 9-18
 Apostrophes (as delimiters), 12-8, A-2
 Appending records to files, 4-26-4-27
 Applications of RPG/3000, 1-1
 Arithmetic operations, 8-22-8-24
 Arrays
 Compile-Time, 5-7, 5-22-5-24, A-5
 Creating, 5-22, 5-23
 Defining, 5-23
 Execution-time, 5-7, 5-29
 General discussion of, 3-19, 5-1, 5-7-5-33
 Indexes, 5-33, 7-32
 Loading, 5-24-5-29
 Names, 5-9-5-10, 5-20-5-21, 7-31-7-32
 Pre-execution time, 5-7, 5-22-5-23, 5-30
 Related, 5-10-5-14, 8-52
 Searching, 5-30, 8-52-8-54
 Using, 5-32
 Writing to output files, 5-33
 ASCII Collating Sequence, 3-14-3-18, 3-44, A-2, A-4, G-1-G-11
 ASCII Options, 4-32

B

BADFLD, V/3000 interface, definition, 13-7
 BATCH continuation option, 4-45, F-8
 Batch file,
 V/3000 interface, 4-45, 13-2, F-8
 write data to, V/3000 interface, 13-7
 Batch jobs samples, 10-13-10-20, 11-13-11-14

Batch record,
 delete, V/3000 interface, 13-7
 READ, V/3000 interface, 13-7
 BEGSR operation, 8-13, 8-38
 Binary format, 7-24
 Binary look-up, 3-19
 BITOFF operation, 8-12, 8-42
 BITON operation, 8-12, 8-42
 Bits,
 Setting off, 8-12, 8-42
 Setting on, 8-12, 8-42
 Blank After Field (of Output Specifications), 3-23-3-24, 9-29, 9-44, F-23
 Blank-zero indicators, 3-23-3-24, F-2
 Block Length Field (of File Description Specification), 4-9, 4-47, F-5
 Body portion of edit word, 9-36, 9-38-9-40
 Branching and exiting operations, 8-34-8-43
 BREAK KEY,
 disable for V/3000 interface, 13-3
 enable for V/3000 interface, 13-3
 Browse data, V/3000 interface, 13-7
 BUFCHK
 Current data checking, 3-18
 No-read checking, 3-18
 Update-protect checking, 3-18, 4-31
 Defaults, general group fields, 3-18, 3-44, 4-31
 option, 3-18, 3-44, 4-31
 Buffer,
 find field in V/3000 interface, 13-7
 write, data and field, V/3000 interface, 13-7
 BYPASS, options, 4-32

C

Calculation Specifications,
 Coding rules, 8-1-8-89, F-17-F-20
 Purposes, 1-4
 Sample, 10-8
 Capabilities and advantages of RPG/3000, 1-1-1-2
 Card read/punch/interpreter, 4-32, 7-22-7-24, 9-6, 10-44-10-45, A-1
 Carriage-Control Type Field (of Control Record Specification), 3-44, F-3
 CHAIN operation, 8-13, 8-67-8-73, 10-23-10-26
 Chained files, 4-6, 5-3-5-8, 8-67-8-74
 Chaining, 7-39, 7-46-7-50, 8-12, 8-67-8-74, 10-23-10-26
 Chaining Code Identifier Field (of File Extension Specification), 5-3-5-8, 10-23-10-26, F-10
 Chaining Field Record Sequence Field (of File Extension Specification), 5-3-5-8, F-10
 Chaining fields, 7-46-7-50
 Chaining files, 5-3-5-8, 8-67-8-74, 10-23-10-26
 Chaining Table, C-3, C-19

INDEX (continued)

- Change, next form, V/3000 interface, 13-6
 - Channel Number/OL/FL Field (of Line Counter Specification), 6-3, 6-5-6-7, F-13
 - Channel Number Option (of Line Counter Specification), 6-1-6-5
 - Character Subfield (of Input Specification), 7-19, F-15
 - CHGNXT, definition, V/3000 interface, 13-6
 - CHRONOLOGICAL OPTION, 4-45
 - CLOSE operation, 8-13, 8-78
 - CLRMSG, definition, V/3000 interface, 13-7
 - Coding and Planning Forms, Blank, 1-4, J-1
 - Coding lines, 2-1, 2-3
 - Collating sequence
 - Alternate, 3-14-3-20, 3-44, C-2, F-1
 - ASCII, 3-14-3-18, 3-44, 4-30, A-2, A-4, G-1-G-11
 - EBCDIC, 3-14, 3-16, 4-31, A-2, A-4
 - Collect mode, resume, V/3000 interface, 13-7
 - Combined files, 4-6, 7-1
 - Commands
 - Compiler subsystem, 1-7, 12-1-12-17
 - MPE/3000, 11-1-11-29
 - Command Keys, 7-10-7-11, 9-15, K-1
 - Command Key Indicators, 7-9, 7-10, 7-51, 8-6, 8-19, 9-15, F-13, F-17, F-18, F-20, F-22, K-1
 - Command Record Indicator Field (common to all specifications), 2-4, 2-5
 - Comment Record Indicator, 2-4
 - Comments Field (of Calculation Specification), 8-22, 8-89, F-20
 - Comments Field (of File Extension Specification), 5-19, F-10
 - Comments Field (of Output Specification), 9-41, F-24
 - COMP operation, 8-13, 8-31, 8-33
 - Compare and test operations, 8-31-8-33
 - Compatibility with other systems, 1-11, A-1-A-5
 - Compilation,
 - Arranging source program for, 1-3-1-7, 11-1
 - Conditional, 12-9-12-11
 - Compile-Time Tables and Arrays, 5-7, 5-22-5-26, A-5
 - Compiler listing output, 11-14-11-16
 - Compiler secondary entry points, 11-29
 - Compiler subsystem commands,
 - Coding rules for, 12-5-12-17
 - Comments in, 12-3
 - Continuation records for, 12-3-12-4
 - Effects of, 12-4
 - Parameters for, 12-2
 - Purpose of, 1-7, 12-1
 - Summary of, 12-5
 - Syntax and format, 12-1
 - Compiling programs, 11-8-11-12
 - Complex edit codes, 9-26-9-28
 - Conditional compilation, 12-9-12-11
 - Constant/Edit Word Field (of Output Specification), 9-35-9-41, 9-44
 - Constants, 9-35-9-41, 9-44
 - Continuation characters, MPE, 11-5
 - Continuation Code Field (of File Description Specifications Continuation Records), 4-29
 - Continuation Records for File Description Specifications, 4-29
 - Control breaks, 7-38-7-39, 8-1, D-1
 - Control fields, 7-38-7-39, 7-51
 - Control groups, 8-1
 - Control Level Field (of Calculation Specification), 8-2-8-4, 8-89, F-18
 - Control Level Field (of Input Specification), 7-38-7-39, 7-51, F-16
 - Control level indicators, 7-11, 7-38-7-39
 - Control Record Specification
 - Coding rules, 3-1-3-44, F-1-F-3
 - Purpose, 1-4, 2-2, 2-5-2-6, 3-1
 - Sample, 10-2-10-3
 - V/3000 interface, 13-18-13-19
 - Conversion, alphanumeric/numeric field, 7-28-7-29
 - Conversion routines, 5-10, 5-11, 8-80-8-82
 - Copylib records
 - Inserting, 14-1
 - Modifying, 14-2-14-4
 - COERR, definition, V/3000 interface, 13-6
 - Correct field errors, V/3000 interface, 13-6
 - Cross-Reference Listing, 1-4, 1-10, 3-34-3-35, 11-18-11-21
 - Cross-Reference Listing Field (of Control Record Specification), 3-34-3-35, 3-37, 3-44, F-2
- ## D
- Data,
 - display at terminal, V/3000 interface, 13-6
 - read from buffer, V/3000 interface, 13-5
 - write to batch file, V/3000 interface, 13-7
 - write to buffer, V/3000 interface, 13-7
 - Data Base Management (IMAGE), 4-36-4-38
 - Data buffer,
 - read, V/3000 interface, 13-7
 - write to, V/3000 interface, 13-7
 - Data entry, V/3000 interface, 13-4
 - Data Format Fields (of File Extension Specifications), 5-14, 5-18, 5-20, 5-21, F-11
 - Data Format Field (of Input Specification), 7-24-7-29, 7-52, F-16
 - Data Fields, 9-24, 9-44
 - Data Structures, 7-16, 7-35-7-37, 7-57, F-15
 - DBFIND, 8-69
 - DBGET, 8-69
 - Debug Field (of Control Record Specification), 3-3, 3-44
 - DEBUG Operation, 3-3, 3-44, 8-13, 8-78, C-1, F-19
 - Debugging techniques, C-1-C-20
 - Decimal Positions Field (of Calculation Specification), 8-16, 8-89, F-20

Decimal Position Fields (of File Extension Specification), 5-14-5-15, 5-18-5-19, 5-21, F-11, F-12, F-16
 Decimal Position Field (of Input Specifications), 7-31, 7-57, F-20
 Demand files, 4-7
 Detail record/line, 9-5, 9-46, 9-47
 Detail-Time Calculations, 8-1, D-1-D-3, E-2, E-5
 Detail-Time Output, D-1-D-3, E-1, E-3
 Device Class Name Field (of File Description Specifications), 4-22-4-23, A-2, F-6
 Diagnostic messages, 1-11, E-1-B-79
 Digit portion of character, 7-20
 Disc extents, 4-27
 Disc Labels Field (of File description Specifications), 4-47, F-6
 Display,
 current form, V/3000 interface, 13-6
 data at terminal, V/3000 interface, 13-6
 message, V/3000 interface, 13-6
 Display files, 4-5, 4-28
 DIV operation, 8-13, 8-23
 Dollar-Sign Substitute Field (of Control Record Specification), 3-30, 3-44
 Domestic print format, 3-12
 Downloading, Forms, 13-3
 DSPLM operation, 8-13, 8-57, 8-60-8-62, 8-63-8-65, F-19, L-1
 DSPLY operation, 8-13, 8-57, 8-60-8-62, 8-63-8-65, F-17
 Dump, Error, C-1-C-20
 Duplicate Key Option, 4-44-4-45

E

EBCDIC zone/digit tests, 3-21
 EBCDIC or EBCDIK Collating Sequence, 3-14, 3-16, 4-31, 4-32-4-33, A-2, A-4, F-1, F-8
 EBCDIC and EBCDIK options, 4-31, 4-32, F-1, F-8
 Edit,
 definition, V/3000 interface, 13-6
 fields, V/3000 interface, 13-6
 Edit Code Field (of Output Specification), 9-25-9-28, 9-44, F-23
 Edit Codes,
 Complex, 9-26-9-28
 General, 3-30, 9-25
 Simple, 9-26, 9-44
 Edit words, 3-30, 9-36-9-41
 End-of-File Field (of File Description Specification), 4-7, F-4
 End Position Field (of Output Specification), F-23
 ENDSR operation, 8-13, 8-40
 Entries Per Record Fields (of File Extension Specification), 5-10-5-11, 5-20, 5-21, F-11

Entries per Table/Array Field (of File Extension Specification), 5-12, 5-20, 5-21, F-11
 Entry Length Fields (of File Extension Specification), 5-13, 5-17-5-18, 5-20, 5-21, F-11
 Equal Subfield (of Calculation Specification), 8-19
 ERPGC operation, 8-13, 8-82
 Error
 dump, V/3000 interface, 13-14
 handling, V/3000 interface, 13-2, 13-14-13-15
 in fields, V/3000 interface, 13-7
 messages, status, V/3000 interface, 13-7
 Error dump, C-1-C-13
 Error Log Field (of Control Record Specification), 3-39, 3-44, F-3
 Error messages, B-1-B-79
 ERROR options, 4-30
 Error processing, 4-30
 Error Response Field (of Control Record Specification), 3-40-3-41, 3-44, F-3
 Event codes, V/3000 interface, description, 13-4
 Events, V/3000 interface overview, 13-2
 Exception record/line, 9-5
 EXCPT operation, 8-13, 8-56
 Executing programs, 11-10-11-14
 Execution-Time Arrays, 5-7, 5-30
 EXIT operation, 8-13, 8-42
 Exiting operations, 8-36-8-38
 Expansion portion of edit word, 9-41
 EXSR operation, 8-13, 8-40
 EXTVCV operation, 8-13, 8-82
 Extended Binary Coded Decimal Interchange Code (EBCDIC) Collating
 Sequence, 3-14, 3-16, 4-32-4-33, A-2, A-4
 Extension Code Field (of File Description Specification), 4-20, F-6
 Extents, disc, 4-27, 4-47
 Extents Field (of File Description Specifications), 4-27, 4-47, F-9
 External conversion routines, 8-82
 External subroutines, 8-41-8-48, A-3

F

Factor 1 Field, 8-8-8-12, F-18
 Factor 2 Field, 8-8-8-12, 8-27, F-18
 Fetch overflow, 6-1, 9-8
 Fetch Overflow Field (of Output Specification), 9-8-9-9, 9-44,
 Field, edit failed, V/3000 interface, 13-7
 Field conversion, alphanumeric/numeric, 7-28-7-29
 Field Description Fields (of Input Specifications), 7-1, 7-24-7-56
 Field Description Fields (of Output Specifications), 9-1, 9-19-9-29

INDEX (continued)

- Field Indicator Field (of Input Specification), 7-54-7-55, F-17
- Field indicators, 8-5-8-10
- Field Length Field (of Calculation Specification), 8-16, 8-89, F-20
- Field Name Field (of Input Specification), 7-31-7-34, F-16
- Field Name Field (of Output Specification), 9-19, F-22
- Field names, 7-31-7-34, 7-57, F-16
- Field Record Relation Field (of Input Specification), 7-51-7-52, F-17
- Field translation, partial, 1-2, 4-31, F-8
- Figurative constants, 8-11
- Figurative literals, 8-11-8-12
- File Addition Field (of File Description Specifications), 4-26-4-27, F-5
- File Addition Field (of Output Specification), 9-5-9-6, 9-44, F-21
- File Conditioner Field (of File Description Specifications), 4-28, 4-47
- File Description Specifications
 - Coding rules, 4-1-4-47, F-4-F-10
 - Purpose, 1-4, 4-1
 - Sample, 10-2
 - V/3000 interface, 13-18, 13-19
- File Designation Field (of File Description Specifications), 4-6, 4-47, F-4
- File Extension Specifications
 - And extension code, 4-20-4-22
 - Coding rules, 5-1-5-34, F-10-F-12
 - Purpose, 1-4, 5-1
 - Sample, 10-4-10-5
- Filename Field (of Line Counter Specification), 6-1-6-3, 6-5, F-13
- File Name Field (of File Description Specification), 4-3, F-4
- File Name Field (of Input Specification), 7-1-7-3, F-14
- File Name Field (of Output Specification), 9-3, F-21
- File Operations, 8-54-8-69
- File Organization/Additional I/O Area Field (of File Description Specifications), 4-15-4-17, 4-47, F-5
- File Sharing, 4-35-4-36
- File Table, C-3, C-14-C-18
- File Translation Field (of Control Record Specification), 3-25-3-31, 3-44, F-2
- File Translation Tables, C-2
- File Type Field (of File Description Specifications), 4-5-4-6, F-4
- Files,
 - ADDROUT, 4-12-4-16
 - Chained, 4-6-4-7, 5-3-5-10, 8-67-8-73
 - Chaining, 5-3-5-10, 7-46-7-50, 8-12, 8-67-8-73
 - Combined, 4-5, 7-1
 - Demand, 4-7
 - Display, 4-5
 - IMAGE/3000, 1-2, 3-13, 4-4, 4-6, 4-12-4-17, 4-19, 4-27, 4-29, 4-35-4-44, 8-67, 9-6, 10-36-10-44, F-5
 - Input, 4-5, 7-1
 - KSAM, 1-2, 3-13, 4-3, 4-4, 4-6, 4-12-4-17, 4-19-4-20, 4-27, 4-29, 8-67-8-68, 9-5-9-6, 10-20-10-36, F-5, F-9, F-10
 - Listfile, 11-27
 - Masterfile, 3-38, 11-25, 12-13-12-17
 - Matching, 4-8, 7-41-7-44
 - Multifile, processing of, 4-7, 7-41-7-44
 - Names, 4-3, 6-2-6-3, 7-1-7-3, 9-3, F-4, F-10, F-13, F-14, F-21
 - Newfile, 11-25, 12-13-12-17
 - Object program, 1-7, 1-11
 - Output, 4-5, 9-1
 - Pre-execution table/array, 4-5, 4-6
 - Primary, 4-6, 7-41-7-44
 - Release, F-21
 - RSAM, 1-2, 3-13, 4-12-4-17, 4-20, 8-67, F-5, F-10
 - Secondary, 4-6, 7-41-7-44
 - Single-file processing, 7-44
 - Textfile, 11-8-11-13, 11-25, 12-13-12-17
 - Update, 4-5, 7-1
 - Uslfile, 1-7, 2-6, 11-8-11-13, 12-7
- Find field in buffer, V/3000 interface, 13-7
- FINDJCW, 8-87
- FINDJW, 8-87
- Finish,
 - definition, V/3000 interface, 13-7
 - phase processing, V/3000 interface, 13-7
- First-page indicator, 7-9, 7-12, 7-51, 7-54, 8-6, 8-19, 9-14, 9-17
- First record number, 3-13
- FNUM, 8-13, 8-88
- FORCE operation, 8-13, 8-57
- Form,
 - display at terminal, V/3000 interface, 13-6
 - initialize, V/3000 interface, 13-6
 - print with, V/3000 interface, 13-6
- Form length, 6-1, 6-6
- Form positioning Field (of Control Record Specification), 3-22, 3-44, F-2
- Form Type Field
 - Common to all specifications, 2-4
 - Of Calculation Specification, 8-2, F-17
 - Of Control Record Specification, 3-1, 3-3, F-1
 - Of File Description Specification, 4-3, F-4
 - Of File Extension Specification, 5-1, F-10
 - Of Input Specification, 7-1, F-14
 - Of Line Counter Specification, 6-2, 6-5, F-12-F-14
 - Of Output Specification, 9-1, F-20-F-24
- FORMS continuation option, 4-45, F-8
- Forms Downloading, 13-3
- Format
 - Alphanumeric, 7-26
 - Binary, 7-27
 - Domestic Print, 3-12
 - European Print, 3-12
 - Packed Decimal, 7-27

United Kingdom, 3-12, 3-13
 Unpacked Decimal, 7-26
 Forms file, V/3000 interface, 13-2
 From Filename Fields (of File Extension Specification), 5-5, 5-20, 5-21, F-10
 From Field Position Field (of Input Specification), 7-29, 7-57, F-16
 Function Key Indicators, 7-8, 7-10, 7-51, 7-54, 8-6, 8-19, 8-57, 8-60, 8-63-8-64, 9-15, F-15, F-17, F-18, F-20, F-22
 Function Keys, 8-60, 9-15
 Function keys, read, V/3000 interface, 13-5

G

General indicators, 7-9-7-10, 7-51-7-54, 8-6, 8-8, 8-19, 9-14-9-15, C-2
 Get next form
 GETDTA, definition, V/3000 interface, 13-7
 GETFLD, definition, V/3000 interface, 13-7
 GETNXT, definition, V/3000 interface, 13-6
 V/3000 interface, 13-6
 GOTO Operation, 8-13, 8-34-8-36
 Group Sequence Field (of Input Specification), 7-3-7-4, F-14

H

Half-Adjust Field (of Calculation Specification), 8-17-8-18, 8-89, F-20
 Halt indicators, 7-13, 7-51, 7-54, 8-6, 8-7, 8-19, 9-14, 9-17
 Header Portion of Spread Record, 7-13-7-16
 Header Specification, (see Control Record Specification)
 Heading record/line, 9-4-9-5, 9-45
 High Subfield (of Calculation Specifications), 8-19

I

IMAGE/3000
 Data Base Management, 4-36-4-40
 Files, 1-2, 3-13, 4-4, 4-6, 4-12-4-17, 4-19, 4-27, 4-29, 4-35-4-44, 8-67, 9-6, 10-36-10-44, F-5
 Simulation of ISAM functions, H-1-H-9
 Table, C-18
 INDEX/3000 files, 1-2, 3-13, 4-4, 4-6, 4-12-4-17, 4-19, 4-27, 4-29, 4-35-4-44, 8-67, 9-6, 10-36-10-44, F-5
 Indexed Sequential Access Method, H-1-H-9
 Indexes, array, 5-32, 7-32, 7-33
 Indicator and bit-setting operation, 8-48-8-50
 Indicator Setting Field (of Control Specification), 3-23, 3-44, F-2
 Indicators
 Blank-Zero, 1-2, 3-23-3-24, F-2
 Control Level, 7-11, 7-38-7-39, 7-51-7-52, 8-2, 8-4-8-7, 9-14, 9-15

Field, 8-7, 8-10
 First-Page, 7-9, 7-12, 7-51, 7-54, 8-6, 8-19, 9-14, 9-17
 General, 7-8, 7-10, 7-51, 7-54, 8-6, 8-9, 8-19, 9-14-9-15, C-2
 Halt, 7-13, 7-51, 8-6, 8-19, 9-14
 H0-H9 (see halt)
 Last record, 7-11, 7-51, 7-54, 8-2, 8-4, 8-6, 8-7, 9-14, 9-15
 LR, (see Last record)
 Matching record, 7-11, 7-51, 7-54, 8-6, 8-7, 8-19, 9-14, 9-15
 MR (see Matching Record)
 Output, 9-13-9-14, 9-44, F-21
 Overflow, 4-18, 4-47, 6-1, 7-12, 7-51, 8-6, 8-19, 9-14, 9-15, 9-17
 OA-OG, OV (see Overflow)
 Record, 7-6, 7-54, F-15
 Resulting, 8-17-8-19, 8-20, 8-89, F-20
 User, 4-28, 7-51, 7-52, 8-19, 9-14, 9-17
 01-99, (see general)
 1P (see First-Page)
 Indicators Field (of Calculation Specification), 8-5-8-8, 8-89, F-18
 INIT, V/3000 interface, 13-6
 Initialize form, V/3000 interface, 13-6
 Input, record formats, V/3000 interface, 13-9-13-10
 Input files, 4-5, 7-1, 10-6
 Input Sequence Check Field (of File Description Specification), 4-8-4-9, 4-47, F-4
 Input Specifications,
 Coding rules, 7-1-7-57
 Purpose, 1-4, F-14-F-17
 Sample, 10-4, 10-6
 V/3000 interface, 13-20
 Interactive Session Mode, 3-4, 3-7-3-9
 Interactive sessions
 Samples, 11-13
 Interface control, 4-24, F-6
 Interface type, 4-23, F-6
 Internal conversion routines, 8-80
 Internal subroutines, 8-38-8-40
 Inverted Print Field (of Control Record Specification), 3-11-3-13, 3-44, F-1
 ISAM
 Simulation with IMAGE, H-1-H-9
 User-written procedures, I-1-I-3

J

JIS (Japanese Industrial Standard) Collating Sequence, 3-15, 3-44, A-2, A-4
 Jobs,
 Sample, 10-13-10-19, 11-13-11-14
 Job Control Word, 3-4-3-9

INDEX (Continued)

K

Key field starting location, F-6
Key file name, 4-5, 4-44-4-45
KEYFL continuation option, 4-5, 4-44-4-45, 10-21
Key punching instructions, 2-1
Keyword parameters, 11-4
KSAM files, 1-2, 3-13, 4-3, 4-4, 4-6, 4-12-4-17, 4-19-4-20, 4-27, 4-29, 8-67-8-68, 9-5-9-6, 10-20-10-36, F-5, F-9, F-10
KSAM files
 creating, 4-5, 10-20-10-22

L

Labels
 Disc, 4-24, 4-47, F-6
Last record indicator, 7-11, 7-51, 7-53, 8-4, 8-6, 8-7, 8-19, 9-14, 9-15
LDA (Local Data Area), 7-6-7-7, 7-35, F-14
Length of data buffer, V/3000 interface, 13-5
Line Counter Specification
 Coding rules, 6-1-6-7, F-12-F-14
 Extension code, 4-20-4-22
 Line number option
 Purpose, 1-4
 Sample, 10-4, 10-5
Line Number Field (of Line Counter Specification), 6-3, 6-5-6-7, F-12, F-13
Line Number Option (of Line Counter Specifications), 3-11, 6-1, 6-5
Listfile, 11-22, 11-27
Listing and compilation options, 12-5-12-8
Listing output,
 Compiler, 11-14-11-15, 11-21
 Cross-Reference, 1-4, 1-10, 3-37, 11-18-11-20
 Source Program, 1-4, 1-7, 1-9, 11-27
 Symbol Table, 11-17-11-18
Literals
 Alphanumeric, 8-8, 8-11
 Numeric, 8-8, 8-11
Local Data Area, 7-6-7-7, 7-35, F-14
LOCK operator, 8-74-8-77
LOCK Option, 4-33
Locking files,
 IMAGE, 4-38-4-42, 8-74-8-77
KSAM and MPE, 4-31-4-33, 8-74-8-77
Logging, Program Name, 3-13, 3-44
Logical device numbers, 4-22-4-23
Logical Record Length Field (of File Description Specification), 4-11, F-5
Logical records, 4-11
LOKUP Operation, 8-13, 8-50-8-54
Long Name Option Target, 4-29, 4-35, 4-42, 4-43
Look-ahead fields, 7-13, 7-14
Low Subfield (of Calculation Specification), 8-20

LR Indicator, 3-40, 7-11, 7-51, 7-53, 8-4, 8-6, 8-7, 8-19, 9-14, 9-15
L0 Indicator, 3-23, 3-44
L1—L9 indicators, 7-8, 7-11, 7-12, 7-38-7-44, 7-51, 7-53, 8-2, 8-4, 8-6, 8-7, 8-19, 9-14, 9-15

M

Masterfile, 3-28, 11-24-11-25, 12-13-12-17
Matching/Chaining Field (of Input Specification), 7-39-7-44, 7-57, F-17
Matching fields, 7-39-7-44
Matching files, 4-8, 7-39-7-44
Matching-record indicator, 7-11, 7-39, 7-44, 7-51, 8-6, 8-7, 8-19, 9-14, 9-15
Matching records, 7-39-7-44
Message
 Clearwindow, V/3000 interface, 13-7
 display at terminal, V/3000 interface, 13-6
 display error, V/3000 interface, 13-14
 message identification, 8-63, 8-64, 8-65
 move to window, V/3000 interface, 13-6
Messages, error, 1-7, 1-11, 12-5-12-7, A-4, B-79
MHHZO operation, 8-13, 8-29
MHLZO operation, 8-13, 8-30
MLHZO operation, 8-13, 8-31
MLLZO operation, 8-13, 8-30
MOVEA operation, 8-13, 8-28
MOVE operation, 8-13, 8-26, 8-27
MOVEL operation, 8-13, 8-27-8-28
Move operations, 8-24-8-31
Move zone operations, 8-29-8-31
MPE/3000, 1-1, 11-1-11-29
MPE/3000 commands
 Format description, 11-3-11-5
 Continuation characters, 11-5
 Purpose of, 11-1
 Summary of, 11-6
 Structure of, 11-3-11-5
 Errors, 11-5
MSG operation, 8-13, 8-65-8-66, F-19, L-1
MULT operation, 8-13, 8-23
Multifile processing, 4-7, 7-41-7-44
Multiprogramming Executive Operating System,
 (see MPE/3000)
MVR operation, 8-14, 8-24

N

N Subfield (of Calculation Specification), 8-6
NAME = __\$CONTROL option, 2-5, 12-8
Names
 Array, 5-9-5-10, 5-20-5-21, 7-31-7-32
 Field, 7-31-7-34, 7-57, F-16

INDEX (Continued)

File, 4-3, 6-2-6-3, 7-1-7-3, 9-3, F-4, F-10, F-13, F-14, F-21
Subroutine, 8-38, 8-40
Table, 5-9, 5-10, 5-20, 5-21, 7-32
Newfile, 11-25, 12-13-12-17
NEXT, definition, V/3000 interface, 13-7
Next record, V/3000 interface, 13-7
NOLOCK option, 4-33
Not Subfield (of Input Specification), 7-18, F-15
Not Subfield (of Output Specification), 9-18, 9-44
Number of field edit errors, V/3000 interface, 13-5
Number of Records Field (of Input Specifications), 7-4-7-6, F-14
Numeric/alphanumeric field conversion, 7-28-7-29
Numeric literal, 8-8, 8-11
NUMBER, definition, V/3000 interface, 13-6

O

Object program execution, 1-11
Object Program File, 1-7, 1-11
Object Program Logic Flow Chart, E-1-E-5
Object Program Operating Cycle, 1-11, D-1-D-3, E-1-E-5
Object Program Output, 1-12
Operating Cycle, Object Program, 1-11, D-1-D-3, E-1-E-5
Operation Field (of Calculation Specification), 8-12-8-14, F-18
Operations
Arithmetic, 8-22-8-23, F-18
Branch/Exit, 8-34-8-38, F-19
Compare/Test, 8-31-8-34, F-19
Debug, 8-78-8-81, F-19
File, 8-56-8-57, F-18
Indicator/Bit Setting, 8-48-8-50, F-18
Move, 8-24-8-31, F-19
Record Number Conversion, 8-80, F-18
System, 8-84-8-88
Table/Array Look-Up, 8-50-8-54, F-19
Option Field (of Input Specifications), 7-6, 7-57, F-14
Option Target Field (of File Description Specification Continuation Record), 4-30-4-32
Option Type Field (of File Description Specification Continuation Records), 4-30-4-32
OR Lines, 7-22, 7-51, 8-2, 8-4, 8-5, 9-15, 9-17, 9-18-9-19
Output files, 4-5, 9-1
Output Indicators Field (of Output Specification), 9-13-9-14, 9-44, F-21
Output record formats, V/3000 interface, 13-11-13-13
Output Specifications,
Coding rules, 9-1
Purpose, 1-4, 9-1
Sample, 10-9, 10-11, F-20-F-24
V/3000 interface, 13-23
Overflow Indicator Field (of File Description Specifications), 4-18, 4-44, F-6
Overflow indicators, 4-18, 4-44, 6-1, 7-12, 7-51, 7-54, 8-6, 8-7, 8-19, 9-14, 9-15
Overflow line, 6-1, 6-3, 6-5-6-7, 9-8-9-9

P

Packed/Binary Field (of Output Specification), 9-34, 9-44, F-23
Packed decimal format, 7-27
PAGE Field, 7-32-7-33, 9-20-9-22
Page Overflow Test, 3-33
Page title, 12-11-12-12
PAGE1-PAGE7 Fields, 7-32-7-33, 9-20
Parameters
Keyword, 11-4-11-5
Positional, 11-4-11-5
PARM operation, 8-14, 8-42
Partial field translation, 1-2, 4-31, A-4, F-8
PARTTR Option, 4-31, 4-33
*PLACE METHOD, 3-34
Portion Subfield (of Input Specification), 7-18, F-15
Position Subfield (of Input Specification), 7-16-7-18, F-15
Pre-execution table/array files, 4-7, 4-8
Pre-Execution Time Tables and Arrays, 5-7, 5-22-5-23, 5-30
PREV, definition, V/3000 interface, 13-7
Previous record, V/3000 interface, 13-7
Primary files, 4-6, 4-7, 7-41-7-46
Print,
definition, V/3000 interface, 13-6
form, V/3000 interface, 13-6
Printer Spacing Chart, 1-6, 9-45-9-47, 10-9-10-10
Processing
Multifile, 4-7, 7-41-7-44
Random, 4-12
Sequential, 4-12
Single File, 7-44
Processing Mode Field (of File Description Specification), 4-12-4-13, F-5
Program deck structure, 1-4, 1-5, 11-1, 11-2
Program listing, 11-15-11-16, 12-6
Program Name Field (common to all specifications), 2-5-2-6, 4-28, F-3, F-10, F-14, F-24
Program Name Logging Field (of Control Record Specification), 3-13, 3-44
Programs
Compiling, 11-8-11-13
Purpose of RPG, 1-1
Put data in buffer, V/3000 interface, 13-7
PUTDTA, definition, V/3000 interface, 13-7
PUTFLD, definition, V/3000 interface, 13-7
PUTJW, 8-87
PUTMSG, definition, V/3000 interface, 13-6

Q

Quotation marks (as delimiters), 9-35, 9-36

INDEX (Continued)

R

RAF (Record Address Files), 4-7, 4-12, 5-1, 5-5-5-10, 10-31-10-39, F-5
Random processing, 4-12
RDBTNU, definition V/3000 interface, 13-7
RDEXIT options, 4-32
RDTERM, definition, V/3000 interface, 13-6
Read,
 batch record, V/3000 interface, 13-7
 data buffer, V/3000 interface, 13-5, 13-7
 data from terminal, V/3000 interface, 13-6
 form from forms file, V/3000 interface, 13-6
 next record, V/3000 interface, 13-7
 previous record, V/3000 interface, 13-7
READ operation, 8-14, 8-66-8-67
Record
 input format, V/3000 interface, 13-9-13-10
 number, batch record, V/3000 interface, 13-5
 types, V/3000 interface, 13-9
 read from batch file, V/3000 interface, 13-7
 read previous, V/3000 interface, 13-7
 write to batch file, V/3000 interface, 13-7
Record Addition/Deletion, 9-5-9-6
Record Address Field Length Field (of File Description Specification), 4-13, 4-14
Record address files, 4-7, 4-12, 5-1, 5-5-5-10, 10-31-10-39, F-5
Record Address Type Field (of File Description Specification), 4-13, 4-14, 4-47
Record Description Fields (of Input Specification), 7-1-7-57
Record Description Fields (of Output Specification), 9-1-9-2
Record Format Field (of File Description Specification), 4-9,, F-4
Record Identification Codes Field (of Input Specification), 7-16, 7-57
Record Indicator/Look-Ahead/Trailer Field (of Input Specifications), 7-8, 7-57, F-15
Record length check, 3-33
Record number adjust (of Control Record Specification), 3-13, 3-44
Record number conversion operations, 8-80
Record Sequence
 in arrays, 5-15-5-16
 in chaining files, 5-3
 in groups, 7-3-7-5
 in tables, 5-15-5-16
Record types, 7-3-7-4, 7-16-7-18
Relative Record Numbers, 3-13, 3-43
Related Tables/Arrays, 5-10-5-12, 8-52
Relative end position, 9-29-9-31, F-23
Release file, F-21
REREAD, definition, V/3000 interface, 13-7
Result Field (of Calculation Specification), 8-15, 8-27, 8-89
Resulting Indicators Field (of Calculation Specification), 8-17, 8-19, 8-20, 8-89, F-20
RESUME, definition, V/3000 interface, 13-7

Rewind operations, A-2
RLABL operation, 8-14, 8-42
RPGCOPY, 14-1-14-4
RPGCV operation, 8-14, 8-80
RPG/3000
 Advantages of, 1-1, 1-2
 Applications of, 1-1
 Capabilities of, 1-1, 1-2
 Compatibility with other systems, 1-11, A-1-A-5
 Compiler, 1-1
 Form name, 9-31, 13-1-13-26
 Library, 1-1
 Object Program Cycle, 1-11, D-1-D-3
 Programming Language, 1-1, 1-2
 Sample Program, 10-1-10-20
 Screen interface, 8-66, F-6, K-1, L-1
 Source Programs, 1-4, 1-7
 System Requirements, 1-11
 Using, 1-2
 VPLUS interface, 4-23, 4-24, 7-10, 9-15, F-6
RPGINIT utility, 7-6
RSPACE= _\$CONTROL option, 9-30, 12-9

S

Sample batch job output, 10-13-10-20
Sample program, 10-1-10-20
Secondary files, 4-6, 7-41-7-44
Sequence
 Chaining File, 5-3
 Group, 7-3-7-4, F-14
 Input, 4-8-4-9, 4-47, F-4
 Match File, 7-39
 Table and Array, 5-15-5-16, 5-19, 5-20, 5-21
 Textfile, 3-38, 3-44, F-3
Sequence-checking through match fields, 7-44
Sequence Number Field (common to all specifications), 2-3, F-1, F-4, F-10, F-12, F-14, F-17, F-20
Sequential look-up, 3-18
Sequential processing, 4-11
SETLL operation, 8-14, 8-50, F-19, H-9
SET operation, 8-14, 8-57, 8-60-8-62, F-19
SETOF operation, 8-14, 8-48
SETON operation, 8-14, 8-48
SETTING indicators on/off, 8-22
SHODTA, definition, V/3000 interface, 13-6
SHOMSG, definition, V/3000 interface, 13-6
SHOW, definition, V/3000 interface, 13-6
Sign-Process Field (of Control Record Specification), 3-21, 3-44, F-2
Simple edit codes, 9-25, 9-26, 9-42
Single-file processing, 7-44
Skip Field (of Output Specification), 9-12-9-13, 9-44, F-21
Skip requests, 3-35, 3-44, 9-12-9-13, 9-44
Skip-Suppress Field (of Control Record Specification), 3-32, 3-44, F-2

Sharing files, 4-35-4-36
 Softkeys
 labeling, 8-57, 8-60-8-61
 SORTA operation, 5-31, 8-14, 8-52, 8-54-8-55, F-19
 Source Program Listing, 1-8, 1-9
 Source Text
 Editing, 12-13
 Merging, 12-13
 Sequence-checking, 12-15
 Space Field (of Output Specification), 9-10, 9-44
 SPECIAL files, 4-22, 4-23, 4-28, 4-47
 Split chaining fields, 7-46, 7-49
 Spread records, 7-13-7-16
 SQR T operation, 8-14, 8-24
 Stacker-select, 7-22-7-24, 9-6, 9-18, 10-45, F-21
 START continuation option, 4-46, F-8
 STATUS continuation option, 4-46, F-8
 Status portion of edit word, 9-40
 Sterling notation, A-3
 SUB operation, 8-14, 8-23
 Subfields, 7-35-7-37
 Subroutines, External, 8-41-8-48, A-3
 Subroutines, Internal, 8-38-8-40
 Subsystem commands, compiler,
 Coding rules for, 12-5-12-17
 Comments in, 12-3
 Continuation records for, 12-3-12-4
 Effects of, 12-4
 Parameters for, 12-2
 Purpose of, 1-7, 12-1
 Summary of, 12-5
 Syntax and Format, 12-1
 Symbol table listing, 11-17-11-18
 System operations, 8-84-8-88

T

Table/Array Look-Up Field (of Control Record Specification), 3-19, 3-44, F-1
 Table/Array Name Field (of File Extension Specification), 5-16-5-17, 5-20, 5-21, F-11
 Table/Array, or Routine Name Fields (of File Extension Specification), 5-9-5-10, 5-20, 5-21
 Table/Array Sequence Fields (of File Extension Specifications), 5-51-5-16, 5-19, 5-20, 5-21, F-11, F-12
 Table/Array Table, C-3, C-20
 TAG operation, 8-14, 8-36-8-37
 Tables
 Alternating, 5-10-5-12, 5-17
 Chaining, C-3, C-19
 Compile-time, 5-7, 5-22-5-24, A-5
 Creating, 5-22-5-23
 Defining, 5-23-5-24
 File, C-3, C-14-C-18
 File Translation, C-2

General discussion, 5-1-5-2, 5-5, 5-7-5-34
 IMAGE/3000, C-18
 Loading, 5-24-5-29
 Look-up (LOKUP) operations, 8-13, 8-50-8-54
 Names, 5-9, 5-10, 5-20, 5-21, 7-32
 Pre-execution time, 4-7, 4-8, 5-7, 5-22-5-23
 Related, 5-10-5-12, 8-52
 Searching, 5-31
 Table/Array, C-3, C-20
 Using, 5-29-5-34
 Writing to output files, 5-34
 Tape labels, 4-24, 4-47
 Telecommunications, A-3
 Terminal, read data from, V/3000 interface, 13-6
 Test operations, 8-31-8-34
 TESTB operation, 8-14, 8-34
 Testing input field contents, 7-54
 Testing results of operations, 8-19-8-21
 TESTN operation, 8-14, 8-32
 TESTZ operation, 8-14, 8-32
 Textfile, 3-38, 11-8-11-13, 11-25, 12-13-12-17
 Textfile Sequence Check Field (of Control Record Specification), 3-38, 3-44, F-3
 TIME operation, 8-14, 8-84
 TIME2 operation, 8-14, 8-85
 To Field Position Field (of Input Specification), 7-30-7-31, 7-33
 To Filename Field (of File Extension Specification), 5-8-5-9, 5-20, 5-21, F-10
 Total calculations, 8-1, 8-4, E-2, E-4
 Total record/line, 6-1, 9-5-9-7, 9-45-9-47
 Total-Time Operations, 8-1, 8-4, D-2, D-3, E-2, E-4
 TRACE continuation option, 4-46, F-8
 Trace file, V/3000 interface, 4-46, F-8
 Trailer Portion of Spread Record, 7-13-7-17
 Translation Table and Alternate Collating Sequence Coding Sheet, 3-14-3-18, 3-27-3-29
 TRMID continuation option, 4-46, F-8
 Type Field (of Output Specification), 9-4-9-5, F-21

U

UPDATE Field, 3-9, 9-20, 9-24
 UDATE Source, 3-9
 UDAY Field, 3-9, 9-20, 9-24
 UDS (User Data Structure), 7-6-7-7, 7-35, F-14
 ULABL operation, A-3
 UMONTH Field, 3-9, 9-20, 9-24
 United Kingdom print format, 3-12, 3-13
 UNLCK operator, 8-74-8-77
 Unpacked decimal format, 7-26
 Update files, 4-5, 7-1
 User Data Structure, 7-6-7-7, 7-35, F-14
 User file characteristics, 11-28-11-29
 User indicators, 4-28, 7-6, 7-13, 7-51, 7-52, 8-19, 9-14, 9-17

INDEX (Continued)

User Message Catalog, 8-63-8-66, L-1, L-2
User-written ISAM procedures, I-1-I-3
Using RPG/3000, General Discussion of, 1-2
Uslfile (USL File), 1-7, 2-6, 11-8-11-13, 12-7
USWITCH command, 3-4-3-9, 4-28
USWITCH Records, 3-4
UYEAR Field, 9-20

V

V/3000,
 example using, 13-16
 interface, 4-22
 overview, 13-1
 run time errors, 13-14

W

Window,
 clear, V/3000 interface, 13-7
 move message to, V/3000 interface, 13-6
Word,
 Edit, 9-36-9-41
 Job Control, 4-28
WORKSTNC file, 4-9, 4-22
WORKSTN file, 4-9, 4-22, 4-23, 4-24, 4-28, 7-10, 13-1-13-4,
 13-7, 13-9, 13-14, 13-15, 13-17, 13-18, 13-21, F-6
WORKSTN interface group, 4-45-4-46
WORKSTNR file, 4-9, 7-11, 9-15
Write data to,
 batch file, V/3000 interface, 13-7
 buffer, V/3000 interface, 13-7
Write field to buffer, V/3000 interface, 13-7
WRTBAT, definition, V/3000 interface, 13-7

X

XFOOT operation, 8-14, 8-24

Z

Z-ADD operation, 8-14, 8-23
Zone portion of character, 7-18
Z-SUB operation, 8-14, 8-23

\$CONTROL subsystem command, 12-5-12-8
\$COPY subsystem command, 12-5, 12-9,14-1
\$EDIT subsystem command, 12-5, 12-14, 12-15-12-17
\$IF subsystem command, 12-5, 12-9-12-10
\$INCLUDE subsystem command, 12-5, 12-8, 14-1, 14-2
\$NEWPASS, 11-8, 11-9, 11-10, 11-11, 11-12, 11-13, 11-26
\$NULL, 11-22, 11-24, 11-25
\$OLDPASS, 11-8, 11-9, 11-10, 11-12, 11-14, 11-26, 12-7
\$PAGE subsystem command, 12-5, 12-12
\$SET subsystem command, 12-5, 12-10-12-11
\$STDIN, 4-22
\$STDLST, 4-22
\$TITLE subsystem command, 12-5, 12-10, 12-11-12-12
*BLANK, 8-11, 8-27, 8-28
*BLANKS, 8-11, 8-27, 8-28
*ERROR Field, 9-19, 9-25
*PLACE Field, 9-19, 9-22-9-23
*PRINT Field, 9-19, 9-24
*ZERO, 8-11, 8-27, 8-28
*ZEROS, 8-11, 8-27, 8-28
:RPG command, 11-8-11-9, 11-13-11-14
:RPGGO command, 11-10
:RPGPREP, 11-11-11-12

READER COMMENT SHEET

**HP 3000 COMPUTER SYSTEMS
RPG
Reference Manual**

32033-90058

July 1986

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate? Yes No (If no, explain under Comments, below.)

Are the concepts and wording easy to understand? Yes No (If no, explain under Comments, below.)

Is the format of this manual convenient in size, arrangement, and readability? Yes No (If no, explain or suggest improvements under Comments, below.)

Comments:

FROM:

DATE: _____

Name _____

Company _____

Address _____

FOLD

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1070 CUPERTINO,CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

Publications Manager
Hewlett-Packard Company
Computer Language Lab
19447 Pruneridge Avenue
Cupertino, California 95014



FOLD

FOLD