HP RPG/iX Reference Manual

900 Series HP 3000 Computers

HP Part No. 30318-90003
Printed in U.S.A. 1993

E1193

Printing History

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

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

First Edition	December 1988	30318A.00.00
Second Edition	October 1989	30318A.00.04
Third Edition	November 1993	30318A.00.10

Preface

The *HP RPG/iX Reference Manual* explains all of the language features of RPG. It is written in a reference style, assuming the reader is familiar with RPG.

This manual is directed to experienced RPG programmers, who may or may not be familiar with Hewlett-Packard computers. The manual discusses the language features available with the MPE iX operating system.

This manual is organized as follows:

Chapter 1	Explains the general steps involved in writing and testing an RPG program.
Chapter 2	Describes the fields that are used in all RPG Specifications.
Chapter 3	Describes the fields in the Header Specification.
Chapter 4	Describes the fields in the File Description Specification.
Chapter 5	Describes the fields in the File Extension Specification.
Chapter 6	Describes the fields in the Line Counter Specification.
Chapter 7	Describes the fields in the Input Specification.
Chapter 8	Describes the fields in the Calculation Specification.
Chapter 9	Describes the fields in the Output Specification.
Chapter 10	Explains how to use VPLUS with RPG. VPLUS is a forms management product that lets you create screens and process them in an RPG program.
Chapter 11	Explains how to use the RPG Screen Interface (RSI) in an RPG program. RSI is a forms management facility unique to RPG. You can use it to create and process screen forms in an RPG program.
Chapter 12	Discusses the RPG compiler and compiler subsystem commands.
Appendix A	Lists the RPG compiler messages.
Appendix B	Lists the run-time error messages.
Appendix C	Lists the ASCII and EBCDIC character sets.

Related Documentation

Refer to the following documents for further information on features available in the RPG programming language:

HP RPG/iX Programmer's Guide (30318-90001) - This manual describes how to use the RPG language elements to contruct an RPG program. This manual contains many examples and programs.

HP RPG/iX Utilities Reference Manual (30318-90006) - This manual explains how to use these RPG utilities: XSORT, RISE, SIGEDITOR, and RPGINIT.

HP RPG Pocket Guide (30318-90002) - This guide is a template showing the meaning and placement of each column and line for all RPG/iX specifications.

Data Entry and Forms Management System VPLUS/3000 (32209-90001) - This manual includes a complete discussion about the screen management software product, VPLUS. You can use this product within RPG programs when using a terminal.

EDIT/3000 Reference Manual (03000-90012) - This manual explains how to use the text processor software product, EDITOR.

KSAM/3000 Reference Manual (30000-90079) - This manual explains how to use KSAM disc files and how to access them.

TurboIMAGE/iX Database Management System (30391-90001) - This manual discusses the TurboIMAGE database software product.

MPE iX Intrinsics Reference Manual (32650-90028) - This manual discusses the operating system routines that can be used by external subroutines in an RPG program.

Native Language Programmer's Guide (32650-90022) - This manual discusses how to create and use Native Language Support message files.

Message Catalogs Programmer's Guide (32650-90021) - This manual discusses how to create and use non-Native Language Support message files.

FCOPY Reference Manual (03000-90064) - This manual explains how to use the FCOPY file utility.

SORT-MERGE/iX Programmer's Guide (32650-90080) - This manual explains how to use the SORT/MERGE file utility.

Accessing Files Programmer's Guide (32650-90017) - This manual discusses the ways MPE/iX files can be processed.

MPE/iX General User's Reference Manual (32650-90002) - This manual discusses file, group, and account structures.

MPE/iX Commands Reference Manual (32650-90003) - This manual describes the MPE/iX commands, such as FILE.

Example Conventions

Throughout this manual, examples of RPG program code are shown using figures similar to the one below. The first two lines are a ruler to help you quickly see the column positions for the code. The shaded numbers on the left are not sequence numbers. Rather, they are used as reference numbers for comments in the text. Lines are referenced only to highlight specific concepts. Additionally, some examples show lines containing dots only. Dots indicate that, to clarify examples, code has been omitted.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

(1)   C           MESSAGE   DSPLY
      C                       .
      C                       .
(2)   C                       DSPLY          FIELD1
      C                       .
      C                       .
(3)   C           FIELD2    DSPLY          FIELD3
```

Figure 8-42. Using the DSPLY Operation

Syntax Conventions

NOTATION	DESCRIPTION
UPPERCASE	Within syntax statements, characters in uppercase must be entered in exactly the order shown, though you can enter them in either uppercase or lowercase. For example: `SHOWJOB` Valid entries are: `showjob` `ShowJob` `SHOWJOB` Invalid entries are: `shojwob` `Shojob` `SHOW_JOB`
italics	Within syntax statements, a word in italics represents a formal parameter or argument that you must replace with an actual value. In the following example, you must replace *filename* with the name of the file you want to release: `RELEASE` *filename*
punctuation	Within syntax statements, punctuation characters (other than brackets, braces, vertical parallel lines, and ellipses) must be entered exactly as shown.

{ }	Within syntax statements, when several elements within braces are stacked, you must select one. In the following equivalent examples, you select `ON` or `OFF`:
	`{ON }`
	`SETMSG {OFF}`

[ ]	Within syntax statements, brackets enclose optional elements. In the following example, brackets around `,TEMP` indicate that the parameter and its delimiter are not required:
	`PURGE` *filename*`[,TEMP]`
	When several elements within brackets are stacked, you can select any one of the elements or none. In the following equivalent examples, can select *devicename* or *deviceclass* or neither:
	`[`*devicename* `]`
	`SHOWDEV [`*deviceclass*`]`

Example Conventions

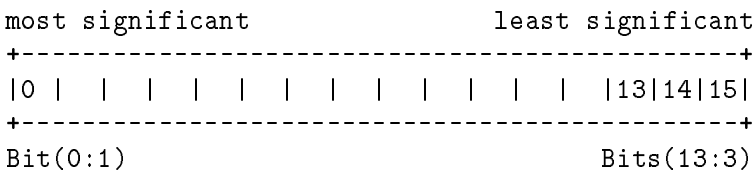
NOTATION	DESCRIPTION

`[ ... ]`	Within syntax statements, a horizontal ellipsis enclosed in brackets indicates that you can repeatedly select elements that appear within the immediately preceding pair of brackets or braces. In the following example, you can select *itemname* and its delimiter zero or more times, each instance of *itemname* preceded by a comma:
	`[,`*itemname*`][...]`
	If a punctuation character precedes the ellipsis, you must use that character as a delimiter to separate repeated elements. However, if you select only one element, the delimiter is not required. In the following example, the comma does not precede the first instance of *itemname*:
	`[`*itemname*`][,...]`

`| ... |`	Within syntax statements, a horizontal ellipsis enclosed in parallel vertical lines indicates that you can select more than one element that appears within the immediately preceding pair of brackets or braces. However, each element can be selected only one time. In the following equivalent examples, you must select `,A` or `,B` or `,A,B`or `,B,A` :
	`{,A}`
	`{,B}|...|`
	If a punctuation character precedes the ellipsis, you must use that character as a delimiter to separate repeated elements. However, if you select only one element, the delimiter is not required. In the following example, you must select `A` or `B` or `A,B` or `B,A` (the first element is not preceded by a comma):
	`{A}`
	`{B}|,...|`

... ⋮	Within examples, horizontal or vertical ellipses indicate where portions of the example are omitted.
␣	Within syntax statements, the space symbol ␣ shows a required blank. In the following example, you must separate *modifier* and *variable* with a blank: `SET [(`*modifier*`)]␣(`*variable*`);`
KEY	The symbol KEY indicates a key on the terminal's keyboard. For example, RETURN indicates the carriage return key.
CONTROL *char*	CONTROL *char* indicates a control character. For example, CONTROL Y means you must simultaneously press the control key and Y key on the terminal's keyboard.

NOTATION	DESCRIPTION
base prefixes	The prefixes %, #, and $ specify the numerical base of the value that follows: **%***num* specifies an octal number **#***num* specifies a decimal number **$***num* specifies a hexadecimal number When no base is specified, decimal is assumed.
Bit (*bit:length*)	When a parameter contains more than one piece of data within its bit field, the different data fields are described in the format Bit (*bit:length*), where *bit* is the first bit in the field and *length* is the number of consecutive bits in the field. For example, Bits (13:3) indicates bits 13, 14, and 15:



Contents

Figures

Tables

1

Introduction to RPG

The HP Report Program Generator (RPG) language is a machine-independent, task-oriented language that enables you to easily create programs that print reports, update files, and perform many other general file functions. RPG is available on non-HP computers and programs running on these computers require little or no conversion to run on HP 3000 Series 900 computers. For information on converting RPG programs that run on non-HP computers or that run under the MPE V operating system, see Appendix A of the *HP RPG Programmer's Guide*.

RPG is used frequently in business and commercial applications. You can use RPG to print mailing labels. It can also be used to compute complex payrolls including the printing of paychecks, payroll registers, and other payroll reports. RPG is also ideal for producing inventory lists, invoices, insurance benefit notices, customer transactions, and summaries of sales and losses. In addition to printing reports, you can use RPG to update the files from which the reports are produced.

In summary, you can use RPG programs to:

- Process large tables and arrays of data.
- Process data stored on several types of devices.
- Perform extensive calculations and save the results on disk or tape, or display them on a terminal.
- Update large disk files and databases.
- Generate several reports of varying complexity in a single program.
- Process records randomly or sequentially using the the Keyed Sequential Access Method (KSAM) or TurboIMAGE subsystems.

RPG programs differ from programs written in other languages. You do not decide the main logic of the program. Your source statements, *specifications*, are executed in a pre-determined order. This is called the RPG *logic cycle*. Take, for example, a payroll program that reads employee hours-worked, then computes pay for the period. The logic cycle reads the first employee's hours, and executes the specifications that compute earnings, deductions, and withholdings. It then prints the employee's paycheck and reads the next employee's hours. This sequence is repeated until the last employee's hours are processed. To effectively use RPG, you should understand what actions are performed during each phase of the logic cycle.

This manual devotes a separate chapter to each of the seven types of specifications. See the *HP RPG Programmer's Guide* for a complete discussion of the RPG logic cycle.

The Components of RPG

The RPG language subsystem consists of the following components:

- **The RPG symbolic programming language**.

  The rules of language syntax that you use when entering RPG programs.

- **The RPG compiler**.

  The RPG compiler translates source statements into a format that can be executed. In addition, it helps to detect errors at the source level with extensive diagnostic messages. Programs written originally to run on other computer systems can be compiled successfully with few, if any, changes.

- **The RPG library**.

  The RPG library contains run-time procedures for certain functions, such as input and output operations, output field editing, and specialized calculations. Run-time procedures are used by compiler-generated code. RPG's use of the library procedures is transparent to the user.

Using RPG

The next seven sections give an overview of how to use RPG. Figure 1-1 gives a pictorial representation of this process.

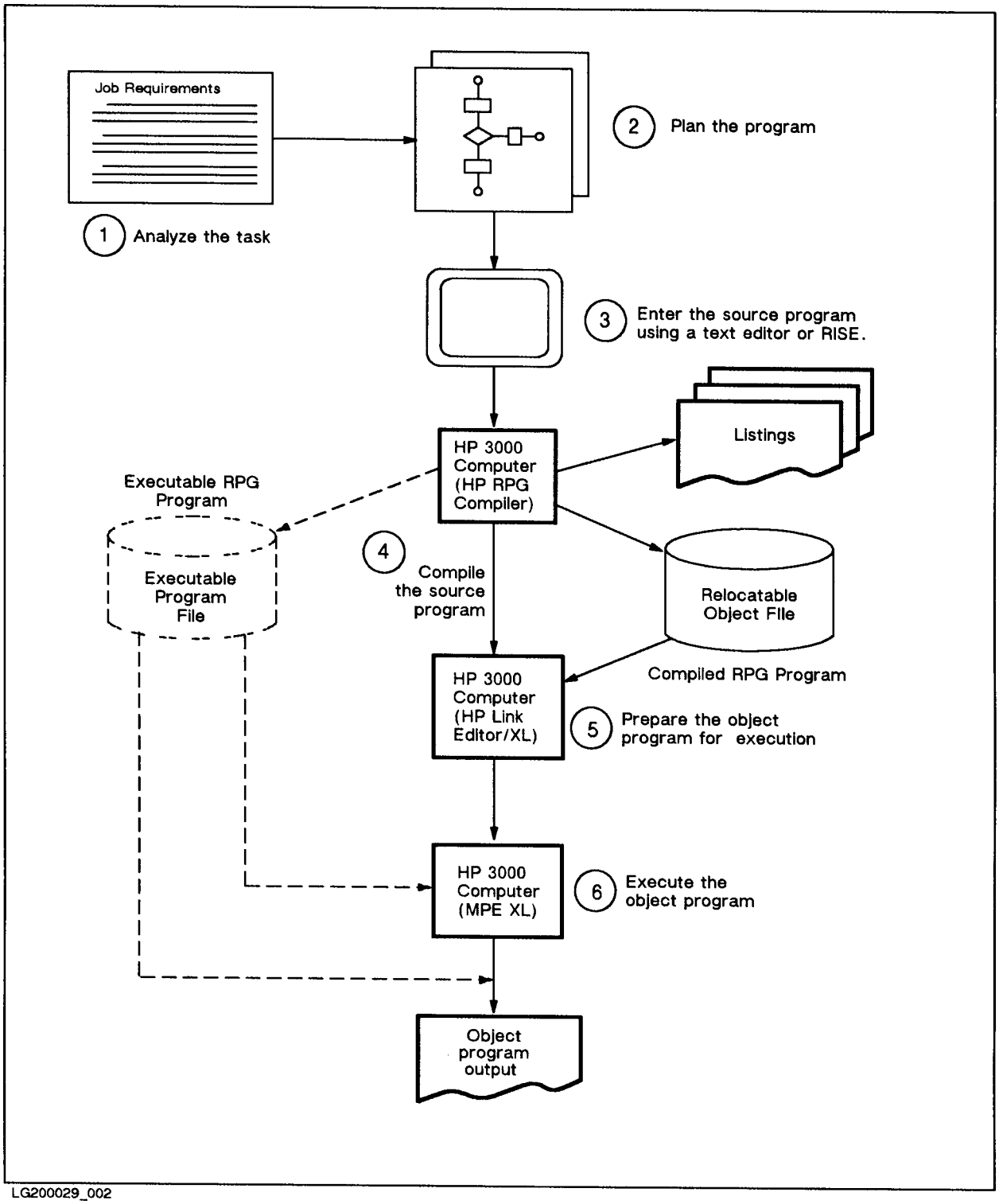


Figure 1-1. Steps in Preparing and Running an RPG Program

Step 1: Analyze the Task

Analyze the data to be processed by the program. Determine the calculations and other operations that must be performed on the data. Decide what data should be output and in what format. Also decide what devices to use for input and output files.

For example, Figure 1-2 shows the layout of a new report that needs to be produced. The report is detailed on a printer spacing chart. The chart shows all of the fields on the report along with headings.

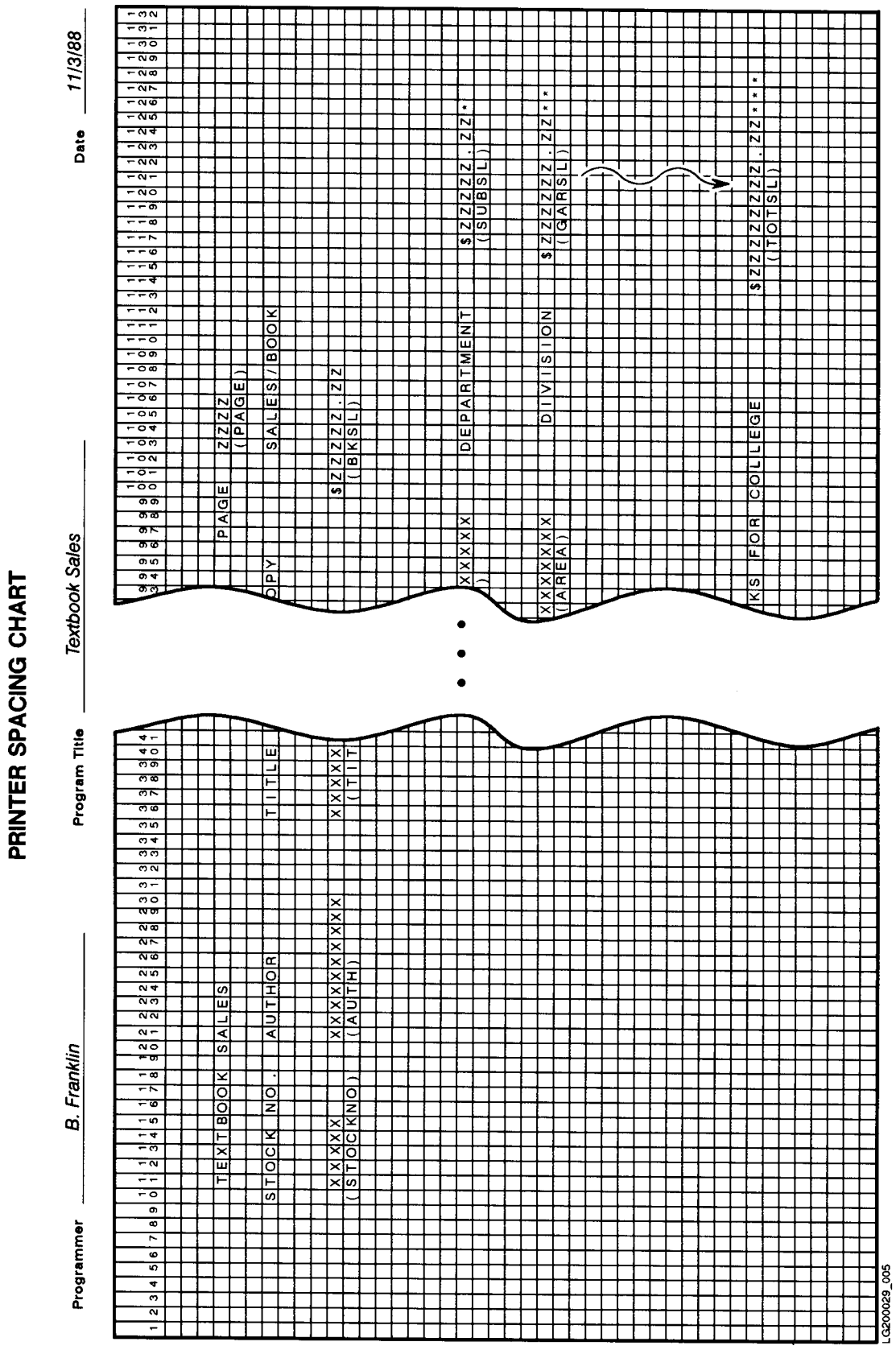


Figure 1-2. A Report Defined on the Printer Spacing Chart

Step 2: Plan the Program

Plan the general program steps required to produce the desired output. In doing this, you may want to prepare a flowchart of the program.

Step 3: Create the Source Program

Create RPG specifications for the program. You can enter them using any line editor or word processor that produces a standard ASCII file. For example, you may want to use EDITOR or TDP. Alternatively, you can use the RPG Interactive System Environment (RISE) to enter the specifications. The *HP RPG Programmer's Guide* gives more details on entering a program. Also, RISE is discussed in detail in the *RPG Utilities Reference Manual*.

The eight types of RPG specifications are listed below. Each specification is identified by a unique letter code in the sixth position (column). The specifications are listed below in the order that you enter them. For general rules about entering specifications, see Chapter 2. For details about entering a particular specification, turn to the chapter in this manual where it is discussed.

- **Header Specification (H)**

  Contains run-time options such as how to handle certain editing codes and run-time errors, and whether to print a compiler Cross-Reference listing. This specification is optional. Use just one Header Specification in each program.

- **File Description Specification (F)**

  Defines a file used in the program, assigns a name to it, describes its type and record size and how it is accessed. This specification is optional.

- **File Extension Specification (E)**

  Describes a table or array used in the program. This specification is also used to supply additional File Description file information. This specification is optional.

- **Line Counter Specification (L)**

  Provides information about a line printer file, such as the form length and printer carriage control information. This specification is optional.

- **Input Specification (I)**

  Describes the types of input records in a file and the fields they contain. This specification is optional.

- **Calculation Specification (C)**

  Defines an operation to perform on data once it is read into memory. Operations include moving data in memory, performing arithmetic and branching operations, and calls to subroutines. This specification lets you directly control input and output. It is optional.

- **Output Specification (O)**

  Defines the output records in a file, including the fields that it contains. This specification is optional.

- **Array/Table File Name Specification (A)**

  Names a file containing a table or array to be processed.

Figure 1-3 shows how specifications are used in an RPG program. This program was written using the printer layout chart shown in Figure 1-2

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

(1)  H                                              XLS

(2)  FINPUT   IP  F      80            DISK
     FTABFILE IT  F      80           EDISK
     FREPORT  O   F     132     OF     LLP

(3)  E    TABFILE         TABA    8 160  5  ATABB    5 0 ALTERNATING TAB

(4)  LREPORT   66FL 55OL

(5)  IINPUT   AA  15
     I                                       1   5 STOKNO
     I                                       6  15 GSAREAL2
     I                                      16  25 SUBJ  L1
     I                                      26  35 AUTH
     I                                      36  55 TITLE
     I                                      56  58 EDITN
     I                                      59  622PPCOPY
     I                                      64  680NUMSHP

(6)  C           PPCOPY    MULT NUMSHP    BKSL    72       SALES/BOOK
     C           BKSL      ADD  SUBSL     SUBSL   92       SALES/SUBJECT
     C           BKSL      ADD  GARSL     GARSL  102       SALES/GEN AREA
     C           BKSL      ADD  TOTSL     TOTSL  122       TOTAL SALES
     C           STOKNO    LOKUPTABA      TABB           10
     C   10      TABB      SUB  NUMSHP    TABB             UPDATE INV TAB
```

Figure 1-3. An RPG Source Program

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

7    OREPORT  H         1P
     O                                      24 "TEXTBOOK SALES"
     O                           UDATE Y    50
     O                                     100 "PAGE"
     O                           PAGE      106
     O         H   6     OF
     O                                     100 "PAGE"
     O                           PAGE      106
     O         H 33      1P
     O        OR         OF
     O                                      18 "STOCK NO."
     O                                      26 "AUTHOR"
     O                                      40 "TITLE"
     O                                      68 "EDN"
     O                                      77 "NO. SOLD"
     O                                      95 "PRICE/COPY"
     O                                     112 "SALES/BOOK"
     O         D 1       15
     O                                       .
     O                                       .
     O                                       .
```

Figure 1-3. An RPG Source Program (Continued)

Comments

1. This line is the Header Specification for the program.
2. This line begins the File Description Specifications.
3. This line is the File Extension Specification.
4. This line is the Line Counter Specification.
5. This line begins the Input Specifications.
6. This line begins the Calculation Specifications.
7. This line begins the Output Specifications.

Step 4: Compile the Source Program

Once you enter the specifications for a program and desk-check it, you can compile it. The following command compiles the program, SALES:

```
:RPGXL SALES.SOURCE, SALEO.SOURCE
```

The RPG compiler translates the source program into an object program assigning storage areas and creating routines to handle input and output. The object program is saved as the relocatable object file, SALEO.SOURCE. Optionally, the compiler prints a Cross-Reference listing. When there are source errors, appropriate error messages are generated.

Appendix A in this manual lists compile-time error messages. See the *HP RPG Programmer's Guide* for additional examples on how to compile an RPG program.

Note

You can compile and link in one step (bypassing Step 5) by using the `RPGXLLK` command. Or, you can compile, link, and execute in one step (bypassing Steps 5 and 6) by using the `RPGXLGO` command.

Step 5: Prepare the Object Program for Execution

When you compile a program using the RPGXL command (see the previous section), the program cannot be executed directly. You must prepare the compiled program for execution by linking it to external subroutines and procedures that it uses. The result is an executable program file. The following MPE/iX command links the program, SALEO, which was compiled in the previous section:

```
:LINK SALEO.SOURCE, SALEP.PROGRAM
```

See the *HP RPG Programmer's Guide* for additional examples on how to link an RPG program.

Step 6: Execute the Object Program

Once a compiled program is linked or prepared for execution, you can execute it. The following command executes the program, SALEP, which was prepared for execution in the previous section:

```
:RUN SALEP.PROGRAM
```

If errors occur during execution, appropriate messages are displayed (see Appendix A for a complete description of run-time error messages). See the *HP RPG Programmer's Guide* for hints on debugging RPG programs.

A valuable feature of the MPE operating system is the ability to assign files to specific hardware devices at run time. This enables you to run a program using different devices without recompiling the program.

Figure 1-4 shows the output sales report generated by the program in Figure 1-3. The report can be assigned to the printer or to a disk file at run time.

TEXTBOOK SALES 10/02/88 PAGE 0001

STOCK NO.	AUTHOR	TITLE	EDN 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 ADMINISTRA				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	
00052	MESSICK	BASIC ELECTRONICS	1	00020	$6.00	$120.00	
		SALES FROM ALL BOOKS FOR ELECT 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***

LG200029_006

Figure 1-4. Output From an RPG Program

2

General Specifications

This chapter discusses the fields that are used in all of the RPG Specifications. They are:

- Sequence Number Field (columns 1-5).

  This field is optional.

- Specification Type Field (column 6).

  This field is required.

- Program Name Field (columns 75-80).

  This field is optional.

If you're using RISE to enter source programs, field names appear on the terminal screen to help you enter information in the correct positions. If you're using a general-purpose editor, it is up to you to ensure that specification data is entered in the correct columns.

Note

Since the Sequence Number and Program Name Fields are optional, they are not shown in examples throughout this manual.

Sequence Number (Columns 1-5)

This field lets you assign a unique sequence number to each line in your source program. You can use sequence numbers to help you locate lines in the source program listing or lines that have been added or changed.

Enter values into this field as follows:

Columns 1-5	Description
00000 - 99999	The sequence number of the specification. You can omit leading zeros (leading, trailing, or embedded blanks are treated as zeros). Enter sequence numbers in ascending order. Sequence numbers are used for documention purposes only - RPG does not sequence-check programs.
blank	No sequence numbers are used.

A convention that you might use when entering sequence numbers is to reserve columns 1 and 2 for page numbers and columns 3 and 4 for line numbers within each page. Use column 5 for line insertions. For example, sequence number 01013 indicates page 01, line 01. Since the line insertion number is 3, the line was added after the original sequence numbers were assigned.

Specification Type (Column 6)

This field identifies the type of specification or statement this line contains. You must enter one of the following characters in this field:

Column 6	Description
H	Header Specification
F	File Description Specification
E	File Extension Specification
L	Line Counter Specification
I	Input Specification
C	Calculation Specification
O	Output Specification
A	Array/Table File Name Specification
$	Compiler Subsystem Command
*	Comment line
	All characters in columns 8-74 of comment lines are ignored by the compiler, but they are printed in the source program listing. Use comment lines liberally to document operations, fields, and methods used in the program. (You can also specify that a line is a comment line by entering an asterisk in column 7.)

Program Name (Columns 75-80)

This field gives identifying information about the program. You can enter a name to be printed on each page of the source program listing. You can also enter documentary information on individual specification lines that will be printed along with the specification in the source program listing.

Follow these rules when using this field:

Columns 75-80	Description
Header Specification:	
Valid program name. (Program names can contain up to six characters, beginning with a letter A-Z. The remaining characters can be upper case or lower case letters, the digits 0-9, or an apostrophe. Do not embed blanks in the name.)	The name of the source program. This name appears on each page of the source program listing. (This name is not used to name the source or relocatable object files on disk.) An alternative to entering a program name in this field is to use the NAME= option of the $CONTROL compiler subsystem command. This option names the source program but lets you reduce the length of each specification from 80 characters to 74, thus saving disk space.
blank	If you do not use the NAME= option of the $CONTROL compiler subsystem command, the name RPGOBJ is printed on each page of the source program listing.
All Other Specifications:	
Any characters	The program or subroutine name, or comments of any kind. For example, you can enter a name that identifies a block of code. The name is printed on the specification line of the source program listing.
blank	No identifying information is used for the specification.

3

Header Specifications

The Header Specification is the first specification in a program. (If you do not enter a Header Specification, the compiler warns you in the compiler listing and uses the default values shown in the section "The Header Specification Default Summary"). The only statements which can precede the Header Specification are comment lines and the $COPY and $CONTROL compiler subsystem commands.

The Header Specification lets you:

- Enable the RPG DEBUG feature.
- Specify where to get the initial values for USWITCH and UDATE.
- Use a collating sequence other than the ASCII Collating Sequence during certain compare operations.
- Translate input files from one code to another (for instance, EBCDIC to ASCII) before and after processing.
- Allow non-numeric data in numeric fields.
- Allow compatibility with other implementations of RPG.
- Print a Cross-Reference listing.

The Header Specification is identified by an H in column 6:

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

H
```

Figure 3-1. The Header Specification

The Header Specification Fields

The fields you can use in the Header Specification are described in the sections which follow in this chapter. Each field has a unique name and occupies specific positions (columns) in the specification.

Sequence Number (Columns 1-5)

The Sequence Number Field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

This field contains an H to identify this line as a Header Specification.

Error Dump File Name (Columns 7-14)

This field identifies the file to which Error Dumps are written.

Columns 7-14	Description
File name	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.) If the file does not exist at run time, the Error Dump is written to the standard list device ($STDLIST).
blank	Write the Error Dump to the standard list device ($STDLIST).

Debug (Column 15)

This field determines whether or not Calculation Specification DEBUG operations will be executed.

Column 15	Description
1	Execute all DEBUG operations.
blank	Suppress all DEBUG operations; the compiler treats DEBUG operations as comments.

The DEBUG Calculation Specification operation lets you monitor the status of various indicators while the program is executing. This is very helpful when debugging the program. You can use as many DEBUG operations as you wish, and this field acts as a switch to allow or disallow their execution.

USWITCH Source (Column 16)

This field lets you specify where to get the initial values for the user indicators (U1-U8). You only need to specify the source for those indicators actually used in the program. When the program ends, it updates the JCW to reflect the condition of the indicators that were actually used; the others are left unchanged.

Column 16	Description
F	Initialize user indicators from USWITCH records contained in the file whose formal designator is USWITCH.
blank (job stream)	If running a job stream, initialize user indicators from USWITCH records contained in the job file.
J	Initialize user indicators from the job or session Job Control Word (JCW).
blank (interactive)	If running an interactive session, initialize user indicators by prompting for their initial settings.

F (USWITCH File)

USWITCH records are read from the file whose formal designator is USWITCH. USWITCH records can have either a long or short format (see "Long Format USWITCH Records" and "The Short Format USWITCH Record" which follow).

When the program is executed from a job stream, initial settings of the user indicators are written to the job stream listing.

Run-time USWITCH errors are listed separately in appendix B.

Blank (Job Stream)

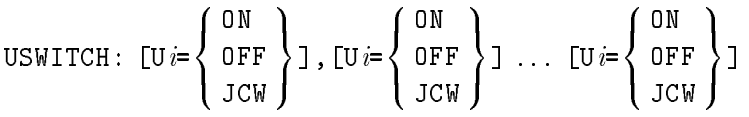
USWITCH records are read from the batch job stream immediately following the command that executes the program. USWITCH records can have either a long or short format (see "Long Format USWITCH Records" and "The Short Format USWITCH Record" which follow).

When the program begins, the initial settings of the user indicators are written to the job stream listing.

Run-time USWITCH errors are listed separately in appendix B.

Long Format USWITCH Records

Long Format records are used with USWITCH Source codes F and blank (job stream). You can use as many Long Format records as necessary. Enter them in the following format:



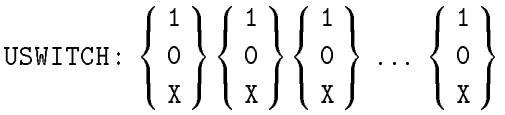
Start USWITCH: in column 1 and end it in column 8. The Ui parameters can appear in any order (for instance, U3=ON,U4=ON,U1=ON ...). If you omit a user indicator, that indicator is turned OFF. If you enter U1 only, or if you use an entry other than ON, OFF, or JCW, the corresponding user indicator is turned OFF. If the same indicator is entered twice, the second parameter overrides the first. For instance, U1=ON,U3=OFF,U1=OFF is equivalent to U1=OFF,U3=OFF.

You can enter Long Format records in free format; that is, blanks can appear between USWITCH: and the parameter list, or between individual parameters or subparameters. You can continue the parameter list onto one or more USWITCH records without using a continuation character, but do not split a parameter between records.

If Long Format USWITCH records reside in a job file, end them with a line containing two asterisks (columns 1-2).

The Short Format USWITCH Record

The Short Format USWITCH record is compatible with other implementations of RPG and it is easy to use. Enter settings for all eight user indicators in this format:



Enter USWITCH: in positions 1-8. Each parameter after USWITCH: stands for the settings of user indicators U1-U8. Do not include blanks between user indicator parameters. One (1) turns the corresponding user indicator ON, zero (0) turns the corresponding user indicator OFF, and X initializes the corresponding indicator to the value previously established for it in the JCW.

If you use the short format record in a USWITCH file, you may follow it with a line containing asterisks in positions 1-2 (this is optional). If you use the short format record in a job stream, do not follow the record with asterisks in positions 1-2.

Examples

The following example shows how to enter a job file containing a program that reads user indicators from a USWITCH file (the contents of the USWITCH file are listed after the job file):

```
!JOB USER.ACCT              (Begin job)
      .
      .
      .
!FILE USWITCH=SWITCH2       (USWITCH recs read from file SWITCH2)
!RUN PROG3                  (Execute PROG3: Header column 16 = F)
                            (No USWITCH recs read from job stream)
!EOJ                        (Terminate job)
```

The contents of the file, SWITCH2, are:

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

The next example is the same as the previous one except that it is run in a session, not in a job stream:

```
:HELLO USER.ACCT            (Begin session)
     .
     .
     .
:FILE USWITCH=SWITCH2       (USWITCH recs read from file SWITCH2)
:RUN PROG3                  (Execute PROG3: Header column 16 = F)
     .                      (No prompting for USWITCH values)
     .
     .
:BYE                        (End session)
```

The following example shows how to enter Long Format USWITCH records in a job file:

```
!JOB USER.ACCT              (Begin job)
      .
      .
      .
!RUN PROG1                  (Execute PROG1: Header column 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)
!EOJ                        (Terminate job)
```

The last example shows how to enter the Short Format USWITCH record in a job file:

```
!JOB USER.ACCT              (Begin job)
      .
      .
      .
!RUN PROG2                  (Execute PROG2: Header column 16 = blank)
USWITCH: 1X000001           (Set U1, U8 ON; U2 to previous setting)
                            (No ** record after Short Format record)
!EOJ                        (Terminate job)
```

J (JCW)

Entering a J in column 16 initializes the user indicators with the values in the system JCW. The JCW may have been set by another program running in the same job or session, or by the system command SETJCW. You can pass indicator settings from one RPG program to another through the JCW. RPG always updates the JCW with the final settings of any user indicators actually used in the program, regardless of the entry in column 16. Just specify a J in column 16 of the Header Specification in the second program, and the user indicators are set the same as when the first program ended.

The user indicators occupy bits 8-15 of the JCW. For example, if bit 8 is ON, user indicator 1 is ON. If bit 9 is ON, user indicator 2 is ON and so on.

If the RPG program is executed from a job stream, the initial settings of all user indicators are printed in the job listing after the command that executes the program.

Example

The following session dialogue shows how to turn ON user indicators U1, U2, and U8 in the JCW. This is accomplished by the SETJCW command. The JCW is set to the octal value, 301. Since the binary equivalent of octal 301 is 00000000 11000001, bits 8, 9, and 15 are turned ON. The session then executes PROG6 and PROG7 which are RPG programs that contain a J in column 16 of the Header Specification. User indicators are passed automatically from PROG6 to PROG7.

```
:HELLO USER.ACCT            (Begin session)
     .
     .
     .
:SETJCW JCW=%301            (JCW = %301 = Binary [11000001])
:RUN PROG6                  (Execute PROG6: Header col 16 = J)
                            (no prompting for USWITCH values)
:RUN PROG7                  (Execute PROG7: Header col 16 = J)
     .
     .
:BYE                        (End session)
```

Blank (Interactive)

If column 16 contains a blank and the program is run from a session, RPG prompts you as follows to enter the initial user indicator settings:

```
PLEASE INPUT USER SWITCH Ui=?
```

Enter ON, OFF, or JCW (typing RETURN is the same as OFF). JCW directs RPG to get the user indicator settings from the JCW (see the discussion of the value J for this field). Entering some other value turns the user indicator OFF. The above prompt is repeated for each user indicator in the program. When all of the indicators are entered, the program begins.

Example

The following session dialogue shows how to turn ON user indicators U1 and U2:

```
:HELLO USER.ACCT                    (Begin session)
     .
     .
     .
:RUN PROG5                          (Execute PROG5)
PLEASE INPUT USER SWITCH U1=?ON     (Turn ON user indicator U1)
PLEASE INPUT USER SWITCH U2=?ON     (Turn ON user indicator U2)
     .
     .
     .
:BYE                                (End session)
```

UDATE Source (Column 17)

This field lets you determine where to get the initial values for the predefined fields: UDATE, UDAY, UMONTH, and UYEAR.

Column 17	Description
F	Initialize UDATE, UDAY, UMONTH, and UYEAR using the date record in the file whose formal designator is RPGUDATE.
blank	Initialize UDATE, UDAY, UMONTH, and UYEAR using the system date.

F (Date Record)

Before running the program, enter an operating system `:FILE` command to equate the formal file designator RPGUDATE to the actual file that holds the date record. Equating RPGUDATE to $STDIN lets you enter the date record interactively or include it in a job file. Using a date record in the job file simulates the IBM //DATE OCL command.

You can enter the date (in the date record) in a flexible manner. The first number begins the date (the number does not have to start in column 1). Enter the date as six consecutive digits (`mmddyy`) or in three sets of two digits (`mm`, `dd`, and `yy`). Enter the month number (`mm`), the year (`yy`) then the day (`dd`).

If `mm`, `dd`, and `yy` are all equal to zero, or the file RPGUDATE is equated to $NULL, RPG uses the system date (this is a technique for defaulting to the system date without removing the F from column 17 and recompiling the program).

The following lines show valid ways to enter date records. Assuming the system date is 04/10/88, all of the examples produce this date. Since the first and last dates are zeros, they default to the system date:

```
000000
041088
//DATE 041088
DATE is 041088
FORMAT OF DATE IS MMDDYY = 041088
4/10/88
MONTH IS 4; DAY IS 10; YEAR IS 88
0 0 0
```

The following are examples of invalid date records:

```
4/10            (the year is missing)
4/10/1988       (the year must be 00 to 99)
040088          (the day must be 01 to 31)
```

If the date file does not exist, is empty, or contains invalid records, the program ends immediately and an error message is displayed.

Example

The following job file executes the program, MYPROG, which reads the current date from the job file. The date is 041088:

```
!FILE RPGUDATE = $STDIN
!RUN MYPROG
USWITCH: U1 = ON
**
//DATE 041088
```

Line Number Option (Column 20)

This field determines whether the source program line number is displayed when run-time errors occur. When this option is enabled, RPG keeps track of each source line number that is executed. Disabling this option results in a smaller executable program and faster execution.

Column 20	Description
N	Do not keep track of source program line numbers.
blank	Keep track of source program line numbers.

Inverted Print (Column 21)

This field determines how numeric literals, edit code data, and UDATE dates are formatted. (UDATE and numeric literals are used in Calculation and Output Specifications. Edit codes are used in Output Specifications.)

Column 21	Description
D	United Kingdom format.
I	European format.
J	European format with leading zero.
blank	Domestic format.

D (United Kingdom Format)

Periods are used as decimal points; the thousands positions are separated by commas. Leading zeros are suppressed. Dates are printed in the format `dd/mm/yy`.

I (European Format)

Commas are used as decimal points; the thousands positions are separated by periods. Leading zeros are suppressed. Dates are printed as `dd.mm.yy`.

J (European Format with Leading Zero)

Commas are used as decimal points; the thousands positions are separated by periods. Leading zeros are suppressed up to one position before a decimal point. Dates are printed in the format `dd.mm.yy`.

Blank (Domestic Format)

This format is used in the United States. Periods are used as decimal points in numeric literals and edited data; thousand-units are separated by commas. Leading zeros are suppressed. When used with Edit Code Y, dates are printed in the format `mm/dd/yy`.

Table 3-1 shows the various Inverted Print options and the different results they produce.

Table 3-1. Results of the Inverted Print Options

Column 21	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
D	United Kingdom	1986.19	9,762.55	.14	24/06/86
I	European	1986,19	9.762,55	,14	24.06.86
J	European (w/lead zero)	1986,19	9.762,55	0,14	24.06.86
blank	Domestic	1986.19	9,762.55	.14	06/24/86

Record Number Adjust (Column 22)

This field gives the adjustment to relative record numbers before they are used to retrieve records in direct-access MPE and Keyed Sequential Access Method (KSAM) files (the *KSAM/3000 Reference Manual* discusses the use of relative record numbers with KSAM).

This field has no effect for TurboIMAGE files.

Column 22	Description
1	RPG subtracts one from all relative record numbers obtained from an ADDROUT file, chaining file, or chaining operation before using the numbers for direct file access. This option is ignored for KSAM files built with the FIRSTREC=1 option.
+	RPG adds one to all relative record numbers obtained from an ADDROUT file, chaining file, or chaining operation before using the numbers for direct file access. This option does not apply to KSAM files built with the FIRSTREC=1 option.
0 or blank	RPG does not adjust relative record numbers before using them for direct file access.

Program Name Logging (Column 25)

This field determines whether a program identification line is written to the standard list device ($STDLIST) when the program begins and when it ends normally.

Column 25	Description
L	Write a program identification line on $STDLIST when the program begins and when it ends.
blank	Do not write program identification lines.

The beginning identification line lists the source program name entered in columns 75-80 of the Header Specification (or entered in the NAME= parameter of the $CONTROL compiler subsystem command) and the date and time.

Example

The program, AP320P, in the group PROGRAM and account AP is executed. Its source program name is AP320S. The beginning identification line looks like this:

```
Program: AP320S = AP320P.PROGRAM.AP   Wed, Nov 02, 1988, 07:40 AM
```

The ending identification line looks like this:

```
Pgm-End: AP320S = AP320P.PROGRAM.AP   Wed, Nov 02, 1988, 08:10 AM
```

Alternate Collating Sequence (Column 26)

This field lets you use a collating sequence other than the ASCII Collating Sequence (or JIS-Japanese Industrial Standard-Katakana) for certain comparison operations. Alternate collating sequences do not affect control levels, numeric comparisons, table look-up operations, or the SORTA operation.

Column 26	Description
E	EBCDIC (Extended Binary Coded Decimal Interchange Code) collating sequence applies.
K	EBCDIK collating sequence applies.
O	Alternate collating sequence, specified in octal format, applies.
S	Alternate collating sequence, specified in hexadecimal format, applies.
blank	Normal ASCII (JIS) collating sequence applies.

E (EBCDIC)

An E in the Alternate Collating Sequence field selects the Extended Binary Coded Decimal Interchange Code (EBCDIC). Use this when you need to read data prepared on EBCDIC-based computer systems or when the data must be interpreted in that sequence.

In EBCDIC, alphabetic characters are lower than numeric characters. In ASCII sequence, numbers are lower than alphabetic characters. Codes for special characters are also different for the two collating sequences.

O (Alternate Collating Sequence - Octal)

You must enter one or more ALTSEQ records that specify (in octal) the alternate sequences for specific characters. (Refer to Appendix D for ASCII and octal equivalences.) You can enter ALTSEQ records in a job stream or in a separate disk file. Placing them in a separate file means that other programs can use them also.

Table 3-2 shows the format for ALTSEQ records in octal.

Table 3-2. ALTSEQ Record Description for Octal Format

Column	Value	Description
1-6	ALTSEQ	This is an ALTSEQ record.
7-8	blank	None.
9-14	Two ASCII (JIS) characters, specified by sequence number in octal.	The first character (columns 9-11) replaces the second character (columns 12-14) in the ASCII (JIS) Collating Sequence.
15-80	Two ASCII (JIS) characters, specified by sequence number in octal.	Same meaning as columns 9-14, repeated as needed.

Example

Assume that the letter A (octal 101) must be changed to the letter Z (octal 132) and the letter B (octal 102 must be changed to the letter Y (octal 131). Y and Z retain their normal values. The upper case alphabetical character set would look like this using the alternate collating sequence:

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

The following ALTSEQ record specifies this alternate sequence for A and B:

```
ALTSEQ  132101131102
```

Figure 3-2 shows how to enter specifications that use an alternate collating sequence expressed in octal. Line 1 contains an O in column 26 to select the alternate collating sequence in octal. Line 2 contains an Array/Table File Name Specification (A) to name the file (FILEB) containing the ALTSEQ record shown above.

```
          1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
          ___________________________________________________________________________________

(1)       H                   O
            .
            .
            .

(2)       AFILEB
```

Figure 3-2. Specifying an Alternate Collating Sequence in Octal Code

S (Alternate Collating Sequence - Hexadecimal)

You must enter one or more ALTSEQ records that specify (in hexadecimal) the alternate sequences for specific characters. (Refer to Appendix D for ASCII and hexadecimal equivalences.) You can enter ALTSEQ records in a job stream or in a separate disk file. Placing them in a separate file means that other programs can use them also.

Table 3-3 shows the format for ALTSEQ records in hexadecimal.

Table 3-3. ALTSEQ Record Description for Hexadecimal Format

Column	Value	Description
1-6	ALTSEQ	This is an ALTSEQ record.
7-8	blank	None.
9-12	Two ASCII (JIS) characters, specified by sequence number in hexadecimal.	The first character (columns 9-10) replaces the second character (columns 11-12) in the ASCII (JIS) Collating Sequence.
13-80	Two ASCII (JIS) characters, specified by sequence number in hexadecimal.	Same meaning as columns 9-12, repeated as needed.

Example

(This is the same example as shown for Alternate Collating Sequence code O except that hexadecimal notation is used to specify the alternate sequence.)

Assume that the letter A (hexadecimal 41) must be changed to the letter Z (hexadecimal 5A) and the letter B (hexadecimal 42) must be changed to the letter Y (hexadecimal 59). Y and Z retain their normal values. The upper case alphabetical character set would look like this using the alternate collating sequence:

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

The following ALTSEQ record specifies this alternate sequence for A and B:

```
ALTSEQ  5A415942
```

Figure 3-3 shows how to enter the specifications that process an alternate collating sequence in hexadecimal. Line 1 contains an S in column 26 to select the alternate collating sequence in hexadecimal. Line 2 contains an Array/Table File Name Specification (A) to name the file (FILEA) containing the ALTSEQ record shown above.

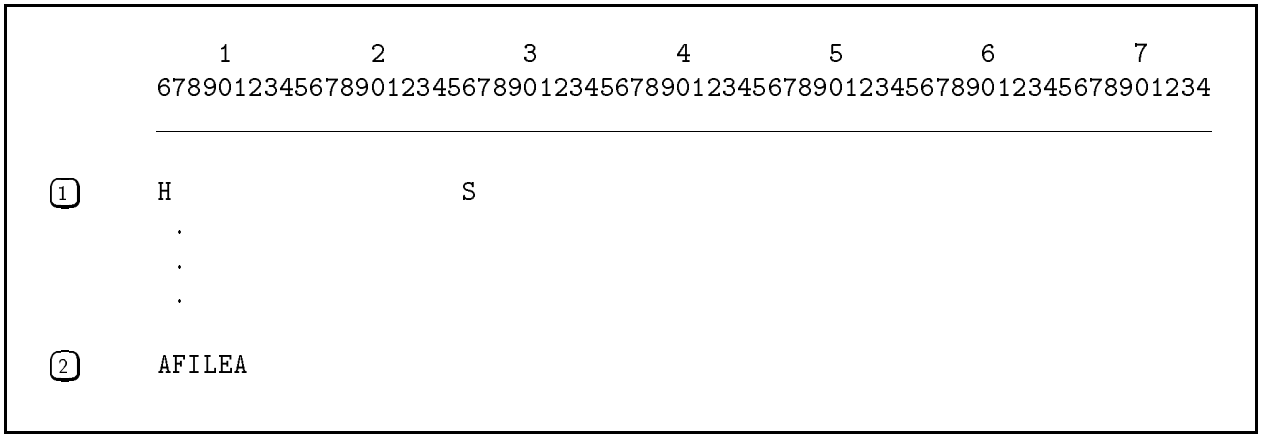


Figure 3-3. Specifying an Alternate Collating Sequence in Hexadecimal Code

BUFCHK Defaults (Column 28)

This field lets you specify the BUFCHK options to apply to all files. The options are: Current Data Checking (CDC), No-Read Checking (NRC), and Update-Protect Checking (UPC). You can override these values for individual files by using the Option Type Field of the File Description Continuation line (see the BUFCHK entry for the Option Type Field in Chapter 4).

Column 28	Description
B	Enable both CDC and NRC.
C	Enable CDC for all Sequential MPE and KSAM files that specify a LOCK File Description Continuation line and that are linked to another file at run time by MPE FILE equations.
N	Enable NRC for all update files.
U	Enable UPC and NRC for all update files.
X	Enable CDC, NRC, and UPC.

Table/Array Look-Up (Column 34)

This field specifies how tables and arrays are searched.

Column 34	Description
B	Perform binary look-ups. Binary look-ups are ideal for large tables and arrays.
blank	Perform sequential look-ups. Sequential look-ups are effective for small tables and arrays (less than 16 entries) and for tables and arrays whose initial entries are the most frequently searched.

B (Binary)

When you enter B, the tables and arrays must be in ascending sequence.

A sequential look-up is performed (regardless of this B entry) for execution-time tables/arrays and for compile-time and preexecution-time tables/arrays that are loaded "short" (that is, not all entries are initialized).

EBCDIC Zone/Digit Tests (Column 39)

This field specifies whether record identification codes that use the zone (leftmost 4 bits) or digit (rightmost 4 bits) portions are converted to EBCDIC before being compared. See the Record Identification Codes Field (columns 21-41) in Chapter 7 for a complete description of the Z (zone) and D (digit) entries.

This field provides compatibility with EBCDIC-based computer systems.

Column 39	Description
E	Convert the record identification codes (defined as types Z and D in the Record Identification Codes Field (columns 21/41) of the Input Specification) to EBCDIC before comparing them. Do not enter E if the record identification codes contain packed numeric data.
blank	Use the original ASCII character representation for record identification code testing.

Sign Processing (Column 40)

This field determines whether the signs in numeric fields are changed when the fields are moved. You can use this field, for example, to change a +1 to the ASCII character 1; the number is written as 1 rather than the character A.

The results produced by this field depend on the data format of the fields that are moved (see the Data Format field (column 43) in Chapter 7). This field affects only positive and unsigned fields. It does not apply to fields having L (left-sign) and R (right-sign) data formats.

Column 40	Description
B or blank	Sign processing takes place only when writing unpacked numeric fields. Original signs are retained on all other field moves.
I	Sign processing does not take place for unpacked numeric input fields. Sign processing does take place for the Calculation Specification operations MOVE, MOVEL, MLLZO, and MHLZO, when alphanumeric fields are moved to numeric fields. It also takes place when unpacked numeric fields are written.
N	No sign processing takes place. Thus, original signs are retained whenever fields are moved.
O	Sign processing does not take place for unpacked numeric output fields. Sign processing does take place for the Calculation Specification operations MOVE, MOVEL, MLLZO, and MHLZO, when alphanumeric fields are moved to numeric fields. It also takes place when unpacked numeric fields are read.
S	Standard sign processing takes place during Calculation Specification move operations (MOVE, MOVEL, MLLZO, and MHLZO) when alphanumeric fields are moved to numeric fields. It also takes place when unpacked numeric fields are read or written.

Table 3-4 shows how this field affects signs in numeric fields. Positive fields have sign bits = 1100. Unsigned fields have sign bits = 1111. The asterisk (*) refers to the MOVE, MOVEL, MLLZO, and MHLZO Calculation Specification operations. No sign processing occurs for MOVEL when the Result Field length equals or exceeds the length of the Factor 2 Field.

Table 3-4. Sign Processing Options

Column 40	Input unpacked numeric fld	Move* alphanumeric to numeric fld	Output unpacked numeric fld
B or blank	Signs unchanged	Signs unchanged	Positive sign-> Unsigned
I	Signs unchanged	Unsigned-> Positive sign	Positive sign-> Unsigned
N	Signs unchanged	Signs unchanged	Signs unchanged
O	Positive sign-> Unsigned	Unsigned-> Positive sign	Signs unchanged
S	Positive sign-> Unsigned	Unsigned-> Positive sign	Positive sign-> Unsigned

Form Positioning (Column 41)

This field lets the operator align printer forms before processing begins for printer files.

Column 41	Description
1	Perform printer forms alignment at the start of printer file processing. Allow forms alignment until the operator signals that the forms are positioned properly.
blank	Do not perform forms alignment.

1 (Forms Alignment)

Use this option when using preprinted forms such as paychecks. Before printer file processing begins, a forms-alignment record is printed and a message is displayed to the operator. A pause gives the operator time to adjust the forms, if necessary. When the forms are positioned correctly, the operator resumes the program (the report page count is incremented only when the program resumes). The first line is printed according to the carriage control method entered in the Carriage Control Type Field (column 53).

Forms-positioning takes place for both spooled and nonspooled output files.

Indicator Setting (Column 42)

This field lets you specify how certain field and resulting indicators are initialized at the beginning of the program and following Blank-After operations (see the Blank After Field, column 39, of the Output Specification).

The indicators affected by this field are:

- Those entered in the Zero or Blank Subfield (columns 69-70) of Input Specifications.
- Those entered in the Equal Subfield (columns 58-59) of any of these Calculation Specification operations: ADD, Z-ADD, SUB, Z-SUB, MULT, DIV, MVR, or XFOOT.

Column 42	Description
B	Do not alter the indicator settings at program initialization or following Blank-After operations.
S	Turn ON the indicators at program initialization and following Blank-After operations.
T	Turn ON the indicators at program initialization, but not following Blank-After operations.
blank	Do not alter the indicators at program initialization; turn them ON following Blank-After operations.

If you leave this field blank or enter a B into it, the indicators are OFF at program initialization, even though RPG sets their associated fields to zeros or blanks. If you want an indicator used in Input and Calculation Specifications to reflect its associated field's actual initialized state, enter S or T into this field.

If the indicator is associated with a Blank-After field (see the Blank After Field (column 39) of the Output Specification), it is turned ON after the field is cleared to zeros or blanks. If you don't want the indicator to turn ON (to simulate the Blank-After feature of other RPG implementations such as IBM System/36), enter either B or T in this field.

File Translation (Column 43)

This field lets you translate one or more characters in an input file before they are processed by the program. You can also translate characters before they are written to an output file.

The translation feature is useful for processing files containing non-ASCII data. It is also useful for encoding sensitive data on reports. Translation is performed twice (once on input and once on output) for combined input-output and update files.

To translate characters, enter one or more file translation records which specify (in hexadecimal or octal) the characters to translate and their translated equivalents.

Column 43	Description
F	Translate characters according to a file translation record specified in hexadecimal format.
O	Translate characters according to a file translation record specified in octal format.
blank	Do not translate characters.

F (Hexadecimal)

Enter one or more file translation records that specifies (in hexadecimal) the characters to be translated and the translated equivalents. (See Appendix D for ASCII and hexadecimal equivalents.) You can enter file translation records in a job stream or in a separate disk file. Placing them in a separate file allows others to use them also.

Table 3-5 shows the format for file translation records in hexadecimal format.

Table 3-5. File Translation Records in Hexadecimal

Columns	Value	Description
1-8	*FILES *or* A valid file name used in a File Description Specification. File names can contain up to eight characters and must begin with a letter A-Z. The remaining characters can be letters or digits. Do not embed blanks in the name.	Translate all files used in the program. Translate the named file.
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 as columns 9-12, repeated as needed.

Example

A personnel report shows the salary curves of a company's employees. Since most of the employees recognize salary curves C and D, these curves are "camouflaged" by being translated into characters that only the personnel department understands. The character C (hexadecimal 43) is translated to X (hexadecimal 58) and the character D (hexadecimal 44) is translated to W (hexadecimal 57).

The file translation record looks like this:

```
FILEC  58435744
```

Figure 3-4 shows how to enter the specifications that use the file translation record shown above. Line 1 contains F in column 43 to specify file translation in hexadecimal. Line 2 contains an Array/Table File Name Specification (A) that names the file (FILEC) containing the file translation record.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
      ------------------------------------------------------------------------------------

(1)   H                                       F
        .
        .
        .

(2)   AFILEC
```

Figure 3-4. Specifying File Translation in Hexadecimal

O (Octal)

Enter one or more file translation records that specifies (in octal) the characters to be translated and the translated equivalents. (Refer to Appendix D for ASCII and octal equivalents.) You can enter file translation records in a job stream or in a separate disk file. Placing them in a separate file allows others to use them also.

Table 3-6 shows the format for file translation records in octal format.

Table 3-6. File Translation Records in Octal

Column	Value	Description
1-8	*FILES *or* A file name used in a File Description Specification. (File names can contain up to eight characters, beginning with a letter A-Z. The remaining characters can consist of letters and digits. Do not embed blanks in the file name.)	Translate all files used in the program. Translate the named file.
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 as columns 9-14, repeated as needed.

Example

A personnel report shows the salary curves of a company's employees. Since most of the employees recognize salary curves C and D, those curves are "camouflaged" on the report by being translated into characters that only the personnel department understands. The character C (octal 103) is translated to X (octal 130) and the character D (octal 104) is translated to W (octal 127).

The file translation record looks like this:

```
FILED  130103127104
```

Figure 3-5 shows how to enter the specifications that use the file translation record shown above. Line 1 contains O in column 43 to specify file translation in octal. Line 2 contains an Array/Table File Name Specification (A) that names the file (FILED) containing the file translation record.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

(1)  H                                        O
       .
       .
       .

(2)  AFILED
```

Figure 3-5. Specifying File Translation in Octal

Non-Numeric Digits (Column 44)

This field lets you move special characters, such as ! and %, to unpacked numeric fields. When you do this, the special characters are converted to digits using their numeric bit values (this usually results in the value zero). You can also convert invalid packed decimal digits to zero.

This field provides compatibility with other implementations of RPG.

Column 44	Description
P	Allow invalid digits in packed decimal fields to be converted to 0. Also enable capability provided by N below. Note that data may be modified, and invalid decimal digits are not trapped.
N	Allow non-numeric data (special characters) to be moved to numeric fields during input, data structure moves and during MOVE and MOVEL operations.
blank	Do not allow non-numeric data (special characters) to be moved to numeric fields and do not allow numeric fields to compare equal to *BLANK or *BLANKS. If either of these conditions occur, generate run-time error 13 (`INVALID NUMERICAL DATA ...`).

Dollar Sign Substitute (Column 46)

This field lets you substitute another character for the dollar sign ($) during output editing (see the Edit Code Field, column 38, and the Constant/Edit Word Field, columns 45-70, in the Output Specification).

Column 46	Description
A special character (not a letter or digit)	Use this character as a dollar sign during output editing. (Be sure to this character in edit words instead of $.)
blank	Use $ for the dollar sign.

This field makes edit words and codes more flexible. Edit words are fields that format output fields character by character. Edit codes are single-character codes which provide a general method of editing output data. In edit words and codes, the dollar sign is an insertion character. You may want to substitute another character for the dollar sign, for example, when you're producing a report that shows figures in another currency.

Skip-Suppress (Column 47)

For a printer file, this field lets you specify whether to skip to head-of-form during program initialization.

Column 47	Description
S	Suppress skipping. Printing begins at the current forms position. Use this field if you entered 1 in the Form Positioning Field (column 41). This minimizes the amount of paper required for forms alignment.
blank	Skip to a new page when the program begins. (Often, the forms are already positioned to top-of-form. In this case, no action is taken.)

DSPLY Options (Column 48)

This field specifies how the Calculation Specification operations, DSPLY and DSPLM, are performed.

Column 48	Description
B	Display the Factor 1 Field and the Result Field value; prompt for a new Result Field value on the next line, below the displayed value.
D	Display "DSPLY" and the Factor 1 Field on the same line; display the Res Field and prompt for a new Result Field value on the same line.
N	Display the Factor 1 Field; display the Result Field and prompt for a new Result Field value on the same line.
blank	Display "DSPLY", the Factor 1 Field and the Result Field on the same line; prompt for a new Result Field value on the next line, below the displayed value.

Record Length Check (Column 49)

This field specifies how record length errors are handled at run time.

Column 49	Description
E	When a record length error occurs, display this warning message: `ACTUAL<LOGICAL RECORD LENGTH FOR FILE xxxx` The programs halts with run-time error `#2 (Unidentified Record)`. See the Error Response Field (columns 56-71) for information on how to pre-respond to this run-time error. If you pre-respond with a 0 and the actual length is less than the length you specify, excess numeric characters in the buffer are treated as zeros and excess alphanumeric characters are treated as spaces.
N	When a record length error occurs, suppress the warning message and run-time error.
blank	When a record length error occurs, display the warning message below, suppress the run-time error, and continue processing: `ACTUAL<LOGICAL RECORD LENGTH FOR FILE xxxx`.

Page Overflow Test (Column 50)

For printer files, this field determines when page overflow is detected. Page overflow is detected before printing the *next print line* (the line contained in the line counter after skip-after and space-after operations for the current line).

Column 50	Description
P	Page overflow occurs when the next print line is beyond the defined overflow line. This option provides compatibility with other implementations of RPG.
blank	Page overflow occurs when the next print line is on or beyond the defined overflow line.

*PLACE Method (Column 51)

This field determines how *PLACE (see the Field Name Field (columns 32-37) in the Output Specification) is processed in the program.

Column 51	Description
1	When you use more than one *PLACE, each one repeats only the fields that have not been repeated by previous *PLACE entries. This option provides compatibility with other implementations of RPG.
blank	When you use more than one *PLACE, each one repeats all fields preceding it regardless of whetherthe fields were repeated by other *PLACE entries.

Cross-Reference Listing (Column 52)

This field lets you request a Cross-Reference listing during compilation. (You can also request a Cross-Reference listing using the MAP option of the $CONTROL compiler subsystem command.)

The *HP RPG Programmer's Guide* shows an example of a Cross-Reference listing. The Cross-Reference listing shows:

- The line numbers of the source lines that define each file name, field name, and indicator.
- The line numbers of source lines 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.
- The type and designation of each file.

Column 52	Description
X	Print a Cross-Reference listing.
blank	Do not print a Cross-Reference listing.

Carriage Control Type (Column 53)

This field selects the method by which printer forms are advanced in response to a skip request (see the Skip Field (columns 19-22) of the Output Specification). RPG determines whether a file is a printer file by the use of vertical spacing controls, such as space and skip, in the Output Specifications.

Column 53	Description
1	**LINE OPTION** (Simulates other implementations of RPG) Skip requests use actual line numbers. The initial forms position is line 1. Top-of-form is reached using a series of line-feeds. You can enter Line Counter Specifications to override the default form length and overflow line.
L	**LINE OPTION** (HP implementation - line 6 is the first line) Skip requests use actual line numbers. The initial forms position is line 6. Top-of-form is reached by a series of line-feeds (including top-of-form for the first page when the first page line number is less than 6). You can enter Line Counter Specifications to override the default form length and overflow line.
blank	**CHANNEL OPTION** (Logical channels) Skip requests use logical carriage control channel numbers, whose associated line numbers are defined in Line Counter Specifications. The initial forms position is Channel 1 (line 6). Top-of-form is reached by a formfeed to Channel 1.

Table 3-7 shows how this field and the Line Counter Specifications work together to accomplish forms control.

Table 3-7. Carriage Control Values

Column 53	Top-of-form (Line Counter),	Channels 2-12	Overflow Line	Forms Length
1	Line 1	Not used	OL as defined in Line Counter Specification (default is 60)	FL as defined in Line Counter Specification (default is 66)
L	Line 6	Not used	OL as defined in Line Counter Specification (default is 60)	FL as defined in Line Counter Specification (default is 66)
blank	CHAN1 value as defined in the Line Counter Specification (default is 6)	Channel values as defined in the Line Counter Specification (default is channel number times 5)	CHAN12 or OL as defined in the Line Counter Specification (default is 60	Not used

Textfile Sequence Check (Column 54)

This field is used for compatibility with RPG V. It lets you compile a program, without errors, that contains an S or N in this field (S and N are sequence-checking options).

Ensure that this field contains one of these RPG V values:

Column 54	Description
S	Sequence-check records in the text file using the sequence numbers in columns 1-5. Print a message if a sequence error occurs.
N or blank	Do not sequence-check the text file.

Error Log (Column 55)

This field lets you specify how RPG run-time errors are handled. (RPG run-time errors are listed in Appendix B.)

Column 55	Description
N	When an RPG run-time error occurs and a preselected response is entered for it in the Error Response Field (columns 56-71), suppress (do not write) the error message.
S	When an RPG run-time error occurs, write an error message to the operator's console (job mode) or to the user's terminal (session mode) and print an Error Dump (the *HP RPG Programmer's Guide* explains how to interpret Error Dumps). Ignore the Error Response Field and terminate the program.
blank	When an RPG run-time error occurs and a response is not entered for it in the Error Response Field, display the error message on the operator's console or on the user's terminal and let the operator or user select a response. When an RPG run-time error occurs and a response is entered for it in the Error Response Field, redirect the error message to the standard list device (the list device is normally a printer for job streams and the user's terminal for sessions) and perform the chosen response.

Blank

If an RPG run-time error occurs and you leave its corresponding position blank in the Error Response Field (columns 56-71) the error message is written to the operator's console or user's terminal. The operator or user must select one of the following actions:

- Continue processing.
- Skip the record or program statement causing the error and continue processing.
- Terminate the program.

To eliminate operator intervention when RPG run-time errors occur, enter either N or S into this field, or if you leave it blank, enter a pre-response for the error in the Error Response Field (columns 56-71).

Error Response (Columns 56-71)

This field lets you enter responses to one or more RPG run-time errors. The responses are executed automatically when the errors occur. The responses provide an alternative to operator intervention for the errors. When you use this field, you must leave the Error Log Field (column 55) blank or enter an N into it.

There are 17 possible RPG run-time errors. They are listed in Appendix B. To enter a pre-response to a specific RPG run-time error, choose a response character listed below and place it in the column that corresponds to the number of the RPG error.

Columns 56-71	Description
0	Redirect or suppress the error message (see the Error Log Field, column 55) and continue program execution.
1	Redirect or suppress the error message, skip (bypass) the error and continue execution.
2	Redirect or suppress the error message and terminate the program normall All control-level indicators (L1-L9 and LR) are turned on, totals are calculated, tables are written, and all files are closed.
3	Redirect or suppress the error message, close all files and terminate the program immediately. Do not turn on control-level indicators or calculate and write totals and table output.
4	Redirect or suppress the error message, print an Error Dump and terminate the program. Turn on all control-level indicators, calculate and write totals and tables and close all files.
5	Redirect or suppress the error message, print an Error Dump and terminate the program immediately. Do not turn on control-level indicators or calculate and write totals and tables.
blank	Display the error message to the operator and let the operator choose the error response. (The Error Log Field, column 55, must also be blank.)

Example

The following Header Specification directs RPG to log the "divide by zero" run-time error message to the standard list device, print an Error Dump and terminate the program immediately.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

H                                                                  5
```

Figure 3-6. Entering an RPG Run-time Pre-Response

Program Name (Columns 75-80)

This field contains the program name. The format of this field is discussed in Chapter 2.

The Header Specification Default Summary

If you leave the optional fields of the Header Specification blank, the default specifications shown in Table 3-8 apply.

Table 3-8. Header Specification Defaults

Columns	Field	Default Values
1-5	Sequence Number	No sequence number applies.
7-14	Error Dump File Name	Error Dump directed to the standard list device ($STDLIST).
15	Debug	Suppress all DEBUG operations.
16	USWITCH Source	Initialize using the job stream USWITCH records or interactive prompting
17	UDATE Source	Initialize using 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 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 Processing	Do not process signs.
41	Form Positioning	Do not align forms and do not pause for the operator to do this.

Table 3-8. Header Specification Defaults (continued)

Columns	Field	Default Values
42	Indicator Setting	Turn on the 1P and L0 indicators.
43	File Translation	Do not translate files.
44	Non-Numeric Digits	Do not allow non-numeric data to be moved to numeric fields.
46	Dollar Sign Substitute	Do not substitute a character for $ during output editing.
47	Skip-Suppress	Skip to Channel 1 (new page) of the printer's carriage control tape.
48	DSPLY Options	Print DSPLY and DSPLM literals and prompt for a new Result Field value underneath the old displayed value.
49	Record Length Check	For each record length error, display a warning message and continue program execution.
50	Page Overflow Test	Signal overflow when the current line is on or beyond the overflow line.
51	*PLACE Method	Repeat all preceding fields, including those produced by other *PLACE entries.
52	Cross-Reference Listing	Do not print a Cross-Reference listing.
53	Carriage Control Type	Use logical channel numbers for printer skip requests.
54	Textfile Sequence Check	Do not sequence-check textfiles.
55	Error Log	When an RPG run-time error occurs and a response is entered in the Error Response Field, write the error to the standard list device. If no response is entered in the Error Response F write the message to the operator's console or to the user's terminal.
56-71	Error Response	Send RPG error messages to the operator's console and let the operator choose a response.
75-80	Program Name	Assign the program name, RPGOBJ (unless you use the NAME option of the $CONTROL command).

4

File Description Specifications

The File Description Specification describes the general characteristics of files used in a program. These characteristics include:

- The file name.
- The type of file (such as input, output, or combined).
- The size of the logical records in the file.
- The format of the records in the file (fixed-length or variable-length).
- The class name of the device where the file resides (disk, tape, or other media).

You must enter a File Description Specification for each file except LDAFILE, RPGUDATE, and USWITCH files and those containing compile-time tables and arrays, ALTSEQ records and file translation records.

If you're processing KSAM or TurboIMAGE files, you must also include one or more File Description Continuation lines. The Continuation lines give additional information about the file. File Description Continuation lines are also discussed in this chapter. The *HP RPG Programmer's Guide* contains a complete discussion of processing KSAM and TurboIMAGE files.

The File Description Specification is identified by an F in column 6:

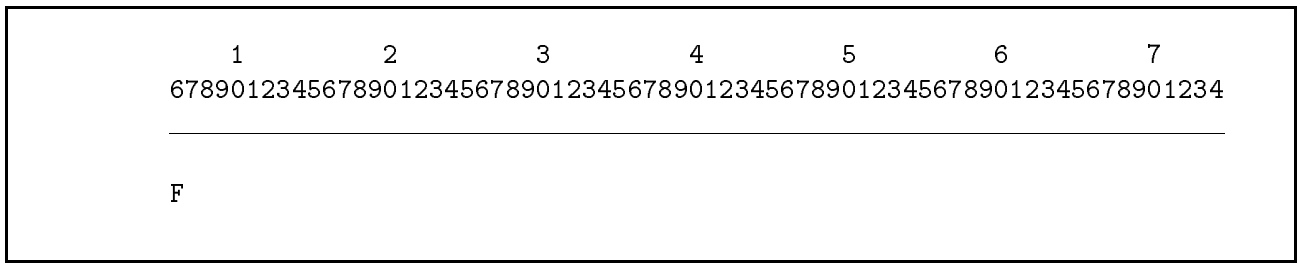


Figure 4-1. The File Description Specification

The File Description Specification Fields

The fields you can use in the File Description Specification are described in the sections which follow in this chapter. Each field has a unique name and occupies specific positions (columns) in the specification.

Sequence Number (Columns 1-5)

The Sequence Number Field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

This field contains an F to identify this line as a File Description Specification.

File Name (Columns 7-14)

This field assigns a name to a file. Use this name throughout the program to reference the file.

You can process MPE (sequential and direct), KSAM, and TurboIMAGE files in the program.

For compile-time tables and arrays, use the File Extension Specification to name them. If the tables and arrays are contained in a file, use Array/Table File Name Specifications to name the files.

Columns 7-14	Description
File name	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.) For TurboIMAGE files, this entry is the data set name.

You can process MPE files sequentially or randomly by relative record number.

You can process KSAM files both randomly and sequentially using a primary record key or one of fifteen alternate record keys. You can also process KSAM files in record number or chronological order. Each KSAM file has a key file and a data file. The key file contains a record directory (index) to the data file. KSAM files are analogous to Indexed Sequential Access Method (ISAM) files on other computer systems. For information about the structure and details of KSAM files, see the *KSAM/3000 Reference Manual*.

TurboIMAGE files are databases and their associated data sets and schemas. The TurboIMAGE subsystem (that interfaces RPG to TurboIMAGE) consists of a set of library procedures for accessing, modifying, and reporting database data. For details about TurboIMAGE, see the *TurboIMAGE/iX Database Management System* manual. When using TurboIMAGE files, these 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 that you're reading. (See the *TurboIMAGE/iX Database Management System* manual for a discussion of access levels.)
- Because TurboIMAGE files are accessible only at execution time, you cannot save compile-time or preexecution arrays and tables in them.

RPG supports all TurboIMAGE input modes (except Re-read) of the DBGET operation. Since some operations use DBFIND, RPG provides an additional input mode. It lets you read down a chain until the search key changes.

You can enter a TurboIMAGE data set name in either this field or in a DSNAME line (see the Data Set Name (DSNAME) Line in the File Description Continuation line). You must also enter the database name in a Database Name (IMAGE) File Description Continuation line.

Example

The following names are examples of valid file names.

```
PRINTER
A
TAPE2
D1234567
LIMBO
```

File Type (Column 15)

This field indicates how the file is used in the program. This field is required for all files.

Column 15	Description
C	Combined file
D	Display file
I	Input file
O	Output file
U	Update file

C (Combined File)

A combined file is used for both input and output. Existing records are read and records that are written are appended to the end of the file. Since input records remain unchanged, combined files are ideal for applications that accumulate data records.

When you use a combined file, enter Input and Output Specifications for it. The input and output records definitions can be different. The input definition determines the data that is read and the output record definition determines the data that is written.

Do not define KSAM or TurboIMAGE files as combined files.

D (Display File)

When using display files, you must enter DSPLY and DSPLM Calculation Specifications to read data from and to write data to them.

Display files use $STDLIST. If the program is running in session mode, $STDLIST is the user terminal; in job mode, it is the system console. You cannot change these assignments; MPE FILE equations are ignored for display files.

Do not enter Input or Output Specifications for display files and do not define KSAM and TurboIMAGE files as display files.

I (Input File)

Input files contain data that is read by the program. An input file can reside on tape or disk. Input files include preexecution time (but not compile-time) table and array files and Record Address Files (RAFs).

You must include one or more Input Specifications for each input file. If the input file contains a preexecution-time table or array or is a Record Address File, enter a File Extension Specification for it.

O (Output File)

Output files are used to write data to tape, disk, a printer or a terminal. They cannot be used to read data. Output files can contain preexecution-time tables and arrays.

Enter one or more Output Specifications for each output file. If you're using preexecution-time table and array output files, you must include File Extension Specifications for them. If you're creating a KSAM file, include a KEYFL File Description Continuation line to define its key file name and build parameters.

U (Update File)

Records in an update file can be read and updated. When a record is updated, it is read, modified, and written back to the same location in the file overwriting the old record. Update files can reside on disk or on any device with the device class name SPECIAL (see the Device Class Name Field, columns 40-46).

If you use update files, include Input and Output Specifications for them. The record definitions in the Input and Output Specifications should be the same.

File Designation (Column 16)

This field indicates the function of a file. You can use it for input, update, and combined files (see the File Type Field, column 15) and for output files used for chaining.

Column 16	Description
C	Chained file
D	Demand file
F	Full procedural file
P	Primary file
R	Record Address File (input only)
S	Secondary file
T	Preexecution-time table/array file (input only)
blank	Sequential output or display file

C (Chained File)

Chained files are input, output, and update disk files that are accessed randomly via Input Specification chaining fields or that are read directly by the Calculation Specification CHAIN operation. When you use CHAIN, input and update files are read and output files are written. You can write update files using Output Specifications or the Calculation Specification EXCPT operation.

Chained files cannot reside on devices with the device class name SPECIAL.

D (Demand File)

This is an input or update file that is processed sequentially by a Calculation Specification READ operation. For input demand files, use only the READ operation. For update demand files, use the READ operation for input and normal indicator processing for output.

F (Full Procedural File)

A full procedural file is an input or update file that can be processed both sequentially and randomly by the same key field. You can read records from a full procedural file using both the Calculation Specification CHAIN and READ operations. For example, you can use CHAIN to randomly read a record, then use READ to sequentially retrieve records from that point forward in the file. Specifying a file as full procedural is easier than specifying both a chained and a sequential file, and then equating them with two DSNAME File Continuation lines.

P (Primary File)

Primary files are the main files used for input. Use only one primary file in a program. A primary file can have an input, update, or combined file type. When there is only one input file, it is usually the primary file. However, it can also be a chained or demand file. If there are other input files in the program, the primary file is processed first and it controls the order in which the others are processed. See the Matching/Chaining Fields Field (columns 61-62) of the Input Specification for details about how primary files are processed during file matching.

R (Record Address File)

This is a sequential input file containing record numbers or keys. The record numbers or keys select records in another file. You can use Record Address Files (RAFs) to access records in KSAM and TurboIMAGE files.

Use only one RAF in a program and include a File Extension Specification for it. Do not enter Input Specifications for it or define it with the device class name SPECIAL (see the Device Class Name Field, columns 40-46).

You can create a RAF using any standard word processor or editor. The way that you enter record numbers or keys into lines in the RAF depends on how you're using them to access records.

If you want to use the RAF to access records sequentially within record number or key limits, enter a lower limit record number or key into the first position of a RAF line. Immediately follow it with the record number or key for the upper limit, with no intervening spaces. Enter as many lines as necessary. If you're entering record keys and the ones that you're accessing are in packed decimal format, enter the RAF keys in packed decimal format also. When a RAF is used to process a KSAM file within limits, processing begins with the lower limit record (or if it does not exist, the next higher record in sequence) and proceeds sequentially

until a record that matches or exceeds the upper limit is read or until end-of-file is encountered in the KSAM file. Since you can enter any number of upper and lower limit lines in a RAF, you can process the same records more than once, if necessary.

If you want to use the RAF to access records randomly, enter a record key in the first position of a RAF line. You can enter more than one key per line but do not separate them with spaces (spaces causes RPG to read the next RAF line). The number of keys per RAF line can vary, but the length of the keys must be the same. Keys in a RAF must have the same format and length as the keys in the file you're accessing. When you use a RAF to process a KSAM file randomly, RPG retrieves each key in the RAF and accesses the corresponding record in the KSAM file. Processing continues until end-of-file is encountered in the RAF.

See the *HP RPG Programmer's Guide* for examples of how to process a KSAM file sequentially and randomly using a RAF.

S (Secondary File)

You use a secondary file in programs that have more than one input file, to indicate that it is not the most important file in the input processing sequence. Secondary files can have input, update, or combined file types. You can enter as many secondary files in a program as you need. If you're not using matching files, secondary files are processed in the order in which you enter their File Description Specifications.

T (Preexecution Array/Table File)

This file contains one or more array or table. They are read into memory before the program begins execution. If you use preexecution tables/arrays, include File Extension Specifications for them.

End-of-File (Column 17)

For an input, update, or combined file, this field determines whether the program can end before all records in the file are processed. Do not use this field for demand, chained, or preexecution-time array/table files.

Column 17	Description
E	The program cannot end until it reads all records from this file.
blank	If other files contain an E in this field, the program can end whether or not it reads all records from this file. If this field is blank for input, update, and combined files, the program ends only when it reads all records from all of these files.

Use this field for files that you want to process in their entirety. For example, when producing a student attendance report for a certain time period, you need to read all student records in a master student enrollment file, those that have attendance transactions for the period and those that do not. You would enter an E in this field for the master student enrollment file and leave this field blank for the attendance transaction file.

Input Sequence (Column 18)

This field determines whether matching files are sequence-checked. Use this field only for update, combined, primary, and secondary input files.

This field is ignored if you do not use matching fields in the Input Specification.

Column 18	Description
A or blank	If this is a matching file, check the records for ascending sequence.
D	If this is a matching file, check the records for descending sequence.

The Matching/Chaining Fields (columns 61-62) of the Input Specification let you compare two or more input, update, or combined files to determine whether their records match. Records in all matching files must be in the same sequence (ascending or descending). When a record is out of sequence, an error message is printed. Depending on the entry in the Error Response Field (columns 56-71) of the Header Specification, the program either bypasses the record and continues, or it terminates.

The ASCII collating sequence is used for sequence-checking unless you define an alternate collating sequence or request the EBCDIC collating sequence (see Alternate Collating Sequence, column 26, and EBCDIC Zone/Digit Tests, column 39, in the Header Specification).

Record Format (Column 19)

This field specifies whether records in the file are fixed-length or variable-length. This field is required.

Column 19	Description
F	Fixed-length record format. Use this format for all printer and TurboIMAGE files.
V	Variable-length record format. Use this format for all WORKSTN, WORKSTN and WORKSTNC files.

Block Length (Columns 20-23)

This field determines the length of the blocks (physical groups of records) read from or written to tape files or written to disk files.

This field is ignored for disk input files; records are blocked according to the blocking information in the file label. It is also ignored for line printer and terminal files. In these cases, the block length is determined by the device type. The block length for a line printer file is one line (either 80 or 132 characters).

Columns 20-23	Description
1 - 9999 (right-justified, leading zeros are not required)	For tape and disk files, the length of the block in characters.
blank	No blocking; the block and logical record lengths are identical.

Data is read from files, and written to them, in units known as blocks. A block is transferred as a unit between the file's device and main memory. For tape or disk files, a block can consist of one or more logical records. For instance, a block can consist of 2 records or 256 records. Blocking for tape files reduces execution time and the amount of space required for the file on tape.

Be sure to enter the logical length of records in the Logical Record Length Field (columns 24-27). RPG uses the block and logical record lengths to determine the number of records in each block. This number is called the blocking factor and is always an integer.

By using the `:FILE` command with the REC parameter you can enter a different block length at run time to override the block length that you enter in this field (see the *MPE/iX Commands Reference Manual*). RPG does all the input/output for you when using a file equation to override compile-time options. For example, do not specify `NOBUF` in your file equation because you cannot do your own buffering.

Logical Record Length (Columns 24-27)

This field specifies the length of logical records in the file. This field is required.

Columns 24-27	Description
1 - 9999 (right-justified, leading zeros are not required)	For fixed-length records, enter the length of the record. For variable-length records, enter the length of the longest record in the file.

See the Record Length Check Field (column 49) of the Header Specification for information on how record length errors are handled at run time.

Processing Mode (Column 28)

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

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

L (Sequential Processing Between Limits)

You can process portions of KSAM files by processing them sequentially within limits. You supply the limits (boundaries) by entering the record key values for them. You can specify the lower limit using the Calculation Specification SETLL operation or you can enter sets of lower and upper limits in a RAF. Record keys do not have to match existing keys in the file. If a record for the lower limit key does not exist, the record with the next higher key is selected.

R (Random Processing)

You can access records directly in this mode; you do have to pass over records in order to Locate the records you want to process. Depending on the file organization (see the File Organization/Additional I/O Area Field, column 32), you can access a record randomly by supplying either a relative record number or a record key.

You can access KSAM, TurboIMAGE, and MPE direct files by relative record number. The File Organization/Additional I/O Area Field (column 32) must be blank for these files or contain a digit 1-7 or the letter D. You can use an ADDROUT file, the Calculation Specification CHAIN operation or a chaining file to supply the relative record numbers.

A relative record number is a series of digits that identifies the position of a record relative to the first record in the file. For TurboIMAGE files, the first record is Record 1. For KSAM files, the first record is either Record 0 or 1, depending on which one you choose when you create the file. For all other files, the first record is Record 0 (for example, record 0 is the first record in the file). You can change the beginning relative record number to 1 by entering a 1 in the Record Number Adjust Field (column 22) of the Header Specification.

You can use an ADDROUT file for storing relative record numbers. An ADDROUT file is a special type of RAF that contains four-character relative record numbers in fixed-length, binary records. ADDROUT files are created by XSORT (see the *RPG Utilities Reference Manual*). When using an ADDROUT or chaining file, RPG reads relative record numbers until the end-of-file in ADDROUT is reached.

You can access KSAM and TurboIMAGE files by record key. (A key is a string of characters that is used for finding individual records in a file; the string is matched with a field in the record.) To access a file by record key, the File Organization/Additional I/O Area Field (column 32) must contain I, M, or X. You can use a RAF or a chaining file to supply key values.

KSAM files have key files containing indexes that let you access records directly. KSAM indexes are updated automatically when you process the file. Record keys can be alphanumeric, packed, or unpacked numeric fields. Define the key field type using the Record Address Type Field (column 31). Specify the length of the key field and its starting position using the Record Address or Key Field Length Field (columns 29-30) and the Key Field Starting Location Field (columns 35-38).

Blank (Sequential Processing)

Sequential files are processed from beginning to end. If this is an MPE file, an ADDROUT file or in input KSAM file (the File Organization/Additional I/O Area Field, column 32, is blank, a digit 1-7, T, or C), records are processed in their physical order. If this is a KSAM or TurboIMAGE file (the File Organization/Additional I/O Area Field is I, M, or X), records are processed in key sequence.

Record Address or Key Field Length (Columns 29-30)

If this file is an input RAF, specify the length of the relative record numbers. If this is an input KSAM or TurboIMAGE file, enter the number of positions occupied by the record key.

Columns 29-30	Description
1 - 99 (right-justified, leading zeros are not required)	For a RAF, enter the length (in characters) of the relative record number or key field in the RAF; for an ADDROUT file, enter 4.
	For a KSAM or TurboIMAGE file, enter the length of the key field (the maximum key length is 99). If this is an input file controlled by a RAF, use the same key length as the RAF.
blank	This is not a RAF, KSAM, or TurboIMAGE file. Or, this is a KSAM or MPE file controlled by an ADDROUT RAF.

Record Address Type (Column 31)

This field describes the key field used for accessing a file by key. This field is required for MPE direct files and KSAM and TurboIMAGE files that are chained files or are accessed by a RAF.

Column 31	Description
A	This file is a KSAM or TurboIMAGE file processed by alphanumeric or numeric keys. Numeric fields are unpacked before chaining.
I	This file is a MPE direct, KSAM, or TurboIMAGE file that is processed by relative record number via chaining or a RAF; or this is a RAF.
K or P	This file is a KSAM or TurboIMAGE file processed by packed decimal numeric keys (do not use split chaining fields with the file); or this is an MPE direct file processed by relative record number. For TurboIMAGE files, the sign of numeric input fields is changed to C or D. Numeric Result Fields produced by Calculation Specifications have a C or D sign. Thus, if the sign of the key field in the TurboIMAGE file is not C or D, a record-not-found error occurs when chaining to the file.
blank	This is an MPE sequential file or a RAF; or it is an MPE direct, KSAM or TurboIMAGE file that is processed randomly by relative record number (the Processing Mode Field, column 28, is R).

File Organization/Additional I/O Area (Column 32)

This field specifies how the file is organized (for example, KSAM or TurboIMAGE) and it lets you select the number of buffers to use.

Column 32	Description
C	This is an input (only) KSAM file. The records are read chronologically.
D	This is an MPE direct file; assign two buffers to the file. (This entry is not required; it is provided for compatibility with other RPG systems.)
I or X	This is a KSAM file. An error occurs at run time if the file is not a KSAM file. (X provides compatibility with other RPG systems.)
M	This is a TurboIMAGE file; I, X, or any other entry defaults to M if IMAGE File Description Continuation lines are present.
T	This is an ADDROUT file, with two input/output buffers.
1 - 7	This is an MPE sequential or direct file; the specified number of buffers (1-7) are assigned to the file. Accepted (but not used) for TurboIMAGE files. The default number of buffers is 2.
8 or 9, blank	This is an MPE sequential or direct file; assign two buffers to the file. (Entries 8 and 9 are available for compatibility with other RPG systems.)

C (Chronological)

Records that have been marked as deleted (the first two characters are hexadecimal F's) are not bypassed when a KSAM file is read chronologically. You must include code in the program to bypass them, if necessary. You must reorganize the file to drop deleted records.

Table 4-1 summarizes the valid combination of entries for this field, the Processing Mode Field (column 28) and the Record Address Type Field (column 32).

Table 4-1. Valid Entries for Columns 28, 31, and 32

Processing Mode (Column 28)	Record Address Type (Column 31)	File Organization (Column 32)
blank (entire file)	blank (key not used)	blank (sequential)
blank (entire file) L (within limits) R (randomly)	A (alphanumeric key) P (packed key) I (record number) K (record key)	M (TurboIMAGE)
R (randomly)	I (record number)	D (direct)
blank (ADDROUT file)	blank (ADDROUT file)	T (ADDROUT file)
blank (entire file) L (within limits) R (randomly)	A (alphanumeric key) P (packed key) I (record number) K (record key)	C or I (KSAM)

Overflow Indicator (Columns 33-34)

When you use an overflow indicator to enable or suppress the output of overflow lines to a printer, you must name that overflow indicator using this field.

Enter an indicator only for line printer files or for disk files controlled by Line Counter Specifications and ultimately destined to be printed. Enter just one indicator per file and do assign that indicator to another file.

Columns 33-34	Description
OA OB OC OD OE OF OG OV	} } } } }-Assign the named indicator } } }
blank	Do not assign an overflow indicator

You determine the lines where output begins and ends on a printed page by entering a Line Counter Specification. The last line on a page is the overflow line. When the overflow line is reached and no overflow indicator is assigned to the file, the paper is advanced to top-of-form and normal output continues. If the overflow line is reached and an overflow indicator is assigned to the file, one of the following actions takes place (the overflow indicator is turned ON and OFF by Input and Calculation Specifications and it is turned ON each time the overflow line is reached):

Example Conventions

1. If the overflow indicator is ON, records associated with the indicator in the Output Specifications are printed at the bottom of the current page or at the top of the next page, or both. Associating the overflow indicator with output records in this manner is often used to condition the printing of totals and subtotals at the bottom of a page, or to print headings at the top of the following page.
2. If the overflow indicator is OFF, normal output continues on the line following the overflow line. No output records are printed and the paper is not advanced to top-of-form.

Key Field Starting Location (Columns 35-38)

This field defines the starting location of the key field in a KSAM file. You can enter the starting location of any one of the possible 16 keys for the file. This field is required when the file is processed by record key. If this is a TurboIMAGE file, do not use this field.

Columns 35-38	Description
1 - 9999	Starting position of the key field.
blank	A key field is not used.

Examples

Line 1 in Figure 4-2 shows how to specify sequential processing for an entire KSAM file. The record with the lowest key is processed first followed by records with successively higher keys, until end-of-file is encountered. Columns 15-16 are IS to define the file as an input, secondary file. Column 28 is blank to request sequential processing and columns 29-31 are 10P to specify that the key is a ten-digit packed field. Column 32 is I to indicate that this is a KSAM file. Column 38 is 4 to indicate that the key begins in position 4 of each input record.

Line 2 in Figure 4-2 shows how to specify sequential processing between limits for a KSAM file. Processing begins with a lower limit key in a RAF and continues sequentially until the record for the upper limit key is processed. Columns 15-16 are IP to specify that this is an input primary file. Column 28 is L to request sequential processing between limits and columns 30-31 are 6A to define the key as a six-position alphanumeric field. Column 32 is X (you can also use I) to define this as a KSAM file. And finally, column 38 is 1 to indicate that the key field starts in the first position of the KSAM file.

Line 3 in Figure 4-2 shows how to specify random processing for a KSAM file. (The keys of records to be processed are stored in a RAF.) Columns 15-16 contain UP to specify that this is an update, primary file. Column 28 is R to indicate that the file is processed randomly. Columns 30-31 are 2A to specify that the key is a two-position alphanumeric field. And finally, columns 37-38 contain the key field starting location for each record, 10.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
     ------------------------------------------------------------------------------

1    FFILE1   IS  F      80 10PI     4
2    FFILE2   IP  F 120  40L 6AX     1
3    FFILE3   UP  F     104R 2AI    10
```

Figure 4-2. File Description Specifications for KSAM Files

Extension Code (Column 39)

This field indicates whether File Extension and Line Counter Specifications are used for this file.

Column 39	Description
E	File Extension Specifications are included for the file. You must enter File Extension Specifications for RAFs, chaining files and execution-time tables and arrays.
L	Line Counter Specifications are included for the file. You must enter a Line Counter Specification if the program uses overflow sensing.
blank	File Extension and Line Counter Specifications are not used for the file. If included, they are ignored and a warning message is displayed.

Device Class Name (Columns 40-46)

This field associates a device class name or device identifier to the file. (Device class names and device identifiers are assigned during system configuration by systems personnel. Contact these individuals for specific assignments.)

This field is optional for display files, since the terminal is always used in session mode and the job file in job mode.

Columns 40-46	Description
Device identifier. (This can contain up to seven letters, digits and special characters.)	Device class name or logical device number of the device on which the file resides.
SPECIAL	The file is located on a device that requires a user-defined external subroutine to handle input/output. Enter the name of this routine in the Name of Label Exit Field (columns 54-59) and also furnish the routine itself.
$STDIN	A special device class that should only be used with input files.
$STDLST	A special device class that should only be used with display and output.
WORKSTN	A special device class used only with RPG for RPG Screen Interface (RSI) and VPLUS files (it is not defined by the operating system as a device class name). The file assigned to this device can have any legal file name, but must be defined as an update or combined file (WORKSTNC can have 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 displaying a form, reading data entered on the form or writing data to a file. For most applications, you define WORKSTN as an update demand file. Update demand files are processed by the Calculation Specification operations READ and EXCPT. Assign only one file in a program to WORKSTN. Specify that the file has variable-length records.
blank	The device class name, DISK.

Device Identifier

The device identifier can be a device class name or a logical device number. A device class name names the general type of device used for the file (the compiler does not verify that it is a valid name). A logical device number refers to a specific device, such as a particular printer or tape unit.

While the operating system permits up to eight characters for the device identifier, RPG permits only seven. If you're using a file that contains eight characters, enter a :FILE command at run time to equate the file to the device identifier.

You can override the name in this field by entering a :FILE command with the DEV= parameter.

Interface Type (Column 47)

This field defines the WORKSTN Interface to be used in the program.

Column 47	Description
C	This is an RPG Screen Interface (RSI) WORKSTN CONSOLE file.
R	This is an RPG Screen Interface (RSI) WORKSTN file.
blank	This is an RPG VPLUS Interface WORKSTN file.

Interface Control (Columns 48-52)

This field is used for options relating to WORKSTN file interfaces. See Chapters 10 and 11 for information on how it is used with VPLUS and RPG Screen Interface (RSI) files, respectively.

Disk Labels (Column 53)

This field indicates if a disk file has user labels and, if they do, how to process them. You can read and write up to nine user labels.

Column 53	Description
E	Process standard labels. Call a user-written routine to process a single user label, or bypass the label if the Name of Label Exit Field (columns 54-59) is blank.
S or blank	Process standard labels.
2 - 9	Process standard labels. Call a user-written routine to process the number of user labels specified by this entry or bypass them if the Name of Label Exit Field (columns 54 is blank).

Name of Label Exit (Columns 54-59)

This field names the user-written routine that processes user labels or files for SPECIAL devices.

Columns 54-59	Description
Routine name. (This name contains up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	The name of the user-written routine that processes user labels or SPECIAL files.
blank	The program does not process user labels or SPECIAL files.

Although standard labels are not supported for magnetic tape, you can process them with your own routine. Enter the name of the routine in this field and leave the Disk Labels Field (column 53) blank. You may want to use the system intrinsics, FREAD and FWRITE, in the routine to perform reading and writing. See the *MPE/iX Intrinsics Reference Manual* for details about these system intrinsics.

The following lines show how to enter statements in a routine written in the C language. You can use the system intrinsics, FREADLABEL and FWRITELABEL, in the procedure body to read and write the labels.

```
struct file_table {
          .
          .
};
void exitname (ptr)
    struct file_table *ptr;
{
         .
         .
     (procedure body)
         .
         .
}
```

exitname is the name of the procedure and ptr is the name of the pointer that gives the entry in the File Table for the file. (The File Table is an integer array and is described in the *HP RPG Programmer's Guide*.)

The following code shows how to enter a C language procedure to process a SPECIAL file:

```
struct file_table {
          .
          .
};
void exitname (ptr,type,returncode);
    struct file_table *ptr;
    int *type, *returncode;
{
         .
         .
    (procedure body)
         .
         .
}
```

Note

When coding in C, the procedure name must be in lowercase. If coding in Pascal, FORTRAN, or COBOL, the compilation process will do any downshifting necessary.

exitname is the name of the procedure and ***ptr** is the name of the pointer that gives the entry in the File Table for this file (see the File Table in the *HP RPG Programmer's Guide*). `TYPE` is a pointer to an integer parameter that contains one of the following: 0 (read a record), 1 (write a record), 2 (close the file), or 3 (open the file). `returncode` is a pointer to an integer that passes the outcome of the routine back to the RPG program. Ensure that it contains one of the following: 0 (normal file-processing occurred), -1 (the procedure encountered an error), or +1 (the procedure encountered end-of-file while reading the file).

After reading a record in the routine, place it in the buffer reserved for it by RPG. The buffer's length is determined by the Block Length Field (columns 20-23). The word-pointer to this buffer is found in the 26th word of the File Table.

For SPECIAL files with carriage controls specified in the Output Specification, the FPARAM entry (word 54) in the File Table points to a four-word integer array containing the skip-before, skip-after, space-before and space-after options for the current record.

File Addition (Column 66)

This field determines whether records added to an existing sequential file are written at the beginning (over-writing existing records) or are appended to the end (following existing records). You may use this field for KSAM and TurboIMAGE input and update files, though it is not required.

When you add records to KSAM and TurboIMAGE input and update files, the records do not have to be in key order nor do they have to be greater than the highest key in the file.

Column 66	Description
A	Append new records to the end of MPE sequential files; add new records to KSAM and TurboIMAGE files.
blank	Write new records to the beginning of MPE sequential and KSAM files; add new records to TurboIMAGE update files. Do not add records to other update files or to TurboIMAGE output files.

A (Append)

When you enter an A in this field, also enter ADD in Record Addition/Deletion Field (columns 16-18) of the field with the same name in the Output Specification. Alternatively, instead of entering an A in this field and ADD in individual record descriptions, you can enter a :FILE command at run time that includes the ACC=APPEND parameter.

Blank

If you're creating a KSAM file, enter a KEYFL File Description Continuation line for it.

Extents (Columns 68-69)

This field specifies the number of extents to be used for disk output files. (For disk input files, the number of extents comes from the disk file label).

Columns 68-69	Description
1 - 15 (right-justified, leading zeros are not required)	Create the disk file with this number of extents.
blank	Create the disk file with eight disk extents.

The operating system manages each disk file as a set of extents. Each extent is an integral number of contiguous disk sectors. All extents (except possibly the last) are the same size. You can override the extents entered in this field with a :FILE command containing the NUMEXTENTS parameter.

File Conditioner (Columns 71-72)

This field identifies the user indicators that condition this file.

Columns 71-72	Description
U1 - U8	Use the file only when this user indicator is turned ON.
blank	Use the file unconditionally.

U1-U8

When you enter a user indicator in this field, it must be ON at run time to read or write this file. You can enter a user indicator for any file except display, WORKSTN, and SPECIAL files.

This field is commonly used to assign a user indicator to an optional file. By turning the user indicator OFF before the program begins, you can bypass (not process) the file.

You turn user indicators ON and OFF by setting them interactively or by using USWITCH records or the system Job Control Word (JCW). Although you can change a user indicator's setting with Input, Calculation, and Output Specifications, its initial setting determines whether or not the file is used. If you initialize the user indicator with the system JCW, the final setting of the indicator is written back to the JCW when the program ends.

Program Name (Columns 75-80)

This field contains the program name. The format of this field is discussed in Chapter 2.

The File Description Continuation Line

When you use KSAM, TurboIMAGE, or WORKSTN files in a program, you must enter one or more File Description Continuation lines for them. The Continuation lines give additional information about the files, such as file locking requirements and database and data set names.

The remaining sections of this chapter give details about the Continuation lines that you can use. There are four different types of Continuation lines, each distinguished by the fields located in columns 54-74. These lines are summarized below:

- **General lines.**

  This line contains fields that specify error-handling procedures, exit routines, file locking, ASCII/EBCDIC conversion and partial field translation for a file. Enter as many of these lines as required for the file.

- **File-sharing line.**

  This line contains fields that associate more than one name to a single KSAM or TurboIMAGE file. Enter one of these lines per file.

- **Database lines.**

  This line contains fields that define KSAM files, TurboIMAGE databases, and TurboIMAGE data sets. Enter as many of these lines as required for the file.

- **WORKSTN interface lines.**

  This line contains fields that name the files and fields associated with the RPG Screen and VPLUS Interfaces. Enter as many of these lines as necessary for the file.

Sequence Number (Columns 1-5)

The Sequence Number Field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

This field contains an F to identify this line as a File Description Specification.

Leave columns 7-19 and 52 of each Continuation line blank.

Long Name Option Target (Columns 20-51)

This field is a longer alternative to the Option Target Field (columns 60-74). Use it only in DSNAME and ITEM Continuation lines when you need to enter more than 15 characters for a file or key name. Enter up to 16 characters in columns 20-35. Also enter an asterisk in column 60 to signal that you're using this field instead of the Option Target Field for the file or key name.

Continuation Code (Column 53)

This field identifies this as a File Description Continuation line. It is required.

Column 53	Description
K	This is a File Description Continuation line.

General Fields (Columns 54-74)

There are two general fields: the Option Type Field (columns 54-59) and the Option Target Field (columns 60-65). The entries in the Option Type Field determines the values that you can enter in the Option Target Field. For example, if you enter ERROR in the Option Type Field, you must enter an error routine name in the Option Target Field. The Option Type Field is described in detail in the pages which follow.

Option Type (Columns 54-59)	Description	Option Target (Columns 60-65)	Description
ASCII	The file is in ASCII format. (This entry is not required; it is provided for compatibility with other RPG compilers.)	(Not applicable)	(Not applicable)
BUFCHK	Enable/Disable the buffer integrity checking options CDC, NRC, and UPC.	N (in column 60)	Disables Current Data Checking (CDC). A blank enables CDC.
		N (in column 61)	Disables No-Read Checking (NRC). A blank enables NRC.
		N (in column 62)	Disables Update-Protect Checking (UPC). A blank enables UPC.
BYPASS	If an input/output error occurs, bypass the current logical record and increment the error counter named in the Option Target Field by one.	Field name.(This can contain up to six characters, beginning with a letter or @, $, or #; the remaining characters can be letters or digits.)	The name of the error counter field. It must be a five-digit numeric field with zero decimal positions.

Option Type (Columns 54-59)	Description	Option Target (Columns 60-65)	Description
EBCDIC	The file is in EBCDIC format. If it is an input file, it is translated into ASCII before processing. If it is an output file, it is translated into EBCDIC before being written.	P (in column 60)	A partial field translation is performed; input and output packed decimal and binary fields are not translated.
EBCDIK	The file is in EBCDIK format. If it is an input file, it is translated to JIS before processing. If it is an output file, it is translated into EBCDIK before being written.	P (in column 60)	A partial field translation is performed; input and output packed decimal and binary fields are not translated.
ERROR	If an input/output error occurs, transfer control to the user-written routine (named in the Option Target Field) to handle the error.	Routine name. (This can contain up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	The name of the routine that handles error processing.
FATAL	Provide user control over run-time fatal file errors.	Enter 0-5 in column 60.	See H specs columns 56-71 for a description of the response.
LOCK	Lock this file during input and output operations. (LOCK applies only to KSAM and MPE files.)	(Not applicable)	(Not applicable)

Option Type (Columns 54-59)	Description	Option Target (Columns 60-65)	Description
NOLOCK	Open this file so that concurrent users of the file can use LOCK. This option does not dynamically lock and unlock the file. (NOLOCK applies only to KSAM and MPE files.)	(Not applicable)	(Not applicable)
PARTTR	When you use file translation records to translate a record containing packed decimal or binary fields, those fields are not translated. (A partial file translation occurs.) If this is an EBCDIC input file, use the EBCDIC entry.	(Not applicable)	(Not applicable)
RDEXIT	Each time a record is read, transfer control to a user-written routine (named in the Option Target Field) that retains or bypasses the record.	Routine name. (This can contain up to six characters, beginning with a letter; the remaining characters can be letters or digits.)	The name of the routine that retains or bypasses the record.
RDSEQ	After adding records to a sequential chained file, reposition the record pointer to the next sequential record; do not leave the record pointer positioned to the record that was added.	(Not applicable)	(Not applicable)
blank	Options not used.	(Not applicable)	(Not applicable)

Option Type (Columns 54-59)

The following paragraphs describe certain options that you can enter in this field:

BUFCHK

If the Option Target Field contains a value other than blank or N, a compiler warning (2086W, 2087W, or 2089W) is displayed and the corresponding option is enabled. If you do not enter a BUFCHK Continuation line, the buffer option is determined by the BUFCHK Defaults Field (column 28) of the Header Specification. The buffer integrity checking options are described in the following paragraphs.

You may want to use Current Data Checking (CDC) when two or more files are equated to the same physical file via the MPE FILE command. CDC performs special input/output processing to ensure that all data buffers have current data. Current Data Checking is available only for KSAM and MPE files that have the locking facility enabled (the LOCK or NOLOCK Continuation lines are used).

You should use Current Data Checking when the same physical file is an update, output, or update/output file in the same program. When the physical file is equated to two or more files in the program using an MPE FILE equation, multiple access paths to the physical file are created. Since the operating system maintains separate data buffers for each of these access paths, output, and update operations may not be synchronized. Writing data from one buffer to the physical file can inadvertently overlay data (including records and record pointers) previously written from another buffer. This can result in corruption of the KSAM or MPE file.

No-Read Checking (NRC) ensures that output operations for the first record in an update file are executed only after the record is read.

Update-Protect Checking (UPC) ensures that the appropriate input record is placed in the file buffer before update operations (in Output Specifications) are performed on it. UPC prevents the following situations:

- Two or more consecutive update operations with no new input.
- For an update file having records added to it (the File Addition Field (column 66) of the File Description Specification is A), the interference of add and update operations for the same file. For example, since add and update operations share the same buffer, an add operation may "dirty" the buffer as follows: an update operation reads the record to be updated; the add operation adds a record to the file; the update operation formats the added record, rather than the original record, and writes it. If Update-Protect Checking is not specified for this file, this compiler message (2081I) results:

  ```
  FOR UPDATE-ADD FILES WITHOUT 'UPDATE-PROTECT CHECK', ENSURE THAT NO UPDATE IS
  INTERRUPTED BY AN ADD.
  ```

When you specify UPC, and a buffer conflict occurs, the program aborts with a `NO RECORD FOUND` run-time error and this message is displayed:

```
Attempted update on same record, or on intervening Add record.
```

ERROR, RDEXIT

The ERROR and RDEXIT routines must have a procedure head and body format similar to that shown below. The procedure shown below is written in the C language:

```
struct file_table {
            .
            .
};
void exitname (ptr,returncode);
    struct file_table *ptr;
    int *returncode;
{
            .
            .
     (procedure body)
            .
            .
}
```

The procedure name is `exitname`. The pointer to the pertinent File Table (File Table formats are discussed in the *HP RPG/iX Programmer's Guide*) for this file is `ptr`. `returncode` is a pointer to an integer that passes the outcome of the routine back to the RPG program. Ensure that it contains one of the following: 0 (continue the program without rewriting the record), 1 (bypass or rewrite the record), or 2 (abort the program).

Note

When coding in C, the procedure name must be in lowercase. If coding in Pascal, FORTRAN, or COBOL, the compilation process will do any downshifting necessary.

FATAL

You may use the FATAL continuation record option to pre-program an error response to RPG run-time error number one, FATAL FILE ERROR. The error responses allowed are 0-5, exactly the same options that are available for other run-time errors. If you specify FATAL, but put an invalid error response in column 60 or leave column 60 blank, RPG will default your error response to 0 (=CONTINUE). Note that this is different from not specifying FATAL at all, in which case RPG would default fatal file errors to error response 5 (=IMMEDIATE TERMINATION WITH DUMP).

The FATAL option is intended go give control to the RPG user at run-time in the event of a FATAL FILE ERROR. You must, therefore, specify a STATUS array to go with it. You would do this on a following continuation record, using the same syntax as the STATUS array for IMAGE files. Once a fatal file error occurs, RPG will post the error number from KSAM, MPE, or IMAGE to the first word of your status array. You then have the capability to check this word in your calculation specs, and perhaps take a variety of actions depending on the particular file error.

RPG will also post an "F" to the *ERROR field when a fatal file error occurs. You may wish to use this feature in conjunction with the STATUS array in your calculation specifications.

Note that the FATAL option is effective once your program enters the RPG cycle and not during RPG initialization time. FATAL allows you to catch and take action on file errors that occur once a program has begun execution (for example, I/O errors, IMAGE errors, etc.). If RPG cannot find or build a file at initialization time, it is assumed that the safest course to take is to abort the program, as there may be data integrity issues related to running with some of the expected files open.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

H
FINPUTA  IP  F      80            DISK
FOUT2    O   F      80            DISK
F                                                 KFATAL 0
F                                                 KSTATUSSA
INPUTA   NS
I                                          2   6 FLD1
C                    EXCPT
C           *ERROR   DSPLY
C           SA,1     DSPLY
OOUT2    E                          20 "REC OUT"
```

Figure 4-3. RPG Program Using the KFATAL Option

LOCK

LOCK obtains exclusive access to a file during input/output operations. Use LOCK for programs that run concurrently and that update the same file. LOCK prevents simultaneous update of the same records in the file.

When LOCK is used, RPG opens the file with the dynamic locking facility enabled (to allow the file to be shared with other programs that enable locking) and performs automatic locking and unlocking each time a record is accessed as follows:

- It locks an input file before reading a record from it and unlocks it after reading the record.
- It locks an output file before writing a record and unlocks it after writing the record.
- It locks an update file before reading a record and unlocks it after the record is updated or before it locks and reads the file again.

  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.

When you use a DSNAME Continuation line (see the File-Sharing Fields (columns 54-74)) to link files to one physical file, enter one LOCK Continuation line following the first file that uses a DSNAME Continuation line (the first DSNAME line determines how the file is opened at run time and whether or not the dynamic locking facility is enabled). If you enter a LOCK Continuation line for some other file in the same DSNAME group, the following run-time error occurs:

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

NOLOCK

NOLOCK opens the file with the dynamic locking facility enabled (to allow other programs that enable locking to share the file), but does not lock and unlock the file (as described for the LOCK option above). If you want to lock and unlock the file, you must do so manually by using the LOCK and UNLCK Calculation Specification operations.

When you use a DSNAME Continuation line (see the File-Sharing Fields, columns 54-74) to link files to another physical file, enter one NOLOCK Continuation line following the first file that uses a DSNAME Continuation line (the first DSNAME line determines how the file is opened at run time and whether or not the dynamic locking facility is enabled). If you enter a NOLOCK Continuation line for some other file in the same DSNAME group, the following run-time error occurs:

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

File-Sharing Fields (Columns 54-74)

The File-Sharing Fields in the Continuation line let you assign the same file name to two or more files. You can then process the same physical file more than one way in the program. For instance, you can process it randomly by relative record number and sequentially by key. To do this, enter two File Description Specifications in the program and specify the appropriate type of access for each. Then, equate the file names in one of the following ways:

- Equate the file names by using an MPE FILE command. When you do this, RPG treats the files as separate entities, each having its own unique file number (assigned by the operating system) and current record pointer.
- Use a DSNAME Continuation line to equate the file names (see the DSNAME Field below). Enter a DSNAME line for each file to be equated. RPG opens one logical file (identified by one file number), to which both names apply. The file is opened once and all references to it use the same file number. The program can access the file using different access methods.

DSNAME (Columns 54-59)	Description	File Name (Columns 60-74)	Description
DSNAME	This is a DSNAME line 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 7-14 of the File Description Specification when opening the file; share the file number with other files having the same DSNAME file name. (These files access the same same logical file.)
DSNAME	This is a DSNAME line for file sharing.	* (in column 60)	Use the file name (up to 16 characters) specified in columns 20-35 of the Long Name Option Target Field.

When equating files using DSNAME, be sure you understand the affects of different access methods on the current record pointer (that keeps track of file accesses). For instance, a program cannot read a KSAM file sequentially by key and then chain to it by key without altering the sequential current record pointer. A program can read a KSAM file sequentially and chain to it by relative record number without affecting the sequential current record pointer. Table 4-2 shows how the type of access affects the setting of the current record pointer. (TurboIMAGE files are the same except that direct access by relative record number does affect the current record pointer.)

Table 4-2. How Access Type for KSAM Files Affects the Record Pointer

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 the same record.
Delete	Next key sequential record	Next record
Add	Not affected	Record added

Because of the nature of the RPG logic cycle, special care must be used when processing DSNAMEd files, particularly when one or more of them are used as primary or secondary files. During the initialization phase, the first record of the primary file and each secondary file is read into its local buffer. Processing then proceeds with the primary file, followed by the secondary files in the order given in the F-specifications. For example, assume we have a program with four files; A, B, C, and D. A is the input primary, B and C are input secondary (in that order in the F specifications), and D is input chained, DSNAMEd to C. The files are then processed by the logic cycle in order A, B, C. If either A or B use a field to chain to D, the record pointer is altered to C, since C and D share the same MPE file number. When it is time to process C as the second secondary, we will obtain its first record (from the initialization phase), followed by those records starting with the last setting of the record pointer. This may be avoided for non-IMAGE files by using MPE file equations to equate the files rather than DSNAMEing them, or by naming all files to be processed sequentially earlier in F specifications, before their use as a chained file.

Another approach when using DSNAME with an IMAGE data set is to enter an IMAGE and a DSNAME Continuation line for each file accessing the same data set. Enter the IMAGE lines with different dummy DB names, and the DSNAME lines with the name of the appropriate data set. Then use file equations to equate all the dummy DB names to the actual database name. A separate DBOPEN is executed for each access path to the database and data set and separate current record pointers are established.

Example

Figure 4-4 shows how to access a KSAM file two different ways. FILEA is a KSAM file that is processed sequentially by key (see line 1). FILEB is the same physical KSAM file but is defined as a separate file (see line 3). FILEB is processed randomly by relative record number. After each File Description Specification, include a DSNAME Continuation line that associates the same DSNAME file name to each file. In this example, the DSNAME file name is FILEK (lines 2 and 4). All file descriptions followed by a DSNAME record for FILEK share the same file number.

```
        1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

1       FFILEA   IP  F      80   AI    10 DISK
2       F                                          KDSNAMEFILEK
3       FFILEB   IC  F      80R  II       DISK
4       F                                          KDSNAMEFILEK
```

Figure 4-4. KSAM File Sharing

Database Fields (Columns 54-74)

The Database Fields require several Continuation lines. They give information about how KSAM files are locked and accessed. For TurboIMAGE databases, they give the database name, locking and access information.

Each of the Continuation lines are described in the sections which follow and are listed below along with a brief description:

- **Database Name (IMAGE) Line**

  Used to name TurboIMAGE databases.

- **Item Name (ITEM) Line**

  Identifies TurboIMAGE databases key fields.

- **Password (LEVEL) Line**

  Designates passwords for TurboIMAGE databases.

- **Data Set (DSNAME) Line**

  Identifies the TurboIMAGE data sets to be processed.

- **Input/Output Status Array (STATUS) Line**

  Allows user-error handling with TurboIMAGE databases.

- **Key File Name (KEYFL) Line**

  Identifies KSAM key files and specifies KSAM duplicate key processing.

Database Name (IMAGE) Line

This line names TurboIMAGE databases and specifies how they are accessed. It also specifies whether the database or data set is used concurrently by other programs and whether locking is used.

Column	Value	Description
Record Type: (columns 54-50)	IMAGE	This is a TurboIMAGE database name line.
Database Name: (columns 60-65)	Database name	Name of the database being accessed. The name can be six characters long.
Open Mode: (column 66)	B	Read/Write Shared Mode. The database is locked for the duration of the program.
	S	Read/Write Shared Mode. The data set is locked for the duration of the program.
	1	Read/Write Shared Mode. The database is locked and unlocked whenever a record is accessed in it.
	9	Read/Write Shared Mode. The data set is locked and unlocked whenever a record is accessed in it.
	R	Read/Write Shared Mode. The data record is locked and unlocked whenever it is accessed.
	L	Read/Write Shared Mode. Enable dynamic locking for the database.
	2	Update Shared Mode.
	3 or blank	Exclusive Mode.
	4	Semi-exclusive Modify Access Mode.
	5	Shared Read Access Mode (concurrent with Open Modes B, S, 1, 9, R, and L).
	6	Shared Read Access Mode (concurrent with Open Mode 4).
	7	Exclusive Read Access Mode.
	8	Shared Read Access Mode (concurrent with Open Mode 6).

Column	Value	Description
Input/Output Mode: (column 67)	2	Serial read. Records are read in their physical sequence (not key sequence).
	3	Backward serial read (the reverse of mode 2).
	4	Directed read. Records are read by relative record numbers.
	5	Chained read.
	6	Backward chained read.
	7	Calculated read.
	8	Primary calculated read.
	C	Chained sequential read.
	R	Backward chained sequential read.
	blank	Write, Output File. Data is written to the output file.

Open Mode Field (Column 66)

Open Modes 2-8 are equivalent to the MPE open modes 2-8. Open Modes B, S, 1, 9, R, and L are equivalent to MPE open mode 1 with the added capability of allowing automatic and user-controlled locking and unlocking at the database, data set, and data record level. (Open Mode 2 also allows locking and unlocking.)

Additional details about some of the values you can enter in this field follow:

1 (Read/Write Shared Mode)

Other programs that access the database concurrently must use Open Modes B, S, 1, 9, R, L, or 5. 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 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.

9 (Read/Write Shared Mode)

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 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.

R (Read/Write Shared Mode)

When you use R, enter 5, 6, C, or R in the Input/Output Mode Field (column 67). If you do not, the database is opened with Open Mode 1 (database locking per record). You must also enter an ITEM Continuation line for the file.

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 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.

L (Read/Write Shared Mode)

This mode lets you control all locking and unlocking (RPG does not perform automatic locking or unlocking). This mode is equivalent to using the KNOLOCK Continuation line for KSAM and other non-TurboIMAGE files.

2 (Update-Shared Mode)

The program cannot add or delete records or perform locking. Other programs that run concurrently and update the database must use Open Mode 2 or 6.

3 (Exclusive Mode)

This program, only, can read, write, and update the database.

4 (Semi-exclusive Modify Access Mode)

This is the same as Open Mode 3 except that other programs using Open Mode 6 can read the database.

5 (Shared Read Access Mode)

The program has read-only access to the database. Other programs running concurrently with Open Mode 1 have read/write shared access to it.

6 (Shared Read Access Mode)

The program has read-only access to the database. Other programs running concurrently must use Open Modes 2, 4, 6, or 8.

7 (Exclusive Read Access Mode)

This program, only, can read the database. It cannot write to the database.

8 (Shared Read Access Mode)

This program has read-only access to the database. Other programs using Open Mode 6 or TurboIMAGE DBSTORE routines also have read-only access to it.

Locking Precedence

If you define a database or data set more than once in a program, you must use the same locking precedence level (Open Modes B, S, 1, 9, R, and L) for each definition. For example, if a database is first defined with database locking for duration (Open Mode B) and later defined with record-level locking (Open Mode R), a conflict occurs. There is no reason to perform record-level locking on the database when the entire database is already locked for the duration of the program. When the Open Modes do not have the same locking precedence level, the compiler issues a warning and defaults the record-level locking to the locking precedence level of the first Open Mode specified (in this case, B).

The locking precedence levels follow. To avoid locking precedence level conflicts, consistently use the same locking precedence levels for the database or data set for each definition in the program. For example, if the highest locking level needed in the program is B, then all file definitions for the database or dataset should also be B.

```
B          - Highest level
S          - Middle level
1, 9, R, L - Lowest level
```

When a locking precedence level error occurs, the compiler makes the following adjustments:

- If the first Open Mode is of higher precedence than all of the succeeding Open Modes, the compiler issues a warning and defaults all lower precedence locking levels to the first one specified.
- If the first Open Mode is of lower precedence than the other locking precedence levels, a compiler error results.

If you need more than one outstanding lock, you must have Multiple RIN special capability. Be very careful to prevent deadlocks when executing with Multiple RIN. (See the *TurboIMAGE/iX Database Management System manual* for information on the MR capability.)

The *TurboIMAGE/iX Database Management System manual* has more information on locking levels and strategies.

Input/Output Mode Field (Column 67)

Additional details on some of the values you can enter in this field are listed below:

4 (Directed Read)

The chaining or record address fields must contain numeric relative record numbers. If no record exists for the record number, the H0 indicator is turned ON. (For the CHAIN operation in the Calculation Specification, the error indicator is turned ON.) The first record in the file is Record 1 (unlike MPE files, where the first record in the file is Record 0).

5 (Chained Read)

You must provide the key for each record read. Using the same key more than once causes the same record to be read. A DBFIND operation is done before each record is read. You must enter an ITEM Continuation line. If you want to read down a chain, use Input/Output Mode C.

6 (Backward Chained Read)

This is the same as Input/Output Mode 5 except that if duplicate keys exist, the last record having this key is read. You must enter an ITEM Continuation line. If you want to read up a chain, use Input/Output Mode R.

7 (Calculated Read)

This was formerly Associative Read. Use it with master data sets only. The record with a matching key is read.

8 (Primary Calculated Read)

This was formerly Primary Associative Read. It applies to master data sets only. The record key is used to locate the place where a record should be, then reads whatever record it finds at that location. Because of the way TurboIMAGE keys are hashed, the record that is read may not have the same key as that expected.

C (Chained Sequential Read)

This mode lets you use chaining or Record Address Files. With chaining, consecutive duplicate keys cause RPG to read sequentially down the chain. When there are no more records in the chain, the low resulting indicator (columns 56-57 of the Calculation Specification CHAIN operation) is turned ON. For input chaining, H0 is turned ON. With Record Address Files, RPG reads down the entire chain before processing the next key. Enter an ITEM Continuation line for this mode.

R (Backward Chained Sequential Read)

This is the same as Input/Output Mode C except that the chains are read in reverse sequence.

Item Name (ITEM) Line

This Continuation line defines TurboIMAGE item (key) names.

This line is required when the database has a Database Name (IMAGE) Continuation line containing 5, 6, 7, 8, C, or R in the Input/Output Modes Field (column 67).

Record Type (Columns 54-59)	Description	Item Name (Columns 60-74)	Description
ITEM	This is an IMAGE item (key) name line.	Item (key) name	The name of the key used for accessing this file. Use the same name defined in the database schema.
ITEM	This is an IMAGE item (key) name line.	*(in column 60)	The item name (up to 16 characters) is specified in columns 20-35 of the Long Name Option Target Field.

Password (LEVEL) Line

This Continuation line specifies the passwords that permit access to the database.

Since RPG processes TurboIMAGE records, not individual fields, the password must include permission to read and write entire records.

Record Type (Columns 54-59)	Description	Password (Columns 60-67)	Description
LEVEL	This is a password line	Password. (Maximum length of eight characters.)	A password that establishes a user-class identification that lets the program access the database.

If you do not enter this Continuation line, User Class 0 is used. (See the *TurboIMAGE/iX Database Management System Manual* for information on passwords.)

Data Set Name (DSNAME) Line

This Continuation line has two purposes:

1. It lets you process the same data set more than one way. For instance, you can process a data set randomly by relative record number and sequentially by key. To do this, you must define two files and use DSNAME Continuation lines to equate these files to the same physical data set.
2. It lets you use a longer data set name than the eight-character file name allowed in the File Name Field (columns 7-14) of the File Description Specification. The name that you enter overrides columns 7-14.

DSNAME (Columns 54-59)	Description	File Name (Columns 60-74)	Description
DSNAME	This is a DSNAME line 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 7-14 of the File Description Specification when opening the file; share the file number with other files having the same DSNAME file name. (These files access the same logical file.)
DSNAME	This is a DSNAME line for file sharing.	*(in column 60)	Use the file name (up to 16 characters) specified in columns 20-35 of the Long Name Option Target Field.

Input/Output Status Array (STATUS) Line

This Continuation line lets you use TurboIMAGE status arrays. Status arrays let you control processing for certain input TurboIMAGE errors. When you use this Continuation line, RPG does not automatically terminate the program when the positive errors 10, 11, and 14-17 are encountered. These errors are treated as exceptions; they turn ON the H0 indicator. You must process these errors yourself or leave the H0 indicator ON. If you leave it ON, the program ends on the next cycle unless you specify otherwise in the Error Response Field (columns 56-71) of the Header Specification. Negative errors and output errors always cause the program to terminate.

Record Type (Column 54-59)	Description	Array Name (Columns 60-65)	Description
STATUS	This is an input/output status array line.	Array name.	The name of the array. The array must be a six-element numeric array; each element containing ten digits with zero decimals positions.

Define the status array with the six elements (words) shown below. You do not need to define it further with a File Extension Specification.

1. Condition word. This is zero if no exception conditions occur. (Exception conditions are defined in the *TurboIMAGE/iX Database Management System manual.*) Otherwise, this word contains the TurboIMAGE error number (see Appendix B).
2. Length (in words) or the record 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.

Key File Name (KEYFL) Line

This Continuation line defines the name of the KSAM key file and, if the KSAM file is being created, the duplicate key options for the file.

Column	Value	Description
Record Type: (columns 54-59)	KEYFL	This is a KSAM key file name line.
Key File Name: (columns 60-67)	Key file name	Name to be given to the KSAM key file. Enter an 8-character name that conforms to MPE file naming conventions.
First Record Option: (column 68)	1	The first record is 1.
	0 or blank	The first record is 0.
Duplicate Key Option: (column 69)	D	Allow duplicate keys.
	blank	Do not allow duplicate keys.
Chronological Option: (column 70)	C	Maintain the chronological order of duplicate keys. Use this option only when the Duplicate Key Option Field is D.
	R	Add duplicate keys randomly to the key file (this improves performance; see the *KSAM/3000 Reference Manual.)*
	blank	The file does not have duplicate keys, or records are being added to it randomly (see R above).

WORKSTN Interface Fields (Columns 54-74)

This Continuation line specifies control parameters, files, and field names associated with WORKSTN (terminal) files. See Chapters 10 and 11 and the *HP RPG Programmer's Guide* for examples of how to use WORKSTN (VPLUS and RSI) files.

Option Type (Columns 54-59)	Description	Option Target (Columns 60-74)	Description
BATCH	The file name specified in the Option Target Field is the file to which data entered on the form is written during standard VPLUS data entry operations.	File name.	File name of the VPLUS batch file.
FIRST	The RSI form named in the Option Target Field is displayed during program initialization. (This option speeds the display of the first form in a program.	RSI form name.	The name of an RSI form contained in an RSI forms file. The forms file must be declared by a FORMS File Description Continuation line.
FORMDL	The number of forms specified in the Option Target Field determines the size of the VPLUS "form storage directory" used in forms downloading.	A three-digit number in columns 60-62. The number must be in the range 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 Forms Library created with SIGEDITOR.	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.

Option Type (Columns 54-59)	Description	Option Target (Columns 60-74)	Description
LOADFM	The array name in the Option Target Field contains the names of forms to download using the LOADFM VPLUS action (Do not use it for RPG Screen Interface files). The array is predefined as an alphanumeric compile-time array. Each entry is 16 characters long and the number of entries is determined by the number of forms that you enter in the FORMDS File Description Continuation line.	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.)	Compile-time array containing the forms to download. You can modify the contents of the array at run time. Form names must not exceed 15 characters and must be followed by a blank.
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. The field is predefined 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.

Option Type (Columns 54-59)	Description	Option Target (Columns 60-74)	Description
STATUS	The array name specified in the Option Target Field is used by the RPG Screen and VPLUS Interfaces to return status information. The field is predefined 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 that contains RPG Screen and VPLUS Interface status information.
TRACE	The file name specified in the Option Target Field is the VPLUS trace file which contains a record for every action, event or run-time error occurring during execution of the RPG VPLUS Interface.	File name.	File name of VPLUS trace file.
TRMID	The field name specified in the Option Target Field is the field which the RPG Screen Interface (not RPG VPLUS Interface) initializes with the terminal identification of the WORKSTN file. This is predefined 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.

Program Name (Columns 75-80)

This field contains the program name. The format of this field is discussed in Chapter 2.

The File Description Specification Default Summary

If you leave the optional fields in the File Description Specification blank, the default specifications shown in Table 4-3 apply.

Table 4-3. File Description Specification Defaults

Columns	Field	Default Values
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	If this is a matching file, check the records for ascending sequence.
20-23	Block Length	No blocking. The block and logical record lengths are identical unless this default is overridden by the file label or a :FILE command.
28	Processing Mode	Sequential processing.
29-30	Record Address or Key Field Length	This is not a RAF.
31	Record Address Type	The file is a direct-access file not processed by a RAF, or it is a sequential file.
32	File Organization/Additional I/O Area	This is a non\ADDROUT, direct-access file. Assign two buffers to the file.
33-34	Overflow Indicator	Do not assign an overflow indicator.
39	Extension Code	File Extension and Line Counter Specifications do not apply to this file.
53	Disk Labels	Process standard labels.
54-59	Name of Label Exit	No user labels or SPECIAL files are processed.
66	File Addition	Write new records to the beginning of the file.
68-69	Extents	Allow eight disk extents.
71-72	File conditioner	Use the file unconditionally.
75-80	Program Name	None.

5

File Extension Specifications

You must enter a File Extension Specification for every compile-time table and array, execution-time array, Record Address File (RAF), and chaining file used in a program. You enter the following information in a File Extension Specification:

- Table or array names, descriptions of entries in the table or array, and the names of files in which they reside or to which they are written.
- Chaining fields (used to process chained files) and the chaining and chained file names.
- The name of a RAF and the name of the file it accesses.

The File Extension Specification is identified by an E in column 6:

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

E
```

Figure 5-1. The File Extension Specification

The File Extension Specification Fields

The fields you can use in the File Extension Specification are described in the sections which follow in this chapter. Each field has a unique name and occupies specific positions (columns) in the specification.

Sequence Number (Columns 1-5)

The Sequence Number Field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

This field contains an E to identify this line as a File Extension Specification.

Chaining File Record Sequence (Columns 7-8)

This field gives the record sequence of the file and is used only with chaining files. (See the next field, Chaining Code Identifier (columns 9-10), and the Matching/Chaining Fields (columns 61-62) in the Input Specification for information on chaining files.) This field is not checked by HP RPG; it is available for compatibility with other implementations of RPG.

Enter the same value in the Group Sequence Field (columns 15-16) of the Input Specification.

Columns 7-8	Description
Two digits (00-99) or two letters (A-Z)	The same record sequence specified for the chaining file in the Input Specifications.
blank	None.

Example

Figure 5-2 shows how to assign the sequence number 01 to all records in the file ACCTS. A 01 is entered in this field and also in the Group Sequence Field (columns 15-16) of the Input Specification.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

E01C1ACCTS   DISKA                                    CHAINED FILE
  .
  .
  .

IACCTS   01                               I   50CODE    C1
```

Figure 5-2. Entering a Chaining File Record Sequence Number

Chaining Code Identifier (Columns 9-10)

This field identifies the chaining field used for chaining. Use this field only for chaining files.

Columns 9-10	Description
C1-C9	The chaining field code that identifies the chaining field in the Matching/Chaining Field of the Input Specification.
blank	The file is not a chaining file.

A chaining field is a field containing a record number or key that points to a related record in another file. For instance, a customer transaction file can contain a chaining field that points to customer addresses in an address file. Once customer addresses are accessed, mailing labels can be printed.

You can enter up to nine chaining fields or field combinations in a chaining file. Associate each chaining code to a field on an Input Specification using the Matching/Chaining Fields (columns 61-62) of the Input Specification. You can assign the same chaining code to more than one field.

This field is required if you use a chaining field code in an Input Specification. You can use the same chaining field code in more than one File Extension Specification if the associated files access the same chained file. The From File Names associated with the chaining field code must be different in each instance, but the To File Names must be the same. (The From File Name and the To File Name Fields are described in the next two sections.)

Example

Figure 5-2 shows how to assign the Chaining Code Identifier C1 to the field named CODE in the file ACCTS. This is done by:

1. Entering C1 in the Chaining Code Identifier Field of the File Extension Specification.
2. Entering C1 in the Matching/Chaining Fields (columns 61-62) of the Input Specification, next to the field CODE.

From File Name (Columns 11-18)

This field contains the name of a chaining file, preexecution-time table or array, or a Record Address File.

Columns 11-18	Description
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 file name must be defined in the File Name Field of a File Description Specification.)	The name of a chaining file defined in an Input Specification with a chaining code in the Matching/Chaining Field, *or* The name of a Record Address File (RAF) containing relative record numbers or record keys. *or* The name of the file from which a preexecution-time table or array is read.
blank	This is a compile-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.

Table and array files contain groups of related data arranged for efficient, systematic reference. Tables and arrays can be searched sequentially and randomly (see the Table/Array Look-Up Field (column 34) in the Header Specification). When you search a table or array sequentially, each entry (starting with the first) is read until the entry is found. When you perform a binary search on a table or array, entries are accessed directly. You do not pass through other elements looking for the one you need. For arrays, you can process the entire array by using only its name.

The types of tables and arrays are:

- **Compile-time tables and arrays**

  These tables and arrays are loaded during compilation and compiled with the source program. They become a permanent part of the object program. Compile-time tables and arrays are loaded in the order in which you enter the File Extension Specifications. Compile-time tables and arrays may be appended to the end of the source program or contained in separate disk files. If they reside in separate disk files, you must name those files using Array/Table File Name Specifications.

- **Preexecution-time tables and arrays**

  These tables and arrays are loaded from disk by the object program before the RPG logic cycle begins. When you use more than one preexecution-time table or array in the same file, they are read in the order in which you enter their File Extension Specifications.

- **Execution-time arrays**

  These arrays can reside in separate disk files or you can create them in the program. If they reside on disk, they are loaded at execution time the same way input data is loaded. You define them using Input Specifications. To create them in the program, use Calculation Specification operations. There are no execution-time tables in RPG.

For information about creating and loading tables and arrays, see the last section in this chapter titled "Tables and Arrays".

To File Name (Columns 19-26)

This field contains the target file for the chaining file or RAF entered in the From File Name Field. This field can also contain an output disk or tape file to which compile-time and preexecution time tables and arrays are written when the program ends. (Execution-time arrays cannot be written when the program ends.)

Columns 19-26	Description
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 chained file processed by the chaining file named in the From File Name Field, *or* The name of the input or update file processed by the RAF named in the From File Name Field, *or* The name of a sequential output file to which the compile-time or preexecution-time table or array named in the Table/Array Name Field (columns 27-32) is written when the program ends.
blank	No file is used. Tables and arrays, if any, are not copied.

When you enter an output file for tables and arrays, the tables and arrays (except those destined for a printer or controlled by Line Counter Specifications) have the same format as used for input. If you're writing more one table or array to the file, RPG writes a separator record after each one. The record has asterisks in positions 1-2 and a blank in position 3. When you use alternating tables, they are written as alternating fields. If you write data to the file also, the tables and arrays immediately follow the data with no separator records.

Tables and arrays destined for the line printer or that are controlled by Line Counter Specifications, have a unique format and an identifying header.

You can rearrange table and array output by using an Output Specification.

The record size for the file you enter in this field must be large enough to accommodate the number you enter in the Entries Per Record Field (columns 33-35). If the record size is larger, entries are written until the record is full.

Table/Array Name (Columns 27-32)

This field names a table or array used in the program.

Columns 27-32	Description
Table or array name. (Table names can contain from three to six characters, beginning with TAB; array names can contain from one to six characters, beginning with a letter or one of the special characters @, $, or #. For 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 that is searched in the program.
blank	Tables and arrays are not used.

The table or array name that you enter in this field is the one that you use to reference the table or array elsewhere in the program. You can reference tables in Calculation and Output Specifications and you can reference arrays in Input, Calculation and Output Specifications.) You can name any kind of table or array in this field: compile-time, preexecution-time, or execution-time. Table and array names, however, must be unique.

The table or array that you enter in this field can have an alternating table or array associated with it. Define the alternating table or array in columns 46-57 of this specification.

Entries Per Record (Columns 33-35)

This field contains the number of table or array entries in each input record of a compile-time or preexecution-time table or array.

Columns 33-35	Description
1-999 (right-justified, leading zeroes are not required)	The number of table/array entries in each table or array input record.
blank	This is an execution-time array.

Each record (except the last) in the table or array file, must contain exactly the number of entries you specify. The last record may contain fewer entries, but not more.

When you use an alternating table or array, every entry in the table or array is associated with the same entry in the alternating table or array. You must format the input records for the tables or arrays in alternating form. For example, if TABLEB is the alternating table for TABLEA, entry A1 corresponds to B1, A2 corresponds to B2, and so forth.

TABLEA	TABLEB
A1	B1
A2	B2
A3	B3
A4	B4

Each pair of corresponding entries (for example, A1 and B1) must appear in the same input record in alternating form; you cannot split them between records. Entries in the alternating pair are treated as one table entry. Entries in TABLEA and TABLEB are entered on an input record as shown below:

```
A1B1A2B2A3B3A4B4
```

For TABLEA and TABLEB, the number of entries per record is 4.

Entries Per Table/Array (Columns 36-39)

This field contains the maximum number of entries in a table or array.

Columns 36-39	Description
1-9999 (right-justified, leading zeros are not required)	The number of entries reserved by the compiler for the table or array.
blank	This is not a table or array.

Tables and arrays can contain fewer entries during execution than you specify, but not more. For alphanumeric tables and arrays, unused entries contain blanks. For numeric tables and arrays, unused entries contain zeros. You can add or replace entries in tables and arrays during execution (see the section at the end of this chapter titled "Tables and Arrays").

For alternating tables, each pair of related entries is considered one table entry.

Example

The alternating tables (TABLEA and TABLEB) shown in Figure 5-3 have 50 entries each. To indicate this, 0050 is entered in columns 36-39 to indicate the number of entries in the alternating tables.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

E                    TABLEA0030050010L2ATABLEB 12R4ATABLE FILE
```

Figure 5-3. Specifying the Number of Entries in Alternating Tables

Entry Length (Columns 40-42)

This field specifies the length of entries in a table or array.

Columns 40-42	Description
1-256 (right-justified, leading zeroes are not required)	The entry length (space allocated for each item). The maximum length for alphabetic entries is 256 characters; for numeric entries, 15 digits
blank	This is not a table or array.

Entries in each table or array must be the same length. If alphanumeric entries are different lengths, add leading or trailing blanks to make them the same length. If numeric entries are different lengths, add leading or trailing zeros or blanks to make them the same length. If you enter an L or R in the Data Format Field (column 43), include the sign position in the length. For numeric tables and arrays in packed decimal format, enter the unpacked decimal length. For numeric tables or arrays in binary format, enter 5 for 16-bit numbers or 6 for 32-bit numbers (this allocates 2 or 4 bytes, respectively, for the numbers).

If you're using the Calculation Specification LOKUP operation, make sure the entry length is the same as the field being used as the search argument.

If this table or array has an alternating table or array, enter the alternating table or array length in the Entry Length Field (columns 52-54).

Data Format (Column 43)

This field identifies the data format for numeric compile-time or preexecution-time tables and arrays.

Column 43	Description
B	Binary format (1 or 2 word integer).
L	Unpacked decimal format with leading plus or minus sign.
P	Packed decimal format.
R	Unpacked decimal format with trailing plus or minus sign.
blank	Alphanumeric or unpacked decimal (with no leading or trailing signs) format. For unpacked decimal format, enter a digit (0-9) in the Decimal Positions Field (column 44). For alphanumeric format, leave column 44 blank.

If this table or array has an alternating table or array, enter the format for the alternating table or array in the Data Format Field (column 55).

For details about each of the formats, see the Data Format Field (column 43) of the Input Specification.

Decimal Positions (Column 44)

This field contains the number of decimal positions (positions to the right of the decimal) in each entry of a table or array.

Column 44	Description
0-9	This is a numeric table or array with the specified number of decimal positions in each field. (The number of decimal positions cannot exceed the number of digits in the field.)
blank	This is an alphanumeric table or array.

Since numeric entries in tables and arrays do not contain an actual decimal point, this field determines where the decimal point is assumed to be during calculations and output editing operations.

If this table or array has an alternating table or array, enter the decimal positions for the alternating table or array in the Decimal Positions Field (column 56).

Example

To specify that the entries in the following array have two decimal positions, enter 2 in column 44 of the File Extension Specification.

```
ARRAY

0000
0100
0150
0200
0250
  .
  .
  .
```

If the entries in this array were printed using edit code 1 (suppress leading zeros), you would see:

```
.00     1.00     1.50     2.00     2.50...
```

Table/Array Sequence (Column 45)

This field lets you sequence-check compile-time and preexecution-time tables and arrays.

Sequence-checking is not affected by the Alternate Collating Sequence Field (column 26) of the Header Specification.

Column 45	Description
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 and low LOKUP operations prohibited for unsequenced tables or arrays; binary searching prohibited.

Table and array entries can be arranged in ascending, descending, or random order. 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 sequence-check an ascending or descending table or array, RPG checks the order while loading it. If the compiler detects a sequence error, processing halts immediately. The computer operator may then continue processing in spite of the error, or terminate the program.

If this table or array has an alternating table or array, specify sequence-checking for the alternating table or array in the Table/Array Sequence Field (column 57).

Although RPG does not sequence-check execution-time arrays, you must enter an A or D in this field to use LOKUP operations with high or low indicators, or to perform binary searching.

Alternating Table/Array Name (Columns 46-51)

This field contains the name of the alternating table or array associated with the table or array named in the Table/Array Name Field (columns 27-32). Do not enter the name of an execution-time array.

When you enter an alternating table or array into this field, you must also enter the next four fields (through column 57).

Columns 46-51	Description
Alternating table or array name	The name of the second alternating table or array.
blank	The table named in columns 27-32 does not have an alternating table or array.

Entry Length (Columns 52-54)

This field specifies the length of entries in an alternating table or array. The length you enter in this field applies to the table or array named in the Alternating Table/Array Name Field (columns 46-51).

Columns 52-54	Description
1-256 (right-justified, leading zeroes are not required)	The entry length (space allocated for each item). The maximum length for alphabetic entries is 256 characters; for numeric entries, 15 digits
blank	This is not a table or array.

Entries in each table or array must be the same length. If alphanumeric entries are different lengths, add leading or trailing blanks to make them the same length. If numeric entries are different lengths, add leading or trailing zeros or blanks to make them the same length. If you enter an L or R in the Data Format Field (column 55), include the sign position in the length. For numeric tables and arrays in packed decimal format, enter the unpacked decimal length. For numeric tables or arrays in binary format, enter 5 for 16-bit numbers or 6 for 32-bit numbers (this allocates 2 or 4 bytes, respectively, for the numbers).

If you're using the Calculation Specification LOKUP operation, make sure the entry length is the same as the field being used as the search argument.

Data Format (Column 55)

This field identifies the data format for the alternating compile-time or preexecution-time table or array. Use this field only for numeric tables and arrays. This field applies to the table named in the Alternating Table/Array Name Field (columns 46-51).

Column 55	Description
B	Binary format (1 or 2 word integer).
L	Unpacked decimal format with leading plus or minus sign.
P	Packed decimal format.
R	Unpacked decimal format with trailing plus or minus sign.
blank	Alphanumeric or unpacked decimal (with no leading or trailing signs) format. For unpacked decimal format, enter a digit (0-9) in the Decimal Positions Field (column 44). For alphanumeric format, leave column 44 blank.

For details about each of the formats, see the Data Format Field (column 43) of the Input Specification.

Decimal Positions (Column 56)

This field contains the number of decimal positions (positions to the right of the decimal) in each entry of an alternating table or array. This field applies to the table or array named in the Alternate Table/Array Name Field (columns 46-51).

Column 56	Description
0-9	This is a numeric table or array with the specified number of decimal positions in each field. (The number of decimal positions cannot exceed the number of digits in the field.)
blank	This is an alphanumeric table or array.

See the Decimal Positions Field (column 44) for an example of how to enter this field.

Table/Array Sequence (Column 57)

This field lets you sequence-check an alternating compile-time or preexecution-time table or array. Sequence-checking applies to the table or array named in the Alternate Table/Array Name Field (columns 46-51).

Sequence-checking is not affected by the Alternate Collating Sequence Field (column 26) of the Header Specification.

Column 57	Description
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; binary searching prohibited.

Table and array entries can be arranged in ascending, descending, or random order. 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 sequence-check an ascending or descending table or array, RPG checks the order while loading it. If the compiler detects a sequence error, processing halts immediately. The computer operator may then continue processing in spite of the error, or terminate the program.

Although RPG does not sequence-check execution-time arrays, you must enter an A or D in this field to use LOKUP operations with high or low indicators, or to perform binary searching.

Comments (Columns 58-74)

You can enter comments of any kind in this field.

Program Name (Columns 75-80)

This field contains the program name. The format of this field is discussed in Chapter 2.

Tables and Arrays

The table and array sections which follow in this chapter explain how to create and use tables and arrays in an RPG program.

Creating Tables and Arrays

Compile-time and preexecution-time tables and arrays reside in disk files and are loaded before a program begins execution. Execution-time arrays can reside in files on disk or you can create them with Calculation Specification operations at run time.

When creating a table or array, ensure that:

- Each entry in the table or array has the same field length, data type, and the same number of decimal positions.
- Each alphanumeric entry contains 256 characters or less.
- Each numeric entry contains 15 digits or less.
- Entries are not split between records.
- There are no blanks between entries. All entries must be continuous on each record. (You can, however, embed blanks as part of an entry.)
- The entries are in the appropriate sequence - ascending, descending or random.

Creating Compile-Time and Preexecution-Time Tables and Arrays

In addition to the guidelines in the previous section, when you create compile-time and preexecution-time tables and arrays, ensure that:

- The first entry in each record starts in position 1.
- All records (except the last) contain the same number of entries. The last record can contain fewer entries, but it cannot contain more. For example, if the first record contains seven entries, all but the last record must contain seven entries. The last record can contain from one to seven entries.
- There are the same or fewer entries in the table or array than the number you enter in the Entries Per Table/Array Field (columns 36-39) of the File Extension Specification. A full table or array contains the same number of entries as you specify in the File Extension Specification. Tables and arrays containing fewer entries than specified, are called short tables and arrays and the unused entries are automatically set to zeros. The unused entries in short alphanumeric tables and arrays contain blanks. Short tables and arrays let you start with only a few entries, then add to them at a later time. You must include at least one entry.
- The number of entries in each input record matches the number in the Entries Per Record Field (columns 33-35) of the File Extension Specification and the previous rule. You can fill an entire record if necessary. Leave the remaining positions blank, or use them for comments.

- When entering input records for tables and arrays and their alternating tables and arrays, start with the first entry of the table or array (named in columns 27-32 of the File Extension Specification) and follow it by the first entry in the alternating table or array (named in columns 46-51 of the File Extension Specification). End all records, including the last, with an entry from the alternating table or array.
- The table or array file has fixed-length records and is a sequential file.

Defining Tables and Arrays

The next two sections explain how to use the File Extension Specification to define tables and arrays. If you're using compile-time tables or arrays on disk, you must also enter an Array/Table File Name Specification that names the disk file containing the table or array.

Defining Compile-Time and Preexecution-Time Tables and Arrays

Use this checklist when you define compile-time and preexecution-time tables and arrays:

- To sequence check the table or array when it is loaded, enter A or D in the Table/Array Sequence Field (column 45) of the File Extension Specification.

  If you're going to use the LOKUP operation with high and low indicators in a Calculation Specification, you must specify sequence-checking.
- If the table or array entry contains a leading or trailing sign, enter the appropriate character in the Data Format Field (column 43), and allow for the sign when specifying the Entry Length Field (columns 40-42).
- For table and array files having a T in the File Designation Field (column 16) of the File Description Specification, enter a fixed-length record format in the Record Format Field (column 19) of that specification.
- For numeric tables and arrays, specify the data format in the Data Format Field (column 43) of the File Extension Specification and the the number of decimal positions in the Decimal Positions Field (column 44) of that specification.

Example

Figure 5-4 shows how to define two arrays using File Extension Specifications. ARRA is a compile-time array containing 10 entries (2 per record). Each entry is 12 digits long, has two decimal places, and is in unpacked decimal format. ARRB is a preexecution-time array, residing in the disk file DISKER. It contains 300 entries (10 per record). Each entry is 6 positions long, has no decimal places, and is in unpacked decimal format.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

E                    ARRA    2  10 12 2             COMPILE-TIME
     .
     .
     .

E    DISKER          ARRB   10 300  6 0             PREEXEC-TIME
```

Figure 5-4. Defining Compile-Time and Preexecution-Time Arrays

Defining Execution-Time Arrays

To define an execution-time array, leave columns 7-26 and the Entries Per Record Field (columns 33-35) blank in the File Extension Specification.

If it is a numeric array, enter the data format in the Data Format Field (column 43) and the number of decimal positions in the Decimal Positions Field (column 44).

Example

Figure 5-5 shows how the execution-time array ARRC is defined. It has 10 alphanumeric entries, each 2 positions long.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

E                    ARRC       10  2 0A             EXEC-TIME
```

Figure 5-5. Defining Execution-Time Arrays

Loading Tables and Arrays

The next three sections describe how RPG loads table and arrays. (Tables and arrays must be loaded before you can access them in the program.)

Loading Compile-Time Tables and Arrays

The RPG compiler makes compile-time tables and arrays an integral part of the object program and they are loaded along with it at run time.

You can place compile-time tables and arrays at the end of the source program, keep them in files on disk, or any combination of the two. Additionally, the same disk file can contain more than one table or array. If this is the case, you must enter the File Extension Specifications in the same order as the tables and arrays appear in the job stream or disk file.

ALTSEQ records and file translation records are also processed at compile-time. ALTSEQ records, file translation records and compile-time tables and arrays are loaded in the following order:

1. ALTSEQ (alternate collating sequence) records.
2. Compile-time tables and arrays, in the same order as the File Extension Specifications.
3. File translation records.

Loading Compile-Time Tables and Arrays Appended to the Source Program

When tables, arrays, ALTSEQ, and file translation records are appended to the source program or when they are contained in a job file, enter them as follows:

1. After the last record in the source program, enter a separator record containing **␣ (where ␣ represents a blank) in columns 1-3.
2. ALTSEQ records, if any.

   End the ALTSEQ records by a separator line containing **␣ in columns 1-3. (Only use the separator line when there are ALTSEQ records.)
3. Enter compile-time tables and arrays according to the format discussed in the "Creating Tables and Arrays" section of this chapter. Be sure the format conforms to the entries in the File Extension Specifications. End each table or array with a separator record (**␣ in columns 1-3).
4. If you're using file translation records, enter a separator line (**␣ in columns 1-3). Enter the file translation records, separating each new file name record with a separator record (**␣ in columns 1-3).

Example

The following source program file contains two alternate collating sequence records, four array records and a file translation record.

```
.
.                          Source program
.
**␣
ALTSEQ 51415942
ALTSEQ 5843                Alternate collating sequence records

**␣
ALPHA
BETA
GAMMA                      Compile-time array
DELTA
**␣
OUTPUT 58445745            File translation record
```

Loading Compile-Time Tables and Arrays from Disk Files

When you want to place ALTSEQ records, compile-time tables and arrays, and file translation records into a disk file, enter them as follows:

1. Enter the ALTSEQ records, tables, arrays, and file translation records into a disk file according to the instructions in the preceding section "Loading Compile-Time Tables and Arrays Appended to the Source Program". Use separator records as needed but do not start the file with a separator record.
2. Enter Array/Table File Name Specifications (as described in Table 5-1) after the last specification in the program. For instance, if you have Output Specifications, the Array/Table File Name Specifications follow the last one. Do not place a separator record between the last specification and the first Array/Table File Name Specification.

Table 5-1. Array/Table File Name Specification Format

Columns	Values	Description
1-5	00000-99999	Sequence number, as defined in Chapter 2.
6	A	This is an Array/Table File Name Specification.
7-14	Valid file name	The name of a file containing at least one alternate collating sequence, table, array, or file translation record.
15-80	blank	None.

The following lines show how to enter three Array/Table File Name Specifications:

```
.
.                          Source program
.

ADISKFILE
ATABFILE1                  Array/Table File Name Specifications
ATABFILE2
```

When you compile the program, the ALTSEQ records, file translation records and the tables and arrays are made part of the program.

Example

The following lines show how to place two ALTSEQ records, four compile-time array records and one file translation record in a disk file.

```
ALTSEQ 51415942
ALTSEQ 5843                 Alternate collating sequence records

**␣
ALPHA
BETA
GAMMA                       Compile-time array
DELTA
**␣
OUTPUT 58445745             File translation record
```

Loading Compile-Time Tables and Arrays from the Source Program and Disk

You can load ALTSEQ records, tables, arrays, and file translation records from the source program and from disk in the same program.

Make sure that the order of the data records, whether loaded from the source program or from disk is: ALTSEQ records, table and array records, then file translation records.

The compiler loads data records in the order in which you enter the Array/Table File Name Specifications. When entering these specifications, be careful not to enter a separator record before them. If you do, they will be treated as data.

Example

The following lines show how to load ALTSEQ records from a disk file TABFILE, and how to load four compile-time array records and two file translation records from records appended to the source program.

```
.
.                           Source program
.
                            File containing ALTSEQ records
ATABFILE                    (can also contain compile-time
                            tables or arrays).
**␣
ABLE
BAKER                       Another compile-time array
CHARLIE
DOG
**␣
OUTPUT 58445745             File translation record
**␣
INPUT  59426043             File translation record
```

Loading Preexecution-Time Tables and Arrays

Preexecution-time tables and arrays can reside on the same device or on different devices. They are loaded before the program is executed and in the order in which they are listed in the File Extension Specifications. When you create table and array files, observe these rules:

- If you're sequence-checking a table or array, ensure that the last record in the table or array is not blank. The last entry for a table or array in ascending sequence has the highest value. For descending sequence, the last entry has the lowest value.
- Ensure that the tables and arrays are in the same order on disk as shown in the File Extension Specifications.
- Follow each table or array in an input file with a separator record (containing asterisks in columns 1-2 and a blank in column 3). This record delimits the tables or arrays; it is not treated as data.
- When input data follows a table or array in a file, place a separator record (two asterisks and a blank) between the table or array and the data.

Loading Execution-Time Arrays

There are two ways to load execution-time arrays.

You can create the array entries and place them in a file on disk, then define the array entries using an Input Specification. Or you can create the array entries yourself at run time using Calculation Specification operations.

If you load the array using Input Specifications, entries may occupy consecutive positions in the input records or they can be scattered throughout the records with intervening spaces. When an array is contained on one input record and has consecutive entries, enter just one Input Specification. When the entries are scattered, enter an Input Specification for each entry.

When entering Input Specifications for execution-time arrays, follow these rules (see Chapter 7 for details about the Input Specification fields):

- Enter I in the File Type Field (Column 6).
- Leave columns 7-42 blank.
- Enter the appropriate value in the Data Format Field (column 43).
- For arrays containing consecutive entries, enter the field location of the entire array in columns 44-51. For arrays containing scattered entries, enter the field location of each entry in columns 44-51. Enter the beginning location in the From Field (columns 44-47) and the ending location in the To Field (columns 48-51).

Example Conventions

- If you indicated in the Decimal Position Field of the File Extension Specification that the entries have one or more decimal positions, enter the same value in the Decimal Position Field of the Input Specification (column 52). Otherwise, leave the Decimal Position Field blank.
- Enter the name of the array in the Field Name Field (columns 53-58). This must be the same name entered in the Table/Array Name Field (columns 27-32) of the File Extension Specification. To define an individual entry in the array, enter an index (see the section which follows titled "Searching Arrays" for information on indexing.)
- Leave columns 59-62 blank.
- Enter the field record relation indicator, if you're using one, in the Field Record Relation Field (columns 63-64).

When an array requires two or more input records, you can define the array entries collectively or individually in the Input Specification. Use variable indexes and (or) record-identifying indicators to avoid overlaying the entries in one record with those in another.

Since the HP RPG logic cycle processes only one record at a time, several cycles are required to load a multirecord array. Because of this, you may need to suppress calculations and output until the entire array has been read. To do this, enter indicators in the Indicators Field of Calculation Specifications and the Output Indicators Field (columns 23-31) of Output Specifications.

Example

Figure 5-6 shows the File Extension and Input Specifications for three different kinds of arrays. ARR1 contains eight entries (each 10 positions long) that are loaded from a single record in the disk file FILEA. ARR2 contains five entries, all in a single input record. The entries are five positions long and have intervening blanks. (Note that the entries are listed individually on the Input Specification.) ARR3 contains 30 entries each 10 positions long. The first four input records contain seven entries each (in columns 3-72). The fifth input record contains two entries (in columns 3-22).

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

E                    ARR1        8 10
E                    ARR2        5  5
E                    ARR3       30 10
     .
     .
     .

IFILEA   AA  01
I                                       1  80 ARR1
IFILEA   AA  01
I                                       1   5 ARR2,1
I                                       7  11 ARR2,2
I                                      13  17 ARR2,3
I                                      19  23 ARR2,4
I                                      25  29 ARR2,5
IFILEA   AA  01
I                                       1   20X
I                                       3  72 ARR3,X
```

Figure 5-6. Loading Execution-Time Arrays

Searching Tables

The Calculation Specification LOKUP operation lets you search tables for specific values.

When you use LOKUP, enter the same name for the table that you used in a File Extension Specification. The table is searched sequentially (unless you request a binary search and the entries are in sequence) until an entry is found that matches the search argument. The entry can then be used in calculations. Subsequently, when you reference the table name as an operand in a Calculation Specification operation, the entry found by the most recent LOKUP operation is selected.

You can use the SORTA Calculation Specification operation to ensure that a table or array is sequenced properly.

Searching Arrays

You can search arrays using the LOKUP Calculation Specification operation (see the previous section, "Searching Tables"). In addition, you can access individual entries in an array by using an index.

When referencing an entire array, use the array name entered in the File Extension Specification.

Example Conventions

When you want to access a specific entry in an array using a Calculation Specification, enter the array name followed by a comma and an index. The index can be a number or the name of a numeric field containing the number of the entry you want to access. The index value for the first entry is 1 (not zero). An index field must not have decimal positions and can be no longer than 9 digits.

For example, to reference the fourth entry in the array XARY, enter:

```
XARY,4
```

Alternatively, if N is an index field containing the number 4, you can reference the fourth entry by entering:

```
XARY,N
```

Table 5-2 shows which Calculation Specification operations allow indexing and which do not.

Table 5-2. Indexing with Calculation Specification Operations

You must use indexing with these operations:	You may use indexing with these operations:	You cannot use indexing with these operations:
BITOF	ADD	XFOOT
BITON	DEBUG	
COMP	DIV	
DSPLY	LOKUP	
TESTB	MHHZO	
TESTZ	MHLZO	
	MLHZO	
	MLLZO	
	MOVE	
	MOVEA	
	MOVEL	
	MULT	
	SQRT	
	SUB	
	Z-ADD	
	Z-SUB	

When you enter an array name without an index, the following rules apply:

- When you enter arrays in one of the Factor Fields and the Result Field, they are searched sequentially. If the arrays do not contain the same number of entries, the search ends when the last entry in the shortest array is processed.
- When one Factor Field is a field or constant and the other Factor Field and the Result Field are both arrays, the search continues until every entry in the shorter array has been processed. The same field or constant is used in all of the operations.
- Since multiple operations are performed, resulting indicators can only be used with the XFOOT and LOKUP operations.

Changing Table and Array Entries During Execution

You can temporarily change table or array values by entering the table or array name in the Result Field of an arithmetic or move Calculation Specification operation. Additionally, you can add or modify table or array values using Input or Calculation Specifications. Changes that you make remain in effect for the duration of the program; when the program ends, they are lost.

To permanently change table or array values, you can save the updated table or array into a file on disk (see the section which follows, "Writing Tables and Arrays") or you can manually modify the original table or array file.

You can add entries to short tables and arrays using Calculation Specification operations or you can read them from new input records at run time.

Note

If a run-time array indexing error is detected (index < one or > array size) and a "continue" response is selected, HP RPG/iX will reset the index to one before continuing.

Writing Tables and Arrays

You can write a table or array to a disk file, terminal, or to the line printer.

To write an entire table or array (except an execution-time array), including temporary modifications, enter the name of the table or array output file in the To File Name Field (columns 19-26) of the File Extension Specification. To write an execution-time array, enter Output Specifications for it.

To write individual entries in a table or array, enter Output Specifications only for those entries. Describe them just as you would normal output fields. Using the table name in the Output Specification causes the last entry found (not the entire table) to be written.

When you specify editing for an output array, the editing applies to all entries in the array. If entries have different editing requirements, reference them individually. When you enter an edit code for the entire array in an Output Specification, RPG separates edited array entries by two blanks to help you distinguish one from another. However, if the table or array is directed to a line printer, array entries are separated by two blanks regardless of the edit code.

The File Extension Specification Required Entries

Certain entries in the File Extension Specification are required, optional, or cannot be used, depending upon the type of file you are describing. The rules governing entries for each type of file are summarized in Table 5-3.

Table 5-3. File Extension Specification - Required/Optional/Prohibited Entries

Field (columns)	Chaining File	Record Address File	Compile Time Table/ Array File	Preexecution Time Table/ Array File	Execution Time Array File
Chaining File Record Sequence (7-8)	Optional	Prohibited	Prohibited	Prohibited	Prohibited
Chaining Code Identifier (9-10)	Required	Prohibited	Prohibited	Prohibited	Prohibited
From File Name (11-18)	Required	Required	Prohibited	Required	Prohibited
To File Name (19-26)	Required	Required	Optional	Optional	Prohibited
Table/Array Name (27-32)	Optional	Optional	Required	Required	Required
Entries Per Record (33-35)	Prohibited	Prohibited	Required	Required	Prohibited
Entries Per Table/Array (36-39)	Prohibited	Prohibited	Required	Required	Required
Entry Length (40-42)	Prohibited	Prohibited	Required	Required	Required
Data Format (43)	Prohibited	Prohibited	Optional	Optional	Optional
Decimal Positions (44)	Prohibited	Prohibited	Required for numeric entries; prohibited for others	Required for numeric entries; prohibited for others	Required for numeric entries; prohibited for others
Table/Array Sequence (45)	Prohibited	Prohibited	Optional	Optional	Optional
Alternating Table/Array Name (46-51)	Prohibited	Prohibited	Required for alternating table, array	Required for alternating table, array	Required for alternating table, array
Entry Length (52-54)	Prohibited	Prohibited	Required for alternating table, array	Required for alternating table, array	Required for alternating table, array
Data Format (55)	Prohibited	Prohibited	Optional	Optional	Optional
Decimal Positions (56)	Prohibited	Prohibited	Optional	Optional	Optional
Table/Array Sequence (57)	Prohibited	Prohibited	Optional	Optional	Optional

The File Extension Specification Default Summary

If you leave the optional fields of the File Extension Specification blank, the defaults shown in Table 5-4 apply:

Table 5-4. File Extension Specification Defaults

Columns	Field	Default Values
1-5	Sequence Number	No sequence number applies.
7-8	Chaining File Record Sequence	None.
9-10	Chaining Code Identifier	This file is not a chaining file.
11-18	From File Name	This is a compile-time table or array if an entry appears in columns 33-35; it is an execution-time array if no entry appears in columns 33-35.
19-26	To File Name	Table and arrays, if any, are not written.
27-32	Table/Array Name	None.
33-35	Entries Per Record	This is an execution-time array.
36-39	Entries Per Table/Array	This is not a table or array.
40-42	Entry Length	This is not a table or array.
43	Data Format	The table or array entries have alphanumeric or unsigned external decimal format.
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 is sequential. SORTA operation assumes ascending sequence.
46-51	Alternating Table/Array Name	The table or array named in columns 27-52 does not have an alternating table or array.
52-54	Entry Length	No entry is specified.
55	Data Format	The table or array entries have alphanumeric or unsigned external decimal format.
56	Decimal Positions	This is an alphanumeric table or array.
57	Table/Array Sequence	Do not sequence check the table or array.
57	Table/Array Sequence	Do not sequence check the table or array.
58-74	Comments	None.
75-80	Program Name	None.

6

Line Counter Specifications

This specification lets you change the defaults used during skipping operations (see the Skip Field (columns 19-22) in the Output Specification). If you do not use a Line Counter Specification, the defaults are those shown in Table 6-1. Use the Line Counter Specification for print files and for disk files that you want to keep in printable format. When you enter a Line Counter Specification, use either the Channel Number Fields, or the Line Number Fields:

- Channel Number Fields

  These fields let you assign line numbers to logical printer carriage control channels (only Channel 1 on an actual printer carriage control tape is used in RPG). Then, in the Output Specifications for the file, enter the channels in the Skip Field (columns 19-22) to advance the printer paper to the associated line number. When you use the Channel Number Fields, leave the Carriage Control Type Field (column 53) blank in the Header Specification.

- Line Number Fields

  These fields let you change the line where page overflow occurs and also lets you change the number of lines per page. Any entries that you make in the Skip Field (columns 19-22) of Output Specifications are interpreted as line numbers and RPG advances the paper to those line positions. When you use the Line Number Fields, enter an L or 1 in the Carriage Control Type Field of the Header Specification.

The Line Counter Specification is identified by an L in column 6:

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

L
```

Figure 6-1. The Line Counter Specification

The Line Counter Specification Fields

The fields you use in the Line Counter Specification are described in the sections which follow in this chapter. Each field has a unique name and occupies specific positions (columns) in the specification.

Use the Channel Number Fields (columns 15-74) when you want to simulate the use of channels in the printer carriage control tape. Use the Line Number Fields (columns 15-24) when the lines to which you need to skip vary.

Sequence Number (Columns 1-5)

This field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

This field contains an L to identify this line as a Line Counter Specification.

File Name (Columns 7-14)

This field names the output file to which this specification applies. This field is required.

Columns 7-14	Description
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 the printable file to which this Line Counter Specification applies.

Channel Number Fields (Columns 15-74)

The next three fields let you assign up to twelve logical printer carriage control channels to specific line numbers.

Line Number (Columns 15-17)

This field contains the line number that corresponds to the channel or overflow line entered in the next field.

Columns 15-17	Description
1-112 (right justified, leading zeros are not required)	The line number to which the channel number in the next field, or the overflow line, refers.
blank	If the channel in the next field has already been defined, this line number is ignored (the previous line number is used). If the channel in the next field has not been defined previously, the line number equal the channel number multiplied by 5. (For instance, Channel 3 is line 15.)

Channel Number/OL (Columns 18-19)

This field identifies the channel number to which the line number in the preceding field applies. Or, it identifies the line number (in the previous field) as the overflow line.

Columns 18-19	Description
1-12 (right-justified, leading zeros are not required)	The channel number to which the line number in the preceding field is assigned.
OL	The line number in the preceding field is the overflow line. If the previous field is blank, line 60 (Channel 12) is the overflow line.
	When you space, skip, or print beyond the overflow line (but not beyond the current page) the overflow indicator, if used, is turned on and detail and total lines (including those conditioned by the overflow indicator) are printed.
blank	If the previous field contains a line number, it is ignored.

Line Number and Channel Number/OL (Columns 20-74)

Columns 20-74 let you assign additional channels to line numbers. The two previous fields, Line Number (columns 15-17) and Channel Number/OL (columns 18-19) are repeated to allow you to enter a total of twelve channel number assignments. Enter data into columns 20-74 according to the rules presented for those fields.

The first pair of blank Line Number and Channel Number/OL Fields ends the assignments. You can enter overflow and channel number assignments in any order and you can assign two or more channels to the same line number.

Example

Figure 6-2 assigns Channel 1 to line 12, Channel 3 to line 15, and Channel 4 to line 20 for the file PRINTX. It also designates line 63 as the overflow line for the file.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

LPRINTX  01201015030200406 30L
```

Figure 6-2. Using the Channel Number Fields

Line Number Fields (Columns 15-24)

The next three fields let you specify the number of lines per printed page and the line that signals page overflow.

Line Number (Columns 15-17)

This field identifies the last line number on the page or the line at which page overflow occurs. This field is used in conjunction with the next field, OL/FL (columns 18-19).

Columns 15-17	Description
1-112 (right-justified, leading zeros are not required)	The line number of the overflow line or the page length, as determined by the next field.
blank	None.

OL/FL (Columns 18-19)

This field identifies whether the line number in the previous field is the overflow line or the last line on a page.

Columns 18-19	Description
OL	The line number in the preceding field is the overflow line. If the previous field is blank, line 60 is used. When you space, skip, or print beyond the overflow line (but not beyond the current page), the overflow indicator, if used, is turned on and the detail and total lines (including those conditioned by the overflow indicator) are printed.
FL	The line number in the previous field is the form (page) length. If the previous field is blank, line 66 is used.
blank	Line 60 is the overflow line; there are 66 lines per page.

When you enter the overflow line and the page length but do not use Fetch Overflow, allow enough space between the overflow and page length lines for detail and total lines and skip accordingly.

Line Number and OL/FL (Columns 20-24)

Columns 20-24 let you specify either the overflow line or the page length (the one you have not already assigned in columns 15-19). Columns 20-24 consist of two fields that correspond to Line Number (columns 15-17) and OL/FL (columns 18-19), respectively. Enter data into these columns according to the directions for those fields. You can assign the page length and overflow lines in any order in columns 15-24.

Example

Figure 6-3 defines line 68 as the overflow line for the file PRINTY. It also specifies that line 74 is the last line on a page.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

LPRINTY  0680L074FL
```

Figure 6-3. Using the Line Number Fields

Program Name (Columns 75-80)

This field contains the program name. The format of this field is discussed in Chapter 2.

The Line Counter Specification Default Summary

If you leave the optional fields of the Line Counter Specification blank, the defaults shown in Table 6-1 apply.

Table 6-1. Line Counter Specification Defaults

Columns	Fields	Default Values
1-5	Sequence Number	No sequence number applies.
15-24	Line Number, Channel Number/OL	Channel Number fields: line 6 is assigned to Channel 1; other line numbers equal the channel numbers multiplied by 5; line 60 (Channel 12) is the overflow line.
	Line Number, OL/FL	Line Number fields: line 60 is the overflow line; line 66 is the form length.
25-74	Line Number, Channel Number/OL	Channel Number fields: line numbers equal the channel numbers multiplied by 5; line 60 is the overflow line.
75-80	Program Name	None.

7

Input Specifications

This specification gives additional information about input, update, and combined files. If you're using any of those files, you must enter one or more Input Specifications for them.

The Input Specification:

- Identifies the types of records in the file and shows how they relate to one another. (Use the File and Record Description Fields in columns 7-42 to enter this information.)
- Describes the format and location of the fields in the input records and provides directions for testing and using their contents. (Use the Field Description Fields in columns 43-70 to enter this information.)

The Input Specification is identified by an I in column 6:

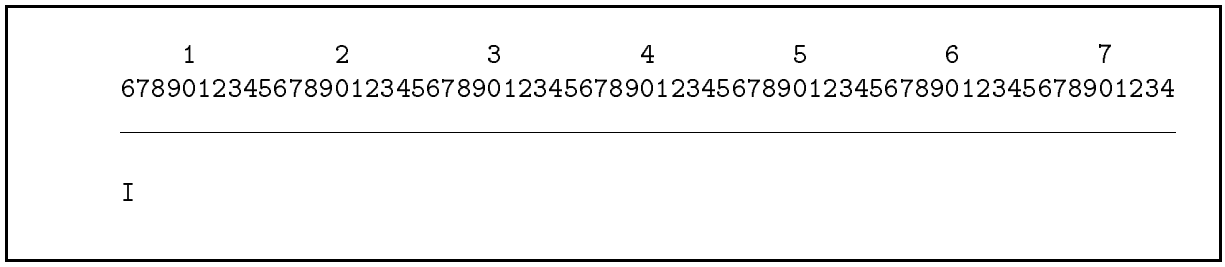


Figure 7-1. The Input Specification

The Input Specification Fields

The fields you can use in the Input Specification are described in the sections which follow in this chapter. Each field has a unique name and occupies specific positions (columns) in the specification.

Sequence Number (columns 1-5)

This field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

This field contains an I to identify this line as an Input Specification.

File and Record Description Fields (columns 7-41)

These fields describe the record types contained in input, combined and update files.

Group all Input Specifications for a file together. Define the first record type by entering the File and Record Description Fields, leaving columns 43-70 on that specification line blank. Follow this line by one or more specifications that describe the fields for the record type (see the Field Description Fields (columns 43-70)). Repeat this specification sequence until all record types are defined.

Example

Figure 7-2 shows two record types for the file ORDER. Lines 1 and 5 contain the File and Record Description Fields for each record type. Lines 2 through 4 and 6 through 7 contain the Field Description Fields for the record types.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

1     IORDER   NS  01  44 CA
2     I                                      1  10 CUSNUM
3     I                                     11  40 CUSNAM
4     I                                     41  42 DSCNT
5     I        NS  02  44 CB
6     I                                      1   8 PROD
7     I                                      9  12OQTY
```

Figure 7-2. Entering Input Specifications for a File

File Name (columns 7-14)

This field names the file to which this and subsequent Input Specifications apply. Enter the name of an input, combined, or update file defined by a File Description Specification. Do not enter the name of an output or display file.

Columns 7-14	Description
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 an input, update, or combined file being described by this Input Specification.
Data structure name (columns 7-12).	Name of the data structure being described by the Input Specification.

If you enter more than one Input Specification for this file, you do not have to repeat the file name on each line. The name remains in effect until a new one is encountered.

AND/OR (columns 14-16)

This field lets you assign a record identification code longer than three characters to a record type (AND). It also lets you assign more than one record identification code to the same record type (OR). This field is used in conjunction with the Record Identification Codes Field (columns 21-41).

Columns 14-16	Description
AND	Identifies this Input Specification as an AND line.
OR	Identifies this Input Specification as an OR line.

You can intermix AND and OR lines.

To enter AND lines, follow these steps:

1. Make sure that the first three characters of the record identification code are described by the previous specification.
2. Enter AND in columns 14-16 with up to three additional record identification code character descriptions in columns 21-41. Leave columns 17-20 blank. AND lines indicate that *all* record-identifying characters must exist before the associated record-identifying indicator is turned ON.
3. Continue entering AND lines until all of the record identification codes are defined. A record must contain all characters defined for the code before its associated record-identifying indicator is turned ON.

Example Conventions

To enter OR lines, follow these steps:

1. Make sure that from one to three characters of the record identification code are described by the previous specification.
2. Enter OR in columns 14-15 with up to three additional record identification code character descriptions in columns 21-41. Leave columns 16-18 blank. Record identification codes in an OR line and any AND lines which follow it have an AND relationship (all of them must exist before the associated record-identifying indicator is turned ON).
3. Continue entering OR lines until you define all of the record identification codes.
4. You can enter a record-identifying indicator in columns 19-20 of each OR line. If you do not use one, the last indicator that you entered applies.

Example

Figure 7-3 shows how three record types are identified by record identification codes. Lines 1 and 2 specify that records for the first record type have the letter A in positions 1-3 and the letter B in position 4. Records for the second record type (lines 3 and 4) have the letter A in positions 1-3 or a Z in position 80. Records for the third record type (lines 5-8) have the letter A in positions 1-5 or the letter Z in positions 76-80.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

1    IMYFILE  AA  01   1 CA   2 CA   3 CA
2    I       AND       4 CB
     I                 .
     I                 .
3    I        BB  02   1 CA   2 CA   3 CA
4    I       OR       80 CZ
     I                 .
     I                 .
5    I        CC  03   1 CA   2 CA   3 CA
6    I       AND       4 CA   5 CA
7    I       OR       76 CZ  77 CZ  78 CZ
8    I       AND      79 CZ  80 CZ
```

Figure 7-3. Using AND and OR Lines to Identify Record Types

Group Sequence (Columns 15-16)

This field lets you specify the order of record types, if any, in a file. Records types (see the Record Identification Codes Field (columns 21-41)) identify different data record formats in a file. This field is required.

Columns 15-16	Description
01-99 (right-justified, leading zeros are not required)	Assign this sequence number to the record type specified in the Record Identification Codes Field (columns 21-41), and verify this sequence on input.
Two alphabetic characters	No group sequence applies. Do not sequence-check input record types.

Use this field when a file contains more than one record type, the record types are in a specific order, and you want that order verified when the file is processed. For example, when a file contains a customer identification record followed by one or more invoice records for each customer, you may want this order verified. If the record types are not in the order you specify, an error message is printed and the program terminates. Group sequences only ensure that within a group, all data records are in order by the record type entered in the Record Identification Codes Field (columns 21-41). Other fields on the record are not sequence-checked. To sequence-check other data fields, use the Sequence Field (column 18) of the File Description Specification.

You must use 01 for the first record type. For the remaining record types, enter any set of numbers that are in ascending sequence. For example, 01, 02, 05, and 52 is a valid sequence.

Within a file, you can use group sequences for some records and not for others. Enter the Input Specifications for the records that do not use group sequences first. (The actual data records for these specifications can appear anywhere in the file and can be interspersed with group sequence records.) You cannot enter group sequence numbers on AND and OR lines. For more information see the AND/OR Field (columns 14-16). For OR lines, the group sequence from the previous line is used.

Example

The file INP contains three different types of records that relate to a company's sales staff. Each type is identified by a letter in position 10 of the record: A gives the sales person's name, B gives the sales person's territory and C contains the sales person's sales quota.

```
NAME      A
TERRITORY B               Group 1
QUOTA     C

NAME      A
TERRITORY B               Group 2
QUOTA     C

NAME      A
TERRITORY B               Group 3
QUOTA     C
```

Example Conventions

To specify that record type A is first for each sales person followed by record types B and C, enter the numbers 01, 02, and 03 in the Group Sequence Field on consecutive Input Specifications. On the group 01 sequence line, enter an A in column 27. On the group 02 sequence line, enter B in column 27 and on the group 03 sequence line, enter C in column 27.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

IINP     011     10 CA
I                                          1  20 NAME
I        021     10 CB
I                                          1   5 TERRIT
I        031     10 CC
I                                          1   92QUOTA
```

Figure 7-4. Defining Group Sequences

Number of Records (Column 17)

This field specifies the number of records allowed for the record type in the group sequence. Use this field only if you entered a number in the previous field, Group Sequence (columns 15-16).

Column 17	Description
N	One or more records of this type are allowed in the group.
1	Only one record of this type is allowed in the group.
blank	There are no restrictions on the number of records per record type in the group, or you entered alphabetic characters in the Group Sequence Field.

Do not enter this field on specifications containing AND or OR in columns 14-16. (If this is an OR line, the number of records for the record type is taken from the previous Input Specification).

Option/LDA (Column 18)

This field specifies whether the record type is required or optional in a group sequence (see the Group Sequence Field (column 15-16)). This field also identifies a data structure as a Local Data Area.

Column 18	Description
O	Records of this type are optional and may or may not be present in each group. (Prevents sequence errors when a record is absent.)
U	The data structure defined in this specification is the Local Data Area.
blank	At least one record with this record type must be present in each group, or alphabetic characters are entered in the Group Sequence Field.

Do not enter this field on specifications containing AND or OR in columns 14-16. (If this is an OR line, the value for this field comes from the previous Input Specification).

U (Local Data Area)

The Local Data Area is a file named LDAFILE which contains 1 record. The record can contain up to 32 segments of 256 bytes each (the number of segments depends on how it is created). The Local Data Area provides a way to pass data between RPG programs.

The Local Data Area is created and initialized (to blanks) by the RPGINIT utility. RPGINIT is often run by a logon UDC. See the *RPG Utilities Reference Manual* for information about RPGINIT.

The Local Data Area is loaded into the User Data Structure at run time. To create a User Data Structure, define it in the File and Record Description line of the Input Specification with a U in the Option Field (column 18). Also enter a DS in the Record Indicator/Look-Ahead/Trailer/Data Structure Field (columns 19-20) of that specification. You can follow the File and Record Description line with Field Description lines describing the LDAFILE record, though this is optional. Define the record using an array that is the same length as the LDAFILE record, with an element length of one byte. You can use and modify the array using Calculation Specifications. When the program ends, the User Data Structure contents overwrite the Local Data Area.

Example Conventions

The following steps show how Program A uses a Local Data Area to pass information to Program B. (Assume that the Local Data Area has already been created using the RPGINIT utility.)

1. Program A loads the Local Data Area File into its User Data Structure.
2. Program A modifies the User Data Structure during calculations.
3. At end-of-job, Program A updates the contents of the Local Data Area.
4. Program B loads the Local Data Area File into its User Data Structure.
5. Program B uses the data passed in in the Local Data Area from Program A.

Example

To use the Local Data Area in a program, enter a U in column 18, and a DS in columns 19-20. Figure 7-5 shows how to define a Local Data Area in the Input Specifications.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
I            UDS
I                                        1  20 USER
I                                       21  21 OPT1
I                                       22  22 OPT2
I                                       23  23 OPT3
I                                       24  29OBGNDAT
I                                       30  35OENDDAT
```

Figure 7-5. The Input Specifications for a Local Data Area

Record Indicator/Look-Ahead/Trailer/Data Structure (Columns 19-20)

This field assigns a record-identifying indicator to the record type. It also lets you specify whether the record type contains look-ahead fields, whether the next lines define the trailer portion of a spread record, or whether they define a data structure.

Columns 19-20	Description
Record-Identifying Indicators:	
01-99	Assign this general indicator.
F0-F9	Assign this function key indicator.
H0-H9	Assign this halt indicator.
KA-KN, KP-KY	Assign this command key indicator.
L1-L9	Assign this control-level indicator.
LR	Assign the last-record indicator.
MR	Assign the matching-record indicator.
OA-OG, OV	Assign this overflow indicator.
U1-U8	Assign this user indicator.
1P	Assign the first-page indicator.
Look-Ahead:	
**	This record contains one or more look-ahead fields.
Trailer:	
TR	The next specification lines define the trailer portion of a spread record.
Data Structure:	
DS	The next specification lines define a data structure.
blank	Do not assign an indicator to the record type, or this record does not have look-ahead fields, or the next lines do not define the trailer portion o spread record or do not define a data structure.

Record-Identifying Indicators

When a file contains several types of records and you want to perform different operations on each type, use this field to assign a different record-identifying indicator to each type. At the beginning of each logic cycle, all assigned record-identifying indicators are turned OFF. Each time a record is read and selected for processing, the indicator assigned to it is turned ON and remains on for the entire logic cycle. You can use a record-identifying indicator to condition a Calculation Specification operation by entering it in one of the indicator fields (columns 9-17) of the specification. You can use a record-identifying indicator to condition an output operation by entering it into one of the output indicator fields (columns 23-31) of the Output Specification. You can also use a Calculation Specification to change the setting of a record-identifying indicator.

Use record-identifying indicators only when:

- You are processing more than one record type in a file.
- You are updating a file (to ensure that updates are made only after the appropriate records are read).

When using record-identifying indicators, be sure you understand how they are processed in the RPG logic cycle, especially how they are used in processing heading, detail, and total records. See the *HP RPG Programmer's Guide* for information on the RPG logic cycle.

When you're using chained files, several record-identifying indicators can be on simultaneously, since several records can be processed at the same time.

When you're doing multiple reads from one or more demand files during the same logic cycle, the record-identifying indicators assigned to the file(s) remain on throughout the cycle if the previous READ operations were executed successfully. To make sure that these indicators are off, you must explicitly turn them off using a Calculation Specification.

You can assign record-identifying indicators in any order on the Input Specifications. You can use OR lines (see the AND/OR Field (columns 14-16)) to assign the same indicator to several record types. Normally, when using AND lines (see the AND/OR Field), you enter only one record-identifying indicator (since the AND lines specify all of the conditions to be met by the record type).

01-99 (General Indicators)

General indicators identify record types in a file. They are the most frequently-used indicators. When a record associated with a general indicator is read and selected for processing, the indicator is turned ON and all operations conditioned by that indicator are performed.

Example

Line 1 in Figure 7-6 shows how to assign general indicator 03 to the first record type in file READX. Every time a record is processed for this record type, the ADD Calculation Specification operation (line 2) is executed.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ____________________________________________________________

(1)       IREADX   011003  05 CX
          I                  .
          I                  .
          I         0210     05 CY
          I                  .
          I                  .
          I         0310     05 CZ
          I                  .
          I                  .

(2)       C   03       STOREA    ADD  STOREB    RESF
```

Figure 7-6. Using a General Indicator

F0-F9 (Function Key Indicators)

You can use function key indicators the same way you use general indicators.

Function key indicators have special meanings when used with the RPG VPLUS Interface. They are used by VPLUS to signal "events" that take place at the user terminal. When the user presses (ENTER), F0 is turned ON. When the user presses (f1) to (f8), the corresponding function key indicator is turned ON. When an event 9 or greater takes place, function key indicator F9 is turned ON. See Chapter 10 for a complete discussion of the RPG VPLUS Interface.

Function key indicators F1-F8 also have special meanings when used in conjunction with the SET, DSPLY, and DSPLM Calculation Specification operations (see these operations in Chapter 8 for details).

H0-H9 (Halt Indicators)

Assign these indicators to record types that you want to cause a halt. When a halt indicator is turned ON, a message is displayed on the terminal, and the program stops at the end of the current cycle's detail-time processing. You can continue the program, if you wish. When the program is continued, the halt indicator is turned OFF.

KA-KN, KP-KY (Command Key Indicators)

You can use command key indicators the same way you use general indicators.

When you use an RPG Screen Interface (RSI) file, the command keys may have a special meaning. A user at a terminal keyboard presses f1 followed by a key from the top row of the keyboard to turn on one of the twenty-four command key indicators. The RPG Screen Interface then performs the appropriate action. You enable the command key indicators when you build the screen forms file. You can use command keys that have not been enabled the same way you use general indicators. See Chapter 11 for information on the RPG Screen Interface and the *RPG Utilities Reference Manual* (SIGEDITOR) for information on creating an RSI forms file.

Note

In an RSI application program, all command key indicators are set off prior to the READ of the workstation file.

L1-L9 (Control-Level Indicators)

You can use control-level indicators the same way you use general indicators. They are turned OFF before the next record is read, regardless of how you use them.

You can assign control-level indicators to record types in any order. For example, you can assign the indicators L5, L1, and L7 in that order. You can also associate control-level indicators with individual fields on a record (see the Control Level Field (columns 59-60)). Doing this identifies where the fields fit in the control-break hierarchy.

LR (Last-Record Indicator)

This indicator identifies the record that signals end-of-file. When this record is encountered, calculations and output conditioned by this indicator are performed, and the program ends.

MR (Matching-Record Indicator)

This indicator is turned ON before detail-time calculations when a match occurs on matching fields (see Matching/Chaining Fields, columns 61-62). The indicator is turned OFF if a match does not occur. The indicator remains set through total and output calculations in the next cycle.

OA-OG, OV (Overflow Indicators)

These indicators are normally used in Output Specifications to signal page overflow. However, if you use them on an Input Specification, they function the same as general indicators. When a record associated with an overflow indicator is read, the indicator is turned ON and all operations conditioned by it are performed.

U1-U8 (User Indicators)

You can use these indicators, much the same way you use general indicators, to condition operations.

1P (First-Page Indicator)

Normally, you use this indicator in Output Specifications to print headings on the first page of a report. After the headings are printed, however, you can use it the same way you use a general indicator (the 1P indicator is turned OFF after each detail-time output).

** (Look-Ahead)

Place ** in columns 19-20 when the record contains one or more look-ahead fields. Look-ahead fields let you examine the contents of a record before the record is available for processing. For input files, look-ahead fields are located on the next record. For a update and combined files, look-ahead fields are located on the current record. Do not use look-ahead fields with chained or demand files or with files containing spread records. Do not use this field on AND or OR lines. For more information, see the AND/OR Field (columns 14-16).

You can enter as many look-ahead fields for a record as necessary. They apply to all records in a file, regardless of the record type. Define look-ahead fields after a record type that has alphabetic characters in the Group Sequence Field (columns 15-16). Look-ahead field names must be unique. Do not define them elsewhere in the program.

You cannot alter the contents of look-ahead fields. Do not use them in result fields or clear them. If you want to use the contents of a look-ahead field before and after the record is selected for processing, define it as a look-ahead field and also as a regular field (with a different name).

When end-of-file is reached in a file containing look-ahead fields, the look-ahead fields are automatically filled with ASCII 9's (for alphanumeric fields) and packed +9's (for numeric fields).

Example

Figure 7-7 shows how to define the look-ahead field LOOKHD for the file OVERLD. Line 1 contains ** in columns 19-20. Line 2 defines the field name and its location in the record.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

     IOVERLD  AA  01
     I                                          1  10 MINE
     I                                         11  20 YOURS
(1)  I        AB  **
(2)  I                                          1  10 LOOKHD
```

Figure 7-7. Defining a Look-Ahead Field

TR (Trailer Records)

When a file consists of a header record followed by one or more records that contain information related to that header, you can condense the file by formatting the records in a spread format. When you use a spread record, place the header and related information on the same record. You can use spread records with primary and secondary files. You cannot use spread records with update or combined files or files that have look-ahead fields.

When a file contains several record types, any or all of them can have spread records. On the Input Specifications, define a spread record as one logical record. It consists of a header portion followed by a trailer portion. Follow these steps when defining the spread record:

1. Define the header portion by entering all fields that you're going to use in the program. Define each field as you normally do, using the File and Record Description Fields and Field Description Fields. If you're using record identification codes, control levels or matching fields, define them as part of the header. If you entered a number in the Group Sequence Field (columns 15-16), you must also enter N in the Number of Records Field (column 17).
2. Start the trailer portion by entering TR in the Record Indicator Field (columns 19-20) in the line following the last header field. Leave the rest of the specification blank. (Enter only one TR line per spread record. You can specify as many spread records as necessary for a file.)
3. Define the trailer fields used in the program. If the trailer fields repeat on the data record, define them only once. Define each field as you normally do using columns 44-51 and 53-58 in each specification line (leave columns 7-42, 59-62 and 71-74 blank). Do not overlap the header portion with the trailer portion and do not specify control levels or matching fields for the trailer portion.

   When formatting the data records, you can enter as many sets of trailer fields as the record length allows. If the logical record does not have enough room for all of the trailer fields, continue the data on another line and begin that line with the header. Make sure that the header and trailer fields have the same format as specified in the Input Specifications.

Example

A hardware and carpentry supply house uses a file that contains a header record for each general class of item stocked (such as saws, hammers, and screwdrivers). Each header record is followed by records that show the quantity of each specific type of item on hand, one record for each type. For instance, one item record might show 63 cross-cut saws, another might indicate 15 rip saws. The three records below show how to format the header and item information into spread records. Notice that the fields, TYPE, PART, and QTY are repeated three times in each record.

```
            TOOL      TYPE      PART QTY  TYPE      PART QTY   TYPE     PART QTY

            |         |         |    |    |         |    |     |        |    |
            v         v         v    v    v         v    v     v        v    v
Record 1:   SAW       CIRCULAR  00031   65COPING    00032   14CROSSCUT  00033  107
Record 2:   SAW       HACK      00036   27KEYHOLE   00035   29RIP       00036   75
Record 3:   SCREW DRIVPHILLIPS  00051   35REGULAR   00052  209RIGHT-ANG 00053   82
```

In the above records, the header and trailer fields are as follows:

```
Header Field:       TOOL

Trailer Fields:     TYPE      PART QTY   TYPE     PART QTY TYPE    PART   QTY
```

Notice that the trailer fields for SAW span two logical records, while those for SCREW DRIV are contained in one logical record. Figure 7-8 shows how to define the Input Specifications for the spread records shown above. Lines 1 and 2 define the header field, TOOL. Lines 3-6 define the trailer fields, TYPE, PART, and QTY.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------

(1)       IINVEN   01N AA
(2)       I                                       1  10 TOOL
(3)       I            TR
(4)       I                                      11  20 TYPE
(5)       I                                      21  25 PART
(6)       I                                      26  30 QTY
```

Figure 7-8. Defining a Spread Record

DS (Data Structure)

Data structures let you break a field or an array into subfields that can be referenced in Calculations and Output Specifications. Data structures can also be used to group individual fields together so that they can be used as a unit.

Whenever a data structure is modified its subfields are automatically modified. Conversely, if a subfield is changed, its data structure is also changed.

Data structure specifications follow all other Input Specifications. When defining a data structure, enter the File and Record Description Fields (columns 7-41) as follows:

1. Enter an I in column 6.
2. If you wish, enter a data structure name in columns 7-12 (if you do not, an internal name is assigned by the compiler). The name must conform to the naming conventions used for fields and arrays and cannot exceed 6 characters. The name cannot be an array element or a table. Enter one of the following for the data structure name:
   - A name not used elsewhere in the program. The data structure is created as an array with a one-byte element length. The number of elements is equal to the data structure length (the highest end positions of its subfields).
   - The name of an alphanumeric array defined in a File Extension Specification.
   - The name of a field in an Input Specification.
   - The name of a previous alphanumeric data structure subfield (alphanumeric data structure subfield definitions can be nested, but numeric fields cannot).
   - If this is the User Data Structure for the Local Data Area, enter the name LDA (other names are ignored).
3. If this is the data structure for the Local Data Area, enter a U in column 18. The subfields of the data structure are initialized from the Local Data Area at the beginning of the program and used to update the LDA at end-of-job.
4. Enter DS in columns 19-20 to indicate that this is a data structure.
5. Leave all of the other columns blank.

Define data structure subfields, using the Field Description Fields (columns 43-70), as follows:

1. Enter an I in column 6.
2. Leave columns 7-43 blank.
3. Enter the starting position of the field in columns 44-47. The starting position is relative to the first position of the data structure. The starting position can overlap other subfields in the data structure.
4. Enter the ending position of the field in columns 48-51. The ending position is relative to the first position of the data structure. The ending position can overlap other subfields in the data structure.

   If the subfield is part of a data structure whose length is already defined, the subfield must be entirely contained within the data structure (the ending position cannot extend beyond the length of the data structure).
5. Enter the decimal positions (0-9) for a numeric field in column 52, or leave column 52 blank to indicate that the subfield is alphanumeric.
6. Enter the subfield name in columns 53-58. A subfield name:
   - Can be a new name (not already used in the program).
   - Can be a name which has been previously defined as an input field of a data record. If so, the field length and number of decimals must be the same as the original definition. Use columns 44-51 (From and To Fields) to define the location of the field within the data structure (not within the original data record).
   - Can be an alphanumeric array name defined in a File Extension Specification. Columns 44-51 must specify the entire length of the array. An array element cannot be specified as a data structure subfield.
   - Can be redefined as a data structure, with its own subfields (the data structure and subfield definitions can be nested).
   - Cannot be an array element or a table.
   - Cannot have the same name as a subfield in another data structure.

Note

The HP RPG implementation of data structures does not redefine the same internal area multiple times as other implementations of data structures do. This allows nesting and greater latitude in referencing data structures within calculations.

Example

Figure 7-9 shows how to use a data structure to define subfields of the input field PRODID. The subfields are CATALOG, VENDOR, and PRD#. Changes to any of the subfields will automatically be reflected in the data structure. If, for example, a calculation changes the subfield VENDOR, PRODID also changes.

```
    1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

IDATA     NS  01
I                                              1  10 PRODID
I                                             11  40 DESC
I                                             41  482PRICE
IPRODID       DS
I                                              1   2 CATLOG
I                                              3   5 VENDOR
I                                              6  10 PRD#
```

Figure 7-9. Using a Data Structure to Subdivide an Input Field

Figure 7-10 shows how to use a data structure to consolidate two separate fields, PROD and DSCNT. These fields contain a product number (from an order file) and a discount code (from a customer master file). These fields are placed together by the data structure PKEY so that they can be used as the key field for reading another file.

```
    1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

ICUSTMR   NS  01
I                                              1  10 CUSNUM
I                                             11  40 CUSNAM
I                                             41  42 DSCNT
IORDER    NS  02
I                                              1   8 PROD
I                                              9  120QTY
IPRICE    NS  03
I                                              1  10 KEY
I                                             11  172COST
IPKEY         DS
I                                              1   8 PROD
I                                              9  10 DSCNT
```

Figure 7-10. Using a Data Structure to Consolidate Two Separate Fields

Record Identification Codes (Columns 21-41)

When a file contains more than one record type and the program processes each type differently, you must assign a unique code to each type using this field. The code that you assign must exist in one or more data records in the file. (When a file contains just one type of record or when each type is processed the same way, leave this field blank.)

Column	Value	Description
Position: (columns 21-24, 28-31, 35-38)	1-9999 (right-justified, leading zeros are not required)	The position (relative to the first position of the record) where this identification code character appears in the input record.
	blank	A record identification code does not apply.
Not: (columns 25, 32, 39)	N	The character, zone, or digit must not be present in the location specified by the Position subfield.
	blank	A character, zone, or digit must be present.
Portion: (columns 26, 33, 40)	C	Use the entire character (zone and digit portions) for the record identification code.
	D	Use only the digit portion of the character for the record identification code.
	Z	Use only the zone portion of the character for the record identification code.
	blank	No character applies.
Character: (columns 27, 34, 41)	Any letter, digit, or special character	Test for this character (on input) in the location specified by the Position subfield.
	blank	No character applies in this position.

You can use one or more Input Specifications to assign record identification codes. To assign from one to three characters, use one Input Specification. Enter the first character in column 27 and describe it in columns 21-26. Enter the second character in column 34 and describe it in columns 28-33. Enter the third character in column 41 and describe it in columns 35-40. Enter AND (columns 14-16) lines when the record identification codes are longer than three characters. Use OR (columns 14-16) lines to assign more than one record identification code to the same record type.

When a record identification code is recognized at run time, the record-identifying indicators associated with the record type (see the Record Indicator/Look-Ahead/Trailer/Data Structure Field, columns 19-20) are turned ON. If a record meets the requirements of more than one record type, it is processed according to the first record type listed in the program.

Portion (Columns 26, 33, 40)

This field lets you specify which portion of the record identification code character to use in identifying the record type. You can use the entire character, the zone portion, or the digit portion.

Normally, RPG compares record identification codes in ASCII. If you want to compare using the EBCDIC character set, enter an E in the EBCDIC Zone/Digit Tests Field (column 39) of the Header Specification. (Appendix D lists the ASCII and EBCDIC collating sequences.)

C (Character)

The character in the data record is compared to the entire character (each of the eight bit positions) that you enter in column 27, 34, or 41. The corresponding record-identifying indicator is turned ON only when all bit positions match.

D (Digit)

The low-order four bits (digit portion) of the character in the data record are compared to the low-order four bits of the character that you enter in column 27, 34, or 41. If these bits match, regardless of whether the zone portion matches, the corresponding record-identifying indicator is turned ON. For example, if you enter A in column 27, the data characters Q and 1 will result in a match and cause the associated record-identifying indicator to turn ON.

Z (Zone)

The high-order four bits (zone portion) of the character in the data record is compared to the high-order four bits of the character that you enter in column 27, 34, or 41. If each zone bit matches, the corresponding record-identifying indicator is turned ON. For example, if you enter A in column 27, the letters B through O will result in a match and cause the associated record-identifying indicator to turn ON.

Example

Three examples of record identification codes appear in the Input Specification in Figure 7-11. The first record type (line 1) has the letter A in position 1. The second record type (line 2) includes all records with an F in position 3 and any character except D in position 2. The third record type (line 3) contains a character in position 4 whose zone portion matches the letter T.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ____________________________________________________________

(1)       IANYFILE AA  01   1 CA
          I        .
(2)       I        BB  02   2NCD   3 CF
          I        .
(2)       I        CC  03   4 ZT
```

Figure 7-11. Record Identification Codes

Field Description Fields (Columns 43-70)

These fields describe the data fields within the records defined by the File and Record Description Fields (columns 7-41). Data fields include arrays and array elements.

Enter fields on separate lines, starting with the line immediately following the line that contains the File and Record Description Fields for the record. Leave columns 7-42 blank. You only need to define those fields actually used in the program. See Figure 7-2 for an example of how to enter Field Description Fields.

Data Format (Column 43)

This field describes the format of numeric fields as they exist in the input file. Do not enter a value into this field for alphanumeric or unpacked (external) decimal fields.

Column 43	Description
B	Binary format (1 or 2 word integer).
L	Unpacked decimal format with leading plus or minus sign.
P	Packed decimal format.
R	Unpacked decimal format with trailing plus or minus sign.
1-9	The number of digits required by a binary field for internal computation.
blank	Alphanumeric or unpacked decimal (with no leading or trailing signs) format. For unpacked decimal format, enter a digit (0-9) in the Decimal Positions Field (column 52). For alphanumeric format, leave column 52 blank.

If a numeric field is not defined as a packed decimal (P) field, it is converted to that format once it is read from the input file. Internal operations use packed decimal format because it is more efficient. Additionally, when an alphanumeric field is moved to a numeric field or used in computations, it is converted to packed decimal format with the low-order position determining the sign:

If the low-order position is:	The sign is:
`0-9`	Unsigned - hexadecimal F (1111)
`{ or A-I`	Positive - hexadecimal C (1100)
`} or J-R`	Negative - hexadecimal D (1101)

For example, the number -256 looks like this when represented in alphanumeric (ASCII) format:

```
bits:   0 0 1 1   0 0 1 0    0 0 1 1   0 1 0 1    0 1 0 0   1 1 1 1
        --------|--------    --------|--------    --------|--------
        <----  byte  ---->   <----  byte  ---->   <----  byte  ---->

                   2                   5                   0
```

Since the ASCII representation of the low-order character is O, the field is assumed to be negative and looks like this when converted to packed decimal:

```
bits:   0 0 1 0   0 1 0 1    0 1 1 0   1 1 0 1
        --------|--------    --------|--------
        <----  byte  ---->   <----  byte  ---->

            2        5           6        D
```

Similarly, if the number 256 has a positive sign in low-order digit (25F), it is assumed to be positive when converted to packed decimal and looks like this:

```
bits:   0 0 1 0   0 1 0 1    0 1 1 0   1 1 0 0
        --------|--------    --------|--------
        <----  byte  ---->   <----  byte  ---->

            2        5           6        C
```

B (Binary Format)

Numbers are represented in two's complement form. A field is either two or four bytes long.

The two-byte format, also known as fixed-point format, lets you store positive and negative integers in the range -32,768 to +32,767. Bit 0 of the high-order byte is the sign bit. It is 0 for positive numbers and 1 for negative numbers. The remaining bits (1-15) are used for the integer. The following example shows how the number 256 is stored.

```
bits:   0 0 0 0   0 0 0 1    0 0 0 0   0 0 0 0
        --------|--------    --------|--------
        <----  byte  ---->   <----  byte  ---->

                                256
```

The four-byte format, also a fixed-point format, is the same as the two-byte format except that the four bytes allow 32 bits for the integer. Bit 0 of the high-order byte is the sign bit. You can store positive and negative integers in the range -2 billion to +2 billion.

L (Unpacked Decimal Format With Leading Sign)

This format requires one byte (8 bits) to store each digit of the integer. It is the same as unpacked decimal format (see the description for the "blank" entry) except that the first byte is reserved for the sign. For negative numbers, it is the ASCII negative sign (see the beginning of this section). For positive numbers, it can be any character. For example, a field which is located in positions 11-15 of a record has the sign in position 11 and the integer value in positions 12-15.

P (Packed Decimal Format)

Each byte of the field is divided into two parts - each holding one digit. This format saves memory because only four bits are required for each digit. The low-order four bits of the low-order byte contains the sign. If the number is negative, the sign bits are 1101 (the character D in hexadecimal). If the number is positive, the sign bits are 1100 (the character C in hexadecimal), though any bit configuration except 1101 is considered positive. For example, the number 256 is stored as follows:

```
bits:   0 0 1 0   0 1 0 1    0 1 1 0   1 1 0 1
        --------|--------    --------|--------
        <----  byte  ---->   <----  byte  ---->

            2         5          6         D
```

R (Unpacked Decimal Format With Trailing Sign)

This format requires one byte (8 bits) to store each digit of the integer. It is the same as unpacked decimal format (see the description for the "blank" entry) except that the last byte is reserved for the sign. For negative numbers, it is the ASCII negative sign (see the beginning of this section). For positive numbers, it can be any character. For example, a field which is located in positions 11-15 of a record has the sign in position 15 and the integer value in positions 11-14.

1-9 (Internal Binary Digits)

When a binary field is converted to packed decimal format for internal use, RPG assumes that it contains either 5 or 10 digits. If you want to reduce or expand the number of digits during internal calculations, enter the appropriate number in this column. For example, if the field is used in calculations that can result in a larger number, enter the number of digits for that number in this field.

Blank (Unpacked Decimal and Alphanumeric Formats)

The field contains unpacked decimal or alphanumeric data. Unpacked decimal fields can contain any number and alphanumeric fields can contain any alphabetic, numeric or special character. Both types of data have ASCII representation and each character in the fields occupy an eight-bit byte.

In unpacked decimal format, the unsigned number 256 appears as follows:

```
bits:   0 0 1 1   0 0 1 0   0 0 1 1   0 1 0 1   0 0 1 1   0 1 1 0
        --------|--------   --------|--------   --------|--------
        <----  byte  ---->  <----  byte  ---->  <----  byte  ---->

                2                   5                   6
```

Example Conventions

From Field Position (Columns 44-47)

This field defines where the data field starts in the input record. This field is required.

Columns 44-47	Description
1-9999 (right-justified, leading zeros are not required)	The beginning position of the field in the input record.

To Field Position (Columns 48-51)

This field defines the last position of the data field. This field is required.

Columns 48-51	Description
1-9999 (right-justified, leading zeros are not required)	The ending position of the field in the input record. The maximum length for alphanumeric fields is 256 characters; for numeric fields, 15 digits.

Enter a position relative to the first position of the record. It must be greater than or equal to the number in the From Field (columns 44-47) and it must fall within the record length defined in the File Description Specification. (A one position field contains the same number in both this field and the From Field.)

If you want to process part of an input array only, enter only those positions in the From Field and this field that you want to use.

Decimal Positions (Column 52)

This field specifies the number of decimal positions that a numeric field contains. Do not enter a value into this field for alphanumeric fields. This field is required for numeric fields.

Column 52	Description
0-9	The number of decimal positions in the field (not to exceed the number of digits in the field).
blank	The field contains alphanumeric data.

This field indicates where the decimal point belongs in the field, although no actual decimal point appears there. If this is an array, you can leave this field blank but you must enter the number of decimals for it in the Decimal Positions Field (column 44) of the File Extension Specification.

Field Name (Columns 53-58)

This field names the field, array or array element defined by the specification. Every field must have a name; you use this name in the program to reference the field.

Columns 53-58	Description
Field name. (This can 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 can 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 the File Extension Specification.)
Array name, comma, and index. The array name is the same as described above. The index is either a number or the name of a field (see the field name entry above) that contains a number. The array name/comma/index combination is limited to six characters.	The name of the input array item. The name must be defined as an array in the File Extension Specification.
PAGE, or PAGE1 through PAGE7	The name of a field that provides the page number for output files.
*ERROR	The name of one-character field used for run-time error codes.

Field Name

Enter a name for the field.

Normally, you assign unique names to fields that have the same record type. If you use duplicate names, only the last field can be accessed using the name.

If you use the same field name in more than one record type, be sure the fields have the same field length, data format and number of decimal positions. The fields can have different beginning locations. If the fields are in the same locations in the records, you can save time by using OR lines to define them (see the following example).

Example

Figure 7-12 shows how to assign the names FIELD1, FIELD2 and FIELD3 to three fields in the file FILEX (the names start in line 1).

The example also shows how to use the same field names for fields in two different record types in the file FILEX. The field names are FIELDA, FIELDB, and FIELDC and their definitions begin with line 3. The OR line is used to avoid defining the fields twice. The fields have the same length on both record types and are located in the same positions.

```
     1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
------------------------------------------------------------------------

     IFILEX   AA  01   1 CA
(1)  I                                      11  20 FIELD1
     I                                      21  30 FIELD2
     I                                      31  40 FIELD3
     I         BB  02   1 CB
(2)  I        OR   03   1 CC
(3)  I                                      11  20 FIELDA
     I                                      21  30 FIELDB
     I                                      31  40 FIELDC
     I                                       .
     I                                       .
```

Figure 7-12. Entering Field Names

Array Name

When you want to read an entire array, enter the array name in this field. To read a portion of the array, enter the array name with an index (see the next paragraph).

Array Name, Comma, Index

To read one element in an array, enter the array name and an index. The index specifies the element to read. To read more than one element, but not the entire array, enter an index that specifies the starting location of the first element, then in the From Field Position Field (columns 44-47) and the To Field Position Field (columns 48-51) enter the total number of characters to read.

The index can be a number or a field that contains a number. If you enter a field for the index, it must be a numeric field containing no more than 9 digits. The field must not have decimal positions. If the index is an input field, you must have defined it on a previous Input Specification (its value should be read before you use it as an index; otherwise, the value from the previous record is used). If you're calculating the value for the index field in a Calculation Specification operation, make sure you perform the operation before using the field as an index.

PAGE, PAGE1-PAGE7

Using the field PAGE in an Input Specification lets you change the length and initial value of the field PAGE. PAGE is a predefined four-digit field which is set to zeros. It is used for printing page numbers in report files.

Normally, you use the field PAGE only in Output Specifications for printing the page number (see the Field Name Field (columns 32-37). Page numbering starts with one and continues to 9999. Defining PAGE in an Input Specification lets you override these defaults. You can specify any length for the field, but it must be a numeric field without decimal positions. Pagination starts with the number you enter in PAGE, plus one. For example, if you set PAGE to 9, the first page number is 10.

You can define PAGE as a field in a regular data file, as part of a data structure, or in a file by itself. If it is part of a data record, its value changes each time a record is read from the file and pagination restarts with the new value. Place the value for PAGE in the input record right-justified. Leading zeros are not required.

If you have more than one report file, and you want to override the pagination defaults for them (as described for PAGE), use any of the names PAGE1 through PAGE7.

Example

Figure 7-13 shows how to define PAGE so that its value is read from the input file PAGENO. PAGE is five characters long in the record.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------

          IPAGENO  CC  04   1 CP
(1)       I                                    2   60PAGE
```

Figure 7-13. Entering the PAGE Field Name

*ERROR

*ERROR is a predefined, one-character alphanumeric field. When a run-time error occurs that does not cause the program to terminate, RPG places a unique character in *ERROR that identifies the error. If you enter this field in an Input Specification, you can interrogate it to determine the cause of the error. (See Appendix B for the values that are placed in *ERROR for various run-time errors.)

If, for some reason, you want to overwrite *ERROR, define it with a length of one using the From Field Position Field (columns 44-47) and the To Field Position Field (columns 48-51).

Control Level (Columns 59-60)

This field assigns a control-level indicator to the field. You can assign control-level indicators to fields in primary and secondary files. Do not assign them to look-ahead or trailer fields or to fields in chained or demand files.

Column 59-60	Description
L1-L9	Assign this control-level indicator to the field.
blank	The field is not a control field.

A control-level indicator identifies a *control field* in the file. When the value in the control field changes on input, a new *control group* begins and a *control break* occurs. When a control break occurs, the control-level indicator is turned ON along with all indicators of lower rank (see the next paragraph) and all calculation and output operations associated with the indicator(s) are performed. Use control-level indicators to condition:

- Operations for the first record of a new control group.
- Operations to be performed after all records in a control group are read.
- Total and subtotal output operations.

You can assign control-level indicators in any sequence. For example, you can assign L5, L1 and L7 in that order and omit the others. However, within an indicator range such as L1-L9, the indicator's number indicates its rank. For example, L9 is ranked higher than L8, L8 higher than L7, and so forth. When a control break associated with L9 occurs, all lower-ranked indicators (L8 to L1) are also turned ON. When you assign control-level indicators, associate them with the field that corresponds to their rank in the control field group. For example, in a city/county/state control field group, where state is the highest control field, L3 can be assigned to the state field, L2 to the county field and L1 to city field.

The control-level indicator L0 is always turned ON. You cannot assign it to a control field, but you can use it to condition Calculation Specification operations. The LR indicator is turned ON when the last record is read. When it is turned ON, L1-L9 are also turned ON.

A control break often occurs for the first record in a file because the control field is compared to a field of blanks or zeros. When this happens, total-time calculations and output operations are suppressed.

Arithmetic signs and decimal positions are ignored when numeric control fields are compared. For example, if one control field contains +5 and another contains -5, the fields are considered equal. Similarly, 2.11 is equal to 211.

You can combine several fields to form one control field. You do this by defining them in the Input Specifications and assigning the same control-level indicator to them. (The fields are called *split control fields*.) If you're using split control fields (that have the same control-level indicator) in more than one file, the total length of the split control fields in each file must be the same. If the split control fields are in the same record type, they may be separated by other fields, but they cannot be intermixed with other control-level fields. The maximum length for a split control field is 256 characters.

You can assign the same control-level indicator to fields in different files. Control breaks occur the same as if the fields were in the same file. For example, if L3 is assigned to a field in one file and L3 is assigned to a field in another file, a change in either of these fields turns ON L3 and causes a control break.

Example

Figure 7-14 shows how to enter the Input Specifications for a program that prints a voter count by county, city and precinct. The file VOTERS contains voter registration data for a particular state. The fields on each input record specify the voter's name, street address, city, county, and precinct. After each record in VOTERS is read, the program increments the voter count 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 (line 3), associated with the L1 indicator, changes. The operations conditioned by the L1 indicator are then performed. The program prints the total number of voters in the precinct, adds this value to a value counter for total voters in the city, and clears the precinct counter.

When the contents of the control field CITY (line 1) changes, a higher level control break occurs because L2 and L1 are turned ON. The program now performs the 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 (line 2) change, L3, L2 and L1 are turned ON. Totals for the county, in addition to those for the city and precinct, are calculated and printed.

Finally, when end-of-file is encountered for VOTERS, the program turns ON the last-record (LR) indicator. It then performs all operations conditioned by this indicator (and indicators L1-L9), such as calculating and printing the total number of voters in the state as well as those for precinct, city, and county. The program then ends.

```
     1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
-----------------------------------------------------------------------

     IVOTERS  AA  01   1 CA
     I                                       5  25 NAME
     I                                      26  30 ADDR
(1)  I                                      31  35 CITY  L2
(2)  I                                      36  40 COUNTYL3
(3)  I                                      41  45 PRECNTL1
```

Figure 7-14. Using Control-Level Indicators in a Voter Count Program

Matching/Chaining Fields (Columns 61-62)

This field contains either a matching or a chaining code for the field. If you're not using matching or chaining fields, leave this field blank.

Columns 61-62	Description
C1-C9	This is a chaining field identified by this code.
M1-M9	This is a matching field identified by this code.
blank	The field is neither a matching nor a chaining field.

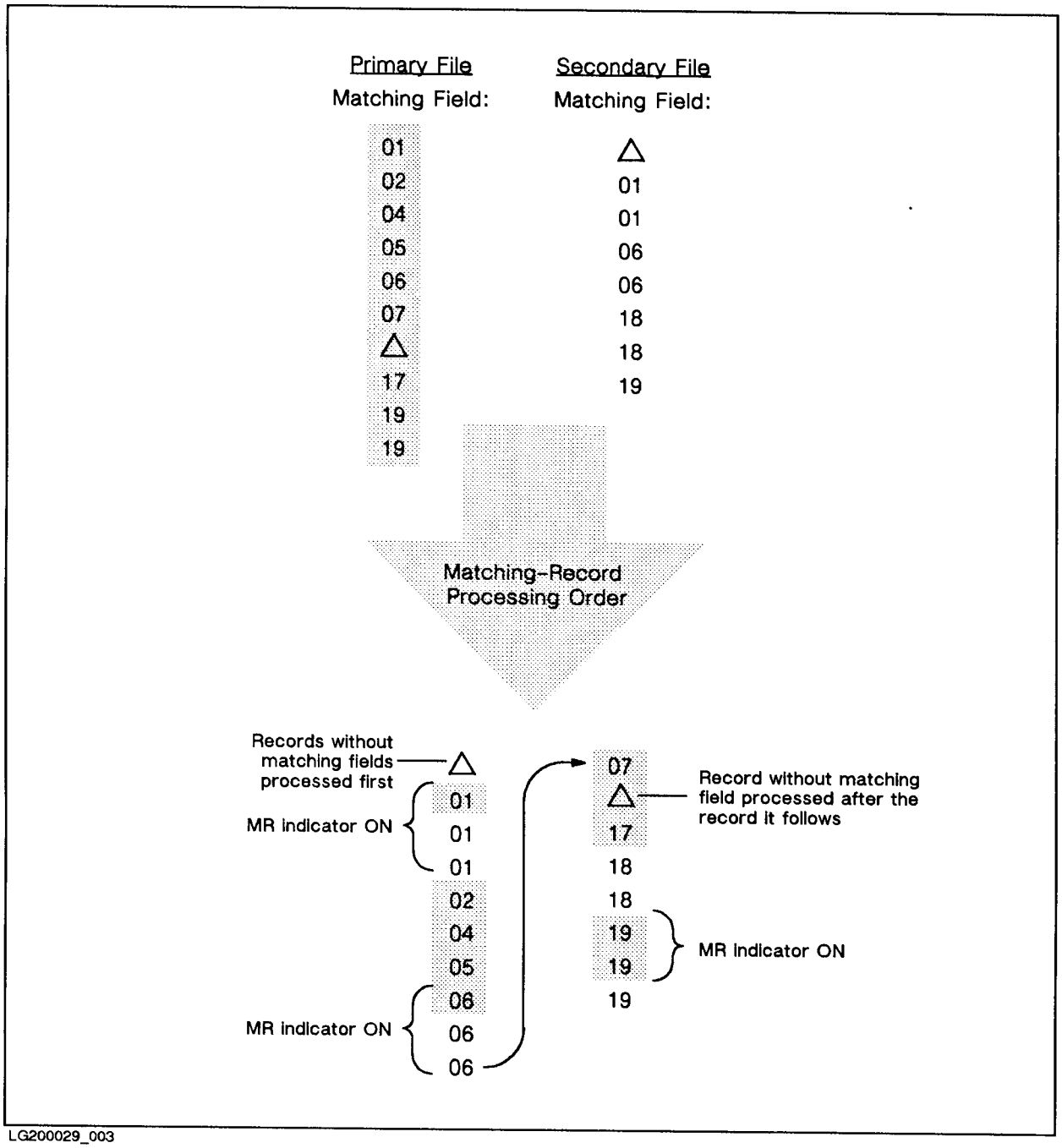
C1-C9 (Chaining fields)

Chaining field codes let you identify the fields to be used in retrieving records from a direct-access chained MPE file. (A chained file can also be a TurboIMAGE or KSAM file.) A chaining field code marks a field as a chaining field. A chaining field contains addresses of records that you want to access in a chained file. Chaining takes place before detail-time processing.

There are two ways to access a chained file:

- You can use the CHAIN Calculation Specification operation. This method may be more flexible than the next method because you can use indicators to condition the operation.
- You can assign one or more chaining field codes as described in this section (this is called "input chaining"). This method reads the chained file automatically each time a record is selected for processing that has a chaining field. You must enter a File Extension Specification to associate the chaining file to the chained file.

Figure 7-15 shows how chaining takes place. A program reads a record from a transaction (chaining) file on disk that contains a customer number and the customer charge amount. This file does not contain the mailing address of the customer. The program obtains this address by using the chaining field (customer name) to access the customer's record in the chained file. At detail time in the HP RPG logic cycle, the mailing address is retrieved and the program can print the customer bill with mailing label. Figure 7-16 shows how to enter the specifications for this example using chaining field codes.



LG200029_003

Figure 7-15. Chaining to a Direct-Access File

You can have up to nine chaining fields per record type. (This lets you access more than one chained file for each record.) The chaining field codes do not have a rank based on their number. You determine the order of the chaining fields by the sequence in which you enter the Input Specifications. For example, if you assign C3 to the first field in a record and C2 to the next field and C5 to the third, the chained files are read in this order: C3, C2, C5.

Chaining fields can contain:

- Numeric relative record addresses of records to be accessed in the chained file.
- Alphanumeric or numeric keys for records to be accessed in the chained file. (The chained file must be a KSAM or TurboIMAGE file processed by key.)

You can assign the same chaining field code to more than one field, creating a *split chaining field.* Split chaining fields are combined to form one chaining field. They can be adjoining or non-adjoining fields. The first chaining field for the specification defines the leftmost position of the combined chaining field. The last field defines the rightmost position of the combined chaining field.

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). If you do, chaining field codes specified for the record type are ignored.
3. Do not intermix lines containing split chaining fields with those containing regular chaining fields.
4. If the chaining field contains relative disk addresses, the field must be defined as numeric.
5. If you want to use the same chaining field to read more than one chained file, define a separate chaining field for each file but use the same starting and ending positions for each of them. Also use different names and chaining codes.

Example

Figure 7-16 shows the specifications that perform the chaining illustrated in Figure 7-15. The customer names and charge amounts are stored in the chaining file TRANSAC. The customer mailing addresses and other information are contained in the KSAM file MASTER. The first field in TRANSAC, CUST, contains the customer name and it is identified as the chaining field by the code C3. When MASTER is read, the record whose CUST field matches the name in CUST (TRANSAC) is selected.

```
        1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

(1)     FTRANSAC IP  F      80            CARD
(2)     FMASTER  IC  F     256RI5AI     1 DISK

(3)     EAAC3TRANSAC MASTER

        ITRANSAC AA  01  80 CT
(4)     I                                    01  15 CUST    C3
        I                                    16  200RECNO
        I                                    21  25 AMOUNT
        IMASTER  BB  02  80 CM
        I                                    01  15 CUST
        I                                    16  25 STREET
        I                                    25  35 CITY
        I                                    35  40 STATE
        I                                    41  45 ZIP
```

Figure 7-16. Specifying Chaining Fields Using the Input Specification

M1-M9 (Matching fields)

You can use matching fields with sequential primary and secondary files having any file organization (when you do, these files are called *matching files*). Matching field codes perform two functions in a program, depending on the number of files to which you assign them. These functions are summarized below and are explained in detail in the paragraphs which follow:

- When used with one file, they identify the fields to be sequence-checked if the Input Sequence Field (column 18) of the File Description Specification requests sequence-checking.
- When used with two or more files, they identify the fields to be matched on input.

If you use matching field codes for only one file in a program, they identify the fields to be sequence-checked. (You can sequence-check input, combined and update files only.) You can use one or all of the matching field codes. If you use more than one, the fields are treated as a single control field. The fields associated with the higher matching field codes come first, followed by the lower-ranking ones. For example, the field associated with M9 comes first followed by the field associated with M8 and so on. Be sure that the Input Sequence Field (column 18) of the File Description Specification specifies the sequence you want to use. If a sequence error occurs, an error message is printed and the error response specified by the Error Response Field (columns 56-71) of the Header Specification, if any, is performed.

Example

Figure 7-17 shows how to assign matching field codes to three fields (DEPT, DIV and BRANCH) in the file INFILE. M3 is assigned to the major control field, BRANCH.

If INFILE contained the following three records and if ascending sequence is specified for the file in column 18 of the File Description Specification, the DIV code in the third record will cause a sequence error.

```
DEPT:   DIV:   BRANCH:

003     005    01
002     006    01
001     004    01
```

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

IINFILE
I                                       1  10 IDNO
I                                      21  30 DEPT    M1
I                                      31  40 DIV     M2
I                                      41  45 BRANCH  M3
```

Figure 7-17. Using Matching Field Codes to Sequence-Check a File

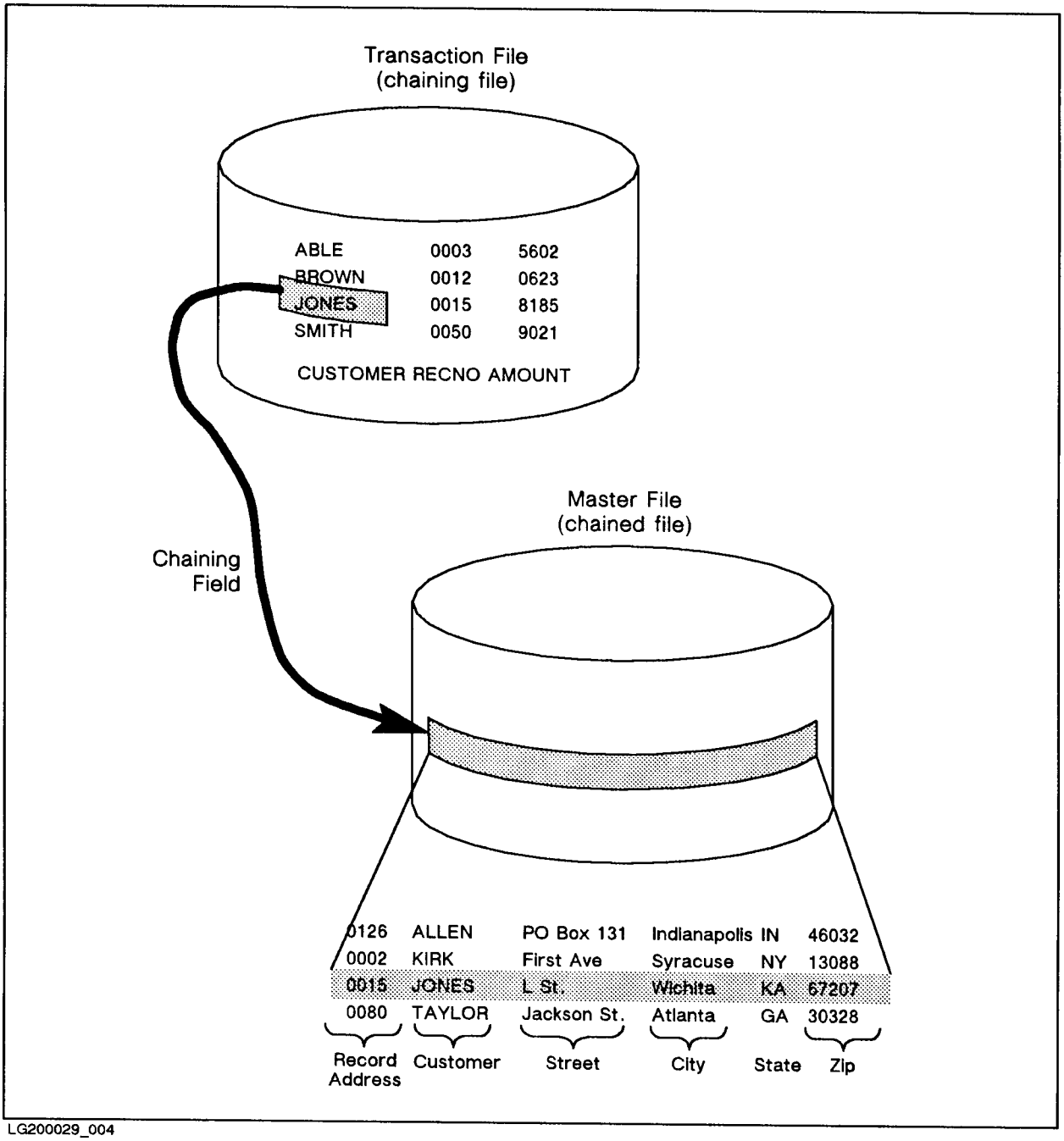
When you assign matching field codes to fields in more than one file, the fields are used as control fields for matching-record processing. Normally (when you do not specify matching-record processing), records are not matched; all records in the primary file are processed first, then records from each secondary file are processed in the order in which you enter them in the File Description Specifications (this is called the "primary/secondary processing order"). When using matching-record processing (when you enter matching code fields for two or more files), records are selected for processing in the order of the matching fields (this is called the "matching-record processing order"). When the matching fields are equal, the matching-record indicator (MR) is turned ON and operations conditioned by the MR indicator are performed. All matching records in the primary file are processed first followed by matching records in the secondary file, in the order in which the File Specifications are entered. Figure 7-18 illustrates how matching-record processing works and the narrative below describes it in detail.

1. The first record in each input, update or combined primary and secondary file is read. A record is selected and processed through the logic cycle.
2. At the start of the next cycle, the next record from the file just processed is read.
3. If one of the files is forced (see the FORCE Calculation Specification operation), its record is selected.
4. If a file does not use matching fields and it is a primary file, its record is selected. If not, the record from the first secondary file is selected.
5. If all of the files use matching fields, the fields are compared. If you specified an alternating collating sequence, fields are compared using that sequence. If a sequence error occurs, the response that you entered in the Error Response Field (column 56-71) of the Header Specification, if any, is performed.

   If the fields do not match and the records are in ascending order, the record with the lowest matching field is selected. If the records are in descending order, the record with the highest matching field is selected.

   If the fields are equal, the record from the primary file is selected, followed by any other records from the primary file having the same matching field value. Secondary records are processed next in the order in which their File Description Specifications are entered.

   The matching-record (MR) indicator is turned ON before detail-time Calculation Specification processing. It remains ON (unless you turn it OFF with a Calculation Specification operation) until this point in the next cycle.



LG200029_004

Figure 7-18. Matching-Record Processing

When you use matching field codes, follow these rules:

1. For each matching file, enter the same sequence in the Input Sequence Field (column 18) of the File Description Specification.
2. Assign matching field codes to at least one record type in two or more files. If you do not, matching-record processing does not take place. When you use more than one matching field code, all fields in the control group must match before the MR indicator is turned ON. If you assign M5, M4, M3 and M2, all four fields from the primary file must match all four fields from the secondary file. You do not have to assign matching field codes to each record type in a file.

   When any matching field (that has the same matching field code) is numeric, all of them are treated as numeric. For example, if one matching field contains 03 and another contains ␣3, they are considered equal. Also, arithmetic signs and decimal points are ignored in matching fields. For example, a field containing 7 is considered equal to a field containing -0.07.
3. Use the same matching field codes for each matching file. The field names to which you assign them, however, do not have to be the same (field names are not used in matching).
4. Be sure that all fields with the same code are the same length and contain the same format (alphanumeric or numeric). Matching fields in binary format are not allowed.
5. Make sure that the combined length of all matching fields does not exceed 256.
6. When you use more than one code, assign them to fields in the order you want these fields combined to form the control field. The fields are combined in the order M9 to M1. For example, if M3 is assigned to field XXX, M7 is assigned to ZZZ and M1 is assigned to YYY, the fields are combined as follows:

   ```
   ZZZXXXYYY
   ```
7. Do not assign the same matching field code to more than one field within a record type. Matching fields cannot be split in this manner.

Example

Figure 7-19 shows how to assign matching field codes to fields in the files ALPHA and BETA. M1 is assigned to field DATA1 in file ALPHA and to field DATA4 in file BETA. M2 is assigned to field DATA2 in file ALPHA and to field DATA3 in file BETA. When the contents of these matching fields are equal, the MR indicator is turned ON.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

IALPHA   AA  01  80 CA
I                                          1   5 DATA1   M1
I                                          6  10 DATA2   M2
IBETA    BB  021 80 CB
I                                         21  25 DATA3   M2
I                                         31  35 DATA4   M1
```

Figure 7-19. Assigning Matching Field Codes

Field Record Relation (Columns 63-64)

When you use OR lines to define record types, and some of the fields are not contained in all types, you can enter an indicator in this field to associate those input fields with the appropriate record type.

Columns 63-64	Description
01-99	General indicator that relates this field to a specific record type.
F0-F9	Function key indicator that relates this field to a specific record type.
H1-H9	Halt indicator that relates the field to a specific record type.
KA-KN, KP-KY	Command key indicator that regulates the acceptance of data from the file.
L1-L9	Control-level indicator that regulates acceptance and use of data from t if a control break occurs. Data is accepted only when the indicator is turned ON.
LR	Last-record indicator that relates the field to a specific record type.
MR	Matching-record indicator that regulates use and acceptance of data from matching field. Data is accepted only when the indicator is turned ON.
OA-OG, OV	Overflow indicator that relates the field to a specific record type.
U1-U8	User indicator that conditions the use of the field.
1P	First-page indicator, used as a general indicator, that relates the file to a specific record type.
blank	The field appears in all record types for this OR relationship.

When using OR lines, you define a field once even though it appears in different record types. This eliminates duplicate coding for those fields that are identical. To distinguish a field that is unique, associate an indicator with it using this field. Use the same indicator that you associated with the record type in the Record Indicator/Look-Ahead/Trailer/Data Structure Field (columns 19-20). When the indicator is ON, it conditionally identifies the field as belonging to the associated record type. (See the RPG logic cycle section in the *HP RPG*

Programmer's Guide for details on how the indicators are used.) For example, you can use the general indicator 10 in this field. This associates the field with the record type having indicator 10 in columns 19-20.

When two control fields (or matching fields) have the same control-level indicator (or matching field code), you can enter an indicator in one of these fields or both of them. Only the field associated with the indicator that is ON is used. The field that does not have an indicator associated with it is used (if defined first) when the indicator is turned OFF.

When you enter an indicator with control-level, matching or chaining fields, follow these rules:

1. Enter specifications that do not use this field before those that do.
2. Enter an indicator for matching and control-level fields that matches the record-identifying indicators for the record type.
3. Group all fields together that relate to one record type. Within the groups, you can enter the fields in any order.
4. Assign the same indicator to all fields that belong to a split control field. Define them together as a group.

Example

Figure 7-20 defines the file INPUT that contains two different types of records. The general indicator 01 specifies that the field ALPHA belongs to record type A. GAMMA and DELTA also belong to record type A because they do not have an indicator associated with them. The general indicator 02 specifies that the field BETA belongs to record type B. Since GAMMA and DELTA do not have an indicator associated with them, they also belong to record type B.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------

          IINPUT   AA  01   1 CA
          I       OR   02   1 CB
(1)       I                                    2   5 ALPHA     01
(2)       I                                    6  10 BETA      02
          I                                   11  15 GAMMA
          I                                   16  20 DELTA
```

Figure 7-20. Using Indicators to Associate Fields with Record Types

Field Indicators (Columns 65-70)

This field lets you test for positive values, negative values, zeros or blanks in the field.

Column	Value	Description
Plus (65-66):	01-99, F0-F9, H1-H9, KA-KN, KP-KY, L1-L9, LR, MR, OA-OG, OV, U1-U8, 1P	The indicator used for testing numeric fields for positive data.
	blank	Do not test data in this field.
Minus (67-68):	01-99, F0-F9, H1-H9, KA-KN, KP-KY, L1-L9, LR, MR, OA-OG, OV, U1-U8, 1P	The indicator used for testing numeric fields for negative data.
	blank	Do not test data in this field.
Zero or Blank (69-70):	01-99, F0-F9, H1-H9, KA-KN, KP-KY, L1-L9, LR, MR, OA-OG, OV, U1-U8, 1P	The indicator used for testing numeric fields for zeros or blanks or alphanumeric fields for blanks.
	blank	Do not test data in this field.

When you enter an indicator in this field, it is turned ON when the condition that you're testing for is true. It is OFF when the condition is not true. For instance, you can use a halt indicator to check a field for a negative quantity. When a negative quantity is encountered, the indicator turns ON and causes a halt at the end of the current cycle (unless you entered a pre-response in the Error Response Field (columns 56-71) of the Header Specification.

When using indicators in this field, remember that:

1. An indicator used for positive or negative testing is OFF when the program begins. It is turned ON when a record is read containing a positive or negative value in the field.
2. An indicator used for zero or blank testing is OFF when the program begins, unless you turn it ON by using the Indicator Setting Field (column 42) of the Header Specification. It is turned ON when a record is read that contains zeros or blanks in the field.
3. When you assign two or three field indicators to one input field, only the indicator which tests for the true condition is turned ON; the others are turned OFF.
4. When you assign the same indicator to fields in different record types, its ON/OFF status reflects the current record being processed. The indicator remains ON until the field no longer meets the testing criteria of the indicator.
5. You can turn indicators ON and OFF with the SETON and SETOFF Calculation Specification operations.

Example

Figure 7-21 shows how to use field indicators to test input data. The field POSDAT is tested for positive values. NEGDAT is tested for negative numbers and the field NODAT is tested for zeros or blanks. When any of the conditions are true, calculations and output conditioned by the indicators are performed.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
     ------------------------------------------------------------------------

     IINPUT   AA  01   1 CY
     I       OR   02   1 CZ
(1)  I                                    1  102POSDAT    0110
(2)  I                                   11  152NEGDAT    02  11
(3)  I                                   16  202NODAT            12
```

Figure 7-21. Using Field Indicators to Test Input Data

Program Name (Columns 75-80)

This field contains the program name. The format for this field is discussed in Chapter 2.

The Input Specification Default Summary

If you leave the optional fields of the Input Specification blank, the default specifications shown in Table 7-1 apply:

Table 7-1. Input Specification Defaults

Columns	Field	Default Values
1-5	Sequence Number	No sequence number applies.
17	Number of Records	No restrictions on types per group; sequencing is not requested.
18	Option/LDA	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.
19-20	Record Indicator/ Look-Ahead/Trailer/ Data Structure	No record-identifying indicator, look-ahead, trailer field, or data structure applies.
21-41	Record Identification Codes	A record identification code does not apply.
43	Data Format	Unpacked decimal format with no leading or trailing signs, or alphanumeric data.
44-47	From Field Position	No field is defined.
48-51	To Field Position	No field is defined.
52	Decimal Positions	The field contains alphanumeric data.
53-58	Field Name	No field is being described.
59-60	Control Level	This field is not a control field.
61-62	Matching/Chaining Fields	This field is neither a matching nor a chaining field.
63-64	Field Record Relation	This field appears in all record types covered by this OR relationship.
65-70	Field Indicators	Do not test data in this field
75-80	Program Name	None.

8

Calculation Specifications

Introduction

The Calculation Specification lets you perform arithmetic operations and other types of operations such as searching tables and arrays, moving data internally, performing subroutines, and using system intrinsics. (See "Operation Field" in this chapter for a list of the Calculation Specification operations.)

The Calculation Specification specifies:

- Whether to perform the operation at detail time or total time; whether to perform the operation as a subroutine; or, whether to perform the operation at all.

  Detail-time operations are performed for each input record. For instance, a program that calculates employee payroll performs detail calculations for each employee record. Total-time operations use data accumulated from a group of related records (a control group). Total-time operations are performed when a control break occurs (the current record's control field differs from that of the previous record). For instance, a payroll that accumulates employee pay by department contains Calculation Specifications for processing department totals.

  To execute a Calculation Specification operation at detail time, leave the Control Level Field (columns 7-8) blank. (Additionally, to execute detail-time calculations only in certain circumstances, "condition" those specifications by entering general indicators in their Indicator Fields (columns 9-17).) To execute a Calculation Specification operation at total time, enter a control-level indicator (L0-L9, LR) in the Control Level Field.

  Subroutines are executed by the EXSR operation.

- The operation to perform, the data to use in the operation, and where to store the result. Specify the operation in columns 28-32, the data in columns 18-27 (or columns 33-42), and the result field in columns 43-51.
- The tests to perform on the results of the operation. You do this by entering indicators in the Resulting Indicators Field (columns 54-59) of the operations that produce the results. Then you condition subsequent operations with those indicators.

Example Conventions

When you enter Calculation Specification operations, enter them in this order:

1. Detail-time operations.
2. Total-time operations.
3. Subroutine operations.

The Calculation Specification is identified by a C in column 6:

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________

C
```

Figure 8-1. The Calculation Specification

The Calculation Specification Fields

The Calculation Specification fields are described in the sections that follow in this chapter. Each field has a unique name and occupies specific positions (columns) in the specification.

Sequence Number (Columns 1-5)

The Sequence Number field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

The Specification Type field contains a C to identify this line as a Calculation Specification.

Control Level (Columns 7-8)

The Control Level field determines when the operation entered in columns 28-32 of this specification is performed. Calculation specifications can be executed at detail time, total time, at the end of the program, or as a subroutine (see the *HP RPG Programmer's Guide* for information on when these events occur in the RPG logic cycle).

Table 8-1. Control Level (Columns 7-8)

Columns 7-8	Description
L0	The operation is always performed at total time because the L0 indicator is always ON. You can use L0 to condition operations on input records that do not have control fields.
L1-L9	The operation is performed at total time before the first record in the new control group (for this indicator) is processed, or after the indicator is turned ON by a SETON operation.
LR	The operation is performed after the last record is processed, or after the LR indicator is turned ON by a SETON operation.
SR	The operation is part of an internal subroutine and is performed by an EXSR operation at detail time or total time (SR in columns 7-8 is optional).
AN	Establishes an AND relationship between indicators on this and previous lines.
OR	Establishes an OR relation between indicators on this and previous lines or in a previous AND relationship.
blank	The operation is performed at detail time. Or, if the calculation specification is within a sub-routine, it is performed when the sub-routine is invoked.

L1-L9 (Control-Level Indicators)

When you use L1-L9 indicators in a Calculation Specification, the operation is performed when there is a control break in the field associated with it (see the Control Level Field (columns 59-60) of the Input Specification) or in a field associated with an indicator of higher rank. For example, if you assign the control-level indicators L1, L2, and L3, a control break that turns ON L3 also turns ON L1 and L2. When the LR indicator is turned ON, L1-L9 are turned ON.

Control-level indicators can also be turned on by the SETON operation, by the record type identification, and by testing non-control fields. Under these conditions, however, control-level indicators of lower rank are not turned ON.

Example

Figure 8-2 shows two operations conditioned by control-level indicators. Line 1 is conditioned by L2 and line 2 is conditioned by L1. When a control break occurs for the control field associated with L2, L1 and L2 are both turned ON and both operations are performed.

```
          1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------------------------

(1)       CL2         ALPHA     ADD  BETA      RES1   100
(2)       CL1                   MOVE RES       OUTPT  10
```

Figure 8-2. Using Control-Level Indicators to Condition Calculation Specification Operations

LR (Last-Record Indicator)

The LR indicator is turned ON after the last input record is processed. When LR is turned ON, the control-level indicators L1-L9 are also turned ON. Use the LR indicator to condition operations that you want to perform at the end of the program.

You can turn the LR indicator ON by entering it in the Resulting Indicators field (columns 54-59) of another specification.

SR (Subroutine Identifier)

You can use the SR identifier with operations that are part of an internal subroutine, though it is optional. Internal subroutines begin with the BEGSR operation and are executed by specification of EXSR operations at detail time. Enter internal subroutine operations after all other Calculation Specifications in the program.

AN(D), OR Lines

AN(D) and OR lines let you condition an operation using more than three indicators. See the Indicators Field (columns 9-17) for more information.

Columns 7-8 on the line preceding the AN(D) and OR lines determines whether the operation takes place at detail time, total time, or end of program. When you use AN(D) lines, the operation is performed only when all indicator conditions are satisfied. For OR lines, the operation is performed when any of the indicator conditions in the OR lines are satisfied.

When entering AN(D) and OR lines, follow these steps:

1. Precede the AN(D) or OR line with a specification that contains L0-L9, LR, SR, or blanks in the Control Level Field (columns 7-8), or from one to three indicators in the Indicators Field (columns 9-17). Leave columns 18-59 blank.
2. Enter the AN(D) or OR line by placing AN or OR in the Control Level Field and the indicators in the Indicators Field. If this is not the last AN(D) or OR line, leave columns 18-59 blank. (You may enter up to seven of these AN(D) and OR lines.)
3. On the last AN(D) or OR line, enter the operation to perform and the data fields to use for the operation.

Example

Figure 8-3 shows how to use AN(D) and OR lines. The AN(D) line (line 1) causes the operation on that line to be executed when indicators 03, 04, 05, and 06 are all turned ON. The AN(D) and OR lines in lines 2 and 3 cause the operation in line 3 to be performed when indicators 01, 02, 03, and 04 are turned ON, or when indicator 05 is turned ON.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------

          C   03 04 05
(1)       CAN 06      ETA       ADD  ZETA      RES3   100
          C                       .
          C                       .
          C   01 02 03
(2)       CAN 04
(3)       COR 05      OMEGA     SUB  OMICRON   RES4   100
```

Figure 8-3. Using Calculation Specification AN(D) and OR Lines

Indicators (Columns 9-17)

The Indicator field is composed of subfields that let you enter up to three indicators for conditioning the operation. These indicators are used in addition to the indicators entered in the Control Level Field (columns 7-8).

Table 8-2. Indicators (Columns 9-17)

Column	Value	Description
Not: (columns 9, 12, 15)	N	The indicator in the indicators subfield (below) must be OFF to condition the operation.
	blank	The indicator in the indicators subfield (below) must be ON to condition the operation.
Indicators: (columns 10-11, 13-14, 16-17)	01-99	General indicators used elsewhere in the program.
	F0-F9	Function key indicators.
	KA-KN, KP-KY	Command key indicators.
	H0-H9	Halt indicators.
	L0-L9	Control-level indicators.
	LR	Last record indicator.
	MR	Matching-record indicator.
	OA-OG,OV	Overflow indicators.
	U1-U8	User indicators.
	1P	First page indicator.
	blank	The operation is performed for every input record if the Control Level Field does not contain L0-L9 or SR.
(column 11 only)	*	The operation is conditioned by the same indicators as the previous Calculation Specification.

All indicator settings (in addition to what is specified in the Control Level Field) must be satisfied to perform the operation. You can enter AN(D) and OR lines to use more than three indicators to condition an operation (see "Control Level Field" in this chapter).

When conditioning operations with indicators, be sure you understand how they are used in the RPG logic cycle (see the *HP RPG Programmer's Guide*).

When using the Indicator Field, you can:

- Use any indicator entered in the Record Indicator/Look-Ahead/Trailer/Data Structure Field (columns 19-20) of the Input Specifications to condition operations for certain record types.
- Use any indicator entered in the Field Indicators Field (columns 65-70) of the Input Specification to condition operations that use fields that meet certain requirements.
- Use any indicator entered in the Resulting Indicators Field (columns 54-59) to condition operations based on the results of previous operations.

H0-H9 (Halt Indicators)

The halt indicators cause the operation to be stopped or skipped when there are input data errors. You can also use them to perform an operation only under certain error conditions. To use a halt indicator in this field, you must have entered it in the Field Indicators Field (columns 65-70) of an Input Specification or in the Resulting Indicators Field (columns 54-59) of a previously executed Calculation Specification.

L0-L9 (Control-Level Indicators)

The control-level indicators cause the operation to be performed at detail time only for the first record of a new control group (either the field associated with this indicator changes or a field associated with an indicator of higher rank changes). The indicator must have been entered in either the Control Level Field (columns 59-60) of an Input Specification or in the Resulting Indicators Field (columns 54-59) of a previously executed Calculation Specification.

When you enter a control-level indicator in this field, do not enter it in the Control Level Field (columns 7-8).

LR (Last-record indicator)

The last-record indicator causes the operation to be performed at end-of-program. RPG turns it ON automatically when the last input record is read. You can turn it ON yourself by entering it in the Resulting Indicators Field (columns 54-59) of another specification.

MR (Matching-Record Indicator)

The matching-record indicator causes the operation to be performed when matching fields are the same for the current input records. You must define matching fields in the Matching/Chaining Fields (columns 61-62) in the Input Specification.

If you entered a control-level indicator (L1-L9) in the Control Level Field (columns 7-8), MR is turned ON when a matching condition occurs for the previous record, not the record that caused the control break.

OA-OG, OV (Overflow Indicators)

The overflow indicators cause the operation to be performed when page overflow occurs. To associate the indicator with page overflow, you must enter it in the Overflow Indicators Field (columns 33-34) of the File Description Specification.

Example Conventions

U1-U8 (User Indicators)

The user indicators indicators cause the operation to be executed when they are turned ON. You can turn the user indicators on yourself with the SETON operation. You get the initial settings (ON or OFF) of these indicators in three different ways.

- By prompting the operator.
- By taking values from the system job control word (JCW).
- By taking values from a file.

* (Column 11 Only)

If you use the same indicators for several consecutive operations, you do not have to repeat them on each line. Enter them on the first line, then enter an asterisk (*) in column 11 on successive lines.

Examples

Figures 8-4 and 8-5 give two examples for using indicators to condition Calculation Specification operations. In figure 8-4, the operation is performed at total time when indicators L6, 01, and 03 are ON and 02 is OFF.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

CL6 01N02 03FAC1      ADD  FAC2      FACX    50
```

Figure 8-4. Using Four Indicators to Condition a Calculation Specification Operation

Figure 8-5 shows how an indicator is turned ON, then used to condition a Calculation Specification operation. When the field CLASS is blank, the general indicator 01 turns ON (line 1). Line 2 performs the ADD operation only when CLASS is not blank (indicator 01 is turned OFF).

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------

          IINFILE  AA
          I                                                 .
          I                                                 .
(1)       I                                        21  25 CLASS          01

          C                       .
          C                       .
(2)       C  NO1       SUMA       ADD  SUMB       SUMC    50
```

Figure 8-5. Using One Indicator to Condition a Calculation Specification Operation

Factor 1 (Columns 18-27)

The Factor 1 field names the operand to use in the operation. What you enter in this field depends on the operation you're using (see the description for the operation in this chapter).

Table 8-3 contains brief descriptions of the Factor 1 field.

Table 8-3. Factor 1 (Columns 18-27)

Columns 18-27	Description
The name of a field, table, array, array element, or file.	The field containing the data or (if this is a file) the name of the file.
Subroutine name.	The internal subroutine to execute.
A label.	The label for a TAG, ENDSR, or GOTO operation.
An alphanumeric, numeric literal, or a figurative constant.	The actual data to be used.
Blank.	The operation does not use an operand in this field.

Field, Table, Array, Subroutine, and Label Names

You must define field names that you enter in this field somewhere in the program. Some field names, however, are predefined; you can use them without defining them. These fields are UDATE, UMONTH, UDAY, UYEAR, PAGE, PAGE1-PAGE7, and *ERROR.

Alphanumeric Literals

Alphanumeric literals are constants that consist of ASCII characters. They can also specify a message identification number in a User Message Catalog (see the MSG operation). Do not use alphanumeric literals in arithmetic operations.

Alphanumeric literals can contain up to eight characters including blanks. When entering them, enclose the characters in quotation marks. For instance, to use the literal ALPHALIT, enter `"ALPHALIT"`. (If you want to use apostrophes instead of quotation marks to enclose alphanumeric literals, enter the apostrophe in the QUOTE= parameter of the $CONTROL compiler subsystem command.) To include a quotation mark in the literal itself, enter two quotation marks. For example, to enter the literal “NAME”, enter `""""NAME""""`.

Numeric Literals

A numeric literal is the actual number that is used in the operation. Use numeric literals the same way you use field names. Numeric literals can contain up to ten characters, including a decimal point and a leading plus or minus sign (unsigned literals are treated as positive numbers). For example, 123.68 is a valid numeric literal.

Do not embed blanks in numeric literals and do not enclose them in quotation marks.

Figurative Constants

Figurative constants are predefined names that, when used, produce one or more identical characters. The figurative constants *BLANK and *BLANKS produce one or more blanks and are normally used with alphanumeric fields. *ZERO and *ZEROS produce one or more zeros and are used with either numeric or alphanumeric fields. The number of blanks or zeros produced depends on the size of the Factor 2 Field (or, if the Factor 2 Field is not used, the Result Field).

You can use figurative constants with operations such as CHAIN, COMP, LOKUP, MOVE, MOVEL, and MOVEA.

Example

Figure 8-6 gives four examples of how to use the Factor 1 field. The operation in line 1 uses a field name DATAX. Lines 2 and 3 use the numeric literals 10 and -1, respectively. Line 4 uses the figurative constant *ZERO.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ____________________________________________________________

1         C           DATAX     ADD  100.50    RESULT  82
          C                      .
          C                      .
2         C           10        MULT ALPHA     PROD    102
3         C           -1        DIV  ALPHA     QUOTA   102
          C                      .
          C                      .
4         C           *ZERO     COMP QUOTA
```

Figure 8-6. Using the Factor 1 Field

Operation (Columns 28-32)

This field contains a word that directs RPG to perform a certain action.

Each type of operation is discussed in this section of this chapter. The operations are further broken down in the next section with a detailed alphabetical description of every RPG calculation specification operation.

Arithmetic Operations

You perform arithmetic operations by entering numeric fields, literals, tables, arrays, or array elements in the Factor 1 (columns 18-27) and Factor 2 Fields (columns 33-42). Enter a numeric field, table, array, or array element in the Result Field (columns 43-48). The length of fields used in arithmetic operations cannot exceed 15 digits. Do not enter a literal or alphanumeric field in the Result Field. The Factor 1, Factor 2, and Result Fields can all contain the same field or different fields. Fields used in Factor 1 and Factor 2 are unaffected by arithmetic operations unless these fields are also named in the Result Field.

If Factor 1 or Factor 2 contains an array name, the Result Field must also contain an array name (unless this is an XFOOT operation). If you use more than one array name in an operation, and the arrays contain different numbers of elements, the operation is performed for the number of elements in the smallest array. When you use a single field and an array name, the field is applied to each element in the array.

With arithmetic operations, the Result Field will **overflow** and data will be lost if the Result Field is not large enough to hold the result of the operation. With full numeric overflow checking enabled (through the $CONTROL OVFLOCHK compiler command) an arithmetic overflow initiates the overflow error trap. You can control how your program handles the overflow trap with entries in the Header Specification, columns 55 and 65. If you disable full overflow checking for some calculations with the $CONTROL NOOVFLOCHK compiler command (or if you haven't enabled full checking), RPG may left-truncate the calculated value instead of initiating the overflow error trap. This happens when either Factor 1 or Factor 2 is the same size or larger than the calculated value of the operation. If the size of the calculated value is larger than the size of each of Factor 1, Factor 2, and the Result Field, then RPG initiates the overflow error trap, regardless of the setting of OVFLOCHK.

The arithmetic operations are listed below and are described in the pages which follow:

ADD, DIV, MULT, MVR, SQRT, SUB, XFOOT, Z-ADD, Z-SUB

Move Operations

Move operations move all or part of Factor 2 to the Result Field. Factor 2 remains unchanged. Do not use the Factor 1 field, resulting indicators field, or half adjusting field. When data are moved to numeric fields, no decimal alignment takes place. For example, if the number 2.35 has two decimal positions and is moved to a field with one decimal position, the result is 23.5.

You can convert a numeric field to an alphanumeric field by entering the numeric field name in the Factor 2 Field and the alphanumeric field name in the Result Field. Packed decimal data are converted to unpacked format.

You can convert an alphanumeric field to a numeric field by entering the alphanumeric field name in the Factor 2 Field and the numeric field name in the Result Field. Unpacked data

are converted to packed decimal format. The Factor 2 Field must contain numeric data only, except for the sign position which can contain a digit 0-9 or a letter A-R.

The move operations are listed below and are described in the pages which follow:

MOVE, MOVEA, MOVEL

Move Zone Operations

These operations move the sign bits from the high-order (leftmost) or low-order (rightmost) position of the Factor 2 Field to the high-order or low-order position of the Result Field. The digit portion of the high-order or low-order position can be moved in a similar manner. Thus, these operations are useful for converting from one character representation to another.

Factor 2 can be a literal or a field. (Do not use the literal "-0" to make a field negative. RPG does not recognize it as a valid negative quantity. Instead, use the literal "-".) The Result Field must contain a field name. When Factor 2 is a numeric field, the sign moved to the Result Field is either positive (1100), negative (1101), or unsigned (1111).

The move zone operations are listed below and are discussed in detail in the pages which follow:

MHHZO, MHLZO, MLHZO, MLLZO

Compare and Test Operations

The compare and test operations test fields for certain conditions. The results of the test are reflected by the status of the resulting indicators you enter in the Resulting Indicators Field (columns 54-59). The operations do not alter the contents of the fields being tested.

The compare and test operations are listed below and are discussed in the pages which follow:

COMP, TESTB, TESTN, TESTZ

Branching Operations

Calculation Specifications are executed in the order you enter them. Occasionally, you may want to change the order of execution. For instance, under certain conditions you may want to bypass one or more calculations or you may want to repeat a group of operations several times in the same program cycle.

The branching operations are listed below and are discussed in detail in the pages which follow:

GOTO, TAG

Internal Subroutine Operations

When a program performs the same operations under different conditions, you can code the operations once as an internal subroutine and then branch to the subroutine to perform the operations. You code internal subroutines as part of the RPG program. (External subroutines are not part of the program. They are generally routines that are used by many programs.)

Figure 8-7 shows how to enter an internal subroutine. Internal subroutines follow all other Calculation Specifications in the program. You can identify them by entering an SR into the Control Level Field (columns 7-8), although this is optional. The BEGSR operation (line 2) marks the beginning of the internal subroutine. Enter a unique name for the subroutine in the Factor 1 Field. The name can be the same as a field name but must not be the same name used with a TAG operation or another BEGSR operation. The operations that are part of the internal subroutine follow the BEGSR operation (line 3). The ENDSR operation ends the internal subroutine (line 4). The internal subroutine is executed from the main program by the EXSR operation (line 1).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

     C                      .
     C                      .
     C                   (main program code)
     C                      .
     C                      .
(1)  C                   EXSR SUBNAM
     C                      .
     C                      .
(2)  CSR         SUBNAM    BEGSR
     CSR                    .
     CSR                    .
(3)  CSR                   (subroutine body)
     CSR                    .
     CSR                    .
(4)  CSR                   ENDSR
```

Figure 8-7. Using an Internal Subroutine

Within a subroutine, you can condition an operation by entering indicators in the Indicators Field (columns 9-17). You cannot use control-level indicators in the Control Level Field (columns 7-8) to condition them; however, you can use them to condition the EXSR operation that performs the subroutine. You can define fields inside or outside of an internal subroutine. Fields that you define within the subroutine can be referenced outside of it also.

There is no limit to the number of internal subroutines you can use in a program, and you do not have to enter them in the order in which they are executed. Subroutines can contain EXSR operations that call other internal subroutines. However, a subroutine cannot call itself nor can it call a subroutine that calls it. Do not nest subroutines.

You can share internal subroutines with other programs by putting them onto disk and incorporating them into the programs by placing $INCLUDE statements at the points where the subroutine would normally be coded. When you do this, be careful to use the correct names for fields or labels used in the subroutine.

The internal subroutine operations are listed below and are described in the pages which follow:

BEGSR, ENDSR, EXSR

External Subroutine Operations

External subroutines are subroutines written in C, Pascal, COBOL, SPL (CM only), or FORTRAN that are executed from the RPG program. Use external subroutines for routines that are common to many programs.

You must compile external subroutines separately from the RPG program and place them in executable libraries (see the *HP Link Editor/iX Reference Manual* for details on placing compiled subroutines in executable libraries).

You execute (call) external subroutines using the EXIT operation. You pass information (parameters) to and from the subroutine using either, or both, the RLABL or PARM operations. Parameters can be fields, tables, and arrays. Additionally, if you use RLABL, you can pass indicator settings.

RPG passes data to an external subroutine by giving it the byte address of the data. If the data is numeric (all numeric data is stored internally in packed decimal format) and the external subroutine is written in COBOL, define the data as COMP-3. If you're not using COBOL, you must convert the numeric data yourself. When defining (declaring) data in an external subroutine, be sure to use the correct field type and length.

When compiling COBOL external subroutines to be run in MPE/iX compatibility mode, include the ANSIPARM parameter in the $CONTROL compiler directive. This parameter aligns parameters on byte boundaries, which RPG expects. (When compiling COBOL external subroutines in MPE/iX native mode, byte alignment is automatic.)

Note

When coding in C, the procedure name must be in lowercase. If coding in Pascal, FORTRAN, or COBOL, the compilation process will do any downshifting necessary.

The external subroutine operations are listed below and are discussed in the pages which follow:

EXIT, PARM, RLABL

Structured Programming Operations

Structured programming operations let you use structured programming techniques. Structured programming is a coding discipline that seeks to keep routines in manageable, functional units that can be tested and debugged separately. Routines in structured programs are modular and insulated from other routines; no direct branching into and out of them is allowed.

The RPG structured programming operations make a program easier to read, they reduce the number of Calculation Specifications, and they eliminate complex use of indicators. The structured programming operations are listed below and are described in detail in the pages which follow:

CABxx, CASxx, DO, DOUxx, DOWxx, ELSE, END, IFxx

Indicator and Bit Setting Operations

These operations let you turn indicators On and OFF at will. They also let you turn individual bits in a one-character alphanumeric field ON and OFF.

When you're turning record-identifying indicators ON and OFF using the indicator operations, remember that they are reset automatically whenever a record is read from the associated file. Similarly, the record-identifying and the control-level indicators (L1-L9) are turned OFF at the end of detail-time processing.

The indicator and bit setting operations are listed below and are discussed in the pages which follow:

BITOF, BITON, SETOF, SETON

Table and Array Operations

The table and array operations let you search an array, a single table, or alternating tables. To ensure that arrays are in ascending or descending sequence before you search them, you can use the SORTA operation.

The table and array operations are listed below and are described in the pages which follow:

LOKUP, SORTA

File Operations

The file operations let you perform all file handling functions. Some operations read records from chained, demand, indexed and full procedural files. There are operations that let you obtain exclusive access to a TurboIMAGE or KSAM file. You can perform input and output during Calculation Specification processing and you can specify the file from which the next record will be read.

The file operations are listed below and are described in the pages which follow:

CHAIN, CLOSE, EXCPT, FORCE, LOCK, READ, READE, READP, RESET, SETLL, UNLCK

Display Operations

The display operations let you display data and messages on the terminal. You can also use them to enable the function keys on the keyboard and to sense when the user presses them.

The display operations are listed below and are described in the pages which follow:

DSPLM, DSPLY, MSG, SET

Debugging Operation

This operation facilitates the debugging process by displaying (at any point in the Calculation Specifications) the indicators that are ON and, optionally, a field in the program.

The debugging operation is listed below and is described in the next section:

DEBUG

System Operations

The system operations listed below let you use or access various operating system functions without directly using system intrinsics or system commands:

FNDJW, FNUM, PUTJW, SUSP, TIME, TIME2

Two additional system operations let you call system intrinsics directly from your RPG program:

INTR, IPARM

For information on the system intrinsics discussed in the following sections, see the *MPE/iX Intrinsics Reference Manual*.

Operation Definitions

The definitions in this section are listed in alphabetical order, not by type of operation.

ADD

This **arithmetic** operation adds Factor 1 to Factor 2 and places the sum in the Result Field. If Factor 1 is blank, it adds Factor 2 to the Result Field and places the sum in the Result Field.

BEGSR

This **internal subroutine** operation marks the entry point to an internal subroutine (see figures 8-7 and 8-26). Enter the name (label) of the subroutine in the Factor 1 Field. The label can be up to six characters long, and must begin with a letter or the special character @, $, or #. The remaining characters can be letters, digits, or the special characters @, $, or # in any combination. Use this label only in an EXSR operation to perform the subroutine (do not use it in a GOTO operation, in or out of the subroutine.) Enter only one BEGSR operation per subroutine.

BITOF

This **indicator and bit setting** operation turns OFF bits in the Result Field. The bits that are turned OFF are determined by the bit settings of the Factor 2 Field.

The Result Field must be a one-character alphanumeric field. You can enter an alphanumeric literal, a field, a table, or an array element in the Factor 2 Field. In all these cases, Factor 2 must be a single character. If you enter a field, table, or array element in Factor 2, the bits that are turned OFF in the Result Field are the ones that are OFF in the Factor 2 field. If you enter an alphanumeric literal, each digit in it identifies a bit to be turned OFF. Enter up to eight digits for the literal enclosed in quotation marks. Enter a digit for each bit position you want to turn OFF. Zero (0) stands for the high-order bit and seven for the low-order bit. You can enter the bit numbers in any order; if you do not enter a number for a bit, it remains unchanged. For example, "02" turns bits 0 and 2 OFF.

Do not use the Factor 1, Decimal Positions (column 52), Half Adjust (column 53), or Resulting Indicators (columns 54-59) Fields.

See figure 8-8 for an example of how to use BITOF.

BITON

This **indicator and bit setting** operation turns ON bits in the Result Field. The bits that are turned ON are determined by the Factor 2 Field.

The Result Field must be a one-character alphanumeric field. You can enter an alphanumeric literal, a field, a table, or an array element in the Factor 2 Field. In all these cases, Factor 2 must be a single character. If you enter a field, table, or array element in Factor 2, the bits that are turned ON in the Result Field are the ones that are ON in the Factor 2 field. If you enter an alphanumeric literal, each digit in it identifies a bit to be turned ON. Enter up to eight digits for the literal enclosed in quotation marks. Enter a digit for each bit position you want to turn ON. Zero (0) stands for the high-order bit and seven for the low-order bit. You can enter the bit numbers in any order; if you do not enter a number for a bit, it remains unchanged. For example, "02" turns bits 0 and 2 ON.

Do not use the Factor 1, Decimal Positions (column 52), Half Adjust (column 53), or Resulting Indicators (columns 54-59) Fields.

Example

Figure 8-8 shows how to use the BITON and BITOF operations. The BITON operation in line 1 turns ON bits 3 and 4 (the fourth and fifth bits) in the field BITFLD. The BITOF operation in line 2 turns OFF the bits in BITFLD that are also OFF in the field MYFLD.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________

(1)  C                   BITON"34"      BITFLD
     C                    .
     C                    .
(2)  C                   BITOFMYFLD     BITFLD
```

Figure 8-8. Using the BITON and BITOF Operations

CABxx

This **structured programming** operation (Compare And Branch) compares Factor 1 to Factor 2. If the results of the comparison match the condition specified by xx, control branches to the TAG label entered in the Result Field.

Table 8-4. CABxx Operations

Enter this operation:	To test if Factor 1 is:
CAB	No test takes place; the branch is unconditional.
CABEQ	Equal to Factor 2.
CABGE	Greater than or equal to Factor 2.
CABGT	Greater than Factor 2.
CABLE	Less than or equal to Factor 2.
CABLT	Less than Factor 2.
CABNE	Not equal to Factor 2.

If you want to condition the operation, enter one or more indicators in the Indicators Field (columns 9-17).

You can enter a field name or a literal in the Factor 1 and Factor 2 Fields. Numeric fields are aligned by decimal point before they are compared. Shorter fields are padded with zeros (to the left and right) to make them the same size. Blanks in numeric fields are treated as zeros. The maximum numeric field length is 15 digits. Alphanumeric fields are aligned starting with their high-order (leftmost) characters. Shorter fields are padded with blanks (on the right) to make them the same size. If you specified an alternate collating sequence, it is used.

You can compare a numeric field to an alphanumeric field. The numeric field is temporarily converted to alphanumeric format and the two fields are compared as if they both were alphanumeric. The numeric field is not aligned by decimal point before the compare. Negative signs are ignored. For example, the number -123.45 becomes “12345”. To avoid problems when comparing the numeric field, make sure it does not have decimal places, is the same length and is not negative. When you compare a numeric field to an alphanumeric field, you see this compiler message:

```
9016I NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE
```

You may enter one or more resulting indicators in the Resulting Indicators Field (columns 54-59). The resulting indicators are turned ON or OFF depending on the results of the compare and are unaffected by whether a branch is performed. If you use the CAB operation, you must enter at least one resulting indicator.

Example

Figure 8-9 shows how to enter the CABGE operation. The field CODE is compared to the literal 21. If CODE is greater than or equal to 21, control skips to the label SELECT. Indicator 20 is turned ON when CODE is less than 21.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
     ____________________________________________________________________________________

(1)  C           CODE      CABGE21        SELECT       20
     C                        .
     C                        .
(2)  C           SELECT    TAG
     C                        .
     C                        .
```

Figure 8-9. Using the CABGE Operation

CASxx

This **structured programming** operation (CASe) compares Factor 1 to Factor 2. If the results of the comparison match the condition specified by xx, control branches to the internal subroutine named in the Result Field.

Table 8-5. CASxx Operations

Enter this operation:	To test if Factor 1 is:
CAS	No test takes place; the subroutine is performed unconditionally.
CASEQ	Equal to Factor 2.
CASGE	Greater than or equal to Factor 2.
CASGT	Greater than Factor 2.
CASLE	Less than or equal to Factor 2.
CASLT	Less than Factor 2.
CASNE	Not equal to Factor 2.

Enter all CASxx operations together as a group without other operations interspersed. Terminate the group with an END operation. When the CASxx operations are executed, each Factor 1 Field is compared to each Factor 2 Field in the operations until a condition being tested is satisfied or until a CAS operation is encountered. The subroutine in the Result Field of that line is executed and control returns to the line following the END operation for the CASxx group.

If you want to condition the CASxx operation, enter one or more indicators in the Indicators Field (columns 9-17).

You can enter a field name or a literal in the Factor 1 and Factor 2 Fields. Numeric fields are aligned by decimal point before they are compared. Shorter fields are padded with zeros (to the left and right) to make them the same size. Blanks in numeric fields are treated as zeros. The maximum numeric field length is 15 digits. Alphanumeric fields are aligned starting with their high-order (leftmost) characters. Shorter fields are padded with blanks (on the right) to make them the same size. If you specified an alternate collating sequence, it is used.

You can compare a numeric field to an alphanumeric field. The numeric field is temporarily converted to alphanumeric format and the two fields are compared as if they both were alphanumeric. The numeric field is not aligned by decimal point before the compare. Negative signs are ignored. For example, the number -123.45

becomes "12345". To avoid problems when comparing the numeric field, make sure it does not have decimal places, is the same length and is not negative. When you compare a numeric field to an alphanumeric field, you see this compiler message:

```
9016I NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE
```

You may enter one or more resulting indicators in the Resulting Indicators Field (columns 54-59). If you use a CASxx operation with one or more resulting indicators, you must use the Factor 1 and Factor 2 Fields.

Example

Figure 8-10 shows how to enter the CASxx operation. The CASxx group of operations are entered in lines 3-5 and are executed by the GOTO operation in line 1. The CASEQ operation in line 3 performs the subroutine SUB21 when indicator 20 is OFF and the field CODE equals 21. The CASGE operation in line 4 performs the subroutine SUB5X when indicator 20 is ON and the field CODE2 is greater than or equal to 5. If neither of the CASxx conditions in lines 3-4 are satisfied, the CAS operation in line 5 executes the subroutine SUBXXX.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________

C                    SETON                     20
C                      .
C                      .
C                    GOTO SELECT
C                      .
C                      .
C           SELECT   TAG
C  N20      CODE     CASEQ21        SUB21
C   20      CODE2    CASGE5         SUB5X
C                    CAS            SUBXXX
C                    END
```

1 2 3 4 5

Figure 8-10. Using the CASxx Operation

CHAIN

This operation reads a record from a KSAM, TurboIMAGE, or MPE direct-access file. CHAIN lets you retrieve records randomly; you enter identifying information for the record you want to read. Similarly, you can use CHAIN to write records randomly to these files.

There are two ways to access records with CHAIN. You can access a file by relative record number or record key. A relative record number is a record number that is "relative" to the first record in the file. For example, record 5 is the fifth physical record in a file. For KSAM and TurboIMAGE files, you can access records by entering a record's key value. For example, to access the record having a key value of 06355, enter the number 06355. You can calculate the record number or key value in the program or you can get it from records in other input files.

You must define the file that you're chaining to with a D (direct-access), I or X (KSAM), or M (TurboIMAGE) in the File Organization Field (column 32) of the File Description Specification. Enter C (chained) in the File Designation/Additional I/O Area (column 32) of that specification and if this is an output file, enter O in the File Type Field (column 15). When you chain successfully to a file, the record-identifying indicators (entered in the Record Indicator/Look-Ahead/Trailer/Data Structure Field (columns 19-20) of the Input Specification) for the file remain ON throughout the cycle. If you chain to the same output file more than once in the same cycle, only the last record is updated unless you use exception lines with each CHAIN operation.

To condition this operation, enter indicators in the Control Level (columns 7-8) or the Indicators Field (columns 9-17), or both.

Enter a relative record number or record key in the Factor 1 Field. If reading by record number or packed record key, use a numeric literal or field. If reading by record key, use an alphanumeric literal, field, array, or table. You can enter a *partial* key when reading by record key. (A partial key contains characters that are compared to the corresponding characters in the high-order, leftmost, position of the record key. As long as the characters in the partial key match, a record is selected for processing regardless of whether the remaining characters in the key match.) Additionally, you can enter a relational operator in the Result Field that specifies the relationship that the record key value must have with Factor 1 to be selected for processing. For example, to select a record whose key value is equal to or less than Factor 1, enter the relational operator *LT in the Result Field.

Enter the name of the chained or full procedural file in the Factor 2 Field.

Enter the indicator in the High Subfield (columns 54-55) that you want to turn ON when a record is not found. For TurboIMAGE files, enter the indicator in the Low Subfield that you want to turn ON when the end-of-chain is reached. (A *chain* is a group of records having the same key value.) When you leave either or both the High and Low Subfields blank, H0 is turned ON.

Use the Low Subfield for chained sequential (the input/output mode field for the file in the File Description Specification is C) and backward chained sequential (the input/output mode field is R) operations. When performing these operations, RPG sequentially retrieves all of the records in a chain. You initiate a chained sequential read by placing a new key value in Factor 1. RPG finds and reads the first record in the chain (or the last record if reading backward). For TurboIMAGE files, RPG uses the master data set to find the head of chain in the detail data set. (Do not, however, define the master data set in a File Description Specification.) If you leave the contents of Factor 1 unaltered, a subsequent CHAIN operation reads the next record in the chain. When the end-of-chain is reached, the Low Subfield indicator is turned ON. Table 8-6 summarizes when the High and Low Subfield indicators are turned ON and OFF.

Table 8-6. How CHAIN Sets the High and Low Resulting Indicators

High Resulting Indicator (Columns 54-55)	Low Resulting Indicator (Columns 56-57)	Description
OFF	OFF	Successful retrieval of the next record in the chain.
OFF	ON	End-of-chain for the current group of records with the same key value (the second or subsequent CHAIN operation for the same key value has failed).
ON	ON	No records for this key value exist in this detail data set, but an entry for this key was found in the master data set (the first CHAIN operation with a new key value has failed).
ON	OFF	No records for this key value exist in the master and detail data sets (the first CHAIN operation with a new key value has failed).

Example

Figure 8-11 shows how to use the CHAIN operation to read records from the direct-access file CHAINFL (line 2). Records are read by record number. The CHAIN operation in line 4 reads records in CHAINFL using the field ID (line 3) to identify the record numbers. The record numbers in ID are obtained from the input file INFILE (line 1).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
_____________________________________________________________________________

1    FINFILE  IPE F      80             DISK
2    FCHAINFL UC  F     256    D       DISK

     IINFILE  AA  01  80 CX
     I                                        1  10 NAME
3    I                                       11  15 ID
     I                                       16  20 PROD
     I                                       21  25 AMT
     I
     ICHAINFL BB  02  01 CY
     I                                       05  10 ID
     I                                       11  20 STREET
     I                                       21  30 CITY
     I                                       31  35 STATE
     I                                       36  40 ZIP

4    C           ID        CHAINCHAINFL              50
```

Figure 8-11. Chaining to an Input File

Figure 8-12 shows how to create a direct-access chained file CHFILE from a sequential input file INFILE. During each logic cycle, a record is read from INFILE (line 1). When the CHAIN operation (line 3) is executed the field DATA is written to CHFILE (line 4) using the record number in the field RECNO (see line 2). The first time CHAIN is executed, it sets the five extents allocated for the chained file to blanks.

The *HP RPG/iX Programmer's Guide* gives additional examples on chaining to a KSAM file.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
     ------------------------------------------------------------------------------------

1    FINFILE  IPE F      80            DISK
     FCHFILE  0C  F 240  80R   D       DISK

     IINFILE  AA  01
2    I                                       1   50RECNO
     I                                       1  80 DATA

3    I           RECNO     CHAINCHFILE

     0CHFILE           01
     0                       DATA      80
```

Figure 8-12. Creating a Chained File

CLOSE

This **file** operation closes an open file. It performs normal end-of-file actions before the program actually ends. Use CLOSE to release a file so that others can gain access to it or to release system resources used by a file that you've finished processing. You can also use CLOSE to release printer files for printing. Once you use CLOSE for a file, you cannot use the file in subsequent input/output operations.

To use CLOSE, enter the name of the file to close in the Factor 2 Field and enter an indicator in the Low Subfield (columns 56-57). The Low Subfield indicator is turned ON if the file cannot be closed.

CLOSE is similar to the Release Option (see the Fetch Overflow/Release Field (column 16) of the Output Specification).

COMP

This **compare and test** operation compares Factor 1 to Factor 2. You can enter a field name, literal, or figurative constant into either field. When COMP is executed, the resulting indicators that you enter in columns 54-59 are turned ON according to the results of the compare.

Table 8-7 briefly describes the COMP operations.

Table 8-7. COMP Operations

When:	This Resulting Field Indicator is Turned ON:
Factor 1 is greater than Factor 2	High (columns 54-55).
Factor 1 is less than Factor 2	Low (columns 56-57).
Factor 1 equals Factor 2	Equal (columns 58-59).

Numeric fields are aligned by decimal point before they are compared. Shorter fields are padded with zeros (to the left and right) to make them the same size. Blanks in numeric fields are treated as zeros. The maximum numeric field length is 15 digits.

Alphanumeric fields are aligned starting with their high-order (leftmost) characters. Shorter fields are padded with blanks (on the right) to make them the same size. If you specified an alternate collating sequence, it is used.

You can compare a numeric field to *ZERO(S) and an alphanumeric field to *BLANK(S) or *ZERO(S). You can also compare a numeric field to an alphanumeric field. The numeric field is temporarily converted to alphanumeric format, and the two fields are compared as if they both were alphanumeric. The numeric field is not aligned by decimal point before the compare. Negative signs are ignored. For example, the number -123.45 becomes "12345". To avoid problems when comparing the numeric field, make sure it does not have decimal places, is the same length, and is not negative. When you compare a numeric field to an alphanumeric field, you see this compiler message:

```
9016I NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE
```

Example

Figure 8-13 shows two COMP operations. The COMP operation in line 1 compares FIELD1 to FIELD2. If they are equal, indicator 03 turns ON. If FIELD1 is greater than FIELD2, indicator 01 turns ON. If FIELD1 is less than FIELD2, indicator 03 turns ON. The COMP operation in line 2 compares the field STATUS to the word EXEMPT. If they are equal, indicator 05 turns ON.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
_____________________________________________________________________________

1    C           FIELD1    COMP FIELD2               010203
     C                       .
     C                       .
2    C           "EXEMPT"  COMP STATUS                  05
```

Figure 8-13. Using the COMP Operation

DEBUG

This **debugging** operation assists you in debugging a program by showing the indicators that are turned ON and optionally, a field in the program. Place the DEBUG operation at one or more points in the Calculation Specifications where you want to capture DEBUG information. The captured information is saved in a file that you name.

To enable DEBUG, enter 1 in the Debug Field (column 15) of the Header Specification. If you leave this field blank, DEBUG operations in the program are ignored and compiler warning messages are printed.

To condition this operation, enter indicators in the Control Level Field (columns 7-8) or the Indicators Field (columns 9-17), or both. Leave columns 48-59 blank.

When you use DEBUG, you can enter a literal, field, or array element in the Factor 1 Field that contains up to eight characters. This value identifies the DEBUG information and is helpful when you use more than one DEBUG operation or when DEBUG is included in a loop. Enter the name of the file to which you want to write the debugging information in the Factor 2 Field (make sure that you define this file with File Description Specifications, just as you would any other file). You may use different files for each DEBUG operation if you wish.

You may enter a field, table, array name, or array element in the Result Field (the maximum number of characters that it can contain is 256); its value is displayed after the indicator settings.

Table 8-8 describes how RPG displays the indicators and field information. The "Indicator lines" show the indicators that are ON and the contents of the field or literal you entered in the Factor 1 Field. If there are more indicators than will fit on one line, additional lines are written (positions 1-40 are blank on these lines, and the indicators values start in position 41). The "Result Field lines" show the contents of the field, table element, array, or array element entered in the Result Field.

Table 8-8. Output from the DEBUG Operation

Position	Contents
Indicator lines:	
1 - 8	DEBUG=
9 - 12	The source program sequence number assigned to this DEBUG Calculation Specification. Along with the Factor 1 Field, this number helps to identify this particular DEBUG operation.
13 - 14	Blank.
15 - 22	The contents of the field literal entered in the Factor 1 Field.
23	A minus sign if the Factor 1 Field is negative; a blank if it is zero or positive.
24	Blank.
25 - 39	INDICATORSON=
40	Blank.
41 - 80	The indicators that are ON. The indicator values are separated by blanks.
Result Field Lines:	
1 - 13	FIELDVALUE=
14	Blank.
15 - 80	Th 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 - 80 of the first line following the indicator listing. If this data exceeds 667 characters, it is continued in positions 15 - 80 of as many additional lines as required, up to a maximum of 256 characters. If this is numeric data, it is written in unpacked format and zero surpressed, with an sign appearing to the right of the field (a minus sign for negative values and a blank for positive values). Fields containing zeros show only the rightmost zero and the sign. No other editing is performed. If the Result Field contains an array name, the elements are written in order, each on an individual line.

Example

Figure 8-14 shows how to use DEBUG to display the indicators that are ON and to display the contents of an array element. This information is written to a file rather than being displayed on the terminal. The DEBUG operation writes the debug information to the file BADFILE and tags this output with the word FIRST. The contents of the third array element of ARR1 is also written. Sample DEBUG output for this example is shown below:

```
DEBUG= 0019   FIRST     INDICATORS ON= 01 02 03 05 07 11 LO L1 MR OF
FIELD VALUE= 370
```

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
     ------------------------------------------------------------------------------------

     C           'FIRST'   DEBUGBADFILE   ARR1,3
```

Figure 8-14. Using the DEBUG Operation

DIV

This **arithmetic** operation 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 is zero. If Factor 2 is zero, an error occurs (you can determine the action to take by entering the appropriate code in column 66 of the Header Specification). If DIV produces a remainder, it is lost unless you immediately follow DIV with an MVR operation that moves it to the Result Field. (When you use MVR, you cannot half adjust the quotient.)

The result of DIV must not exceed 15 digits. If it does, excess digits are truncated.

Example

Figure 8-15 shows how to perform a divide operation and save the remainder. The DIV operation divides DIVIDN by DIVSOR saves the result in QUOTNT. The remainder is saved in the field REMAIN by the MVR operation.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678
-------------------------------------------------------------------------

C   26      DIVIDN    DIV  DIVSOR    QUOTNT 102
C   26                MVR            REMAIN  52
```

Figure 8-15. Using the DIV and MVR Operations

DO

This **structured programming** operation executes the Do Block following it until a specified condition is met. With the DO operator you specify the initial value for the loop variable in Factor 1 and the limit value in Factor 2. These values must be numeric if specified. You may leave Factor 1 or Factor 2 blank. RPG will assume a value of 1 for either or both Factors which are left blank.

In the Result Field you specify the name of a numeric field to control the loop. Make sure this field is large enough to hold a number that is one larger than the limit value for the loop (or the loop will never end). For example, a two-digit field is not large enough for the control variable when the loop limit is 99 (99 plus 1 does not fit in two digits). If you leave the Result Field blank, RPG will generate an internal field as the loop counter.

The DO loop must be terminated with an END statement. To determine the loop increment for the DO loop, specify a numeric value in Factor 2 of the END statement. If no numeric value is specified, the loop increment will default to 1.

RPG will compare the loop control variable (Result Field) with the limit (Factor 2). If the index exceeds the limit, control will pass to the statement following END. Otherwise, the loop contents will be executed, the loop control variable will be incremented at the bottom of the loop, and the DO loop will repeat.

Note that a conditioning indicator on the DO statement will condition the entire loop. If such a conditioning indicator, or indicators, evaluate to false, control will skip to the statement following END, not to the statement following DO.

Leave resulting indicators blank for the DO operator.

Example

In Figure 8-16 the DO loop will execute four times, for INDX values 18, 20, 22, and 24. When INDX is incremented to 26, the test 26 <= 25 will fail and control will pass to the instruction following END.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
_______________________________________________________________________________

C           18        DO   25        INDX    30
C                     .
C                     .
C                     END  2
```

Figure 8-16. Using the DO Operation

DOUxx

This **structured programming** operation (DO Until) performs the block of operations (Do Block) immediately following it until Factor 1 and Factor 2 meet the criteria specified by xx. The Do Block is performed at least once.

Table 8-9. DOUxx Operations

Enter this operation:	To perform the Do Block until Factor 1 is:
DOUEQ	Equal to Factor 2.
DOUGE	Greater than or equal to Factor 2.
DOUGT	Greater than Factor 2.
DOULE	Less than or equal to Factor 2.
DOULT	Less than Factor 2.
DOUNE	Not equal to Factor 2.

End the operations in a Do Block with an END operation. Do not split a Do Block between detail, total, or subroutine operations. You may branch into or out of a Do Block, but be sure that you understand what the results will be. Do Blocks can be nested; that is, a Do Block can be contained within another Do Block as shown below (the maximum number of levels of nesting is 100).

```
Begin Level 1 DO Block    IFEQ          (If equal then do)
                          .
                          .
                          ELSE          (Else do)
                          .
                          .
Begin Level 2 DO Block    DOULT         (Do until less than)
                          .
                          .
Begin Level 3 DO Block    DOWNE         (Do while not equal)
                          .
                          .
End Level 3 DO Block      END
End Level 2 DO Block      END
End Level 1 DO Block      END
```

Example Conventions

To conditionally execute the DOUxx operation, enter one or more indicators in the Indicators Field (columns 9-17). You can prematurely end execution of a Do Block by entering one or more indicators in the Indicators Field of the END operation. When the indicator conditions are no longer satisfied, control skips to the operation following END.

You can enter a field name or a literal in the Factor 1 and Factor 2 Fields. Numeric fields are aligned by decimal point before they are compared. Shorter fields are padded with zeros (to the left and right) to make them the same size. Blanks in numeric fields are treated as zeros. The maximum numeric field length is 15 digits. Alphanumeric fields are aligned starting with their high-order (leftmost) characters. Shorter fields are padded with blanks (on the right) to make them the same size. If you specified an alternate collating sequence, it is used.

You can compare a numeric field to an alphanumeric field. The numeric field is temporarily converted to alphanumeric format and the two fields are compared as if they both were alphanumeric. The numeric field is not aligned by decimal point before the compare.

Negative signs are ignored. For example, the number -123.45 becomes "12345". To avoid problems when comparing the numeric field, make sure it does not have decimal places, is the same length and is not negative. When you compare a numeric field to an alphanumeric field, you see this compiler message:

```
9016I NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE
```

Do not use the Result Field or the Resulting Indicators Field (columns 54-59).

Example

Figure 8-17 shows how to use the DOUxx operation. The DOULT operation in line 1 is executed if indicator 21 is OFF. The operations between lines 1 and 2 are executed until the X element of TBL is less than zero or until indicator 44 is turned OFF (see line 2). In these cases, execution resumes with the operation following line 2.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________

(1)       C  N21       TBL,X     DOULTO
          C                      MOVEANAME,X    CHAR
          C                      Z-ADD1         Y
          C                        .
          C                        .
(2)       C   44                 END
```

Figure 8-17. Using the DOULT Operation

DOWxx

This **structured programming** operation (DO While) performs the block of operations (Do Block) immediately following it as long as Factor 1 and Factor 2 meet the criteria specified by xx.

Table 8-10. DOWxx Operations

Enter this operation:	To perform the Do Block while Factor 1 is:
DOWEQ	Equal to Factor 2.
DOWGE	Greater than or equal to Factor 2.
DOWGT	Greater than Factor 2.
DOWLE	Less than or equal to Factor 2.
DOWLT	Less than Factor 2.
DOWNE	Not equal to Factor 2.

End the Do Block of operations with an END operation or an ELSE if the IF operation is also used. (See the DOUxx operation for a description of Do Blocks.) Do not split a Do Block between detail, total, or subroutine operations. You may branch into or out of a Do Block, but be sure that you understand what the results will be. Do Blocks can be nested; that is, a Do Block can be contained within another Do Block as shown below (the maximum number of levels of nesting is 100).

To conditionally execute the DOWxx operation, enter one or more indicators in the Indicators Field (columns 9-17). You can prematurely end execution of a Do Block by entering one or more indicators in the Indicators Field of the END operation. When the indicator conditions are no longer satisfied, control skips to the operation following END.

You can enter a field name or a literal in the Factor 1 and Factor 2 Fields. Numeric fields are aligned by decimal point before they are compared. Shorter fields are padded with zeros (to the left and right) to make them the same size. Blanks in numeric fields are treated as zeros. The maximum numeric field length is 15 digits. Alphanumeric fields are aligned starting with their high-order (leftmost) characters. Shorter fields are padded with blanks (on the right) to make them the same size. If you specified an alternate collating sequence, it is used.

You can compare a numeric field to an alphanumeric field. The numeric field is temporarily converted to alphanumeric format and the two fields are compared as if they both were alphanumeric. The numeric field is not aligned by decimal point before the compare. Negative signs are ignored. For example, the number -123.45

becomes "12345". To avoid problems when comparing the numeric field, make sure it does not have decimal places, is the same length and is not negative. When you compare a numeric field to an alphanumeric field, you see this compiler message:

```
9016I NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE
```

Do not use the Result Field or the Resulting Indicators Field (columns 54-59).

Example

Figure 8-18 shows how to use the DOWxx operation. The DOWNE operation in line 1 executes the operations between it and the END operation (line 2) as long as the Y element of the array CHAR is not equal to blanks.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ____________________________________________________________

(1)       C           CHAR,Y    DOWNE" "
          C                       .
          C                       .
(2)       C                     END
```

Figure 8-18. Using the DOWNE Operation

DSPLM

This display operation displays messages on the terminal from a User Message Catalog. You can also use it to display a field on the terminal and accept changes to it from the user. If you enable the function keys (f1 through f8) using the SET operation, DSPLM automatically turns the function key indicators ON or OFF when the user presses the function keys on the keyboard.

When you run a program containing DSPLM from a job file, data is read and displayed using the system console. When running in session mode, DSPLM uses the user terminal.

Use the Factor 1 and Result Fields in one of the two ways shown below. These fields determine how DSPLM works.

- To display a message from the User Message Catalog (only), enter the identification number of the message in the Factor 1 Field.
- To display a message from the User Message Catalog and to display a field in the program, enter the identification number of the message in the Factor 1 Field and the field name in the Result Field. The user at the terminal can update the Result Field data and also use the function keys.

When DSPLM is executed for method 2 above, the Factor 1 Field is displayed on one line and the Result Field on the next. RPG pauses to let the user update the data displayed for the Result Field. To change the field, the user enters new data directly underneath the displayed data, then presses RETURN. If there are no changes to the field, the user types an asterisk (*) followed by RETURN or presses RETURN by itself. If the field is displayed on the operator's console, the operator must respond using the REPLY console command (see the *MPE/iX Commands Reference Manual*). The operator does not have to enter leading zeros for numeric fields or trailing blanks for alphanumeric fields. RPG automatically aligns the fields and pads them with these characters. For the user or operator to use the function keys, you must have enabled them previously using the SET operation.

When entering the Factor 1 Field, specify the identification number of the message in the User Message Catalog. Ensure that the actual message text does not exceed 249 characters. (See the MSG operation for more information about message identification numbers.)

You may enter the name of the file in the Factor 2 Field, although it is not used. It is provided for compatibility with other implementations of RPG.

Example Conventions

When you use the Result Field, enter a field name into it. If the field is alphanumeric, the maximum number of characters it can contain is 256. If it is numeric, the maximum number of digits is 15. If you're using the system console, the field must not exceed 31 characters for input and 56 characters for output. If you're using a user terminal, the field must not exceed the terminal record length defined at system configuration. You can enter indicators in the Resulting Indicators Field (columns 54-59). They are turned ON or OFF according to the final contents of the Result Field.

You can use the Header Specification to alter the way data is displayed on the terminal and read from it. The DSPLY Options Field (column 48) lets you suppress the display of "DSPLY". It also lets you determine whether the user updates the Result Field on the same line where it is displayed, or on the next line.

Example

The code in figure 8-20 displays instructions to a user on how to use the function keys to print reports. The instructions come from a User Message Catalog and are displayed by the DSPLM operations starting in line 1. (Figure 8-19 lists the messages in the User Message Catalog that are used for this example.) The example also shows SETON operations (starting at line 3) which use the function key indicators turned ON by the DSPLM operations.

```
      $SET 1
      0001Product Sales Analysis - F1 for Current Month Detail
      0002                       - F2 for Current Month Summary
      0003                       - F3 for Year-to-date Detail
      0004                       - F4 for Year-to-date Summary
      0005                       - F5 for Comparative Detail
      0006                       - F6 for Comparative Summary
      0007                       - F7 for Month-end Processing
      0008                       - F8 for Year-end Processing
      0009Use the function keys to select the report to print
```

Figure 8-19. Sample Message Set in a User Message Catalog File

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

     E                    KEYLBL  1   8 16                FUNC KEY LABELS

     C                     SETOF                      U1U2U3
     C                     SETOF                      U4U5U6
     C                     SETOF                      U7U8
(1)  C           "01:1"    DSPLM
     C           "02:1"    DSPLM
     C           "03:1"    DSPLM
     C           "04:1"    DSPLM
     C           "05:1"    DSPLM
     C           "06:1"    DSPLM
     C           "07:1"    DSPLM
     C           "08:1"    DSPLM
     C           KEYLBL    SET                        f@
(2)  C           "09:1"    DSPLM          KEYIN   2
     C                       .
     C                       .
     C                       .
(3)  C   F1                SETON                      U1
     C   F2                SETON                      U2
     C   F3                SETON                      U3
     C   F4                SETON                      U4
     C   F5                SETON                      U5
     C   F6                SETON                      U6
     C   F7                SETON                      U7
     C   F8                SETON                      U8
     C                     SETON                      LR
```

Figure 8-20. Using the DSPLM Operation

DSPLY

This **display** operation displays data on the terminal and reads data from it. You can also use DSPLY to turn the function key indicators ON or OFF when the user presses the function keys on the keyboard.

When running in batch mode, DSPLY uses the system console. When running in session mode, it uses the user terminal.

Use the Factor 1 and Result Fields in one of three ways shown below. These fields determine how DSPLY works.

- To display a single data field on the terminal, enter the name of the field, literal, array element, or table to display in the Factor 1 Field.
- To display a data field on the terminal and allow the user to update it, enter the field name in the Result Field and leave the Factor 1 Field blank. If you have already enabled one or more function keys using the SET operation, the function key indicators are automatically turned ON or OFF according to the function keys that the user presses.
- To display two data fields and allow one of them to be updated by the user, enter both the Factor 1 and Result Fields. If you have already enabled the functions keys using the SET operation, the function key indicators are automatically turned ON or OFF depending on the function keys that the user presses.

When DSPLY is executed for methods 2 and 3 above, the Factor 1 Field (if used) is displayed on one line and the Result Field on the next line. RPG pauses to let the user update the data displayed for the Result Field. To change the field, the user enters new data directly underneath the displayed data, then presses RETURN. If there are no changes to the field, the user types an asterisk (*) followed by RETURN or presses RETURN by itself. If the field is displayed on the operator's console, the operator must respond using the REPLY console command (see the *MPE/iX Commands Reference Manual*). The user does not have to enter leading zeros for numeric fields or trailing blanks for alphanumeric fields. RPG automatically aligns the fields and pads them with these characters. You can also use the function keys if you have previously enabled them with the SET operation. Specify a result field at least two characters long if you use function keys.

When entering the Factor 1 Field, specify the name of a field, literal, array element, or table to display. If you enter a table name, the element found by the last LOKUP operation is displayed. If an array is named in the DS line of an Input Specification, and the array is named in factor 1 without an index, the entire array is displayed, within limits. The data in Factor 1 cannot exceed 249 characters. (Seven characters of this are used for the word DSPLY plus two blanks.) If the data exceeds the terminal line length, it is displayed on successive lines. (Data displayed on the system console is limited to 56 characters per line.)

You may enter the name of the file in the Factor 2 Field although it is not used. It is provided for compatibility with other implementations of RPG.

When you use the Result Field, enter a field name in it. If the field is alphanumeric, the maximum number of characters it can contain is 256. If it is numeric, the maximum number of digits is 15. If you're using the system console, the field must not exceed 56 characters. If you're using a user terminal, the field must not exceed the terminal record length defined at system configuration. You can enter indicators in the Resulting Indicators Field (columns 54-59). They are turned ON or OFF according to the final contents of the Result Field.

You can use the Header Specification to alter the way data is displayed on the terminal and read from it. The DSPLY Options Field (column 48) lets you suppress the display of "DSPLY". It also lets you determine whether the user updates the Result Field on the same line where it is displayed, or on the next line.

Example

Figure 8-21 illustrates the three ways to use DSPLY. The first line (line 1) displays the contents of the field MESSAGE. If MESSAGE contains the characters RE-RUN PROGRAM WITH FILE 2, the following line appears on the terminal:

```
DSPLY  RE-RUN PROGRAM WITH FILE 2
```

Line 2 shows how to display the contents of the field FIELD1 and allow the user to change it. If FIELD1 contains 0023, it is displayed as follows (RPG displays the field, then pauses for user input):

```
0023
```

If the operator enters 43 and presses RETURN, 43 is placed in FIELD1, and RPG continues with the next operation.

Line 3 shows how to display two fields and accept changes for the second one. The fields FIELD2 and FIELD3 are displayed on separate lines. If FIELD2 contains ENTER NEW ID NO; PRESENT NO IS and FIELD3 contains 32, these lines are displayed on the terminal (RPG displays the fields, then pauses for user input):

```
DSPLY ENTER NEW ID NO; PRESENT NO IS
32
```

If the operator enters 5, and presses RETURN, 5 is placed in FIELD3 and RPG continues with the next operation.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------------

(1)       C           MESSAGE   DSPLY
          C                       .
          C                       .
(2)       C                     DSPLY          FIELD1
          C                       .
          C                       .
(3)       C           FIELD2    DSPLY          FIELD3
```

Figure 8-21. Using the DSPLY Operation

ELSE

This **structured programming** operation is used with the IF operation. It marks the beginning of the operations that are performed when the comparison criteria of the IF test are not met.

Enter ELSE in the Operation Field and leave all other fields on the specification blank.

Example

Figure 8-22 shows how to use the ELSE operation. The operation in line 2 is executed when the field CODE is equal to zero (line 1). If CODE is not equal to zero the lines between ELSE (line 3) and END (line 4) are executed.

```
      1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
_____________________________________________________________________

1     C           CODE      IFEQ 0
2     C                     MOVE CODE      CODE2
3     C                     ELSE
      C                     SUB  SHIFT     CODE2         2121
      C  N21                Z-ADD0         X
      C                       .
      C                       .
4     C                     END
```

Figure 8-22. Using the ELSE Operation

END

This **structured programming** operation marks the end of a Do Block of operations. (See the IFxx or DOUxx operations for examples of Do Blocks.) When used with IFxx, ELSE and CASxx, it terminates execution of the Do Block. When used with the DO, DOUxx, or DOWxx operations, it transfers control to the beginning of the Do Block.

You can prematurely end execution of a Do Block by entering one or more indicators in the Indicators Field (columns 9-17) of the END operation. When the indicator conditions are no longer satisfied, control skips to the operation following END.

When using the END operation with the DO operation, enter a numeric value in Factor 2 to specify the loop increment. If no value is specified, the default increment is 1. You cannot specify an increment for the IF, CASxx, DOUxx, and DOWxx operations.

Example

Figure 8-23 shows how to use the END operation with CASxx operations. The END operation (line 2) follows the last line in the CASxx group, which starts with line 1.

```
      1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________

      C           SELECT    TAG
(1)   C  N20      CODE      CASEQ21        SUB21
      C   20      CODE2     CASGE5         SUB5X
      C                     CAS
(2)   C                     END
```

Figure 8-23. Using the END Operation

ENDSR

This **internal subroutine** operation ends an internal subroutine (see figures 8-7 and 8-26). Execution continues in the main program immediately after the EXSR operation that called the subroutine. If you want to branch to ENDSR within the subroutine (do not branch to it from outside the subroutine), enter a label in the Factor 1 Field. The label can be the same as a field name.

Do not use the Factor 2 and Result Fields and do not enter indicators in the Indicators Field (columns 9-17). Enter just one ENDSR operation per subroutine.

EXCPT

This **file** operation lets you produce output during detail and total calculations. This output is in addition to the records that are normally written at detail time and total time. (Normally, the number of records defined in the Output Specifications are written during each program cycle.) For example, you may want several copies of a form that contains the same heading information.

To condition this operation, enter indicators in the Control Level (columns 7-8) and the Indicators Field (columns 9-17). You can further condition or limit the lines that are written by entering a file name in the Factor 2 Field (see the section which follows titled "Conditioning by File Name") or by entering an EXCPT Name in the Result Field (see the section which follows titled "Conditioning by EXCPT Name"). Leave the other fields on the specification blank.

When you use EXCPT, you must also enter an E into the Type Field (column 15) of the Output Specification that defines the record you want to write. Because the record lines that are written are exceptions to the normal program cycle, they are called exception lines. EXCPT writes all exception lines whose conditioning indicators, file name and EXCPT Name conditions are satisfied. When the exception lines are written, execution resumes with the operation following EXCPT.

Example

Figure 8-24 shows how to use the EXCPT operation to print several copies of a mailing address label. The CUST record beginning in line 1 contains the mailing address of a company's customers. When a name matches the name in the NAMEA field (line 2), indicator 50 is turned ON. If indicator 05 is ON, the EXCPT operation in line 3 is executed. It causes the mailing address label (exception lines) defined by MAILER (line 4) to be printed. Since EXCPT is included in a loop that prints five mailing labels for the customer, exception output is performed five times.

```
          1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

(1)  ICUST    AA
     I                                          1  20 NAME
     I                                         21  30 STREET
     I                                         31  40 CITY
     I                                         41  45 STATE
     I                                         46  50 ZIP

     C                      .
     C                      .
(2)  C           NAMEA     COMP NAME                    50
     C           REPEAT    TAG
(3)  C   50                EXCPT
     C   50      1         ADD  COUNT     COUNT   10
     C   50      COUNT     COMP 5                    60
     C   60                GOTO REPEAT
     C           RES1      ADD  RES2      RES3    52
     C                      .
     C                      .

(4)  OMAILER  E
     O                        NAME      20
     O                        STREET    30
     O                        CITY      40
     O                        STATE     45
     O                        ZIP       50
```

Figure 8-24. Using the EXCPT Operation

Conditioning by File Name

You can write exception lines for a particular file (only) by entering the name of the file in the Factor 2 Field. The exception lines that are written for the file must also satisfy the conditioning indicators and the EXCPT Name conditions.

See figure 8-25 for an example of how to condition by file name.

Conditioning by EXCPT Name

You can write exception lines for a certain record in a file by entering the EXCPT Name for the record in the Result Field. (Note that some implementations of RPG use the Factor 2 Field instead of the Result Field.) Define the record you want to write and enter the EXCPT Name in the Field Name Field (columns 32-37) of that Output Specification.

When you enter an EXCPT Name, follow the naming conventions for field names. Do not use an existing array, data structure, field, file label, subroutine, or table name. You can enter the

same EXCPT Name for more than one record. The maximum number of EXCPT Names that you can use in a program is 245.

When EXCPT is executed, the exception lines are written only if the conditioning indicators are satisfied and if the record belongs to the file named in the Factor 2 Field (if Factor 2 is used).

Example

Figure 8-25 shows how to use file and EXCPT Names to write additional records to the output files LIST and OUTFILE. The EXCPT operation in line 1 writes the output record containing GROUP1 in columns 32-37 (line 4). The EXCPT operation in line 2 writes the output record containing GROUP2 in columns 32-37 (line 5). The EXCPT operation in line 3 writes the output record containing GROUP5 in columns 32-37 (line 6) of the file OUTFILE. In addition, only those exception lines in GROUP5 are written when indicator 81 is ON (and they are conditioned by 81).

```
         1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

1  C                   EXCPT          GROUP1
   C                     .
   C                     .
2  C                   EXCPT          GROUP2
   C                     .
   C                     .
3  C                   EXCPTOUTFILE   GROUP5

4  OLIST    E  3           GROUP1
   O                         .
   O                         .
5  O        E  1           GROUP2
   O                         .
   O                         .
6  OOUTFILE E  2     81    GROUP5
```

Figure 8-25. Using the EXCPT Operation with File and EXCPT Names

EXIT

This **external subroutine** operation executes an external subroutine. When the subroutine finishes, control returns to the next executable operation in sequence.

Enter the external subroutine name in the Factor 2 Field. It can contain up to six characters. Do not use the Factor 1 and Result Fields.

The RPG program can share data with the external subroutine in two ways. If the subroutine is in an XL and will be linked at run time, use the PARM operation to pass data to and from the subroutine. Enter statements using the PARM operation, one for each data element you are passing, immediately after the EXIT operation. If the subroutine is not in an XL, but is linked directly to your program, subroutines can share data in an additional way. Data elements can be made global with the RLABL operation. After a data item is named with the RLABL operation, the data item is accessible to any subroutine executed by an EXIT operation. Subroutines in an XL cannot access the global data items named in the RLABL operation.

Examples

See figure 8-33 for an example of how to use EXIT with PARM. See figure 8-37 for an example of how to use EXIT with RLABL.

The following example shows linking an HP C subroutine SUBSRC directly to an RPG program named RPGSRC:

```
:RPGXL RPGSRC, RPGOBJCT
:CCXL SUBSRC, SUBOBJCT
:LINKEDIT
>LINK FROM=RPGOBJCT, SUBOBJCT;TO=RPGPROG
>EXIT
:RUN RPGPROG
```

The next example shows linking a subroutine that is in an XL. The subroutine must have data items passed by the PARM operation, not the RLABL operation.

```
:CCXL SUBSRC, SUBOBJCT
:LINKEDIT
>BUILDXL SUBXL
>ADDXL FROM=SUBOBJCT;TO=SUBXL
>EXIT
:RPGXLLK RPGSRC, RPGPROG
:RUN RPGPROG; XL="SUBXL"
```

If the XL is not in your current group and account, fully qualify the "XL=" option of the run command.

EXSR

This **internal subroutine** operation starts the execution of an internal subroutine (see figures 8-7 and 8-26) You can enter this operation anywhere in the program. When the subroutine is finished, control returns to the operation following EXSR.

Enter the name of the subroutine to execute in the Factor 2 Field. This name must be defined in the Factor 1 Field of the BEGSR operation in the subroutine. Do not use the the Factor 1 and Result Fields. You can conditionally execute EXSR by entering indicators in the Control Level Field (columns 7-8) and the Indicators Field (columns 9-17).

Example

Figure 8-26 shows two subroutines; one calls another. The subroutine SUBA starts at line 3 and the subroutine SUBB starts at line 5. In the main program, SUBA is called at lines 1 and 2. When SUBA is executing, it calls SUBB. SUBB contains a GOTO operation that branches to the end of that subroutine (line 6) when indicator 01 is ON. When SUBB finishes, control returns to the line following the EXSR operation at line 4.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

     C                    .
     C                    .
(1)  C                   EXSR SUBA
     C                    .
     C                    .
(2)  C                   EXSR SUBA
     C                    .
     C                    .
(3)  CSR         SUBA    BEGSR
     CSR                 .
(4)  CSR                 EXSR SUBB
     CSR                 .
     CSR                 ENDSR
(5)  CSR         SUBB    BEGSR
     CSR                 .
     CSR 01              GOTO TAIL
     CSR                 .
(6)  CSR         TAIL    ENDSR
```

Figure 8-26. Using One Internal Subroutine to Call Another

FNDJW

This **system** operation locates a JCW in the system JCW table and returns its value. This operation uses the system intrinsic FINDJCW.

To use FNDJW, leave the Factor 1 Field blank and enter the name of the JCW you want to locate in the Factor 2 Field. The name can be contained in an alphanumeric variable or literal. It must begin with a letter. In the Result Field, enter the field where you want to store the returned JCW value. The field must be numeric with no decimal positions.

You must enter at least one indicator in the Resulting Indicators Field (columns 54-59). When FNDJW is executed, the indicators are set to indicate the outcome of the operation. If the indicator in the High Subfield is ON, the JCW was not found. If the indicator in the Low Subfield is ON, the JCW name that you entered does not begin with a letter. If the JCW was found and its value returned, the indicator in the Equal Subfield is turned ON.

FNUM

This **system** operation obtains the MPE file number for a file and returns it to the program. FNUM does not use a system intrinsic.

To use FNUM, leave the Factor 1 Field blank and enter the name of the file in the Factor 2 Field. You can use an alphanumeric literal or a field name. The file name must be defined in a File Description Specification. In the Result Field, enter the numeric field where you want to save the returned file number. The field must be numeric with no decimal positions. The file number returned is in packed decimal format. If you pass it to an external subroutine, it is passed in packed decimal format, not integer format. (Many intrinsics require that the file number be in binary format. This means that you must convert the number in the external subroutine before you can use it.)

FORCE

This **file** operation lets you name the file from which the next record is selected for processing. The record is selected at the beginning of the next logic cycle (the FORCE operation is in effect only for that cycle).

You can use FORCE to alter the normal multifile processing sequence. Do not use FORCE for the first record read by the program.

To use the FORCE operation, enter the name of the file you want to process next in the Factor 2 Field. Leave the Factor 1 and Result Fields blank. You can use FORCE at detail time in the main program or in a subroutine. Do not use FORCE at total time. When you force a file that is accessed with a RAF, the RAF record is also accessible. When a forced record is processed, the MR indicator is turned OFF (the forced record is processed as if it has no matching fields). If end-of-file is encountered in a forced file, the next record is chosen according to the normal record selection process.

Example

Figure 8-27 shows how to force the file CFILE. Suppose that a program processes records from three files, AFILE, BFILE, and CFILE. Also assume that they are processed in the same order as they are listed. To read a record from CFILE first, ahead of AFILE, a FORCE operation is used. The operation is only performed when indicator 09 is ON.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________

C   09                FORCECFILE
```

Figure 8-27. Using the FORCE Operation

GOTO

This **branching** operation lets you alter the sequential execution of Calculation Specification operations. You can skip to any TAG operation or, if you're within an internal subroutine, you can skip to the ENDSR operation for that subroutine.

To use GOTO, enter the name (label) of the TAG or ENDSR operation to which you want to skip in the Factor 2 Field. You can enter indicators in the Indicators Field (columns 9-17) to condition this operation. Do not use the Factor 1 and Result Fields.

You can branch within detail-time or total-time operations. You can also branch from detail-time to total-time operations and vice versa. When branching from detail to total operations, be very careful that it is allowed in the RPG logic cycle. Do not branch into or out of an internal subroutine.

See figure 8-44 for examples on how to use the GOTO operation with the TAG operation.

IFxx

This **structured programming** operation performs the associated block of operations (Do Block) immediately following it if Factor 1 and Factor 2 meet the criteria specified by xx.

Table 8-11. IFxx Operations

Enter this operation:	To perform the Do Block if Factor 1 is:
IFEQ	Equal to Factor 2.
IFGE	Greater than or equal to Factor 2.
IFGT	Greater than Factor 2.
IFLE	Less than or equal to Factor 2.
IFLT	Less than Factor 2.
IFNE	Not equal to Factor 2.

End the operations in the Do Block with an END operation. If you're using ELSE with the IF operation, place the END operation after the last operation in the ELSE Do Block. If the IFxx condition is not satisfied, the operations following the ELSE or END operation are executed.

Do not split a Do Block between detail, total, or subroutine operations. You may branch into or out of a Do Block, but be sure that you understand what the results will be. Do Blocks can be nested; that is, a Do Block can be contained within another Do Block as shown below (the maximum number of levels of nesting is 100).

```
Begin Level 1 DO Block      IFEQ          (If equal then do)
                            .
                            .
                            ELSE          (Else do)
                            .
                            .
Begin Level 2 DO Block      DOULT         (Do until less than)
                            .
                            .
Begin Level 3 DO Block      DOWNE         (Do while not equal)
                            .
                            .
End Level 3 DO Block        END
End Level 2 DO Block        END
End Level 1 DO Block        END
```

To conditionally execute the IFxx operation, enter one or more indicators in the Indicators Field (columns 9-17). The indicators condition the execution of the entire set of IF/ELSE/END Do Blocks.

Example Conventions

You can enter a field name or a literal in the Factor 1 and Factor 2 Fields. Numeric fields are aligned by decimal point before they are compared. Shorter fields are padded with zeros (to the left and right) to make them the same size. Blanks in numeric fields are treated as zeros. The maximum numeric field length is 15 digits. Alphanumeric fields are aligned starting with their high-order (leftmost) characters. Shorter fields are padded with blanks (on the right) to make them the same size. If you specified an alternate collating sequence, it is used.

You can compare a numeric field to an alphanumeric field. The numeric field is temporarily converted to alphanumeric format and the two fields are compared as if they both were alphanumeric. The numeric field is not aligned by decimal point before the compare. Negative signs are ignored. For example, the number -123.45 becomes "12345". To avoid problems when comparing the numeric field, make sure it does not have decimal places, is the same length, and is not negative. When you compare a numeric field to an alphanumeric field, you see this compiler message:

```
9016I NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE
```

Do not use the Result Field or the Resulting Indicators Field (columns 54-59).

Example

Figure 8-28 shows how to use the IFxx and the ELSE operations. The IFEQ operation in line 1 executes the MOVE operation in line 2 when the field CODE is equal to zero. If CODE is not equal to zero, the operations in lines 4 through 5 are executed.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
_____________________________________________________________________________

(1)   C           CODE      IFEQ 0
(2)   C                     MOVE CODE      CODE2
(3)   C                     ELSE
(4)   C                     SUB  SHIFT     CODE2         2121
      C  N21                Z-ADD0         X
      C                       .
(5)   C                       .
      C                     END
```

Figure 8-28. Using the IFEQ Operation

INTR

This **system** operation lets you call MPE intrinsics directly from an RPG program. Specify the name of the intrinsic beginning in the Factor 2 field, continuing through the Result Field (columns 33-48) if necessary. You can call only intrinsics whose names are 16 characters long or less.

To pass parameters to the intrinsic, follow the INTR operation statement with an IPARM operation statement for each parameter.

The statements defining the intrinsic call (the INTR statement and its associated IPARM statements) can be placed anywhere in the calculations, but they cannot begin in Detail Calculations and continue in Total Calculations.

If the intrinsic is a function, you can specify a field in Factor 1 to hold the value returned by the intrinsic. Define the field in columns 49-52 if it has not already been defined. If the intrinsic function returns a numeric value, you can specify an alphanumeric field in Factor 1. This field will contain alpha-binary data, which is discussed under the IPARM operation following.

The intrinsic's condition code, indicating the success of the call, is returned in the Resulting Indicators Field (columns 54-59). In general, if the intrinsic call is successful, the indicator in columns 58-59 is turned on; indicators in columns 54-55 (less than) and columns 56-57 (greater than) may indicate the intrinsic call was unsuccessful. Refer to the *MPE/iX Intrinsics Reference Manual* for condition codes returned by each intrinsic.

IPARM

IPARM is a **system** operation that allows the RPG program to pass parameters to an MPE intrinsic specified by the preceding INTR operation. Specify the parameter to be passed in the Result Field of the IPARM operation.

RPG uses the intrinsic mechanism to determine the correct parameter type and calling method. Since RPG stores all numeric fields internally in packed decimal format, RPG converts IPARM parameters to the proper type (for instance binary) before calling the intrinsic. If the parameter is passed by reference, the parameter is converted back to packed decimal format after the intrinsic call. Refer to the *MPE/iX Intrinsics Reference Manual* for more information about intrinsics.

If the parameter is a numeric type, you can specify an alphanumeric field in the Result Field. The data returned will be alpha-binary data (see the discussion on alpha-binary data next). This is for single fields only, not for numeric arrays.

If the intrinsic allows it, and you want to use the default value for a particular parameter, leave the Result Field blank for that IPARM operation. If the intrinsic does not require you to specify all the parameters, you can leave off the parameters after the last required one is defined.

The statements defining the intrinsic call (the INTR statement and its associated IPARM statements) can be placed anywhere in the calculations, but they cannot begin in Detail Calculations and continue in Total Calculations.

Alpha-Binary Data

Even though a parameter for an intrinsic may be defined in the *MPE/iX Intrinsics Reference Manual* as a numeric data type (i.e. integer, double, or real), you do not have to assign a numeric field to hold the data in your RPG program unless you want to use it in an arithmetic calculation. When you assign a numeric field, RPG automatically converts the values from binary format (which the intrinsics use) to packed decimal (which RPG uses).

You can avoid unnecessary data conversions and save processing time by assigning a numeric parameter to an alphanumeric field in your RPG program if you do not need to use the field as a number. The numeric value is stored in the alphanumeric field in its binary bit pattern. This data representation is called alpha-binary because the alphanumeric field stores the binary-coded data. Note that output field editing is not possible with alpha-binary data. Numeric arrays cannot be stored as alpha-binary data.

Alpha-binary storage is useful when numeric data that is returned from one intrinsic is passed directly to another intrinsic without being processed by the RPG program. For example, if you call the CREATE intrinsic, an integer PIN number is returned. This number is used in subsequent calls to the ACTIVATE and KILL intrinsics. It is not used in a numerical calculation, so it does not need to be stored as an RPG numeric field. Specifying the PIN field as alphanumeric allows the integer data to pass directly between intrinsics without conversion.

Alpha-binary data is also useful when intrinsic parameters contain bit fields you need to set or test. With alpha-binary data, you can use the TESTB, BITON, and BITOF operations to manipulate bit fields.

Limitations and Guidelines for Intrinsic Parameters

For intrinsic parameters specified as 16-bit integers (I16 or U16), specify an RPG numeric field of 1 to 5 digits with 0 decimal places. For 32-bit integers, specify a numeric field of 6 to 10 digits with 0 decimal places. RPG/iX does not support 64-bit numeric items because it takes 20 digits to hold an item of this size, and the maximum field size for RPG/iX is 15. This means, for example, that you cannot access item 64 of the FFILEINFO intrinsic to retrieve the 64-bit virtual address of the file.

RPG/iX supports only integral values for floating-point numbers used in intrinsics. For example, the PAUSE intrinsic can be used to pause for 8 seconds, but not for 8.5. Define RPG fields of 6 to 10 digits with 0 decimal places for parameters defined as 32R.

Make sure the value of the numeric field does not exceed the maximum value for the parameter. For example, a 5-digit field can contain a value up to 99,999, but an I16 parameter must stay between -32768 and 32767 to keep from overflowing.

If an intrinsic uses a bit mask for a parameter, you can either pass an RPG numeric field loaded with a number whose binary bit pattern forms the bit mask you want (RPG automatically converts the number into binary), or you can define an alphanumeric field and set the bit pattern with the BITON and BITOF operations. This applies to single fields only, not to arrays.

Define numeric and alphanumeric arrays with an Extension Specification. For numeric arrays, specify zero decimal places.

For intrinsics that accept literals for parameters, first copy the literal into a field or array of the proper type, then use the field or array in the Result Field of the IPARM statement.

RPG/iX does not support the following:

- intrinsics that require passing pointers to a procedure
- user-supplied addresses of items (RPG/iX determines all reference addresses)
- parameters defined as sets

Examples

Following are examples of RPG source segments that use the INTR and IPARM operations to call intrinsics.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
_______________________________________________________________________________


C* NUMCHR := ASCII(NUM, BASE, STR);
C* /* FUNCTION RETURN NUMCHR DEFINED NUMERIC, 4 DIGITS, 0 DEC PLACES */
C*
C           NUMCHR    INTR ASCII             40
C                     IPARM          NUM
C                     IPARM          BASE
C                     IPARM          STR


C* WHO(,CAP);     /* SKIP THE FIRST PARAMETER */
C*                /* ALSO, OPTION EXTENSIBLE  */
C                     INTR WHO
C                     IPARM
C                     IPARM          CAP


C* BIN := BINARY(ASCI, LEN);   /* SETS CONDITION CODE */
C*
C           BIN       INTR BINARY               010203
C                     IPARM          ASCI
C                     IPARM          LEN


$ALIAS INTNAM = BINARY
.
.
.
C           BIN       INTR INTNAM               010203
C                     IPARM          ASCI
C                     IPARM          LEN
```

Using the INTR and IPARM operations

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

H
FOUTPUT  O   F      80            $STDLST
E                    DATERR      2  5 0
E                    CMND        6  1
E                    DATBUF     28  1
C                     MOVE *ZERO     DATE    50
C                     SETON                     22
C           DATE      INTR CALENDAR
C           DATE      DSPLY
C                     Z-ADD0         YEAR    50
C                     Z-ADD0         MONTH   50
C                     Z-ADD0         DAY     50
C                     Z-ADD0         WKDAY   50
C                     INTR ALMANAC
C                     IPARM          DATE
C                     IPARM          DATERR
C                     IPARM          YEAR
C                     IPARM          MONTH
C                     IPARM          DAY
C                     IPARM          WKDAY
C           DATE      DSPLY
C           DATERR,1  DSPLY
C           YEAR      DSPLY
C           MONTH     DSPLY
C           DAY       DSPLY
C           WKDAY     DSPLY
C                     MOVEA"LISTF "  CMND
C                     BITOF"01234567"CMND,6
C                     BITON"457"     CMND,6
C                     Z-ADD0         CMERR   50
C                     Z-ADD0         PARM    50
C   22                INTR COMMAND
C                     IPARM          CMND
C                     IPARM          CMERR
C                     IPARM          PARM
C           CMERR     DSPLY
C                     INTR DATELINE
C                     IPARM          DATBUF
C                     SETON                     LR
OOUTPUT  D
O                         DATBUF    28
```

Using the INTR and IPARM operations (continued)

LOCK, UNLCK

These **file** operations let you conditionally or unconditionally lock and unlock TurboIMAGE, KSAM, and MPE files. You can lock TurboIMAGE files at the database, data set, or record level. You can lock KSAM and MPE files at the file level. (For all of these file types, MPE/iX performs page level locking automatically.)

Unconditionally locking a file means that the process is suspended, if necessary, until the database, data set, record, or file is unlocked by another process which has it locked. The interval during which the process is suspended can seriously degrade performance in an interactive processing environment. To avoid this, use conditional locking.

Conditionally locking a database, data set, record, or file means that if the entity is already locked by another process, the lock fails and a resulting indicator is turned ON. The process that performed the lock continues with the next operation.

For the LOCK and UNLCK operations to be allowed for shared files, the appropriate locking facility must be enabled. For TurboIMAGE files, enter a B, S, 1, 9, or L (L is recommended) in the Open Mode Field (column 66) in the Database Name (IMAGE) line of the File Description Continuation line. For KSAM and MPE files, you enable locking by specifying either LOCK or NOLOCK in the Option Type Field (column 54-59) of a File Description Continuation line for the file.

Note

You can use LOCK and UNLCK even though you use a locking mode that causes RPG to automatically lock and unlock the file for you. However, you should exercise extreme caution when doing this. You must ensure that your manual locking does not interfere with RPG automatic locking.

Table 8-12 summarizes the type of locks that you can perform and the fields that you must enter in the Calculation Specification to accomplish them.

Table 8-12. Calculation Specification Fields Used with LOCK and UNLCK

File and Level of Lock	Factor 1	Operation	Factor 2	Result Field	Field Length
TurboIMAGE Database	blank	LOCK/UNLCK	filename	database	1-256
TurboIMAGE Data set	blank	LOCK/UNLCK	filename	blank	blank
TurboIMAGE Record	key value	LOCK/UNLCK	filename	blank	blank
KSAM file	blank	LOCK/UNLCK	filename	blank	blank
MPE file	blank	LOCK/UNLCK	filename	blank	blank

TurboIMAGE Files

The following items discuss how to lock TurboIMAGE files at the database, data set, and data set record level.

- Locking and Unlocking a TurboIMAGE database.

  Use LOCK to lock the database and UNLCK to unlock it. Leave the Factor 1 Field blank. Enter the database file name in the Factor 2 Field. This is the name defined in File Name Field (columns 7-13) of the File Description Specification. Enter the actual database name

(not enclosed in quotes) in the Result Field. This is the name entered in columns 60-65 of the Database Name (IMAGE) line of the File Description Specification Continuation line. Enter a number from 1 to 256 in the Field Length Field (columns 49-51). This number is required for compatibility with other RPG implementations but is not used.

Enter an indicator in the High Resulting Indicators Field (columns 54-55) to perform conditional locking. Leave this field blank to specify unconditional locking. Enter either or both a Low Resulting indicator and an Equal Resulting indicator. When LOCK is executed, the Equal Resulting indicator is turned ON if the operation is successful. Table 8-13 lists the resulting indicators and the conditions that cause them to be turned ON. Since the TurboIMAGE subsystem actually performs the locks, the TurboIMAGE status code is shown also. (For further information about the TurboIMAGE status codes, see the *TurboIMAGE/iX Database Management System* manual.)

- Locking and Unlocking a TurboIMAGE data set.

  Use LOCK to lock the data set and UNLCK to unlock it. (UNLCK actually unlocks the entire database, releasing all previous locks for it.) Leave the Factor 1 Field blank. Enter the database file name in the Factor 2 Field. This is the name defined in File Name Field (columns 7-13) of the File Description Specification. Leave the Result Field blank.

  Enter an indicator in the High Resulting Indicators Field (columns 54-55) to perform conditional locking. Leave this field blank to specify unconditional locking. Enter either or both a Low Resulting indicator and an Equal Resulting indicator. When LOCK and UNLCK are executed, the Equal Resulting indicator is turned ON if the operation was successful. Table 8-13 lists the resulting indicators and the conditions that cause them to be turned ON. Since the TurboIMAGE subsystem actually performs the locks, the TurboIMAGE status code is shown also. (For further information about the TurboIMAGE status codes, see the *TurboIMAGE/iX Database Management System* manual.)

- Locking and Unlocking a TurboIMAGE data set record.

  Use LOCK to lock the data set record and UNLCK to unlock it. (If all previous locks for the database are released, UNLCK unlocks the entire database.) Enter the key for the record you want to lock in the Factor 1 Field. This is the field defined as the key in the Item Name (ITEM) line (columns 60-74) of the File Description Specification Continuation line. Enter the database file name in the Factor 2 Field. This is the name defined in File Name Field (columns 7-13) of the File Description Specification. Leave the Result Field blank.

  Enter an indicator in the High Resulting Indicators Field (columns 54-55) to perform conditional locking. Leave this field blank to specify unconditional locking. Enter either or both a Low Resulting indicator and an Equal Resulting indicator. When LOCK and UNLCK are executed, the Equal Resulting indicator is turned ON if the operation was successful. Table 8-13 lists the resulting indicators and the conditions that cause them to be turned ON. Since the TurboIMAGE subsystem actually performs the locks, the TurboIMAGE status code is shown also. (For further information about the TurboIMAGE status codes, see the *TurboIMAGE/iX Database Management System* manual.)

Table 8-13.
How Resulting Indicators Are Set For LOCK/UNLCK (TurboIMAGE Files)

Resulting Indicator Turned ON	LOCK Database	LOCK Data Set	LOCK Record	UNLCK
High (conditional locking only).	Database locked or contains locks (TurboIMAGE status 20).	Database locked or contains locks (TurboIMAGE status 20). Data set locked by another process (TurboIMAGE status 22). Entries locked within data set (TurboIMAGE status 23).	Database locked or contains locks (TurboIMAGE status 20). Data set locked by another process (TurboIMAGE status 22). Entries locked within data set (TurboIMAGE status 23). Item conflicts with current locks (TurboIMAGE status 24). Entries already locked (TurboIMAGE status 25).	Exceptional error (TurboIMAGE status > 0).
Low	File system or memory manager failure (TurboIMAGE status < 0).	See error messages in the TurboIMAGE Reference Manual.	See error messages in the TurboIMAGE Reference Manual.	File system or memory manager failure (TurboIMAGE status - < 0).
Equal	Request granted (TurboIMAGE status 0).	Request granted (TurboIMAGE status 0).	Request granted (TurboIMAGE status 0).	Request granted (TurboIMAGE status 0).
None of the above indicators are turned ON.	A condition not listed above was encountered (TurboIMAGE status not listed above).	A condition not listed above was encountered (TurboIMAGE status not listed above).	A condition not listed above was encountered (TurboIMAGE status not listed above).	Does not apply; at least one resulting indicator is ON.

Example Conventions

KSAM Files

You can lock KSAM files at the file level. To monitor the locking and unlocking of KSAM files from a terminal, use the KSAM utility, KSAMUTIL. For complete information about KSAMUTIL, see the *KSAM/3000 Reference Manual*.

You can lock a KSAM file at the file level only if you have specified LOCK or NOLOCK in the Option Type Field (columns 54-59) of a File Description Specification Continuation line for the file. Use LOCK to lock the file and UNLCK to unlock it. Leave the Factor 1 Field blank. Enter the file name for the KSAM file in the Factor 2 Field. This is the name defined in File Name Field (columns 7-13) of the File Description Specification. Leave the Result Field blank.

Enter an indicator in the High Resulting Indicators Field (columns 54-55) to perform conditional locking. Leave this field blank to specify unconditional locking. Enter either or both a Low Resulting indicator and an Equal Resulting indicator. When LOCK is executed, the indicators that you enter are turned ON to indicate if the operation was successful. Table 8-14 lists the resulting indicators and the conditions that cause them to be turned ON. Since the KSAM subsystem actually performs the locks, the KSAM condition code is shown also. (For further information about the KSAM condition codes, see the *KSAM/3000 Reference Manual*.)

MPE Files

Use LOCK to lock an MPE file and UNLCK to unlock it. Leave the Factor 1 Field blank. Enter the file name for the MPE file in the Factor 2 Field. This is the name defined in File Name Field (columns 7-13) of the File Description Specification. Leave the Result Field blank.

Enter an indicator in the High Resulting Indicators Field (columns 54-55) to perform conditional locking. Leave this field blank to specify unconditional locking. Enter either or both a Low Resulting indicator and an Equal Resulting indicator. When LOCK is executed, the indicators that you enter are turned ON to indicate if the operation was successful. Table 8-14 lists the resulting indicators and the conditions that cause them to be turned ON.

Table 8-14.
How Resulting Indicators Are Set For LOCK/UNLCK
(KSAM and MPE Files)

Resulting Indicator Turned ON	LOCK	UNLCK
High (conditional locking only)	Locked by another process (KSAM condition code >).	Not already locked (KSAM condition code >).
Low	Not opened with dynamic locking facility enabled or need MR capability (KSAM condition code <).	Not opened with dynamic locking facility enabled or need MR capability (KSAM condition code <).
Equal	Request granted (KSAM condition code =).	Request granted (KSAM condition code =).

LOKUP

This **table and array** operation retrieves an element from a table or array and makes it available for use in subsequent operations. The element is retrieved when it satisfies the search criteria that you enter.

Enter the search argument (the element you're looking for) in the Factor 1 Field. It can be an alphanumeric or numeric constant, a field name, an array element, or a table name. Enter the table or array to search in the Factor 2 Field. Be sure that Factor 1 and Factor 2 have the same length. They do not have to contain the same number of decimal places.

Enter at least one but not more than two indicators in the Resulting Indicators Field (columns 54-59). The indicators define the search criteria and reveal the results of the search. Enter an indicator in the Equal Subfield (columns 58-59) to search for an element in the table or array that is equal to Factor 1 (if there is more than one equal element, the first one is chosen). Enter an indicator in the Low Subfield (columns 56-57) to search for the element that is nearest to, but less than Factor 1. Enter an indicator in the High Subfield (columns 54-55) to search for the element that is nearest to, but higher than Factor 1. You cannot specify both the High and the Low Subfield in the same LOKUP operation. When the search is successful, the indicator that you enter is turned ON. For example, if the indicator 05 is entered in the High Subfield, a search for the nearest element that is higher than Factor 1 takes place and if an element is found, indicator 5 is turned ON.

You can search for a table or array element that is greater than or equal to Factor 1, or less than or equal to Factor 1. Enter indicators in both the Equal and Low Subfields or the Equal and High Subfields. Either condition satisfies the search (equal has precedence) and the indicator associated with that condition is turned ON. When you enter an indicator in the Low Subfield, High Subfield, the Equal and Low Subfields, or the Equal and High Subfields, make sure the table or array is in ascending or descending sequence, or you may not retrieve the element you expect. (You can use the SORTA operation to sequence arrays.)

Searching A Table

To search a table that has no alternating table, use the Factor 1, Factor 2, and Resulting Indicator Fields. To condition the LOKUP operation, enter indicators in the Control Level (columns 7-8) or Indicators Fields (columns 9-17), or both.

When an element is found, use the table name in subsequent operations to reference it. The table name references the element that was found until another LOKUP operation is performed or until you use the table name in the Result Field of another operation. If an element is not found, the table name references the element found by the previous LOKUP operation.

Example

Figure 8-29 shows how to search a table. Suppose that the field ENTRY1 (line 1) contains the value 300. The LOKUP operation searches the table TABLEA for 300. If it is found, indicator 10 is turned ON. The ADD operation in line 2 is executed (since indicator 10 is turned ON). It adds 300 (the element found by the LOKUP in line 1) to 100, and replaces the element containing 300 in TABLEA with the result (400).

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------

(1)       C           ENTRY1    LOKUPTABLEA                    10
(2)       C   10      TABLEA    ADD  100       TABLEA
```

Figure 8-29. Searching a Table

Searching Alternating Tables

This section explains how to search a table along with its alternating table. The alternating table is not actually searched, although you name it in the LOKUP operation. However, when an element is found in a table, its corresponding alternating table values become available.

To search a table and its alternating table, enter the table in the Factor 2 Field and the alternating table in the Result Field. Use the Factor 1 and Resulting Indicator Fields as you normally do, but do not enter an array element in the Factor 1 Field. The table and its alternating table should be the same length. If not, the search stops at the end of the shorter table. Once an element is found in the table, use the table name to reference it or use the alternating table name to reference elements in the alternating table.

Example

Figure 8-30 shows how to search the table TABLEB and its alternating table TABLEC. The LOKUP operation in line 1 searches TABLEB for the element that matches the field ENTRY2. If an element is found, indicator 20 is turned ON. The MULT operation in line 2 is executed when indicator 20 is turned ON. The element in TABLEC (found by the LOKUP operation) is multiplied by 20 and the result is placed in the field STORA.

```
          1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------------------------

(1)       C           ENTRY2    LOKUPTABLEB    TABLEC          20
(2)       C   20      TABLEC    MULT 20        STORA
```

Figure 8-30. Searching Alternate Tables

Searching An Array

To search an array, enter the search argument in Factor 1 and use the Resulting Indicators Fields as you would in searching a table. Do not use the Result Field. Unlike with tables, the last element in an array cannot be referenced by using the array name alone; it can, however, be referenced as noted below.

Enter the name of the array to be searched in Factor 2. If you use the array name alone, the search begins at the first element of the array. If you use an array name with an index (the index can be a field or a literal), the search begins with the element specified by the index. When the index is a field and the search is successful, the number of the matching array element is placed in the field; otherwise the field is set to 1.

Example

Figure 8-31 shows how to search an array. The LOKUP operation in line 1 searches the array ARR1 (beginning with the element specified by INDX) for an element that matches the field ENTRY3. Assuming that a match is found on the seventy-third element of ARR1, 73 is placed in INDX and indicator 30 is turned ON. The MOVE operation in line 2 is executed since indicator 30 is turned ON. It moves the ARR1 element (found in line 1) to the field STORB.

```
          1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
          ____________________________________________________________________________

(1)       C           ENTRY3    LOKUPARR1,INDX                30
(2)       C   30                MOVE ARR1,INDX STORB
```

Figure 8-31. Searching an Array

MHHZO

This **move zone** operation (Move High to High ZOne) moves the zone bits of the high-order position of Factor 2 to the high-order position of the Result Field. Both fields must be alphanumeric.

The following illustration shows how this operation works. Z stands for the zone bits of each character and D stands for the digit portion. The shaded area shows the affected characters in the Factor 2 and Result Fields.

```
Factor 2 Field:                                  Result Field:

 Z D Z D Z D Z D   (alphanumeric)   ------>    Z D Z D Z D Z D   (alphanumeric)
|---|---|---|---|                              |---|---|---|---|
```

MHLZO

This **move zone** operation (Move High to Low ZOne) moves the zone bits from the leftmost position of Factor 2 to the rightmost position of the Result Field. Factor 2 must be alphanumeric. The Result Field can be either alphanumeric or numeric.

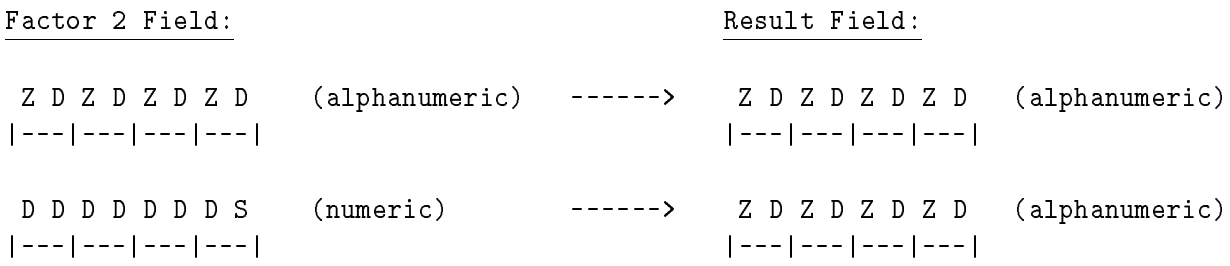
The illustrations below show how MHLZO works. The letter Z stands for the zone portion of the character, D stands for the digit portion, and S stands for the sign portion. The shaded areas show the affected characters in the Factor 2 and Result Fields.

```
Factor 2 Field:                                  Result Field:

 Z D Z D Z D Z D   (alphanumeric)   ------>    Z D Z D Z D Z D    (alphanumeric)
|---|---|---|---|                              |---|---|---|---|

 Z D Z D Z D Z D   (alphanumeric)   ------>    D D D D D D D S    (numeric)
|---|---|---|---|                              |---|---|---|---|
```

MLHZO

This **move zone** operation (Move Low to High ZOne) moves the zone portion of the low-order position of Factor 2 to the high-order position of the Result Field. Factor 2 can be alphanumeric or numeric. The Result Field must be alphanumeric.

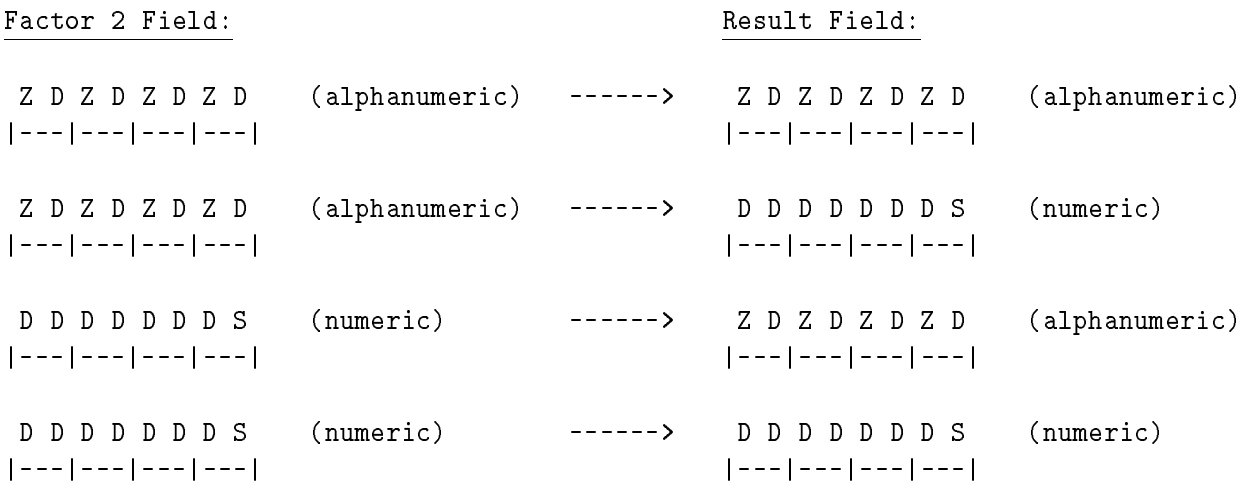
The illustrations below show how MLHZO works. The letter Z stands for the zone portion of the character, D stands for the digit portion, and S stands for the sign portion. The shaded areas show the affected characters in the Factor 2 and Result Fields.



MLLZO

This **move zone** operation (Move Low to Low ZOne) moves the zone bits from the low-order position of Factor 2 to the high-order position of the Result Field. The fields can be either alphanumeric or numeric.

The illustrations below show how MLLZO works. The letter Z stands for the zone portion of the character, D stands for the digit portion, and S stands for the sign portion. The shaded areas show the affected characters in the Factor 2 and Result Fields.



Example

Table 8-15 gives examples of how MHHZO, MHLZO, MLHZO, and MLLZO work. In these examples, "—" indicates that the type of move does not apply and ⊔ stands for a blank.

Table 8-15. Move Zone Operations

Type of Move	Factor 2 Contents	Result Field before Move	Result Field after Move: MHHZO	MHLZO	MLHZO	MLLZO
Alphanumeric to alphanumeric	AH5SR	S51T	B51T S51L	S51C	K51T	
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	

MOVE

This **move** operation moves characters from Factor 2 to the Result Field. Characters are moved beginning at the rightmost (low-order) position, continuing to the leftmost (high-order) position.

If Factor 2 is longer than the Result Field, excess characters are not moved. If Factor 2 is shorter that the Result field, excess characters in the Result Field remain unchanged.

You can use this operation to convert an alphanumeric field or constant to packed decimal format. Each alphanumeric character is assumed to be a digit. The zone portion of the low-order alphanumeric character is the sign and it is stored in the low-order position of the Result Field. Zone bits for other characters are stripped and the digits are compressed into packed decimal format. Blanks are converted to zeros.

You can also use this operation to convert a packed decimal field to an alphanumeric field. The low-order position of Factor 2 contains the low-order digit and sign.

To move data to an array element, enter the array name with index in the Result Field. To duplicate data in every element of an array, enter the array name without an index in the Result Field.

You can move one or more zeros or blanks to the Result Field using the figurative constants *BLANK, *BLANKS, *ZERO, or *ZEROS.

Table 8-16 gives examples of the combinations of numeric and alphanumeric MOVEs that you can perform.

Table 8-16. MOVE Operation Examples

Type of MOVE	Factor 2 Contents	Result Field Contents before MOVE (numeric fields shown in hexadecimal format)	Result Field Contents after MOVE (numeric fields shown in hexadecimal format)
Alphanumeric to alphanumeric	ABC23	1234567	12ABC23
Alphanumeric to numeric	1232E	1234567C	1212325C
Alphanumeric to alphanumeric	ABC23DEFG	1234567	C23DEFG
Alphanumeric to numeric	12323456P	7654321C	3234567D
Alphanumeric to alphanumeric	ABC23DE	1234567	ABC23DE
Alphanumeric to numeric	3212345	1234567D	3212345F
Numeric to alphanumeric Numeric to alphanumeric	1234567F 1234567C	ABC23DE AB12CDF	1234567 123456G
Numeric to numeric	1234567C	8910123D	1234567C

Example

Figure 8-32 shows three ways to use the MOVE operation.

Line 1 moves the value 12492 to each element of the array ARY. If the elements of the ARY array were defined as five characters long or more (the size of the value 12492), then the entire value 12492 is moved into the rightmost positions of each element. If the element size is only 1 character, then only the number 2 is moved to each element. (If you want to move 1 to the first element, 2 to the second element, 4 to the third and so on, use the MOVEA operation.)

Line 2 shows how to fill an array with zeros, regardless of the number or the size of elements.

After the move in line 3, the fifth element of the array contains only blanks. The other elements are unaffected.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------

1         C                   MOVE 12492     ARY
          C                     .
          C                     .
2         C                   MOVE *ZEROS    ARY
          C                     .
          C                     .
3         C                   MOVE *BLANKS   ARY,5
```

Figure 8-32. Using the MOVE Operation

MOVEA

This **move** operation moves the Factor 2 Field to the Result Field. Data is moved beginning at the leftmost (high-order) position, continuing to the rightmost (low-order) position. Both fields must be alphanumeric and at least one of them must be an array. Do not use the same array for both the Factor 2 and Result Fields.

MOVEA lets you:

- Move contiguous elements from one array to contiguous elements of another.
- Move contiguous array elements to a single field.
- Move a single field to several contiguous array elements.

When you enter an array element in Factor 2, it is the first field that is moved. Subsequent elements are moved until the last array element is moved or filled, or until all of the characters in the shorter field are moved. Depending on the length of the Factor 2 and Result Fields, MOVE may end in the middle of a field or array element.

If Factor 2 is longer than the Result Field, excess characters in Factor 2 are not moved. If Factor 2 is shorter than the Result Field, excess characters in the Result Field remain unchanged.

When you use a figurative constant (*BLANK, *BLANKS, *ZERO, or *ZEROS) in Factor 2 and an array name in the Result Field, the array is filled with blanks or zeros. If the array is indexed, the operation begins with the element corresponding to the index number and continues to the end of the array.

MOVEL

This **move** operation moves Factor 2 to the Result Field. Data is moved beginning at the leftmost (high-order) position, continuing to the rightmost (low-order) position.

You can use the figurative constants *BLANK(S) and *ZERO(S) in the Factor 2 Field. If Factor 2 is longer than the Result Field, the excess low-order characters are not moved. If Factor 2 is shorter than the Result Field, excess characters in the Result Field remain unchanged.

You can use this operation to convert an alphanumeric field or constant to a number by moving it to a numeric field. The alphanumeric field must contain ASCII digits (except in the low-order position). The digit portion of each character is converted to the corresponding number before it is moved. When the number of characters in the alphanumeric field equals or exceeds the number of digits in the Result Field, the sign of the alphanumeric field is moved to the low-order position of the Result Field. Zones are stripped from the other characters, and the numeric equivalents are saved in packed decimal format.

You can also use MOVEL to convert a numeric field or a constant to an alphanumeric value by moving it to a alphanumeric field. Each digit is converted to its corresponding ASCII character. If the entire field is moved and the sign is negative, the rightmost zone and digit bits are converted to one character.

MSG

This **display** operation retrieves a message from a User Message Catalog file (created by the Native Language Support, NLS or by MAKECAT) and places it in the field that you specify. You may want to use MSG to tailor corporate report headings contained in a User Message Catalog, for example, to fit a particular division's requirements.

The name of the User Message Catalog file is CATALOG and it is assumed to be in your logon group and account. If you want to use another file or if CATALOG is in another group or account, enter an operating system :FILE command to change these values. For example, the command `:FILE CATALOG=MYMSG.PUB.PAYROL` specifies that the file MYMSG is the User Message Catalog and it is located in the PUB.PAYROL group and account. You can use only one User Message Catalog file in a program. As a result, if you're using MSG and the message features of the RPG Screen Interface, (RSI) combine all messages into one file.

To use MSG, enter the identification number of the message in the Factor 1 Field. You can enter a literal or alphanumeric field. The message identification consists of a message number and, optionally, a set number. The message identification has the format,

```
nnnn[:ss]
```

where `nnnn` is the message number and `ss` is the set number. The colon (:) is used to separate the two numbers. If you omit the set number, the first set is used.

In the Result Field, enter the name of the field where you want to store the message. You can use an alphanumeric field or array. If the message exceeds the length of the Result Field, the low-order characters are truncated. The maximum length of a message is 256 bytes.

Enter the indicator in the High Subfield (columns 54-55) that you want turned ON when the message cannot be retrieved. If you do not enter an indicator in this field and the message cannot be retrieved, the H0 indicator is turned ON.

Do not use the Factor 2, Decimal Positions, Half Adjust, and the Low and Equal Subfield Fields.

Example

Four examples of valid message identifications are shown in Table 8-17. See the DSPLM operation for examples of how to code message identifications.

Table 8-17. Valid Message Identifications

Message Identification:	Message Number:	Set Number:
`"12"`	12	1 (default)
`"3."`	3	1 (default)
`"32:6"`	32	6
`"0004:02"`	4	2

MULT

This **arithmetic** operation multiplies Factor 1 by Factor 2 and places the product in the Result Field. Be sure that the Result Field is large enough to hold the largest possible result. Excess digits are truncated. If Factor 1 is blank, the Result Field is multiplied by Factor 2 and the product is placed in the Result Field.

MVR

This **arithmetic** operation moves the remainder from the previous DIV operation to the Result Field. Use MVR immediately following the DIV operation and condition it with the same indicators (see figure 8-15). Do not use the Factor 1 or Factor 2 Fields.

The remainder is stored as a decimal number. When you define the field entered in the Result Field, make sure that it contains the same number of whole number positions as the Factor 2 Field of the DIV operation. If must also contain the number of decimal positions which is the greater of:

- The number of decimal positions in Factor 1 of the DIV operation.
- The sum of the decimal positions in the Factor 2 and Result Fields of the DIV operation.

PARM

This **external subroutine** operation passes data to an external subroutine. PARM is used in conjunction with EXIT. Enter a PARM operation for each data element that you want to pass to the subroutine. Begin the PARM operations immediately after the EXIT operation that calls the subroutine.

Enter the data element to be passed in the Result Field. You can enter a field, table, or array name. Do not enter an indicator. Data elements are passed as byte arrays in the same order you enter them in the PARM operations. Leave the Factor 1 and Factor 2 Fields blank.

Example

Figure 8-33 shows how to use the EXIT operation to call an external subroutine and how to use the PARM operation to exchange information with it.

The RPG program reads two numbers (FLD1 and FLD2) and a six-character string (FLD3) from the terminal. It then calls the external subroutine EXSUB in line 1. EXSUB is written in COBOL and is shown in figure 8-34. It adds the fields passed to it as parameters (FLD1 and FLD2 starting in line 2) and displays the result. The result (RESLT1) along with FLD1 and FLD2 are returned to the RPG program when EXSUB finishes.

The second external subroutine EXSUB2 is then called in line 3. EXSUB2 is written in C and is shown in figure 8-35. The field FLD3 is passed to it as a parameter at line 4. EXSUB2 moves FLD3 to the field RESLT2 and returns to the RPG program. The RPG program writes the input fields (FLD1, FLD2) and the fields created by the external subroutines (RESLT1 and RESLT2) to the output file OUTFLE.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

FTERMIN  IDE V      14
FOUTFLE  O   F      50            DISK

ITERMIN  AA  01   1 C0
I                                        3   80FLD1
I                                        9  140FLD2
I                                       15  20 FLD3

C                     READ TERMIN                    LR
C  NLR                EXIT EXSUB
C                     PARM           FLD1
C                     PARM           FLD2
C                     PARM           RESLT1  70
C  NLR                EXIT EXSUB2
C                     PARM           FLD3
C                     PARM           RESLT2  6

OOUTFLE  D        01
O                                       5 "FLD1="
O                          FLD1        11
O                                      17 "FLD2="
O                          FLD2        23
O                                      31 "RESULT="
O                          RESLT1      38
O                                      44 "FLD3="
O                          RESLT2      50
```

1 2 3 4

Figure 8-33. Using EXIT, PARM and External Subroutines

```
$CONTROL SUBPROGRAM
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  EXSUB.
 ENVIRONMENT DIVISION.
 DATA DIVISION/
 LINKAGE SECTION.
*
*NUMERIC PARAMETER DATA TYPED AS COMP-3
*
 01 FLD-1  PIC S9(6) COMP-3.
 01 FLD-2  PIC S9(6) COMP-3.
 01 RESULT PIC S9(7) COMP-3.
 PROCEDURE DIVISION USING FLD-1 FLD-2 RESULT.
 START-LINK.
      ADD FLD-1 FLD-2 GIVING RESULT.
      DISPLAY "FLD-1= " FLD-1 "  FLD-2 " FLD-2 "  RESULT= " RESULT.
      GOBACK.
```

Figure 8-34. The External Subroutine EXSUB Written in COBOL

```
exsub2 (A,B)
   char *A;
   char *B;
{
   strncpy (b, a, 6);
}
```

Figure 8-35. The External Subroutine EXSUB2 Written in C

PUTJW

This **system** operation locates a JCW in the system JCW table and changes its value. If the JCW does not exist, a new one is created. PUTJW uses the system intrinsic PUTJCW.

To use PUTJW, enter the value to place in the JCW in the Factor 1 Field. You can use a numeric literal or field. It must be no more than 8 digits long with no decimal places. It must contain a number in the range, 0-65,535. If the value is outside of this range, the run-time error "Invalid Numerical Data" is printed. Enter the name of the JCW you want to locate in the Factor 2 Field. If the JCW does not exist, a new one is created with this name. The name can be contained in an alphanumeric variable or literal. It must begin with a letter. Leave the Result Field blank.

You must enter at least one indicator in the Resulting Indicators Field (columns 54-59). When PUTJW is executed, the indicators are set to indicate the outcome of the operation. If the indicator in the High Subfield is ON, there is no more room in the JCW table for the new entry. If the indicator in the Low Subfield is ON, the JCW name that you entered does not begin with a letter. If the JCW was found and altered successfully, or if a new JCW was created, the indicator in the Equal Subfield is turned ON.

READ

This **file** operation reads a record from a sequential demand file (that may or may not reside on a SPECIAL device) or a full procedural file. The record is made available during the present cycle instead of the next one (FORCE makes a record available during the next cycle). READ is similar to CHAIN except that CHAIN processes files randomly.

To use READ, leave the Factor 1 Field blank and enter the name of the demand or full procedural file in the Factor 2 Field. Leave the Result Field blank.

You can enter an indicator in the Equal Subfield (columns 58-59). It is turned ON when end-of-file is encountered. If you do not enter an indicator, the H0 indicator is turned ON. Do not use the High and Low Subfields (columns 54-57). Do not use control level indicators, matching fields, or look-ahead fields for the file. Also, do not specify sequence-checking for the file in the Input Specification.

When READ is executed, the appropriate record-identifying indicators are turned ON and the input fields are made available. Unidentified record types cause run-time errors.

Example

Figure 8-36 shows how to read the demand file SURTAX (see line 1). The file is read only when indicator 07 is ON.

```
     1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________

C                   MOVE TAX2      NETAXP
C            NETAXP COMP 15000                07
1)   C  07          READ SURTAX
```

Figure 8-36. Reading a Demand File

READE

This **file** operation reads the next record from an indexed demand file or a full procedural file. If the record key matches the Factor 1 Field, the record is made available for processing.

To use READE, enter the name of the field that contains the key value in the Factor 1 Field. Enter the name of the demand or full procedural file in the Factor 2 Field. Leave the Result Field blank.

You can enter an indicator in the Equal Subfield (columns 58-59). It is turned ON when the record key matches the Factor 1 Field and when end-of-file is encountered. If you do not enter an indicator, the H0 indicator is turned ON when the record key does not match the Factor 1 Field or when end-of-file is encountered. Do not use the High and Low Subfields (columns 54-57). Do not use control level indicators, matching fields, or look-ahead fields for the file. Also, do not specify sequence-checking for the file in the Input Specification.

When READE is executed, the appropriate record-identifying indicators are turned ON and the input fields are made available. Unidentified record types cause run-time errors.

READP

This **file** operation reads the previous record from a sequential or indexed demand or full procedural file. If the file is indexed, the previous record is read by key (not chronologically).

To use READP, leave the Factor 1 Field blank and enter the name of the demand or full procedural file in the Factor 2 Field. Leave the Result Field blank.

You can enter an indicator in the Equal Subfield (columns 58-59). It is turned ON when the beginning of the file is encountered. (If you do not enter an indicator, the H0 indicator is turned ON) Do not use the High and Low Subfields (columns 54-57). Do not use control level indicators, matching fields, or look-ahead fields for the file. Also, do not specify sequence-checking for the file in the Input Specification.

When READP is executed, the appropriate record-identifying indicators are turned ON and the input fields are made available. Unidentified record types cause run-time errors.

RESET

This **file** operation sets a file to its initial open condition. This operation allows you to process a file more than once without restarting the program.

The file must be specified in the File Specification as an INPUT or UPDATE type with a designation of DEMAND or FULL PROCEDURAL. The file can be an MPE (flat) file, a KSAM file, or a TurboIMAGE file. You can read a file sequentially until its end-of-file indicator is set (or end-of-chain for TurboIMAGE chained sequential files), and then use this indicator to condition the RESET operation to set the file to its initial state.

For MPE and KSAM files, RESET calls the MPE intrinsic FCONTROL with a parameter of 5 (refer to the *MPE/iX Intrinsics Reference Manual*), and for TurboIMAGE files, RESET does a DBCLOSE mode 3 (refer to the *TurboIMAGE/iX Database Management System* manual). For KSAM files, FCONTROL parm 5 resets to the lowest primary key, so this operation is not allowed if you are reading the file chronologically (file specification column 32 is C).

To use RESET, leave the Factor 1 Field blank and enter the name of the file you want to reset in the Factor 2 Field. Leave the Result Field blank. Enter an indicator in the High Subfield (columns 54-55); this is turned ON if the RESET operation fails. Do not use the Low or Equal Subfields (columns 56-59), the decimal positions field (column 52), or the half-adjust field (column 53).

RLABL

This **external subroutine** operation names a field, table, array, or indicator to be passed to an external subroutine. RLABL is used in conjunction with the EXIT operation. Note that you cannot use or set indicators in external procedures such as user-trap routines that are not executed with the EXIT operation.

Enter the field, table, array, or general indicator (01-99) in the Result Field. To pass an indicator, enter it in this format: `INxx`. For instance, indicator 01 is entered as `IN01`. If the program has other external routines that use the field, array, or indicator, define it only once using RLABL. Do not enter the same name used in RLABL operations with a GOTO, TAG, BEGSR, or ENDSR operation elsewhere in the program.

In the external subroutine, declare all fields, tables and arrays (passed from the RPG main program) as external character arrays and all indicators as external integers. The indicators contain zero when they are OFF and one when they are ON. Be sure the names for fields, tables, arrays, and indicators passed to the external subroutine are the same as those in the RPG program.

Example

Figure 8-37 shows how to use the EXIT operation to call an external subroutine and how to use the RLABL operation to exchange information with it.

The program reads two numbers (FLD1 and FLD2) and a six-character string (FLD3) from a disk file. It then calls the external subroutine EXSUB in line 3. EXSUB is written in COBOL and is shown in figure 8-38. It adds the fields passed to it as parameters (FLD1 and FLD2 starting in line 1) and displays the result. It also displays FLD3. If the result of the ADD is zero, indicator 22 is turned on. The result (RESLT1) along with FLD1, FLD2, and indicator 22 are returned to the RPG program when EXSUB finishes. The second external subroutine EXSUB2 is then called in line 4. EXSUB2 is written in C and is shown in figure 8-39. Indicator 22 and the field FLD3 are passed to it as parameters (starting at line 2). Depending on whether indicator 22 is ON or OFF, EXSUB2 moves FLD3 to the field RESLT2 or clears RESLT2 to blanks. It then returns to the RPG program. The RPG program writes the input fields FLD1 and FLD2 and the fields (RESLT1 and RESLT2) created by the external subroutines to the output file OUTFLE.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

FINFILE  IP  F      80            DISK
FOUTFLE  O   F      50            DISK

IINFILE  AA  01   1 C0
I                                        3   80FLD1
I                                        9  140FLD2
I                                       15  20 FLD3
I        NS
C                      RLABL          FLD1
C                      RLABL          FLD2
C                      RLABL          RESLT1  70
C                      RLABL          IN22
C                      RLABL          FLD3
C                      RLABL          RESLT2  6
C                      SETOF                     22
C                      EXIT EXSUB
C                      EXIT EXSUB2
C   22      "MOVED"    DSPLY

OOUTFLE  D        01
O                                       5 "FLD1="
O                           FLD1       11
O                                      17 "FLD2="
O                           FLD2       23
O                                      31 "RESULT="
O                           RESLT1     38
O                                      44 "FLD3="
O                           RESLT2     50
```

1 2 3 4

Figure 8-37. Using EXIT, RLABL, and External Subroutines

```
001000$CONTROL SUBPROGRAM
001100 IDENTIFICATION DIVISION.
001200 PROGRAM-ID.  EXSUB.
001300 ENVIRONMENT DIVISION.
001400 DATA DIVISION.
001500 WORKING-STORAGE SECTION.
001520 01 FLD1 EXTERNAL PIC S9(6) COMP-3.
001530 01 FLD2 EXTERNAL PIC S9(6) COMP-3.
001532 01 RESLT1 EXTERNAL PIC S9(7) COMP-3.
001534 01 FLD3 EXTERNAL PIC X(6).
001540 01 IN22 EXTERNAL PIC S9(9) COMP.
001550    88 IND-22-ON  VALUE 1.
001600 PROCEDURE DIVISION.
001700 START-IT.
001800        ADD FLD1 FLD2 GIVING RESLT1.
001810        DISPLAY "FLD-1 = " FLD1, "FLD-2 = " FLD2, "RESULT = " RESLT1.
001820        DISPLAY "FLD-3 = " FLD3.
001830        IF RESLT1 IS EQUAL TO 0
001840           SET IND-22-ON TO TRUE.
001850        GOBACK.
```

Figure 8-38. The External Subroutine EXSUB Written in COBOL

```
      extern char fld3[];
      extern char reslt2[];
      extern int in22;
   exsub2 ()
   {
      if (in22)
         strncpy (reslt2, fld3, 6)
      else
         strncpy (reslt2, "      ", 6);
   }
```

Figure 8-39. The External Subroutine EXSUB2 Written in C

SET

This **display** operation enables the function keys (f1 through f8) on the keyboard. Once enabled, they can be used with the DSPLY and DSPLM operations to set the function key indicators. If you have a terminal that supports the function key labeling feature, you can also use this operation to label the function keys. See Chapter 11, "The RPG Screen Interface (RSI)", for information about setting function key labels for RSI applications.

To label the function keys, enter an array name in the Factor 1 Field. You must define the array in a File Extension Specification. Each element of the array defines the text to be displayed on the screen for a function key. The array must contain eight elements, each sixteen characters long. The first eight characters are displayed in the top half of the label and the last eight characters are displayed in the bottom half of the label.

To use function keys with DSPLY or DSPLM, enter up to three function key indicators in the Resulting Indicators Field (columns 54-59). To enable all of the function keys, enter F@ in columns 54-55. Function keys that you do not enable are disabled and, when pressed, cause this message to be displayed:

```
Function Key not enabled!
```

Example

Figure 8-40 shows how to use SET to enable all of the function keys and to define the text for their labels. The SET operation in line 2 identifies the array KEYLBL (line 1) that contains the labels for the function keys and it also enables the function keys (F@). In this particular example, the function keys are used to prompt a user to select a report to print. The function key labels are shown below as they appear in the KEYLBL array.

```
Current Detail
Current Summary
Y-T-D   Detail
Y-T-D   Summary
Compare Detail
Compare Summary
MonthEndProcess
Year-EndProcess
```

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
     ____________________________________________________________________________________

1    E                    KEYLBL  1   8 16                FUNC KEY LABELS

2    C           KEYLBL    SET                      F@
```

Figure 8-40. Using the SET Opertion

SETLL

This **file** operation sets the lower record limit for sequential read operations in KSAM and TurboIMAGE demand files.

To condition the operation, enter indicators in either or both the Control Level Field (columns 7-8) and the Indicators Field (columns 9-17).

Enter a literal or field name in the Factor 1 Field that gives the value of the lower limit. The length of Factor 1 must be the same as the key field length for the file except when you're using partial keys. You can use partial (shorter) keys with KSAM files. If you do, they must be unpacked fields and you must enter a relational operator in the Result Field (columns 43-45). When you use a partial key, RPG gets the first record whose key (leftmost characters) matches the characters in the Factor 1 Field.

Enter the file name in the Factor 2 Field. The file must be a KSAM or TurboIMAGE demand file.

You may enter one of the following relational operators in columns 43-45 of the Result Field in KSAM files only:

Enter this relational operator:	To set the lower limit to record keys that are:
*EQ	Equal to Factor 1.
*GT	Greater than Factor 1.
*GE or blank	Greater than or equal to Factor 1.

You can enter an indicator in the High Subfield (columns 54-55) of the Resulting Indicators Field. It is turned ON if SETLL fails to locate a record specified by the relational operator (the Equal Subfield indicator (columns 58-59) is turned ON by a subsequent READ to the file).

If a status array has been specified for a file, the SETLL operation will return the status of a DBFIND operation on an image demand file. This is useful for finding the length of a chain. If the information from the DBFIND is needed later in the program, it should be saved in some other variables prior to any following READ since the status from its DBGET will replace that from the DBFIND.

Example

Figure 8-41 shows how to use the SETLL operation to set the lower record limit for the file named IFILE. The lower limit is any key value greater than the value in the field LOWLIM.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________

C           LOWLIM    SETLLIFILE     *GT
```

Figure 8-41. Using the SETLL Operation

SETOF

This **indicator and bit setting** operation turns OFF the indicators that you enter in the Resulting Indicators Field (columns 54-59). You can enter from one to three indicators in the field. Do not use the Factor 1 and Factor 2 Fields.

SETON

This **indicator and bit setting** operation turns ON the indicators that you enter in the Resulting Indicators Field (columns 54-59). You can enter from one to three indicators in the field. Do not use the Factor 1 or Factor 2 Fields.

Example

Figure 8-42 shows how to use the SETON operation. Indicators 03 and L2 are turned ON.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234

C                     SETON                     03L2
```

Figure 8-42. Using the SETON Operation

SORTA

This **array** operation arranges array elements into ascending or descending sequence in both alphanumeric and numeric arrays.

Enter the array to be sorted in the Factor 2 Field. To condition the operation, enter indicators in the Control Level Field (columns 7-8) or the Indicators Field (columns 9-17), or both. Leave all other columns blank.

If the Table/Array Sequence Field (column 45) of the File Extension Specification for the array is A or blank, the array is sorted in the ascending order. If the Table/Array Sequence Field is D, the array is sorted in descending order.

Alphanumeric arrays are sorted using the ASCII collating sequence. You cannot specify an alternate collating sequence.

Example

Figure 8-43 shows how to use SORTA to sort an array in ascending sequence and in descending sequence. The array ARSEQ is read from the file ANYFILE and moved to the array ARSEQA. ARSEQA is then sorted in ascending sequence in line 1. ARSEQ is then moved to the array ARSEQD and sorted into descending sequence in line 2.

Table 8-18 gives an example of the elements in each of the arrays after line 2 is executed.

Table 8-18. Elements when using SORTA

Element #:	ARSEQ:	ARSEQA:	ARSEQD:
1	791006	470711	840315
2	470711	530820	830512
3	530820	640218	791006
4	640218	791006	640218
5	830512	830512	530820
6	840315	840315	470711

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

E                    ARSEQ       6  6
E                    ARSEQA      6  6  A
E                    ARSEQD      6  6  D

IANYFILE NS  01
I                                          1  36 ARSEQ

C   01                MOVE ARSEQ     ARSEQA
C   01                SORTAARSEQA
C   01                MOVE ARSEQ     ARSEQD
C   01                SORTAARSEQD
```

(1) (2)

Figure 8-43. Using the SORTA Operation

SQRT

This **arithmetic** operation computes the square root of Factor 2 and places it in the Result Field. You can enter an array name in the Factor 2 and Result Fields. When you do this, the square root of each element in the Factor 2 array is placed in the corresponding element of the Result Field array.

You can use half-adjust with this operation. But do not use Factor 1, and do not enter a negative value in Factor 2. Make sure the number of decimal positions in the Result Field is not larger than the size of the Result Field, and do not use resulting indicators.

SUB

This **arithmetic** 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, Factor 2 is subtracted from the Result Field and the difference is placed in the Result Field.

SUSP

This **system** operation immediately suspends execution of the program and returns control to the parent process (the RPG program must be activated as a child process by the parent process). Before the program is suspended, these actions are performed:

- The user indicators (U1-U8) are posted to the system Job Control Word (JCW).
- The Local Data Area (LDA) is posted to the LDAFILE.
- If the program uses a WORKSTN file, the terminal is cleared and removed from block mode. The BREAK key is enabled.

When the RPG program is reactivated by the parent process, the actions listed below take place and the RPG program resumes execution with the operation following SUSP:

- The user indicators (U1-U8) are restored.
- The Local Data Area (LDA) is restored.
- If the program uses a WORKSTN file, the terminal is set to block mode and the previous screen is redisplayed.

To condition the SUSP operation, enter one or more indicators in the Control Level Field (columns 7-8) and (or) the Indicators Field (columns 9-17). Enter SUSP in the Operation Field and leave the remaining specification fields blank.

When a suspended RPG program is resumed, record pointers are not reset. This means that, if you're processing a file sequentially, the next record (not the first one in the file) is presented. For this reason, you may want to use SUSP only in those programs that process files randomly. Also, when using file or record locking, be sure to unlock all files and records before executing SUSP. You can unlock them by either forcing a write to the file or by using UNLCK.

Suspending an RPG Program

There are two ways to suspend an RPG program. You can suspend the program when it ends normally or you can suspend the program at any point in the Calculation Specifications by using the SUSP operation.

The suspend feature is designed for programs that are called from menu programs and the PROCMON menu processing system. Suspending saves time when switching back and forth between a menu program (or PROCMON) and an RPG program because files remain open and the RPG program does not have to be reloaded when execution starts again. When an RPG program ends or is suspended, control returns to the menu or parent process (the menu program or PROCMON). The program remains suspended until you specifically start it again.

To suspend the program when it terminates normally, enter a command to set "suspend mode", then follow this by a RUN command to start the program. The example below shows how to do this:

```
:SETJW RPGSUSP=1          Sets "suspend" mode
:RUN GL45P                Starts execution of GL45P
   .                      This line is executed as soon as GL45P ends
   .
   .
:RUN GL45P                Starts execution of GL45P at the beginning
```

If you want to control when the program is suspended, enter a command to create a process for the RPG program, then activate the process. The following lines create a process for the program PAY0680, then activate (execute) the program. The program contains a SUSP Calculation Specification to suspend the program. Once suspended, the program "sleeps" until the next ACTIVATE command. When the program resumes, execution begins with the operation following SUSP.

```
:CREATE PAY0680        Creates process only; does not execute it
:ACTIVATE PAY0680      Starts RPG program PAY0680
       .               This line is executed after SUSP encountered
       .
       .
:ACTIVATE PAY0680      Starts PAY0680 with the operation following SUSP
```

TAG

This **branching** operation labels the operation immediately following it. This allows you to skip to the operation from other places in the program.

To use TAG, enter it immediately before the operation to which you want to branch. Enter a label in the Factor 1 Field. Do not enter a label that is used in another TAG operation. A TAG label can be the same as a field name. Do not use the Factor 2 and Result Fields.

If the TAG label is associated with a total-time operation, enter a control-level indicator in the Control Level Field (columns 7-8). Do not enter indicators in the Indicators Field (columns 9-17).

Example

Figure 8-44 shows how to use GOTO and TAG. The TAG operation in line 1 assigns the label START to the ADD operation that follows it. When the ADD operation turns ON indicator 01, the GOTO operation in line 2 branches to the operation whose label is CALCS (line 4). This branch is a conditional branch because it occurs only when indicator 01 is turned ON.

The GOTO operations in lines 3 and 5 are unconditional branches, since no indicator is used with them.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123
_____________________________________________________________________

C                      .
C                      .
C                      .
C           START      TAG
C           SUM1       ADD  SUM2      TOTAL  102     01
C   01                 GOTO CALCS
C           MXRA       MULT MXRB      PRODCT 102
C                      GOTO START
C                      .
C                      .
C                      .
C           CALCS      TAG
C                      .
C                    (calculations)
C                      .
C                      GOTO START
```

1 2 3 4 5

Figure 8-44. Using the GOTO and TAG Operations

TESTB

This **compare and test** operation (TEST Bit) tests the bits in a one-character, alphanumeric field to determine if they are ON or OFF. Enter the one-character field to test in the Result Field. Enter the bit pattern to use for testing in the Factor 2 Field. When TESTB is executed, the indicators in the Resulting Indicators Field (columns 54-59) are turned ON and OFF according to the results of the test.

You can enter an alphanumeric literal, a field, a table, or an array element in the Factor 2 Field. In all these cases, Factor 2 must be a single character. If you enter a field, table, or array, its ON bits are compared with the Result field. If you enter an alphanumeric literal, you specify the bits to be tested. Enter up to eight digits enclosed in quotation marks. Enter a digit for each bit position you want to test. Zero (0) stands for the high-order bit and seven for the low-order bit. You can enter the bit numbers in any order; if you do not enter a number for a bit, it is not tested. For example, "02" tests bits 0 and 2 to determine whether they are turned ON. If the bit pattern for the character in the Result Field is 10100000, the equal resulting indicator is turned ON.

You must enter at least one indicator in the Resulting Indicators Field and you may enter up to three. You can enter the same indicator in two of the fields, but not in all three. If all of the bits in Factor 2 are turned OFF, no resulting indicators are turned on. If the bits specified for Factor 2 are OFF in the Result Field, the resulting indicator in the High Subfield (columns 54-55) is turned ON. If two or more bits do not match in these fields, the resulting indicator in the Low Subfield (columns 56-57) is turned ON. If all bits that are turned ON in the Factor 2 Field are also turned ON in the Result Field, the indicator in the Equal Subfield (columns 58-59) is turned ON.

Example

Figure 8-45 shows how to use the TESTB operation. The operation compares the alphanumeric literal "0356" to the one-character field RESLT. The literal "0356" stands for the bit pattern 10010110 and indicates that bits 0, 3, 5, and 6 in the Result Field are to be tested. If the Result Field contains the bit pattern 01101001, indicator 01 is turned ON. If the Result Field contains 11110000, indicator 02 is turned ON. If the Result Field contains 10010110, indicator 03 is turned ON.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------

C                     TESTB"0356"    TESLT   1  010203
```

Figure 8-45. Using the TESTB Operation

TESTN

This **compare and test** operation (TEST Numeric) tests an alphanumeric field for numeric characters. You can use this operation before arithmetic or editing operations to avoid unpredictable results or program termination. When TESTN is executed, the indicators in the Resulting Indicators Field (columns 54-59) are turned ON and OFF according to the results of the test.

If all characters in the Result Field are numeric or blank, the indicator in the High Subfield (columns 54-55) is turned ON. In this case, each character except the low-order character must contain a digit to be numeric. The low-order character is numeric if it contains A-R, or +0 (octal 173), or -0 (octal 175.) If the Result Field contains numeric characters and leading blanks, the indicator in the Low Subfield (columns 56-57) is turned ON. (Do not enter an indicator in this subfield when you are testing a field one character long, because the field must contain a character and a leading blank for the test to be valid.) If the field contains blanks only, the indicator in the Equal Subfield (columns 58-59) is turned ON.

You can use the same indicator to test for more that one condition. The indicator is turned ON when any of the conditions are met.

TESTZ

This **compare and test** operation (TEST Zone) tests the zone bits of the high-order character of an alphanumeric field, and sets the indicators in the Resulting Indicators Field (columns 54-59) accordingly.

Enter the field to test in the Result Field. Enter an indicator in the High Subfield (columns 54-55) to turn ON that indicator when the character has a 12-zone (the characters &, +0, or A-I). Enter an indicator in the Low Subfield (columns 56-57) to turn ON that indicator when the character has an 11-zone (the characters -, -0, and J-R). Enter an indicator in the Equal Subfield (columns 58-59) to turn ON that indicator when the character has some other bit pattern. Do not use the Factors 1 and 2 Fields.

TIME

This **system** operation returns the time of day (system) and, optionally the current date. TIME uses the system intrinsics CLOCK and optionally, CALENDAR.

To use TIME, leave the Factor 1 and Factor 2 Fields blank. Enter the name of the numeric field where you want to save the time (and optionally, the date) in the Result Field. The length of the field that you enter determines whether the date is also returned. The field must be either 6 or 12 digits long; 6 for returning the time and 12 for returning the time and date. Specify that the Result Field has zero decimal positions. The following lines show the time and date values that are placed in the Result Field.

This value is placed in the Result Field:	**When:**
`hhmmss`	The Result Field length is 6. The time is based on a 24-hour clock where `hh` represents hours, `mm` represents minutes, and `ss` represents seconds.
`hhmmssmmddyy`	The Result Field length is 12 and the Inverted Print Field (column 21) of the Header Specification is blank. The date is represented in Domestic Format where `mm` is the month, `dd` is the day, and `yy` is the year.
`hhmmssddmmyy`	The Result Field length is 12 and the Inverted Print Field (column 21) of the Header Specification is I, J, or D. The date is represented in Foreign Format where `mm` is the month, `dd` is the day, and `yy` is the year.

Leave the Half Adjust and Resulting Indicators Fields blank.

When TIME is executed, the date is retrieved dynamically from the system. It is not taken from the static UDATE field (which can be different from system date). For example, if a program begins shortly before midnight on 12/31/88, UDATE is initialized to 123188 and remains unchanged for the duration of the program. If the program runs past midnight, the TIME operation must be used to obtain 010189.

Example

Figure 8-46 shows the two ways to use TIME. Line 1 returns the time in the field TIMEA. Line 2 returns the time and date in the field TIMEB.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
     ____________________________________________________________________________________

1    C                   TIME           TIMEA   60       TIME OF DAY
     C
2    C                   TIME           TIMEB   120      TIME AND DATE
```

Figure 8-46. Using the TIME Operation

TIME2

This **system** operation extracts all or part of a formatted system date and time string and stores it in a field in the program. TIME2 uses the DATELINE and CALENDAR intrinsics.

To use TIME2, leave the Factor 1 Field blank and, in the Factor 2 Field, enter the position where extraction begins. You can use a numeric literal or field for Factor 2. If Factor 2 is a numeric literal, it must be an unsigned number from 1 to 40 (entered in columns 33-34). If Factor 2 is a numeric field, it must be less than 5 digits and contain no decimal positions. Enter the name of the alphanumeric field where you want to save the returned date and time in the Result Field. The length of the Result Field determines how many characters are extracted. It can contain a maximum of 40 characters (the sum of the Result and Field Length Fields must not exceed 40). Leave the Decimal Positions, Half Adjust, and Resulting Indicators Fields blank.

The 40-character system date and time string has the following format (the starting positions of sections of the string are also shown):

```
day,  mon  dd,  year,  hh:mm  xM    JULIAN:nnn
 ^    ^    ^    ^       ^      ^           ^
 |    |    |    |       |      |           |
 1    6    10   14      20     26          38   (starting positions)
```

The `day` is the day of the week; `mon` is the month; `dd` is the day number; `year` is the 4-digit year; `hh` is hours; `mm` is minutes; `xM` is AM or PM; and `nnn` is the day of the year in Julian format.

Example

Figure 8-46 shows several TIME2 operations. Given the date 11/7/88 and the time 9:25 AM, the extracted results are:

```
1   MON, NOV 7,1988, 9:25 AM JULIAN:042

2   MON, NOV 7,1988, 9:25 AM

3   NOV 17, 1988

4   1988

5   9:25 AM

6   042
```

Figure 8-46. Using the TIME2 Operation

UNLCK

See the LOCK operation.

XFOOT

This **arithmetic** operation sums all elements of a numeric array and places the sum in the Result Field. Factor 2 names the array to sum and it must be numeric. The Result Field can be a field or an array element. If it is an element of the array entered in Factor 2, the element's value (before the XFOOT operation) is used during summing. Do not use the Factor 1 field. You can half adjust the total in the Result Field. When you do this, rounding is done after all elements are added.

Z-ADD

This **arithmetic** operation (Zero and ADD) replaces the Result Field by Factor 2. Factor 1 is not used. High-order truncation occurs if the Result Field length is less than the Factor 2 length.

Z-SUB

This **arithmetic** operation (Zero and SUBtract) replaces the Result Field by the negative representation (complement) of Factor 2. High-order truncation occurs if the Result Field length is less than the Factor 2 length.

Factor 2 (Columns 33-42)

The Factor 2 field names an operand to use in the operation. What you enter in this field depends on the operation you're using (see the description for the operation in this chapter).

Columns 33-42	Description
The name of a field, table, array, array element, or file.	The field containing the data or (if this is a file) the name of the file.
Subroutine name.	The internal subroutine to execute.
A label.	The label for a TAG, ENDSR, or GOTO operation.
An alphanumeric, numeric literal, or a figurative constant.	The actual data to be used.
Blank.	The operation does not use an operand in this field.

Field, Table, Array, Subroutine, and Label Names

You must define field names that you enter in this field somewhere in the program. Some field names, however, are predefined; you can use them without defining them. These fields are UDATE, UMONTH, UDAY, UYEAR, PAGE, PAGE1-PAGE7, and *ERROR.

Alphanumeric Literals

Alphanumeric literals are constants that consist of ASCII characters. Alphanumeric literals can contain up to eight characters including blanks. Do not use alphanumeric literals in arithmetic operations.

When entering alphanumeric literals, enclose them in quotation marks. For instance, to use the literal ALPHALIT, enter `"ALPHALIT"`. To include a quotation mark in the literal itself, enter two quotation marks. For example, to enter the literal NAME, enter `"""NAME"""`.

If you want to use apostrophes instead of quotation marks to enclose alphanumeric literals, enter the apostrophe in the QUOTE= parameter of the $CONTROL compiler subsystem command.

Numeric Literals

A numeric literal is the actual number that is used in the operation. Use numeric literals the same way you use field names. Numeric literals can contain up to ten characters, including a decimal point and a leading plus or minus sign (unsigned literals are treated as positive numbers). For example, 123.68 is a valid numeric literal.

Do not embed blanks in numeric literals and do not enclose them in quotation marks.

Figurative Constants

Figurative constants are predefined names that, when used, produce one or more identical characters. The figurative constants *BLANK and *BLANKS produce one or more blanks and are used with alphanumeric fields only. *ZERO and *ZEROS produce one or more zeros and are used with either numeric or alphanumeric fields. The number of blanks or zeros produced depends on the size of the Result Field.

You can use figurative constants only with the operations CHAIN, COMP, LOKUP, MOVE, MOVEL, and MOVEA.

Example

Figure 8-47 gives four examples of how to use the Factor 2 Field. The operations starting in line 1 use alphanumeric literals. The ADD operation in line 2 uses a numeric literal and the operation in line 3 uses a field name. Finally, the operations starting in line 4 use figurative constants.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

1)  C                   MOVE "AFRES"   SERV   10
    C                   MOVE "2001"    TITLE  15
    C                     .
    C                     .
2)  C           DATAX   ADD  100.50    RESULT  82
    C                     .
    C                     .
3)  C           10      MULT ALPHA     PROD   102
    C                     .
    C                     .
4)  C                   MOVE *ZEROS    RESULT
    C                   MOVE *BLANK    TITLE
    C                   MOVE *ZEROS    QUOTA  64
    C                   MOVE *BLANKS   SERV
```

Figure 8-47. Using the Factor 2 Field (columns 33-42)

Result (Columns 43-48)

The Result Field identifies the field that contains the result of the operation. You may use a field name that is not defined elsewhere in the program (if you do this, enter the length of the field in the Field Length Field (columns 49-51).

Table 8-19. The Result Field

Columns	Value	Description
43-48:	Name of a field, table, array, or indexed array element	The area where the result is stored.
	blank	The Result Field is not used by this operation.
43-45:	*EQ	The record that sets the lower limit (SETLL) for sequential read operations must be equal to the value in the Factor 1 Field.
	*GT	The record that sets the lower limit (SETLL) for sequential read operations must be greater than the value in the Factor 1 Field.
	*GE	The record that sets the lower limit (SETLL) for sequential read operations must be greater than or equal to the value in the Factor 1 Field.

The programmer is responsible for making sure the Result Field is large enough to hold the result of the operation. If an alphanumeric Result Field is too small, the field is truncated on the left. If the Result Field is used in arithmetic or numeric compare operations, or if it is an Output Specification field that is edited or zero suppressed, it must be numeric.

For arithmetic operations, the Result Field will **overflow** and data will be lost if the Result Field is not large enough to hold the result of the operation. With full numeric overflow checking enabled (through the $CONTROL OVFLOCHK compiler command) an arithmetic overflow initiates the overflow error trap. You can control how your program handles the overflow trap with entries in the Header Specification, columns 55 and 65. If you disable full overflow checking for some calculations with the $CONTROL NOOVFLOCHK compiler command (or if you haven't enabled full checking), RPG may left-truncate the calculated value instead of initiating the overflow error trap. This happens when either Factor 1 or Factor 2 is the same size or larger than the calculated value of the operation, but the Result Field is too small to hold the value. If the size of the calculated value is larger than the size of each of Factor 1, Factor 2, and the Result Field, then RPG initiates the overflow error trap, regardless of the setting of OVFLOCHK.

Field Length (Columns 49-51)

The field length field specifies the number of characters in the Result Field. This field is optional when the Result Field is defined elsewhere in the program. (If it is already defined, the number that you enter must be the same as the original definition.)

Table 8-20. The Field Length Field

Columns 49-51	Description
1-256 (right-justified, leading zeros are not required)	The length of the alphanumeric Result Field.
1-15 (right-justified, leading zeros are not required)	The length of the numeric Result Field.
blank	This is an alphanumeric or numeric field, described elsewhere in the program.

Make sure that the Result Field is long enough for the largest possible result. If it is too small, excess significant digits are truncated (or for numeric fields, the overflow trap may be initiated). If you specify rounding in the Half Adjust Field (column 53), the number that you enter is the length of the result after rounding.

Decimal Positions (Column 52)

The decimal positions field specifies the number of decimal positions in the Result Field. This field is optional when the Result Field is defined elsewhere in the program. (If it is already defined, the number that you enter must be the same as the original definition.)

Table 8-21. The Decimal Positions Field

Column 52	Description
0-9	The number of decimal positions in the Result Field.
blank	This is an alphanumeric or a numeric field defined elsewhere in the program.

If the field has no decimal positions, enter 0. The number of decimal positions can be larger or smaller than the number of decimals produced by the operation, but must not be greater than the length of the Result Field. Data is aligned by decimal point when placed in the Result Field. When the number of decimals is less than the result of the operation, low-order digits are truncated. When the number of decimals is greater than the result of the operation, zeros fill the excess decimal positions.

Half Adjust (Column 53)

The half adjust field rounds numeric data when it is placed in the Result Field. Do not half adjust alphanumeric fields or use half adjusting with the MVR operation or with a DIV operation followed by an MVR operation.

Table 8-22. The Half Adjust Field

Column 53	Description
H	Half adjust data.
blank	Do not half adjust data.

Half adjusting is only performed when the Result Field has fewer decimal positions that the result of the operation. Half adjusting is performed by adding 5 (or -5 for negative numbers) to the most significant digit that will be truncated. The sum is then truncated to fit in the Result Field.

Example

Figure 8-48 shows how to half adjust the result of an ADD operation. The sum of 50 and the number in DATAN is rounded to fit into the field RESULT which contains 2 decimal positions. Assuming that the sum is 325.5769, 5 is added to the digit 6 (most significant digit to be truncated). The sum becomes 325.5819. This rounded sum is stored in the field RESULT as 325.58.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

C           50        ADD  DATAN     RESULT  52H
```

Figure 8-48. Using the Field Length and Half Adjust Fields

Resulting Indicators (Columns 54-59)

The resulting indicators field lets you set one to three indicators to reveal the outcome of the operation. At a later point in the program, you can then use the indicators to direct program execution.

Column	Value	Description
High Subfield (54-55):	01-99, F0-F9, H0-H9, KA-KN, KP-KY, L0-L9, LR, MR, OA-OG, OV, U1-U8 1P	The indicator that is turned ON when a high condition exists (see the operation for specific information).
	blank	Do not test data in this field.
Low Subfield (56-57):	01-99, F0-F9, H0-H9, KA-KN, KP-KY, L0-L9, LR, MR, OA-OG, OV, U1-U8 1P	The indicator that is turned ON when a low condition exists (see the operation for specific information).
	blank	Do not test data in this field.
Equal Subfield (58-59):	01-99, F0-F9, H0-H9, KA-KN, KP-KY, L0-L9, LR, MR, OA-OG, OV, U1-U8 1P	The indicator that is turned ON when a equal condition exists (see the operation for specific information).
	blank	Do not test data in this field.

When you use an indicator in this field, be sure you understand how it is used in the RPG logic cycle (see the *HP RPG Programmer's Guide*).

Indicators that you enter in this field are turned OFF before the operation, and are turned ON only when the condition (indicated by the subfield in which you enter it) is satisfied. If you use the same indicator with consecutive operations, the last operation determines the indicator setting. If you enter a general indicator, you can use it later to condition calculation or output operations. If you enter a halt indicator, you can use it to suppress output operations or to halt the program.

High Subfield (Columns 54-55)

An indicator entered in the high subfield is turned ON:

- When an arithmetic operation produced a positive (greater than zero) Result Field.
- In a COMP operation, when Factor 1 is greater than Factor 2.
- In a LOKUP operation, when the Factor 2 table or array element is greater than Factor 1.
- During a CHAIN operation, when a record was not found.
- When a TESTB operation reveals that the bits in a field are zero.
- When a TESTZ operation reveals that the zone bits of a field are positive, representing the ASCII characters &, A-I.
- When a TESTN operation reveals that an alphanumeric field contains ASCII digits. (The low-order bits can still represent A-R.)
- When a SETON or SETOF operation uses the indicator.

Low Subfield (Columns 56-57)

An indicator entered in the low subfield is turned ON:

- In arithmetic operations, when the Result Field is negative.
- During the COMP operation, when Factor 1 is less than Factor 2.
- During the LOKUP operation, when the Factor 2 table or array element is less than Factor 1.
- When a TESTB operation reveals that the bits in a field are mixed; some ON and some OFF. (The operation is not meaningful if you test a single bit.)
- When a TESTZ operation reveals that the zone bits in a field are all negative, representing the ASCII characters minus (-) and J-R.
- When a TESTN operation reveals that an alphanumeric field (that must be at least two characters long) contains ASCII digits and leading blanks.
- For a TurboIMAGE chaining file (Input/Output Mode Field (column 67) in the File Description Specification Continuation line is C or R), when end-of-chain is encountered.
- When a SETON or SETOF operation uses the indicator.

Equal Subfield (Columns 58-59)

An indicator entered in the equal subfield is turned ON:

- In arithmetic operations, when the Result Field is zero.
- During the COMP operation, when Factor 1 equals Factor 2.
- During the LOKUP operation, when the Factor 2 table or array element equals Factor 1.
- When a TESTB operation reveals that the bits in a field are all ON.
- When a TESTZ operation reveals that the zone bits in a field are not all positive or all negative; they represent characters other than &, A-I, -, or J-R.
- When a TESTN operation reveals that an alphanumeric field contains all blanks.
- When a READ operation for a demand file encountered end-of-file.
- When a SETON or SETOF operation uses the indicator.

Example

Figure 8-49 shows how to set a resulting indicator and how to use it in subsequent Calculation Specifications. The MULT operation in line 1 turns indicator 20 ON when COST is negative. The ZSUB operation in line 2 is executed only when COST is negative (indicator 20 is ON). If the result (TCOST) of the SUB operation in line 3 is negative, indicator 20 is turned ON and it is turned OFF if the result is positive or zero. (At the beginning of the next program cycle, indicator 20 is turned OFF automatically.)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

1    C           MILES     MULT .48       COST   102   20
     C
2    C   20              Z-SUBCOST      COST
     C
3    C           COST      SUB  .05       TCOST  102   20
```

Figure 8-49. Using Resulting Indicators

Comments (Columns 60-74)

Enter comments of any kind in the comments field.

Program Name (Columns 75-80)

The program name field contains the program name. The format for this field is discussed in Chapter 2.

The Calculation Specification Default Summary

If you leave the optional fields of the Calculation Specifications blank, the defaults shown in Table 8-18 apply:

Table 8-23. Calculation Specification Defaults

Columns	Field	Default Values
1-5	Sequence Number	No sequence number applies.
7-8	Control Level	The operation is performed at detail time.
9-17	Indicators	Operation done for every record if the Control Level Field does not contain L0-L9 or SR.
43-48	Result	No result applies.
49-51	Field Length	This is an alphanumeric field, or a numeric field described elsewhere in the program.
52	Decimal Positions	This is an alphanumeric field, or a numeric field described elsewhere in the program.
53	Half Adjust	Do not half adjust data.
54-59	Resulting Indicators	No indicator assigned.
60-74	Comments	No comments made.
75-80	Program Name	None.

9

Output Specifications

For each output, update, and combined file in a program, you must enter an Output Specification.

The Output Specification describes:

- The records to be written to the output file. (Use the Record Description Fields, columns 7-31, to enter record information.)
- The formats and locations of fields in the output records. (Use the Field Description Fields, columns 23-70, to enter field information.)

The Output Specification is identified by an O in column 6:

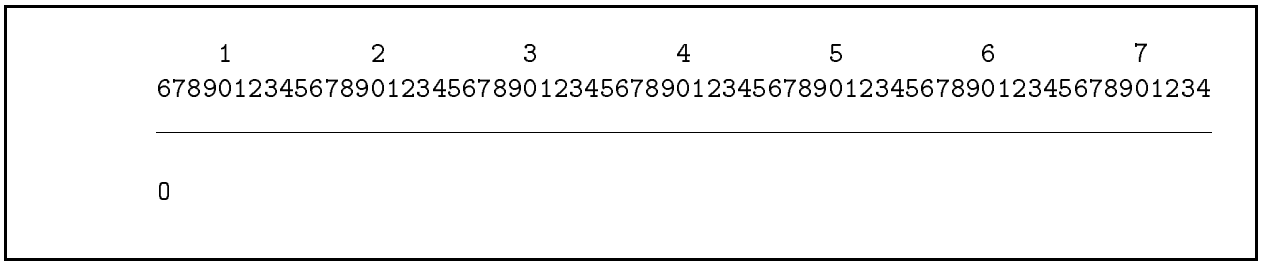


Figure 9-1. The Output Specification

The Output Specification Fields

The fields you can use in the Output Specification are described in the sections which follow in this chapter. Each field has a unique name and occupies specific columns in the specification.

Sequence Number (Columns 1-5)

The sequence number field contains the source record sequence number, described in Chapter 2.

Specification Type (Column 6)

The specification type field contains an O to identify this as an Output Specification.

Record Description Fields (Columns 7-31)

The record description fields describe the record types contained in the output, update, or combined file.

A good programming practice is to group all Output Specifications for a file together. Define the first record type by entering the appropriate Record Description Fields and leaving columns 32-70 blank. You may follow this line by one or more AND or OR lines (see the AND/OR Field, columns 14-16). Next, enter one or more specifications that describe the fields for the record type (see the Field Description Fields, columns 32-70). Repeat this specification sequence until all record types are defined.

Example

Figure 9-2 shows three output records together with their field descriptions. Line 1 contains the Record Description Fields for the file OUT. Lines 3 and 6 show the Record Description Fields for two records in the file OUTDIR. Lines 2, 4-5, and 7 contain the Field Description Fields for these records.

```
      1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

1     OOUT      H
2     O                                        60 "REPORT - 1988"
      O                                       .
      O                                       .
3     OOUTDIR   DADD      07
4     O                           FIELDA       5
5     O                           FIELDB      10
6     O         TADD      L1
7     O                           FIELDC      15
```

Figure 9-2. Entering Output Specification for Two Files

File Name (Columns 7-14)

The file name field names the file to which this and subsequent Output Specifications apply. Enter the name of an output, combined, or update file defined by a File Description Specification.

Columns 7-14	Description
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.)	The name of the output, update, or combined file being described by this Output Specification.

If you enter several Record Description lines for the same file, you only need to enter the file name on the first one. The file name remains in effect until a new name is encountered.

If the file is an output file, RPG creates it with the characteristics shown in Table 9-1. You can override some of these characteristics using the MPE/iX FILE command.

Table 9-1. Output File Characteristics

FOPEN	
Formal File Designator:	File name or DSNAME.
Foptions:	
Domain:	The file is opened as an OLD (11) file. If this fails, it is opened 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).
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 Chain: 2 Append Only: 3 Combined: 4 Update: 5

Table 9-1. Output File Characteristics (Continued)

Multirecord Access:	No (0).
Dynamic Locking:	No (0) if no LOCK File Description Continuation line; otherwise, yes (1).
Exclusive:	Default (00) if no LOCK File Description Continuation line; otherwise, share (11).
Inhibit Buffering:	No (0).
Record Size:	As specified in columns 24-27 of the File Description Specification. Default: 80 bytes.
Device Name:	As specified in columns 40-46 of the File Description Specification. Default: DISK.
Forms Message:	If specified, RPG points to a buffer containing blanks, causing the standard alignment line to print.
User labels:	As specified in column 53 of the File Description Specification. Default: none.
Blocking Factor:	As specified in columns 20-23 and 24-27 of the File Description Specific Default: 1.
Number of Buffers:	Default (2).
File Size:	Default (1023).
Number of Extents:	As specified in columns 68-69 of the File Description Specification. Default: 8.
Initial Allocation of Extents:	1.
File Code:	0.
FCLOSE	
Disposition:	No change.
Security Code:	Normal (0)

AND/OR (Columns 14-16)

The AND/OR field lets you specify additional indicators that determine whether the output record is written. (Without using AND and OR lines, the maximum number of indicators that you can use is 3.) You cannot use this field to condition individual fields in the output record.

Columns 14-16	Description
AND	Identifies this Input Specification as an AND line.
OR	Identifies this Input Specification as an OR line.

You can intermix AND and OR lines. (The total number of AND and OR lines that you can use for each output record is 20.) Each OR line begins a new decision path (see the example which follows).

When you use AND lines, all indicators that you enter in the Output Indicators Field (columns 23-31) must be satisfied before the record is written. When entering AND lines, follow these steps:

1. Make sure that the previous Output Specification contains three indicators in the Output Indicators Field (columns 23-31).
2. Enter AND in columns 14-16 and up to three indicators in the Output Indicators Field. Leave columns 17-22 blank.

   If this AND line is followed by other AND lines and you want to use Fetch Overflow or Release (the Fetch Overflow/Release Field, column 16) with any of them, you must enter F or R on this line only (the first AND line).
3. Continue entering AND lines (or OR lines) until you've entered all of the conditions (indicators) for writing the record.

When OR lines are executed, the record is written when all of the indicators in the Output Indicators Field (columns 23-31) are satisfied. When the OR line is followed by one or more AND lines, the record is also written when all of the indicators in the AND lines are satisfied. When you enter OR lines, follow these steps:

1. Make sure that if the previous specification is the first Record Description line, there is at least one indicator in the Output Indicators Field (columns 23-31).
2. Enter OR in columns 14-15 and up to three indicators in the Output Indicators Field.

   If you want to use different spacing or skipping for records that satisfy the OR line indicators, enter the appropriate values in the Space Field (columns 17-18) or the Skip Field (columns 19-22), or both. If you leave these fields blank, the spacing and skipping specified for the previous specification is used.

   If you want to use Fetch Overflow or Release (the Fetch Overflow/Release Field, column 16), enter F or R on this line but not on subsequent AND lines, if there are any.
3. Continue entering OR lines (or AND lines) until you're entered all of the conditions (indicators) for writing the record.

Example

Figure 9-3 shows how to use AND and OR lines to condition an output record. The record beginning in line 5 is written when:

- Indicators 01, 02, and 03 are ON (line 1).

  or

- Indicators 06, 07, 08, *and* 09 are ON (lines 2, 3).

  or

- Indicators 10 and 11 are ON (line 4).

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------

(1)       O         T 2       01 02 03
(2)       O        OR         06 07 08
(3)       O        AND        09
(4)       O        OR         10 11
(5)       O                             PAYTOT    10
```

Figure 9-3. Using AND or OR Lines to Condition Record Output

Type (Column 15)

The type field specifies when the record is written in the logic cycle. You can enter one of the following four values:

Column 15	Description
D	Detail record.
E	Exception record.
H	Heading record.
T	Total record.

Within each of the above record types, records are written in the same order as their specifications are entered. You can enter all records together that have the same type (for example, you can place all heading records for all files together), or you can enter records for each type consecutively within each file (for example, you can enter heading records, then detail records, then total records, and finally exception records for a particular file.)

D

Detail records are written at detail time in the logic cycle. They contain the most basic level of information of all the record types. Usually, this information is closely related to input data. It is either read directly from the input records or is calculated using the input records.

E

Exception records are written at detail or total time at the point at which an EXCPT Calculation Specification operation is encountered. (The records are written at total time when conditioned by a control-level indicator.) Use exception records for output that occurs infrequently, under unusual circumstances, or when the number of output records varies during each logic cycle.

Do not condition exception records for primary and secondary update files with control-level indicators or with the last-record indicator. If you do, the results are unpredictable.

H

Heading records are written during pre-cycle processing in the logic cycle. (To print heading records at other times in the logic cycle, condition the output record with an overflow indicator.) Heading records contain data that generally remains constant, such as report titles and column headings. Heading records can contain a page number which is incremented automatically.

T

Total records are written at total time in the logic cycle. They typically contain totals accumulated from one or more detail records.

Example

Figure 9-4 shows heading, detail, and total records as they appear on a report. The first two lines are heading records containing the date, a report title, a page number, and column headings. The detail records are the unshaded lines containing the division, department, name, and days absent for each employee. Total records are shaded and start with DEPARTMENT and DIVISION. They list the total number of days employees were absent within each department and division.

```
DATE   09/15/88    ABSENTEE REPORT                PAGE NO. 1


DIVISION    DEPARTMENT     EMPLOYEE          DAYS OUT

01          11             BABCOCK, J. D.       2
01          11             JONES, F. D.         1
01          11             MASON, M. M.         1

DEPARTMENT  11             TOTALS               4

01          12             MOONEY, J. P.        3
01          12             POMEROY, G.          2
01          12             RASKIN, K. T.        1

DEPARTMENT  12             TOTALS               6
.            .               .                  .
.            .               .                  .
.            .               .                  .

DEPARTMENT  20             TOTALS              10

DIVISION     1             TOTALS              53
```

Figure 9-4. Output Record Types As They Appear on a Report

Record Addition/Deletion (Columns 16-18)

The record addition/deletion field specifies whether the record is to be added to the output file (ADD) or deleted from it (DEL). Use this field only with output and update disk files.

Columns 16-18	Description
ADD	Add this record to an update or output file.
DEL	Delete this record from an update file. Use DEL with KSAM and TurboIMAGE files only.
blank	For sequential or KSAM files, write this record to the beginning of the file; for TurboIMAGE files, insert this record in the file; for update files, update the current record.

ADD

When you use ADD with MPE sequential and KSAM files, you must enter ADD on each record in the file. You must also enter A in the File Addition Field (column 66) of the File Description Specification for the file. (If you omit A for KSAM files, records are added to the file in primary key sequence.)

New records are appended to MPE sequential files. They are added to KSAM and TurboIMAGE files in primary key sequence.

Fetch Overflow/Release (Column 16)

The fetch overflow/release field lets you print overflow lines ahead of pending detail and total lines when the overflow line is reached (Fetch Overflow). It also lets you end the processing of a file before the program terminates normally (Release File).

Column 16	Description
blank	Do not write overflow records at this point, and do not release the file.
Fetch Overflow:	
F	Write overflow records at this point (by fetching the RPG Overflow Routine).
Release File:	
R	Release the file (close, or close and reopen the file, as shown in Table 9-1).

F (Fetch Overflow)

When the overflow line is reached, this option immediately suspends the printing of pending detail and total records for the file until all overflow records (those conditioned by overflow indicators) are printed. Use this field only for print files and files whose output is controlled by Line Counter Specifications and that contain an indicator in the Overflow Indicator Field (columns 33-34) of the File Description Specification.

Normally, when the overflow line is reached, pending detail and total records for the file are printed and overflow processing begins. Overflow records are printed and the paper is advanced to top-of-form. When you use this option, the overflow routine is "fetched" immediately; pending detail and total lines are printed after the overflow records. RPG does not automatically advance the paper to top-of-form after printing the overflow lines. You must specify this in the Skip Field (columns 21-22) of a specification that is conditioned by an overflow indicator.

If you're using OR lines and want to use Fetch Overflow in each of them, enter F in each OR line.

See the description of the RPG logic cycle in the *HP RPG Programmer's Guide* for details about Fetch Overflow processing.

R (Release File)

The release file option lets you release the file associated with this record after the record is written. If the record has no fields, the file is released without writing a blank line (or record). This option is similar to the CLOSE Calculation Specification operation.

The Release File option works differently for certain file types. Table 9-2 describes the actions that are performed for them.

Table 9-2. Actions Performed - Release File

Type of File	Actions
Print file: Any update, combined, or output file whose File Organization (column 32 of the File Description Specification) is not I, S, or M; and that uses space and/or skip entries (columns 17-22) or that is opened at run time with the CCTL (Carriage Control) option of the MPE FILE command; and whose Record Addition/Deletion Field (columns 16-18) is blank.	The file is closed and then reopened (the output spoolfile is released for printing and a new one is created). If a Print file is redirected to disk with an MPE file equation (or with DISK in the Device Class Name Field, columns 40-46, of the File Description Specification), the disk file is closed and then reopened in append mode. All output is concatenated into one disk file. Create the disk file using the CCTL (Carriage Control) option of the operating system FILE command. This enables you to print the file at a later time (using FCOPY, for example) with forms control.
WORKSTN file.	The WORKSTN file is closed and not reopened. This releases the terminal from block mode operation, allowing the program to continue terminal I/O in non-block mode. (This operation is equivalent to using the Calculation Specification CLOSE operation.)
Any other file.	The file is closed and not reopened. This operation is the same as the Calculation Specification CLOSE operation.

Space (Columns 17-18)

The space field lets you space up to three lines before and after printing the record. Use this field only for printer or terminal records or records controlled by Line Counter Specifications.

Column	Description
Before (17):	
blank	Do not space a line before printing. If this is a terminal file, or if the Skip Field (columns 19-22) is blank, single space before printing.
0	Do not space a line before printing. If this is terminal file, single space before printing.
1	Space one line.
2	Space two lines.
3	Space three lines.
After (18):	
blank	Do not space a line after printing. If the Space and Skip Fields (columns 17-22) are blank, single space after printing each record.
0	Do not space a line after printing.
1	Space one line.
2	Space two lines.
3	Space three lines.

When you space beyond the overflow line, normal overflow processing takes place (see the previous field, Fetch Overflow/Release (column 16) for a description of overflow processing).

The Skip Field (columns 19-22) lets you space to a certain line number or printer carriage control channel. When you use this field and the Skip Field in the same specification, spacing is performed in this order:

1. Skip before printing.
2. Space before printing.
3. Skip after printing.
4. Space after printing.

If you use this field and the Skip Field (columns 19-22) for a file, the file is assumed to be a print file and is written with forms control information (such as auto page eject). Additionally, to be considered a print file, the file must be an output, update, or combined file whose File Organization Field (column 32 of the File Description Specification) is not Indexed (I, S, or M) and that is blank in the AND/OR Field (columns 14-16).

The following actions may take place for print files:

- The file is opened using the CCTL (Carriage Control) option. (See the CCTL option of the MPE FILE command.)
- Forms positioning is performed if requested by column 41 of the Header Specification.
- Post spacing and auto page eject directives are set in the output spoolfile.
- Initial page eject is performed unless suppressed by columns 41-47 of the Header Specification.
- Carriage Control (CCTL) directives are added to the output records.
- Line counting and channel operations are performed according to the Carriage Control Field (column 53) of the Header Specification.
- Spacing and skipping operations are performed according this field and the Skip Field (columns 19-22).
- Release file operations are performed if you enter R in the Fetch/Overflow Release Field (column 16): close the output spoolfile and then open a new one to receive additional output. (Even if a file does not meet the criteria for print files, you can perform these release file operations by entering an MPE FILE command with the CCTL option.)

Skip (Columns 19-22)

The skip field lets you skip to a particular line or logical printer channel before and after printing the record.

Column	Description
Before (19-20):	
01-12	Skip to this line number before printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
	Skip to this Channel Number before printing when the Carriage Control Type Field (column 53) of the Header Specification is blank.
13-99	Skip to this line number before printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
A0-A9	Skip to this line (100-109) before printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
B0-B2	Skip to this line (110-112) before printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
blank	Do not skip lines.
After (21-22):	
01-12	Skip to this line number after printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
	Skip to this Channel Number after printing when the Carriage Control Type Field (column 53) of the Header Specification is blank.
13-99	Skip to this line number after printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
A0-A9	Skip to this line (100-109) after printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
B0-B2	Skip to this line (110-112) after printing when the Carriage Control Type Field (column 53) of the Header Specification contains L or 1.
blank	Do not skip lines.

When skipping to printer channels, you should use a Line Counter Specification to define those channels. If you do not, the line number positions for the channels are computed as follows: line number = channel number times 5 (Channel 1 is always line 6). For instance, if you enter 10 in this field, the printer paper is advanced to line 50 (10 x 5).

If you skip beyond the overflow line (but not to a new page), the overflow indicator (if used) is turned ON and normal page overflow processing is performed. (See the Fetch Overflow/Release Field (column 16) for details about overflow processing.) Do not skip beyond the printer forms length defined in the Line Counter Specification.

When you use this field, the file is normally assumed to be a print file. See the Space Field (columns 17-18) for more information on the processing that occurs for print files.

Output Indicators (Columns 23-31)

The output indicators field specifies the indicators that must be ON or OFF for the record to be written. You can condition an entire record or a field in the record using this field. To condition a record, enter the indicators in the record description line (see the Record Description Fields). To condition a field, enter the indicators in the field description line (see the Field Description Fields).

Column	Description
23, 26, 29:	
N	Write this record only if its corresponding indicator (in columns 24-25, 27-28, or 30-31) is OFF.
blank	Write this record only if its corresponding indicator (in columns 24-24, 27-28, or 30-31) is ON.
24-25, 27-28, 30-31:	
01-99	A general indicator set by a previous Calculation Specification operation or that identifies a record type or field.
F0-F9	A function key indicator previously set as a general indicator by the RPG VPLUS Interface or by a DSPLY or DSPLM operation.
H0-H9	A halt indicator.
KA-KN,\KP-KY	A command key indicator set previously as a general indicator or that was set previously by the RPG Screen Interface.
L0-L9	A control-level indicator (L0 is always ON).
LR	The last-record indicator.
MR	The matching-record indicator.
OA-OG, OV	An overflow indicator assigned to this file.
U1-U8	A user indicator (normally set before the program is executed).
1P	The first-page indicator.
blank	Do not use an output indicator.

You can use one or more AND and OR lines when you need to condition the record with more than three indicators (see the AND/OR Field, columns 14-16).

You cannot use AND or OR lines to condition a field. If you need to condition a field with more than three indicators, consolidate them to three by using the SETON Calculation Specification operation. For example, suppose you want to suppress the printing of a field when any one of five possible error indicators (02, 04, 06, 08, and 10) are ON. Use the SETON operation to turn indicator 12 ON when indicators 02, 04, and 06 are ON. Then condition the field with indicators 12, 08, and 10.

N (Not)

The not value specifies that, for the record or field to be written, the indicator in the associated field (columns 24-25, 27-28, or 30-31) must be OFF.

If this is a Header or Detail record (the file type is H or D), make sure that at least one indicator does not have an N associated with it; otherwise, the record is written at the beginning of the logic cycle after records conditioned by the 1P indicator. (This happens because all indicators are OFF at the beginning of the logic cycle.)

01-99 (General Indicators)

General indicators identify input record types in a file and show the results of Calculation Specification operations. They are the most frequently-used indicators.

F0-F9 (Function Key Indicators)

You can use function key indicators the same way you use general indicators.

Function key indicators have special meanings when used with the RPG VPLUS Interface. They are used by VPLUS to signal "events" that take place at the user terminal. When the user presses Enter, F0 is turned ON. When the user presses f1 to f8, the corresponding function key indicator is turned ON. When an event 9 or greater takes place, function key indicator F9 is turned ON. See Chapter 10 for a complete discussion of the RPG VPLUS Interface.

Function key indicators F1-F8 also have special meanings when used in conjunction with the SET, DSPLY, and DSPLM Calculation Specification operations (see these operations in Chapter 8 for details).

H0-H9 (Halt Indicators)

Halt indicators are normally used to print error messages and to halt the program at the end of the current cycle (before standard RPG error processing begins). You can also use them to write records before the standard error processing begins.

KA-KN, KP-KY (Command Key Indicators)

You can use command key indicators the same way you use general indicators.

When you use an RPG Screen Interface (RSI) file, the command keys may have a special meaning. A user at a terminal keyboard presses f1 followed by a key from the top row of the keyboard to turn on one of the twenty-four command key indicators. The RPG Screen Interface then performs the appropriate action. You enable the command key indicators when you build the screen forms file. You can use command keys that have not been enabled the same way you use general indicators. See Chapter 11 for information on the RPG Screen Interface and the *RPG Utilities Reference Manual* (SIGEDITOR) for information on creating an RSI forms file.

L1-L9, L0 (Control-Level Indicators)

Control-level indicators are turned ON when they are assigned to input fields and control breaks occur in those fields. Use them with fields and records that you want to write when control breaks occur. The L0 indicator is always ON. Use it for records or fields that you want to write at total time regardless of whether a control break occurs.

If you enter both a control-level and an overflow indicator, the record is written when the overflow line is reached. If this is a total record (the Type Field, column 15, contains a T) and it is not conditioned by a overflow indicator, the record is written after the last record in the control group is processed. If this is a detail record (the Type Field contains a D) and it is not conditioned by an overflow indicator, the record is written only after the first record of the new control group is processed.

LR (Last-Record Indicator)

This indicator is turned ON when there are no more input records to process. During the next logic cycle, all output records conditioned by LR are written provided the other indicators that condition the output are also satisfied.

OA-OG, OV (Overflow Indicators)

Overflow indicators are turned ON when the overflow line (as specified by the Line Counter Specification) is encountered. When it is ON, records conditioned by it are written (provided the records satisfy the other indicators entered in this field).

Use only one overflow indicator per file. You should also enter this indicator in the Overflow Indicator Field (columns 33-34) of the File Description Specification. If you do not use an overflow indicator, the paper is advanced automatically to a new page when the end-of-page is reached. If some records or fields in a file do not use the assigned overflow indicator but specify skipping to a new page (see the Skip Field (columns 21-22), the overflow indicator is turned OFF before the paper advances.

When you use an overflow indicator, you normally enter a Line Counter Specification to define the overflow line and other printer line positions.

You can enter an overflow indicator in an AND or OR line. However, be sure that there is only one overflow indicator that satisfies the AND or OR relationship. The indicator must be defined in the Overflow Indicator field of the File Description Specification.

Be careful when using an overflow indicator and a record-identifying indicator in an AND relationship. If overflow occurs but the current record type is not the one specified by the record-identifying indicator, no output is produced. Therefore, if possible, use overflow and record-identifying indicators in OR relationships when conditioning record output.

Do not use an overflow indicator to condition output of exception records (those that have an E in the Type Field, column 15); however, you can use this indicator to condition exception record fields.

U1-U8 (User Indicators)

These indicators, when turned ON and OFF automatically by the Job Control Word (JCW), let you condition the output of an entire file. Otherwise, you can use them like general indicators to condition the output of records and fields (you must turn them ON yourself via the SETON Calculation Specification operation).

1P (First-Page Indicator)

The first-page indicator is normally used to print headings on the first page of a report. The headings are printed during pre-cycle processing. You can also use this indicator, together with an overflow indicator, to print the headings on each page of the report. (Use the 1P indicator for this purpose only when no other indicators are available.)

Use the 1P indicator only with heading and detail records (see the Type Field, column 15). Do not use it with total or exception records. Do not use 1P with other indicators (except the user indicators, U1-U8) when conditioning output (if you precede it with N, you can enter other indicators preceded by N).

Since 1P lines are written during pre-cycle processing, be careful that they do not contain data derived from input or that are calculated by the program. You can use 1P to condition lines containing the predefined fields: PAGE, PAGE1 - PAGE7, UDATE, UDAY, UMONTH, UYEAR. You can also use 1P with compile-time and preexecution time array elements and User Data Structure fields.

Example

Figure 9-5 shows an output record that is conditioned by two general indicators. When indicator 20 (only) is turned ON, the fields JOB, EXP, and PAY are written. When indicators 20 and 30 are turned ON, fields NAME, JOB, EXP, and PAY are written. When indicator 20 is turned OFF, no fields are written.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

OPRINTX  D 1      20
O                 30      NAME      10
O                         JOB       20
O                         EXP       30
O                         PAY       40
```

Figure 9-5. Using Output Indicators

Field Description Fields (Columns 32-70)

The field description fields describe the data fields within the records defined by the Record Description Fields (columns 7-31).

Enter fields on separate lines, starting with the line following the Record Description line. Leave columns 7-22 blank. You only need to define those fields that you want to write in the program. See Figure 9-2 for an example of how to enter Field Description Fields.

You can enter indicators in the Output Indicators Field (columns 23-31) to condition individual fields in the output record.

Field Name (Columns 32-37)

This field contains the name of the field, table, array, or array element to be written.

Columns 32-37	Description
Field name previously defined in the program. (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. (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.
Array name. (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 follows the rules for array names defined above. The index is either a number or the name of a field (see Field name above) that contains a number. The array name/comma/ind combination is limited to six characters.	The name and index of the output array element.
EXCPT (Exception) Name. The name used in the Result Field of an EXCPT Calculation Specification operation. EXCPT Names follow the rules for Field names (see above). They cannot be the same as: an array, data structure, field, file label, subroutine, or table name.	The name that identifies this exception record line.

Columns 32-37	Description
PAGE, PAGE1-PAGE7	A predefined field that contains the current page number for print files.
*PLACE	A predefined field that repeats the output fields defined previously within the record description.
UDATE	A predefined field that contains the current date.
UDAY	A predefined field that contains the current day of the month.
UMONTH	A predefined field that contains the current month.
UYEAR	A predefined field that contains the current year.
*ERROR	A predefined field used for run-time error codes.
blank	A value is entered in the Constant/Edit Word Field (columns 45-70).

Field Name, Table Name, Array Name, EXCPT Name

Except for update files, you must name each field that you want to write. For update files, you do not have to name fields that are unchanged from their input records. Do not use this field if you enter a constant in the Constant/Edit Word Field (columns 45-70).

Be sure that the names you enter are defined in a File Description, File Extension, or Input or Calculation Specification.

You can enter names in any order, since their positions in the output record are determined by the End Position Field (columns 40-43). Fields are written in the order of their specifications, so that if one field overlaps another, the first field is partially or totally overwritten.

For signed numeric fields, the sign is written as part of the low-order character. Unless you edit the field using the Edit Code Field (column 38), this character is a letter. For instance, -3 is printed as L and +3 is printed as C. An unsigned field that is not a result field, is printed without a sign.

PAGE

This predefined field contains a number that is written whenever PAGE is used. You may use PAGE, for example, to print page numbers in a report heading. You can also use it to stamp records with a number. Normally, PAGE is set to zero during pre-cycle processing and incremented by one before it is used in an Output Specification.

PAGE is a four-digit field with no decimal positions. You can redefine it so that is contains from 1-15 digits, if you wish. Do this by entering PAGE in an Input Specification and defining it with a different field length in the From Field Position and To Field Position Fields. In Calculation Specifications, use an End Position value that reflects this new size. When you redefine PAGE, do not specify decimal positions. Whether or not you redefine PAGE, leading zeros are suppressed and no arithmetic sign appears unless you use an edit word or edit code when printing.

You can reset the page number to 1 by using the Blank After Field (column 39) with PAGE. You can restart the page-numbering sequence by conditioning the PAGE specification with an indicator (when the indicator is turned ON, PAGE is set to zero, then incremented by one before printing). To start paging with a value other than 1, define PAGE in an input record and ensure that the field contains the starting page number minus 1.

PAGE1-PAGE7

These predefined fields provide up to seven additional PAGE fields (PAGE1 through PAGE7). Normally, you use only one PAGE field per file. If you use the same PAGE field for more than one file, be sure that it will produce the correct numbers for all of them.

*PLACE

*PLACE repeats one or more fields in a record. This lets you output the fields one or more times but code them only once in the program. Put *PLACE after the fields you want to repeat on output. Enter the last position to be occupied by the repeated fields in the End Position Field (columns 40-43). The last position must be at least twice the value of the end position of the last repeated field. If you do not allow enough space, some or all of the previous fields are overlaid.

When *PLACE is encountered, all of the preceding fields in the record are written until the character position specified by the End Position Field (columns 40-43) is reached. The fields are written starting with their beginning positions. All characters are written in the same relative positions.

Each time you want a field or group of fields repeated, you must enter *PLACE in a separate specification. Two consecutive *PLACE entries repeat the initial fields four times.

You can condition *PLACE specifications with one or more output indicators (see the Output Indicators Field, columns 23-31).

Example

The contents of four output fields (DATA1, DATA2, DATA3, and DATA4) are shown below (the character ␣ is a blank):

Field Name:	Contents:
DATA1	␣␣AAA
DATA2	␣␣BBB
DATA3	␣␣CCC
DATA4	␣␣DDD

Figure 9-6 shows how to use *PLACE to produce the output shown below. The fields, DATA1, DATA2, and DATA3 are repeated in the output record. Since DATA4 follows *PLACE, it is not repeated.

```
␣␣AAA␣␣BBB␣␣CCC␣␣AAA␣␣BBB␣␣CCC␣␣DDD
```

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------

          OPRINTR  H  4     01
          O                                    60 "SPECIAL REPORT"
          O
          O        D  1     02
          O                        DATA1       5
          O                        DATA2      10
          O                        DATA3      15
(1)       O                        *PLACE     30
          O                        DATA4  B   35
```

Figure 9-6. Using *PLACE

UDATE, UDAY, UMONTH, and UYEAR

These fields let you include the current date, in various formats, in the utput record. Table 9-3 shows what each of the formats look like.

Table 9-3. Editing Date Fields

Field Name	Contents	Unedited Example	Edited Example	Description
UDATE	Current date.	101188	10/11/88	October 11, 1988 (Domestic Format)
		111088	11/10/88	October 11, 1988 (United Kingdom Format)
		111088	11.10.88	October 11, 1988 (European Format)
UDAY	Current day of month.	11	11	11th day
UMONTH	Current month.	10	10	October
UYEAR	Current year.	88	88	1988

The date that UDATE produces depends on the entry in the Inverted Print Field (column 21) of the Header Specification. You can produce the date in European, United Kingdom, and Domestic formats. Additionally, you can further edit the date by entering Y in the Edit Code Field. This edit code inserts slashes or periods (see the fourth column in Table 9-3). Unedited, the UDATE field is 6 characters long; edited, it is 8.

You cannot change the contents of any of these date fields. Their values are set during pre-cycle processing and are not altered during program execution. The date comes from the operating system unless the UDATE Source Field (column 17) of the Header Specification contains an F. In this case, the date comes from the RPGUDATE file.

*ERROR

When the H0 indicator is turned ON, RPG places a unique character that corresponds to the error in the field *ERROR. (See Appendix B for the values that are placed in *ERROR for various run-time errors.) *ERROR is predefined as a 1-character alphanumeric field.

When an error does not terminate the program, you can interrogate *ERROR to determine the cause for the error. If you want to use *ERROR as a normal data field, enter it in the Field Name Field of an Input Specification defining it as a 1-character field. You can use *ERROR in the Factor 1, Factor 2, and Result Fields of a Calculation Specification. When you use *ERROR with the RLABL Calculation Specification operation, the name passed to the subroutine is ERROR, not *ERROR (you must use ERROR when referencing it).

Edit Code (Column 38)

The edit code field lets you edit numeric output fields. You can suppress leading zeros and arithmetic signs and you can insert characters.

Column 38	Description
1	Insert commas and do not zero suppress.
2	Insert commas and zero suppress.
3	Do not insert commas or zero suppress.
4	Zero suppress and do not insert commas.
A	Insert commas and do not zero suppress. For negative fields, append the CR (credit) sign.
B	Insert commas and zero suppress. For negative fields, append the CR (credit) sign.
C	Do not insert commas or zero suppress. For negative fields, append the CR (credit) sign.
D	Zero suppress and do not insert commas. For negative fields, append the CR (credit) sign.
J	Insert commas and do not zero suppress. For negative fields, append the - (negative) sign.
K	Insert commas and zero suppress. For negative fields, append the - (negative) sign.
L	Do not insert commas or zero suppress. For negative fields, append the - (negative) sign.
M	Zero suppress and do not insert commas. For negative fields, append a - (negative) sign.
X	Remove the plus sign from the units position of the field; do not remove a minus sign or suppress leading zeros; do not print decimal points. If the Sign Processing Field (column 40) of the Header Specification is blank or contains an S, the plus sign is suppressed.

Column 38	Description

Y	Insert slashes or periods into numeric fields (normally used for date fields).
	Slash marks can be used with fields having three to six digits as follows:
	`Field length:` `Edited output:`
	`3` `nn/n`
	`4` `nn/nn`
	`5` `nn/nn/n`
	`6` `nn/nn/nn`
	If the first digit is zero, it is suppressed. If the Inverted Print Field (column 21) of the Header Specification contains an I or J, a period is inserted instead of a slash.

Z	Suppress leading zeros and do not print a decimal point (if specified). Remove the arithmetic sign in the units position.
blank	No edit code applies.

1-4, A-M

These edit codes insert characters, such as a comma and the CR (credit) sign, into numeric fields. For example, if you enter the edit code A and the field is negative, a CR (for credit) is appended to it. If you enter the edit code J, a minus sign is appended.

All of the edit codes suppress leading zeros (zeros to the left of the decimal point, preceding non-zero digits) unless you enter a J in the Inverted Print Field (column 21) of the Header Specification. J specifies that the edited result is in European Format; zero suppression stops with the units position of the number. For example, the number .04 is shown as 0,04 and a zero is shown as 0,00 (the comma is the decimal position).

If the edit code specifies zero suppression and the field contains decimals and is not zero, the decimal point is printed. If the field is zero, it is suppressed to blanks.

If the edit code does not specify zero suppression and the field contains decimals and is equal to zero, the decimal point is printed followed by the same number of zeros as there are decimal places. If there are no decimal places, a zero is printed in the units position of the field.

Make sure that you include character positions for the inserted characters when entering a value in the End Position Field (columns 43-43). If the field is not large enough, it will overlap another field.

You can use a floating dollar sign ($) or asterisk (*) in conjunction with these edit codes. Include the $ in an edit word (see the Constant/Edit Word Field, columns 45-70) to print the $ immediately to the left of the most significant digit. If you're suppressing zeros and the field is zero, the $ does not print. Include the * (check protection character) in an edit word to print the * in all zero suppressed positions to the left of the decimal point. If the number is zero, asterisks fill the entire field.

When you use an edit code to print an entire array, elements in the array are separated by two blanks.

Examples

Table 9-4 gives examples of various numbers and the effects that the edit codes have on them. The character ⊔ represents a blank. The right brace (}) is a zero with a negative sign.

Table 9-4. Effects of the Edit Codes

Edit Code	Positive Number, Two Decimal Positions	Positve Number, No Decimal Positions	Negative Number, Three Decimal Positions	Negative Number, No Decimal Positions	Zero Balance, Two Deciaml Positions	Zero Balance No Decimal Positions
	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⊔⊔	1,234,567⊔⊔	.120CR	120CR	.00⊔⊔	0⊔⊔
B	12,345.67⊔⊔	1,234,567⊔⊔	.120CR	120CR		
C	12345.67⊔⊔	1234567⊔⊔	.120CR	120CR	.00⊔⊔	0⊔⊔
D	12345.67⊔⊔	1,234,567⊔⊔	.120CR	120CR		
J	12,345.67⊔	1,234,567⊔	.120-	120-	.00⊔	0⊔
K	12,345.67⊔	1,234,567⊔	.120-	120-		
L	12345.67⊔	1234567⊔	.120-	120-	.00⊔	0⊔
M	12345.67⊔	1234567⊔	.120-	120-		
X	1234567	1234567	00012}	00012}	000000	000000
Z	1234567	1234567	120	120		

The lines below illustrate how the Y edit code works. Assuming that the number to be edited is 120188, the results for 3, 4, 5, and 6-character Result Fields are:

```
12/0, 12/01, 12/01/8, 12/01/88
```

Blank After (Column 39)

The blank after field resets the contents of a field to blanks or zeros after it is written.

Column 39	Description
B	Reset the contents of the field after output. Reset numeric fields to zero and alphanumeric fields to blanks.
blank	Do not reset the contents of the field after output.

This field is often used to clear control totals when control breaks occur. Subtotals can be cleared after being rolled forward to the next control level total field. For example, if B is assigned to a field that is conditioned by control-level indicator L2, it is cleared after being printed for a level 2 control break.

Do not use B with constants, look-ahead fields or the date fields UDATE, UDAY, UMONTH, or UYEAR.

If there is an input field with the same name as you enter in this specification, it is cleared also. If you use the same field name more than once in the Output Specifications, enter B only on the last one. Otherwise, the field is reset before all the output is written.

If you enter B for a table, only the entry found in the last LOKUP operation is reset. If no LOKUP has been performed, the first entry is reset.

If you enter a B into this field and the Indicator Setting Field (column 42) of the Header Specification is blank or S and you enter an indicator in the Output Indicators Field (columns 23-31), the indicator is turned ON after the field is cleared to zeros or blanks.

End Position (Columns 40-43)

The end position field specifies the last position of the output field. If this specification follows the record description specification for an RSI WORKSTN file, this field may specify the length of the RSI form name.

Columns 40-43	Description
1-9999 (right-justified, leading zeros are not required)	The rightmost character position of the output field.
blank	The compiler calculates the end position of the output field. (This is called the relative end position.)
± 1-999	Add or subtract this number from the computed end position (relative end position) of the output field.
K and RSI form name length	The specification contains the name of an RSI WORKSTN form. Enter K in columns 40, 41, or 42. Enter the length (1-8) of the RSI form name in column 43.

1-9999

To explicitly enter the end position yourself, enter a number. The number is the low-order (rightmost) character position of the field. For print fields, do not enter an end position that exceeds the printer line length. For disk files, do not enter an end position that exceeds the number of characters in the record. A field's length is calculated by subtracting the previous field's end position from the value entered in this field. (The first field in a record begins in position one). If the field is edited, be careful to include enough space for insertion characters as well as digits.

When entering the end position for *PLACE, use a number that is at least double the end position of the last field that is repeated. If you do not, data is lost due to field overlap.

The Packed/Binary Field (column 44) determines how numeric fields are written. To provide enough space for an output field, follow these guidelines:

If the format of the field that you're writing is:	The output field size should be:
Alphanumeric.	The same as the number of characters in the field.
Unpacked (ASCII) decimal without a leading or trailing sign.	The same as the number of digits in the field.
Unpacked (ASCII) decimal with a leading or trailing sign.	One byte longer than the number of digits in the field. For example, for an 8-digit number, use a field length of 9.
Packed decimal.	One-half the number of digits in the field plus 1.
Binary.	The number of digits in the field divided by 2.5. For example, a 5-digit number requires 2 positions and a 10 digit number requires 4 positions.

Example

Example Conventions

Figure 9-7 shows how to enter end positions for fields in two output records. The fields in both records are the same. The spacing between them is different. The fields in the disk record (DISCREC) are adjacent to each other while the fields in the print record (REPORT) are separated by intervening spaces.

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
_____________________________________________________________________

ODISCREC D        01
O                         QTY        5
O                         PROD      13
O                         DESC      38
O                         COST      41P
OREPORT  D        01
O                         QTY   1    6
O                         PROD      18
O                         DESC      47
O                         COST  J   58
O                                   63 "***"
```

Figure 9-7. Entering End Positions

Blank

The end position of the field is calculated by the compiler. It is calculated by adding this field's edited length to the previous field's end position. If the end position is blank, the compiler calculates it. (The first character of each output record is assumed to be 1.)

You can use the RSPACE option of the $CONTROL compiler subsystem command to provide fixed spacing between fields.

± 1-999

When you enter a plus or minus sign followed by a number, you're specifying the number of spaces between this field and the previous one. A plus sign adds spaces between fields and a minus sign causes the fields to overlap. For example, -␣␣3 causes the field to overlap the previous field by three characters; +002 leaves two spaces between the fields.

This option overrides the RSPACE option of the $CONTROL compiler subsystem command for this field.

Example

Figure 9-8 shows how to use relative end positions with $CONTROL. This example produces the same result as the Output Specifications shown in Figure 9-7.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

ODISCREC D        01
O                         QTY
O                         PROD
O                         DESC
O                         COST        P
$CONTROL RSPACE=4
OREPORT  D        01
O                         QTY   1
O                         PROD
O                         DESC
O                         COST  J
O                                 +  2 "***"
```

Figure 9-8. Using Relative End Positions with $CONTROL

Figure 9-9 shows what the compiled Output Specifications for Figure 9-8 look like. The compiler prints the end positions as if you entered them. An R at the end of each line indicates that the End Position Field is a relative end position; that is, it is calculated by the compiler. The +002 at the end of the last line indicates that there are two spaces between the field COST and the last Output Specification field.

```
ODISCREC D        01
O                         QTY        5                          R
O                         PROD      13                          R
O                         DESC      38                          R
O                         COST      41P                         R
$CONTROL RSPACE=4
OREPORT  D        01
O                         QTY   1    6                          R
O                         PROD      18                          R
O                         DESC      47                          R
O                         COST  J   58                          R
O                                   63 "***"                    +002
```

Figure 9-9. How Relative End Positions Appear in a Compiler Listing

K and RSI Form Name Length

Enter a K in columns 40, 41, or 42 to indicate that this specification names an RSI WORKSTN file. Enter the number of characters (1-8) in the RSI form file name in column 43.

Enter the actual form name as a constant in the Constant/Edit Word Field (columns 45-70) or enter a field in the Field Name Field (columns 32-37) that holds it.

Example

In Figure 9-10, the form, FM01, is displayed when indicator 61 is ON. When indicator 62 is ON, the form contained in the field FORM is displayed.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

OSCREEN  E        61
O                                       K4 "FM01"
O                         DATE  Y    8
O                         FLD1      14
O                         FLD2      20
O        E        62
O                         FORM      K8
O                         DATE  Y    8
O                         FLD3      16
O                         FLD4      24
```

Figure 9-10. Entering an RSI Form Name

Packed/Binary (Column 44)

The packed/binary field specifies the format in which numeric fields are written. Do not use this field for fields that you're going to edit.

Column 44	Description
2	Write the field in binary format, two bytes long.
4	Write the field in binary format, four bytes long.
8	Write the field in binary format, eight bytes long.
B	Write the field in binary format. A field of one to five digits is written as two bytes, a field of six to ten digits is written as four bytes, and a field of eleven to fifteen digits is written as eight bytes.
L	Write the field in unpacked format with an arithmetic sign (plus or minus) preceding the field.
P	Write the field in packed decimal format.
R	Write the field in unpacked format with an arithmetic sign following the field.
blank	This is an unpacked numeric or alphanumeric field, a numeric field containing data to be edited or a constant.

2, 4, 8

A 2-byte binary field can hold a number up to 2^{15}, a 4-byte binary field can hold a number up to 2^{31}, and an 8-byte binary field can hold a number up to $2^{49.8}$. The compiler warning 7057W is issued when a 5-digit number is written to a 2-byte binary field, or when a 10-digit number is written to a 4-byte binary field.

L, R

Enter an L or R for a field when you want the sign to precede or follow the number. (Normally, the sign appears in the low-order position.) Be sure to allow room for the sign when you enter the End Position Field (columns 40-43).

Constant/Edit Word (Columns 45-70)

The constant/edit word field lets you use either a constant or an edit word when formatting a numeric field for output.

Columns 45-70	Description
From 1 to 24 characters (any letters, digits, or special characters) surrounded by quotation marks.	A constant (if a field name does not appear on this line) or an edit word (if a field name does appear on this line). If there is an edit code in the Edit Code Field (column 38), the Constant/Edit Word Field may contain a $ (floating dollar sign) or a * (check protection character).
blank.	No constant is used or no edit word applies.

Constants

A constant is a group of characters that are written exactly as you enter them. You frequently use constants for report titles, page and column headings, and record-identifying information. You often use them in heading records.

To enter a constant:

- Leave the Field Name Field (columns 32-37) blank.
- Begin the constant by entering a quotation mark in column 45.
- Enter the characters in the constant. You can use up to 24 ASCII characters (the last character can be in any column from 46 to 69). If the constant is longer than 24 characters, enter another constant line containing the remaining constant characters. You can continue a constant in this manner using as many of these lines as necessary.
- End the constant with a quotation mark. The quotation mark can appear in any column up to 70.
- To include a quotation mark in the constant itself, enter two quotation marks. For example, to enter the constant, A PROGRAM NAMED “ALPHA”:

```
"A PROGRAM NAMED ""ALPHA"""
```

- Enter the last position occupied by the constant in the End Position Field (columns 40-43).

Example

Figure 9-11 shows several constants. The constant defined in line 1 contains quotation marks. The constant starting in line 2 is continued to line 3.

```
          1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------------------------

          OPRINTT  H  3     01
(1)       O                                    50 "REPORT ON ""PROSPECTS"""
          O        H  3     02
          O                                     9 "LAST NAME"
          O                                    20 "FIRST NAME"
          O                                    25 "AGE"
          O                                    35 "OCCUPATION"
(2)       O                                    50 "PERSONNEL EVALUATION BAS"
(3)       O                                    72 "ED ON BACKGROUND CHECK"
```

Figure 9-11. Entering Constants

Edit Words

Edit words let you insert and replace characters in numeric output fields. They provide greater flexibility than edit codes see the Edit Code Field, column 38). You can specify the format of the output field, character by character. You can insert commas, decimal points, dollar signs, and other punctuation into the field. For negative fields, you can insert a minus sign or CR and you can use a floating dollar sign and an asterisk for check protection.

When you enter an edit word, you enter characters that become the editing template for the field. Edit words contain replaceable and non-replaceable characters. Non-replaceable characters are inserted into the field. If they are to the left of the high-order digit position, they are replaced by blanks or asterisks (if check protection is used). If they follow the last replaceable digit position, they are appended to it. You can use any character as a non-replaceable character except blank (to use blank, enter an ampersand). Replaceable characters are place-holders for digits in the field and they are replaced by them. Replaceable characters include a blank, 0, * and $. Do not include more than 15 replaceable characters in an edit word.

To enter an edit word:

- Enter the name of the numeric field to be edited in the Field Name Field (columns 32-37).
- Leave the Edit Code Field (column 38) blank.
- Enter the last position of the edited field in the End Position Field (columns 40-43).
- Begin the edit word by entering a quotation mark in column 45.
- Enter the characters in the edit word. See Table 9-5 for a description of the characters that you can include. Enter up to 24 ASCII characters with the last character placed on or before column 69.
- End the edit word with a quotation mark.

Table 9-5. Edit Word Characters

Edit Word Characters	Description
blank	Replace a 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 (check protection).
$	If $ is used to the left of a zero, replace leading zeros with spaces and float the dollar sign to the left of the most significant digit in the field.
&	Insert a blank into the edited field.
. or ,	Insert a decimal point or comma into the edited field.
""	Insert a quotation mark in the edited text.
+, - or CR	Insert an arithmetic sign into the edited field or insert a CR into the edited field when the number is negative.
other character	Insert the character into the edited field at this character location.

0

Normally, leading zeros in a data field are replaced by blanks on output. You can stop suppression at any point by entering a zero as a replaceable character in the edit word. The zero is replaced by a character from the corresponding position of the source data field, unless that character is also zero (in which case a blank is substituted). Any zeros in the source data field that appear to the right of the zero suppression character are printed unaltered. If a non-zero digit appears to the left of the zero, suppression stops with that non-zero digit.

If you want leading zeros in an edited field, ensure that the edit word contains at least one more character than the data field, and enter a zero in the leftmost position of the edit word.

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 required by the edit word.

*

Use an asterisk to halt zero suppression and to replace the suppressed zeros with asterisks (check protection). Place the asterisk in the position where you want zero suppression to stop. If an asterisk precedes a zero in an edit word, suppression stops with the asterisk and the zero is printed as a constant.

$

Enter a floating dollar sign to the left of the character position where you want to stop zero suppression. Zeros to the left of the dollar sign are suppressed and the dollar sign is placed immediately to the left of the first non-zero digit. Do not use a floating dollar sign together with check protection. Allow an additional character position in the edit word for the floating dollar sign (the number of replaceable characters must equal or exceed the number of digits in the number).

To print a dollar sign in the leftmost position of the field, enter it in the leftmost position of the edit word.

&

Use an ampersand to produce a blank in the edited field (the ampersand is replaced by a blank).

. or ,

Enter a decimal point or comma in an edit word to insert those characters in the corresponding positions of the edited result. If they precede the most significant digit in the field, they are replaced by blanks (or asterisks if you're using check protection).

""

To include a quotation mark in the edited result, enter two adjacent quotation marks.

+, - or CR

To insert the numeric sign of the field, enter a plus or minus sign to the right of the low-order digit position in the edit word. If you enter a negative sign, the sign is printed only when the field is negative. Likewise, if you enter a positive sign, the sign is printed only when the field is positive. Enter a credit sign to print CR when the field is negative. If the field contains all zeros (either positive or negative) the sign or CR is suppressed to blanks.

Other character

Enter any ASCII character (not listed above) to insert into the field.

Example

Several edit word examples appear in Table 9-6. The first column shows the numbers to be edited (the numbers are positive and negative). The second column shows the edit words and the third column shows the edited results. ␣ represents a blank.

Table 9-6. Examples of Edit Words

Input	Edit Word	Edited Result
0042	"0␣HRS.␣␣MINS.&0""CLOCK"	␣0HRS.42MIN.␣0"CLOCK
000000	"␣,␣␣0.␣␣"	␣␣␣␣␣.00
000000	"␣,␣␣␣.0␣"	␣␣␣␣␣␣␣0
000000	"␣,␣0␣DOLLARS␣␣CENTS&CR"	␣␣␣␣0DOLLARS00CENTS␣␣␣␣
+000000	"␣,␣␣␣DOLLARS␣␣CENTS"	␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣CENTS
+000002	"␣␣0LBS.&␣␣0Z.TARE&-"	␣␣␣␣0LBS.␣02OZ.TARE
-000002	"␣␣␣␣0LBS.␣␣0Z.TARE&-"	␣␣␣␣␣LBS.02OZ.TARE␣-
001234	"0,␣␣␣,0␣␣"	␣,012,034
013579	"&␣,*␣0,␣␣␣"	***130,579
093066	"␣␣-␣␣-␣␣&LATER"	␣9-30-66␣LATER
093066	"␣␣&␣␣&␣␣&LATER"	␣9␣30␣66␣LATER
100166	"␣␣/␣␣/␣␣"	10/01/66
00000000	"␣␣,␣␣␣,␣0␣-&ON&HAND"	␣␣␣␣␣␣␣␣␣0␣␣␣ON␣HAND
-00000000	"␣␣␣,␣␣0.␣␣&CR*"	␣␣␣␣␣␣␣.00␣␣␣*
00123456	"␣,␣␣$,0␣␣.␣␣"	␣␣␣1$,234.56
000000005	"␣,␣␣␣,␣0␣.␣␣-"	␣␣␣␣␣␣␣␣0.05␣
-000000015	"␣,␣␣␣,␣␣␣.␣␣-"	␣␣␣␣␣␣␣␣␣␣15-
095140036	"0␣␣-␣␣-␣␣␣␣"	␣95-14-0036
0000000000	"␣␣,␣␣␣,␣0␣*␣␣"	␣␣␣␣␣␣␣␣␣0*00
0000000000	"$␣␣,␣␣␣,␣␣0.␣␣"	$␣␣␣␣␣␣␣␣␣.00
0000000000	"␣␣,␣␣␣,␣␣␣.␣␣CR"	␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
0000000000	"␣␣␣␣␣␣␣␣␣␣"	␣␣␣␣␣␣␣␣␣␣
+0000000000	"␣,␣␣␣,␣␣␣,␣0␣-OLD&BALANCE"	␣␣␣␣␣␣␣␣␣␣␣␣␣0␣OLD␣BALANCE

Table 9-6. Examples of Edit Words (continued)

Input	Edit Word	Edited Result
-0000000000	"␣␣,␣␣␣,␣␣*.␣␣-"	**********.00␣
-0000000000	"␣␣␣␣␣␣␣␣*␣␣&CR"	********00␣␣␣
-0000000000	"␣␣␣␣␣␣␣␣␣␣*&CR"	**********␣␣␣
-0000000000	"␣␣␣␣␣␣␣␣␣␣$0"	␣␣␣␣␣␣␣␣␣␣$
-0000000000	"␣␣␣␣␣␣␣␣$0␣␣&CR&GROSS"	␣␣␣␣␣␣␣␣$00␣␣␣␣GROSS
-0000000000	"$␣␣␣␣␣␣␣␣0␣␣&CR&GROSS"	$␣␣␣␣␣␣␣␣00␣␣␣␣GROSS
-0000000000	"␣␣␣␣␣␣␣␣␣␣0&-&TOTAL"	␣␣␣␣␣␣␣␣␣␣␣␣␣␣TOTAL
-0000000000	"␣␣␣␣␣␣␣␣␣␣$-&TOTAL"	␣␣␣␣␣␣␣␣␣␣␣␣␣␣TOTAL
0000000005	"␣␣␣,␣␣␣,␣$9.␣␣"	␣␣␣␣␣␣␣␣␣␣$.05
0000000005	"␣␣␣,␣␣␣,&0␣.␣␣&NET"	␣␣␣␣␣␣␣␣␣$0.05␣.NET
+0000000005	"␣␣␣,␣␣␣,$0␣.␣␣-*"	␣␣␣␣␣␣␣␣␣$0.05␣*
-0000000005	"␣␣,␣␣␣,␣␣0.␣␣&CR"	␣␣␣␣␣␣␣␣␣.05␣CR
-0000000123	"␣␣␣,␣␣␣,␣␣␣.␣␣-"	␣␣␣␣␣␣␣␣␣␣1.23-
0000001234	"␣␣,␣␣*,␣␣␣*␣␣"	******,012*34
0000001234	"␣␣␣,␣$0,␣␣␣.␣␣-SALES"	␣␣␣␣␣␣&,012.34␣SALES
0000135678	"␣␣,␣␣␣,␣␣␣.␣␣&CR&-NET"	␣␣␣␣␣1,356.78␣CR␣␣NET
0000135678	"␣␣␣␣␣␣␣␣␣␣&&PROFIT"	␣␣␣␣135678␣␣PROFIT
0000135678	"$0␣␣␣␣␣␣␣␣␣-$NET"	$␣000135678␣␣NET
0000135678	"␣␣,␣␣␣,␣␣␣DOLLARS␣␣CENTS"	␣␣␣␣␣1,356DOLLARS78CENTS
+0000135678	"$␣␣␣␣␣␣␣␣␣␣&-&NET"	$␣␣␣␣135678␣␣␣NET
+0000135678	"␣␣␣␣␣␣␣␣␣␣&CR&NET"	␣␣␣␣135678␣␣␣␣NET
+0000135678	"0␣␣␣␣␣␣␣␣␣"	␣000135678
+0000135678	"␣␣␣␣␣␣␣␣␣␣"	␣␣␣␣135678
+0000135678	"$&0␣,␣␣␣,␣␣␣.␣␣&NET"	$␣␣0,001,356.78␣NET
+0000135678	"␣␣,␣␣␣,␣␣*.␣␣*CR**"	*****1,356.78*␣␣**
-0000135678	"*␣␣␣␣␣␣␣␣␣"	*000135678
-0000135678	"␣␣␣␣␣␣␣␣␣*"	****135678
-0000135678	"␣␣␣␣␣␣␣␣␣␣"	␣␣␣␣135678

Table 9-6. Examples of Edit Words (continued)

Input	Edit Word	Edited Result
-0000135678	"␣␣␣␣␣␣␣␣␣␣0"	␣␣␣␣135678
-0000135678	"␣␣␣␣␣␣␣␣␣␣␣&CR&NET"	␣␣␣␣135678␣CR␣NET
-0000135678	"␣␣␣␣␣␣␣␣␣␣␣&-&&NET"	␣␣␣␣135678␣-␣␣NET
0000135678	"␣␣␣␣␣␣␣␣␣␣␣&NET&CR"	␣␣␣␣135678␣NET␣␣␣
-0000135678	"␣␣␣␣␣␣␣␣␣␣␣&NET&CR"	␣␣␣␣135678␣NET␣CR
-0000135678	"$␣␣␣␣␣␣␣␣␣␣␣&-&NET"	$␣␣␣␣135678␣-␣NET
-0000135678	"␣␣␣␣␣␣␣␣&0␣␣&CR"	␣␣␣␣$135678␣CR
-0000135792	"␣␣,␣␣␣,␣*␣.␣␣&-"	*****1,357.92␣-
-0000135678	"␣␣,␣␣␣,␣␣␣.␣␣&CR&&NET"	␣␣␣␣␣␣1,356.78␣CR␣␣NET
-0001356789	"␣␣␣,␣␣␣␣&0.␣␣CR"	␣␣␣␣$13,567.89CR
-0034567890	"␣␣␣,␣␣␣,␣&0.␣␣CR**"	␣␣␣$345,678.90CR**
-1234567809	"␣␣␣␣␣␣␣␣&0␣␣&CR!"	$1234567809␣CR
+1234567890	"*␣␣␣␣␣␣␣␣␣␣"	1234567890
-1234567890	"␣␣␣,␣␣␣,␣&0.␣␣-"	$12,345,678.90-
-1234567890	$&␣␣,␣␣␣,␣0␣.␣␣&-&GROSS"	$␣12,345,678.90␣-␣GROSS
-1234567890	$&␣␣,␣␣␣,␣␣0.␣␣CR"	$␣12,345,678.90CR
-1234567890	"␣,␣␣␣,␣␣␣,␣␣␣-OLD&BALANCE"	1,234,567,890-OLD␣BALANCE

Comments (Columns 71-74)

Enter comments of any kind in the comments field.

Program Name (Columns 75-80)

The program name field contains the program name. The format of this field is discussed in Chapter 2.

The Output Specification Default Summary

If you leave the optional fields of the Output Specifications blank, the default specifications shown in Table 9-7 apply.

Table 9-7. Output Specification Defaults

Columns	Field	Default Values
1 - 5	Sequence Number	No sequence number applies.
16 - 18	Record Addition/Deletion	For sequential or KSAM output files, write new records to beginning of the file. For TurboIMAGE output files, insert new records in the file. For update files, update the current record.
16	Fetch Overflow/ Release	Do not fetch overflow or release the file.
17 - 22	Space and Skip	Space one line after printing each record.
23, 26, and 29	Not	Write the data defined on this line only if the indicator specified in the next field is ON.
24 - 25, 27 - 28, and 30 - 31	Output Indicators	Do not assign an indicator.
38	Edit Code	Do not use an edit code.
39	Blank After	Do not reset 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 - 70	Constant/Edit Word	No constant is used or no edit word applies.
71-74	Comments	No comments are used.
75-80	Program Name	None.

10

RPG Interface to VPLUS

The RPG interface to VPLUS provides functions similar to those provided by the VPLUS procedures for other languages. You can use the VPLUS interface to:

- Retrieve a form from a VPLUS forms file and display it at the terminal.
- Display a message in the window area of the terminal screen.
- Display initial field values specified when the form was created.
- Accept input from the terminal.
- Determine if the input fields contain errors and, if so, flag them and display an error message.
- Write data entered from the terminal to the user program or to a batch file. (A batch file lets you record screen data in a file for later use.)
- Transfer data from the program or batch file to the terminal screen.
- Transfer data between the program and batch file.

The RPG interface to VPLUS provides all of the facilities to read and write screen forms, edit data entered on the form and record that data in a screen transaction file (batch file). Figure 10-1 illustrates how VPLUS works with RPG.

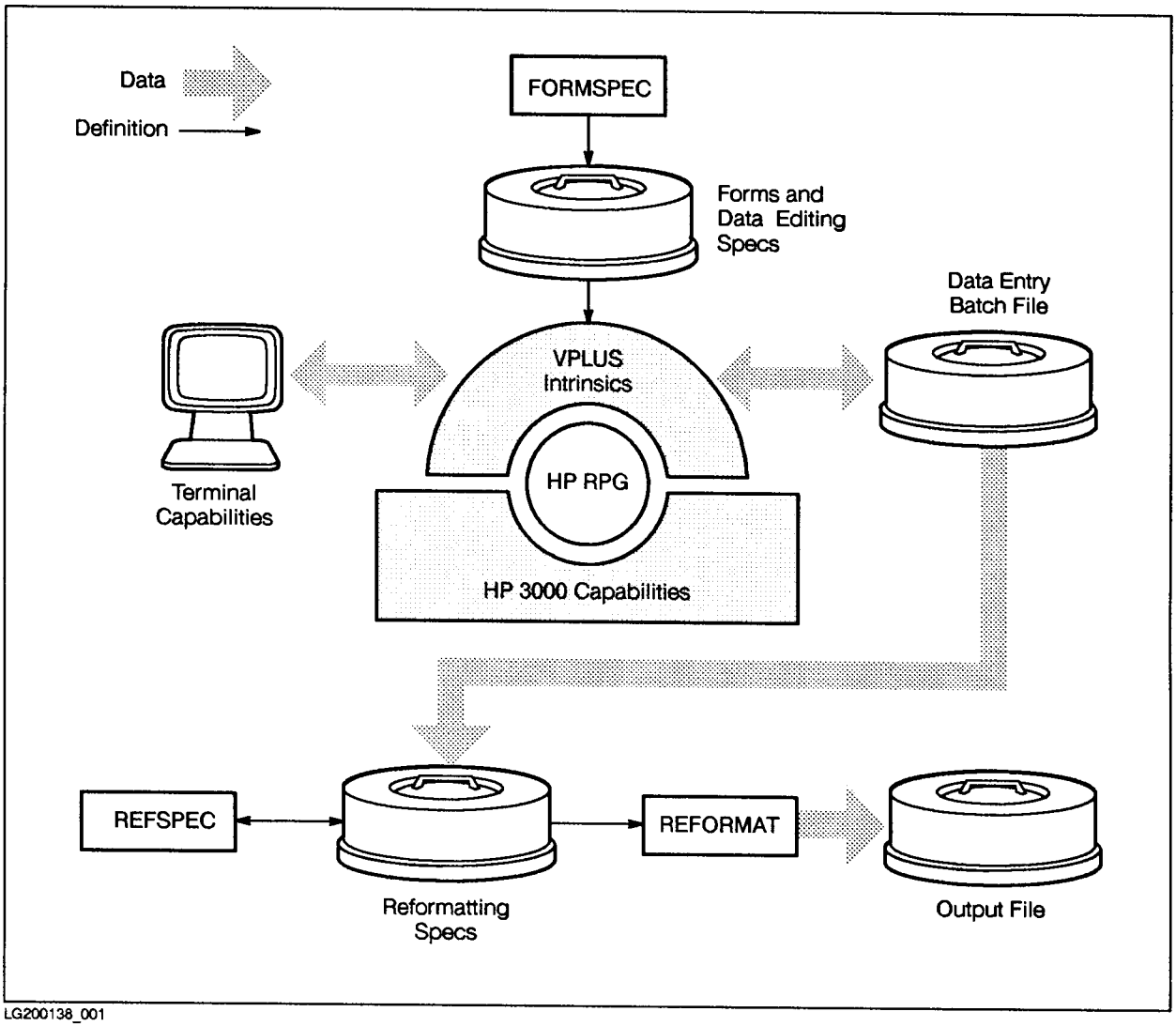


Figure 10-1. How VPLUS Works With RPG

For complete information about VPLUS, see the *Data Entry and Forms Management System VPLUS/3000* manual. This manual lists an RPG general purpose data entry program that uses VPLUS. Use that program along with the one at the end of this chapter to understand how to tailor a VPLUS program to meet your specific requirements.

Using the RPG Interface to VPLUS

To use VPLUS in an RPG program, perform the following steps (each of the steps is explained in detail in the following sections in this chapter):

1. Create the VPLUS forms file using the VPLUS utility, FORMSPEC.
2. Specify screen-related options using the Header Specification.
3. Specify screen-related options and define the VPLUS forms file using File Description Specifications. Optionally, you can use this specification to define VPLUS batch and trace files.
4. Enter Input Specifications to describe the records that VPLUS uses to report the results of screen operations to the program.
5. Enter Calculation Specifications to initiate VPLUS actions and to return the results of these actions. (Use Calculation Specifications with demand files only.)
6. Enter Output Specifications that describe the records that you use to initiate VPLUS screen operations.

When you use VPLUS, define one or more input records for VPLUS to use in communicating the results of its actions. You normally enter a separate input record for each result (*event*) that you expect for the screen functions that VPLUS performs. *Actions* direct VPLUS to perform some screen function such as displaying a form. To initiate an action, place an action code and, depending on the action, other information in the WORKSTN output record. If the WORKSTN file is a demand file, you must perform exception output to execute the action. To illustrate how action and events work, the following steps show how you commonly use a VPLUS form to read terminal data:

General screen procedure:	Action/event that performs the procedure:
Display an empty form.	Action: Get next form to display.
	Action: Display a form with initial data values.
Read input entered by the terminal operator.	Action: Read input from the terminal.
	Event: What was entered?
Perform VPLUS data edits.	Action: Perform VPLUS edits.
	Event: Did VPLUS edit errors occur?
Transfer data to the program buffer.	Action: Transfer data from the VPLUS buffer to the program buffer.
	Event: What is the data that was read?

You can perform input editing in the program or you can have VPLUS edit according to the edits entered when the VPLUS forms file was created by FORMSPEC (see the next section). When VPLUS edits data, it uses the VPLUS buffer. If you're doing the editing yourself, substitute your own edits for the VPLUS edits in Step 3 above.

Using FORMSPEC

Before you can use VPLUS to process data from the terminal, you must create the VPLUS forms file. The forms file defines the data fields on the screen and constants such as field names. Create the forms file using the VPLUS utility, FORMSPEC. FORMSPEC is described in the *Data Entry and Forms Management System VPLUS/3000* manual.

Entering the Header Specification

The next two sections describe the Header Specification facilities (run-time errors and the Error Dump) that function differently when used with the RPG interface to VPLUS.

Handling Run-Time Errors

Normally, you determine how run-time errors are handled by your entries in the Error Log Field (column 55) and the Error Response Field (columns 56-71) of the Header Specification. When you're using the RPG interface to VPLUS, however, error messages are displayed in the message window and the terminal operator selects a response by pressing the appropriate function key. The function keys and their corresponding responses are shown in Table 10-1.

Table 10-1. Entering Error Responses Using the Function Keys

Function Key	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.

Requesting an Error Dump

The Error Dump is normally written to the standard list device, $STDLIST. When using VPLUS, the terminal is in session mode and $STDLIST is the terminal.

Since the Error Dump is of limited use when displayed on the terminal, you may want to direct it to a file. Enter the name of the file in Error Dump File Name Field (columns 7-14) of the Header Specification.

Entering File Description Specifications

The next six sections explain how to define VPLUS files and how RPG screen-related options work with the RPG interface to VPLUS.

Defining VPLUS Files

To use VPLUS in an RPG program, you must define a terminal WORKSTN file. (You can use only one WORKSTN file in a program.) Use this file for all communication with VPLUS.

To define a WORKSTN file, enter a file type of update in the File Type Field (column 15) of the File Description Specification. Describe the records as variable in the Logical Record Length Field (columns 24-27). Enter WORKSTN in the Device Class Name Field (columns 40-46) for the file.

Usually, a VPLUS WORKSTN file is a demand file, but you may define it as a primary file in the File Designation Field (column 16) of the File Description Specification. Primary files are processed by the RPG logic cycle. An update primary file may be used to advantage, for example, when processing is limited to a single cycle that includes displaying a form, reading data from it and writing data to it. You may find it more convenient and flexible to define the WORKSTN file as an update demand file. In this case, processing is performed by Calculation Specifications which use demand READs and exception output.

There are two other files that you may use with a WORKSTN file: a batch file and a trace file. (These files are optional.) You can use batch files for saving screen data; data is automatically written to the batch file from the VPLUS buffer, one record per form. You can use the reformatter facility (REFSPEC) of VPLUS to rearrange the form fields so that the batch file can be used by other applications. To use a batch file, enter a BATCH File Description Continuation line (BATCH in the Option Type Field, columns 54-59). Trace files are handy for finding program errors. Each action and event (see the "VPLUS Action Codes" and "VPLUS Event Codes" sections) causes at least one record to be written to the file. You can examine the trace file to find errors. Run-time error messages are written to the trace file and are displayed at the terminal. If the trace file becomes full, the program continues although tracing stops. To use a trace file, enter a TRACE File Description Continuation line for it.

Specifying the Error Message Display Interval

Run-time error messages are displayed on the terminal for 3 seconds. You can change this interval by entering a number from 0-9 (0 suppresses the message) in the Interface Control Field (column 51) of the File Description Specification. The display interval applies only to WORKSTN errors. It does not apply to RPG errors, VPLUS edit checking errors or to messages placed directly in the window by the PUTMSG and SHOMSG actions (see the "VPLUS Action Codes" section).

Enabling the BREAK Key

On some terminals, the BREAK key is physically positioned near the ENTER key. As a result, it is easy to press BREAK by accident. Since it is difficult to recover from a break, the BREAK key is disabled when a WORKSTN file is being used.

If you need to use the BREAK key, enable it by entering a B in the Interface Control Field (column 52) of the File Description Specification for the WORKSTN file.

Enabling the Function Key Labels

You can define eight function key labels for a VPLUS form. To display the labels at run time, enable them by entering an L in the Interface Control Field (column 50) of the File Description Specification.

Downloading VPLUS Forms

When using terminals that allow forms storage, you can download VPLUS forms. This greatly improves the speed with which screens are displayed. Normally, only one form can be downloaded. You can increase this number by entering a FORMDL File Description Continuation line that specifies this number.

You can download forms in one or more of the following ways:

Preload — The form is displayed using the SHOW action (with the preload option specified). If the form is not already in terminal memory, it is placed there. If you use SHOW without the preload option, the form is displayed directly to the terminal.

You cannot download forms with the SHODTA action; instead, enter a PUTDTA action followed by a SHOW action with the preload option specified.

Look-ahead — The next form, named in the forms file or in a GETNXT action, is loaded before or after the current form is read depending on the data communications in use. If point-to-point is used, the next form is loaded before the current form is read. If multipoint is used, the next form is loaded after the current form is read. Nothing happens when the form is already in memory.

This downloading method is always in effect unless you enter N for the look-ahead option of the RDTERM action.

Load-forms — The LOADFM action loads the forms contained in the array named in the LOADFM File Description Continuation line. Use LOADFM before entering a SHOW or SHODTA action. Forms are loaded in the order specified by the array, until no more terminal memory is available. When this happens, remaining forms are ignored.

Using the STATUS Array

The STATUS array (declared in a STATUS File Description Continuation line) is predefined as a 6-element, 10-digit array with zero decimal places. VPLUS returns the error status in the first element of the array. This status comes from the first word of the VPLUS COMAREA. When you use the LOADFM action, the number of downloaded forms is returned in the second element of the array.

Example

Figure 10-2 shows how to enter a File Description Specification and its Continuation lines to define a VPLUS forms file and the options relating to it. The File Description Specification indicates that the WORKSTN file (TRANSFIL) is an update demand file and that the function keys are used as well as the BREAK key. The FORMS File Description Continuation line (line 2) names the VPLUS forms file to be processed. The BATCH Continuation line (line 3) names the file (BATCHB) that contains data entered from the terminal. The TRACE Continuation line (line 4) names the file (TRACEFL) that logs VPLUS actions, events, and run-time errors. The STATUS Continuation line (line 5) names the array that contains the VPLUS status information. The FORMDL Continuation line (line 6) specifies that there are 3 forms to download into terminal memory. The LOADFM Continuation line (line 7) names the array that contains the form names.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

1    FTRANSFILUD  V      80            WORKSTN   L5B
2    F                                             KFORMS FORMA
3    F                                             KBATCH BATCHA
4    F                                             KTRACE TRACEFL
5    F                                             KSTATUSSARRAY
6    F                                             KFORMDL3
7    F                                             KLOADFMFMARRAY
```

Figure 10-2. Entering VPLUS File Description Specifications

Entering Input Specifications

When VPLUS performs an action, it places the outcome of the action in an input record. Each outcome has a unique event code. You can examine this code and direct processing accordingly. The next two sections describe the VPLUS event codes and the input record formats for each of them.

VPLUS Event Codes

When the VPLUS Interface reports an event, it places the event code in the first field of the WORKSTN input record.

Event codes 00-08 indicate that the terminal operator pressed a function key. Event code 00 indicates that the operator entered data and pressed (ENTER). In response to events 01-08, you must enter an action. Event codes 09-12 are responses to previous actions in the program. (These actions use the VPLUS data buffer and normally take full advantage of the VPLUS data handling features.)

Table 10-2 lists all of the event codes that are returned by the RPG VPLUS Interface.

Table 10-2. VPLUS Event Codes

Event Code	Description	Returned in Response to this Action Code
00	The terminal operator pressed (ENTER). Data is returned in the record.	54 (RDTERM)
01	The terminal operator pressed (f1). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
02	The terminal operator pressed (f2). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
03	The terminal operator pressed (f3). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
04	The terminal operator pressed (f4). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
05	The terminal operator pressed (f5). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
06	The terminal operator pressed (f6). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
07	The terminal operator pressed (f7). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
08	The terminal operator pressed (f8). No data is returned unless F is used with the RDTERM action.	54 (RDTERM)
09	Read the number of fields that failed VPLUS or user edits.	59 (EDITS) or 61 (NUMERR)
10	Read data from the VPLUS data buffer (data is included in the record).	64 (GETDTA)
11	Return the record number of the current batch record, the mode of operation (0= collect, 1= browse), the repeat/append status, the freeze/append status the next form name.	Any action (or no action) except 54, 59, 61, 64, and 74.
12	Return the length and contents of a particular field in the VPLUS data buffer.	74 (GETFLD)

VPLUS Input Record Formats

Depending on the event, input records have one of four possible formats (as shown in Table 10-3). Each of them contains a unique code for the event and the current form name in the first two fields. (The current form name lets you associate the data with the form in which it is entered.)

Table 10-3. VPLUS Input Record Formats

Event Code	Input Field	Input Field Description
00-08, 10	Event code (1-2)	00-08 and 10.
	Current form name (3-17)	
	Data length (18-21)	The total number of characters required by all fields in the current form.
	Data (22-end of record)	This field is separated into sections according to the definition of the current form. Only events 0 and 10 use this field.
09	Event code (1-2)	09.
	Current form name (3-17)	
	Number of errors (18-22)	The total number of fields containing errors. The errors are detected either by VPLUS (using FORMSPEC edit checks) or by edits in the program.
11	Event code (1-2)	11.
	Current form name (3-17)	
	Batch record number (18-22)	When returned at the beginning of a program, this number is the end-of-batch +1. It is useful in determining what the upper bounds is for an existing batch file.
	Mode (23)	`0` Collect mode. `1` Browse mode (the mode of operation that allows data in the batch file to be examined and modified; new batch records cannot be added).
	Repeat/Append (24)	This field is 0, 1, or 2.
	Freeze/Append (25)	This field is 0, 1, or 2.
	Next form name (26-40)	
	Number of non-deleted batch records (41-45)	

Table 10-3. VPLUS Input Record Formats (continued)

Event Code	Input Field	Input Field Description
12	Event code (1-2)	12.
	Current form name (3-17)	
	Field number (18-22)	This field contains a unique identifier for the field being retrieved. It does not change if the field position in the form changes.
	Field length (23-26)	
	Data (27-end of record)	This field contains the data in the returned record.

Example

Figure 10-3 shows how to define an input record for event 09 which reads the number of fields that failed the VPLUS edits.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

I*** NUMBER OF EDIT ERRORS   (09)
I        CC  19   1 C0   2 C9
I                                          3  17 FORMB
I                                         18  22ONUMERR      01
```

Figure 10-3. Entering VPLUS Input Specifications

Entering Calculation Specifications

For demand WORKSTN files, you display a VPLUS form and read the data entered into it at the terminal by entering Calculation Specification operations. Initiate a VPLUS action by placing its action code or mnemonic in a WORKSTN output record, then perform exception output to execute the action. When an action results in more than one VPLUS response, you must retrieve the event code for them in an input record. The next two sections explain how to return an event and how to start an action.

Initiating VPLUS Actions

To perform a VPLUS action (see the action codes in Table 10-4) place the action code or mnemonic in the first positions of the WORKSTN output record then perform exception output.

Example

Figure 10-4 shows how to perform the VPLUS edits (EDITS (59) action) specified in FORMSPEC. (Alternatively, "59 " could be moved to ACTION in line 1 instead of "EDITS ".)

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________

(1)       C   10                MOVEL"EDITS "  ACTION
          C   10                SETON                      35    EDITS - 59
          C   10                EXCPT
```

Figure 10-4. Initiating a VPLUS Action Using Calculation Specifications

Returning VPLUS Events

You learn of an event by performing a READ Calculation Specification operation for the WORKSTN file. The data returned for the event is either from the terminal or is returned in response to a previous action code. Data that is returned is placed in an input record defined for the WORKSTN file.

Example

Figure 10-5 shows how to return the number of edit errors encountered by VPLUS during an EDITS (59) action (see Figure 10-4).

```
     1         2         3         4         5         6         7
6789012345678901234567890123456789012345678901234567890123456789012345
-----------------------------------------------------------------------

C   10                READ TRANSFIL                 HO
```

Figure 10-5. Returning a VPLUS Event Using Calculation Specifications

Entering Output Specifications

The next two sections describe the VPLUS action codes and how to define the output records used with them.

VPLUS Action Codes

The first field in the WORKSTN output record is the VPLUS action code (or action mnemonic). Table 10-4 lists all of the action codes that you can use with the RPG VPLUS Interface.

If you're doing your own input editing (not using VPLUS FORMSPEC edits), you only need to use actions 50-57. If you want VPLUS to perform input editing, or if your screen application is more complex, you can exercise the full capabilities of VPLUS by using any of the actions codes listed in Table 10-4.

Table 10-4. VPLUS Action Codes

Action Code	Mnemonic	Description	Corresponding VPLUS Intrinsic
50	CHGNXT	Specify the next form name and whether the form is repeat/append, freeze or normal.	None
51	GETNXT	Get the next form from the forms file, and set repeat/append and freeze/append status. If repeat mode is set, the form is not retrieved. Follow this action by SHOW (action 53) to display the form at the terminal.	VGETNEXTFORM
52	PUTMSG	Specify a message (and any enhancements) to display in the error/status line of the terminal. The message is displayed only when SHOW (action 53) is executed.	VPUTWINDOW
53	SHOW	Display the current form, any initial data and any message on the terminal screen.	VSHOWFORM
54	RDTERM	Read terminal input into the VPLUS data buffer. Follow this action by a READ Calculation Specification operation to transfer the data to your program. (The returned record will be one of the event types 00-08.) If VPLUS performs input editing, follow RDTERM with EDITS (action 59). Note that a form must be displayed before RDTERM will wait for a user response.	VREADFIELDS VGETBUFFER

Table 10-4. VPLUS Action Codes (Continued)

Action Code	Mnemonic	Description	Corresponding VPLUS Intrinsic
55	SHOMSG	Display a message in the program (with any enhancements) in the error/status line of terminal. If data in the VPLUS buffer has changed, the new data is displayed. This action is a combination of PUTMSG (action 52) and SHOW (action 53).	VPUTWINDOW VSHOWFORM
56	CORERR	Identify the field that failed user edits. The output record contains the field number and a message to display. The terminal user's response is read. Follow this action with RDTERM (action 54) to read the user response (event types 00-08). This action is a combination of BADFLD (action 62), SHOW (action 53), and RDTERM (action 54).	VSETERROR VSHOWFORM VREADFIELDS VGETBUFFER
57	SHODTA	Display data from the program buffer to fields on the screen. This action is a combination of PUTDTA (action 63) and SHOW (action 53).	VPUTBUFFER VSHOWFORM
58	INIT	Initialize fields in the current form according to defaults specified in FORMSPEC. If there are errors, move a message to the window.	VINITFORM VERRMSG VPUTWINDOW
59	EDITS	Perform the edits for the fields in the current form. If there are any errors, display a message in the terminal window for the first error. Follow this action with a READ Calculation Specification operation. The record that is returned (event 09) contains the number of fields the failed the edit.	VFIELDEDITS VERRMSG VPUTWINDOW
60	PRINT	Print the current form (with its data) on the line printer. Before you this action, read the form with GETNXT (action 51).	VPRINTFORM
61	NUMERR	Return the error status. Follow this action by a READ Calculation Specification operation to access the status. The record returned (event 09) contains the number of fields that failed the edits.	None
62	BADFLD	Indicate that a field failed a user edit. The output record contains the number of the field and a message for the window.	VSETERROR

Table 10-4. VPLUS Action Codes (Continued)

Action Code	Mnemonic	Description	Corresponding VPLUS Intrinsic
63	PUTDTA	Replace the data in the VPLUS data buffer with the data that you specify. The output record contains the new data for the VPLUS buffer and the length of that data.	VPUTBUFFER
64	GETDTA	Move the data in the VPLUS data buffer to the program buffer. Follow this action by a READ Calculation Specification operation to access the data (event 10).	VGETBUFFER
65	FINISH	Perform the final processing specified for the finish phase of the form. If there are errors, move a message to the window.	VFINISHFORM VERRMSG VPUTWINDOW
66	WRTBAT	Move the contents of the VPLUS data buffer to the batch record corresponding to the current record number. If collect mode is in effect the current record number is incremented by one. If in browse mode, the current record number is unaffected.	VWRITEBATCH
67	PREV	Read data from the previous batch record to the VPLUS data buffer. Place the program in browse mode, if it is not already in that mode. Save the current location in the batch file and the current form name.	VREADBATCH
68	REREAD	Move the data from the current batch record into the VPLUS data buffer. The program must be in browse mode (this mode is set by PREV, action 67).	VREADBATCH
69	NEXT	Read data from the next batch record into the VPLUS data buffer. The program must be in browse mode (this mode is set by PREV, action 67).	VREADBATCH
70	RESUME	Return from browse mode to collect mode (PREV, action 67, must have been used previously). The next form name and the location in batch file are restored.	none
71	DELETE	Delete the current batch record. The program must be in browse mode (see PREV, action 67).	None

Table 10-4. VPLUS Action Codes (Continued)

Action Code	Mnemonic	Description	Corresponding VPLUS Intrinsic
72	RDBTNU	Read the batch record identified by its record number.	None
73	CLRMSG	Clear the message window and, optionally, the message buffer.	VPUTWINDOW VSHOWFORM
74	GETFLD	Locate the specified field in the VPLUS data buffer. Follow this action by a READ Calculation Specification operation. The returned record (event 12). contains the field and its length.	VGETFIELD
75	PUTFLD	Transfer data from the program buffer to the specified field in VPLUS data buffer.	VPUTFIELD
76	LOADFM	Download forms to terminal memory.	VLOADFORM
77	UNLDFM	Unload (remove) the named form from the "form storage directory", thus releasing space in terminal memory.	VUNLOADFORM
78	CHMODE	Set terminal to character mode without affecting the screen display. Useful for writing escape sequences to slaved printers on MPE/iX systems.	VTURNOFF
79	BLMODE	Set terminal to block mode (restore original environment after CHMODE).	VTURNON

VPLUS Output Record Formats

The output records for actions must contain the action code (or action mnemonic). Depending on the action that you use, you may need to enter other fields as well. The output record formats and the fields used with them are described in Table 10-5.

Table 10-5. VPLUS Output Record Formats

Action Code	Output Field	Output Field Description
50(CHGNXT)	Action code/Mnemonic (1-6)	50 in columns 1-2 or CHGNXT in columns 1-6.
	Next form name (7-21)	
	Repeat/Append(22)	"0" - Normal sequence. "1" - Repeat current form. "2" - Append current form to self.
	Freeze/Append(23)	"0" - Clear screen. "1" - Append next form to current form. "2" - Freeze current form and append next form to it.
52(PUTMSG) 55(SHOMSG)	Action code/Mnemonic(1-6)	52 or 55 in columns 1-2; or PUTMSG or SHOMSG in columns 1-6.
	Message length (7-8)	This field specifies the number of characters in the message.

	Enhancement code (9)	This field specifies the display enhancement for the field:
		
		`@, A-O`
		See WINDOWENH in the *Data Entry and Forms Management System V PLUS/3000* manual.
		
		`zero`
		No enhancement.
		
		`blank`
		Do not change existing enhancement.

	Message (10-end of record)	

Table 10-5. VPLUS Output Record Formats (Continued)

Action Code	Output Field	Output Field Description
53(SHOW)	Action code/Mnemonic (1-6)	53 in columns 1-2 or SHOW in columns 1-6.
	Preload option (7)	`P` If the form is not already loaded into terminal memory, it is loaded before being displayed.
		`blank` The form is not loaded into terminal memory before being displayed.
54(RDTERM)	Action code/Mnemonic (1-6)	54 in columns 1-2 or RDTERM in columns 1-6.
	Function key options (7)	`F` Return screen data with events 01-08 when a function key is pressed. Data is always returned with event 00 (when ENTER is pressed).
		`blank` Return screen data with event 00, but not with events 01-08.
	Look-ahead options (8)	`N` Disable the forms downloading look-ahead option.
		`blank` Enable the forms downloading look-ahead option.

Table 10-5. VPLUS Output Record Formats (Continued)

Action Code	Output Field	Output Field Description
56(CORERR) 62(BADFLD)	Action code/Mnemonic (1-6)	56 or 62 in columns 1-2; or CORERR or BADFLD in columns 1-6.
	Field number (7-11)	This field identifies the field in error so that you can enhance it.
	Message length (12-13)	This field specifies the number of characters in the message which is to be displayed in the message window. The length should not exceed 79 displayable characters.
	Enhancement code (14)	This field specifies the display enhancement for the field in error: `@,A-0` See SINDOWENH in the *Data Entry and Forms Management System V PLUS/3000* manual. `zero` No enhancement. `blank` Do not change existing enhancement.
	Message (15-end of record)	
57(SHODATA) 63(PUTDTA)	Action code/Mnemonic (1-6)	57 or 63 in columns 1-2; or SHODTA or PUTDTA in columns 1-6.
	Data length (7-10)	The total number of characters required by all fields in the current form.
	Data (11-end of record)	This field is separated into sections according to the definition of the current form. The program buffer may contain data from a previous action or event. To clear the buffer before using this action, enter ADD in columns 16-18 of the first Output Specification for this action. Also enter A in column 66 of the File Description Specification for the WORKSTN file.

Table 10-5. VPLUS Output Record Formats (Continued)

Action Code	Output Field	Output Field Description
60(PRINT)	Action code/Mnemonic (1-6)	60 in columns 1-2; or PRINT in columns 1-6.
	Print control (7)	`Y` Underline the data field. `blank` Do not underline the data field.
	Page control (8)	`+` Do not space a line. `-` Triple space. `0` Double space. `1` Skip to the next page. `%` Use the control code in columns 9-11. `R` No CCTL; release FORMLIST for printing immediately after this PRINT action. `blank` Single space.
	Carriage Control Codes (9-11)	Enter an octal code (from %0-%377). These codes are described under FWRITE in the *MPE/iX Intrinsics Reference Manual*.
72(RDBTNU)	Action code/Mnemonic (1-6)	72 in columns 1-2 or RDBTNU in columns 1-6.
	Batch record number (7-11)	
73(CLRMSG)	Action code/Mnemonic (1-6)	73 in columns 1-2 or CLRMSG in columns 1-6.
	Clear buffer option (7)	`I` Clear the message from the screen and also from the message buffer. `blank` Clear the message from the message buffer.

Table 10-5. VPLUS Output Record Formats (Continued)

Action Code	Output Field	Output Field Description
74(GETFLD) 75(PUTFLD)	Action code/Mnemonic (1-6)	74 or 75 in columns 1-2; or GETFLD or PUTFLD in columns 1-6.
	Field number (7-11)	This field identifies the field to read or write. The number is assigned by FORMSPEC and does not change even when the field is moved to another position on the screen.
	Field length (12-15)	This field specifies the number of characters in the field.
	Data (16-end of record)	For PUTDTA, this field contains the data record to be written to the VPLUS buffer.
76(LOADFM)	Action code/Mnemonic (1-6)	76 in columns 1-2 or LOADFM in columns 1-6.
	Number of forms (7-9)	This field contains the number of forms to load into terminal memory from the LOADFM File Description Continuation line array. The number should not exceed the number declared in the FORMDL File Description Continuation line. Forms are loaded in the order specified in the LOADFM array until terminal memory is exhausted; then excess forms are ignored.
	or	RPG creates the LOADFM array as an unsequenced alphanumeric compile-time array with an element length of 16 (15 characters for the form name and a blank). The number of elements is determined by the FORMDL line. If you want to enter the form names at run time, rather than at compile time, enter an ** line for the array at the end of the source program. Instead use Input or Calculation Specifications to load the names. If you're using the STATUS array with VPLUS, the second element in it contains the actual number of forms loaded.
	Form name (7-21)	This field names the form to be loaded into terminal memory. If you're using the STATUS array with VPLUS, the second element in it contains the actual number of forms loaded.

Table 10-5. VPLUS Output Record Formats (Continued)

Action Code	Output Field	Output Field Description
77(UNLDFM)	Action code/Mnemonic (1-6) Form name (7-21)	77 in columns 1-2 or UNLDFM in columns 1-6. This field names the form to be removed from the forms storage directory. This releases space in terminal memory. (Forms that have not recently been used are automatically removed to make room for new ones.)
Other actions	Action code/Mnemonic (1-6)	51 in columns 1-2 or GETNXT in columns 1-6. 58 in columns 1-2 or INIT in columns 1-6. 59 in columns 1-2 or EDITS in columns 1-6. 60 in columns 1-2 or PRINT in columns 1-6. 61 in columns 1-2 or NUMBERR in columns 1-6. 64 in columns 1-2 or GETDTA in columns 1-6. 65 in columns 1-2 or FINISH in columns 1-6. 66 in columns 1-2 or WRTBAT in columns in 1-6. 67 in columns 1-2 or PREV in columns in 1-6. 68 in columns 1-2 or REREAD in columns 1-6. 69 in columns 1-2 or NEXT in columns 1-6. 70 in columns 1-2 or RESUME in columns 1-6. 71 in columns 1-2 or DELETE in columns 1-6. 78 in columns 1-2 or CHMODE in columns 1-6. 79 in columns 1-2 or BLMODE in columns 1-6. For NEXT and PREV, if you try to read past either end of a batch file, the read is not actually executed. The current record number is set to out-of-bounds (end of batch + 1 for NEXT and - 1 for PREV). Aside from the current record number, the program has no other indication of an out-of-bounds condition.

Example

Figure 10-6 shows how to enter an output record to start the PUTDTA action (63).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

O        E        37
O                                      6 "PUTDTA"
O                          DATALN     10
O                          DATA       67
```

Figure 10-6. Entering VPLUS Output Specifications

Sample VPLUS Program

This section shows a program that uses the RPG VPLUS Interface. The program is a typical data entry program. Data collected from it is written to a batch file that can be used by other programs and applications. The program, shown in Figure 10-7, does the following:

- Displays a VPLUS form with its initial values.
- Reads data entered into the form by the terminal operator.
- Performs VPLUS and program edits on the input data.
- Writes the edited data to a VPLUS batch file.

The File Description Specifications define the WORKSTN file, the VPLUS file in which the forms are stored, the batch file where the edited data is written and the trace file where a log of the actions and events is written.

The Input Specifications define the input records for events 00 (the ENTER key), 01-08 (function keys f1 through f8), and 09 (the number of fields with edit errors). This program treats events 01-07 as if they were event 08. Data is not returned by events 01-08; they signal that a function key is pressed and cause the program to go to the label EXIT.

The Calculation Specifications perform VPLUS actions using exception output. Events are returned using demand READs. The program ends when f8 is pressed.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

(1)  HDUMPFILE

(2)  FTRANSFILUD  V      80            WORKSTN   L5B
(3)  F                                             KFORMS FORMA
(4)  F                                             KBATCH BATCHA
(5)  F                                             KTRACE TRACEFL

(6)  I*** ENTER KEY (00)
     I*
     ITRANSFILAA  10   1 C0   2 C0
     I       OR        1 C1   2 C0
     I                                      3  17 FORMB
     I                                     18  21 DATALN
     I                                     22  78 DATA
     I                                     22  41 NAME
```

Figure 10-7. A Program that Uses VPLUS

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

I                                       42  61 ADDR
I                                       62  71 CITY
I                                       72  73 ST
I                                       74  78 ZIP
7   I*** EXIT KEYS (01 - 08)
I        BB  18   1 C0   2 C1
I       OR        1 C0   2 C2
I       OR        1 C0   2 C3
I       OR        1 C0   2 C4
I       OR        1 C0   2 C5
I       OR        1 C0   2 C6
I       OR        1 C0   2 C7
I       OR        1 C0   2 C8
8   I*** NUMBER OF EDIT ERRORS   (09)
I        CC  19   1 C0   2 C9
I                                        3  17 FORMB
I                                       18  22ONUMERR      01

9   C*  GET NEXT FORM
C           START     TAG
C                     MOVEL"GETNXT"  ACTION  6
C                     SETON                     35    GETNXT - 51
C                     EXCPT
10  C*  SET INITIAL VALUES
C                     MOVEL"INIT  "  ACTION           INIT - 58
C                     EXCPT
C                     SETOF                     35
11  C*  DISPLAY FORM
C           REPEAT    TAG
C                     SETOF                     01
C                     MOVEL"SHOW  "  ACTION
C                     SETON                     35    SHOW - 53
C                     EXCPT
12  C*  READ FROM TERMINAL
C                     MOVEL"RDTERM"  ACTION           RDTERM - 54
C                     EXCPT
C                     SETOF                     35
13  C*  READ RECORD
C                     READ TRANSFIL                 HO
C*** IF F1 - F8 EXIT ELSE CONTINUE
C   18                SETON                     LR
C   LR                GOTO EXIT                       EXIT IF F1 - F8
```

Figure 10-7. A Program that Uses VPLUS (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

(14) C*  IF ENTER - EDIT DATA
     C   10               MOVEL"EDITS "  ACTION
     C   10               SETON                     35    EDITS - 59
     C   10               EXCPT
     C   10               SETOF                     35
(15) C*  DETERMINE NO. OF ERRORS
     C   10               READ TRANSFIL                 HO
     C***  IF ERRORS - RETURN TO STEP 3
     C   01               GOTO REPEAT
     C*** IF NO ERRORS CONTINUE
     C*
(16) C*  TRANSFER DATA TO PROGRAM
     C                    MOVEL"GETDTA"  ACTION
     C                    SETON                     35    GETDTA - 64
     C                    EXCPT
     C                    SETOF                     35
(17) C*  READ DATA FROM TRANSFIL
     C                    READ TRANSFIL                 HO
     C*
(18) C*  PERFORM USER EDITS
     C           MORERR   TAG
     C*
     C*** SUPPLY USER EDITS HERE
(19) C*  FIND ERRORS, ENHANCE FIELDS, DISPLAY MESSAGE
     C  N20               GOTO NOERRS
     C                    MOVE "     "   FLDNO   5
     C                    MOVE "OO"      MSLEN   2
     C                    MOVE " "       ENHCD   1
     C                    MOVE " "       MSG    47
     C                    SETON                     36    CORERR - 56
     C                    EXCPT
     C                    READ TRANSFIL
     C                    GOTO MORERR
     C           NOERRS   TAG
     C                    SETOF                     36
     C*
(20) C*  MOVE DATA FROM PROGRAM TO BUFFER
     C                    SETON                     37    PUTDTA - 63
     C                    EXCPT
     C                    SETOF                     37
     C*
```

Figure 10-7. A Program that Uses VPLUS (Continued)

```
         1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

(21)     C*  WRITE DATA FROM BUFFER TO BATCH FILE
         C                    MOVEL"WRTBAT"  ACTION
         C                    SETON                     35    WRTBAT - 66
         C                    EXCPT
         C                    SETOF                     35
         C*
(22)     C*  CHECK NEXT FORM FOR STATUS CHANGE
         C                    MOVE "         "NXTFRM 15
         C                    MOVEL"         "NXTFRM
         C                    MOVELFORMB      NXTFRM
         C                    MOVE "0"        RPTAPP  1
         C                    MOVE "0"        FRZAPP  1
         C                    SETON                     38    CHGNXT - 50
         C                    EXCPT
         C                    SETOF                     38
         C*
(23)     C*  RETURN TO STEP 1
         C                    SETOF                     011018
         C                    SETOF                     19
         C                    GOTO START
         C*
(24)     C*  END OF PROCESSING
         C           EXIT     TAG
         O*** INDICATOR 35 FOR ACTIONS 51,53,54,58-61,64-70
         O*** INDICATOR 36 FOR ACTIONS 56 AND 62
         O*** INDICATOR 37 FOR ACTIONS 57 AND 63
         O*** INDICATOR 38 FOR ACTION 50
         O*
(25)     OTRANSFILE        35
         O                          ACTION     6
         O        E        36
         O                                      6 "CORERR"
         O                          FLDNO     11
         O                          MSLEN     13
         O                          ENHCD     14
         O                          MSG       60
         O        E        37
         O                                      6 "PUTDTA"
         O                          DATALN    10
         O                          DATA      67
```

Figure 10-7. A Program that Uses VPLUS (Continued)

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

0         E        38
0                                        6 "CHGNXT"
0                           NXTFRM     21
0                           RPTAPP     22
0                           FRZAPP     23
```

Figure 10-7. A Program that Uses VPLUS (Continued)

Comments

(1) This Header Specification directs RPG to save dump information in the file DUMPFILE if run-time errors occur.

(2) This line defines the WORKSTN (terminal) file TRANSFIL.

Column 15 is U to specify an update file type. You must use this file type with VPLUS.

Column 16 is D to specify that TRANSFIL is a demand file. For this particular example, TRANSFIL must be a demand file. For applications that use the RPG logic cycle, the file may be a primary file.

Column 19 is V to specify variable length records. All WORKSTN files must have variable length records.

Columns 24-27 contain the record length. The record length is dependent on the data buffer length for the longest form, plus 20 characters of control information. (Assume that the longest form in this example contains 60 data characters.)

Columns 40-46 contain WORKSTN to define the device class name for the VPLUS file.

Column 50 is L to enable the function key labels on the terminal.

Column 51 is 5 to specify that 5 seconds is the length of time that a message is displayed at the terminal.

Column 52 is B to enable the BREAK key.

(3) This line defines the VPLUS forms file used in the program. (This line is required.)

Column 53 is K to identify this File Description Specification as a Continuation line.

Columns 54-59 specify that this is a FORMS File Description Continuation line.

Columns 60-74 name the forms file FORMA. If the name is longer than 15 characters, enter an MPE FILE command for it.

(4) This line defines the VPLUS batch file used in the program. (This line is optional.)

Columns 54-59 specify that this is a BATCH File Description Continuation line.

Columns 60-74 name the batch file BATCHA.

(5) This line defines the VPLUS trace file used in the program. (This line is optional.)

Columns 54-59 specify that this is a TRACE File Description Continuation line.

Columns 60-74 name the trace file TRACEFL.

(6) This line begins the Input Specifications for event 00 (the ENTER key).

(7) This line begins the Input Specifications for events 01-08. These events do not read data; they direct program execution to the label EXIT.

(8) This line begins the Input Specifications for event 09 (fields with errors).

(9) The next few lines read the next form from the forms file (GETNXT, action 51). If the form is currently being used, it is not read.

(10) The next few lines initialize the form with the FORMSPEC default values (INIT, action 58).

(11) The next few lines display the initialized form and a message in the message window (SHOW, action 53).

(12) The next few lines indicate that data is to be read from the terminal with RDTERM (action 54).

(13) The next few lines read (demand read) the data from the terminal. The data is returned as events 00-08 (for simplicity in this example, events 01-07 are treated as event 08).

If the event is 01-08 (EXIT), terminate the program. If the event is 00 (ENTER), continue.

(14) Perform all VPLUS field edits specified for the form using EDITS (action 59).

(15) Read the WORKSTN file. The record returned (event 09) specifies the number of errors found in the data.

If errors are found, a message describing the first error is moved to the window buffer, and all fields with errors are enhanced. Return to Step 11 to display the form with the error message and enhancements. Repeat Steps 11-15 until no errors remain. If there are no errors, continue with Step 16.

(16) Transfer data from the VPLUS data buffer to the program using GETDTA (action 64).

(17) Read the WORKSTN file. The record returned (event 10) contains the data from the VPLUS buffer.

(18) Perform editing specified in the program. If errors are found, continue with Step 19; otherwise, go to Step 20.

(19) Using CORERR (action 56) indicate which fields failed the user edits and display the form with these fields. If the program buffer contains a message for the window, display it with the form. Action 56 is followed by a demand READ. If the event returned is 00, terminal operator entered corrections and pressed (ENTER). Repeat Steps 18 and 19 until all data passes the user edits.

(20) Transfer the edited data to the VPLUS buffer using PUTDTA (action 63).

(21) Write the data in the VPLUS buffer to the batch file using WRTBAT (action 66).

(22) If the next form name or the repeat/append or freeze/append status should be changed, CHGNXT (action 50) makes these changes.

(23) Go to Step 9.

(24) End of processing.

(25) This line begins the Output Specifications for actions 56, 63 and 50.

11

RPG Screen Interface (RSI)

The RPG Screen Interface (RSI) lets you read and display data on a terminal using the entire screen. RSI is an alternative to the RPG interface to VPLUS. To use the RPG interface to VPLUS, you must enter action and event operations in the Calculation Specifications which may seem different and cumbersome. RSI lets you use the screen in a way that is similar to performing other RPG input and output functions and it requires just a small amount of additional code to use.

RSI lets you:

- Perform screen input and output using conventional RPG specifications.

  You can retrieve a screen form from an RSI forms file, display it using default field values, read data entered on the form, and detect and flag input errors.
- Use RPG indicators to control form and field attributes.
- Control program logic by using the command key indicators (KA-KN, KP-KY) and function key return codes.
- Use forms that overlay portions of other forms.
- Use forms that do not begin on the first line of the screen.
- Use forms that have been downloaded to the terminal. (You can download RSI forms only with 2394A and 700/94 terminals.)
- Use CONSOLE input files. A CONSOLE file is a special kind of RSI file that is processed automatically by RPG and whose forms file is generated at compile-time using the file's Input Specifications. (See "Using RSI CONSOLE Files" in this chapter.)

This chapter explains how to use RSI. The chapter is divided into three main sections. The first describes how to take full advantage of the RSI features. The second section discusses those RSI features that relate specifically to RSI CONSOLE files. The last section, starting with "Using Messages with RSI", gives additional topics to consider when using RSI files.

Using the RPG Screen Interface (RSI)

To use RSI in an RPG program, perform the following steps (each of the steps is explained in detail in the following sections in this chapter):

1. Create the RSI forms file using the RPG utility, SIGEDITOR.
2. Specify screen-related options and define the RSI forms file using File Description, Input and Output Specifications.
3. If necessary, you can enter Calculation Specifications to perform auxiliary processing for the RSI file. You condition operations using one or more of the RPG indicators including the command key indicators (KA-KN, KP-KY).

   You can perform exception output and demand input of an RSI form by using the EXCPT and READ operations, respectively.

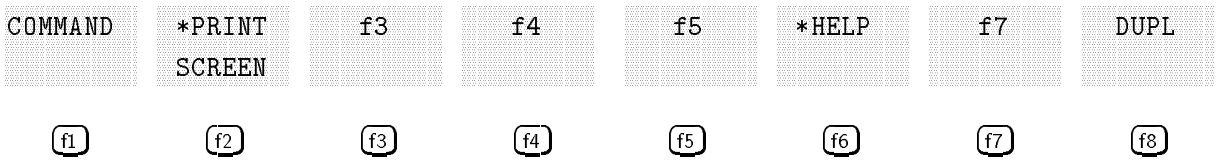
Using SIGEDITOR

To create or modify an RSI forms file, you must use the RPG utility, SIGEDITOR. (See the *RPG Utilities Reference Manual*.) SIGEDITOR gives you complete flexibility in defining an RSI form. You define where the fields are located, their lengths and data types and the values they can contain.

In addition to defining fields, you can enable any or all of the command key indicators (KA-KN, KP-KY). The command key indicators let RSI communicate events to the RPG program at run time. For example, a data entry operator (user) can press a certain key to end the program. This turns ON the associated command key indicator. See the section "Executing an RSI Program" for information on the command keys and the command key indicators associated with them.

Redefining Function Key Labels

An RSI form by default displays the following key labels for enabled function keys:



The key labels for f1 and f8 cannot be changed, but you can specify labels for the keys f2 through f7 by taking these steps:

1. Add a $CONTROL compiler command with the FKEYLBL parameter to your program.
2. Add a File Extension Specification defining an array of 8 elements with 16 alphanumeric characters in each element. The first 8 characters of each element are the first line of the label, and the second 8 characters are the second line of the label.
3. Add a Calculation Specification using the SET operator to load your array into the key labels. Place this statement before the one displaying the RSI form.

Remember that you can redefine labels for only function keys f2 through f7, and of those, for only the ones that were enabled when the RSI form was created with SIGEDITOR. Your label array must include labels for all eight function keys, though labels for the first, last, and disabled keys may be blank because they will be ignored.

Using the RSI Application Help Facility

The RSI application help facility consists of "help areas" and "help forms." Both are created using the SIGEDITOR. (See the *RPG Utilities Reference Manual*.) The SIGEDITOR can also create help areas from IBM "H" specifications.

Help areas are imaginary boundaries that are overlayed on the application form. Up to 256 help areas are allowed on an individual application form. When the user presses F6 (HELP), RSI notes the cursor position and scans a list of help area definitions for the first one that covers the cursor position. The help area definition contains the name of the first help form to display. Additional help forms, if available, can be displayed using F3 (ROLL UP) and F4 (ROLL DOWN) keys. The user can return to the application form by pressing ENTER.

Help forms are ordinary RSI forms, except that they are made of text only. The text can start on any line from 01-24 and can clear up to 24 lines. Help forms can overlay other help forms, but cannot overlay the application form. Help forms must not contain any fields, video enhancements, or message constants. They cannot use the variable starting line number feature. If a help form contains any unsupported specifications, they are ignored when the form is displayed.

An RSI application using the help facility must be prepared with DS capability before use in an RPG program. The application form must have the F6 (HELP) key disabled. No additional RPG programming is required to use the basic RSI application help facility. If you want to use the suppressed selection or boundary features, you must enter RPG calculation specifications to control these indicators.

Entering File Description Specifications

To use an RSI forms file in a program, you must define it as a WORKSTN file in the Device Class Name Field (columns 40-46) of the File Description Specification for the file. You must also enter R in the Interface Type Field (column 47) of the File Description Specification.

Table 11-1 describes other fields in the File Description Specification that you may use with RSI files. The fields that are optional are followed by an asterisk (*). For more information on the RSI STATUS array (columns 54-59), see the next section.

Table 11-1. RSI File Description Specifications

Specification Type	Field	Value
File Description\(F)	File Name(columns 7-14)	The name of the RSI file.
	File Type (column 15)	U (update).
	File Designation\(column 16)	P (primary) or D (demand).
	Record Format (column 19)	V (variable-length records).
	Logical Record Length (columns 24-27)	The length of the longest record defined in the Input and Output Specifications.
	Device Class Name (columns 40-46)	WORKSTN.
	Interface Type (column 47)	R (standard RSI file).
	Interface Control* (column 52)	B (enable) or blank (disable) the BREAK key.
(Continuation line)	Option Type* (columns 54-59)	`STATUS` The RSI STATUS array (containing 6 elements, each 10 digits long with no decimals) is defined automatically by RPG.
		`FORMS` The forms file name containing forms used by the program. If you omit this line, RPG uses the entry in the Program Name Field (columns 75-80) of the Header Specification (with "FM" appended) as the forms file name. If there is no program name in the Header Specification, the forms file name is RPGOBJFM.
		`TRMID` The field name containing the terminal identification number.
		`START` The field name containing the starting line number for forms with variable starting line numbers.
		`FIRST` The name of the form displayed during program initialization.

Example

Figure 11-1 shows how to define an RSI forms file in the File Description Specification. Line 1 defines the file WORKSTN as an RSI WORKSTN file. Line 2 defines the RSI STATUS array #WSTN#. Line 3 names the RSI forms file, NEWLIB.

```
          1         2         3         4         5         6         7
678901234567890123456789012345678901234567890123456789012345678901234
          ------------------------------------------------------------

1         FWORKSTN UP  V      91            WORKSTNR    B
2         F                                              KSTATUS#WSTN#
3         F                                              KFORMS NEWLIB
```

Figure 11-1. Entering RSI File Description Specifications

Using the STATUS Array

The STATUS array (declared by a STATUS File Description Continuation line) is predefined as a 6-element, 10-digit array with zero decimal places. You can use the STATUS array to determine which function key the user pressed and where the cursor is positioned on the screen.

In order for RSI to detect the function keys, you must enable them (in the form's attributes) when running SIGEDITOR. Also, your terminal must return the default escape sequences ESCp through ESCw when the keys f1 through f8 are pressed. See your terminal reference manual to check this.

The first three elements in the STATUS array are used by RSI. They are:

This STATUS element:	is set to:	when:
1	0	ENTER is pressed.
	2	Command key (f1) is pressed.
	1121	Print Screen (f2) is pressed.
	1122	Roll Up (f3) is pressed.
	1123	Roll Down (f4) is pressed.
	1124	Clear (f5) is pressed.
	1125	Help (f6) is pressed.
	1126	Record Backspace (f7) is pressed.
	1127	Duplicate (f8) is pressed.
2	Screen row number.	One of the function keys is pressed.
3	Screen column number.	One of the function keys is pressed.

Entering Input and Output Specifications

There are two ways to enter Input and Output Specifications for an RSI file:

- You can enter the specifications yourself.
- You can have SIGEDITOR generate the specifications when you define the form. Once the specifications are generated, you may need to tailor them for additional requirements.

Table 11-2 describes the fields in the Input and Output specifications that relate to an RSI forms file. The fields that are optional are followed by an asterisk (*).

Table 11-2. RSI Input and Output Specifications

Specification Type	Field	Value
Input (I) (File and Record Description line)	File Name (columns 7-14)	The name associated with the WORKSTNR device.
	AND/OR* (columns 14-16)	AND or OR to complete the identification of the form.
	Group Sequence* (columns 15-16)	A numeric entry for sequence checking or an alphabetic entry for no sequence checking.
	Number of Records* (column 17)	1 or N if the Group Sequence Field is numeric.
	Option* (column 18)	Blank or 0 if the Group Sequence Field is numeric.
	Record Indicator* (columns 19-20)	The record-identifying indicator used to identify the form.
	Record Identification Codes* (columns 21-41)	The record identification codes that identify the form.
(Field Description lines)	From Field Position (columns 44-47)	The first location of the field in the record.
	To Field Position (columns 48-51)	The last location of the field in the record (for signed numeric fields, do not include the sign position).
	Decimal Positions (column 52)	A digit that specifies the number of decimal positions in a numeric field.
	Field Name (columns 53-58)	The name of a field or array element.
	Field Record Relation* (columns 63-64)	The field record relation indicator.
	Field Indicators* (columns 65-70)	The field indicators.

Table 11-2. RSI Input and Output Specifications (continued)

Specification Type	Field	Value
Output (O) (Record Description line)	File Name (columns 7-14)	The name associated with the WORKSTNR device.
	AND/OR* (columns 14-16)	AND or OR to relate output indicators on consecutive lines.
	Type (columns 15)	H (heading), D (detail), T (total), or E (exception).
	Fetch Overflow/Release (column 16)	R to close the WORKSTN file after the specification is executed.
	Output Indicators* (columns 23-31)	Output indicators.
	Field Name* (columns 32-37)	The EXCPT Name if the Type Field is E.
(First Field Description line)	Field Name (columns 32-37)	The name of an alphanumeric field containing the form name (this is not required if the Constant/Edit Word Field contains the name).
	End Position (columns 40-43)	K and the number of characters in the form name in column 43.
	Constant/Edit Word (columns 45-54)	The form name enclosed in quotation marks (not required if you enter the form name in the Field Name Field).
(Remaining Field Description lines)	Output Indicators* (columns 23-31)	Output indicators.
	Field Name (columns 32-37)	A field name.
	Edit Code* (column 38)	An edit code.
	Blank After* (column 39)	Blank or B.
	End Position (columns 40-43)	An ending position (for signed numeric fields, do not include the sign position).
	Constant/Edit Word* (columns 45-70)	An edit word or constant.

Entering Calculation Specifications

You may include Calculation Specifications to condition operations related to RSI form processing. You can condition operations using any of the RPG indicators including the command key indicators enabled in SIGEDITOR (see the previous section "Using SIGEDITOR"). The command key indicators are turned ON when the user presses the corresponding command key. (See the next section for a description of the command keys.)

If you need to read or write the RSI screen on a demand basis, use the READ and EXCPT operations (you must define the file as an update demand file in the File Description Specification). Unless you specified in SIGEDITOR that the form is for exception output, EXCPT output is held until a READ is encountered, then one physical write and one physical read are performed.

Example

Figure 11-2 shows how to use command key indicator KG to end a program. (You can use a command key indicator like a general indicator, but you should only do this when you are not using an RSI WORKSTN file in the program.)

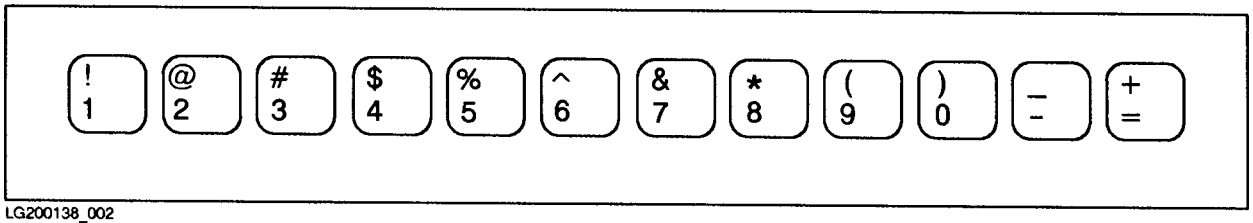
```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

C   KG                GOTO EOF
```

Figure 11-2. Entering an RSI Calculation Specification

Executing an RSI Program

You execute a program that uses RSI the same way you do other RPG programs. Depending on the features you use in the program and the environment in which it is run, you may need to perform additional tasks before running the program. See the next two sections and the sections at the end of this chapter starting with "Using Different Terminals" to see if any of these tasks apply to your particular program.

When you run a program that processes an RSI file, the data entry user issues "commands" to RSI by pressing up to 24 command keys (COMMAND f1 followed by a key on the top row of the keyboard). When a command key is pressed, the associated command key indicator is turned ON. (You must have already enabled the command key indicator in SIGEDITOR.). The keys that you can use on the top row of the keyboard are the shifted and unshifted keys 1-9, -, and =. They are shown in Figure 11-3.



LG200138_002

Figure 11-3. RSI Command Keys

Table 11-3 lists the command keys and the command key indicators that correspond to them. When the user presses a command key, the corresponding command key indicator is turned ON and all other command keys indicators are turned OFF.

Table 11-3. How RSI Command Key Indicators Are Turned ON

COMMAND (f1) followed by:	Turns ON this Command Key Indicator:
1	KA
2	KB
3	KC
4	KD
5	KE
6	KF
7	KG
8	KH
9	KI
0	KJ
-	KK
=	KL
SHIFT 1	KM
SHIFT 2	KN
SHIFT 3	KP
SHIFT 4	KQ
SHIFT 5	KR
SHIFT 6	KS
SHIFT 7	KT
SHIFT 8	KU
SHIFT 9	KV
SHIFT 0	KW
SHIFT -	KX
SHIFT =	KY

Note

The command keys shown in Table 11-3 are supported on 239x and 262x terminals (264x terminals are also supported but the keys on the keyboard may not be the same). If you have a different terminal or a non-standard keyboard, you can can still use RSI with it but you must define the command keys in the file CMDKEYS. CMDKEYS enables you to use RSI with any terminal or keyboard configuration. See the section "Using Different Terminals" for instructions on how to set up the CMDKEYS file.

Using Messages with RSI

If you're using messages with an RSI form and you're accessing them by message identification numbers or you did not merge them with the forms file in SIGEDITOR, RSI will access the message file CATALOG at run time. If CATALOG is not the file you want to use, enter an MPE FILE command to equate CATALOG to that file. For example, if your message file is INVMSGS, enter this file equation:

```
:FILE CATALOG=INVMSGS
```

Displaying an End-of-Program Form

When you write a display-only form to the screen after the user ends program input, that form remains on the screen until the user presses ENTER, an enabled command key, or an enabled function key. To automatically remove the form from the screen after a few moments, set the JCW RSIPAUSE before executing the program or use a PUTJW Calculation Specification operation to set the JCW. When you set the JCW RSIPAUSE, enter the number of seconds that you want the form to remain on the screen. If the number is not in the range 1 to 60, it is ignored. In the following example, the final screen for program INV3470P is displayed for 15 seconds before the program ends:

```
:SETJCW RSIPAUSE=15
:RUN INV3470P
```

Sample RSI Programs

This section shows two versions of a program that uses RSI. One version, shown in Figure 11-7, defines the RSI forms file as a primary file and lets RPG process it normally in the logic cycle. The second version, shown in Figure 11-8, defines the RSI forms file as a demand file and contains Calculation Specifications to read and write screen data using the forms file.

The sample program updates a customer information file that uses social security numbers as its key. The user can add or delete persons by entering their social security number then pressing command keys 1 or 2, respectively. Or the user can change an individual's address or phone information by pressing command key 3.

Two RSI forms are used in each version of the program. The first one that is displayed is shown in Figure 11-4. If the user wants to add a record or update one that already exists, a second screen (Figure 11-5) is displayed. The shaded parts in these figures show where the user enters data and where output data is displayed. (The screen title and messages are shown in Figure 11-6. They are contained in a message file that was created before the program is executed. You create message files using GENCAT or MAKECAT. GENCAT is described in the *Native Language Programmer's Guide* and MAKECAT is discussed in the *Message Catalogs Programmer's Guide*.)

```
               CUSTOMER INFORMATION FORM

               Please enter Social Security Number

                       306-46-7778

               RECORD ALREADY EXISTS

         Press CMD 1 TO ADD NEW RECORD
               CMD 2 TO DELETE EXISTING RECORD
               CMD 3 TO UPDATE EXISTING RECORD
               CMD & TO END JOB



  COMMAND   *PRINT   f3     f4          f5   *HELP     f7     f8
            SCREEN
```

Figure 11-4. Form21 (Contained in RSI Forms File SAMPLIB)

```
                    SOC. SEC. NO. 306-46-7778


               NAME  MARY Q. QUAIL

             ADDRESS  MORSE GROVE, FLA

            ZIP CODE  339080000     PHONE NO.  813-672-1212


                           PRESS  :ENTER:



  COMMAND   *PRINT   f3     f4          f5   *HELP     f7     f8
            SCREEN
```

Figure 11-5. Form22 (Contained in RSI Forms File SAMPLIB)

```
     $SET 1
     1 RECORD ALREADY EXISTS
     2 RECORD DOES NOT EXIST
     3 CUSTOMER INFORMATION FORM
```

Figure 11-6. The Message File

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

HDUMPFILE JF          X     B       B N     P1 1

1   FWORKSTN UP  V      90            WORKSTNR    B
    F                                              KFORMS SAMPLIB
    FCUSTMASTUC  F     128R 9AI     1 DISK                     A
    F                                              KKEYFL KCUSTMST
    F*
    F* SAMPLE RSI PROGRAM USING THE WORK STATION AS A PRIMARY FILE
    F*

2   IWORKSTN NS  01   1 C1
    I* FORM21
    I                                           1   1 FLD1
    I                                           2  10OSSN
    I        NS  02   1 C2
    I* FORM22
    I                                           1   1 FLD1
    I                                           2  27 NAME
    I                                          28  63 ADDR
    I                                          64  72OZIP
    I                                          73  82 PHONE#
    I        NS
3   ICUSTMASTNS
    I                                           1   9OSSN
    I                                          10  35 NAME
    I                                          36  71 ADDR
    I                                          72  80OZIP
    I                                          81  90 PHONE#

    C  N99                SETON
    C*  CLEAN UP                                    02  99
4   C   01                SETOF                    808182
    C   01                SETOF                    656070
    C*  DETERMINE ACTION REQUESTED
5   C   KA                SETON                    80    ADD RECORD
    C   KB                SETON                    81    DEL RECORD
    C   KC                SETON                    82    UPD RECORD
    C   KG                SETON                    LR    END JOB
    C   KG                GOTO EOF
6   C  NO1                GOTO EOF
    C*  READ RECORD FROM CUSTOMER MASTER FILE USING SSN AS A KEY
    C                     SETOF                    42
    C           SSN       CHAINCUSTMAST            42    NO HIT
```

Figure 11-7. Processing an RSI Primary File

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

     C*  SET INDICATORS FOR ERRORS
7    C      80N42          SETON                     6070
     C     N80 42          SETON                     6065
     C           EOF       TAG
8    OWORKSTN D        1P
     O       OR        02
     O       OR        01 60
     O       OR        01 81
     O                                  K6 "FORM21"
     O                    70             6 "0001 1"
     O                    65             6 "0002 1"
     O**
9    O         D        01 80 42
     O       OR        01 82N42
     O                                  K6 "FORM22"
     O                          SSN      9
     O                          NAME    35
     O                          ADDR    71
     O                          ZIP     80
     O                          PHONE#  90
     O*  UPDATE DISK RECORD
10   OCUSTMASTD        02 82
     O                          SSN    B   9
     O                          NAME   B  35
     O                          ADDR   B  71
     O                          ZIP    B  80
     O                          PHONE# B  90
     O*  ADD DISK RECORD
     O         DADD     02 80
     O                          SSN    B   9
     O                          NAME   B  35
     O                          ADDR   B  71
     O                          ZIP    B  80
     O                          PHONE# B  90
     O*  DELETE DISK RECORD
     O         DDEL     01 81N42
     O                          SSN    B   9
     O                          NAME   B  35
     O                          ADDR   B  71
     O                          ZIP    B  80
     O                          PHONE# B  90
     O*  END OF SOURCE STATEMENTS
```

Figure 11-7. Processing an RSI Primary File (Continued)

Comments

(1) This line and the next one define the RSI file called WORKSTN. It is a primary file having the forms file name SAMPLIB.

(2) This line begins the description of the record identification codes and input fields for the two RSI forms shown in Figures 11-4 and 11-5.

(3) This line begins the description of the input record for the file CUSTMAST.

(4) This line turns off the indicators that are used during each cycle.

(5) This line and the ones which follow, determine the action requested by the user. The command key indicators are turned OFF automatically before the form is read from the screen. Only the first form (FORM21) enabled the command keys. This means that record-identifying indicators are not necessary for additional conditioning here. However, user requests are saved using the general indicators (80, 81 and 82). If the user directs the program to end, LR is turned ON and control skips to EOF.

(6) This and succeeding lines read a record from the file CUSTMAST only if form FORM21 was just read (the record-identifying indicator for FORM21 is ON).

(7) This line and the next determine if there are any errors. If an ADD is requested and a record already exists, indicators 60 and 70 are turned ON. If a CHANGE or DELETE action is requested and the record does not exist, indicators 60 and 65 are turned ON. Indicators 65 and 70 show the specific error encountered and 60 is used as the override indicator. (See the *RPG Utilities Reference Manual* for information about override.)

(8) This and succeeding lines define the output for form FORM21. This form is displayed during the first program cycle (1P): after form FORM22 is read (indicator 02 is ON) or after an error (indicators 01 and 60 are ON) is detected or after a DELETE action (indicators 01 and 81 are ON) is requested. Indicators 65 and 70 determine whether error messages are displayed (RSI locates the messages in the message file before the form is displayed).

(9) This and succeeding lines define the output for form FORM22. This form is displayed if the first form is processed without errors and the user requested an ADD (indicators 01, 42 and 80 are ON) or CHANGE (indicators 01 and 82 are ON and 42 is OFF) action.

(10) This and succeeding lines define the fields that are displayed for the ADD, CHANGE and DELETE actions. The Blank After Field (column 39) is used to clear each field before the next cycle begins.

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
-------------------------------------------------------------------------------------------

HDUMPFILE JF          X     B       B N     P1 1

1   FWORKSTN UD  V      90            WORKSTNR    B
    F                                              KFORMS SAMPLIB
    FCUSTMASTUC  F     128R 9AI     1 DISK                     A
    F                                              KKEYFL KCUSTMST
    F*
    F* SAMPLE RSI PROGRAM USING THE WORK STATION AS A DEMAND FILE
    F*

2   IWORKSTN NS  01   1 C1
    I* FORM21
    I                                           1   1 FLD1
    I                                           2  10OSSN
    I        NS  02   1 C2
    I* FORM22
    I                                           1   1 FLD1
    I                                           2  27 NAME
    I                                          28  63 ADDR
    I                                          64  72OZIP
    I                                          73  82 PHONE#
    I        NS
3   ICUSTMASTNS
    I                                           1   90SSN
    I                                          10  35 NAME
    I                                          36  71 ADDR
    I                                          72  80OZIP
    I                                          81  90 PHONE#

    C           START     TAG
4   C                     SETOF                    010242 CLEAN UP
    C*  DISPLAY & READ FIRST FORM
5   C                     EXCPT          FORM21
    C                     READ WORKSTN
    C*  DETERMINE ACTION REQUESTED
6   C      KG             SETON                      LR  END JOB
    C      KG             GOTO EOF
    C                     SETOF                    808182
    C      KA             SETON                      80  ADD RECORD
    C      KB             SETON                      81  DEL RECORD
    C      KC             SETON                      82  UPD RECORD
    C                     SETOF                    606570
7   C  N80N81N82          SETON                      60
    C      60             GOTO START                     NO CMD KEY;LOOP
```

Figure 11-8. Processing an RSI Demand File

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

     C*  READ RECORD FROM CUSTOMER MASTER FILE USING SSN AS A KEY
(8)  C           SSN       CHAINCUSTMAST             42
     C*  SET INDICATORS FOR ERRORS; LOOP
(9)  C   80   N42          SETON                     60  70
     C  N80    42          SETON                     60  65
     C      60             GOTO START
     C*  DELETE ONLY REQUIRES FIRST FORM, DO IT NOW & LOOP
(10) C      81             EXCPT          DELREC
     C      81             GOTO START
     C*  DISPLAY AND READ SECOND FORM
(11) C                     EXCPT          FORM22
     C                     READ WORKSTN
     C*  EXECUTE REQUESTED ACTION & LOOP
(12) C      82             EXCPT          UPDREC
     C      80             EXCPT          ADDREC
     C                     GOTO START
     C           EOF       TAG

(13) OWORKSTN E                FORM21
     O                                    K6 "FORM21"
     O                   70               6 "0001 1"
     O                   65               6 "0002 1"
     O**
(14) O        E                FORM22
     O                                    K6 "FORM22"
     O                         SSN         9
     O                         NAME       35
     O                         ADDR       71
     O                         ZIP        80
     O                         PHONE#     90
     O* UPDATE DISK RECORD
(15) OCUSTMASTE                UPDREC
     O                         SSN     B   9
     O                         NAME    B  35
     O                         ADDR    B  71
     O                         ZIP     B  80
     O                         PHONE#  B  90
     O* ADD DISK RECORD
     O        EADD             ADDREC
     O                         SSN     B   9
     O                         NAME    B  35
     O                         ADDR    B  71
     O                         ZIP     B  80
     O                         PHONE#  B  90
```

Figure 11-8. Processing an RSI Demand File (Continued)

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

0* DELETE DISK RECORD
0        EDEL             DELREC
0                         SSN     B   9
0                         NAME    B  35
0                         ADDR    B  71
0                         ZIP     B  80
0                         PHONE#  B  90
0* END OF SOURCE STATEMENTS
```

Figure 11-8. Processing an RSI Demand File (Continued)

Comments

1. This line and the next one define the RSI file called WORKSTN. It is a demand file having the forms file name SAMPLIB.

2. This line begins the description of the record identification codes and input fields for the two RSI forms shown in Figures 11-4 and 11-5.

3. This line begins the description of the input record for the file CUSTMAST.

4. This line turns OFF the record-identifying indicators used for the demand file because they are not automatically turned OFF by RPG.

5. This line and the next one display and read the first form (FORM21) using the EXCPT and READ Calculation Specification operations. An EXCPT Name is used instead of indicators to control exception output.

6. This and succeeding lines check the command key indicators to determine what action was requested. If the user ends input, the LR indicator is turned ON and control skips to EOF.

7. This line and the next one turn on the RSI override indicator (60) when no command keys indicators are ON. Control branches to the beginning (START) of the Calculation Specifications so that the first form can be redisplayed with override in effect. Override prevents data already entered on the form from being erased.

8. This line reads the file CUSTMAST looking for the record whose key matches the social security number (SSN) entered at the terminal. If the record is not found, indicator 42 is turned ON.

9. If the action requested was an ADD and the record already exists, or if the action was CHANGE or DELETE and the record does not exist, indicator 60 is turned ON. It serves as a general error indicator and as an RSI override indicator. Indicators 65 and 70 are turned ON for the specific error condition.

   If there is an error, control returns to the beginning of the Calculation Specifications so that FORM21 can be redisplayed with the appropriate error message and with override in effect.

(10) This line and the next one perform the DELETE action using exception output and an EXCPT name. Once processing is complete for this action, control skips to the beginning of the Calculation Specifications (START).

(11) This and the next line display and read FORM22 using the EXCPT and READ operations. The EXCPT Name is used instead of indicators to control exception output.

(12) This line and the next one perform the ADD and CHANGE actions using exception output and an EXCPT Name. Control then skips to the beginning of the Calculation Specifications to start the next cycle.

(13) This line and the ones which follow define the output for FORM21. This form is displayed using exception output with an EXCPT Name; no additional conditioning is necessary. Indicators 65 and 70 control the error messages. RSI locates the messages in the message file before the form is displayed.

(14) This line and the ones which follow it define the output for form FORM22. This form is displayed using exception output with an EXCPT Name; no additional conditioning is necessary.

(15) This and succeeding lines define the output for CUSTMAST for each action. The Blank After Field (column 39) is used to clear each field before the next cycle begins.

Using RSI CONSOLE Files

A CONSOLE file is a special RSI file that you can use for simple data collection applications, such as entering timecard data.

When you use a CONSOLE file, the compiler uses SIGEDITOR to generate a forms file from the CONSOLE file's Input Specifications.

Since CONSOLE files are used for input only, do not enter Output Specifications for them. At run time, RPG performs all of the input and output for CONSOLE files.

To use an RSI CONSOLE file in an RPG program, perform the following steps (each step is described in the following sections of this chapter):

1. Define the RSI CONSOLE file using File Description and Input Specifications.
2. If necessary, enter Calculation Specifications to perform auxiliary processing for the RSI CONSOLE file. For example, you may want to print the data on the screen.

Entering File Description and Input Specifications

To use an RSI CONSOLE file in a program, you must define it in a File Description Specification as a WORKSTNC file (the Device Class Name Field (columns 40-46) is WORKSTN and the Interface Type Field (column 47) is C).

Table 11-4 describes the fields in the File Description and Input Specifications that relate to an RSI CONSOLE forms file. The fields that are optional are followed by an asterisk (*).

Table 11-4. RSI CONSOLE File Description and Input Specifications

Specification Type	Field	Value
File Description (F)	File Name (columns 7-14)	The name of the RSI CONSOLE file.
	File Type (column 15)	I (input).
	File Designation (column 16)	P (primary) or D (demand).
	Record Format (column 19)	V (variable-length records).
	Logical Record Length (columns 24-27)	The length of the longest record defined in the Input Specifications.
	Device Class Name (columns 40-46)	WORKSTN.
	Interface Type (column 47)	C (RSI CONSOLE file).
	Interface Control* (column 52)	B (enable) or blank (disable) the BREAK key.
Input (I)		
(File and Record Description line)	File Name (columns 7-14)	The name associated with the WORKSTNC device.
	OR* (columns 14-15)	OR if more than one record type uses all of the same fields. This line must contain the same number of record identification codes (in the Record Identification Code Fields, columns 23-34) as the line it follows.
	Group Sequence* (columns 15-16)	A numeric entry for sequence checking or an alphabetic entry for no sequence checking.
	Number of Records* (column 17)	Blank if the Group Sequence Field (columns 15-16) is alphabetic; 1 if this is the only record for this record type and N if there is more than one record for this record type.

Table 11-4.
RSI CONSOLE File Description and Input Specifications (continued)

Specification Type	Field	Value
Input (I)		
(File and Record Description line)	Option* (column 18)	Blank if the Group Sequence Field is alphabetic; 0 if the record type is optional.
	Record Indicator (columns 19-20)	The record-identifying indicator that identifies the form. It must be in the range 01-10 and cannot be used with more than one record type in the program. (You can have up to 10 record types in the program.)
	Record Identification Codes	The record identification codes that identify the form.
	(columns 21-24)	1 (The record identification code must begin in the first position of the record.)
	(column 26)	C (Use the entire character for the record identification code.)
	(column 27)	A character that identifies this record type.
	(columns 28-34)*	Contains a second record identification code for the record type. Enter information into these columns the same way you did in columns 21-27, except enter 2 in column 31.
(Field Description lines)	From Field Position (columns 44-47)	The first location of the field in the record.
	To Field Position (columns 48-51)	The last location of the field in the record (for signed numeric fields, do not include the sign position) The maximum field length is 66 for alphanumeric fields and 15 digits for numeric fields.
	Decimal Positions (column 52)	A digit that specifies the number of decimal positions in a numeric field.
	Field Name (columns 53-58)	The name of the field. Use a descriptive name since it is used as the input prompt.
	Field Record Relation* (columns 63-64)	The field record relation indicator.
	Field Indicators* (columns 65-70)	The field indicators.

Example

Figure 11-9 shows how to define an RSI CONSOLE file. Line 1 defines the file SCRNFL as an RSI WORKSTNC file. The lines beginning with line 2 define the fields in SCRNFL.

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

1    FSCRNFL  IP  V      48            WORKSTNC    B

2    ISCRNFL  NS  01   1 CT   2 CC
     I                                          1   2 RECID
     I                                          3   8 WEDATE
     I                                          9  140EMPNUM
     I                                         15  182PYRATE
     I                                         19  22 ACCTCD
     I                                         23  261TOTHRS
     I                                         27  291MONHRS
     I                                         30  321TUEHRS
     I                                         33  351WEDHRS
     I                                         36  381THUHRS
     I                                         39  411FRIHRS
     I                                         42  441SATHRS
     I                                         45  471SUNHRS
```

Figure 11-9. Entering RSI CONSOLE File Description and Input Specifications

Entering Calculation Specifications

You may include Calculation Specifications to condition operations related to a CONSOLE file. For example, Figure 11-10 shows an ADD operation that counts the number of transactions (screens of data) entered by the data entry operator (user).

Do not condition Calculation Specifications with command key indicators (you cannot use command key indicators with RSI CONSOLE files).

```
     1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
_____________________________________________________________________________________________

C                     ADD  1         COUNT
```

Figure 11-10. Entering an RSI CONSOLE Calculation Specification

Compiling an RSI CONSOLE Program

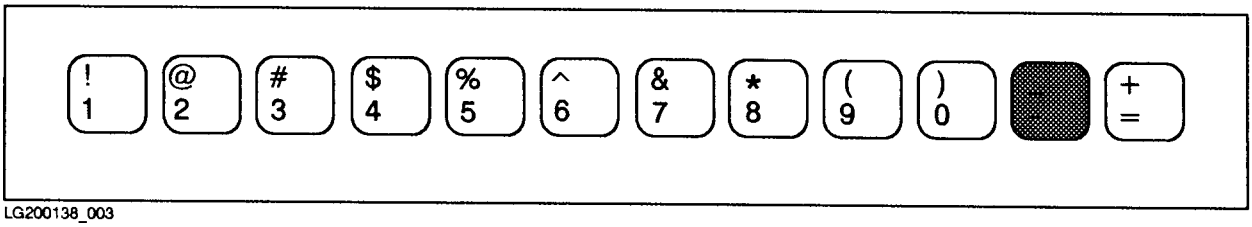
When you compile a program containing an RSI CONSOLE file for the first time, use the GEN option of the $CONTROL compiler subsystem command (or omit the NOGEN option). GEN directs the compiler to generate a forms file for the CONSOLE file. You can tailor this forms file using SIGEDITOR, if necessary (for information on this see the next paragraph). Once the program is debugged, use the NOGEN option of $CONTROL for subsequent compilations. NOGEN directs the compiler not to generate a new forms file.

If you use SIGEDITOR to tailor the form, be careful not to change the top (status) line of the form. In addition, do not change any constants or alter the size or order of the fields. See the *RPG Utilities Reference Manual* for instructions on how to use SIGEDITOR to modify the forms file. Once your form is customized and is formatted properly, be sure to compile the program using the NOGEN parameter of the $CONTROL compiler subsystem command; otherwise the customized forms file will be overwritten.

Whenever you change the Input Specifications for a CONSOLE file, you *must* recompile the program with the GEN parameter (or omit NOGEN) to generate a new forms file.

Executing an RSI CONSOLE Program

When a program that uses CONSOLE files is executed, the data entry operator (user) selects the form to display by pressing one of 10 command keys. They are the unshifted keys 1-0 and = on the top row of the keyboard as shown in Figure 11-11.



LG200138_003

Figure 11-11. RSI CONSOLE Command Keys

When the user presses a command key, the form whose record indicator matches the command key is displayed. For example, if you use record indicator 3 with a form, the user displays it by pressing f1 followed by 3. Table 11-5 lists the command keys along with their descriptions.

Table 11-5. How to Use RSI CONSOLE Command Keys

COMMAND (f1) followed by:	Performs this action:
1	Displays the record type whose record indicator is 1.
2	Displays the record type whose record indicator is 2.
3	Displays the record type whose record indicator is 3.
4	Displays the record type whose record indicator is 4.
5	Displays the record type whose record indicator is 5.
6	Displays the record type whose record indicator is 6.
7	Displays the record type whose record indicator is 7.
8	Displays the record type whose record indicator is 8.
9	Displays the record type whose record indicator is 9.
0	Displays the record type whose record indicator is 10.
=	Ends the program.

Note

The command keys shown in Table 11-5 are supported on 239x and 262x terminals. If you have a different terminal or a non-standard keyboard, you can can still use RSI with it but you must define the command keys in the file CMDKEYS. CMDKEYS enables you to use RSI with any terminal or keyboard configuration. See the section "Using Different Terminals" for instructions on how to set up the CMDKEYS file.

Sample RSI CONSOLE Program

The sample program in this section shows how to use an RSI CONSOLE file to collect weekly payroll information. For each field in the file, the user is prompted to enter the corresponding piece of data. The program writes the payroll data to a disk file for later processing.
Figure 11-12 shows what the CONSOLE file fields look like when displayed on the screen. The program itself is shown in Figure 11-13.

```
TC      01         1               1
WEDATE  A    6 ______
EMPNUM  N  6.0 _______
PYRATE  N  4.2 _____
ACCTCD  A    4 ____
TOTHRS  N  4.1 _____
MONHRS  N  3.1 ____
TUEHRS  N  3.1 ____
WEDHRS  N  3.1 ____
THUHRS  N  3.1 ____
FRIHRS  N  3.1 ____
SATHRS  N  3.1 ____
SUNHRS  N  3.1 ____




COMMAND   *PRINT   f3     f4          f5     f6     f7     f8
          SCREEN
```

Figure 11-12. An RSI CONSOLE Form

The top line of the display in Figure 11-12 is the status line. TC is the record identification code entered in columns 24-34 of the Input Specification that defines the record type. The second field in the status line is the record-identifying indicator (01) for the record type. The third and fourth fields in the status line indicate the other record types that can be selected before and after the user enters data in the form. (In this example, only one record type is defined, so 1 appears in both places.)

The rest of the screen shows the fields defined in the Input Specifications for the record type. The following information is displayed for each field: field name, data format (A=Alpha, N=Numeric), field length, the implied decimal position (for numeric fields) and an underscored area that shows where to enter data. As data entry is completed for each screen, the user presses ENTER. The user presses command key 12 (= key) to end the program.

```
          1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

(1)       $CONTROL NAME=TIMECD,GEN
          HDUMPFILE JF          X     B       B N     P1 1

          F*
          F*  SAMPLE CONSOLE FILE PROGRAM - TIMECARD DATA ENTRY
          F*
(2)       FSCRNFL  IP  V      48            WORKSTNC    B
          FOUTPUT  O   F      48            DISK

(3)       ISCRNFL  NS  01   1 CT   2 CC
(4)       I                                       1   2 RECID
          I                                       3   8 WEDATE
          I                                       9  14OEMPNUM
          I                                      15  18 PYRATE
          I                                      19  22 ACCTCD
          I                                      23  26 TOTHRS
          I                                      27  29 MONHRS
          I                                      30  32 TUEHRS
          I                                      33  35 WEDHRS
          I                                      36  38 THUHRS
          I                                      39  41 FRIHRS
          I                                      42  44 SATHRS
          I                                      45  47 SUNHRS

          C*
(5)       C*  NO CALCULATION SPECIFICATIONS REQUIRED
          C*

(6)       OOUTPUT  D           01
          O                         RECID      2
          O                         WEDATE     8
          O                         EMPNUM    14
          O                         PYRATE    18
          O                         ACCTCD    22
          O                         TOTHRS    26
          O                         MONHRS    29
          O                         TUEHRS    32
          O                         WEDHRS    35
          O                         THUHRS    38
          O                         FRIHRS    41
          O                         SATHRS    44
          O                         SUNHRS    47
```

Figure 11-13. Processing an RSI CONSOLE File

Comments

(1) This subsystem command gives the program name and directs RPG to generate an RSI forms file from the Input Specifications.

The forms file name is TIMECDFM which is the program name with an FM appended. (You can also enter the program name in the Program Name Field (columns 75-80) of the Header Specification.)

(2) This line defines the CONSOLE file SCRNFL. It is an input primary file with a record length of 48.

(3) This line defines the record type TC for the CONSOLE file. The form name for this record type is FORM01 (RPG assigns the name FORM and the record-identifying indicator is 01).

(4) This and succeeding lines define the fields for record type 01.

(5) This program has no Calculation Specifications.

(6) These lines describe the output file to which the payroll data is written. Enter Output Specifications just as you would in any other RPG program.

Using Different Terminals

If the terminal that you're using is not a 239x, 262x or 264x terminal or if you're using a non-English keyboard, you must define the characters that represent the command keys. Enter one character for each command key into the first 24 characters of the first record in the file CMDKEYS. Enter the ASCII representation of the key(s) on the keyboard. You can create or modify CMDKEYS using any text editor, such as EDITOR.

Improving Performance Under PROCMON

If you're running a program that uses RSI under PROCMON, there are several things that you can do to improve the program's performance:

- If you have a 2394A or 700/94 terminal, you can download RSI forms to terminal memory. This saves time because the forms do not have to be transmitted to the terminal each time they are displayed. To download forms, enter a PROCMON DOWNLOAD command before executing the program.
- If your program is run frequently, you can suspend it at end-of-program rather than terminate it. This saves time because the program does not have to be reloaded into memory and the files remain open. To direct PROCMON to suspend a program, rather than terminate it, set the JCW RPGSUSP before executing the program. To rerun a suspended program, enter a RUN command for it. For example, the following lines set "suspend mode" for program INV505P, run it for the first time, then rerun the program after it is suspended:

```
:SETJCW RPGSUSP=1
:RUN INV505P
  .
  .
  .
:RUN INV505P
```

If a program is run in suspend mode as described above, you can save additional time by downloading RSI forms. When you do this, the forms are downloaded once no matter how may times the program is executed. It is up to you to ensure that the forms in terminal memory remain unaltered while the program is suspended. To set suspend mode and to direct PROCMON to leave the forms in terminal memory unaltered when the program resumes, enter this command before executing the program:

```
:SETJCW RPGSUSP=3
```

Additionally, if a program is run in suspend mode, you can direct PROCMON to leave the function key labels unaltered when the program resumes. To activate the suspend feature and leave the function key labels unaltered, enter this command before executing the program:

```
:SETJCW RPGSUSP=5
```

Example Conventions

To set suspend mode and leave both the forms and function keys unaltered when the program resumes (to perform all of the functions discussed above), enter this command:

```
:SETJCW RPGSUSP=7
```

For additional information on the JCW RPGSUSP, see "Suspending an RPG Program" under the SUSP Calculation Specification operation.

Note

If your program behaves abnormally when running with RPGSUSP = 3, 5 or 7, the forms or function keys may have been altered while the program was suspended. If this happens, end the PROCMON session, then run it again.

12

RPG Compiler

When you compile an RPG program, the compiler reads the source program file and produces either a *relocatable object file* (intermediate form of an object program that must be converted to executable form by HP Link Editor/iX) or an *executable program file* (the form of an object program that is directly executable). When a program is compiled, a list file is produced which contains the source program listing and informational, error and warning messages. If you request it, the compiler also produces a Cross-Reference listing.

This chapter discusses the MPE/iX commands that compile a program and the compiler subsystem commands that determine the compiler options that are used. It also discusses the files used during compilation.

To compile a program successfully, the statements in the source program must be in the correct order. Figure 12-1 shows this order (compiler subsystem commands start with $). In summary, a source program file must contain:

- Not more than one Header Specification (optional).
- At least one Input or Output Specification, unless all files are display type (the File Type Field (column 15) of the File Description Specification is D).

```
    1         2         3         4         5         6         7
67890123456789012345678901234567890123456789012345678901234567890123456789012345
-------------------------------------------------------------------------------

$CONTROL              (Compiler subsystem command optional)

H                     (Header Specification optional)

F
F                     (File Description Specifications optional)
F

E                     (File Extension Specification optional)

L                     (Line Counter Specification optional)

I
I                     (Input Specifications optional)
I

C
C                     (Calculation Specifications optional)
C

O
O                     (Output Specifications optional)
O
```

Figure 12-1. The Format of an RPG Source Program File

The Compiler Commands

The compiler commands are MPE/iX commands that compile and execute RPG programs.

You can use the compile commands in a job file (job mode) or interactively (session mode). They are used at the MPE/iX system prompt (:). The compiler commands are listed below and are described later in this chapter:

Compiler Command	Description
:RPGXL	Compiles an RPG source program.
:RPGXLGO	Compiles an RPG source program, links it creating an executable program file and executes it immediately.
:RPGXLLK	Compiles an RPG source program and links it, creating an executable program file.

The next four sections explain how to enter compiler command parameters. It also explains how to list command error messages, compiler error messages, and the compiler version number.

See the *HP RPG Programmer's Guide* for additional examples on how to compile an RPG source program.

Entering Command Parameters

The parameters used with the RPG compiler commands are positional parameters; they are interpreted based on their position in the parameter list. For example, the :RPGXL command has three parameters which are entered as follows:

```
:RPGXL SOURCE,RELOC,LIST
```

To enter the command, but omit the first two parameters, enter commas for the parameters as follows:

```
:RPGXL ,,LIST
```

When you omit one or more parameters at the end of the parameter list, do not enter commas for them (the end of list signifies that the parameters are absent).

Listing Command Error Messages

If you enter a compiler command incorrectly in session mode, an error number and message are displayed and the command is ignored.

If you enter a compiler command incorrectly in a job file and the command is not preceded by a :CONTINUE command, the error number and message are listed in $STDLIST. The invalid command is ignored, the rest of the job flushed and the job aborted.

Listing the Compiler Version Number

You can list the compiler version number (number of the currently installed RPG compiler and run-time library) by entering a RUN command with the following format:

```
:RUN RPGXLC[.PUB[.SYS]]; INFO="VERSION" [;LIB={G}]
                                             {P}
```

If you omit the LIB parameter, the RPG run-time library in the system XL (Segmented Library in PUB.SYS) is used. Enter LIB=G to use the RPG run-time library in a group XL; enter LIB=P to use the run-time library in your account (XL in the PUB group of that account).

Listing Compiler Error Messages

You can list all of the error messages for the currently installed compiler by executing the GENCAT utility of the Native Language Support (NLS) System (see the *Native Language Programmer's Guide*). Before printing, you must use GENCAT to convert the messages from their compressed format in RPGCAT.PUB.SYS to a printable format.

Files Used by the Compiler

The next three sections give additional details about the files that are used during compilation.

In the pages which follow, the term "formal file designator" is often used. This is the name that the compiler uses when referring to an input or output file (the compiler uses an MPE/iX file equation to equate the designator to the file).

The Source Program File

A source program file is any ASCII file. You can create it using RISE or any text editor (for example, EDITOR). The source program file contains compiler subsystem commands, specifications and compile-time arrays and tables. Source programs are kept as separate entities from job files. For example, you cannot embed a source program in a job file that compiles it.

You can use $NULL in place of a source program name in a compile command. When you do this, no compile takes place because the source file is specified as being empty. For this reason, $NULL is handy for performing conditional compilation.

Table 12-1 lists specific file characteristics that RPG expects for the source program file. The characteristics are separated into those expected when the compiler opens the file (FOPEN intrinsic) and when it closes the file (FCLOSE intrinsic). For a definition of these characteristics, see FOPEN and FCLOSE in the *MPE/iX Intrinsics Reference Manual*.

Table 12-1. Source Program File Characteristics

FOPEN	
Formal File Designator:	RPGTEXT.
Foptions: (%7)	
Domain:	New file (00) or old user file (11).
ASCII/BINARY:	ASCII (1).
Default File Designator:	Formal file designator (000).
Record Format:	Fixed length (00).
Carriage Control:	No (0).
Label Option:	Standard label processing (0).
Disallow File Equation:	Disallow (1) if not specified; 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 default.
Forms Message:	None.
User labels:	None.
Block Factor:	MPE default calculation.
Number of Buffers:	2.
File Size:	1023.
Number of Extents:	8.
Initial Allocation of Extents:	1.
File Code:	0.
FCLOSE	
Disposition:	No change.
Security Code:	Normal (0).

The Relocatable Object File

When you compile a program using the :RPGXL command, RPG produces a relocatable object file. Relocatable object files are not directly executable. You must use the MPE/iX LINK command (see the *HP Link Editor/iX Reference Manual*) to create an executable program file for it.

You must produce a relocatable object file when you want to save the program as a permanent file. If the program uses external subroutines, you must create relocatable object files for each of the subroutines (and the RPG program), then link all of them together producing one executable program file.

Table 12-2 lists specific file characteristics that RPG uses when creating a relocatable object file or writing to a relocatable library file. The characteristics are separated into those used when the compiler opens the file (FOPEN intrinsic) and when it closes the file (FCLOSE intrinsic). For a definition of these characteristics, see the *MPE/iX Intrinsics Reference Manual*.

Table 12-2. Relocatable Object File Characteristics

FOPEN	
Formal File Designator:	RPGOBJ.
Foptions:	
Domain:	If an existing file is in passed state, RPGOBJ is an old temporary file in job file domain (10); if no existing file is in passed state, RPGOBJ 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 (0).
Disallow :FILE Equation:	If no file is specified, disallow (1). If a file is specified, allow (0).
Aoptions:	
Access Type:	Input/Output (0100).
Multirecord Access:	No (0).

Table 12-2. Relocatable Object File Characteristics (Continued)

Dynamic Locking:	No (0).
Exclusive:	Exclusive (0).
Inhibit Buffering:	No (0).
Record Size:	128 words.
Device Name:	MPE default.
Forms Message:	None.
User labels:	None written.
Block Factor:	MPE default calculations.
Number of Buffers:	2.
File Size:	400 (if $NEWPASS) or 4000.
Number of Extents:	2 (if $NEWPASS) or 8.
Initial Allocation of Extents:	1 extent.
File Code:	1461 (NMOBJ).
FCLOSE	
Disposition:	Permanent file (1) if $NEWPASS, temporary job file (2) if $OLDPASS, otherwise no change (0.)
Security Code:	Normal (0).

The List File

The compiler uses the list file for all listing output. This includes source, symbol table and Cross-Reference listings, input prompts (in session mode) and error and warning messages. If the list file is not assigned to $STDLIST (the standard listing device), the following information is written to it:

- The HP product number, version letter, update and fix levels of the compiler (beginning of compilation).
- Ending messages, the number of error and warning messages, the central processor time used and time that elapsed during compilation (end of compilation).

For examples of listing output, see the *HP RPG Programmer's Guide*.

When the compiler creates the list file, it does so using the file characteristics shown in Table 12-3. The characteristics are separated into those defined when the compiler opens the file (FOPEN intrinsic) and when it closes the file (FCLOSE intrinsic). For a description of these characteristics, see FOPEN and FCLOSE in the *MPE/iX Intrinsics Reference Manual*.

Table 12-3. List File Characteristics

FOPEN	
Formal File Designator:	RPGLIST.
Foptions: (%2514 if not specified; %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 not specified; formal designator (000) if specified.
Record Format:	Variable-length (01).
Carriage Control:	Yes (1).
Label Option:	Standard label processing (0).
Disallow :FILE Equation:	Yes (1) if not specified. No (0) if specified.
Aoptions (%001)	
Access Type:	Output only (0001).
Multirecord Access:	No (0).

Table 12-3. List File Characteristics (Continued)

Dynamic Locking:	No (0).
Exclusive:	Default (00).
Inhibit Buffering:	No (0).
Record Size:	Device record size.
Device Name:	MPE default.
Forms Message:	None.
User labels:	None written.
Block Factor:	MPE default calculation.
Number of Buffers:	2.
File Size:	5000 (if device is disk).
Number of Extents:	8.
Initial Allocation of Extents:	1 extent.
File Code:	0.
FCLOSE	
Disposition:	Permanent File (1).
Security Code:	Normal (0).

The Compiler Commands Reference

The sections which follow describe the compiler commands in detail. The commands are presented in alphabetical order.

You can use the compiler commands interactively (session mode) or you can place them in a job file (job mode). They can be executed after the BREAK key is pressed or after the CAUSEBREAK intrinsic is executed. Both compiler commands can be executed by the COMMAND intrinsic. See the *MPE/iX Intrinsics Reference Manual* for more information on CAUSEBREAK and COMMAND.

:RPGXL

This command compiles an RPG source program, producing a compiler listing and a relocatable object file. Use this command when you want to compile but not produce an executable program file.

To create an executable program file from a relocatable object file, use the LINK command of HP Link Editor/iX.

Syntax

`:RPGXL` *source_file [,relocatable_object_file] [,list_file]*

Parameters

source_file	The name of the source program file (this can be any ASCII file). The formal file designator for this file is RPGTEXT.
relocatable_object_file	The name of the file to which the relocatable object code is written. If you omit this parameter, $OLDPASS is used. The formal file designator for this file is RPGOBJ.
list_file	The name of the file to which the program listing is written (this can be any ASCII file). If you omit this parameter, the program listing is written to $STDLIST. The formal file designator for this file is RPGLIST.

Examples

1. This example compiles the source program APS540. The relocatable object file APO540 is created. The compiler listing is written to the file APL540.

```
:RPGXL APS540, APO540, APL540
```

:RPGXLGO

This command compiles an RPG source program, producing a relocatable object file; links the relocatable object file creating an executable program file; then executes the executable program file. The executable program file is temporary and is purged when you log off.

Syntax

`:RPGXLGO` *source_file [,list_file]*

Parameters

source_file	The name of the source program file (this can be any ASCII file). The formal file designator for this file is RPGTEXT.
list_file	The name of the file to which the program listing is written (this can be any ASCII file). If you omit this parameter, $STDLIST is used. The formal file designator for this file is RPGLIST.

Example

1. This example compiles the source program APS540. The compiler listing is written to $STDLIST and the executable program file is $OLDPASS.

```
:RPGXLGO APS540
```

:RPGXLLK

This command compiles an RPG source program, producing a relocatable object file. It then links the relocatable object file, producing an executable program file.

Syntax

`:RPGXLLK` *source_file [,executable_program_file] [,list_file]*

Parameters

source_file	The name of the source program file (this can be any ASCII file). The formal file designator for this file is RPGTEXT.
executable_program_file	The name of the file to which the executable program code is written. If you omit this parameter, $OLDPASS is used. The formal file designator for this file is RPGPROG.
list_file	The name of the file to which the program listing is written (this can be any ASCII file). If you omit this parameter, the program listing is written to $STDLIST. The formal file designator for this file is RPGLIST.

Example

1. This example compiles and the source program APS540. A relocatable object file $OLDPASS is created. $OLDPASS is linked, creating the executable program file APP540. The compiler listing is written to the file APL540.

```
:RPGXLLK APS540, APP540, APL540
```

The Compiler Subsystem Commands

Compiler subsystem commands are compiler directives that let you alter the normal defaults used during compilation. For example, you can suppress the listing of warning messages.

Compiler subsystem commands are part of the source program file and they are identified by a $ in column 6. The compiler subsystem commands are listed below and are described in detail later in this chapter:

Compiler Subsystem Command	Description
$CONTROL	Restricts access to the list file; suppresses source and Cross-Reference listings of indicators, fields, tables, arrays and files; suppresses warning messages; sets the maximum number of lines listed per page; suppresses informational messages; sets the maximum number of severe errors allowed; defines the delimiter character for alphanumeric literals; supresses creation of a forms file for WORKSTNC files.
$COPY	Enables the source library facility of RPG.
$IF	Determines whether portions of a program are compiled, based on software switches.
$INCLUDE	Inserts source library lines into the program.
$INCLUDENOW	Inserts source library lines into the program at the place this command is specified.
$PAGE	Establishes or changes the title on the compiler listing and advances the paper to top-of-form.
$SET	Sets the software switches that determine whether compilation is performed.
$TITLE	Establishes or changes the title on the compiler listing.

The Sequence Number Field (columns 1-5) in compiler subsystem command lines are not used (you may enter a number in this field if you wish). You can enter the subsystem commands in either upper-case or lower-case; they are interpreted as upper-case by the compiler.

The next three sections explain how to enter compiler system command parameters and comments and how to continue the subsystem commands from one line to the next. See the *HP RPG Programmer's Guide* for additional examples on using the compiler subsystem commands.

Entering Subsystem Command Parameters

Most of the compiler subsystem commands have one or more parameters. They specify various command options. You can enter parameters in any order. Separate them by commas (spaces are ignored). You can enter parameters in columns up to and including column 72. If you need more space, continue the parameters on another line (see the section "Entering Subsystem Command Continuation Lines").

When a subsystem command is executed, all of the parameters are read and interpreted before the command is executed. Parameters are processed in the order that you enter them. If you enter the same parameter twice or you enter conflicting parameters, the outcome depends on where they occur in the parameter list. For example, in the following $CONTROL subsystem command, LIST and NOLIST appear twice:

```
$CONTROL LIST,NOLIST,NOLIST,LIST
```

When $CONTROL is executed, the effective parameter list is:

```
$CONTROL LIST
```

Entering Subsystem Command Comments

To clarify the purpose and meaning of a compiler subsystem command, you may want to include comments with it. Comments are documentary and have no effect on the command itself.

You can enter comments after the subsystem command name. You can also enter comments before or after any parameter. Begin a comment with two less-than signs (<<) and end it with two greater-than signs (>>). You can enter comments through column 72 and you can include any ASCII character in them except >.

Do not embed comments within a parameter and do not use the & character to continue comments onto a second line. (See the next section, "Entering Subsystem Command Continuation Lines", for information on continuing comments onto a blank line.)

The following examples illustrate various ways to enter comments:

```
$PAGE<<PAGE EJECT,NO TITLE CHANGE.>>

$SET X1=ON,X2=ON,X3=ON<<SWITCHES 1-3 ON.>>

$SET X1=ON,X2=ON,<<LAST SW OFF>> X3=OFF
```

Entering Subsystem Command Continuation Lines

When a subsystem command and its parameters won't fit on one line, continue it onto the next line. Follow the last parameter on the line with an ampersand (&). Start the next line with a $ in column 6 and continue the parameters starting in column 7.

When you use continuation lines, the compiler concatenates them beginning with the character following the $. The $ and & are replaced by spaces.

When you use continuation lines, do not split a subsystem command, a parameter or a parameter string.

For example, to split this subsystem command after the parameter MAP,

```
$CONTROL LIST,SOURCE,WARN,MAP,LINES=36
```

enter two lines as follows:

```
$CONTROL LIST,SOURCE,WARN,MAP,&
$LINES=36
```

To continue comments onto a blank line (that contains no command parameters), enter the & after the ending >>. Continue the comments on the next line just as you would normally, except enter a $ in column 6. The following example shows how to continue comments onto blank lines:

```
$CONTROL NOWARN  <<WARNING MESSAGES ON TRIVIAL ERRORS>>&
$                <<WILL NOT BE LISTED> BUT MESSAGES ON>>&
$                <<FATAL ERRORS WILL APPEAR.>>
```

The Compiler Subsystem Commands Reference

The sections which follow discuss the compiler subsystem commands in detail. They are listed in alphabetical order.

$CONTROL

This subsystem command lets you override one or more of the listing and compiler defaults. You can place the $CONTROL subsystem command at the beginning of the source program file and also interspersed within it.

You can use $CONTROL to:

- Restrict compiler access to the list file.
- Request or suppress the source program and a Symbol Table listing.
- Define the number of lines per page for listing output.
- Print a Cross-Reference listing.

Syntax

```
$CONTROL [ERRORS=nnn] [,EXCQUIT] [,FKEYLBL] [,GEN] [,INFO]
         [,LINES=nnnn] [,LIST] [,MAP] [,NAME=source] [,NEWSAVE] [,NOGEN]
         [,NOINFO] [,NOLIST] [,NOMAP] [,NOOVFLOCHK] [,NOSOURCE] [,NOWARN]
         [,NOVALD] [,OVFLOCHK] [,QUOTE= {" or '}] [,RSPACE=n]
         [,SOURCE] [,WARN]
```

Parameters

ERRORS=*nnn*
Sets the maximum number of severe errors allowed during compilation to *nnn*. If there are more errors, compilation terminates. If *nnn* is already exceeded when this parameter is encountered, compilation terminates immediately. If you omit this parameter, the maximum number of severe errors is 100.

EXCQUIT
If a file open fails due to an EXCLUSIVE VIOLATION error (FSERR 90 or 91), RPG closes files and aborts the program immediately (instead of aborting the program with a fatal file error and printing a formatted dump). The system JCW is set to FATAL90 or FATAL91. It can be tested by MPE commands in a UDC or job stream as follows:

```
:RUN MYPROG (program includes $CONTROL EXCQUIT)
:IF JCW=FATAL90 OR JCW=FATAL91 THEN
:  RUN WAITPROG
:ELSE
:  RUN MYPROG2
:ENDIF
```

FKEYLBL
Allows applications using the RPG Screen Interface (RSI) to specify labels for enabled function keys (f2) through (f7). See the description of the SET operation in Chapter 8, and "Redefining Function Key Labels" in Chapter 11.

GEN
Allows the compiler to create an RSI CONSOLE forms file from Input Specifications in the program. The forms file is created if you omit this parameter.

INFO
Lists informational messages. Informational messages flag source statements that, although permissible, may not produce an intended result. For example, omitting File Description Specifications is legal but is not often done. This parameter

	remains in effect until a NOINFO parameter is encountered. If this parameter is omitted, informational messages are listed.
LINES=*nnnn*	Limits the number of lines printed on each list file page to *nnnn*. When page overflow occurs, the paper is advanced to top-of-form, the standard page heading is printed (followed by two blank lines) then the line that caused page overflow is printed. When you enter *nnnn*, include the heading and blank lines in it. This parameter remains in effect until another LINES parameter is encountered. If you omit this parameter, *nnnn* is 60 lines (for devices other than terminals) and 32767 lines for terminals.
LIST	Gives the compiler unrestricted access to the list file. (This enables you to use the SOURCE, MAP and LINES parameters). This parameter remains in effect until a NOLIST parameter is encountered. If you omit this parameter, the compiler has unrestricted access to the list file.
MAP	Prints a Cross-Reference listing after the source code listing (if LIST is in effect). This parameter remains in effect until NOMAP is encountered. If you omit this parameter at the beginning of compilation, no Cross-Reference listing is produced.
NAME=*source*	Assigns a name to the source program. The name must be a valid program name (see the Program Name Field in Chapter 2). This parameter overrides the name entered in the Program Name Field (columns 75-80) of the Header Specification.
NEWSAVE	Keeps all new files as permanent files (normally, new files are kept as temporary files). This parameter is equivalent to using an MPE file equation with the SAVE option.
NOGEN	Prohibits the compiler from creating an RSI CONSOLE forms file from Input Specifications in the program.
NOINFO	Stops the listing of informational messages. This parameter cancels a previous INFO parameter and remains in effect until another INFO parameter is encountered.
NOLIST	Lists (in the list file) only those source statements that contain errors. It also lists error messages and subsystem initiation and completion messages. This parameter remains in effect until LIST is encountered.
NOMAP	Stops the Cross-Reference listing. This parameter cancels a previous MAP parameter and remains in effect until another MAP is encountered.
NOOVFLOCHK	Resets numeric overflow checking (after using $CONTROL OVFLOCHK for full overflow checking) to partial overflow checking (the default). This allows RPG to left-truncate the computed value of a numeric operation if the Result Field is too small, and if the size of either the Factor 1 Field or the Factor 2 Field is as large as the size needed for the computed value. Otherwise the overflow trap is enabled.

NOSOURCE	Stops the listing of source text. This parameter cancels a previous SOURCE parameter.
NOWARN	Stops the listing of warning messages. This parameter cancels a previous WARN parameter and remains in effect until another WARN parameter is encountered.
NOVALD	Disables numeric validation. This allows nonnumeric characters to be mixed in with digits in numeric items. Beware, since this can cause strange things to happen to your program. This provides compatibility with RPG/V, where embedded blanks in a numeric field are retained as blanks if the field is output as an alphanumeric field. RPG/iX converts the blanks to zeroes if you do not specify this option.
OVFLOCHK	Enables full overflow checking on the arithmetic operations ADD, SUB, MULT, DIV, SQRT, XFOOT, Z-ADD, and Z-SUB. This causes a trap to the error routine if the computed value is too large for the result field. The Result Field is not automatically truncated as it can be with partial overflow checking (the default) or if the $CONTROL NOOVFLOCHK command is in effect.
QUOTE= { " , }	Defines the delimiter character for alphanumeric literals. Enter either a quotation mark (") or an apostrophe ('). You can use this parameter anywhere in the program, but you typically enter it at the beginning. If you omit this parameter, a quotation mark is used. This parameter does not apply to strings entered with compiler subsystem commands.
RSPACE=*n*	Adds additional spaces between output fields that use relative end positions. Enter a number (1-9) for *n*. If you omit this parameter, zero is used.
SOURCE	Lists the source program statements (if LIST is in effect). If you omit this parameter and the source program file and list file are assigned to a terminal, the source program statements and Symbol Table listing are not listed; otherwise, they are listed.
WARN	Lists warnings with the source program. Warnings are also written to the list file. This parameter remains in effect until NOWARN is encountered. If you omit this parameter, warnings are listed.

Example

The following $CONTROL subsystem command directs the compiler to list the source program, warnings, and a Cross-Reference listing. Only 36 lines are printed on each page of the listings.

```
$CONTROL LIST,SOURCE,WARN,MAP,LINES=36
```

$COPY

This subsystem command enables you to copy source statements into the program from other source files or libraries. Place this command at the beginning of the program when the program contains one or more $INCLUDE or $INCLUDENOW compiler subsystem commands.

See the $INCLUDE and $INCLUDENOW compiler subsystem commands for rules on using source libraries. The *HP RPG/iX Programmer's Guide* gives an example of how to use $COPY and $INCLUDE.

Syntax

```
$COPY
```

Parameters None.

$IF

This subsystem command lets you compile parts of a program rather than all of it. Place this command before and after sections of source code to turn compilation ON and OFF.

$IF and $SET are used together. $SET turns up to ten software switches ON or OFF. The $IF command tests these switches and compiles subsequent code when they match. When they do not match, subsequent source code is not compiled.

Even when compilation is turned OFF, source code is listed if the LIST and SOURCE parameters of the $CONTROL subsystem command are in effect.

Syntax

$IF [X*n*= { OFF / ON }]

Parameters

X*n* { OFF / ON }

This parameter names the software switch to test and the value to test for. If you omit this parameter, subsequent source code is compiled.

Enter a number from 0 to 9 (for *n*) that names the software switch.

To suspend compilation of subsequent source code, enter a status (either OFF or ON) that does not match the current status of the specified software switch. (The actions of the subsystem commands $PAGE and $TITLE are unaffected.) To resume compilation of subsequent source code, enter a status (either OFF or ON) that matches the current status of the specified software switch.

Example

The $SET subsystem command in the code below turns ON switches X4 and X5. Subsequent $IF statements turn compilation ON and OFF depending on the status of X4 and X5. Source code block 1 is compiled because the $IF test is true (X4 is turned ON). Source code block 2 is not compiled because the result of the second $IF test is false. Source code block 3 is compiled because the third $IF has no parameters.

```
            .
            .
$SET    X4=ON,X5=ON  <<TURN SWITCHES X4 AND X5 ON.>>
            .
            .
$IF     X4=ON        <<COMPILE SOURCE BLOCK 1.>>
            .
            .
        (source block 1)
            .
            .
$IF     X5=OFF       <<DO NOT COMPILE SOURCE BLOCK 2;>>&
$                    <<CANCEL PREVIOUS $IF COMMAND.>>
            .
            .
        (source block 2)
            .
            .
$IF                   <<CANCEL PREVIOUS $IF COMMANDS;>>&
$                     <<COMPILE SOURCE BLOCK 3.>>
            .
            .
        (source block 3)
            .
            .
```

$INCLUDE

This subsystem command copies source code from a source library into the source program before compilation. Place this command at the point in the source program where you want the source library facility to begin processing statements from another file. To use $INCLUDE, you must enable the source library facility by entering a $COPY subsystem command at the beginning of the main program.

To understand how $INCLUDE works, it helps to understand the source library facility (the preprocessor). Invoked when $COPY is the first record in your source program, the preprocessor opens ten temporary files to hold the following types of specifications:

1. Header and File Specifications
2. File Extension Specifications
3. Line Counter Specifications
4. Input Specifications (excluding DS records)
5. Input Specifications (DS records only)
6. Calculation Specifications (detail-time)
7. Calculation Specifications (total-time)
8. Calculation Specifications (subroutine)
9. Output Specifications
10. Table/Array Specifications

The preprocessor reads statements from the source program, placing them sequentially in the appropriate temporary file based on the specification type. Upon encountering a $INCLUDE command, the preprocessor stops reading the main program source file and reads instead the entire file named in the $INCLUDE command. The preprocessor adds the records from this file, possibly a mixture of specification types, to the end of the appropriate temporary file or files. When all library file records have been processed, the preprocessor resumes reading the original source file at the record following the $INCLUDE command. Any other $INCLUDE commands in the main file are processed the same way. When all records in the main file have been read, the preprocessor then copies the ten temporary files in order by specification into a single file that is passed to the compiler.

If your library file contains a Header Specification, the $INCLUDE command referencing it must be placed before any File Specifications in the source program. When the preprocessor encounters an Array/Table Specification (type A), a Compiler Subsystem Command (type $), a comment (type *), or a blank line, the statement is considered to be the same type as the previous statement. Do not use $INCLUDE within a compile-time table or array appended to the end of the program (after the first "**" separator line).

The $INCLUDE command line is listed on $STDLIST so you can verify its location. The inserted lines from the library file are identified with a C in column 5 of the source listing.

You can modify lines inserted from a source library. You do this by following the $INCLUDE subsystem command with a specification that identifies the line you want to modify and that contains the new information. For a File Description Specification, identify the specification by entering the same values in columns 7-14 as the specification you want to modify and enter new information in columns 15-74. For an Input Specification, identify the specification by entering the same values in columns 1-42 and 53-58 and enter new information in columns 43-52 and 59-70. When you leave a field blank, it remains unchanged. Enter an ampersand (&) in a field to blank it out.

See the *HP RPG Programmer's Guide* for an example of how to use $COPY and $INCLUDE. See also the related command $INCLUDENOW.

Syntax

$INCLUDE *file_name*[.*group*[.*account*]]

Parameters

file_name	Names the source library file containing the source statements to be inserted.
group	Identifies the MPE/iX group where the source library resides.
account	Identifies the MPE/iX account where the source library resides.

Example

Imagine a program with the following statement in the middle of its detail Calculation Specifications:

```
$INCLUDE PAF105.SOURCE
```

Also, a library file named PAF105 in the SOURCE group contains File Specifications, detail-time Calculation Specifications, total-time Calculation Specifications, and Output Specifications.

Because the first statement of the main program file is a $COPY statement, RPG invokes the preprocessor when the program is compiled. The preprocessor begins copying records from the main program into the temporary file created for that particular specification type. By the time the $INCLUDE command is encountered in the middle of the detail Calculation Specifications, the preprocessor has copied a Header Specification and some File Specifications to one temporary file, some Input Specifications to another, and some detail Calculation Specifications to another.

The $INCLUDE command directs the preprocessor to stop copying records from the main program file and to begin copying records from the file named in the statement, in this case PAF105. Since the first records in file PAF105 are File Specifications and detail-time Calculation Specifications, they are appended to the two appropriate temporary files, one containing the File Specifications and the other containing detail-time Calculation Specifications from the main program. Note that when the preprocessor next reads the total-time Calculation Specifications from the library file PAF105, the temporary file built to contain the total-time Calculation Specifications is empty (no total-time Calculation Specifications were read before the $INCLUDE command was encountered in the main source program). The total-time calculations from the library file will therefore precede any total-time calculations that may subsequently be read from the main program. Similarly, the Output Specifications from the library file will precede any Output Specifications in the main program.

If you are not aware of how the preprocessor works, you may inadvertently merge statements into your program in the wrong order, particularly if the library file contains more than one specification type.

$INCLUDENOW

This subsystem command copies source code from a source library into the source program before compilation. Place this command at the point in the source program where you want the library source code inserted.

Unlike the $INCLUDE command, the records in the source library file named in the $INCLUDENOW command must all be of the same specification type. The entire file is added at the point the command appears. Also, $INCLUDENOW positions Input Specification DS records and Calculation Specification subroutine records correctly.

To use $INCLUDENOW, you must enable the source library facility by entering a $COPY subsystem command as the first line of your program.

Do not use $INCLUDENOW within a compile-time table or array (after the first "**" separator line).

The $INCLUDENOW command line is listed on $STDLIST so you can verify its location. The inserted lines from the library file are identified with a C in column 5 of the source listing.

The $INCLUDENOW command does not provide a way to modify source lines brought in from source libraries.

See also the related command $INCLUDE.

Syntax

`$INCLUDENOW` *file_name*[.*group*[.*account*]]

Parameters

file_name	Names the source library file containing the source statements to insert.
group	Identifies the MPE/iX group where the source library resides.
account	Identifies the MPE/iX account where the source library resides.

Example

The following $INCLUDENOW command inserts all the records in the source library file PAF106 into the source program at the point where the command occurs. PAF106 resides in the SOURCE group.

```
$INCLUDENOW PAF106.SOURCE
```

$PAGE

This subsystem command advances the compiler listing to top-of-form, then prints a heading followed by two blank lines. If you wish, you can enter a title for the heading line of the new page (and for subsequent pages). This command makes a program easier to read because it isolates blocks of code and assigns descriptive titles to them.

If you do not enter a $PAGE or $TITLE command, the title is blank. $PAGE does not apply to the first page of a compiler listing, to the Cross-Reference pages or to listings directed to the terminal.

If the NOLIST parameter of the $CONTROL subsystem command is in effect, no paper advance or printing takes place. However, if you enter a parameter for this command, your entry becomes the current title for subsequent printing.

Syntax

```
$PAGE [string[,string]...]
```

Parameters

string — Enter one or more strings that, when concatenated, form the title. Enclose the strings in quotation marks (the quotation marks are not included in the final string). You can enter any ASCII character in the string including lower-case characters. The final string can contain up to 104 characters excluding the quotation marks and spaces between strings. To include a quotation mark in the string itself, enter two quotation marks.

The final string is printed in positions 29-132 of the heading line.

If you omit this parameter, the current title (set by a previous $PAGE or $TITLE subsystem command) is printed on the new page.

Example

The following $PAGE subsystem command skips to a new page and prints the title "GROSS PAY SUBROUTINE" in the heading line of the new page.

```
$PAGE "GROSS PAY SUBROUTINE"
```

$SET

This subsystem command enables you to compile parts of a program rather than all of it. Use this command to turn (up to 10) software switches ON or OFF.

$IF and $SET are used together. $SET turns up to ten software switches ON or OFF. The $IF command tests these switches and compiles subsequent code when they match. When they do not match, subsequent source code is not compiled.

Syntax

```
$SET [Xn= { OFF } [,Xn= { OFF } ] ... ]
         { ON  }       { ON  }
```

Parameters

```
Xn= { OFF }
    { ON  }
```

This parameter names the software switch and specifies whether to turn it ON or OFF. If you omit this parameter, all 10 software switches are turned OFF.

Enter a number from 0 to 9 (for *n*) that names the software switch.

Enter ON to turn the switch ON or OFF to turn it OFF.

Example

The $SET subsystem command in the code below turns ON switches X4 and X5. Subsequent $IF statements turn compilation ON and OFF depending on the status of X4 and X5. Source code block 1 is compiled because the $IF test is true (X4 is turned ON). Source code block 2 is not compiled because the result of the second $IF test is false. Source code block 3 is compiled because the third $IF has no parameters.

```
            .
            .
$SET    X4=ON,X5=ON  <<TURN SWITCHES X4 AND X5 ON.>>
            .
            .
$IF     X4=ON        <<COMPILE SOURCE BLOCK 1.>>
            .
            .
        (source block 1)
            .
            .
$IF     X5=OFF       <<DO NOT COMPILE SOURCE BLOCK 2;>>&
$                    <<CANCEL PREVIOUS $IF COMMAND.>>
            .
            .
        (source block 2)
            .
            .
$IF                   <<CANCEL PREVIOUS $IF COMMANDS;>>&
$                     <<COMPILE SOURCE BLOCK 3.>>
            .
            .
        (source block 3)
            .
            .
```

$TITLE

This subsystem command lets you specify the title to print in the heading line of the compiler listing. The title is printed on each heading line until another $TITLE command (or $PAGE command containing a new title) is encountered. This command lets you tailor the compiler listings by printing appropriate titles for them.

If you do not enter a $PAGE or $TITLE command, the title is blank. $TITLE does not apply to the first page of a compiler listing, to Cross-Reference pages or to listings directed to the terminal.

If the NOLIST parameter of the $CONTROL subsystem command is in effect, the heading is not printed but you can still use this command to specify a title to be used when printing resumes.

Syntax

```
$TITLE [string[,string]...]
```

Parameters

string

Enter one or more strings to concatenate to form the title. Enclose the strings in quotation marks (the quotation marks are not included in the final string). You can enter any ASCII character in the string, including lower-case characters. To include a quotation mark in the string itself, enter two quotation marks. The final string can contain up to 104 characters excluding quotation marks and spaces between strings.

The final string is printed in positions 29-132 of the heading line.

If you omit this parameter, a blank title is printed.

Example

The following $TITLE subsystem command specifies the title, "PAW300 - Calculate Withholding". This title is printed in the heading line of each compiler listing page as long as LIST is in effect and until another $TITLE or $PAGE command is encountered.

```
$TITLE "PAW300 - Calculate Withholding"
```

A

RPG Compiler Messages

This appendix lists the messages that you may encounter when compiling a program. The next two sections explain how to find messages in this appendix and how to determine their severity.

Message Numbers

Messages are listed in this appendix in order by their assigned numbers. Numbers are grouped in blocks, where most blocks correspond to a different specification type. The remaining blocks indicate general, miscellaneous, compiler, and compiler subsystem errors. The message number blocks are:

Message Number	Description
0001-0099	General (any specification)
1000-1099	Header (H)
2000-2099	File Description (F)
3000-3099	File Extension (E)
4000-4099	Line Counter (L)
5000-5099	Input (I)
6000-6099	Calculation (C)
7000-7099	Output (O)
8000-8099	Compiler Subsystem Command ($)
9000-9099	Other errors
15000-17999	Compiler errors. These errors are listed after the source program but before the Symbol Table and Cross-Reference listings and most of them cause the compiler to abort. Report these errors (just the last 4 digits) to your HP Support Engineer.

RPG prints message numbers at the end of the source lines in which they occur. The complete text of all messages is printed at the end of the compiler listing after the Symbol Table and Cross-Reference listings. See the *HP RPG/iX Programmer's Guide* for an example of how messages are shown in the compiler listing.

Message Types

There are three types of compiler messages: informational, warning, and terminal error messages. They are identified by a letter code appended to the error number. For example, the message number 1026W is a warning message because it has the letter code W (warning) appended to the message number.

Message Type	Description
I (Informational)	Reminds you that, although coding may be valid, it may not produce intended results. For example, informational messages notify you when you omit File Description Specifications.
W (Warning)	Indicates a possible problem; a reasonable alternative (default) exists and is chosen.
T (Terminal error)	Prevents successful compilation - no object program is produced. In most cases, the compiler assumes a default and continues processing. This allows the entire program to be scanned for errors.

0001W	MESSAGE	EXPECTS $ CONTINUATION, RECORD IGNORED
	CAUSE	You did not enter a complete compiler subsystem command on this record.
	ACTION	The compiler ignores this record. If necessary, correct the record.
0002T	MESSAGE	SEQUENCE NUMBER ERROR
	CAUSE	You did not enter the records making up the source program in the proper sequence (as indicated by the Sequence Number Field, columns 1-5).
	ACTION	Arrange the source program properly and recompile.
0003T	MESSAGE	NON-EXISTENT SPEC TYPE
	CAUSE	You specified an entry in the Form Type Field (column 6) that indicates a type of specification that no longer exists.
	ACTION	Enter the correct type and recompile the program.
0004T	MESSAGE	SPEC TYPE OUT OF ORDER
	CAUSE	You submitted a source program with at least one specification record out of proper sequence (according to type). For example, a program File Description Specification before the Header Specification would result in this error.
	ACTION	Rearrange and recompile the source program.
0011T	MESSAGE	MORE THAN 500 ERRORS AND WARNINGS FOUND, END COMPILE
	CAUSE	There are too many errors in the program.
	ACTION	Correct the program and recompile.
0012T	MESSAGE	NO SOURCE FILE NAMED. COMPILATION TERMINATED.
	CAUSE	No source file was named in the :RPGXL, :RPGXLLK, or RPGXLGO command, or no file equation was used to equate RPGTEXT to a source file in the :RUN RPGXL command.
	ACTION	Include a source file name in the command.
0013T	MESSAGE	$NULL NOT ALLOWED FOR SOURCE FILE.
	CAUSE	RPGXL does not accept source code input from this system file.
	ACTION	Place all source code into a named file or read it from $STDIN.
0014T	MESSAGE	EMPTY SOURCE FILE.
	CAUSE	No records were found in the named source file.
	ACTION	Be sure the source file contains at least one record to process.

0030W	MESSAGE	UNABLE TO CREATE RPGCOPY PROCESS. NO PREPROCESSING DONE
	CAUSE	RPGCOPY.PUB.SYS could not be run by the compiler.
	ACTION	Ensure that a current version of RPGCOPY exists in the PUB group of the SYS account and recompile.
0040T	MESSAGE	CANNOT CLOSE SOURCE FILE FOR $COPY PROCESSING
	CAUSE	Compiler error.
	ACTION	Contact HP Support Engineering.
0041T	MESSAGE	CANNOT RE-OPEN SOURCE FILE AFTER $COPY PROCESSING.
	CAUSE	Possible compiler error.
	ACTION	Contact HP Support Engineering.
0042T	MESSAGE	CANNOT FIND JCW 'RPGINCLUDE' FOR $COPY PROCESSING
	CAUSE	Compiler error.
	ACTION	Contact HP Support Engineering.
0049T	MESSAGE	COPYLIB FILE CANNOT HAVE BLOCKING FACTOR > 50
	CAUSE	A file specified in a $INCLUDE line has a file blocking factor greater than 50.
	ACTION	Rebuild the $INCLUDE file with a blocking factor no greater than 50.
0050T	MESSAGE	ERROR OPENING COMPILER COPYLIB FILE
	CAUSE	A file specified in a $INCLUDE line could not be opened.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0051T	MESSAGE	ERROR OPENING COMPILER 'RPGLIST' FILE
	CAUSE	File system error occurred when opening the RPGLIST file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0052T	MESSAGE	ERROR OPENING COMPILER 'RPGTEXT' FILE
	CAUSE	File system error occurred when opening the RPGTEXT file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0053T	MESSAGE	ERROR OPENING COMPILER 'RPGSOM' FILE
	CAUSE	File system error occurred when opening RPGSOM.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.

0058T	MESSAGE	ERROR OPENING COMPILER 'RPGTAB' FILE
	CAUSE	File system error occurred when opening the RPGTAB file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0060T	MESSAGE	END OF FILE FOR COMPILER COPYLIB FILE
	CAUSE	Compiler error.
	ACTION	Contact HP Support Engineering.
0061T	MESSAGE	END OF FILE FOR COMPILER 'RPGLIST' FILE
	CAUSE	RPGLIST is too small to hold all output records.
	ACTION	Build an RPGLIST file with more than 5000 records and recompile.
0063T	MESSAGE	END OF FILE FOR COMPILER 'RPGSOM' FILE
	CAUSE	RPGSOM is too small to hold all output records.
	ACTION	Build an RPGSOM file with more than 400 records and recompile.
0070T	MESSAGE	ERROR ACCESSING COMPILER COPYLIB FILE
	CAUSE	File system error occurred during the use of a file specification on a $INCLUDE line.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0071T	MESSAGE	ERROR ACCESSING COMPILER 'RPGLIST' FILE
	CAUSE	File system error occurred during the use of the RPGLIST file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0072T	MESSAGE	ERROR ACCESSING COMPILER 'RPGTEXT' FILE
	CAUSE	File system error occurred during the use of the RPGTEXT file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0073T	MESSAGE	ERROR ACCESSING COMPILER 'RPGSOM' FILE
	CAUSE	File system error occurred during the use of the RPGSOM file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.

0078T	MESSAGE	ERROR ACCESSING COMPILER 'RPGTAB' FILE
	CAUSE	File system error occurred during the use of the RPGTAB file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0080T	MESSAGE	ERROR CLOSING COMPILER COPYLIB FILE
	CAUSE	File system error occurred when closing a file specified on the $INCLUDE line.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0081T	MESSAGE	ERROR CLOSING COMPILER 'RPGLIST' FILE
	CAUSE	File system error occurred when closing the RPGLIST file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0082T	MESSAGE	ERROR CLOSING COMPILER 'RPGTEXT' FILE
	CAUSE	File system error occurred when closing the RPGTEXT file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0083T	MESSAGE	ERROR CLOSING 'RPGSOM' FILE
	CAUSE	File system error occurred when closing the RPGSOM file.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
0088T	MESSAGE	ERROR CLOSING COMPILER 'RPGTAB'
	CAUSE	File system error occurred when closing RPGTAB.
	ACTION	Refer to the File Information Display for the actual file system error that occurred and take appropriate action.
1001W	MESSAGE	MORE THAN ONE HEADER SPEC, SPECIFICATION DROPPED
	CAUSE	You included more than one Header Specification in your source program.
	ACTION	The compiler ignores any Header Specification after the first one. Remove the extra specifications.
1002W	MESSAGE	INVALID ERROR DUMP FILENAME IN COLUMNS 7-14, ASSUME BLANK
	CAUSE	You entered an invalid file name in the Error Dump Filename Field (columns 7-14).
	ACTION	The compiler ignores your entry and assumes you left these columns blank.

1003W	MESSAGE	COLUMN 15 (DEBUG OPTION) NOT BLANK OR 1, ASSUME 1.
	CAUSE	You specified a character other than 1 in the Debug Field (column 15).
	ACTION	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.
1004W	MESSAGE	COLUMNS 18-19 NOT BLANK, ASSUME BLANK.
	CAUSE	You entered data in columns 18-19 (not used in RPG).
	ACTION	The compiler ignores your entry and assumes you left these columns blank.
1005W	MESSAGE	COLUMN 21 (INVERTED PRINT) NOT BLANK, I, J, OR D, ASSUME BLANK.
	CAUSE	You entered a character other than an I, J, or D in the Inverted Print Field (column 21).
	ACTION	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.
1006W	MESSAGE	COLUMNS 23-24 NOT BLANK, ASSUME BLANK.
	CAUSE	You entered data in columns 23-24 (not used by RPG).
	ACTION	The compiler ignores your entry and assumes you left these columns blank.
1007W	MESSAGE	COLUMN 26 (ALTERNATE COLLATING SEQUENCE) NOT BLANK, S, OR O, ASSUME BLANK.
	CAUSE	You specified a character other than an S or O in the Alternate Collating Sequence Field (column 26).
	ACTION	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.
1008W	MESSAGE	COLUMNS 29-33 NOT BLANK, ASSUME BLANK.
	CAUSE	You entered data in columns 29-33 (not used by RPG).
	ACTION	The compiler ignores your entry and assumes you left these columns blank.
1009W	MESSAGE	COLUMN 34 (BINARY SEARCH OPTION) NOT BLANK OR B, ASSUME B.
	CAUSE	You specified a character other than B in the Table/Array Look-Up Field (column 34).
	ACTION	The compiler ignores your entry and assumes you entered a B for a binary look-up. If you do not wish a binary look-up, change this entry to blank.
1010W	MESSAGE	COLUMNS 35-38 NOT BLANK, ASSUME BLANK.
	CAUSE	You entered data in columns 35-38 (not used by RPG).
	ACTION	The compiler ignores your entry and assumes you left these columns blank.

1011W	MESSAGE	COLUMN 40 (SIGN CHECK) NOT BLANK, B, S, I, O, OR N, ASSUME BLANK.
	CAUSE	You entered a character other than a blank or B, S, I, O, or N in the Sign Process Field (column 40).
	ACTION	The compiler ignores your entry and assumes you left the field blank, specifying sign-forcing on output. If you want other sign-forcing or no sign-forcing, specify S, I, O, or N.
1012W	MESSAGE	COLUMN 41 (1P FORMS POSITIONING) NOT BLANK OR 1, ASSUME 1.
	CAUSE	You entered a character other than a 1 in the Form Positioning Field (column 41).
	ACTION	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.
1013W	MESSAGE	COLUMN 42 INDICATOR (INITIALIZATION) NOT BLANK OR S, ASSUME S.
	CAUSE	You entered a character other than S in the Indicator Setting Field (column 42).
	ACTION	The compiler ignores your entry and assumes you entered an S, turning ON 1P, L0, and all field indicators. Correct the entry, if necessary.
1014W	MESSAGE	COLUMN 43 (FILE TRANSLATION) NOT BLANK, F, OR O, ASSUME BLANK.
	CAUSE	You entered a character other than an F or O in the File Translation Field (column 43).
	ACTION	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.
1015W	MESSAGE	COLUMN 45 NOT BLANK, ASSUME BLANK.
	CAUSE	You entered data in column 45 (not used by RPG).
	ACTION	The compiler ignores your entry and assumes you left this column blank.
1016W	MESSAGE	COLUMN 47 (SKIP TO CHANNEL 1 SUPPRESS) NOT BLANK OR S, ASSUME S.
	CAUSE	You entered a character other than an S, in the Skip-Suppress Field (column 47).
	ACTION	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.
1017W	MESSAGE	COLUMN 48 (DSPLY OPTIONS) NOT BLANK B, D, N, OR S, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 48.
	ACTION	The compiler assumes that the entry was blank.

1018W	MESSAGE	COLUMN 52 (CROSS REFERENCE) NOT BLANK OR X, ASSUME X.
	CAUSE	You entered a character other than X in the Cross-Reference Listing Field (column 52).
	ACTION	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 column 52.
1019W	MESSAGE	COLUMN 53 (CARRIAGE CONTROL TYPE) NOT BLANK OR L, ASSUME L.
	CAUSE	You entered a character other than L in the Carriage Control Type Field (column 53).
	ACTION	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.
1020W	MESSAGE	COLUMN 54 (TEXT SEQUENCE CHECK) NOT BLANK OR N, ASSUME BLANK. (SEQ CHECK NOT SUPPORTED)
	CAUSE	You entered a character other than a blank or an N in the Textfile Sequence Check Field (column 54).
	ACTION	The compiler ignores your entry and assumes you left the field blank for no sequence checking.
1021W	MESSAGE	COLUMN 55 (RUN TIME ERROR CONTROL) NOT BLANK, N, OR S, ASSUME S.
	CAUSE	You entered a character, other than an N or S in the Error Log Field (column 55).
	ACTION	The compiler ignores your entry and assumes you entered an S, for transmitting error messages to the operator and providing a dump.
1022W	MESSAGE	COLUMN 56 (RUN TIME ERROR OPTION) NOT BLANK OR 1-5, ASSUME 5.
	CAUSE	You entered a character other than 1-5, in the Error Response Field (column 56).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1023W	MESSAGE	COLUMN 57 (RUN TIME ERROR OPTION) NOT BLANK OR 1-5, ASSUME 5.
	CAUSE	You entered a character other than 1-5, in the Error Response Field (column 57).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.

1024W	MESSAGE	COLUMN 58 (RUN TIME ERROR OPTION) NOT BLANK OR 1-5. ASSUME 5.
	CAUSE	You entered a character other than 1-5, in the Error Response Field (column 58).
	ACTION	The compiler ignores the entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1025W	MESSAGE	COLUMN 59 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5. ASSUME 5.
	CAUSE	You entered a character other than 0 or 2-5 in the Error Response Field (column 59).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1026W	MESSAGE	COLUMN 60 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5. ASSUME 5.
	CAUSE	You entered a character other than 0 or 2-5, in the Error Response Field (column 60).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1027W	MESSAGE	COLUMN 61 (RUN TIME ERROR OPTION) NOT BLANK, 0 OR 2-5. ASSUME 5.
	CAUSE	You entered a character other than 0 or 2-5, in the Error Response Field (column 61).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1028W	MESSAGE	COLUMN 62 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5, ASSUME 5.
	CAUSE	You entered a character other than 0-5, in the Error Response Field (column 62).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1029W	MESSAGE	COLUMN 63 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5, ASSUME 5.
	CAUSE	You entered a character other than 0-5, in the Error Response Field (column 63).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.

1030W	MESSAGE	COLUMN 64 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5. ASSUME 5.
	CAUSE	You entered a character other than 0-5, in the Error Response Field (column 64).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1031W	MESSAGE	COLUMN 65 (RUN TIME ERROR OPTION) NOT BLANK, 0, OR 2-5. ASSUME 5.
	CAUSE	You entered a character other than 0 or 2-5, in the Error Response Field (column 65).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1032W	MESSAGE	COLUMN 66 (RUN TIME ERROR OPTION) NOT BLANK, 0, OR 2-5. ASSUME 5.
	CAUSE	You entered a character other than 0 or 2-5, in the Error Response Field (column 66).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump, and ending the program.
1033W	MESSAGE	COLUMN 67 (RUN TIME ERROR OPTION) NOT BLANK, 0, OR 2-5, ASSUME 5.
	CAUSE	You entered a character other than 0 or 2-5, in the Error Response Field (column 67).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1034W	MESSAGE	COLUMN 68 (RUN TIME ERROR OPTION) NOT BLANK, 0, OR 2-5, ASSUME 5.
	CAUSE	You entered a character, other than 0 or 2-5, in the Error Response Field (column 68).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1035W	MESSAGE	COLUMN 69 (RUN TIME ERROR OPTION) NOT BLANK, 0, OR 2-5, ASSUME 5.
	CAUSE	You entered a character, other than 0 or 2-5, in the Error Response Field (column 69).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.

1036W	MESSAGE	COLUMN 70 (RUN TIME ERROR OPTION) NOT BLANK, 0, OR 2-5, ASSUME 5.
	CAUSE	You entered a character, other than 0 or 2-5, in the Error Response Field (column 70).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and ending the program.
1037W	MESSAGE	COLUMN 71 (RUN TIME ERROR OPTION) NOT BLANK OR 0-5, ASSUME 5.
	CAUSE	You entered a character other than 0 or 2-5 in the Error Response Field (column 71).
	ACTION	The compiler ignores your entry and assumes you entered a 5, for redirecting or suppressing error messages, printing a dump and the ending program.
1038W	MESSAGE	COLUMNS 75-80 (PROGRAM NAME) INVALID, ASSUME RPGOBJ.
	CAUSE	The program name does not begin with an alphabetic character (A-Z), the remaining characters are not alphanumeric, (A-Z, 0-9), or they contain embedded blanks.
	ACTION	The compiler ignores your entry and assumes you assigned the name RPGOBJ. Correct the entry if necessary.
1039W	MESSAGE	NO HEADER SPEC FOR RPG PROGRAM, ASSUME BLANK HEADER SPEC.
	CAUSE	You left the Header Specification out of your RPG program.
	ACTION	The compiler assumes a blank Header Specification. Correct the specification if necessary.
1040W	MESSAGE	COLUMNS 56-70 NOT BLANK WHEN COLUMN 55 CONTAINS S, ASSUME BLANK.
	CAUSE	You entered data in the Error Response Field (columns 56-70) when the Error Log Field (column 55) contains an S.
	ACTION	The compiler ignores your entry and assumes you left columns 56-70 blank.
1041W	MESSAGE	COLUMNS 72-74 NOT BLANK, ASSUME BLANK.
	CAUSE	You entered data in columns 72-74 (not used by RPG).
	ACTION	The compiler ignores your entry and assumes you left these columns blank.
1042W	MESSAGE	COLUMN 16 (U-SWITCH SOURCE) NOT BLANK, J, OR F, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 16.
	ACTION	The compiler assumes that the entry is blank.
1043W	MESSAGE	COLUMN 17 (UPDATE SOURCE) NOT BLANK OR F, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 17.
	ACTION	The compiler assumes that the entry is blank.

1044W	MESSAGE	COLUMN 27 NOT BLANK, ASSUME BLANK.
	CAUSE	You entered a character other than a blank in column 27 (not used by RPG).
	ACTION	The compiler assumes that the entry is blank.
1045W	MESSAGE	COLUMN 28 (BUFCHK DEFAULTS) NOT BLANK, C, N, B, U, OR X, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 28.
	ACTION	The compiler assumes that the entry is blank.
1046W	MESSAGE	COLUMN 25 (NAME LOGGING) NOT BLANK OR L, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 25.
	ACTION	The compiler assumes that the entry is blank.
1047W	MESSAGE	COLUMN 39 (EBCDIC ZONE/DIGIT TESTS) NOT BLANK OR E, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 39.
	ACTION	The compiler assumes that the entry is blank.
1048W	MESSAGE	COLUMN 44 (NON-NUMERIC DIGITS) NOT BLANK OR N, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 44.
	ACTION	The compiler assumes that the entry is blank.
1049W	MESSAGE	COLUMN 22 (RECORD NUMBER ADJUST) NOT BLANK, 1, +, OR 0, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 22.
	ACTION	The compiler assumes that the entry is blank.
1050W	MESSAGE	COLUMN 49 (RECORD LENGTH CHECK) NOT BLANK, N, OR E. ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 49.
	ACTION	The compiler assumes that the entry is blank.
1051W	MESSAGE	COLUMN 50 (PAGE OVERFLOW) NOT BLANK OR P. ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 50.
	ACTION	The compiler assumes that the entry is blank.
1052W	MESSAGE	COLUMN 51 (*PLACE METHOD) NOT BLANK OR 1. ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 51.
	ACTION	The compiler assumes that the entry is blank.

1053W	MESSAGE	COLUMN 20 (SAVE SOURCE LINE NO'S) NOT BLANK OR N. ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 20.
	ACTION	The compiler assumes that the entry is blank.
2001T	MESSAGE	FILE NAME PREVIOUSLY DEFINED IN COLUMNS 7-14, SPEC IS DROPPED.
	CAUSE	You assigned the file name used in the File Name Field (columns 7-14) in a previous File Description Specification.
	ACTION	The compiler ignores the specification. Correct the file name and recompile.
2002T	MESSAGE	INVALID FILETYPE ENTRY IN COLUMN 15, ASSUME I.
	CAUSE	You entered an invalid file type entry in the File Type Field (column 15).
	ACTION	The compiler ignores your file type entry and assumes you entered an I for Input File. If this is not an input file, change the entry to O, U, D, or C.
2003W	MESSAGE	FILE DESIGNATION IN COLUMN 16 IS INVALID FOR FILE TYPE, ASSUME SECONDARY.
	CAUSE	You entered a value in the File Designation Field (column 16) that is invalid for the file type.
	ACTION	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.
2004W	MESSAGE	INVALID END OF FILE ENTRY IN COLUMN 17, ASSUME BLANK.
	CAUSE	You entered an invalid end-of-file entry in the End-of-File Field (column 17).
	ACTION	The compiler ignores your end-of-file entry and assumes you left it blank.
2005W	MESSAGE	INVALID SEQUENCE ENTRY IN COLUMN 18, ASSUME BLANK.
	CAUSE	You entered an invalid sequence entry in the Input Sequence Check Field (column 18).
	ACTION	The compiler ignores your sequence entry and assumes you left it blank (to sequence check the matching file in ascending order). If you want to sequence check in descending order, change this entry to D.

2006W	MESSAGE	FILE FORMAT IN COLUMN 19 IS INVALID, ASSUME F.
	CAUSE	You entered an invalid file format in the Record Format Field (column 19).
	ACTION	The compiler ignores your entry and assumes the records in the file are fixed-length.
2007W	MESSAGE	INVALID BLOCK LENGTH IN COLUMNS 20-23, ASSUME EQUAL TO RECORD LENGTH.
	CAUSE	You entered a block length in the Block Length Field (columns 20-23) other than 1-9999 or blank.
	ACTION	The compiler ignores your entry and assumes the block length and record length are equal. Correct the block length if necessary.
2008W	MESSAGE	INVALID RECORD LENGTH IN COLUMNS 24-27, ASSUME RECORD LENGTH OF 80.
	CAUSE	You entered a record length in the Logical Record Length Field (columns 24-27) other than 1 through 9999.
	ACTION	The compiler ignores your entry and assumes the record length is 80 characters. Correct the record length.
2009T	MESSAGE	INVALID MODE OF PROCESSING ENTRY IN COLUMN 28.
	CAUSE	You entered a character other than L or R in the Processing Mode Field (column 28).
	ACTION	The compiler ignores your entry and assumes sequential processing. Enter a correct value (L or R) for the processing mode.
2010T	MESSAGE	INVALID LENGTH OF KEY FIELD OR RECORD ADDRESS FIELD IN COLUMNS 29-30.
	CAUSE	You entered a key length other than 1-99 or blank in the Record Address or Key Field Length Fields (columns 29-30).
	ACTION	The compiler ignores your entry and assumes that the key length is equal to 3.
2011W	MESSAGE	INVALID RECORD ADDRESS TYPE ENTRY IN COLUMN 31, ASSUME BLANK.
	CAUSE	You entered a character other than A, P, K, or I in the Record Address Type Field (column 31).
	ACTION	The compiler ignores your entry and assumes you left it blank, for a direct-access file not processed by a chaining file or RAF. For example, to specify processing with a chaining file or RAF by record number, change the entry to I.

2012W	MESSAGE	INVALID ENTRY IN COLUMN 32, ASSUME BLANK.
	CAUSE	You entered a character other than I, X, S, T, M, 1-9, D, or C in the File Organization/Additional I/O Area Field (column 32).
	ACTION	The compiler ignores your entry and assumes you left it blank, requesting two buffers.
2013W	MESSAGE	INVALID OVERFLOW INDICATOR IN COLUMNS 33-34, ASSUME BLANK.
	CAUSE	You entered an invalid overflow indicator in the Overflow Indicator Field (columns 33-34).
	ACTION	The compiler ignores your entry and assumes you left this field blank (no indicator assigned). To specify an indicator, enter OA-OG or OV.
2014W	MESSAGE	INVALID KEY FIELD LOCATION IN COLUMNS 35-38.
	CAUSE	You entered an invalid value in the Key Field Starting Location Field (columns 35-38).
	ACTION	The compiler ignores your entry and assumes a key field starting location of 0.
2015T	MESSAGE	IF YOU WISH TO PROCESS YOUR OWN LABELS, COLUMN 53 MUST BE E OR 2-9.
	CAUSE	You entered a character other than E or the digits 2-9 to process your own labels.
	ACTION	The compiler assumes no user label processing. To process labels, enter E or 2-9.
2016W	MESSAGE	COLUMNS 48-49 ARE NOT BLANK.
	CAUSE	You entered invalid data in columns 48-49 (not used by RPG).
	ACTION	The compiler ignores your entry and assumes you left these columns blank.
2017W	MESSAGE	INVALID ENTRY IN COLUMN 53.
	CAUSE	You entered a character, other than an S, E, or 2 through 9, in the Disk Labels Field (column 53).
	ACTION	The compiler assumes that you left the entry blank.
2018W	MESSAGE	COLUMNS 60-65 ARE NOT BLANK.
	CAUSE	You entered characters in these fields but did not enter K (for Continuation line) in column 53.
	ACTION	The compiler ignores your entry and assumes you left these columns blank.
2019W	MESSAGE	COLUMN 66 IS NOT 'A' OR BLANK, ASSUME BLANK.
	CAUSE	You entered a character, other than an A in the File Addition Field (column 66).
	ACTION	The compiler ignores your entry and assumes you left this field blank. To specify appending new records, change this entry to A.

2020W	MESSAGE	COLUMN 67 IS NOT BLANK.
	CAUSE	This column is not used, but you entered a character in it.
	ACTION	The compiler assumes that this column is blank.
2021W	MESSAGE	COLUMN 70 IS NOT BLANK.
	CAUSE	This column is not used, but you entered a character in it.
	ACTION	The compiler assumes that this column is blank.
2022W	MESSAGE	COLUMN 73-74 ARE NOT BLANK.
	CAUSE	This column is not used, but you entered a character in it.
	ACTION	The compiler assumes that this column is blank.
2023W	MESSAGE	INVALID FILE CONDITIONING INDICATOR IN COLUMN 71-72, ASSUME BLANK.
	CAUSE	You entered an invalid file conditioning indicator in the File Conditioner Field (columns 71-72). You may only enter U1-U8.
	ACTION	The compiler ignores your entry and assumes you left this field blank.
2024W	MESSAGE	MULTIPLE PRIMARY FILE DEFINED IN COLUMN 16, ASSUME SECONDARY.
	CAUSE	You defined more than one primary input file in the File Designation Field (column 16).
	ACTION	The compiler assumes that any primary file defined after the first primary file is a secondary file.
2025T	MESSAGE	IF 'SPECIAL' IN COLUMN 40-46, A ROUTINE NAME MUST BE IN COLUMN 54-59.
	CAUSE	You didn't enter a subroutine name in the Name of Label Exit Field (column 54-59) to process user labels or files on SPECIAL devices.
	ACTION	Enter a routine name in column 54-59 or remove SPECIAL from column 40-46 and recompile.
2026W	MESSAGE	INVALID NUMBER OF EXTENTS IN COLUMN 68-69, SYSTEM DEFAULT VALUE IS USED.
	CAUSE	You entered a number of extents less than 1 or greater than 15 in the Extents Field (columns 68-69).
	ACTION	The compiler ignores your entry and by system default assigns eight extents.
2027T	MESSAGE	FILE DESIGNATION ENTRY IN COLUMN 16 INVALID FOR OUTPUT OR DISPLAY FILE, ASSUME BLANK.
	CAUSE	You entered a value in the File Designation Field (column 16) that is invalid for the output or display file.
	ACTION	The compiler ignores your file designation entry and assumes you left it blank. Set the File Designation Field to blank or change the file type to I, U, or C.

2028W	MESSAGE	END OF FILE ENTRY IN COLUMN 17 INVALID FOR FILE TYPE.
	CAUSE	You entered a character in the End-of-File Field (column 17) for file type O or D, or for file designation C, T, or D.
	ACTION	The compiler assumes you entered a blank.
2029W	MESSAGE	ENTRY IN COLUMN 18 INVALID FOR TYPE OF FILE OR MODE OF PROCESSING, ASSUME BLANK.
	CAUSE	You entered a letter or digit, other than an A or D, in the Input Sequence Field (column 18) that is invalid for the file type or mode of processing.
	ACTION	The compiler assumes you entered a blank.
2030W	MESSAGE	OVERFLOW INDICATOR IN COLUMN 33-34 PREVIOUSLY ASSIGNED, ASSUME BLANK.
	CAUSE	You previously assigned an overflow indicator in the Overflow Indicator Field (column 33-34).
	ACTION	The compiler ignores your entry and assumes you left this field blank. If you wish overflow processing, enter an unused overflow indicator.
2031W	MESSAGE	INVALID OR BLANK EXTENSION CODE ENTRY IN COLUMN 39 FOR TABLE OR RECORD ADDRESS FILE, ASSUME E.
	CAUSE	You entered a character other than an E or L in the Extension Code Field (column 39) for a table file or Record Address File.
	ACTION	The compiler ignores your entry and assumes you entered an E to define a table file or RAF in a File Extension Specification.
2032W	MESSAGE	EXTENSION CODE IN COLUMN 39 IS INVALID, ASSUME BLANK.
	CAUSE	You entered a letter or digit, other than an E or L, in the Extension Code Field (column 39).
	ACTION	The compiler ignores your entry and assumes you left it blank.
2033W	MESSAGE	RECORD LENGTH IS LARGER THAN BLOCK LENGTH, ASSUME BLOCK LENGTH EQUAL TO RECORD LENGTH.
	CAUSE	You specified a record length larger than the block length.
	ACTION	The compiler ignores your entry and assumes the block length is equal to the record length.
2034W	MESSAGE	BLOCK LENGTH IS NOT A MULTIPLE OF RECORD LENGTH, ASSUME UNBLOCK.
	CAUSE	You specified a block length that is not a multiple of the record length.
	ACTION	The compiler ignores your entry and assumes there is no blocking (the physical and logical record lengths are identical).

2035I	MESSAGE	NO FILE DESCRIPTION SPECIFICATION FOUND.
	CAUSE	You did not enter a File Description Specification in the source program.
	ACTION	The compiler assumes that no file I/O will be used by the program.
2036W	MESSAGE	NO PRIMARY FILE SPECIFIED IN COLUMN 16, ASSUME FIRST SECONDARY FILE AS PRIMARY.
	CAUSE	You didn't specify a primary file in the File Designation Field (column 16).
	ACTION	The compiler assumes that the first secondary file is the primary file.
2037W	MESSAGE	COLUMNS WHICH SHOULD BE BLANK FOR CONTINUATION LINE ARE NOT BLANK, ASSUME BLANK.
	CAUSE	You made an entry in these columns on a File Description Continuation line.
	ACTION	The compiler assumes these columns are blank.
2038W	MESSAGE	CONTINUATION LINE IS NOT ALLOWED AT THIS POINT.
	CAUSE	The first line in the File Description Specifications is a continuation line.
	ACTION	The compiler ignores this continuation line.
2039W	MESSAGE	CONTINUATION ENTRY IN COLUMN 54-59 IS REPEATED FOR A FILE, SECOND ENTRY IGNORED.
	CAUSE	You repeated a continuation line entry; for example, you specified RDEXIT in two continuation lines for the same file.
	ACTION	The compiler ignores the second entry.
2040W	MESSAGE	ENTRY IN COLUMN 54-59 OF A CONTINUATION RECORD IS INVALID OR MISSING.
	CAUSE	You entered invalid information in the Option Type Field (columns 54-59) or left this field blank.
	ACTION	The compiler ignores this continuation line.
2041W	MESSAGE	INVALID FILENAME IN COLUMN 7-14, SPEC IS DROPPED.
	CAUSE	You entered an invalid file name in the File Name Field (column 7-14).
	ACTION	The compiler drops the specification; redefine the file name and recompile.
2042T	MESSAGE	ONLY ONE RECORD ADDRESS FILE PER PROGRAM IS ALLOWED, ASSUME SECONDARY.
	CAUSE	You already specified one Record Address File in the program.
	ACTION	The compiler assumes the second file designated is a secondary file.

2043W	MESSAGE	A RECORD ADDRESS FILE CANNOT BE A 'SPECIAL' FILE.
	CAUSE	You specified a Record Address File as a SPECIAL file.
	ACTION	The compiler assumes this file is not a special file.
2044I	MESSAGE	NO PRIMARY OR SECONDARY FILE SPECIFICATION IN COLUMN 16.
	CAUSE	In a program that uses multifile input, you defined no primary or secondary input files.
	ACTION	Make the appropriate file definitions and recompile the program.
2045W	MESSAGE	THIS CONTINUATION RECORD IS NOT ALLOWED FOR A NON-IMAGE FILE, SPEC IS DROPPED.
	CAUSE	You entered a File Description Continuation line that is only permitted for an TurboIMAGE file.
	ACTION	The compiler ignores this record.
2046W	MESSAGE	ILLEGAL DATA BASE NAME IN COLUMNS 60-65.
	CAUSE	You entered an invalid TurboIMAGE database name.
	ACTION	Enter a correct TurboIMAGE database name.
2047W	MESSAGE	ILLEGAL OPEN MODE IN COLUMN 66, ASSUME 2.
	CAUSE	You entered a character other than 1, 2, 3, or blank in the Open Mode Field (column 66).
	ACTION	The compiler assumes Mode 2, Update Shared. Enter the correct mode.
2048W	MESSAGE	ILLEGAL INPUT/OUTPUT MODE IN COLUMN 67.
	CAUSE	You entered a character other than 2 through 8, S, B, C, or R, in the Input/Output Mode Field (column 67).
	ACTION	The compiler assumes Mode 2. Enter the correct mode.
2049W	MESSAGE	RECORD ADDRESS TYPE IN COLUMN 31 MUST BE 'A', 'P', 'K', OR 'I' FOR IMAGE FILE, ASSUME 'A'.
	CAUSE	You entered an improper character in the Record Address Type Field (column 31).
	ACTION	The compiler assumes you entered A (alphanumeric keys).
2050W	MESSAGE	RECORD ADDRESS TYPE IN COLUMN 31 MUST BE 'I' FOR DIRECT FILE.
	CAUSE	The TurboIMAGE file is a direct-access file, but you specified a record address type other than I (retrieve by record number).
	ACTION	The compiler assumes Mode I. Enter the correct mode, if necessary.

2051T	MESSAGE	MODE OF PROCESSING IN COLUMN 28 MUST BE 'R' FOR DIRECT FILE, ASSUME R.
	CAUSE	The TurboIMAGE file is a direct-access file, but you specified a processing mode other than R (random).
	ACTION	The compiler assumes you entered R. Enter the correct mode if necessary.
2052T	MESSAGE	RECORD ADDRESS TYPE IN COLUMN 31 MUST BE BLANK FOR SEQUENTIAL OR TAG FILE, ASSUME BLANK.
	CAUSE	The TurboIMAGE file is a sequential or TAG file, but you specified a record address type other than blank (sequential).
	ACTION	The compiler assumes you left this field blank. Enter the correct record address type.
2054T	MESSAGE	L IN COLUMN 28 CAN ONLY BE USED WITH C OR R IN COLUMN 67 FOR PREVIOUS FILE.
	CAUSE	You specified L (for sequential processing between limits for this file) but a conflicting Input/Output Mode for the previous file.
	ACTION	The compiler assumes column 28 is blank. Enter the correct input/output mode in column 28.
2055T	MESSAGE	ITEM NAME MISSING FOR 'L' IN COLUMN 28 FOR PREVIOUS FILE.
	CAUSE	You did not supply the required ITEM Continuation line.
	ACTION	The compiler terminates your job at the end of compilation; enter the ITEM line.
2056T	MESSAGE	ITEM NAME MISSING FOR INPUT/OUTPUT MODE 5, 6, C, OR R FOR PREVIOUS FILE.
	CAUSE	You did not provide the corresponding ITEM Continuation line for this chained file.
	ACTION	The compiler terminates your job at the end of compilation. Enter the ITEM Continuation line and recompile.
2058T	MESSAGE	DATA BASE NAME MISSING FOR PREVIOUS FILE.
	CAUSE	You did not provide a Database Name Record for this file.
	ACTION	The compiler terminates your job at the end of compilation. Enter the Database Name line and recompile.
2059T	MESSAGE	FORMS, BATCH, AND TRACE FILES MAY ONLY BE SPECIFIED FOR WORKSTN FILE.
	CAUSE	You included a FORMS BATCH or TRACE Continuation line for a non-WORKSTN file.
	ACTION	Correct the program and recompile.

2060T	MESSAGE	TRACE FILE NAME MUST BE NON-BLANK AND BEGIN IN COLUMN 60.
	CAUSE	The TRACE file name is missing or does not begin in column 60.
	ACTION	Enter the name beginning in column 60.
2061W	MESSAGE	WORKSTN FILE MUST BE UPDATE OR COMBINED. ASSUME UPDATE.
	CAUSE	A file on device WORKSTN is not declared as update or combined.
	ACTION	The compiler assumes the WORKSTN file is an update file.
2062W	MESSAGE	WORKSTN FILE MUST BE DEMAND OR PRIMARY, ASSUME PRIMARY.
	CAUSE	A file on device WORKSTN is not declared as demand or primary.
	ACTION	The compiler assumes WORKSTN file is primary.
2063W	MESSAGE	WORKSTN FILE MUST HAVE VARIABLE LENGTH RECORDS, ASSUME VARIABLE.
	CAUSE	A file on device WORKSTN has fixed length records.
	ACTION	The compiler assumes the WORKSTN file has variable length records.
2064W	MESSAGE	COLUMN 52 MUST BE BLANK FOR FILES NOT ON DEVICE WORKSTN.
	CAUSE	Column 52 is not blank for the non-WORKSTN file.
	ACTION	The compiler assumes column 52 is blank.
2065W	MESSAGE	INVALID ENTRY IN COLUMN 52, ASSUME BLANK.
	CAUSE	Column 52 contains an invalid entry for the WORKSTN file.
	ACTION	The compiler assumes column 52 is blank.
2066W	MESSAGE	COLUMN 51 MUST BE BLANK FOR FILES NOT ON DEVICE WORKSTN.
	CAUSE	Column 51 is not blank for the non-WORKSTN file.
	ACTION	The compiler assumes column 51 is blank.
2067W	MESSAGE	INVALID ENTRY IN COLUMN 51, ASSUME BLANK.
	CAUSE	Column 51 contains an invalid entry for the WORKSTN file.
	ACTION	The compiler assumes column 51 is blank.
2068W	MESSAGE	FILE ON DEVICE $STDIN MUST BE INPUT, ASSUME INPUT.
	CAUSE	A file declared on device $STDIN is not an input file.
	ACTION	The compiler assumes $STDIN is the input file.
2069W	MESSAGE	FILE ON DEVICE $STDLIST MUST BE OUTPUT OR DISPLAY, ASSUME OUTPUT.
	CAUSE	A file that uses device $STDLST is not an output or display file.
	ACTION	The compiler assumes the $STDLST file is an output file.

2070W	MESSAGE	KEY FIELD/RECORD ADDRESS FIELD LENGTH MUST BE ENTERED FOR THIS FILE.
	CAUSE	No key field length is entered for a chaining type file.
	ACTION	Correct the program and recompile.
2071W	MESSAGE	KEY FIELD STARTING LOCATION MUST BE ENTERED FOR THIS FILE.
	CAUSE	No key field starting location is entered for a chaining file.
	ACTION	Correct the program and recompile.
2072I	MESSAGE	DEVICE NAME NOT LEFT-JUSTIFIED IN FIELD, ASSUME JUSTIFIED.
	CAUSE	You specified a device name that is not left-justified in the Device Class Name Field (columns 40-46).
	ACTION	The compiler left justifies the entry.
2073W	MESSAGE	FILE SPECIFIED AS CHAINED ACCESS MUST BE RANDOM PROCESSED, ASSUMED R IN COLUMN 28.
	CAUSE	You specified a file as chained access, but did not specify random processing.
	ACTION	The compiler assumes you entered R in column 28.
2074W	MESSAGE	INPUT/OUTPUT MODE MUST BE 5, 6, 7, 8, C, OR R FOR RECORD LEVEL LOCKING, ASSUME 1 IN COLUMN 66.
	CAUSE	You specified R (Record Locking) in column 66, but used an incorrect Input/Output Mode in column 67. This sequence is invalid.
	ACTION	The compiler assumes you entered 1 (database locking per record) in column 66.
2075W	MESSAGE	LOCKING MODE INCONSISTENT FOR DATA BASE, FIRST MODE IS B, MODES AFTER ARE DIFFERENT, ASSUME B.
	CAUSE	You specified different locking modes following an initial locking mode of B. This sequence is invalid.
	ACTION	The compiler assumes the mode B for all accesses.
2076W	MESSAGE	LOCKING MODE INCONSISTENT FOR DATA SET, FIRST MODE IS S, MODES AFTER ARE DIFFERENT, ASSUME S.
	CAUSE	You specified different locking modes following an initial locking mode of S. This sequence is invalid.
	ACTION	The compiler assumes the mode S for all accesses.

2077W	MESSAGE	LOCKING MODE INCONSISTENT, MODE FOLLOWING FIRST LOCK MODE ARE DIFFERENT; ASSUME FIRST MODE.
	CAUSE	You specified different locking modes following an initial locking mode. This sequence is invalid.
	ACTION	Correct the program for locking mode consistency and recompile.
2078T	MESSAGE	LOCKING MODE IS INCONSISTENT, FIRST MODE IS S, THE FOLLOWING MODE IS B.
	CAUSE	You specified different locking modes following an initial locking mode.
	ACTION	Correct the program for locking mode consistency and recompile.
2079T	MESSAGE	LOCKING MODE INCONSISTENT, MODES FOLLOWING FIRST LOCK MODE ARE OF HIGHER PRECEDENCE.
	CAUSE	You specified different locking modes following an initial locking mode.
	ACTION	Correct the program for locking mode consistency and recompile.
2080T	MESSAGE	FILE DESCRIBED AS AN OUTPUT CHAIN FILE MUST BE A DIRECT FILE, NOT A KSAM, IMAGE, OR INDEXED FILE.
	CAUSE	You specified a file as OC in columns 15-16; therefore, column 32 must be specified as a direct file.
	ACTION	Correct the program to specify blank, D, or 1-7 in column 32, and recompile.
2081I	MESSAGE	FOR UPDATE-ADD FILE WITHOUT 'UPDATE-PROTECT CHECK', ENSURE THAT NO UPDATE IS INTERRUPTED BY AN ADD.
	CAUSE	You specified a file as update without specifying the update-protect check option in column 28 of the Header Specification.
	ACTION	Add the update protect check option (U or X in column 28) or ensure that the program does not attempt to add a record while an update operation is in progress.
2082W	MESSAGE	COLUMN 50 MUST BE BLANK FOR FILES NOT ON DEVICE WORKSTN.
	CAUSE	You specified an incorrect entry for column 50.
	ACTION	The compiler assumes blank.
2083W	MESSAGE	INVALID ENTRY IN COLUMN 50, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 50.
	ACTION	The compiler assumes blank.
2084W	MESSAGE	COLUMN 32 FOR WORKSTN FILE MUST BE BLANK, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 32.
	ACTION	The compiler assumes blank.

2085W	MESSAGE	BUFCHK SPEC ONLY ALLOWED FOR SEQUENTIAL, DIRECT, AND INDEXED FILES, SPEC IS DROPPED.
	CAUSE	You specified a BUFCHK Continuation line for a file that is not a sequential, direct, or indexed file.
	ACTION	The compiler ignores the BUFCHK Continuation line.
2086W	MESSAGE	BUFCHK SPEC COLUMN 60 (CURRENT DATA CHECK) MUST BE BLANK OR N, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 60.
	ACTION	The compiler assumes blank.
2087W	MESSAGE	BUFCHK SPEC COLUMN 61 (NO-READ CHECK) MUST BE BLANK OR N, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 61.
	ACTION	The compiler assumes blank.
2088I	MESSAGE	BUFCHK 'NO-READ CHECK' DISABLED FOR NON-UPDATE FILE.
	CAUSE	You specified the NO-READ CHECK option in column 28 of the Header Specification and a BUFCHK Continuation line with a blank in column 61 for a non-update file. This sequence is invalid.
	ACTION	The compiler disables the NO-READ CHECK option for this file.
2089W	MESSAGE	BUFCHK SPEC COLUMN 62 (UPDATE-PROTECT CHECK) MUST BE BLANK OR N, ASSUME BLANK.
	CAUSE	You specified an incorrect entry for column 62.
	ACTION	The compiler assumes blank.
2090I	MESSAGE	BUFCHK 'UPDATE-PROTECT CHECK' DISABLED FOR NON-UPDATE FILE.
	CAUSE	You specified the UPDATE-PROTECT CHECK option in column 28 of the Header Specification and a BUFCHK Continuation line with a blank in column 62 for a non-update file. This sequence is invalid.
	ACTION	The compiler disables the UPDATE-PROTECT CHECK option for this file.
2091I	MESSAGE	BUFCHK 'CURRENT DATA CHECK' DISABLED FOR ALL NON-LOCKING FILES HAVING A 'BUFCHK' CONTINUATION REC.
	CAUSE	You specified the CURRENT DATA CHECK option in column 28 of the Header Specification, and a BUFCHK Continuation line for files for which locking is not specified.
	ACTION	The compiler disables the CURRENT DATA CHECK option for the file(s).

2093W	MESSAGE	FORMS DOWN-LOAD VALUE IN COLUMNS 60-62 MUST BE 1-255, ASSUME 1.
	CAUSE	You specified an incorrect entry for columns 60-62 (the entry may not be right-justified).
	ACTION	The compiler assumes 1.
2094I	MESSAGE	BUFCHK 'CURRENT DATA CHECK' DISABLED FOR IMAGE FILE.
	CAUSE	You specified a BUFCHK Continuation line for a TurboIMAGE file. This sequence is invalid.
	ACTION	The compiler disables the CURRENT DATA CHECK option for this file specification.
2095W	MESSAGE	KEYFL SPEC CONTAINS INVALID PARAMETER(S), OR IS SPECIFIED FOR A NON-KSAM FILE.
	CAUSE	Columns 68-70 contained invalid parameter information or you specified the KEYFL Continuation line for a non-KSAM file.
	ACTION	The compiler disregards your entry. Correct the program and recompile.
2096W	MESSAGE	WORKSTN INTERFACE TYPE NOT BLANK, R, OR C, ASSUME BLANK.
	CAUSE	You entered an incorrect value in column 47 of the File Description Specification.
	ACTION	The compiler assumes blank.
2098W	MESSAGE	FORMDL MESSAGE NOT SPECIFIED PRIOR TO LOADFM OPTION - ASSUME NO. DOWNLOADED IS 1.
	CAUSE	A FORMDL Specification must precede a LOADFM Specification.
	ACTION	The compiler assumes that one form is to be loaded.
2100W	MESSAGE	INVALID ERROR OPTION ON KFATAL RECORD, DEFAULTING TO 0.
	CAUSE	You entered an option number that was not 0-5.
	ACTION	The compiler assumes option 0.
2101T	MESSAGE	KSTATUS MUST ALSO BE SPECIFIED FOR FILES USING THE KFATAL OPTION.
	CAUSE	A KSTATUS record for this file was not found, although a KFATAL record was used.
	ACTION	Add a KSTATUS record for the file and recompile.
3001T	MESSAGE	TO FILENAME (COLUMNS 19-26) INVALID OR UNDEFINED - RECORD IGNORED.
	CAUSE	You entered an invalid or undefined file name in the To File Name Field (columns 19-26). (An undefined file name is one that does not appear in the File Description Specifications.)
	ACTION	Define the To File Name and recompile.

3002T	MESSAGE	FROM FILENAME (COLUMNS 11-18) INVALID OR UNDEFINED - RECORD IGNORED.
	CAUSE	You entered an invalid or undefined file name in the From File Name Field (columns 11-18). (An undefined file name is one that does not appear in the File Description Specifications.)
	ACTION	Define the correct From File Name and recompile.
3003W	MESSAGE	E NOT SPECIFIED ON FILE DESCRIPTIONS FOR FROM FILE.
	CAUSE	You did not enter an E on the File Description Specification (column 39) for the From File Name Field (columns 11-18).
	ACTION	Enter an E in column 39 of the File Description Specification for the From File.
3004T	MESSAGE	FROM FILE IS NOT AN INPUT FILE - RECORD IGNORED.
	CAUSE	You specified a non-input file in the From File Name Field (column 11-18).
	ACTION	Enter the correct file in the From File Name Field and recompile.
3005W	MESSAGE	CHAINING OR RECORD ADDRESS FILE AND COLUMNS 33-57 NON BLANK. BLANK ASSUMED.
	CAUSE	You entered data in columns 33-57, which should remain blank for chaining or Record Address Files.
	ACTION	Remove the data from columns 33-57.
3006T	MESSAGE	CHAINING FIELD NUMBER (COLUMN 10) NOT 1-9 - RECORD IGNORED.
	CAUSE	You entered a character other than one of the digits 0 through 9 in column 10.
	ACTION	Specify the correct digit and recompile.
3007W	MESSAGE	CHAINING FIELD CODE (COLUMN 9) NOT C-ASSUME C.
	CAUSE	You did not enter a C in column 9 of the Chaining Field Code Field (columns 9-10).
	ACTION	The compiler assumes you entered a C in column 9, and combines this with the digit in column 10 to form the chaining code. Correct column 9.
3008T	MESSAGE	CHAINING CODE AND FROM FILE PREVIOUSLY SPECIFIED - RECORD IGNORED.
	CAUSE	You previously specified the chaining code in the Chaining Field Code Fields (columns 9-10) and the from file in the From File Name Field (columns 11-18).
	ACTION	The compiler ignores the specification. Remove the duplicate chaining description.

3009T	MESSAGE	TO FILE NOT A CHAINED OR RANDOM FILE - RECORD IGNORED.
	CAUSE	You did not specify a chained or direct-access file in the To File Name Field.
	ACTION	Specify the correct type of file and recompile.
3013T	MESSAGE	RECORD ADDRESS FILE PREVIOUSLY SPECIFIED ON EXTENSION SPECIFICATION - RECORD IGNORED.
	CAUSE	You specified a record address file in a previous File Extension Specification.
	ACTION	Set up program so that there is only one record address file.
3014T	MESSAGE	TO FILENAME (COLUMNS 19-26) MISSING FOR CHAINING OR RAF - SPECIFICATION IGNORED.
	CAUSE	You did not enter a file name in the To File Name Field (columns 19-26) for the chaining file or the Record Address File you specified.
	ACTION	Specify the file name and recompile.
3015T	MESSAGE	TABLE/ARRAY NAME (COLUMNS 27-32) PREVIOUSLY DEFINED - RECORD IGNORED.
	CAUSE	You previously defined the table/array name in the Table/Array Name Field (column 27-32).
	ACTION	Change the name and recompile.
3016T	MESSAGE	ENTRIES PER RECORD (COLUMNS 33-35) INVALID - ASSUME 1.
	CAUSE	You specified an invalid number of entries in one record in the Entries Per Record Field (columns 33-35). (The number of entries must be from 1 to 999.)
	ACTION	The compiler ignores your entry and assumes you entered a 1. Enter valid numeric characters, right-justified with no embedded blanks, and recompile.
3017T	MESSAGE	ENTRIES PER TABLE/ARRAY (COLUMNS 36-39) INVALID - ASSUME 1.
	CAUSE	You entered an invalid number of entries in the Entries Per Table/Array Field (columns 36-39). The number of entries must be 1-9999.
	ACTION	The compiler ignores your entry and assumes you entered a 1. Enter valid numeric characters, right-adjusted with no embedded blanks, and recompile.
3018W	MESSAGE	P/B/L/R FIELD (COLUMN 43) INVALID - BLANK ASSUMED.
	CAUSE	You entered a character other than P, B, L or R, in the Data Format Field (column 43).
	ACTION	The compiler ignores your entry and assumes you left it blank. Enter a valid character in this field.

3019W	MESSAGE	FIELD SIZE OF 1 NOT ALLOWED WITH L OR R SPECIFIED IN COLUMN 43 - 2 ASSUMED.
	CAUSE	You specified a field size of 1 in the Entry Length Field (column 40-42) with an L or R format in the Data Format Field (column 43).
	ACTION	The compiler ignores your entry and assumes you specified a field size of 2 characters. Specify a field size of at least 2 characters.
3020T	MESSAGE	FIELD SIZE (COLUMNS 40-42) BLANK OR INVALID - 1 ASSUMED.
	CAUSE	You entered an invalid field size or left the Entry Length Field (columns 40-42) blank.
	ACTION	The compiler ignores your entry and assumes you specified a field size of 1 character. Enter valid numbers, right-adjusted with no embedded blanks.
3021W	MESSAGE	DECIMAL POSITIONS (COLUMN 44) INVALID - ASSUME NUMERIC WITH ZERO DECIMAL POSITIONS.
	CAUSE	You entered an invalid number of digit positions to the right of the decimal point in the Decimal Positions Field (column 44).
	ACTION	The compiler ignores your entry and assumes a numeric table with zero decimal position. Enter a digit (0 to 9) in this field.
3022W	MESSAGE	P, B, L, OR R SPECIFIED (COLUMN 43) AND BLANK DECIMAL POSITIONS - ASSUME NUMERIC WITH ZERO DECIMALS.
	CAUSE	You entered a P, B, L, or R in the Data Format Field (column 43) and left the Decimal Positions Field (column 44) blank.
	ACTION	The compiler ignores your entry and assumes a numeric table/array with zero decimal positions. Either enter a blank in column 43 or a digit (0-9) in column 44.
3023T	MESSAGE	DECIMAL POSITIONS (COLUMN 44) GREATER THAN NUMBER OF DIGITS - ASSUME EQUAL.
	CAUSE	You specified more digits to the right of the decimal in the Decimal Positions Field (column 44) than there are positions.
	ACTION	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 recompile.
3024W	MESSAGE	SEQUENCE (COLUMN 45) NOT BLANK, A OR D - A ASSUMED.
	CAUSE	You entered a letter or digit, other than an A or D, in the Table/Array Sequence Field (column 45).
	ACTION	The compiler ignores your entry and assumes you entered an A (for ascending sequence).

3025T	MESSAGE	FIELD NAME (COLUMNS 27-32) BLANK OR INVALID - ENTRY IGNORED.
	CAUSE	You entered an invalid table or array name or left the Table/Array Field (column 27-32) blank.
	ACTION	The compiler ignores your entry. Enter a valid field name and recompile.
3026T	MESSAGE	BINARY FIELD NOT LENGTH 5 OR 10.
	CAUSE	You specified a binary field which is neither 5 or 10 digits long.
	ACTION	Redefine the field so that it is either 5 or 10 digits long.
3027T	MESSAGE	TO FILE (COLUMNS 19-26) NOT AN OUTPUT FILE - TO FILE IGNORED.
	CAUSE	You specified the To File as something other than an Output File in its File Description Specification.
	ACTION	The compiler ignores To File Name entry. Enter a valid output file name or blanks and recompile.
3028W	MESSAGE	P/B/L/R FIELD (COLUMN 55) INVALID - BLANK ASSUMED.
	CAUSE	You entered a character other than P, B, L, or R in the Data Format Field (column 55).
	ACTION	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).
3029W	MESSAGE	FIELD SIZE OF 1 NOT ALLOWED WITH L OR R SPECIFIED IN COLUMN 55 - 2 ASSUMED.
	CAUSE	You specified a field size of 1 in the Entry Length Field (columns 52-54) with an L or R format in the Data Format Field (column 55).
	ACTION	The compiler ignores your entry and assumes you specified a field size of 2 characters. Enter a field size of at least two characters.
3030T	MESSAGE	FIELD SIZE (COLUMNS 52-54) BLANK OR INVALID - 1 ASSUMED.
	CAUSE	You entered an invalid field size or left the Entry Length Field (columns 52-54) blank.
	ACTION	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.
3031T	MESSAGE	ALPHA FIELD SIZE GREATER THAN 256 - ASSUME 256.
	CAUSE	You entered an alphanumeric field size in the Entry Length Field (columns 40-42 or columns 52-54) that is greater than 256 characters.
	ACTION	The compiler ignores your entry and assumes you entered 256 characters. Enter a field size of 256 or less.

3032T	MESSAGE	NUMERIC FIELD SIZE GREATER THAN 15 - ASSUME 15.
	CAUSE	You entered a numeric field size in the Entry Length Field (columns 40-42 or columns 52-54) greater than 15 digits.
	ACTION	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.
3033W	MESSAGE	DECIMAL POSITIONS (COLUMN 56) INVALID - ASSUME NUMERIC WITH ZERO DECIMAL POSITIONS.
	CAUSE	You specified an invalid number of digits to the right of the decimal point in the Decimal Positions Field (column 56).
	ACTION	The compiler ignores your entry and assumes numeric data with zero decimal positions. Enter 0-9 in this field.
3034W	MESSAGE	P, B, L, OR R SPECIFIED (COLUMN 55) AND BLANK DECIMAL POSITIONS - ASSUME NUMERIC WITH ZERO DECIMALS.
	CAUSE	You specified a P, B, L or R in the Data Format Field (column 55) and left the Decimal Positions Field (column 56) blank.
	ACTION	The compiler ignores your entry and assumes a numeric table/array with zero decimal positions. Either enter a blank in column 55 or 0-9 in column 56.
3035T	MESSAGE	DECIMAL POSITIONS (COLUMN 56) GREATER THAN NUMBER OF DIGITS - ASSUME EQUAL.
	CAUSE	You specified more positions to the right of the decimal in the Decimal Positions Field (column 56) than there are digits.
	ACTION	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.
3036W	MESSAGE	SEQUENCE (COLUMN 57) NOT BLANK, A, OR D - A ASSUMED.
	CAUSE	You entered a letter or digit, other than an A or D, in the Table/Array Sequence Field (column 57).
	ACTION	The compiler ignores your entry and assumes you entered an A (for ascending sequence).
3037T	MESSAGE	FIELD NAME (COLUMNS 46-51) BLANK OR INVALID - ENTRY IGNORED.
	CAUSE	You entered an invalid table or array name or left the Alternating Table/Array Name Field (column 46-51) blank.
	ACTION	Enter the correct name and recompile.

3038T	MESSAGE	TABLE/ARRAY NAME (COLUMNS 46-51) PREVIOUSLY DEFINED - RECORD IGNORED.
	CAUSE	You previously used this table or array name in the Alternating Table/Array Name Field (columns 46-51).
	ACTION	The compiler ignores the record entry; change the name and recompile.
3039W	MESSAGE	NO EXTENSION SPECIFICATION FOR FILE WITH E IN COLUMN 39 OF FILE SPECIFICATION.
	CAUSE	The program uses a table, array, chaining file or RAF but there is no File Extension Specification that describes it.
	ACTION	Insert the File Extension Specification and recompile the program.
3040W	MESSAGE	ENTRIES PER RECORD BLANK AND FILENAME SPECIFIED - ASSUME 1.
	CAUSE	You did not specify the number of entries per record, yet you specified the file name in the Table/Array Name Field (column 27-32).
	ACTION	The compiler ignores the entry and assumes you specified 1 entry per record. Enter valid numeric characters, right-adjusted with no embedded blanks.
3041I	MESSAGE	EXTENSION SPECIFICATION BLANK - IGNORE RECORD.
	CAUSE	You did not enter any information in this specification.
	ACTION	The compiler ignores the record; enter valid file names in the From File Name Field and the To File Name Field.
3042W	MESSAGE	ENTRIES PER RECORD GREATER THAN TABLE SIZE - ASSUME ENTRIES REVERSED.
	CAUSE	You specified a greater number of entries in one record of the table (Entries Per Record Field, columns 33-35) than the number of entries in the entire table (Entries Per Table/Array Field, columns 36-39).
	ACTION	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 recompile.
3043T	MESSAGE	FROM FILE RECORD LENGTH NOT LARGE ENOUGH TO HOLD ENTRIES PER RECORD.
	CAUSE	You specified a number of entries per record (columns 33-35) that exceeds the space allotted by the length of the records in the From File.
	ACTION	Enter the number of entries per record and recompile.
3044T	MESSAGE	TO FILE RECORD LENGTH NOT LARGE ENOUGH TO HOLD ENTRIES PER RECORD.
	CAUSE	You specified a number of entries per record (columns 33-35) that exceeds the space allotted by the length of the records in the To File.
	ACTION	Enter the number of entries per record and recompile.

4001W	MESSAGE	INVALID, MISSING, OR UNDEFINED FILENAME IN COLUMNS 7-14, SPEC IS DROPPED.
	CAUSE	You specified a file name in the File Name Field (columns 7-14) that was invalid or undefined or you omitted this file name completely.
	ACTION	Include the correct file name and recompile.
4002W	MESSAGE	FILENAME IN COLUMNS 7-14 DOES NOT REFER TO 'L' EXTENSION CODE.
	CAUSE	You specified a file name in the File Name Field (columns 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 column 39.
	ACTION	Define the correct file name and recompile.
4003W	MESSAGE	LINE NO. ENTRY IN COL 15-17 INVALID, LESS THAN 1. OR GREATER THAN 112.
	CAUSE	You specified an invalid line number or a number greater than 112 in the Line Number Field (columns 15-17).
	ACTION	The compiler ignores your entry and assumes line number 66.
4004W	MESSAGE	INVALID OR MISSING ENTRY IN COLUMNS 18-19, ASSUME FL.
	CAUSE	You entered characters other than OL or FL or a number between 1 and 12 in the Channel Number/OL Field (columns 18-19), or left this field blank.
	ACTION	The compiler ignores your entry and assumes you entered FL (for Form Length).
4005W	MESSAGE	LINE NO. ENTRY IN COL 20-22 INVALID, LESS THAN 1, OR GREATER THAN 112.
	CAUSE	You entered an invalid overflow line or specified a line number greater than 112 in the Channel Number/OL Field (columns 20-24).
	ACTION	The compiler assumes line number 66. Correct the entry and recompile.
4006W	MESSAGE	INVALID OR MISSING OL ENTRY IN COLUMN 23-24, ASSUME OL.
	CAUSE	You entered characters other than OL or FL or a number between 01 and 12 in the Line Number and OL/FL Field (columns 23-24) or left this field blank.
	ACTION	The compiler assumes OL. Correct the entry and recompile.

4007W	MESSAGE	OVERFLOW LINE EXCEEDS FORM LENGTH; ASSUME FORM LENGTH.
	CAUSE	You specified an overflow line in the Line Number Field (column 20-22) that exceeds the form length in the previous Line Number Field (columns 15-17).
	ACTION	The compiler assumes that the form length line and overflow line are equal.
4008W	MESSAGE	LINE NUMBER ENTRY IS INVALID, PREVIOUS SPEC. OR DEFAULT IS USED.
	CAUSE	You specified an invalid line number.
	ACTION	The compiler ignores your entry.
4009W	MESSAGE	CHANNEL NUMBER ENTRY IS INVALID, LINE NO. ENTRY IGNORED.
	CAUSE	You specified an invalid channel number.
	ACTION	The compiler ignores your entry.
4011W	MESSAGE	MULTIPLE LINE COUNTER SPECIFICATION FOR SAME FILE IS DROPPED.
	CAUSE	You included more than one Line Counter Specification for the same channel, form length or overflow line.
	ACTION	The compiler ignores the second Line Counter Specification.
4012W	MESSAGE	COLUMNS 25-74 SHOULD BE BLANK.
	CAUSE	You entered data in columns 25-74 when using the Line Number Option of the Line Counter Specifications.
	ACTION	The compiler ignores the entries in columns 25-74 and assumes you left these columns blank.
4013W	MESSAGE	NO LINE SPECIFICATION FOR FILE WITH L IN COLUMN 39 OF FILE SPECIFICATION.
	CAUSE	You entered L in column 39 of a File Description Specification that has no associated Line Counter Specification.
	ACTION	Correct the entry and recompile the program.
4015W	MESSAGE	NO ENTRIES IN L-SPEC RECORD.
	CAUSE	You entered a file name with no other entries in this specification.
	ACTION	The compiler ignores this specification.
4016W	MESSAGE	ENTRIES AFTER 1ST BLANKS ON L-SPEC IGNORED.
	CAUSE	When processing channel specifications, processing stops when the first blank line no./channel no. pair is found. The compiler then checks to see if the remainder of the specification is blank.
	ACTION	The compiler ignores non-blank trailing entries.

4017W	MESSAGE	MULTIPLE FL USAGE; ASSUME COLS 23-24 ARE OL.
	CAUSE	Columns 18-19 and 23-24 both contain FL.
	ACTION	The line number in columns 20-22 is used as the OL line.
4018W	MESSAGE	MULTIPLE OL USAGE; ASSUME COLS 18-19 ARE FL.
	CAUSE	Columns 18-19 and 23-24 both contain OL.
	ACTION	The line number in columns 15-17 is used as the FL line.
5001T	MESSAGE	INVALID OR UNDEFINED FILENAME - DISK FILE ASSUMED.
	CAUSE	You entered, in the File Name Field (columns 7-14), an invalid file name or a name that is not defined in the File Description Specifications.
	ACTION	Specify the correct file name or include a File Description Specification for it.
5002T	MESSAGE	FILE IS NOT AN INPUT, UPDATE, OR COMBINED FILE - INPUT ASSUMED.
	CAUSE	You entered a file name that does not identify an input, update, or combined file.
	ACTION	Correct the File Description Specification to identify an input, update, or combined file.
5003T	MESSAGE	AND/OR LINE CANNOT FOLLOW TRAILER OR LOOK-AHEAD.
	CAUSE	You entered an AND or OR line after specifying a trailer or look-ahead record in the Record Indicator/Look-Ahead/Trailer/Data Structure Field (columns 19-20).
	ACTION	Remove the AND or OR line.
5004T	MESSAGE	AND/OR LINE DOES NOT FOLLOW RECORD IDENTIFICATION LINE.
	CAUSE	You did not specify a record to which this AND/OR line applies.
	ACTION	Enter the first line of the record identification for this record.
5005W	MESSAGE	AND/OR LINE FOLLOWS LINE WITHOUT RECORD IDENTIFICATION CODES.
	CAUSE	You did not include record identification codes in columns 21-41 on the previous line.
	ACTION	Enter the record identification codes (C,Z,D) on the previous line.

5006W	MESSAGE	AND/OR LINE AND NUMBER/OPTION (COLUMNS 17-18) NOT BLANK.
	CAUSE	You entered AND or OR in columns 14-16 and on the same specification line you entered a value in the Number of Records Field (column 17) or Option Field (column 18).
	ACTION	Enter the Number of Records or Option Fields on a different specification line from the AND/OR line, then recompile.
5007W	MESSAGE	AND LINE AND RECORD INDICATOR (COLUMNS 19-20) NOT BLANK.
	CAUSE	You entered an AND line and a record indicator in the Record Indicator/Look-Ahead/Trailer/Data Structure Field (columns 19-20).
	ACTION	Enter a record indicator on a separate specification line from the AND line, then recompile.
5009T	MESSAGE	TRAILER HEADER DOES NOT CONTAIN ALPHA SEQUENCE OR NUMERIC WITH N.
	CAUSE	You specified a trailer record and the previous record identification specification contained a numeric sequence entry, but did not contain N in column 17.
	ACTION	On the previous record identification specification, enter N in column 17 or specify an alphabetic sequence entry.
5010W	MESSAGE	COLS. 7-14, 17, 18, 21-70 FOR LOOK-AHEAD OR 7-18, 21-70 FOR TR MUST BE BLANK.
	CAUSE	You entered data in the wrong fields in the description of a spread or look-ahead record or entered the file name or group sequence in a spread record description.
	ACTION	Remove the data from all columns that should be blank.
5011T	MESSAGE	PREVIOUS LOOK-AHEAD FOR FILE.
	CAUSE	You defined more than one look-ahead record for a file, but only one is allowed.
	ACTION	Remove one of the look-ahead records (consolidate them).
5012W	MESSAGE	LOOK AHEAD RECORD HAS NON-ALPHA SEQUENCE NUMBER (COLUMNS 15-16).
	CAUSE	You specified a numeric sequence for a look-ahead record.
	ACTION	Change the sequence characters (in columns 15-16) to alphabetic characters.
5013W	MESSAGE	ALPHA SEQUENCE (COLUMNS 15-16) FOLLOWS NUMERIC - ASSUME NUMERIC.
	CAUSE	You listed the numeric sequence entries before the alphabetic entries in the Group Sequence Field (columns 15-16)
	ACTION	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.

5014W	MESSAGE	NUMBER/OPTION FIELDS (COLUMNS 17-18) NOT BLANK FOR ALPHA SEQUENCE - ASSUME BLANK.
	CAUSE	You entered an option in the Option Field (column 17) or indicated the number of records in the Number of Records Field (column 18) and also specified an alphabetic sequence in the Group Sequence Field (column 15-16).
	ACTION	The compiler ignores your entry and assumes you left the Number of Records Field blank. Either change the alphabetic sequence characters (columns 25-26) to numeric characters or set columns 17-18 blank.
5015W	MESSAGE	INVALID SEQUENCE NUMBER (COLUMNS 15-16) - ASSUME ALPHA SEQUENCE.
	CAUSE	You entered an invalid sequence number in the Group Sequence Field (column 15-16), and the previous sequence characters were alphabetic.
	ACTION	The compiler ignores your sequence number and assumes you entered an alphabetic sequence. Correct the sequence number.
5016W	MESSAGE	INVALID SEQUENCE NUMBER (COLUMNS 15-16) - ASSUME NUMERIC SEQUENCE.
	CAUSE	You entered an invalid sequence number in the Group Sequence Field (columns 15-16) and the previous sequence characters were numeric.
	ACTION	The compiler ignores your sequence number and assumes you specified a numeric sequence. Correct the sequence number.
5017T	MESSAGE	SEQUENCE (COLUMNS 15-16) NOT ASCENDING - ASSUME ASCENDING SEQUENCE.
	CAUSE	Your numeric sequence character group was not higher than the previous sequence number.
	ACTION	The compiler ignores your sequence and assumes ascending sequence. Correct the numeric sequence character group to fall in proper ascending sequence.
5018W	MESSAGE	FIRST NUMERIC SEQUENCE NOT 1 - ASSUME 1.
	CAUSE	You didn't assign the first sequence as 01 in the Group Sequence Field (columns 15-16).
	ACTION	The compiler ignores your sequence number and assumes you entered 01. Correct the first numeric sequence character group.
5019T	MESSAGE	NUMBER FIELD (COLUMN 17) NOT 1 OR N WITH NUMERIC SEQUENCE -ASSUME N.
	CAUSE	You entered a character other than an N or 1, in the Number of Records Field (column 17) and specified numeric sequencing.
	ACTION	The compiler ignores your entry and assumes you entered an N. Correct the Number of Records entry.

5021T	MESSAGE	INVALID RECORD IDENTIFICATION INDICATOR.
	CAUSE	You used an invalid indicator in the Record Indicator Field (columns 19-20).
	ACTION	In columns 19-20, enter one of the indicators: 01-99, F0-F9, H0-H9, KA-KN, KP-KY, L1-L9, LR, MR, OA-OG, OV, 1P, U1-U8.
5023T	MESSAGE	INVALID RECORD IDENTIFICATION CODE POSITION FIELD (COLUMNS 21-24, 28-31, OR 35-38) - ASSUME 1.
	CAUSE	You specified invalid numbers or embedded blanks in the Record Identification Codes Field (columns 21-24, 28-31, or 35-38).
	ACTION	Correct the field to include valid numeric characters, right-adjusted.
5024T	MESSAGE	RECORD ID CODE POSITION (COLUMNS 21-24, 28-31, OR 35-38) NOT WITHIN RECORD LENGTH - ASSUME 1.
	CAUSE	You specified a record identification code in the Record Identification Codes Field (columns 21-24, 28-31, or 35-38) that doesn't fall within the record length defined.
	ACTION	The compiler ignores the entry and assumes you entered a 1 in the field. Correct the field so that it falls within the record length.
5025W	MESSAGE	NOT FIELD (COLUMNS 25, 32, OR 39) NOT BLANK OR N - ASSUME N.
	CAUSE	You specified a character other than N in columns 25, 32, or 39.
	ACTION	Enter an N or a blank in the field.
5026W	MESSAGE	C/Z/D FIELD (COLUMNS 23, 33, OR 40) NOT C, Z, OR D - ASSUME C.
	CAUSE	You entered a letter or digit other than a C, Z, or D, in column 26, 33, or 40.
	ACTION	The compiler ignores the entry and assumes you entered a C. Change the entry to C, Z, or D.
5027I	MESSAGE	NO FIELD DESCRIPTIONS FOR PREVIOUS RECORD.
	CAUSE	You followed one record description specification with another, without an intervening field description specification.
	ACTION	This entry is probably valid. However, the compiler issues the message just in case this is an AND/OR line or in case you forgot to include field descriptions for the previous record.
5028W	MESSAGE	COLUMNS 43-70 NON-BLANK FOR RECORD DESCRIPTION.
	CAUSE	You entered data in columns 43-70 in a record description specification.
	ACTION	The compiler ignores the data in columns 43-70. Remove the data from these columns.

5029T	MESSAGE	COLUMN 43 NOT P, B, L, R, 1-9, OR BLANK - ASSUME BLANK.
	CAUSE	You entered a character other than a P, B, L, R, or 1-9 in the Data Format Field (column 43).
	ACTION	The compiler ignores your entry and assumes you left the field blank. Enter P, B, L, R, 1-9, or blank in column 43.
5030T	MESSAGE	FROM FIELD LOCATION INVALID - ASSUME 1.
	CAUSE	You specified invalid numeric characters or embedded blanks in the From Field Position (column 44-47).
	ACTION	The compiler ignores your entry and assumes you entered a 1. Correct the From Field Position to contain valid numbers, right-adjusted.
5032T	MESSAGE	'FROM' GREATER THAN 'TO' FIELD LOCATION - ASSUME FIELD LENGTH OF 1.
	CAUSE	You specified that the From Field in the From Field Position Field (columns 44-47) is greater than the To Field in the To Field Position Field (columns 48-51).
	ACTION	The compiler ignores your entry and assumes the From Field length is 1. Correct the From or To Field Positions so that the To Field Position is greater than or equal to the From Field Position.
5033T	MESSAGE	FIELD LOCATION NOT WITHIN RECORD LENGTH.
	CAUSE	You specified a To Field Position that does not fall within the record length defined for the file.
	ACTION	Either correct the record length for the file or change the field location so that it falls within the record length.
5034T	MESSAGE	DECIMAL POSITIONS (COLUMN 52) NOT BLANK OR 0-9 - ASSUME BLANK.
	CAUSE	You entered a character, other than 0-9, in the Decimal Positions Field (column 52).
	ACTION	The compiler ignores your entry and assumes you left the field blank. Correct the field to reflect the number of decimal positions desired.
5035T	MESSAGE	FIELD NAME (COLUMNS 53-58) BLANK OR INVALID - SPECIFICATION DROPPED.
	CAUSE	You entered an invalid field name or left the Field Name Field (columns 53-58) blank.
	ACTION	The compiler ignores this specification. Correct the field name.
5036T	MESSAGE	INVALID INDEX FOR FIELD NAME (COLUMNS 53-58) ASSUME INDEX OF 1.
	CAUSE	You entered invalid numeric characters or embedded blanks for the array index in the Field Name Field (columns 53-58).
	ACTION	The compiler ignores the index and assumes you entered a 1. Enter a numeric character or field for the index.

5037T	MESSAGE	VARIABLE ARRAY INDEX (COLUMNS 53-58) IS NOT NUMERIC VARIABLE.
	CAUSE	You specified a value other than a numeric variable for the variable array index in the Field Name Field (columns 53-58).
	ACTION	Specify a numeric variable with zero decimal positions for the array index.
5038W	MESSAGE	VARIABLE ARRAY INDEX (COLUMNS 53-58) DOES NOT HAVE ZERO DECIMAL POSITIONS - ASSUME 0.
	CAUSE	You specified a variable array index without specifying zero decimal positions for it.
	ACTION	The compiler accepts your entry as if you specified zero decimal positions for the index. Redefine the index to have zero decimal positions.
5039T	MESSAGE	INDEX CANNOT BE AN ARRAY NAME.
	CAUSE	You used an array name as an array index.
	ACTION	Use either a numeric constant or a numeric variable with zero decimal positions as the array index.
5040T	MESSAGE	FIELD NAME (COLUMNS 53-58) IS A NON-ALTERABLE FIELD - RECORD IGNORED.
	CAUSE	The field named is defined previously as a look-ahead field or UDAY, UMONTH, UYEAR, or UDATE. These cannot be used in Input Specifications.
	ACTION	Specify a different field name.
5041T	MESSAGE	PAGE INVALIDLY REDEFINED - MUST BE NUMERIC WITH ZERO DECIMAL POSITIONS.
	CAUSE	You redefined PAGE (Field Name Field, columns 53-58) without entering zero in the Decimal Positions Field (column 52).
	ACTION	Enter a zero in the Decimal Positions Field (column 52) for the field PAGE and recompile.
5042T	MESSAGE	DECIMAL POSITIONS (COLUMN 52) DIFFERS FROM PREVIOUS SPECIFICATION - ASSUME PREVIOUS DEFINITION.
	CAUSE	You entered the same field name that was previously defined with different decimal positions in the field (column 52).
	ACTION	The compiler ignores your entry and assumes the decimal position is the same as the previous definition of this field name. Correct one or the other entry so that both are the same.

5043	MESSAGE	FIELD SIZE IS NOT A MULTIPLE OF THE NUMBER OF ELEMENTS - NEXT LOWEST MULTIPLE ASSUMED.
	CAUSE	You specified an array name, and the field size allows for several array elements plus a fraction of an array element.
	ACTION	Correct the field size to contain an integral number of array elements.
5044T	MESSAGE	ARRAY SIZE EXCEEDED - ASSUME TO END OF ARRAY.
	CAUSE	You specified an array name and the field size allows for more array elements than the array can hold.
	ACTION	Correct either the array definition or the number of array elements defined by the field length.
5045T	MESSAGE	FIELD SIZE NOT SAME AS PREVIOUSLY DEFINED - ASSUME PREVIOUS DEFINITION.
	CAUSE	You specified a field size that isn't the same as previously defined for this field.
	ACTION	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 either field so that they are the same size.
5046T	MESSAGE	FIELD PREVIOUSLY DEFINED AS ALPHA - ASSUME ALPHA.
	CAUSE	You defined this field as numeric, but it is previously defined as alphanumeric.
	ACTION	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.
5047T	MESSAGE	FIELD PREVIOUSLY DEFINED AS NUMERIC - ASSUME NUMERIC.
	CAUSE	You defined the field as alphanumeric, but it is previously defined as numeric.
	ACTION	The compiler assumes that this is a numeric field. Correct the type of either field so that both fields are the same type.
5048T	MESSAGE	ALPHA FIELD SIZE GREATER THAN 256 - ASSUME 256.
	CAUSE	You entered an alphanumeric field size greater than 256 characters; that is, the To Field Position minus the From Field Position plus 1 is greater than 256.
	ACTION	The compiler ignores your entry and assumes you specified 256 characters. Correct the From Field Position or To Field Position Fields to reflect a field length of 256 or less.

5049T	MESSAGE	DECIMAL FIELD LENGTH EXCEEDS 15 DIGITS - ASSUME 15.
	CAUSE	You entered a decimal field length that exceeds 15 digits.
	ACTION	The compiler ignores your entry and assumes you specified 15 digits. Correct the From Field Position or the To Field Position Fields to reflect a digit length of 15 or less.
5050T	MESSAGE	DECIMAL POSITIONS EXCEED NUMBER OF DIGITS - ASSUME EQUAL TO DIGITS.
	CAUSE	You specified, in the Decimal Positions Field (column 52), a value greater than the total number of digits in the field.
	ACTION	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.
5051T	MESSAGE	INVALID FIELD RECORD RELATION (COLUMNS 63-64) INDICATOR - BLANK ASSUMED
	CAUSE	You specified an invalid record-identifying indicator in the Field Record Relation Field (columns 63-64).
	ACTION	The compiler ignores the entry and assumes you left this field blank. Enter a valid indicator (01-99, H0-H9, L0-L9, LR, MR, OA-OG, OV, 1P, F0-F9, KA-KN, KP-KY, U1-U8) or leave this field blank.
5052T	MESSAGE	CONTROL LEVEL (COLUMNS 59-60) NOT BLANK OR L1-L9 - ASSUME BLANK.
	CAUSE	You specified invalid characters, other than the control-level indicators L1-L9, in the Control Level Field (columns 59-60).
	ACTION	The compiler ignores your entry and assumes you left this field blank. Enter L1-L9 in the Control Level Field or leave it blank.
5053T	MESSAGE	CONTROL LEVEL LENGTH DIFFERS FROM PREVIOUS DEFINITION - ASSUMES PREVIOUS DEFINITION.
	CAUSE	You specified a different control-level field length than the length previously defined for this field.
	ACTION	The compiler ignores your entry and assumes you specified the same control-level field length as you previously defined. Correct either length so that they are the same.
5054T	MESSAGE	CONTROL LEVEL (COLUMNS.59-60) OR MATCHING FIELD (COLUMNS.61-62) USED WITH CHAINED OR DEMAND FILE - ASSUME BLANK.
	CAUSE	You specified a control-level indicator in the Control Level Field (columns 59-60), or a matching-record indicator in the Matching/Chaining Fields (columns 61-62) for a chained or demand file.
	ACTION	The compiler ignores your entry and assumes you omitted the control-level indicator or matching-record indicator. Change the File Designation Field (column 16) of the File Description Specification or leave columns 59-62 blank in the Input Specification.

5055T	MESSAGE	CONTROL OR MATCHING FIELD (COLUMNS 59-62) USED FOR TRAILER OR LOOK-AHEAD RECORD - ASSUME BLANK.
	CAUSE	You assigned a control-level or matching-record indicator to a spread record trailer or a look-ahead record.
	ACTION	The compiler ignores your entry and assumes that you left this field blank. Change these columns to blank.
5056T	MESSAGE	INVALID MATCHING OR CHAINING ENTRY (COLS 61-62) - ASSUME BLANK.
	CAUSE	You assigned a code other than M1 through M9, C1 through C9, or blank to the Matching/Chaining Field (columns 61-62).
	ACTION	The compiler ignores your entry and assumes that you left this field blank. Change the entry to M1-M9, C1-C9, or blank.
5057T	MESSAGE	ARRAY USED FOR MATCHING FIELD - ASSUME NO MATCHING FIELD.
	CAUSE	You used an array rather than a data field to test for matching fields.
	ACTION	The compiler assumes that no matching field entry is used. Remove the matching field entry for this record. If you need to use this field as a matching field, redefine it with another name and assign M1-M9 to this new name.
5058T	MESSAGE	FIELD RECORD RELATION PREVIOUSLY USED WITH THIS MATCHING OR CONTROL LEVEL - ASSUME PREVIOUS.
	CAUSE	You used the same field record relation previously in this record with matching or control level fields.
	ACTION	Either remove the matching-record or control-level indicator, change the field record relation or move the specification records to a new place.
5059T	MESSAGE	MATCHING LEVEL ALREADY SPECIFIED FOR THIS FIELD RECORD RELATION.
	CAUSE	The same matching level (M1-M9) has already been used in this record specification with the same field record relation (may be a blank).
	ACTION	Change the matching level or the field record relation.
5060T	MESSAGE	MATCHING LEVEL LENGTH DIFFERS FROM PREVIOUS DEFINITION - ASSUME PREVIOUS.
	CAUSE	The matching field length specified differs from the length previously defined for this field.
	ACTION	The compiler assumes the previous length for this field. Change one field length or the other so that both are the same.

5061T	MESSAGE	COMPLETE SET OF MATCHING FIELDS NOT DEFINED FOR LAST GROUP.
	CAUSE	When you previously specified matching fields in this program, you specified more levels of matching fields than the current definition.
	ACTION	Either delete the extra matching levels specified earlier or add levels to the current definition.
5063T	MESSAGE	CHAINING (COLUMNS 61-62) SPECIFIED WITH LOOK-AHEAD FIELD - CHAINING IGNORED.
	CAUSE	You requested chaining with a look-ahead record.
	ACTION	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.
5064T	MESSAGE	L OR R (COLUMN 43) SPECIFIED WITH CHAINING (COLUMNS 61-62) - CHAINING IGNORED.
	CAUSE	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.
	ACTION	The compiler ignores your request for chaining. Remove the chaining specification or the L or R. The field may be redefined as a regular field (with From Position one larger for L or To Position one smaller for R) with no L or R for the field.
5065T	MESSAGE	ARRAY SPECIFIED AS CHAINING FIELD - IGNORE CHAINING.
	CAUSE	You specified an array rather than a data field as a chaining field.
	ACTION	The compiler ignores your request for chaining. Remove the chaining entry from this record. The field may be redefined as a regular field (same location but different name) with chaining.
5066T	MESSAGE	INCORRECT KEY TYPE FOR CHAINING FILE. SHOULD BE ALPHANUMERIC.
	CAUSE	A non-alphanumeric key was specified for a KSAM or TurboIMAGE file.
	ACTION	Use only alphanumeric keys for chaining with these file types (column 31 of the File Description Specification is A). Correct the program and recompile.
5067T	MESSAGE	CHAINING LEVEL (COLUMNS 61-62) ALREADY SPECIFIED FOR THIS FIELD RECORD RELATION.
	CAUSE	The same chaining level is already in this specification with the same field record relation.
	ACTION	Delete the chaining level (C1-C9) from this record.

5068T	MESSAGE	FIELD RECORD RELATION PREVIOUSLY USED IN THIS RECORD FOR CHAINING.
	CAUSE	All chaining fields for one field record relation were not together.
	ACTION	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.
5069T	MESSAGE	INVALID PLUS RESULT INDICATOR (COLUMNS 65-66) - ASSUME BLANK.
	CAUSE	You entered an invalid indicator or illegal characters in the Plus Subfield (columns 65-66).
	ACTION	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, F0-F9, KA-KN, KP-KY, U1-U8) in these columns.
5070T	MESSAGE	INVALID MINUS RESULT INDICATOR (COLUMNS 67-68) - ASSUME BLANK.
	CAUSE	You entered an improper indicator or illegal characters in the Minus Subfield (columns 67-68).
	ACTION	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, F0-F9, KA-KN, KP-KY, U1-U8) in these columns.
5071T	MESSAGE	INVALID BLANK/ZERO RESULT INDICATOR (COLUMNS 69-70) ASSUME BLANK.
	CAUSE	You entered an invalid indicator or illegal characters in the Zero or Blank Subfield (columns 69-70).
	ACTION	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, F0-F9, KA-KN, KP-KY, U1-U8) in these columns.
5072T	MESSAGE	FIELD USED PREVIOUSLY AS INDEX NOT DEFINED AS NUMERIC WITH ZERO DECIMAL POSITIONS.
	CAUSE	You used an alphanumeric field or a numeric field containing one or more decimal positions as an array element index.
	ACTION	Redefine the index field to be numeric with zero decimal positions.
5073T	MESSAGE	TABLE USED AS INPUT FIELD.
	CAUSE	You specified a table name in the Field Name Field (columns 53-58).
	ACTION	Delete the reference to the table from the Input Specification.
5074T	MESSAGE	MATCHING OR CONTROL LEVEL WITH BLANK FIELD RELATION MUST BE SPECIFIED FIRST.
	CAUSE	You specified matching or control-level fields with a field record relation indicator before the current one (which has a blank field record relation).
	ACTION	Move the blank field record relation specification so that it comes before those that specify an indicator.

5075T	MESSAGE	FIRST INPUT SPECIFICATION DOES NOT HAVE FILE NAME.
	CAUSE	You omitted the file name from the File Name Field (columns 7-14) of the first Input Specification or entered a field description with no preceding record description.
	ACTION	Include the file name on a record description specification.
5076T	MESSAGE	FILENAME ALREADY USED FOR INPUT SPECIFICATIONS.
	CAUSE	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.
	ACTION	Keep all Input Specifications for a file together, without intervening specifications for another file.
5077W	MESSAGE	NUMERIC SEQUENCE (COLUMNS 15-16) SPECIFIED FOR CHAINED FILE - ASSUME BLANK.
	CAUSE	You specified a numeric group sequence for this chained file, in the Group Sequence Field, where an alphanumeric sequence is required.
	ACTION	The compiler assumes that you specified an alphanumeric sequence. Change the sequence number to alphanumeric characters.
5078I	MESSAGE	BLANK RECORD - RECORD IGNORED.
	CAUSE	There is a blank line in the Input Specifications.
	ACTION	The compiler ignores this line.
5079T	MESSAGE	LOOK-AHEAD OR TRAILER RECORD NOT IN A PRIMARY OR SECONDARY FILE.
	CAUSE	You included a look-ahead record or spread record trailer in a file other than a primary or secondary file.
	ACTION	Either change the file designation or remove the look-ahead or trailer records.
5080T	MESSAGE	P, B, L, OR R (COLUMN 43) SPECIFIED FOR ALPHA FIELD - ASSUME BLANK.
	CAUSE	You specified in the Data Format Field, a numeric format for a field previously defined as alphanumeric.
	ACTION	The compiler assumes that you left the Data Format Field blank in this specification. Either change column 43 to blank or define this as a numeric field (by entering a digit in the Decimal Positions Field).
5081T	MESSAGE	BINARY FIELD LENGTH IS NOT 2 OR 4 - ASSUME UNPACKED NUMERIC FIELD.
	CAUSE	You defined an input field length other than 2 or 4 for a binary field; (2 is two bytes and 4 is four bytes).
	ACTION	Either correct the field length or change the type of numeric field.

5082T	MESSAGE	INDEX SPECIFIED FOR NON-ARRAY - IGNORE INDEX.
	CAUSE	You specified an index for a data field that is not an array.
	ACTION	The compiler ignores the index. Either include an array definition (File Extension Specification) for this field name or delete the index.
5083T	MESSAGE	MATCHING RECORD FILE NOT IN SAME SEQUENCE AS LAST.
	CAUSE	A file containing matching records has a different sequence than the last matching record file previously specified.
	ACTION	Change the Input Sequence Field of the File Description Specifications (column 18) for files with matching fields to all A or all D.
5084T	MESSAGE	INPUT, UPDATE, OR COMBINED FILE HAS NO INPUT SPECIFICATIONS.
	CAUSE	You named an input, update, or combined file but failed to specify the required record characteristics.
	ACTION	Include the Input Specifications for the input, update, or combined file or change the file type.
5085T	MESSAGE	DIGIT LENGTH OF FIELD IS ZERO - ASSUME 1.
	CAUSE	You have specified a numeric field containing no digits, but such fields must contain at least one digit.
	ACTION	The compiler assumes that you specified a 1-digit field. Redefine the field to be at least one digit long (fields with L or R in column 43 must be at least 2 bytes long to contain 1 digit).
5086T	MESSAGE	CHAINING CODE AND FILE NOT SPECIFIED ON EXTENSION SPECS - IGNORE CHAINING.
	CAUSE	You failed to specify a chaining field code in columns 11-18 of the File Extension Specifications, even though you have defined such a file in the Input Specifications.
	ACTION	Either correct the File Extension Specifications or delete the chaining entries from the Input Specifications.
5087W	MESSAGE	ARRAY USED FOR LEVEL FIELD - ASSUME NOT A LEVEL FIELD.
	CAUSE	You specified an array name in the Control Level Field (columns 59-60), but a field name is required.
	ACTION	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, redefine it with a new field name and include the chaining level.
5088T	MESSAGE	LOOK-AHEAD FIELD PREVIOUSLY DEFINED - RECORD IGNORED.
	CAUSE	You defined a look-ahead field that has already been defined.
	ACTION	The compiler ignores the line. Rename this field or delete the specification.

5089W	MESSAGE	CHAINING FIELD NUMERIC BUT ALPHA KEYS SPECIFIED ON FILE SPEC.
	CAUSE	You specified a numeric chaining field in conflict with alphanumeric searching keys for the TurboIMAGE file.
	ACTION	Either change the field type or the Record Address Type Field (column 31) of the File Description Specification.
5090W	MESSAGE	CHAINING FIELD LENGTH NOT SAME AS KEY FIELD LENGTH FROM FILE SPEC.
	CAUSE	You specified different lengths for the chaining and key fields; they must be the same.
	ACTION	Either redefine the chaining field length, the key length, or include a conversion routine.
5091W	MESSAGE	THE PRECEDING CONTROL LEVEL LENGTH DIFFERS FROM ITS PREVIOUS DEFINITION.
	CAUSE	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.
	ACTION	Change the program to specify the same total length for all definitions of the same control level, and recompile.
5092W	MESSAGE	PRECEDING CHAINING FIELD LENGTH NOT SAME AS KEY FIELD LENGTH FROM FILE SPEC.
	CAUSE	You specified a chaining field for which the total length differs from the Key Field Length Field (columns 29-30) in the File Description Specification.
	ACTION	Correct the length and recompile.
5093T	MESSAGE	DATA STRUCTURE (DS) SPECIFICATIONS CANNOT BE FOLLOWED BY NON-DATA STRUCTURE SPECS.
	CAUSE	You followed a data structure with non-data structure Input Specifications.
	ACTION	Rearrange the Input Specifications so that all data structures are at the end and recompile.
5094W	MESSAGE	RESERVED NAME FOR UDS IS 'LDA' - NAME SPECIFIED IN COLUMNS. 7-12 IS REPLACED BY 'LDA'.
	CAUSE	You specified a UDS (User Data Structure) in columns 18-20 and either; (1) specified a name other than LDA or blank in columns 7-12 or (2) specified LDA in columns 7-12, but had already defined LDA as a field on a prior Input Specification.
	ACTION	Correct the program and recompile.

5095T	MESSAGE	NESTED NUMERICS IN DS NOT SUPPORTED.
	CAUSE	A numeric field named as a data structure contains numeric subfields.
	ACTION	Do not use numeric subfields in this case. Correct the program and recompile.
5096W	MESSAGE	FIELD ALREADY DEFINED AS PART OF A DS-RECORD IGNORED.
	CAUSE	This field name has already been used in this or a previous data structure.
	ACTION	The compiler ignores this field.
5097T	MESSAGE	CANNOT OPEN FILE 'SIGWORK' FOR RSI PROCESSING.
	CAUSE	Possible system problem. This is an internal file used by the compiler to pass information to SIGEDITOR.
	ACTION	Contact your HP Support Engineer.
5098T	MESSAGE	CANNOT WRITE TO FILE 'SIGWORK'.
	CAUSE	Possible system problem.
	ACTION	Contact your HP Support Engineer.
5099T	MESSAGE	WORKSTNC FILE MUST USE RII 01-10 ONLY.
	CAUSE	Invalid record-identifying indicator.
	ACTION	Use indicators 01 through 10 only.
5100T	MESSAGE	CANNOT CLOSE FILE 'SIGWORK'.
	CAUSE	Possible system problem.
	ACTION	Contact your HP Support Engineer.
5101T	MESSAGE	CANNOT CREATE 'SIGEDIT' PROCESS.
	CAUSE	MPE cannot find the program SIGEDIT in PUB.SYS.
	ACTION	Be sure the program SIGEDIT is installed in PUB.SYS. If this error persists, contact your HP Support Engineer.
5102T	MESSAGE	CANNOT ACTIVATE 'SIGEDIT' PROCESS.
	CAUSE	Process may already be activated, or PH capability may be missing.
	ACTION	Be sure the group containing the RPG compiler has process-handling (PH) capability.
5103T	MESSAGE	COL. 43 NOT BLANK FOR DS FIELD. ASSUME BLANK.
	CAUSE	Input Specification column 43 was not blank for a DS field.
	ACTION	To eliminate this warning, change column 43 to a blank. The compiler assumes a blank by default.
6001T	MESSAGE	INVALID CONTROL LEVEL IN COL. 7-8, ASSUME LATEST LEVEL.
	CAUSE	You entered an indicator other than L0-L9 or LR in the Control Level Field (columns 7-8).
	ACTION	The compiler assumes that you intended the latest level indicator. Correct the control level and recompile.

6002W	MESSAGE	AN/OR LINE OUT OF ORDER OR INVALID, IGNORE AN/OR LINE.
	CAUSE	You entered an AN 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 Indicators Field.
	ACTION	The compiler ignores this line. Put AN/OR line in proper sequence.
6003T	MESSAGE	EXCEEDED 24 INDICATORS OR 7 'OR' LINES.
	CAUSE	You entered more than seven OR lines or more than 24 indicator lines in one grouping, which is not permitted.
	ACTION	The compiler ignores the entire grouping. Change the program so that there are no more than seven OR lines or 24 indicators.
6004W	MESSAGE	'AN' OR 'OR' NOT SPECIFIED IN COLUMNS. 7-8, IGNORE GROUP.
	CAUSE	The statement before this AN/OR record had conditioning indicators, but no operation was specified.
	ACTION	The compiler ignores the entire group. Put the operation in the previous statement or include AN/OR in this specification.
6005T	MESSAGE	OPERATION NOT RECOGNIZED.
	CAUSE	You entered an invalid operation in the Operation Field (columns 28-32).
	ACTION	The compiler ignores this operation. Correct it and recompile.
6006W	MESSAGE	'NOT' ENTRY IN COL. 9, 12, OR 15 NOT N OR BLANK, ASSUME N.
	CAUSE	You entered a character other than N or blank in the Not Subfield (column 9, 12, or 15).
	ACTION	The compiler assumes that you entered N.
6007W	MESSAGE	INVALID OPERATION INDICATOR, IGNORE INDICATOR GROUP.
	CAUSE	You entered an invalid operation indicator in the Indicators Field (columns 9-17).
	ACTION	The compiler ignores the indicators and performs the operation in any event. Correct the indicator in the Indicators Field (columns 9-17).
6008W	MESSAGE	INDICATOR MISSING FOR 'NOT' IN COL. 9, 12, OR 15, IGNORE NOT.
	CAUSE	You entered an N (for Not) in columns 9, 12, or 15, but did not enter an indicator in columns 10-11, 13-14, or 16-17.
	ACTION	The compiler ignores the N entry.

6009W	MESSAGE	INVALID RESULTING INDICATOR, IGNORE INDICATOR GROUP.
	CAUSE	You entered an invalid indicator in the Resulting Indicators Field (columns 54-59).
	ACTION	The compiler ignores this indicator and all others in the group to which it belongs. Correct the indicator and recompile.
6010W	MESSAGE	INVALID FACTOR 1 ENTRY IN COL. 18-27.
	CAUSE	You entered an invalid name, label, or literal in the Factor 1 Field (columns 18-27).
	ACTION	The compiler ignores this operation. Correct Factor 1 to contain a valid field name or a literal.
6011T	MESSAGE	INVALID FACTOR 2 ENTRY IN COL. 33-42.
	CAUSE	You entered an invalid name, label, or literal in the Factor 2 Field (columns 33-42).
	ACTION	The compiler ignores this operation. Correct Factor 2 to contain a valid field name or a literal.
6012W	MESSAGE	INVALID RESULT FIELD ENTRY IN COLUMNS. 43-48.
	CAUSE	You entered an invalid name in the Result Field (columns 43-48).
	ACTION	The compiler ignores the operation associated with this entry. Correct the Result Field so that is contains a valid field name.
6013W	MESSAGE	INVALID INDEX IN FACTOR 1 (COL. 18-27).
	CAUSE	You specified an indexed array in the Factor 1 Field, but did not use a valid index.
	ACTION	The compiler assumes an index value of 1. Correct the index so that it is a valid field (that is a numeric field with zero decimal positions) or a numeric unsigned literal with no decimal positions.
6014W	MESSAGE	INVALID INDEX IN FACTOR 2 (COL. 33-42).
	CAUSE	You specified an indexed array in the Factor 2 Field, but did not use a valid index.
	ACTION	The compiler assumes an index value of 1. Correct the index so that it is a valid field (that is a numeric field with zero decimal positions) or a numeric unsigned literal with no decimal positions.
6015W	MESSAGE	INVALID INDEX IN RESULT FIELD (COL. 43-48).
	CAUSE	You specified an indexed array in the Result Field, but did not use a valid index.
	ACTION	The compiler assumes an index value of 1. Correct the index so that it is a valid field name (that is a numeric field with zero decimal positions) or a numeric unsigned literal with no decimal positions.

6016T	MESSAGE	INVALID FILE NAME ENTRY FOR FACTOR 2 (COL. 33-42).
	CAUSE	You entered an improper file name in the Factor 2 Field.
	ACTION	The compiler ignores the operation. Correct the file name so that it appears as defined in the File Description Specifications.
6017W	MESSAGE	INVALID DIGIT IN RESULT FIELD LENGTH ENTRY (COL. 49-51).
	CAUSE	You entered one or more characters other than 0-9 in the Field Length Field. Embedded blanks are illegal.
	ACTION	The compiler ignores the field length. Enter a valid field length, right-justified, without embedded blanks.
6018W	MESSAGE	INVALID DIGIT IN DECIMAL POSITIONS ENTRY (COL. 52).
	CAUSE	You entered a value other than 0 through 9 in the Decimal Positions Field.
	ACTION	The compiler assumes zero decimal positions. Enter a digit 0-9 in column 52.
6019W	MESSAGE	RESULT FIELD LENGTH MISSING BUT DEC POS SPECIFIED. IGNORE DEC POS.
	CAUSE	You specified a number of decimal positions for the Result Field, but did not specify the field length.
	ACTION	The compiler assumes there are no decimal positions. Enter a valid field length or leave the Decimal Positions Field blank.
6020W	MESSAGE	HALF ADJUST ENTRY IN COL. 53 NOT H OR BLANK, ASSUME H.
	CAUSE	You entered a character other than H or a blank in the Half Adjust Field, which is illegal.
	ACTION	The compiler assumes that you entered H, specifying Half Adjust.
6021W	MESSAGE	FIELD ATTRIBUTE NOT SAME AS PREV DEFN, USE PREV DEFN.
	CAUSE	You entered a different field length or number of decimal positions than was previously assigned to the field.
	ACTION	The compiler uses the previous definition for the field. Correct the field attributes of the incorrect statement.
6022W	MESSAGE	RESULT FIELD HAS INDEX BUT NOT ARRAY NAME, IGNORE INDEX.
	CAUSE	You entered an array index in the Result Field but did not specify an array name.
	ACTION	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.

6023T	MESSAGE	FILE NAME IN FACTOR 2 NOT DEFINED BY FILE DESC SPECS.
	CAUSE	You entered a file name in the Factor 2 Field (columns 33-42) that is not described in the File Description Specifications.
	ACTION	The compiler ignores the operation. Either define this file name in a File Description Specification or use another file name.
6024W	MESSAGE	SET OPERATOR USAGE NOT CONSISTENT WITH WORKSTATION OPERATION. OPERATOR IGNORED.
	CAUSE	Set operator found in a program using a workstation file.
	ACTION	Compiler ignores the operation.
6025W	MESSAGE	FACTOR 1 IS MISSING (COL. 18-27).
	CAUSE	You omitted Factor 1 from an operation that requires it.
	ACTION	The compiler ignores the operation. Enter Factor 1 and recompile.
6026W	MESSAGE	FACTOR 2 IS MISSING (COL. 33-42).
	CAUSE	You omitted Factor 2 from an operation that requires it.
	ACTION	The compiler ignores the operation. Enter Factor 2 and recompile.
6027W	MESSAGE	RESULT FIELD IS MISSING (COL. 43-48).
	CAUSE	You omitted the Result Field from an operation that requires this field.
	ACTION	The compiler ignores the operation. Enter information in the Result Field and recompile.
6028W	MESSAGE	A RESULTING INDICATOR MUST BE SPECIFIED FOR THIS OPERATION.
	CAUSE	You failed to specify a resulting indicator in columns 54-59 for an operation that requires this indicator.
	ACTION	The compiler ignores the operation. Specify a resulting indicator and recompile.
6030W	MESSAGE	MISSING ENDSR STATEMENT, ASSUME ENDSR.
	CAUSE	You omitted the ENDSR operation from an internal subroutine.
	ACTION	The compiler assumes that an ENDSR statement follows the last statement with SR in the Control Level Field.
6031W	MESSAGE	OPERATION INDICATORS MISSING ON AN/OR LINE.
	CAUSE	You entered AN or OR in columns 7-8, but did not enter indicators in the Indicators Field (columns 9-17).
	ACTION	The compiler uses the indicators specified in the preceding lines.

6032W	MESSAGE	INDICATORS IN COL. 9-17 NOT ALLOWED FOR THIS OPERATION. ASSUME BLANK.
	CAUSE	You entered one or more indicators in columns 9-17, but this operation cannot be conditioned with indicators.
	ACTION	The compiler assumes that you left columns 9-17 blank and performs the operation under all conditions.
6033W	MESSAGE	INDICATORS IN COL. 54-59 NOT ALLOWED FOR THIS OPERATION. ASSUME BLANK.
	CAUSE	You entered one or more indicators in the Resulting Indicators Field, but they are not permitted for this operation.
	ACTION	The compiler assumes that you left the Resulting Indicators Field blank.
6034W	MESSAGE	FACTOR 1 SHOULD BE BLANK FOR THIS OPERATION ASSUME BLANK.
	CAUSE	You entered the Factor 1 Field but it is prohibited for this operation.
	ACTION	The compiler assumes that you left the Factor 1 Field blank.
6035W	MESSAGE	FACTOR 2 SHOULD BE BLANK FOR THIS OPERATION ASSUME BLANK.
	CAUSE	You entered the Factor 2 Field but it is prohibited for this operation.
	ACTION	The compiler assumes that you left the Factor 2 Field blank.
6036W	MESSAGE	RESULT FIELD SHOULD BE BLANK FOR THIS OPERATION ASSUME BLANK.
	CAUSE	You entered a field name in the Result Field, but it should remain blank for this operation.
	ACTION	The compiler assumes that you left the Result Field blank.
6037W	MESSAGE	FIELD USED AS RESULT IS NON ALTERABLE. IGNORE SPEC.
	CAUSE	You specified a look-ahead field, UDAY, UMONTH, UYEAR, or UDATE in the Result Field.
	ACTION	The compiler ignores this operation. Specify a different field name and recompile.
6038W	MESSAGE	PAGE FIELD USED AS RESULT FIELD IS ALREADY REDEFINED. USE PREV DEFN.
	CAUSE	You entered PAGE in the Result Field and redefined it again by giving it a new length.
	ACTION	The compiler uses the previous definition for PAGE.
6039W	MESSAGE	NAME ALREADY HAS ANOTHER USE.
	CAUSE	The same name is used for both a subroutine name and a tag.
	ACTION	Correct program and recompile.

6040T	MESSAGE	GOTO OPERATION MAY NOT BRANCH INTO OR OUT OF SUBROUTINE.
	CAUSE	You specified a GOTO operation in a subroutine that transfers to an operation outside of it or vice versa.
	ACTION	The compiler ignores the operation. Correct and recompile.
6041W	MESSAGE	DEC POS ENTRY GREATER THAN FIELD LENGTH. ASSUME ZERO POS.
	CAUSE	The number of decimal positions in the field exceeds the length of the field.
	ACTION	The compiler assumes that the field contains no decimal positions. Correct either the number of decimal positions or the entry length and recompile.
6042T	MESSAGE	NAME REFERENCED SHOULD BE TAG NAME.
	CAUSE	You entered a GOTO operation that does not reference a TAG label. (Tags and subroutines cannot have the same name.)
	ACTION	The compiler ignores the operation. Either rename the tag or rename the subroutine.
6043T	MESSAGE	A SUBROUTINE MAY NOT CALL ITSELF.
	CAUSE	You included, within an internal subroutine, an operation that attempts to invoke that subroutine; this type of recursion is illegal.
	ACTION	The compiler ignores the operation. Correct and recompile.
6044T	MESSAGE	BEGSR STATEMENT MISSING. ASSUME BEGSR.
	CAUSE	You did not begin an internal subroutine with the BEGSR operation.
	ACTION	The compiler assumes that the first operation in the subroutine is BEGSR. Include a BEGSR operation and a recompile.
6045T	MESSAGE	BEGSR OUT OF SEQUENCE. IGNORE BEGSR LINE.
	CAUSE	You included a BEGSR operation within the body of an internal subroutine, after the first operation.
	ACTION	The compiler ignores this BEGSR request. Place BEGSR in correct order and recompile.
6046I	MESSAGE	DEBUG OPTION NOT SPECIFIED IN COL.15 OF HEADER SPEC.
	CAUSE	You included one or more DEBUG operations but did not specify the Debug option in the Debug Field (column 15) of the Header Specification.
	ACTION	The compiler does not execute the DEBUG request.
6047W	MESSAGE	FIELD LENGTH AND DEC POS SHOULD BE BLANK. ASSUME BLANK.
	CAUSE	For a previously defined table or array in the Result Field, you entered values in the Field Length or Decimal Positions Field.
	ACTION	The compiler assumes that the Field Length and Decimal Positions Fields are blank.

6048W	MESSAGE	DEC POS ENTRY (COL 52) SHOULD BE BLANK. ASSUME BLANK.
	CAUSE	You specified a decimal position that should not be specified.
	ACTION	The compiler ignores this specification.
6049W	MESSAGE	HALF ADJUST ENTRY (COL 53) SHOULD BE BLANK. ASSUME BLANK.
	CAUSE	You specified a half adjust that should not be specified.
	ACTION	The compiler ignores this specification.
6050W	MESSAGE	BEGSR STATEMENT SHOULD HAVE SR OR BLANK IN COL. 7-8; ASSUME SR.
	CAUSE	You specified a BEGSR operation that did not include SR or a blank in columns 7-8.
	ACTION	The compiler assumes that you entered SR in these columns.
6051W	MESSAGE	FIELD LENGTH OVER 256 FOR ALPHANUMERIC OR OVER 15 FOR NUMERIC.
	CAUSE	You specified a field length that exceeds the maximum length allowed for the Result Field (columns 49-51).
	ACTION	The specification is ignored. Redefine the field length and recompile.
6052W	MESSAGE	OPERATOR NOT IMPLEMENTED ON THIS VERSION OF RPG.
	CAUSE	You entered an operator (in columns 28-32) that cannot be executed with this version of the compiler.
	ACTION	The compiler ignores this specification.
6053W	MESSAGE	PRECEDING OPERATOR WAS NOT A DIV.
	CAUSE	You specified an MVR (move remainder) operation that is not immediately preceded by a DIV (divide) operation.
	ACTION	The compiler ignores this MVR operation. Enter MVR immediately following the DIV operation and recompile.
6054T	MESSAGE	DUPLICATE TAG NAME.
	CAUSE	You assigned the same label to two or more TAG operations.
	ACTION	The compiler ignores the second TAG operation. Change one of the TAG labels and recompile.
6055T	MESSAGE	DUPLICATE SUBROUTINE NAME.
	CAUSE	You assigned the same name to two or more subroutines.
	ACTION	Change one of the subroutine names and recompile.

6056T	MESSAGE	SYMBOL NOT DEFINED.
	CAUSE	You referenced the name of a data area or label of an operation that is not defined in the program.
	ACTION	Define the symbol by naming it in the Result Field with a defined length, by specifying it as a field name on Input Specifications, or by defining it as an array. Recompile the program.
6057W	MESSAGE	FIELD 1 SHOULD BE ALPHANUMERIC FOR THIS OPERATION.
	CAUSE	You entered a numeric field for Factor 1 (columns 18-27), but it should be alphanumeric.
	ACTION	The compiler uses the field as specified. Change the Factor 1 type and recompile.
6058W	MESSAGE	FIELD 2 SHOULD BE ALPHANUMERIC FOR THIS OPERATION.
	CAUSE	You entered a numeric field for Factor 2 (columns 33-42), but it should be alphanumeric.
	ACTION	The compiler uses the field as specified. Change the Factor 2 type and recompile.
6059W	MESSAGE	RESULT FIELD SHOULD BE ALPHANUMERIC FOR THIS OPERATION.
	CAUSE	You entered a numeric field in the Result Field (columns 43-48), but it should be alphanumeric.
	ACTION	The compiler uses the field as specified. Change the Result Field type and recompile.
6060W	MESSAGE	FIELD 2 MUST BE THE SAME TYPE AS FIELD 1.
	CAUSE	You specified a different data format for Factor 1 than for Factor 2, which is illegal for this operation.
	ACTION	The compiler uses the field as specified. Change either the Factor 1 or Factor 2 type and recompile.
6061W	MESSAGE	FIELD 1 SHOULD BE NUMERIC FOR THIS OPERATION.
	CAUSE	You entered an alphanumeric field for Factor 1 (columns 18-27), but a numeric field is required.
	ACTION	The compiler uses the field as specified. Change the Factor 1 type and recompile.
6062W	MESSAGE	FIELD 2 SHOULD BE NUMERIC FOR THIS OPERATION.
	CAUSE	You entered an alphanumeric field for Factor 2 (columns 33-42), but a numeric field is required.
	ACTION	The compiler uses the field as specified. Change the Factor 2 type and recompile.

6063W	MESSAGE	RESULT FIELD SHOULD BE NUMERIC FOR THIS OPERATION.
	CAUSE	You entered an alphanumeric field for the Result Field (columns 43-48), but a numeric field is required.
	ACTION	The compiler uses the field as specified. Change the Result Field type and recompile.
6064W	MESSAGE	RESULTING INDICATOR REQUIRED IN COL 54-55; ASSUME H0.
	CAUSE	You did not enter the required indicator in the Resulting Indicators Field (columns 54-55).
	ACTION	The compiler assumes that you entered the H0 indicator.
6065W	MESSAGE	RESULTING INDICATOR REQUIRED IN COL 58-59; ASSUME H0.
	CAUSE	You did not enter the required indicator in the Resulting Indicators Field (columns 58-59).
	ACTION	The compiler assumes that you entered the H0 indicator.
6066W	MESSAGE	RESULTING INDICATOR REQUIRED IN COL 56-57; ASSUME H0.
	CAUSE	You did not enter the required indicator in the Resulting Indicators Field (columns 56-57).
	ACTION	The compiler assumes that you entered the H0 indicator.
6067W	MESSAGE	ILLEGAL I/O MODE IN IMAGE FILE.
	CAUSE	You specified an operation that is incompatible with the input/output mode.
	ACTION	The compiler ignores the operation. Correct and recompile.
6068W	MESSAGE	INCOMPATIBLE FIELD 1 LENGTH.
	CAUSE	You specified a field length that is different from the Key Length specified in the File Description Specification. (If packed decimal keys are used, the key field length is the number of digits divided by 2 plus 1.)
	ACTION	The length specified in the File Description Specification is used. Change either the key length in the File Description Specification or the Factor 1 Field length.
6069T	MESSAGE	FILE SPECIFIED IN FACTOR 2 NOT AN INDEXED DEMAND FILE.
	CAUSE	You requested a SETLL operation for a file that is not a demand file.
	ACTION	Change the file type to a demand file.
6070W	MESSAGE	FILE BEING READ IS NOT A DEMAND FILE.
	CAUSE	You used the READ command for a file that is not a demand file.
	ACTION	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.

6071T	MESSAGE	CURRENT FIELD USE INCONSISTENT WITH PREVIOUS USE.
	CAUSE	You either used an alphanumeric field as a numeric field or vice-versa.
	ACTION	Change specifications to make all references consistent.
6072W	MESSAGE	FILE SPECIFIED IN FACTOR 2 NOT A CHAINING FILE.
	CAUSE	A CHAIN operation was attempted to a file that is not a chaining file.
	ACTION	Correct the program and recompile.
6073T	MESSAGE	CAN'T FORCE READ A FILE DESIGNATED BY BLANK IN COL 16 OF FILE SPEC.
	CAUSE	You specified a FORCE operation for a file which contained a blank in column 16 of the File Description Specification.
	ACTION	Correct the program and recompile.
6074T	MESSAGE	PARM OPERATIONS MUST IMMEDIATELY FOLLOW AN EXIT OPERATION.
	CAUSE	You specified a PARM operation which does not follow an EXIT or another PARM operation.
	ACTION	Correct the program and recompile.
6081T	MESSAGE	LOW OR EQUAL RESULT INDICATOR MUST BE SPECIFIED FOR THIS OPERATION.
	CAUSE	You must specify a resulting indicator in the Low Subfield (columns 56-57) or the Equal Subfield (columns 58-59).
	ACTION	Correct the program and recompile.
6082T	MESSAGE	IMAGE FILE LOCKING MODE IS NOT L, B, S, 1, 9, OR R TO ENABLE LOCKING.
	CAUSE	You specified an incorrect locking mode for using the LOCK/UNLCK operations. The locking mode is specified in the Open Mode Field (column 66) of the Database Name (IMAGE) Line of the File Description Specification.
	ACTION	Correct the program and recompile.
6083T	MESSAGE	FILE NOT DESCRIBED WITH NOLOCK OR LOCK CONTINUATION RECORD TO ENABLE LOCKING.
	CAUSE	You are using LOCK/UNLCK operations, but did not enable locking using NOLOCK/LOCK in the File Description Specification.
	ACTION	Correct the program and recompile.
6084T	MESSAGE	DBNAME IN RESULT FIELD MUST BE SAME NAME SPECIFIED IN IMAGE SPEC FOR FILE IN FACTOR 2.
	CAUSE	The database name specified in the Result Field does not match the name specified in the Database Name (IMAGE) Line for the Factor 2 file.
	ACTION	Correct the program and recompile.

6085T	MESSAGE	LENGTH OF A VARIABLE USED TO INDEX AN ARRAY MUST BE 1 TO 9 DIGITS.
	CAUSE	You specified an incorrect entry for the length of a variable.
	ACTION	Correct the program and recompile.
6087I	MESSAGE	NO INDICATOR IN COL 58-59. H0 WILL BE SET ON FOR ERRORS NOT FLAGGED BY INDICATORS IN COL. 54-57.
	CAUSE	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.
	ACTION	Specify and use the equal indicator for this operation.
6088T	MESSAGE	SORTA OPERATOR ONLY ALLOWED ON NON-ALTERNATING ARRAY.
	CAUSE	The SORTA operation is not allowed for the array.
	ACTION	Correct the program and recompile.
6089T	MESSAGE	GENERIC KEY ACCESS ONLY ALLOWED WITH KSAM FILES.
	CAUSE	You specified *EQ, *GT, or *GE in the Result Field for a CHAIN or SETLL operation on a non-KSAM file.
	ACTION	Correct the program and recompile.
6090T	MESSAGE	GENERIC KEY ACCESS ONLY ALLOWED WITH ALPHANUMERIC OR PACKED KEYS.
	CAUSE	You specified *EQ, *GT, or *GE in the Result Field for a CHAIN or SETLL operation on a file whose Record Address Type (column 31) does not contain A or P.
	ACTION	Correct the program and recompile.
6091T	MESSAGE	PACKED GENERIC KEY MUST BE EXACT SIZE DEFINED IN FILE DESC. SPEC.
	CAUSE	You specified *EQ, *GT, or *GE in the Result Field and the generic key size differs from the size defined in the Key Field Length Field (columns 29-30) of the File Description Specification.
	ACTION	Correct the program and recompile.
6096T	MESSAGE	INVALID STRUCTURED OPERATION - MISSING OR EXTRANEOUS 'END', OR TOO MANY NESTED LEVELS
	CAUSE	You specified a structured programming operation IF, ELSE, DOU, or DOW, and either did not specify a matching END operation or specified too many END operations, or nested structured operators to more than 100 levels in depth.
	ACTION	Ensure that you have matched up structured operators and END statements. Correct the program and recompile.
6097W	MESSAGE	INVALID BIT NUMBER IN FACTOR 2 OF 'BITON', 'BITOF', OR 'TESTB'.
	CAUSE	A character other than 0-7 was used in the Factor 2 Field of these operations.
	ACTION	Enter only 0-7 to specify bit positions in the field.

6098W	MESSAGE	INVALID RESULT INDICATORS FOR 'CLOSE', 'READ', OR 'MSG' OPERATOR.
	CAUSE	One or more indicators were used in invalid locations in the Resulting Indicators Field (columns 54-59).
	ACTION	Delete the incorrect indicator(s).
6099T	MESSAGE	EXTRANEOUS OR MISPLACED ELSE STATEMENT.
	CAUSE	There are too many ELSE statements for the number of IF statements in your program. Or the ELSE statement might be after the END statement. This can also occur if the IF statement is in the detail calculations, and the ELSE statement is in total time calculations.
	ACTION	Make sure the ELSE statements are located correctly in the program; delete unnecessary ELSE statements.
6100W	MESSAGE	OPERATOR NOT SUPPORTED ON RPG/iX.
	CAUSE	The RPG/iX compiler detected the RPGCV, ERPGC, or EXTCV operation in the Calculation Specifications of your program. These operations are not supported by the RPG/iX compiler.
	ACTION	Your program must be rewritten to avoid the use of these operations.
6101T	MESSAGE	RESULT FIELD IS REQUIRED (COL 43-48).
	CAUSE	A Calculation Specification using the PARM operation was encountered (used to pass data to external subroutines), but the data to be passed was not specified in the Result Field.
	ACTION	Check statements using the PARM operation for missing Result Field entries.
6102W	MESSAGE	XFOOT OPERATION USES WHOLE ARRAY. INDEX IS IGNORED.
	CAUSE	An array name with an index was specified for Factor 2 in an XFOOT operation. The compiler expects an array name without an index.
	ACTION	To avoid this warning, remove the index from the array in this operation.
6103T	MESSAGE	IPARM OPERATIONS MUST IMMEDIATELY FOLLOW AN INTR OPERATION.
	CAUSE	An operation other than IPARM was found between an INTR operation and its associated IPARM operations.
	ACTION	Remove the improperly located operation.
6104T	MESSAGE	INVALID INTRINSIC NAME IN COL. 33 THROUGH 48.
	CAUSE	The compiler detected an invalid intrinsic name referenced in a Calculation Specification using the INTR operation.
	ACTION	Enter a valid intrinsic name in columns 33-48. Refer to the *MPE/iX Intrinsics Reference Manual* if necessary.
6105T	MESSAGE	COULD NOT OPEN THE INTRINSIC MECHANISM.
	CAUSE	The compiler could not open the SYSINTR.PUB.SYS file.
	ACTION	Make sure this file is on your system, and that its READ access is ANY.

6106T	MESSAGE	COULD NOT FIND THIS INTRINSIC IN SYSINTR.PUB.SYS.
	CAUSE	The intrinsic named in an INTR operation of a Calculation Specification was not found the intrinsic mechanism file SYSINTR.PUB.SYS.
	ACTION	Check the spelling of the intrinsic name used in the INTR operation. If this is a user-created intrinsic, verify that it was added to the SYSINTR.PUB.SYS file.
6107T	MESSAGE	TYPE MISMATCH IN IPARM OR FACTOR 1 OF INTR.
	CAUSE	The intrinsic expects a specific data type (alphanumeric or numeric) for this parameter, but a field of the wrong type was specified.
	ACTION	Check the types of the fields you are using to make sure they are correct for the intrinsic.
6108T	MESSAGE	COULD NOT EXTRACT PARM INFO FOR THIS PARAMETER.
	CAUSE	Information for the parameter named in an IPARM operation or in Factor 1 of an INTR operation was not found in the intrinsic mechanism file SYSINTR.PUB.SYS.
	ACTION	Check your program to make sure your parameters match those specified by the intrinsic.
6109T	MESSAGE	MISSING REQUIRED PARAMETER.
	CAUSE	A parameter that is required by the intrinsic named in an INTR operation was not provided.
	ACTION	Check to see which parameters are required for this intrinsic, and add them to your program.
6111W	MESSAGE	COULD NOT CLOSE THE INTRINSIC MECHANISM.
	CAUSE	The compiler uas unable to close the SYSINTR.PUB.SYS intrinsic mechanism file.
	ACTION	Refer to the file information display for the actual file system error that occurred. Take the appropriate action for the file system error.
6112I	MESSAGE	ALPHANUMERIC FIELD SPECIFIED WHERE NUMERIC WAS EXPECTED.
	CAUSE	The compiler is informing you that an alphanumeric field is being passed to an intrinsic that expects a numeric field. This is allowed. You can, for instance, pass a long real item to the HPINEXT intrinsic as an 8-byte ASCII string.
	ACTION	If this is what you intended, you may ignore this information.
6113T	MESSAGE	NUMERIC FIELD MUST HAVE 0 DEC PLACES.
	CAUSE	The number being passed to an intrinsic must be an integer.
	ACTION	Correct the field definition for this parameter in your program.
6114W	MESSAGE	THIS INTRINSIC IS NOT A FUNCTION. FACTOR 1 IGNORED.
	CAUSE	The intrinsic specified in an INTR operation does not return a value, but an entry was found in Factor 1 as if a value was expected.
	ACTION	The Factor 1 entry is ignored. To avoid this warning, remove the Factor 1 entry from the INTR operation for this intrinsic.

6115T	MESSAGE	RESET OP FILE TYPE MUST BE I OR U, DESIG MUST BE D OR F.
	CAUSE	The file named in a RESET operation does not have the proper type or designation or both.
	ACTION	Make sure the File Specification type is I or U, and the designation is D or F.
6116T	MESSAGE	RESET OPERATOR MUST HAVE INDICATOR IN COLS 54-55.
	CAUSE	The compiler found no indicator specified in columns 54-55 of a Calculation Specification using the RESET operation.
	ACTION	Add an indicator in columns 54-55 of the operation. Test this indicator following the RESET operation to make sure it worked.
6117T	MESSAGE	CANNOT RESET KSAM FILE IF READ CHRONOLOGICALLY.
	CAUSE	A KSAM file named in a RESET operation has C in column 32 of its File Specification.
	ACTION	A KSAM file that is being read in chronological order cannot be reset.
7001T	MESSAGE	INVALID FILENAME IN COL 7-14, SPEC IS DROPPED.
	CAUSE	You used an undefined file in the File Name Field (columns 7-14).
	ACTION	Define the file name in the File Description Specifications or use a valid file name in the Output Specifications and recompile.
7002T	MESSAGE	'AND' OR 'OR' LINE NOT PRECEDED BY RECORD DESCRIPTION.
	CAUSE	You entered an AND or OR line that is not preceded by a specification that describes the record to which it applies.
	ACTION	Enter a record description specification before this AND/OR line and recompile.
7003T	MESSAGE	NO RECORD DESCRIPTION ENTRIES DEFINED YET.
	CAUSE	You did not specify a record description as the first item in your Output Specification.
	ACTION	Define a valid record description and recompile.
7004W	MESSAGE	INVALID ENTRIES IN COL 32-74 FOR A RECORD DESCRIPTION SPECIFICATION, ASSUME BLANK.
	CAUSE	You entered field description information in a record description specification.
	ACTION	The compiler assumes that columns 32-74 are blank. Correct the record description specification and recompile.
7005W	MESSAGE	INVALID ENTRIES IN COL 7-22 FOR A FIELD DESCRIPTION SPECIFICATION, ASSUME BLANK.
	CAUSE	You entered record description information in a field description specification.
	ACTION	Correct the field description specification and recompile.

7006T	MESSAGE	FILE AND RECORD ENTRIES IN COL 7-31 AND FIELD ENTRIES IN COLS 32-74 ON SAME LINE.
	CAUSE	You entered both a record description and a field description on the same line.
	ACTION	Correct this error and recompile.
7007I	MESSAGE	NO FIELD DESCRIPTION ENTRIES FOR PREVIOUS RECORD.
	CAUSE	You entered a record description specification that is not followed by field description specifications.
	ACTION	Supply a field description specification and recompile.
7008T	MESSAGE	NO RECORD DESCRIPTION ENTRIES FOR THIS FIELD DESCRIPTION ENTRY, SPEC IS DROPPED.
	CAUSE	You entered a field description specification that is not preceded by a record description specification.
	ACTION	Insert the record description entry and recompile.
7009T	MESSAGE	MORE THAN 20 AND/OR LINES, SPEC IS DROPPED.
	CAUSE	You exceeded the limit of 20 AND or OR lines in an AND/OR grouping.
	ACTION	Delete the appropriate number of AND/OR lines and recompile.
7010W	MESSAGE	EDIT WORD IN COL 45-70 IS NOT ENCLOSED IN QUOTES, ASSUME BLANK.
	CAUSE	You did not enclose the edit word in quotation marks.
	ACTION	Add quotation marks to the edit word and recompile.
7011W	MESSAGE	THE NUMBER OF REPLACEABLE CHARACTERS IN THE EDIT WORD IS NOT EQUAL TO THE FIELD TO BE EDITED.
	CAUSE	You did not specify an equal number of replaceable characters in the edit word.
	ACTION	The compiler makes the number of characters in the source data equal to the number of replaceable characters in the edit word, either by truncating or padding with zeros. If you want this result, ignore this warning message.
7012W	MESSAGE	INVALID EDIT CODE IN COL. 38.
	CAUSE	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 (column 38).
	ACTION	Enter a correct character and recompile.

7013W	MESSAGE	EDIT CODE INVALID WITH CONSTANT OTHER THAN '*' OR '$'.
	CAUSE	You paired the wrong edit code with the constant that appears in columns 45-70.
	ACTION	Correct this error and recompile.
7014W	MESSAGE	INVALID FIELD NAME IN COL 32-37.
	CAUSE	You used an invalid or undefined field name in the Field Name Field (columns 32-37).
	ACTION	Correct the field name and recompile.
7015W	MESSAGE	INVALID LINE TYPE IN COL 15, ASSUME D.
	CAUSE	You used a character other than H, D, T, or E in the Type Field (column 15).
	ACTION	Enter the proper character and recompile.
7016W	MESSAGE	INVALID FETCH-OVERFLOW OR RELEASE-FILE IN COL 16, ASSUME BLANK.
	CAUSE	You made an invalid entry in the Fetch Overflow/Release Field (column 16).
	ACTION	Correct this error and recompile.
7017W	MESSAGE	INVALID SPACE ENTRIES IN COL 17-18, ASSUME BLANK.
	CAUSE	You made an entry other than 0, 1, 2, 3, or blank in the Space Field (columns 17-18).
	ACTION	Correct this entry and recompile.
7018W	MESSAGE	INVALID SKIP ENTRIES IN COL 19-22 OR GREATER THAN FORM LENGTH SPECIFIED, ASSUME BLANK.
	CAUSE	You made entries other than 01-99, A0-A9, B0-B2 or blank in the Skip Field (columns 19-22) or the entry in this field exceeds the form length specified.
	ACTION	Correct this error and recompile.
7019W	MESSAGE	INVALID SKIP ENTRIES IN COL 19-22 OR CHANNEL NUMBER GREATER THAN 12, ASSUME BLANK.
	CAUSE	You made invalid skip entries or referred to a channel number greater than 12 in the Skip Field (columns 19-22).
	ACTION	Enter the correct skip request or channel number and recompile.
7020W	MESSAGE	INVALID NOT ENTRY IN COL. 23, 26, OR 29: ASSUME N.
	CAUSE	You entered a character other than N or blank in the Not Subfield (columns 23, 26, or 29).
	ACTION	Leave the field blank or enter N and recompile.

7022W	MESSAGE	INVALID OUTPUT INDICATOR IN COL. 24-25, 27-28, 30-31; ASSUME BLANK.
	CAUSE	You entered an invalid indicator in the Output Indicators Field (columns 24-25, 27-28, or 30-31).
	ACTION	Enter a valid indicator and recompile.
7023W	MESSAGE	INVALID FIELD LENGTH FOR Y EDIT CODE.
	CAUSE	You specified a field less than 3 or greater than 6 digits long for the Y edit code.
	ACTION	Redefine the field length to 3-6 digits.
7024W	MESSAGE	END POSITION NOT LARGE ENOUGH TO CONTAIN *PLACE FIELDS, OR ENTRY IS INVALID.
	CAUSE	You entered a value in the End Position Field (columns 40-43), that is too small to allow output of the *PLACE data without overlapping the previous field.
	ACTION	Correct this error and recompile.
7027W	MESSAGE	MISSING OR INCORRECTLY SPECIFIED END POSITION IN COL. 40-43; ASSUME END POSITION 1.
	CAUSE	You omitted the End Position Field (columns 40-43), or entered a value in it other than 1 through 9999.
	ACTION	Enter the appropriate value in columns 40-43 and recompile.
7028W	MESSAGE	OUTPUT FILE DESCRIBED AS 'ADD' TYPE, EACH RECORD LINE MUST HAVE 'ADD' IN COL. 16-18, ASSUME 'ADD'.
	CAUSE	You did not include ADD in columns 16-18 of each record line for an output file to which new records are to be appended.
	ACTION	Correct this error and recompile.
7029W	MESSAGE	INVALID BLANK AFTER IN COL 39, ASSUME BLANK.
	CAUSE	You entered the Blank After Field with a constant, look-ahead field, or special DATE field.
	ACTION	Remove this entry from the constant or field description and recompile.
7030W	MESSAGE	INVALID CONSTANT IN COLUMNS 45-70.
	CAUSE	You entered an invalid constant in the Constant/Edit Word Field (columns 45-70).
	ACTION	Define the constant properly and recompile.

7031W	MESSAGE	INVALID PACKED/BINARY OUTPUT ENTRY IN COL 44, ASSUME BLANK.
	CAUSE	You made an entry to this field that applies to alphanumeric data but the entry should apply to numeric data; or you entered a character other than P, B, L, R, 2, or 4.
	ACTION	Correct this entry and recompile.
7032W	MESSAGE	OVERFLOW INDICATOR INVALID FOR AN EXCPT RECORD.
	CAUSE	You entered an overflow indicator in the Output Indicators Field (columns 23-31) on an exception record.
	ACTION	Delete the overflow indicator and recompile.
7033W	MESSAGE	FETCH OVERFLOW INVALID WITH OVERFLOW INDICATOR ENTERED IN COL. 23-31; ASSUME NO FETCH.
	CAUSE	You requested Fetch Overflow (column 16) in a line conditioned by an overflow indicator.
	ACTION	Correct this error and recompile.
7034W	MESSAGE	OVERFLOW INDICATOR USED IS NOT ASSIGNED TO THIS FILE.
	CAUSE	You used an overflow indicator in the Output Indicators Field (columns 23-31), but the indicator is not assigned to this file.
	ACTION	The compiler assumes that you assigned the indicator to this file. If this is what you intend, ignore this message; otherwise, correct the record and recompile.
7035W	MESSAGE	1P INDICATOR INVALID WITH TOTAL OR EXCPT RECORDS.
	CAUSE	You used the first-page indicator to condition a total or exception record.
	ACTION	Correct the record description and recompile.
7036W	MESSAGE	FETCH OVERFLOW INVALID WITH 1P INDICATOR, ASSUME NO FETCH OVERFLOW.
	CAUSE	You requested Fetch Overflow (column 16) for a record conditioned with the first-page indicator.
	ACTION	Correct the error and recompile.
7037W	MESSAGE	1P INDICATOR INVALID FOR A COMBINED FILE.
	CAUSE	You entered the first-page (1P) indicator for a combined file.
	ACTION	Correct the error and recompile.
7038W	MESSAGE	INVALID INDICATORS USED IN AN 'AND' RELATIONSHIP WITH 1P.
	CAUSE	You used an indicator with 1P in an AND relationship.
	ACTION	Correct this error and recompile.

7039T	MESSAGE	END POSITION ENTRY IN COL. 40-43 FOR CONSTANT, EDIT WORD, FIELD, OR ARRAY EXCEEDS RECORD LENGTH.
	CAUSE	You specified an end position for an output record that exceeds the maximum length possible for that record. (For instance, position 150 for a printer record that has 132 characters per line.
	ACTION	Correct the end position to conform with the maximum length possible and recompile.
7040W	MESSAGE	CONSTANTS IN COL. 45-70 INVALID FOR X, Y, OR Z EDIT CODES.
	CAUSE	You did not leave columns 45-70 blank for an X, Y, or Z edit code.
	ACTION	Remove the characters in columns 45-70 and recompile.
7041W	MESSAGE	DECIMAL POSITIONS INVALID FOR FIELD EDITED BY Y CODE.
	CAUSE	You applied the Y edit code to a non-integer numeric field.
	ACTION	Correct this error and recompile.
7042W	MESSAGE	NAME OF FIELD TO BE EDITED, BY CODE SPECIFIED IN COL 38, MISSING.
	CAUSE	You specified an edit code in column 38 but did not enter a field name in columns 32-37.
	ACTION	Enter a field name in columns 32-37 and recompile.
7043T	MESSAGE	INVALID FILE TYPE FOR OUTPUT RECORD.
	CAUSE	You directed output to an input or display file.
	ACTION	Correct this error and recompile.
7044W	MESSAGE	T OR E ENTRY IN COL. 15 INVALID FOR COMBINED FILE.
	CAUSE	You directed total or exception records to a combined file.
	ACTION	Correct this error and recompile.
7045W	MESSAGE	BLANK AFTER ENTRY IN COL 39 INVALID WITH RESERVED WORD OTHER THAN PAGE; ASSUME BLANK.
	CAUSE	You entered the Blank After Field with *PLACE, *PRINT, UDATE, UMONTH, UYEAR, UDAY, or *ERROR.
	ACTION	Delete the blank after request from column 39.
7046W	MESSAGE	BLANK AFTER SPECIFIED FOR A CONSTANT.
	CAUSE	You tried to blank out a constant instead of a data field.
	ACTION	Correct this error and recompile.

7047I	MESSAGE	FORMS POSITIONING SPECIFIED ON CONTROL RECORD, BUT MISSING A RECORD CONDITIONED BY 1P.
	CAUSE	You entered a 1 in the Form Positioning Field (column 41) of the Header Specification, but did not specify an output record conditioned by the first-page indicator (1P).
	ACTION	This is an informational message only, no action is necessary.
7048W	MESSAGE	'ADD' IN COL 16-18 NOT ALLOWED ON AND/OR LINES, ASSUME BLANK.
	CAUSE	You used ADD on an AND or OR line.
	ACTION	Remove the ADD and recompile.
7049W	MESSAGE	INVALID ENTRY IN COL. 17-22 FOR 'AND' LINE, ASSUME BLANK.
	CAUSE	You entered one or more characters in columns 17-22 on an AND line.
	ACTION	Remove the characters from these columns and recompile.
7050W	MESSAGE	INVALID INDEX IN COLS. 32-37.
	CAUSE	You used an improper array index in the Field Name Field.
	ACTION	Correct the invalid index and recompile.
7053W	MESSAGE	ALPHANUMERIC DATA CANNOT BE EDITED.
	CAUSE	You attempted to apply an edit code or edit word to alphanumeric data, but they can only be used with numeric data.
	ACTION	Correct this record and recompile.
7054W	MESSAGE	PACKED/BINARY IN COLUMN 44 CANNOT APPEAR TOGETHER WITH EDIT CODE OR EDIT WORD.
	CAUSE	You attempted to apply a packed binary field to an edit code or edit word.
	ACTION	Correct this error and recompile.
7055W	MESSAGE	END POSITION SPECIFIED IN COLUMNS 40-43 CANNOT CONTAIN THE WHOLE DATA ITEM.
	CAUSE	You did not specify an edit field long enough to hold the entire edited item.
	ACTION	Specify the proper field length and recompile.
7056W	MESSAGE	PACKED/BINARY IN COLUMN 44 IS NOT BLANK FOR ALPHANUMERIC DATA, ASSUME BLANK.
	CAUSE	You specified numeric data (column 44) but defined the field previously as alphanumeric.
	ACTION	Correct this error and recompile.

7057W	MESSAGE	TARGET LENGTH SPECIFIED BY PACKED/BINARY IN COLUMN 44 MAY BE TOO SMALL TO CONTAIN THE DATA.
	CAUSE	You may have specified an output field too short to hold the output data.
	ACTION	Specify the proper output field length and recompile.
7058W	MESSAGE	THE SKIP IN COLUMNS 19-22 MUST NOT BE BEYOND THE FORM LENGTH DEFINED ON THE LINE COUNTER SHEET.
	CAUSE	You requested a skip to a line beyond the form length but still on the current page.
	ACTION	Correct the skip request and recompile.
7059W	MESSAGE	INVALID ENTRY IN COL 7-13 FOR 'AND/OR' LINE, ASSUME BLANK.
	CAUSE	You entered characters in these columns, but none are allowed for an AND or OR line.
	ACTION	Remove the characters from columns 7-13 and recompile.
7060T	MESSAGE	'DEL' IN COLS. 16-18 ONLY ALLOWED FOR AN IMAGE/KSAM/INDEX UPDATE FILE.
	CAUSE	You specified DEL for the wrong type of file.
	ACTION	Correct this error and recompile.
7061W	MESSAGE	'ADD' IN COLS. 16-18, BUT FILE IS NOT 'ADD' TYPE, BLANK IS ASSUMED.
	CAUSE	You specified ADD in columns 16-18 of the Output Specifications but did not also specify A in the File Addition Field (column 66) of the File Description Specifications.
	ACTION	Remove ADD from columns 16-18 of the Output Specifications, or specify A in column 66 of the File Description Specifications.
7063T	MESSAGE	IMAGE OPEN MODE 2 NOT ALLOWED FOR DATABASE SPECIFIED AS OUTPUT, UPDATE-ADD, OR UPDATE-DEL.
	CAUSE	You cannot specify Open Mode 2 for this database.
	ACTION	Correct the program and recompile.
7064I	MESSAGE	OUTPUT, UPDATE, OR COMBINED FILE HAS NO OUTPUT SPECIFICATIONS.
	CAUSE	You have not entered Output Specifications for an output, update, or combined file.
	ACTION	This is an informational message only, no action is necessary.
7065W	MESSAGE	FETCH OVERFLOW INVALID IF NO OVERFLOW INDICATOR ASSIGNED TO FILE. ASSUME NO FETCH OVERFLOW.
	CAUSE	You entered F in column 16 of the Output Specifications but did not enter an overflow indicator in columns 33-34 of the File Description Specification.
	ACTION	Change the program and recompile.
7066W	MESSAGE	SPACE BEFORE/AFTER INVALID FOR INDEXED FILE, ASSUME BLANK.
	CAUSE	You specified spacing in columns 17-18 of an indexed file.
	ACTION	The compiler assumes the entry is blank.

7067W	MESSAGE	COL 16-18 NOT BLANK, 'ADD' OR 'DEL' FOR UPDATE FILE, ASSUME UPDATE OPERATIONS - NOT ADD.
	CAUSE	You did not enter a blank, ADD or DEL in columns 16-18 for an update file.
	ACTION	The compiler performs update operations on the file, not add operations. Correct the program and recompile.
7068I	MESSAGE	PREVIOUS RECORD CONTAINS NO INDICATORS FOR 'AND' RECORD.
	CAUSE	The previous record does not have indicators in this AND sequence.
	ACTION	Verify that this is what you want. Modify the program if necessary.
7069W	MESSAGE	EXCPT GROUP NAME NOT USED - SEE WARNING AFTER O-SPECS
	CAUSE	In a calculation specification using the EXCPT operation, you provided a name in the result field for an EXCPT operation, but you did not use this name to condition any exception output records.
	ACTION	If you misspelled or forgot to enter the EXCPT name in colums 32-37 of the appropriate exception output specifications (E in column 15), enter the name exactly as it appears in the calculation specification.
7070W	MESSAGE	UPDATE FILE WITH ADD OPTION, WITH NO ADD IN COLS 16-18. CHECK USAGE.
	CAUSE	A file specification indicated that a file would be updated by appending records to it (F-spec, columns 15=U and 66=A), but an output record was not marked as an ADD record (O-spec, columns 16-18="ADD").
	ACTION	Enter "ADD" in columns 16-18 of the output record specification if you want to add the record to the end of the file. Ignore the warning if you entered "DEL" to delete the last record read, or blanks to update the last record read.
8001W	MESSAGE	ERROR IN COMMENT
	CAUSE	The set of broken brackets is not correct.
	ACTION	The compiler ignores the command image from the comment on. Correct and recompile.
8002W	MESSAGE	MISSING SYMBOL
	CAUSE	The compiler expects a symbol after the $ in column 6.
	ACTION	The compiler ignores the rest of the command image, including any incomplete option present. Correct and recompile.
8003W	MESSAGE	ILLEGAL SYMBOL
	CAUSE	The compiler encountered an unexpected symbol.
	ACTION	The compiler ignores the symbol. Correct and recompile.
8004W	MESSAGE	MISSING OPTION OR INVALID COMMAND
	CAUSE	The compiler either did not encounter an option for the $CONTROL subsystem command or it encountered an invalid option for it.
	ACTION	The compiler ignores the command image. Correct and recompile.

8005W	MESSAGE	MISSING COMMA
	CAUSE	You did not include a comma, which is expected as a separator.
	ACTION	The compiler assumes that a comma is present. Correct and recompile.
8006W	MESSAGE	MISSING EQUAL SIGN
	CAUSE	You did not include an equal sign expected after a keyword.
	ACTION	The compiler assumes that an equal sign is present. Correct and recompile.
8007W	MESSAGE	ILLEGAL NUMBER
	CAUSE	You specified a alphanumeric entry where a numeric entry is required.
	ACTION	The compiler ignores the option currently being scanned. Correct and recompile.
8008W	MESSAGE	NUMBER OUT OF RANGE
	CAUSE	A numeric entry is out of range.
	ACTION	The compiler ignores the option currently being scanned. Correct and recompile.
8009W	MESSAGE	STRING TOO LARGE
	CAUSE	The concatenation of character substrings exceeds a total of 103 characters.
	ACTION	The string is truncated at 103 characters. Correct and recompile.
8010W	MESSAGE	ILLEGAL STRING
	CAUSE	You failed to terminate a character string with a quotation mark.
	ACTION	The compiler ignores the string. Correct and recompile.
8012W	MESSAGE	MAXIMUM ERRORS EXCEEDED.
	CAUSE	The maximum number of errors allowed by the ERRORS= parameter of the $CONTROL subsystem command (or the default value of 100) has been exceeded.
	ACTION	Reduce errors and recompile.
8013W	MESSAGE	DUPLICATE ENTRY - IGNORED.
	CAUSE	You specified a $ command or option more than once.
	ACTION	Remove duplicate entry.
8014W	MESSAGE	ILLEGAL NAME
	CAUSE	You specified an incorrect value in $CONTROL NAME= option.
	ACTION	Correct the option and recompile.
8015W	MESSAGE	$COPY NOT FIRST LINE OF PROGRAM OR COPYLIB PREPROCESSING NOT DONE - SPECIFICATION DROPPED.
	CAUSE	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.
	ACTION	Specify the $COPY command properly. Make sure RPGCOPY.PUB.SYS is available and can be executed; then recompile.

8016I	MESSAGE	$SECOND '$' IGNORED.
	CAUSE	There is a $ in column 7 of a command line. This column is not used.
	ACTION	None, the entry is ignored.
8019W	MESSAGE	CONTINUATION RECORD MUST HAVE '$' IN COLUMN 6.
	CAUSE	A Continuation line for a compiler subsystem command was found that does not have $ in column 6.
	ACTION	The Continuation line is ignored. Correct the program and recompile.
8020W	MESSAGE	INVALID 'IF' PARAMETER.
	CAUSE	An incorrectly formatted $IF compiler subsystem command was found.
	ACTION	The Continuation line is ignored. Correct the program and recompile.
8021W	MESSAGE	INVALID 'SET' PARAMETER.
	CAUSE	An incorrectly formatted $SET compiler subsystem command was found.
	ACTION	The Continuation line is ignored. Correct the program and recompile.
8022W	MESSAGE	INVALID 'CONTROL' PARAMETER.
	CAUSE	An incorrectly formatted $CONTROL compiler subsystem command was found.
	ACTION	The Continuation line is ignored. Correct the program and recompile.
8023W	MESSAGE	OPTION NOT SUPPORTED ON RPG/iX.
	CAUSE	An unsupported entry (for example, $EDIT) was found.
	ACTION	The entry is ignored. Correct the program and recompile.
9001I	MESSAGE	INDICATOR DEFINED BUT NOT REFERENCED.
	CAUSE	You defined an indicator that is not referenced in a specification.
	ACTION	If you intended to reference this indicator, enter a correct reference and recompile.
9002W	MESSAGE	INDICATOR REFERENCED BUT NOT DEFINED.
	CAUSE	Your program referenced an indicator that is not defined by a specification.
	ACTION	Correct the error and recompile.
9003T	MESSAGE	INVALID FILENAME IN COLUMNS 7-14, SPEC DROPPED.
	CAUSE	You used an improper file name in the File Name Field for a Array/Table File Name (A) Specification.
	ACTION	The compiler ignores this specification.

9004T	MESSAGE	CANNOT OPEN FILE SPECIFIED BY 'A' SPEC.
	CAUSE	The system could not open the file named on the Array/Table File Name (A) Specification.
	ACTION	Check to see if the file exists and determine the reason for this file error. Correct the error and recompile.
9005T	MESSAGE	CANNOT OPEN TEMPORARY FILE TO PROCESS COMPILE-TIME TABLE.
	CAUSE	The compiler cannot open the temporary file for processing a compile-time table.
	ACTION	Check to see if disk space is full and determine the reason for this file error. Correct the error and recompile.
9006W	MESSAGE	THE FILE DEFINED BY 'A' SPEC IS EMPTY.
	CAUSE	The file named on the Array/Table File Name Specification does not contain table or array records.
	ACTION	Supply the compiler with the correct file and recompile your program.
9007T	MESSAGE	I/O ERROR OCCURRED DURING READING COMPILE-TIME TABLE.
	CAUSE	The system could not read the compile-time table.
	ACTION	Check to determine the reason for this file error. Correct it then recompile.
9008T	MESSAGE	I/O ERROR OCCURRED DURING WRITING COMPILE-TIME TABLE.
	CAUSE	An input/output error occurred while the compiler was copying your table/array files to a temporary file; the most probable cause was insufficient disk space.
	ACTION	Check to determine the reason for this file error. Correct it then recompile.
9009W	MESSAGE	ALTERNATE COLLATING SEQUENCE, COMPILE-TIME TABLE/ARRAY OR TRANSLATION TABLE NOT FOUND.
	CAUSE	You did not define an alternate collating sequence, table or array, or file translation table in the file named on the required Array/Table File Name (A) Specification.
	ACTION	Supply an Array/Table File Name Specification (after the last Output Specification) and recompile.
9010W	MESSAGE	THE FILES SPECIFIED BY 'A' SPEC ARE NOT ENOUGH TO INITIALIZE COMPILE-TIME TABLES.
	CAUSE	The files named on the Array/Table File Name (A) Specification do not contain enough entries to fill the tables or arrays.
	ACTION	Supply the compiler with a proper file (or files) and recompile.

9011W	MESSAGE	ILLEGAL HEXADECIMAL DIGIT IN COMPILE-TIME TABLE.
	CAUSE	You entered an invalid hexadecimal value (other than 0-9 or A-F) in the compile-time table. This value does not represent an ASCII character.
	ACTION	The compiler assumes that this character is zero. Correct and recompile.
9012W	MESSAGE	ILLEGAL OCTAL DIGIT IN COMPILE-TIME TABLE.
	CAUSE	You entered an invalid octal value (other than 0-7) in a compile-time table. This value does not represent an ASCII character.
	ACTION	The compiler assumes that this character is zero. Correct and recompile.
9013W	MESSAGE	THE NAMES IN COLUMNS 1-8 OF A FILE TRANSLATION TABLE ARE NOT THE SAME.
	CAUSE	You specified more than one file name in columns 1-8 of a group of file translation records supposedly belonging to the same file.
	ACTION	The compiler assumes that the file names are the same.
9014I	MESSAGE	FIELD NAME(S) DEFINED BUT NOT REFERENCED.
	CAUSE	You did not reference the field names you defined.
	ACTION	Correct the program and recompile.
9015I	MESSAGE	FILE NAME(S) DEFINED BUT NOT REFERENCED.
	CAUSE	You did not reference file names you defined.
	ACTION	Correct the program and recompile.
9016I	MESSAGE	NUMERIC FIELD TEMPORARILY CONVERTED TO ALPHANUMERIC FOR ALPHANUMERIC COMPARE.
	CAUSE	You are attempting to compare an alphanumeric field to a numeric field.
	ACTION	RPG temporarily converts the numeric field to alphanumeric to do the comparison. This is an information message only.
9017T	MESSAGE	AUTOMATIC RECORD-LEVEL LOCKING FOR IMAGE OUTPUT REQUIRES ENTRY IN FILE DESC. SPEC COLUMNS 35-38.
	CAUSE	You entered R in the Open Mode Field (column 66) of the Database Name (IMAGE) Line without entering a Key Field Starting Location (columns 35-38) in the File Description Specification.
	ACTION	Correct the program and recompile.
9018W	MESSAGE	COMPILE-TIME TABLE ERROR - SEQUENCE ERROR.
	CAUSE	You specified an incorrect order for the compile-time table.
	ACTION	Correct the program and recompile.

9019W	MESSAGE	COMPILE-TIME TABLE ERROR - BAD DATA (INVALID DIGIT).
	CAUSE	You specified invalid digits in a numeric table or array.
	ACTION	Correct the program and recompile.
9020W	MESSAGE	COMPILE-TIME TABLE ERROR - TOO MANY ITEMS FOR TABLE/ARRAY.
	CAUSE	You specified too many items in the compile-time table.
	ACTION	Correct the program and recompile.
9021T	MESSAGE	FOR LOKUP OPERATION, STORAGE LENGTHS OF FACTOR 1 AND FACTOR 2 ARE NOT THE SAME.
	CAUSE	You specified different lengths for FACTOR 1 (search argument of LOKUP) and FACTOR 2 (the table or array being searched).
	ACTION	Correct the program and recompile.
9022I	MESSAGE	ALTSEQ WILL NOT BE APPLIED TO THIS OPERATION.
	CAUSE	You declared an alternate collating sequence; however, the compiler does not use the alternate sequence for this operation.
	ACTION	Correct the program and recompile, if necessary.
9023T	MESSAGE	FACTOR 2 OR RESULT FIELD MUST BE A NUMERIC VALUE FROM 1 TO 40 FOR TIME2.
	CAUSE	You entered incorrect values in either the Factor 2 Field or the Field Length Field for this operation.
	ACTION	Correct the program and recompile.
9024T	MESSAGE	START POSITION SPECIFIED IN FACTOR 2 PLUS RESULT FIELD LENGTH IS BEYOND 40 FOR TIME2.
	CAUSE	You specified an incorrect combination of entries for the Factor 2 Field and the Field Length Field.
	ACTION	Correct the program and recompile.
9025W	MESSAGE	MOVE OF FIELD TO WHOLE ARRAY RESULTS IN DATA TRUNCATION AND REPETITION THROUGH THE ARRAY.
	CAUSE	You specified a simple field for Factor 2 in a MOVE operation and an non-subscripted name in the Result Field. This results in the field being moved independently into each array element rather than overlaying the array.
	ACTION	Correct the program and recompile. If you want to overlay the array, change the MOVE operation to MOVEA.
9026W	MESSAGE	NO. OF COMPILE-TIME TABLES/ARRAYS FOUND EXCEED NUMBER DEFINED ON E-SPECS.
	CAUSE	There are more compile-time tables or arrays found at the end of the program than there are defined in E-specs.
	ACTION	Correct the program and recompile.

9030T	MESSAGE	RESULT FIELD LENGTH IN COLUMN 49-51 OF C SPEC MUST BE 6 OR 12 FOR TIME OPERATION.
	CAUSE	You specified an incorrect entry in columns 49-51 for this operation.
	ACTION	Correct the program and recompile.
9031T	MESSAGE	DECIMAL POSITIONS FOR FNDJW, PUTJW, TIME, TIME2, OR FNUM MUST BE 0.
	CAUSE	An incorrect decimal position was specified for a field in one or more of the above operations.
	ACTION	Correct the program and recompile.
9032T	MESSAGE	DIGIT LENGTH FOR VARIABLE IN FACTOR 2 MUST BE 1 TO 4 DIGITS FOR TIME2 OPERATION.
	CAUSE	You specified an incorrect digit length for this field.
	ACTION	Correct the program and recompile.
9033T	MESSAGE	FIELD SIZE MUST BE 1 TO 8 DIGITS FOR PUTJW OPERATION.
	CAUSE	The variable specified in Factor 1 for this operation has an incorrect number of digits specified for its field length.
	ACTION	Correct the program and recompile.
9034T	MESSAGE	TRIED TO BUILD MORE THAN 1 TRANSLATION TABLE FOR FILE.
	CAUSE	A file using a translation table was named more than once in an Array/Table File Name Specification.
	ACTION	Correct the error and recompile.
9035T	MESSAGE	INVALID FILE NAME IN FILE TRANSLATION RECORD.
	CAUSE	An invalid file name was found in a file translation record.
	ACTION	Check the File Extension Specification(s); you may have too many compile-time tables. Correct the error and recompile.
9036W	MESSAGE	FACTOR 1 IN SET OPERATION, IF USED, MUST BE AN ARRAY NAME.
	CAUSE	Something besides a valid array name was specified for Factor 1 in a SET operation.
	ACTION	Make sure Factor 1 is either blank or is a valid array name.
9040W	MESSAGE	TAG NOT FOUND IN CURRENT SUBROUTINE.
	CAUSE	The label referenced by a CAB or GOTO operation does not exist in the current subroutine.
	ACTION	Check the spelling or location of the label referenced in the CAB or GOTO operation, and its associated TAG operator. You may not branch out of a subroutine.

B

Run-Time Messages

This chapter lists the errors that you may encounter when running an RPG program. There are four types of errors. They are summarized below and are listed (except for TurboIMAGE errors) in the same order in this appendix:

- RPG errors

  These errors are detected and reported by compiler-generated code. They are displayed on the operator's console (for batch jobs) or on the terminal (for interactive sessions). The program then pauses while the operator decides what action to take. The operator can:

  1. Continue processing.
  2. Skip the error record and continue processing.
  3. Terminate the program.

- TurboIMAGE-detected errors

  These errors are detected by the TurboIMAGE Subsystem. See the *TurboIMAGE/iX Database Management System* manual for an explanation of these errors.

- USWITCH command errors

  These errors may occur when you enter an F in the USWITCH Source Field (column 16) of the Header Specification. Errors in USWITCH records may terminate the program.

- BUFCHK errors

  These errors may occur when you enter BUFCHK in the Option Type Field (columns 54-59) of a File Description Continuation line.

RPG Errors

RPG errors are detected and reported by compiler-generated code. You can specify in the program what action to take when an error occurs. You can redirect or suppress the error action. To do this, enter the appropriate values in both of these Header Specification fields:

1. The Error Log Field (column 55).
2. The Error Response Field (columns 56-71).

The messages listed in this appendix explain the cause of the RPG error and the action you must take to correct it. In addition to the "Message", "Cause" and "Action" fields, the three additional fields shown below are listed for each error:

- ERROR RESPONSE COLUMN

  This is the column number (56-71) in the Error Response Field that corresponds to the error.

- *ERROR

  This is the predefined, one-character field where RPG saves a unique letter code for the error. When you enter 0 or 1 in the Error Response Field, you can check the contents of *ERROR using Calculation Specifications.

- ERROR RESPONSE ENTRIES

  This field lists the values that you can enter in the Error Response Field.

1	MESSAGE	FATAL FILE ERROR, FILENAME = xxxx
	CAUSE	An operating system error (see ERROR NUMBER in the File Information Display) or an TurboIMAGE error.
	ACTION	Determine the cause of the error from the text displayed with it and correct the program. If you need additional information, look up the error in the operating system manual or the *TurboIMAGE/iX Database Management System* manual.
	ERROR RESPONSE COLUMN	NONE (this error is processed with error number 5).
	*ERRORF	
	ERROR RESPONSE ENTRIES	NONE.
2	MESSAGE	UNIDENTIFIED RECORD, FILENAME = xxxx, RECORD NUMBER = nnn
	CAUSE	Record number **nnn** on the input file named **xxxx** is not identifiable.
	ACTION	Choose one of the error response options. Either include this record type on Input Specifications or delete the record from the file.
	ERROR RESPONSE COLUMN	56.
	*ERROR	A.
	ERROR RESPONSE ENTRIES	0, 1, 2, 3, 4, 5.
3	MESSAGE	MATCHING RECORD SEQUENCE ERROR, FILENAME = xxxx, RECORD NUMBER = nnn
	CAUSE	The matching record with record number = **nnn** and filename = **xxxx** is not in sequence.
	ACTION	Choose one of the error response options and correct the record sequence.
	ERROR RESPONSE COLUMN	57.
	*ERROR	B.
	ERROR RESPONSE ENTRIES	1, 2, 3, 4, 5.

4	MESSAGE	INPUT SEQUENCE ERROR, FILENAME = xxxx, RECORD NUMBER = nnn
	CAUSE	Record number **nnn** on the input file named **xxxx** is not in the proper sequence.
	ACTION	Put the record in the proper sequence and execute the program.
	ERROR RESPONSE COLUMN	58.
	*ERROR	C.
	ERROR RESPONSE ENTRIES	1, 2, 3, 4, 5.
5	MESSAGE	INDEX INVALID, LINE NUMBER =nnn, INDEX = nnn NOT EVALUATED
	CAUSE	The array index is less than 1 or greater than the number of array elements.
	ACTION	Choose one of the error response options and correct the program.
	ERROR RESPONSE COLUMN	59.
	*ERROR	D.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.
6	MESSAGE	NEGATIVE SQUARE ROOT, LINE NUMBER = nnn
	CAUSE	Your program calculated a square root that was negative.
	ACTION	Choose one of the error response options and correct the data.
	ERROR RESPONSE COLUMN	60.
	*ERROR	E.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.

7	MESSAGE	TABLE/ARRAY {SEQUENCE, BAD DATA, OVERFILL} FILE = xxxx, LINE NUMBER = nnn, VALUE = xxx
	CAUSE	The table or array on file **xxxx** with a line number = **nnn** and value = **xxx** had a sequence, bad data or overfill error.
	ACTION	Choose one of the error response options and correct the table or array.
	ERROR RESPONSE COLUMN	61.
	*ERROR	H.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.
8	MESSAGE	RECORD NOT FOUND, FILENAME = xxx, STATUS = xxx
	CAUSE	The record sought was not found. (The STATUS portion appears only for a TurboIMAGE file.)
	ACTION	Choose one of the error response options and make the appropriate changes.
	ERROR RESPONSE COLUMN	62.
	*ERROR	U.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.
9	MESSAGE	SPECIAL FILE ERROR, FILENAME = xxxx
	CAUSE	The special file processing routine returned an error.
	ACTION	Check the processing routine and make the appropriate changes.
	ERROR RESPONSE COLUMN	63.
	*ERROR	J.
	ERROR RESPONSE ENTRIES	0, 1, 2, 3, 4, 5.

10	MESSAGE	RDEXIT ERROR, FILENAME =xxxx
	CAUSE	The Read Exit file processing routine returned an error.
	ACTION	Check the processing routine and make the appropriate changes.
	ERROR RESPONSE COLUMN	64.
	*ERROR	J.
	ERROR RESPONSE ENTRIES	0, 1, 2, 3, 4, 5.
11	MESSAGE	ARITHMETIC OVERFLOW, LINE NUMBER = nnn, VALUE = nnn
	CAUSE	The arithmetic result exceeded the number of digits the field can hold.
	ACTION	Correct the program (make the field size larger).
	ERROR RESPONSE COLUMN	65.
	*ERROR	N.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.
12	MESSAGE	DIVIDE BY ZERO, LINE NUMBER = nnn
	CAUSE	The number was divided by zero.
	ACTION	Choose one of the error response options and correct the program.
	ERROR RESPONSE COLUMN	66.
	*ERROR	O.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.

13	MESSAGE	INVALID NUMERICAL DATA, LINENUM = nnn, FILENAME = xxxx, RECORD NUMBER = nnn, COLUMN = nnn, VALUE = nnn
	CAUSE	Either an input field or an alphanumeric field, being moved to a numeric field, contained characters other than numeric characters or embedded blanks. (FILENAME, RECORD NUMBER, COLUMN and VALUE are printed only when applicable.)
	ACTION	Choose one of the error response options and correct the program.
	ERROR RESPONSE COLUMN	67.
	*ERROR	P.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.
14	MESSAGE	BINARY CONVERSION OVERFLOW, LINENUM = nnn, VALUE = nnn
	CAUSE	The binary output field was not large enough to hold the value.
	ACTION	Define a larger binary field.
	ERROR RESPONSE COLUMN	68.
	*ERROR	Q.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.
15	MESSAGE	INDICATOR HX IS ON (H0-H9)
	CAUSE	These halt indicators were either turned on programmatically or turned on due to a file error.
	ACTION	Choose one of the error response options and correct the program.
	ERROR RESPONSE COLUMN	69.
	*ERROR	NONE.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.

16	MESSAGE	REL REC# INVALID, FILENAME = xxxx
	CAUSE	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.
	ACTION	Choose one of the error response options and correct the program.
	ERROR RESPONSE COLUMN	70.
	*ERROR	M.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.
17	MESSAGE	DUPLICATE KEY, FILENAME = xxxx, KEY = kkkk
	CAUSE	The program attempted to write a record having the same key as a record already in the file and duplicate keys are not allowed.
	ACTION	Redefine the file to permit duplicate keys or delete one of the records.
	ERROR RESPONSE COLUMN	71.
	*ERROR	Y.
	ERROR RESPONSE ENTRIES	0, 2, 3, 4, 5.

USWITCH Errors

If you're running a program that uses the USWITCH feature (F in the USWITCH Source Field (column 16) of the Header Specification), you may encounter one of the following errors. Each USWITCH error is preceded by the message: `ERROR IN USER SWITCH INITIALIZATION.`

`MESSAGE`	`I/O ERROR ON $STDLIST`
`CAUSE`	An attempt to write output to a standard list device failed (this occurs when the display of a user prompt fails).
`ACTION`	The job or session terminates. Determine the reason; correct and rerun.
`MESSAGE`	`I/O ERROR ON $STDIN`
`CAUSE`	An attempt to read input from the standard input device failed (this occurs when getting USWITCH settings from either the user or from a job file.)
`ACTION`	The job or session terminates. Determine the reason; correct and rerun.
`MESSAGE`	`ILLEGAL USWITCH SETTING IN 8 BYTE FORMAT`
`CAUSE`	A character other than 0, 1 or X is in 8-byte (short) format.
`ACTION`	Correct the input data (in the job file or in the USWITCH file) and rerun the program.
`MESSAGE`	`INVALID INPUT DATA`
`CAUSE`	You included invalid information in a USWITCH command.
`ACTION`	If the switch setting comes from a USWITCH file, the program terminates. If it comes from $STDIN, the setting defaults to OFF. Correct and rerun.
`MESSAGE`	`UNABLE TO OPEN FILE 'USWITCH'`
`CAUSE`	An error occurred while opening the USWITCH file.
`ACTION`	Determine the cause of the error and rerun.
`MESSAGE`	`UNABLE TO READ RECORD IN FILE 'USWITCH'`
`CAUSE`	An error occurred while reading the USWITCH file.
`ACTION`	Determine the cause of the error and rerun.
`MESSAGE`	`UNEXPECTED END OF FILE`
`CAUSE`	An end-of-file indicator was unexpectedly encountered in the USWITCH file.
`ACTION`	The program terminates. Correct and rerun.

BUFCHK Errors

The following errors may occur when you use the BUFCHK option (see the Option Type Field (columns 54-59) of the File Description Continuation line). In addition to showing the cause of the error and remedy for it, an *ERROR entry is listed for each message. *ERROR is a one-character predefined field that contains the error code. You can test it in Calculation Specifications.

0-10	MESSAGE	INTERNAL OR INTRINSIC ERROR
	CAUSE	Internal RPG error.
	ACTION	Contact HP Support Engineering.
	*ERROR	F.
11	MESSAGE	ATTEMPTED UPDATE BEFORE INPUT OF FIRST RECORD.
	CAUSE	Attempted to perform record update before reading a record from a file.
	ACTION	Correct the program and recompile.
	*ERROR	U.
12	MESSAGE	ATTEMPTED UPDATE ON SAME RECORD OR AN INTERVENING "ADD" RECORD.
	CAUSE	You tried to update a record that has already been updated (you did not read a new record). Or, the current record was not written before you attempted to add a new one (the current record updates the added record).
	ACTION	Correct the program and recompile.
	*ERROR	U.

C

ASCII and EBCDIC Character Sets

Table C-1, which follows, lists the ASCII and EBCDIC character sets along with the decimal, octal, and hexadecimal values that correspond to them. The ASCII and EBCDIC characters are listed in ascending sequence according to their binary value.

Table C-1. ASCII and EBCDIC Character Sets

Dec	Oct	Hex	Binary	ASCII EBCDIC (alpha/cntrl)	ASCII EBCDIC (numeric)
0	000	00	0000 0000	NUL NUL	
1	001	01	0000 0001	SOH SOH	
2	002	02	0000 0010	STX STX	
3	003	03	0000 0011	ETX ETX	
4	004	04	0000 0100	EOT PF	
5	005	05	0000 0101	ENQ HT	
6	006	06	0000 0110	ACK LC	
7	007	07	0000 0111	BEL DEL	
8	010	08	0000 1000	BS	
9	011	09	0000 1001	HT	
10	012	0A	0000 1010	LF SMM	
11	013	0B	0000 1011	VT VT	
12	014	0C	0000 1100	FF FF	
13	015	0D	0000 1101	CR CR	
14	016	0E	0000 1110	SO SO	
15	017	0F	0000 1111	SI SI	
16	020	10	0001 0000	DLE DLE	
17	021	11	0001 0001	DC1 DC1	
18	022	12	0001 0010	DC2 DC2	
19	023	13	0001 0011	DC3 TM	
20	024	14	0001 0100	DC4 RES	
21	025	15	0001 0101	NAK NL	
22	026	16	0001 0110	SYN BS	
23	027	17	0001 0111	ETB IL	
24	030	18	0001 1000	CAN CAN	
25	031	19	0001 1001	EM EM	
26	032	1A	0001 1010	SUB CC	
27	033	1B	0001 1011	ESC CU1	
28	034	1C	0001 1100	FS IFS	
29	035	1D	0001 1101	GS IGS	
30	036	1E	0001 1110	RS IRS	
31	037	1F	0001 1111	US IUS	
32	040	20	0010 0000	SP DS	
33	041	21	0010 0001	! SOS	
34	042	22	0010 0010	" FS	
35	043	23	0010 0011	#	
36	044	24	0010 0100	$ BYP	
37	045	25	0010 0101	% LF	
38	046	26	0010 0110	& ETB	
39	047	27	0010 0111	' ESC	
40	050	28	0010 1000	(	
41	051	29	0010 1001	)	
42	052	2A	0010 1010	* SM	
43	053	2B	0010 1011	+ CU2	
44	054	2C	0010 1100	,	

Table C-1. ASCII and EBCDIC Character Sets (continued)

Dec	Oct	Hex	Binary	ASCII EBCDIC (alpha/cntrl)	ASCII EBCDIC (numeric)
45	055	2D	0010 1101	- ENQ	
46	056	2E	0010 1110	. ACK	
47	057	2F	0010 1111	/ BEL	
48	060	30	0011 0000	0	0
49	061	31	0011 0001	1	1
50	062	32	0011 0010	2 SYN	2
51	063	33	0011 0011	3	3
52	064	34	0011 0100	4 PN	4
53	065	35	0011 0101	5 RS	5
54	066	36	0011 0110	6 UC	6
55	067	37	0011 0111	7 EOT	7
56	070	38	0011 1000	8	8
57	071	39	0011 1001	9	9
58	072	3A	0011 1010	:	
59	073	3B	0011 1011	; CU3	
60	074	3C	0011 1100	< DC4	
61	075	3D	0011 1101	= NAK	
62	076	3E	0011 1110	>	
63	077	3F	0011 1111	? SUB	
64	100	40	0100 0000	@ SP	
65	101	41	0100 0001	A	+1
66	102	42	0100 0010	B	+2
67	103	43	0100 0011	C	+3
68	104	44	0100 0100	D	+4
68	105	45	0100 0101	E	+5
70	106	46	0100 0110	F	+6
71	107	47	0100 0111	G	+7
72	110	48	0100 1000	H	+8
73	111	49	0100 1001	I	+9
74	112	4A	0100 1010	J	-1
75	113	4B	0100 1011	K .	-2
76	114	4C	0100 1100	L <	-3
77	115	4D	0100 1101	M (	-4
78	116	4E	0100 1110	N +	-5
79	117	4F	0100 1111	O |	-6
80	120	50	0101 0000	P &	-7
81	121	51	0101 0001	Q	-8
82	122	52	0101 0010	R	-9
83	123	53	0101 0011	S	
84	124	54	0101 0100	T	
85	125	55	0101 0101	U	
86	126	56	0101 0110	V	
87	127	57	0101 0111	W	
88	130	58	0101 1000	X	
89	131	59	0101 1001	Y	

Table C-1. ASCII and EBCDIC Character Sets (continued)

Dec	Oct	Hex	Binary	ASCII EBCDIC (alpha/cntrl)	ASCII EBCDIC (numeric)
90	132	5A	0101 1010	Z !	
91	133	5B	0101 1011	[$	
92	134	5C	0101 1100	\ *	
93	135	5D	0101 1101	])	
94	136	5E	0101 1110	^ ;	
95	137	5F	0101 1111	_	
96	140	60	0110 0000	` -	
97	141	61	0110 0001	a /	
98	142	62	0110 0010	b	
99	143	63	0110 0011	c	
100	144	64	0110 0100	d	
101	145	65	0110 0101	e	
102	146	66	0110 0110	f	
103	147	67	0110 0111	g	
104	150	68	0110 1000	h	
105	151	69	0110 1001	i	
106	152	6A	0110 1010	j |	
107	153	6B	0110 1011	k ,	
108	154	6C	0110 1100	l %	
109	155	6D	0110 1101	m _	
110	156	6E	0110 1110	n >	
111	157	6F	0110 1111	o ?	
112	160	70	0111 0000	p	
113	161	71	0111 0001	q	
114	162	72	0111 0010	r	
115	163	73	0111 0011	s	
116	164	74	0111 0100	t	
117	165	75	0111 0101	u	
118	166	76	0111 0110	v	
119	167	77	0111 0111	w	
120	170	78	0111 1000	x	
121	171	79	0111 1001	y	
122	172	7A	0111 1010	z :	
123	173	7B	0111 1011	{ #	+0
124	174	7C	0111 1100	| @	
125	175	7D	0111 1101	} '	-0
126	176	7E	0111 1110	=	
127	177	7F	0111 1111	DEL "	
128	200	80	1000 0000		
129	201	81	1000 0001	a	
130	202	82	1000 0010	b	
131	203	83	1000 0011	c	
132	204	84	1000 0100	d	
133	205	85	1000 0101	e	
134	206	86	1000 0110	f	

Table C-1. ASCII and EBCDIC Character Sets (continued)

Dec	Oct	Hex	Binary	ASCII EBCDIC (alpha/cntrl)	ASCII EBCDIC (numeric)
135	207	87	1000 0111	g	
136	210	88	1000 1000	h	
137	211	89	1000 1001	i	
138	212	8A	1000 1010		
139	213	8B	1000 1011		
140	214	8C	1000 1100		
141	215	8D	1000 1101		
142	216	8E	1000 1110		
143	217	8F	1000 1111		
144	220	90	1001 0000		
145	221	91	1001 0001		
146	222	92	1001 0010	j	
147	223	93	1001 0011	k	
148	224	94	1001 0100	l	
149	225	95	1001 0101	m	
150	226	96	1001 0110	n	
151	227	97	1001 0111	o	
152	230	98	1001 1000	p	
153	231	99	1001 1001	q	
154	232	9A	1001 1010	r	
155	233	9B	1001 1011		
156	234	9C	1001 1100		
157	235	9D	1001 1101		
158	236	9E	1001 1110		
159	237	9F	1001 1111		
160	240	A0	1010 0000		
161	241	A1	1010 0001		
162	242	A2	1010 0010		
163	243	A3	1010 0011	~	
164	244	A4	1010 0100	s	
165	245	A5	1010 0101	t	
166	246	A6	1010 0110	u	
167	247	A7	1010 0111	v	
168	250	A8	1010 1000	w	
169	251	A9	1010 1001	x	
170	252	AA	1010 1010	y	
171	253	AB	1010 1011	z	
172	254	AC	1010 1100		
173	255	AD	1010 1101		
174	256	AE	1010 1110		
175	257	AF	1010 1111		
176	260	B0	1011 0000		
177	261	B1	1011 0001		
178	262	B2	1011 0010		
179	263	B3	1011 0011		

Table C-1. ASCII and EBCDIC Character Sets (continued)

Dec	Oct	Hex	Binary	ASCII EBCDIC (alpha/cntrl)	ASCII EBCDIC (numeric)
180	264	B4	1011 0100		
181	265	B5	1011 0101		
182	266	B6	1011 0110		
183	267	B7	1011 1111		
184	270	B8	1011 1000		
185	271	B9	1011 1001		
186	272	BA	1011 1010		
187	273	BB	1011 1011		
188	274	BC	1011 1100		
189	275	BD	1011 1101		
190	276	BE	1011 1110		
191	277	BF	1011 1111		
192	300	C0	1100 0000	{	+0
193	301	C1	1100 0001	A	+1
194	302	C2	1100 0010	B	+2
195	303	C3	1100 0011	C	+3
196	304	C4	1100 0100	D	+4
197	305	C5	1100 0101	E	+5
198	306	C6	1100 0110	F	+6
199	307	C7	1100 0111	G	+7
200	310	C8	1100 1000	H	+8
201	311	C9	1100 1001	I	+9
202	312	CA	1100 1010		
203	313	CB	1100 1011		
204	314	CC	1100 1100		
205	315	CD	1100 1101		
206	316	CE	1100 1110		
207	317	CF	1100 1111		
208	320	D0	1101 0000	}	-0
209	321	D1	1101 0001	J	-1
210	322	D2	1101 0010	K	-2
211	323	D3	1101 0011	L	-3
212	324	D4	1101 0100	M	-4
213	325	D5	1101 0101	N	-5
214	326	D6	1101 0110	O	-6
215	327	D7	1101 0111	P	-7
216	330	D8	1101 1000	Q	-8
217	331	D9	1101 1001	R	-9
218	332	DA	1101 1010		
219	333	DB	1101 1011		
220	334	DC	1101 1100		
221	335	DD	1101 1101		
222	336	DE	1101 1110		
223	337	DF	1101 1111		
224	340	E0	1110 0000	\	

Table C-1. ASCII and EBCDIC Character Sets (continued)

Dec	Oct	Hex	Binary	ASCII EBCDIC (alpha/cntrl)	ASCII EBCDIC (numeric)
225	341	E1	1110 0001		
226	342	E2	1110 0010	S	
227	343	E3	1110 0011	T	
228	344	E4	1110 0100	U	
229	345	E5	1110 0101	V	
230	346	E6	1110 0110	W	
231	347	E7	1110 0111	X	
232	350	E8	1110 1000	Y	
233	351	E9	1110 1001	Z	
234	352	EA	1110 1010		
235	353	EB	1110 1011		
236	354	EC	1110 1100		
237	355	ED	1110 1101		
238	356	EE	1110 1110		
239	357	EF	1110 1111		
240	360	F0	1111 0000	0	1
241	361	F1	1111 0001	1	2
242	362	F2	1111 0010	2	3
243	363	F3	1111 0011	3	4
244	364	F4	1111 0100	4	5
245	365	F5	1111 0101	5	6
246	366	F6	1111 0110	6	7
247	367	F7	1111 0111	7	8
248	370	F8	1111 1000	8	9
249	371	F9	1111 1001	9	
250	372	FA	1111 1010	|	
251	373	FB	1111 1011		
252	374	FC	1111 1100		
253	375	FD	1111 1101		
254	376	FE	1111 1110		
255	377	FF	1111 1111		

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z