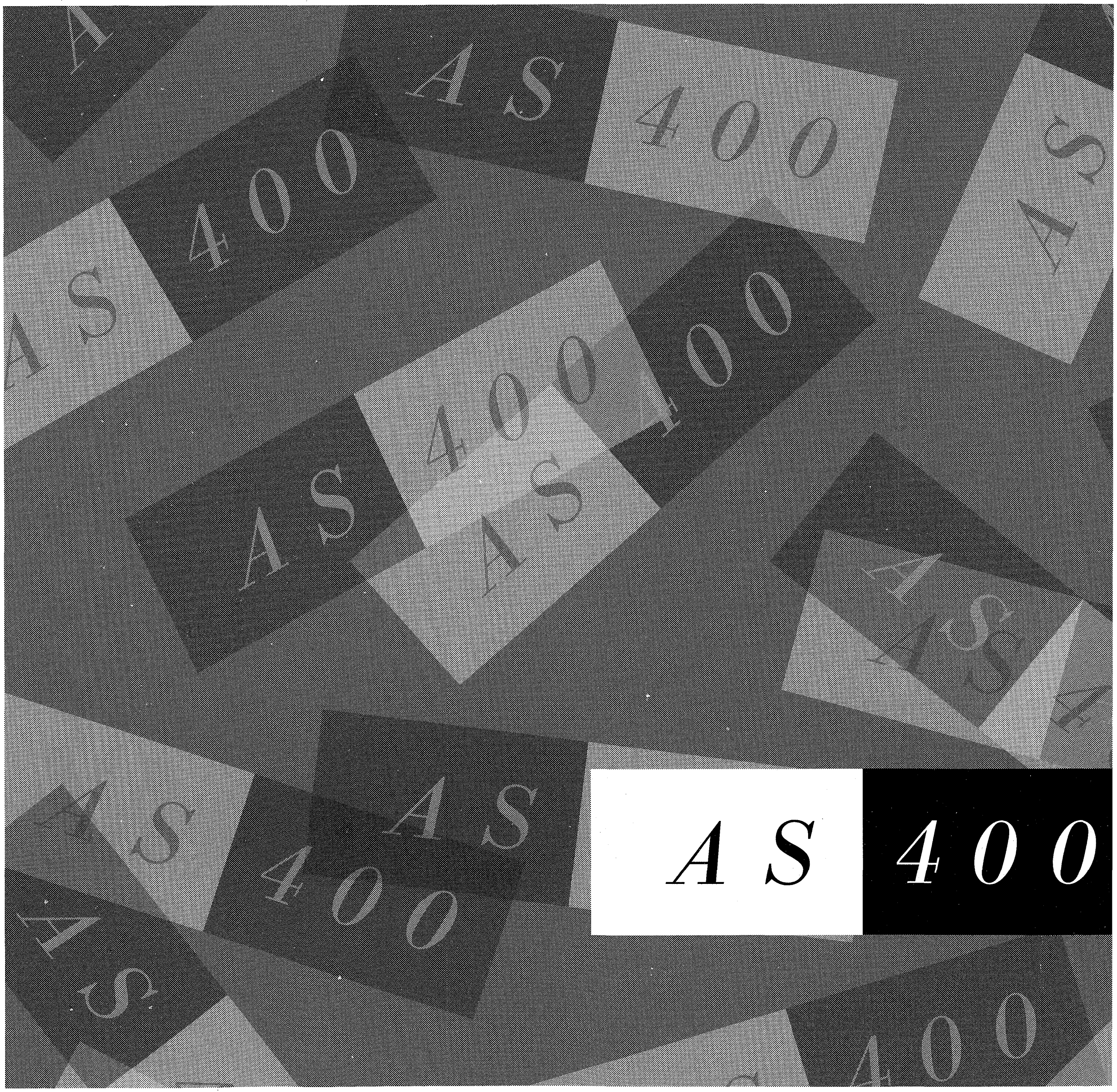
IBM

AS/400 ILE RPG/400 Reference

Version 3



SC09-1526-00

IBM

AS/400

SC09-1526-00

ILE RPG/400 Reference

Version 3

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page xvii.

First Edition (September 1994)

This edition to Version 3 Release 1 Modification 0, of IBM Application System/400 ILE RPG/400 (program 5763-RG1) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory, Information Development
2G/345/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada. M3C 1H7

You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See "Communicating Your Comments to IBM" for a description of the methods. This page immediately precedes the Readers' Comment Form at the back of this publication.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, Connecticut, USA 06904-2501.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Programming Interface Information

This publication is intended to help you create programs using RPG IV source. This publication documents *general-use programming interfaces and associated reference information* provided by the ILE RPG/400 compiler.

General-use programming interfaces allow you to write programs that request or receive services of the ILE RPG/400 compiler.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*), in this publication, are trademarks of the IBM Corporation in the United States or other countries:

Application System/400
IBM
Integrated Language Environment
OS/2
RPG IV
400

AS/400
ILE
Operating System/400
OS/400
RPG/400

About This Reference

This guide provides information about the RPG IV* programming language in the Integrated Language Environment*. (RPG IV* language is an implementation of the ILE RPG/400 language on the AS/400* system with the Operating System/400* (OS/400*) operating system.)

This reference covers:

- The RPG IV character set
- Symbolic names
- Special words
- The RPG IV cycle
- Error handling
- Data types
- The RPG IV specifications
- Built-in functions
- Expressions
- Operation codes

This reference may refer to products that are announced but are not yet available.

You may need to refer to other IBM* guides for more specific information about a particular topic. The *Publications Ordering*, SC41-3000, provides information on all of the guides in the AS/400 library. For a list of related publications, see the bibliography on page 505.

Who Should Use This Reference

This reference is for programmers who are familiar with the RPG IV programming language.

This reference provides a detailed description of the RPG IV language. It does not provide information on how to use the RPG IV compiler or converting RPG III programs to RPG IV. For information on those subjects, see the *ILE RPG/400 Programmer's Guide* SC09-1525.

Before using this reference, you should

- Know how to use applicable AS/400 menus and displays or Control Language (CL) commands.
- Have a firm understanding of ILE as described in detail in the *ILE Concepts*, SC41-3606.

RPG IV Concepts

This section describes some of the basics of RPG IV:

- Symbolic names
- Compiler directives
- The RPG IV program cycle
- Indicators
- Error Handling
- General file considerations

Chapter 1. Symbolic Names and Reserved Words

The valid character set for the RPG IV language consists of:

- The letters A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- RPG IV accepts lowercase letters in symbolic names but translates them to uppercase during compilation
- The numbers 0 1 2 3 4 5 6 7 8 9
- The characters + – * , . ' & / $ # : @ _ > < = ( ) %
- The blank character

Symbolic Names

A symbolic name is a name that uniquely identifies specific data in a program. Its purpose is to allow you to access that data. In the RPG IV language, symbolic names are used for the following:

- Arrays (on page 3)
- Data structures (on page 4)
- Exception output records (on page 4)
- Fields (on page 4)
- Key field lists (on page 4)
- Labels (on page 4)
- Named constants (on page 120)
- Parameter lists (on page 4)
- Record names (on page 4)
- Subroutines (on page 5)
- Tables (on page 5).

The following rules apply to all symbolic names except for deviations noted in the description of each symbolic name:

- The first character of the name must be alphabetic. This includes the characters $, #, and @.
- The remaining characters must be alphabetic or numeric. This includes the underscore (_).
- The name must be left-adjusted in the entry on the specification form except in fields which allow the name to float (definition specification, keyword fields, and the extended factor 2 field).
- A symbolic name cannot be an RPG IV reserved word.
- A symbolic name can be 1 to 10 characters.
- A symbolic name must be unique.

Array Names

The following additional rule applies to array names:

- An array name cannot begin with the letters TAB.

Data Structure Names

A data structure is an area in storage and is considered to be a character field. The following additional rule applies to data structure names:

- A name can be defined as a data structure only once.

EXCEPT Names

An EXCEPT name is a symbolic name assigned to an exception output record. The following additional rule applies to EXCEPT names:

- The same EXCEPT name can be assigned to more than one output record.

Field Names

The following additional rules apply to field names:

- A field name can be defined more than once if each definition using that name has the same data type, the same length, and the same number of decimal positions. All definitions using the same name refer to a single field (that is, the same area in storage). However, it can be defined only once on the definition specification.
- A field can be defined as a data structure subfield only once.
- A subfield name cannot be specified as the result field on an *ENTRY PLIST parameter.

KLIST Names

A KLIST name is a symbolic name assigned to a list of key fields.

Labels

A label is a symbolic name that identifies a specific location in a program (for example, the name assigned to a TAG or ENDSR operation).

Named Constants

A named constant is a symbolic name assigned to a constant.

PLIST Names

A PLIST name is a symbolic name assigned to a list of parameters.

Record Names

A record name is a symbolic name assigned to a record format in an externally described file. The following additional rules apply to record names in an RPG IV program:

- A record name can exist in only one file in the program.

Note: See “RENAME(Ext_format:Int_format)” on page 195 for information on how to overcome this limitation.

Subroutine Names

The name is defined in factor 1 of the BEGSR (begin subroutine) operation.

Table Names

The following additional rules apply to table names:

- A table name can contain from 3 to 10 characters.
- A table name must begin with the letters TAB.

RPG IV Words with Special Functions/Reserved Words

The following RPG IV reserved words have special functions within a program:

- The following reserved words allow you to access the job date, or a portion of it, to be used in the program.

  UDATE
  *DATE
  UMONTH
  *MONTH
  UYEAR
  *YEAR
  UDAY
  *DAY

- The following reserved words can be used for numbering the pages of a report, for record sequence numbering, or to sequentially number output fields.

  PAGE
  PAGE1-PAGE7

- Figurative constants are implied literals that allow specifications without referring to length.

  *BLANK/*BLANKS
  *ZERO/*ZEROS
  *HIVAL
  *LOVAL
  *NULL
  *ON
  *OFF
  *ALLX'x1..'
  *ALLG'oK1K2i'
  *ALL'X..'

- The following reserved words allow indicators to be referred to as data.

  *IN
  *INxx

- Special words used with date and time

  *DMY
  *EUR
  *HMS
  *ISO
  *JIS
  *JUL

*MDY
*YMD
*USA

- Special words used with translation.

  *ALTSEQ
  *EQUATE
  *FILE
  *FTRANS

- *PLACE allows repetitive placement of fields in an output record. (See "*PLACE" on page 251 for more information.)
- *ALL allows all fields that are defined for an externally described file to be written on output. (See "Rules for Figurative Constants" for more information on *ALL)
- Special words used with Built-in Functions

  *ALL
  *NULL

- Special words used with parameter passing

  *OMIT

User Date Special Words

The user date special words (UDATE, *DATE, UMONTH, *MONTH, UDAY, *DAY, UYEAR, *YEAR) allow the programmer to supply a date for the program at run time. The user date special words access the job date that is specified in the job description. The user dates can be written out at output time; UDATE and *DATE can be written out using the Y edit code in the format specified by the control specification. (For a description of the job date, see the &wrkmgmtl..)

Rules for User Date

Remember the following rules when using the user date:

- UDATE, when specified in positions 30 through 43 of the output specifications, prints a 6-character numeric date field. *DATE, when similarly specified, prints an 8-character (4-digit year portion) numeric date field. These special words can be used in three different date formats:

  Month/day/year
  Year/month/day
  Day/month/year

  Use the DATEDIT keyword on the control specification to specify the the editing to be done. If this keywords are not specified, the default is *MDY.

- For an interactive program, the user date special words are set when the job starts running. For a batch program, they are set when the job is sent to the job queue. In neither case are they updated when the program runs over midnight or when the job date changes. Use the TIME operation code to obtain the time and date while the program is running.
- UMONTH, *MONTH, UDAY, *DAY, and UYEAR when specified in positions 30 through 43 of the output specifications, print a 2-position numeric date field. *YEAR can be used to print a 4-position numeric date field. Use UMONTH or

*MONTH to print the month only, UDAY or *DAY to print the day only, and UYEAR or *YEAR to print the year only.

- UDATE and *DATE can be edited when they are written if the Y edit code is specified in position 44 of the output specifications. The "DATEDIT(fmt{separator})" keyword on the control specification determines the format and the separator character to be inserted; for example, 12/31/88, 31.12.88., 12/31/1988.
- UMONTH, *MONTH, UDAY, *DAY, UYEAR and *YEAR cannot be edited by the Y edit code in position 44 of the output specifications.
- The user date fields cannot be modified. This means they cannot be used:
  - In the result field of calculations
  - As factor 1 of PARM operations
  - As the factor 2 index of LOOKUP operations
  - With blank after in output specifications
  - As input fields
- The user date special words can be used in factor 1 or factor 2 of the calculation specifications for operation codes that use numeric fields.
- User date fields are not data data type fields but are numeric fields.

PAGE, PAGE1-PAGE7

PAGE is used to number the pages of a report, to serially number the output records in a file, or to sequentially number output fields. It does not cause a page eject.

The eight possible PAGE fields (PAGE, PAGE1, PAGE2, PAGE3, PAGE4, PAGE5, PAGE6, and PAGE7) may be needed for numbering different types of output pages or for numbering pages for different printer files.

PAGE fields can be specified in positions 30 through 43 of the output specifications or in the input or calculation specifications.

Rules for PAGE, PAGE1-PAGE7

Remember the following rules when using the PAGE fields:

- When a PAGE field is specified in the output specifications, without being defined elsewhere, it is assumed to be a four-digit, numeric field with zero decimal positions.
- Page numbering, unless otherwise specified, starts with 0001; and 1 is automatically added for each new page.
- To start at a page number other than 1, set the value of the PAGE field to one less than the starting page number. For example, if numbering starts with 24, enter a 23 in the PAGE field. The PAGE field can be of any length but must have zero decimal positions (see Figure 1 on page 8).
- Page numbering can be restarted at any point in a job. The following methods can be used to reset the PAGE field:
  - Specify blank-after (position 45 of the output specifications).
  - Specify the PAGE field as the result field of an operation in the calculation specifications.

- Specify an output indicator in the output field specifications (see Figure 2). When the output indicator is on, the PAGE field will be reset to 1. Output indicators cannot be used to control the printing of a PAGE field, because a PAGE field is always written.
- Specify the PAGE field as an input field as shown in Figure 1.

- Leading zeros are automatically suppressed (Z edit code is assumed) when a PAGE field is printed unless an edit code, edit word, or data format (P/B/L/R in position 52) has been specified. Editing and the data format override the suppression of leading zeros. When the PAGE field is defined in input and calculation specifications, it is treated as a field name in the output specifications and zero suppression is not automatic.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC...............................
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IINPUT     PG  50   1  CP
I                                  2    5 0PAGE
```

Figure 1. Page Record Description

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O*  When indicator 15 is on, the PAGE field is set to zero and 1 is
O*  added before the field is printed.  When indicator 15 is off, 1
O*  is added to the contents of the PAGE field before it is printed.
OPRINT     H    L1                       01
O               15      PAGE          1     75
```

Figure 2. Resetting the PAGE Fields to Zero

Chapter 2. Compiler Directives

The compiler directive statements /TITLE, /EJECT, /SPACE, and /COPY allow you to specify heading information for the compiler listing, to control the spacing of the compiler listing, and to insert records from other file members during a compile. The compiler directive statements must precede any compile-time arrays or tables records, translation records, and alternate collating sequence records (that is, ** records).

/TITLE (Positions 7-12)

Use the compiler directive /TITLE to specify heading information (such as security classification or titles) that is to appear at the top of each page of the compiler listing. The following entries are used for /TITLE:

Positions	Entry
7-12	/TITLE
13	Blank
14-100	Title information

A program can contain more than one /TITLE statement. Each /TITLE statement provides heading information for the compiler listing until another /TITLE statement is encountered. A /TITLE statement must be the first RPG IV specification encountered to print information on the first page of the compiler listing. The information specified by the /TITLE statement is printed in addition to compiler heading information.

The /TITLE statement causes a skip to the next page before the title is printed. The /TITLE statement is not printed on the compiler listing.

/EJECT (Positions 7-12)

Positions	Entry
7-12	/EJECT
13-49	Blank
50-100	Comments

Enter /EJECT in positions 7 through 12 to indicate that subsequent specifications are to begin on a new page of the compiler listing. Positions 13 through 49 of the /EJECT statement must be blank. The remaining positions may be used for comments. If the spool file is already at the top of a new page, /EJECT will not advance to a new page. /EJECT is not printed on the compiler listing.

/SPACE (Positions 7-12)

Use the compiler directive /SPACE to control line spacing within the source section of the compiler listing. The following entries are used for /SPACE:

Positions	Entry
7-12	/SPACE
13	Blank

14-16	A positive integer value from 1 through 112 that defines the number of lines to space on the compiler listing. The number must be left-adjusted.
17-49	Blank
50-100	Comments

If the number specified in positions 14 through 16 is greater 112, 112 will be used as the /SPACE value. If the number specified in positions 14 through 16 is greater than the number of lines remaining on the current page, subsequent specifications begin at the top of the next page.

/SPACE is not printed on the compiler listing, but is replaced by the specified line spacing. The line spacing caused by /SPACE is in addition to the two lines that are skipped between specification types.

/COPY (Positions 7-11)

The /COPY compiler directive causes records from other files to be inserted, at the point where the /COPY occurs, with the file being compiled. The inserted files may contain any valid specification except /COPY.

The /COPY statement is entered in the following way:

Positions	**Entry**
7-11	/COPY
12	Blank
13-49	Identifies the location of the member to be copied (merged). The format is: libraryname/filename,membername (RPG IV AS/400 environment) • A member name must be specified. • If a file name is not specified, QRPGLESRC is assumed. • If a library is not specified, the library list is searched for the file. All occurrences of the specified source file in the library list are searched for the member until it is located or the search is complete. • If a library is specified, a file name must also be specified.
50-100	Comments

Figure 3 shows some examples of the /COPY directive statement.

```
C/COPY MBR1 1

I/COPY SRCFIL,MBR2 2

O/COPY SRCLIB/SRCFIL,MBR3 3

O/COPY "SRCLIB!"/"SRC>3","MBR¬3" 4
```

Figure 3. Examples of the /COPY Compiler Directive Statement

1 Copies from member MBR1 in source file QRPGLESRC. The current library list is used to search for file QRPGLESRC.

2 Copies from member MBR2 in file SRCFIL. The current library list is used to search for file SRCFIL. Note that the comma is used to separate the file name from the member name.

3 Copies from member MBR3 in file SRCFIL in library SRCLIB.
4 Copies from member "MBR¬3" in file "SRC>3" in library "SRCLIB!"

Results of the /COPY during Compile

During compilation, the specified file members are merged into the program at the point where the /COPY statement occurs. All /COPY members will appear in the COPY member table.

Chapter 3. Program Cycle

The RPG IV compiler supplies part of the logic for an RPG IV program. The logic the compiler supplies is called the *program cycle* or *logic cycle*. The program cycle is a series of ordered steps that the program goes through for each record read.

The information that you code on RPG IV specifications in your source program need not explicitly specify when records should be read or written. The RPG IV compiler can supply the logical order for these operations when your source program is compiled. Depending on the specifications you code, your program may or may not use each step in the cycle.

Primary (identified by a P in position 18 of the file description specifications) and secondary (identified by an S in position 18 of the file description specifications) files indicate input is controlled by the program cycle. A full procedural file (identified by an F in position 18 of the file description specifications) indicates that input is controlled by program-specified calculation operations (for example, READ and CHAIN).

A program can consist of:

- One primary file and, optionally, one or more secondary files
- Only full procedural files
- A combination of one primary file, optional secondary files, and one or more full procedural files in which some of the input is controlled by the cycle, and other input is controlled by the program.
- No files (for example, input can come from a parameter list or a data area data structure).

General RPG IV Program Cycle

Figure 4 on page 14 shows the specific steps in the general flow of the RPG IV program cycle. A program cycle begins with step 1 and continues through step 7, then begins again with step 1.

The first and last time a program goes through the RPG IV cycle differ somewhat from the normal cycle. Before the first record is read the first time through the cycle, the program resolves any parameters passed to it, writes the records conditioned by the 1P (first page) indicator, does file and data initialization, and processes any heading or detail output operations having no conditioning indicators or all negative conditioning indicators. For example, heading lines printed before the first record is read might consist of constant or page heading information or fields for reserved words, such as PAGE and *DATE. In addition, the program bypasses total calculations and total output steps on the first cycle.

During the last time a program goes through the cycle, when no more records are available, the LR (last record) indicator and L1 through L9 (control level) indicators are set on, and file and data area cleanup is done.

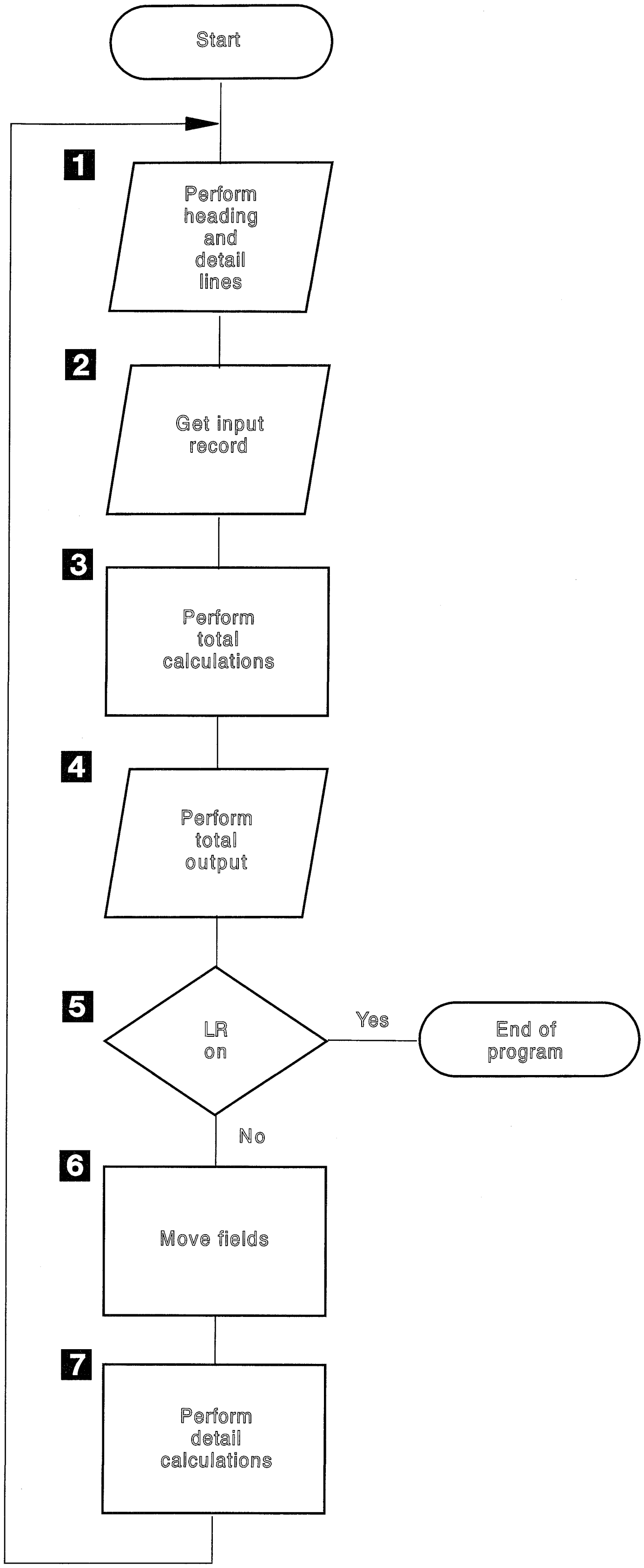

Start
1
Perform heading and detail lines
2
Get input record
3
Perform total calculations
4
Perform total output
5
LR on
Yes
End of program
No
6
Move fields
7
Perform detail calculations

Figure 4. RPG IV Program Logic Cycle

1 All heading and detail lines (H or D in position 17 of the output specifications) are processed.

2 The next input record is read and the record identifying and control level indicators are set on.

3 Total calculations are processed. They are conditioned by an L1 through L9 or LR indicator, or an L0 entry.

4 All total output lines are processed. (identified by a T in position 17 of the output specifications).

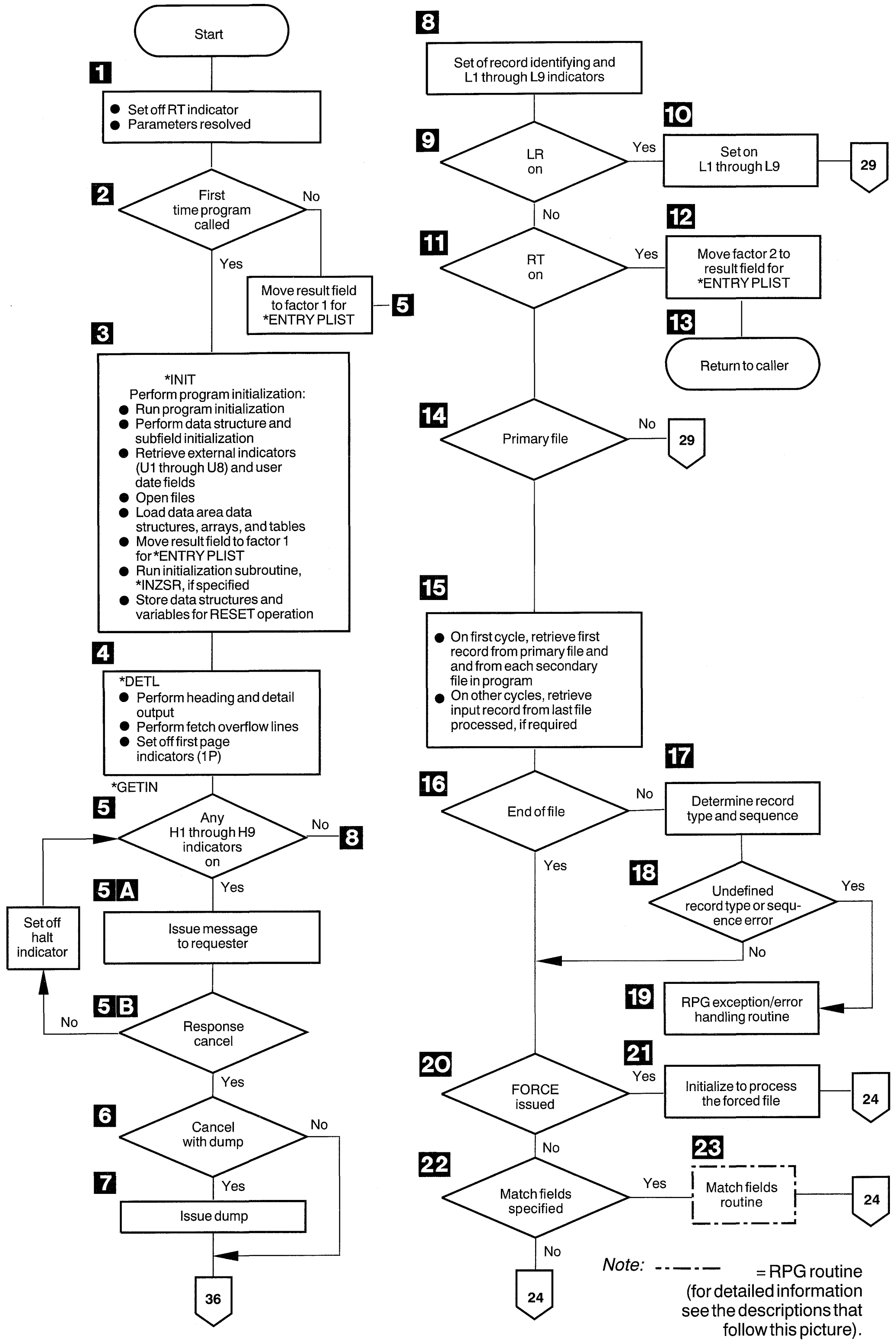
5 It is determined if the LR indicator is on. If it is on, the program is ended.

6 The fields of the selected input records are moved from the record to a processing area. Field indicators are set on.

7 All detail calculations are processed (those not conditioned by control level indicators in positions 7 and 8 of the calculation specifications) on the data from the record read at the beginning of the cycle.

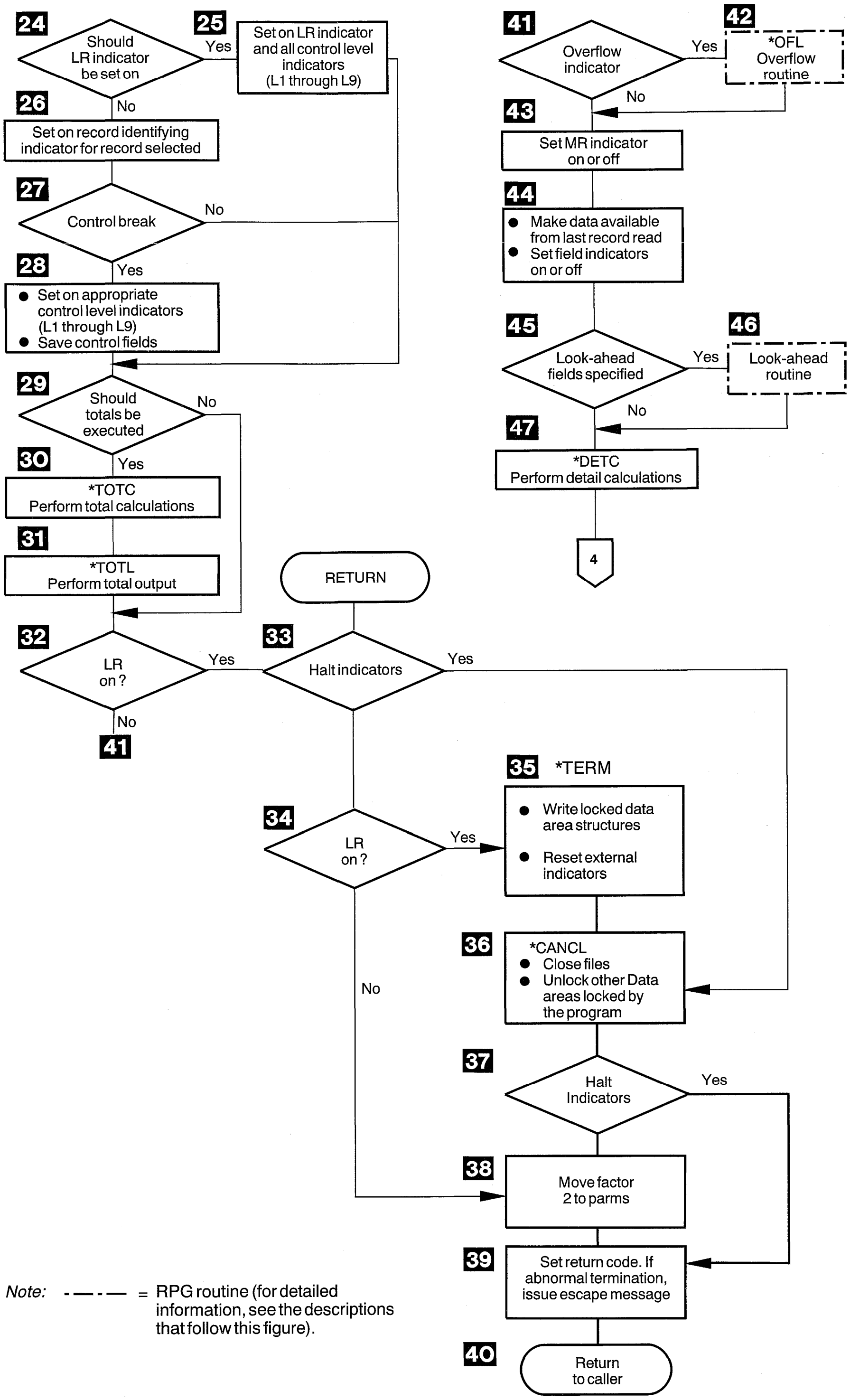
Detailed RPG IV Program Cycle

In “General RPG IV Program Cycle” on page 13, the basic RPG IV Logic Cycle was introduced. The following figures provide a detailed explanation of the RPG IV Logic Cycle.


Start
1
● Set off RT indicator
● Parameters resolved
2
First time program called
No
Yes
Move result field to factor 1 for *ENTRY PLIST
5
3
*INIT
Perform program initialization:
● Run program initialization
● Perform data structure and subfield initialization
● Retrieve external indicators (U1 through U8) and user date fields
● Open files
● Load data area data structures, arrays, and tables
● Move result field to factor 1 for *ENTRY PLIST
● Run initialization subroutine, *INZSR, if specified
● Store data structures and variables for RESET operation
4
*DETL
● Perform heading and detail output
● Perform fetch overflow lines
● Set off first page indicators (1P)
*GETIN
5
Any H1 through H9 indicators on
No
8
Yes
5A
Issue message to requester
Set off halt indicator
5B
No
Response cancel
Yes
6
Cancel with dump
No
7
Yes
Issue dump
36
8
Set of record identifying and L1 through L9 indicators
9
LR on
Yes
10
Set on L1 through L9
29
No
11
RT on
Yes
12
Move factor 2 to result field for *ENTRY PLIST
13
Return to caller
14
Primary file
No
29
15
● On first cycle, retrieve first record from primary file and from each secondary file in program
● On other cycles, retrieve input record from last file processed, if required
16
End of file
No
17
Determine record type and sequence
Yes
18
Undefined record type or sequence error
Yes
No
19
RPG exception/error handling routine
20
FORCE issued
Yes
21
Initialize to process the forced file
24
No
22
Match fields specified
Yes
23
Match fields routine
24
No
24

Note: -.-.- = RPG routine (for detailed information see the descriptions that follow this picture).

Figure 5 (Part 1 of 2). Detailed RPG IV Object Program Cycle


24
Should LR indicator be set on
Yes
25
Set on LR indicator and all control level indicators (L1 through L9)
No
26
Set on record identifying indicator for record selected
27
Control break
No
Yes
28
● Set on appropriate control level indicators (L1 through L9)
● Save control fields
29
Should totals be executed
No
Yes
30
*TOTC Perform total calculations
31
*TOTL Perform total output
32
LR on ?
Yes
No
41
RETURN
33
Halt indicators
Yes
41
Overflow indicator
Yes
42
*OFL Overflow routine
No
43
Set MR indicator on or off
44
● Make data available from last record read
● Set field indicators on or off
45
Look-ahead fields specified
Yes
46
Look-ahead routine
No
47
*DETC Perform detail calculations
4
34
LR on ?
Yes
No
35
*TERM
● Write locked data area structures
● Reset external indicators
36
*CANCL
● Close files
● Unlock other Data areas locked by the program
37
Halt Indicators
Yes
38
Move factor 2 to parms
39
Set return code. If abnormal termination, issue escape message
40
Return to caller

Note: -.—.— = RPG routine (for detailed information, see the descriptions that follow this figure).

Figure 5 (Part 2 of 2). Detailed RPG IV Object Program Cycle

Detailed RPG IV Object Program Cycle

Figure 5 on page 16 shows the specific steps in the detailed flow of the RPG IV program cycle. The item numbers in the following description refer to the numbers in the figure. Routines are flowcharted in Figure 8 on page 26 and in Figure 6 on page 22.

1 The RT indicator is set off. If *ENTRY PLIST is specified the parameters are resolved.

2 RPG IV checks for the first invocation of the program. If it is the first invocation, program initialization continues. If not, it moves the result field to factor 1 in the PARMS statement in *ENTRY PLIST and branches to step 5.

3 The program is initialized at *INIT in the cycle. This process includes: performing data structure and subfield initialization, setting user date fields; opening files; loading all data area data structures, arrays and tables; moving the result field to factor 1 in the PARMS statement in *ENTRY PLIST; running the initialization subroutine *INZSR; and storing the structures and variables for the RESET operation. Files are opened in reverse order of their specification on the File Description Specifications.

4 Heading and detail lines (identified by an H or D in position 17 of the output specifications) are written before the first record is read. Heading and detail lines are always processed at the same time. If conditioning indicators are specified, the proper indicator setting must be satisfied. If fetch overflow logic is specified and the overflow indicator is on, the appropriate overflow lines are written. File translation, if specified, is done for heading and detail lines and overflow output. This step is the return point in the program if factor 2 of an ENDSR operation contains the value *DETL.

5 The halt indicators (H1 through H9) are tested. If all the halt indicators are off, the program branches to step 8. Halt indicators can be set on anytime during the program. This step is the return point in the program if factor 2 of an ENDSR operation contains the value *GETIN.

a. If any halt indicators are on, a message is issued to the user.

b. If the response is to continue, the halt indicator is set off, and the program returns to step 5. If the response is to cancel, the program goes to step 6.

6 If the response is to cancel with a dump, the program goes to step 7; otherwise, the program branches to step 36.

7 The program issues a dump and branches to step 36 (abnormal ending).

8 All record identifying, 1P (first page), and control level (L1 through L9) indicators are set off. All overflow indicators (OA through OG, OV) are set off unless they have been set on during preceding detail calculations or detail output. Any other indicators that are on remain on.

9 If the LR (last record) indicator is on, the program continues with step 10. If it is not on, the program branches to step 11.

10 The appropriate control level (L1 through L9) indicators are set on and the program branches to step 29.

11 If the RT indicator is on, the program continues with step 12; otherwise, the program branches to step 14.

12 Factor 2 is moved to the result field for the parameters of the *ENTRY PLIST.

13 If the RT indicator is on (return code set to 0), the program returns to the caller.

14 If a primary file is present in the program, the program continues with step 15; otherwise, the program branches to step 29.

15 During the first program cycle, the first record from the primary file and from each secondary file in the program is read. File translation is done on the input records. In other program cycles, a record is read from the last file processed. If this file is processed by a record address file, the data in the record address file defines the record to be retrieved. If lookahead fields are specified in the last record processed, the record may already be in storage; therefore, no read may be done at this time.

16 If end of file has occurred on the file just read, the program branches to step 20. Otherwise, the program continues with step 17.

17 If a record has been read from the file, the record type and record sequence (positions 17 through 20 of the input specifications) are determined.

18 It is determined whether the record type is defined in the program, and if the record sequence is correct. If the record type is undefined or the record sequence is incorrect, the program continues with step 19; otherwise, the program branches to step 20.

19 The RPG IV exception/error handling routine receives control.

20 It is determined whether a FORCE operation was processed on the previous cycle. If a FORCE operation was processed, the program selects that file for processing (step 21) and branches around the processing for match fields (steps 22 and 23). The branch is processed because all records processed with a FORCE operation are processed with the matching record (MR) indicator off.

21 If FORCE was issued on the previous cycle, the program selects the forced file for processing after saving any match fields from the file just read. If the file forced is at end of file, normal primary/secondary multifile logic selects the next record for processing and the program branches to step 24.

22 If match fields are specified, the program continues with step 23; otherwise, the program branches to step 24.

23 The match fields routine receives control. (For detailed information on the match fields routine, see "Match Fields Routine" on page 22.)

24 The LR (last record) indicator is set on when all records are processed from the files that have an E specified in position 19 of the file description specifications and all matching secondary records have been processed. If the LR indicator is not set on, processing continues with step 26.

25 The LR (last record) indicator is set on and all control level (L1 through L9) indicators, and processing continues with step 29.

26 The record identifying indicator is set on for the record selected for processing.

27 It is determined whether the record selected for processing caused a control break. A control break occurs when the value in the control fields of the record being processed differs from the value of the control fields of the last record processed. If a control break has not occurred, the program branches to step 29.

28 When a control break occurs, the appropriate control level indicator (L1 through L9) is set on. All lower level control indicators are set on. The program saves the contents of the control fields for the next comparison.

29 It is determined whether the total-time calculations and total-time output should be done. Totals are always processed when the LR indicator is on. If no control level is specified on the input specifications, totals are bypassed on the first cycle and after the first cycle, totals are processed on every cycle. If control levels are specified on the input specifications, totals are bypassed until after the first record containing control fields has been processed.

30 All total calculations conditioned by a control level entry (positions 7 and 8 of the calculation specifications). are processed. This step is the return point in the program if factor 2 of an ENDSR operation contains the value *TOTC.

31 All total output is processed. If fetch overflow logic is specified and the overflow indicator (OA through OG, OV) associated with the file is on, the overflow lines are written. File translation, if specified, is done for all total output and overflow lines. This step is the return point in the program if factor 2 of an ENDSR operation contains the value *TOTL.

32 If LR is on, the program continues with step 33; otherwise, the program branches to step 41.

33 The halt indicators (H1 through H9) are tested. If any halt indicators are on, the program branches to step 36 (abnormal ending). If the halt indicators are off, the program continues with step 34. If the RETURN operation code is used in calculations, the program branches to step 33 after processing of that operation.

34 If LR is on, the program continues with step 35. If it is not on, the program branches to step 38.

35 RPG IV program writes all arrays or tables for which the TOFILE keyword has been specified on the definition specification and writes all locked data area data structures. Output arrays and tables are translated, if necessary.

36 All open files are closed. The RPG IV program also unlocks all data areas that have been locked but not unlocked by the program. If factor 2 of an ENDSR operation contains the value *CANCL, this step is the return point.

37 The halt indicators (H1 through H9) are tested. If any halt indicators are on, the program branches to step 39 (abnormal ending). If the halt indicators are off, the program continues with step 38.

38 The factor 2 fields are moved to the result fields on the PARMs of the *ENTRY PLIST.

39 The return code is set. 1 = LR on, 2 = error, 3 = halt.

40 Control is returned to the caller.

Note: Steps 32 through 40 constitute the normal ending routine. For an abnormal ending, steps 34 through 35 are bypassed.

41 It is determined whether any overflow indicators (OA through OG OV) are on. If an overflow indicator is on, the program continues with step 42; otherwise, the program branches to step 43.

42 The overflow routine receives control. (For detailed information on the overflow routine, see “Overflow Routine” on page 23.) This step is the return point in the program if factor 2 of an ENDSR operation contains the value *OFL.

43 The MR indicator is set on and remains on for the complete cycle that processes the matching record if this is a multifile program and if the record to be processed is a matching record. Otherwise, the MR indicator is set off.

44 Data from the last record read is made available for processing. Field indicators are set on, if specified.

45 If lookahead fields are specified, the program continues with step 46; otherwise, the program branches to step 47.

46 The lookahead routine receives control. (For detailed information on the lookahead routine, see “Lookahead Routine” on page 24.)

47 Detail calculations are processed. This step is the return point in the program if factor 2 of an ENDSR operation contains the value *DETC. The program branches to step 4.

Initialization Subroutine

Refer to Figure 5 on page 16 to see a detailed explanation of the RPG IV initialization subroutine.

A specific subroutine that is to be run at program initialization time can be defined by specifying *INZSR in factor 1 of the subroutine's BEGSR operation. Only one subroutine can be defined as an initialization subroutine. It is called at the end of the program initialization step of the program cycle (that is, after data structures and subfields are initialized, external indicators and user data fields are retrieved, files are opened, data area data structures, arrays, and tables are loaded, and PARM result fields moved to factor 1 for *ENTRY PLIST). *INZSR may not be specified as a file/program error/exception subroutine.

If a program ends with LR off, the initialization subroutine does not automatically run during the next invocation of that program because the subroutine is part of the initialization step of the program.

The initialization subroutine is like any other subroutine in the program, other than being called at program initialization time. It may be called using the EXSR or CASxx operations, and it may call other subroutines or other programs. Any operation that is valid in a subroutine is valid in the initialization subroutine, with the exception of the RESET operation. This is because the value used to reset a variable is not defined until after the initialization subroutine is run.

Any changes made to a variable during the initialization subroutine affect the value that the variable is set to on a subsequent RESET operation. Default values can be defined for fields in record formats by, for example, setting them in the initialization subroutine and then using RESET against the record format whenever the default values are to be used. The initialization subroutine can also retrieve information such as the current time for 1P output.

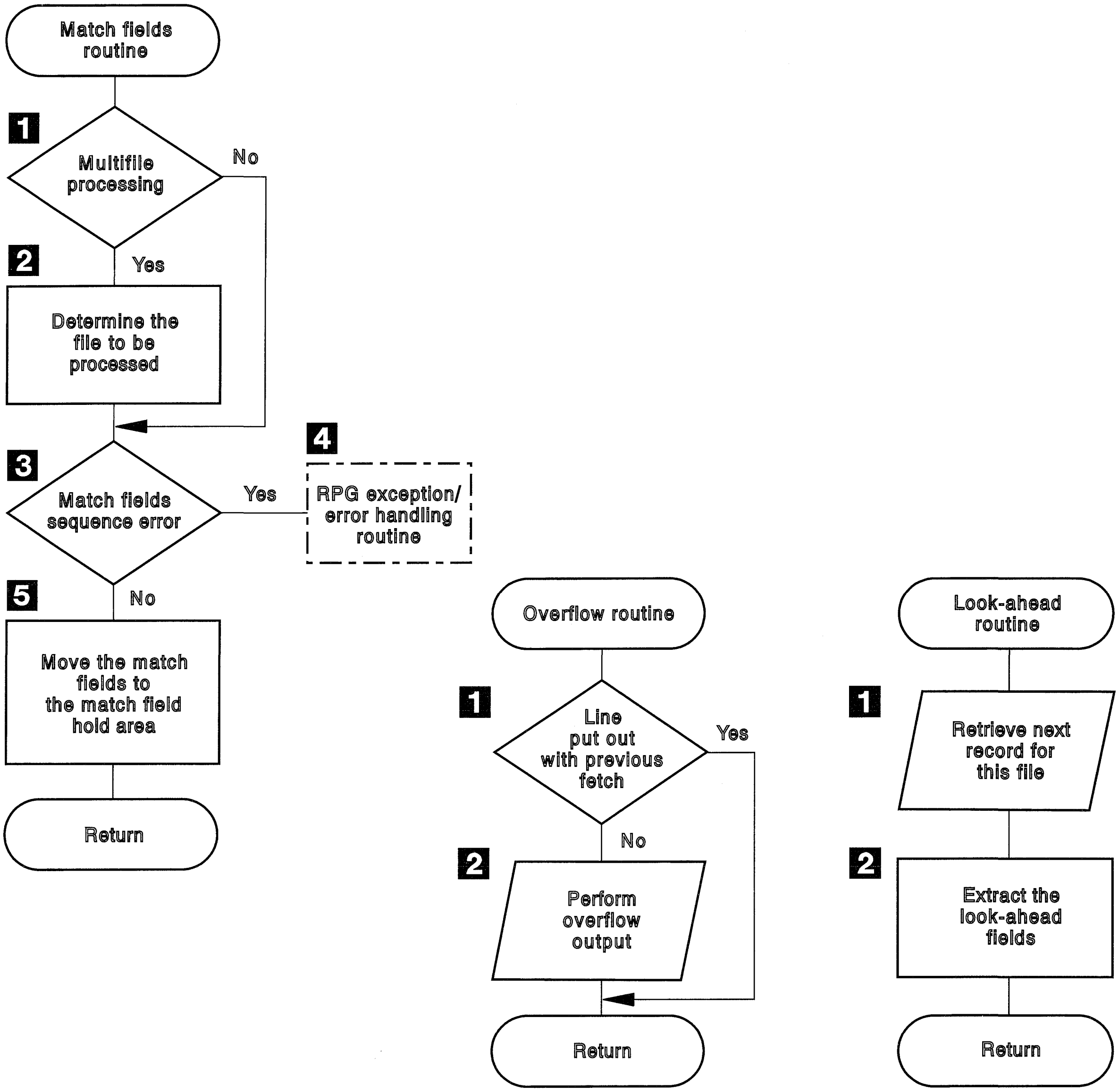

Match fields routine
1
Multifile processing
No
2
Yes
Determine the file to be processed
4
3
Match fields sequence error
Yes
RPG exception/ error handling routine
5
No
Move the match fields to the match field hold area
Return
Overflow routine
1
Line put out with previous fetch
Yes
No
2
Perform overflow output
Return
Look-ahead routine
1
Retrieve next record for this file
2
Extract the look-ahead fields
Return

Figure 6. Detail Flow of RPG IV Match Fields, Overflow, and Lookahead Routines

Match Fields Routine

Figure 6 shows the specific steps in the RPG IV match fields routine. The item numbers in the following descriptions refer to the numbers in the figure.

1 If multifile processing is being used, processing continues with step 2; otherwise, the program branches to step 3.

2 The value of the match fields in the hold area is tested to determine which file is to be processed next.

3 The RPG IV program extracts the match fields from the match files and processes sequence checking. If the match fields are in sequence, the program branches to step 5.

4 If the match fields are not in sequence, the RPG IV exception/error handling routine receives control.

5 The match fields are moved to the hold area for that file. A hold area is provided for each file that has match fields. The next record is selected for processing based on the value in the match fields.

Overflow Routine

Figure 6 on page 22 shows the specific steps in the RPG IV overflow routine. The item numbers in the following descriptions refer to the numbers in the figure.

1 The RPG IV program determines whether the overflow lines were written previously using the fetch overflow logic (step 30 in Figure 5 on page 16). If the overflow lines were written previously, the program branches to the specified return point; otherwise, processing continues with step 2.

2 All output lines conditioned with an overflow indicator are tested and written to the conditioned overflow lines.

The fetch overflow routine allows you to alter the basic RPG IV overflow logic to prevent printing over the perforation and to let you use as much of the page as possible. During the regular program cycle, the RPG IV program checks only once, immediately after total output, to see if the overflow indicator is on. When the fetch overflow function is specified, the RPG IV program checks overflow on each line for which fetch overflow is specified.

Specify fetch overflow with an F in position 18 of the output specifications on any detail, total, or exception lines for a PRINTER file. The fetch overflow routine does not automatically cause forms to advance to the next page.

During output, the conditioning indicators on an output line are tested to determine whether the line is to be written. If the line is to be written and an F is specified in position 18, the RPG IV program tests to determine whether the overflow indicator is on. If the overflow indicator is on, the overflow routine is fetched and the following operations occur:

- Only the overflow lines for the file with the fetch specified are checked for output.
- All total lines conditioned by the overflow indicator are written.
- Forms advance to a new page when a skip to a line number less than the line number the printer is currently on is specified in a line conditioned by an overflow indicator.
- Heading, detail, and exception lines conditioned by the overflow indicator are written.
- The line that fetched the overflow routine is written.
- Any detail and total lines left to be written for that program cycle are written.

Position 18 of each OR line must contain an F if the overflow routine is to be used for each record in the OR relationship. Fetch overflow cannot be used if an overflow indicator is specified in positions 21 through 29 of the same specification line. If this occurs, the overflow routine is not fetched.

Use the fetch overflow routine when there is not enough space left on the page to print the remaining detail, total, exception, and heading lines conditioned by the overflow indicator. To determine when to fetch the overflow routine, study all possible overflow situations. By counting lines and spaces, you can calculate what happens if overflow occurs on each detail, total, and exception line.

Lookahead Routine

Figure 6 shows the specific steps in the RPG IV lookahead routine. The item numbers in the following descriptions refer to the numbers in the figure.

1 The next record for the file being processed is read. However, if the file is a combined or update file (identified by a C or U, respectively, in position 17 of the file description specifications), the lookahead fields from the current record being processed is extracted.

2 The lookahead fields are extracted.

Ending a Program without a Primary File

If your program does not contain a primary file, you *must* specify a way for the program to end:

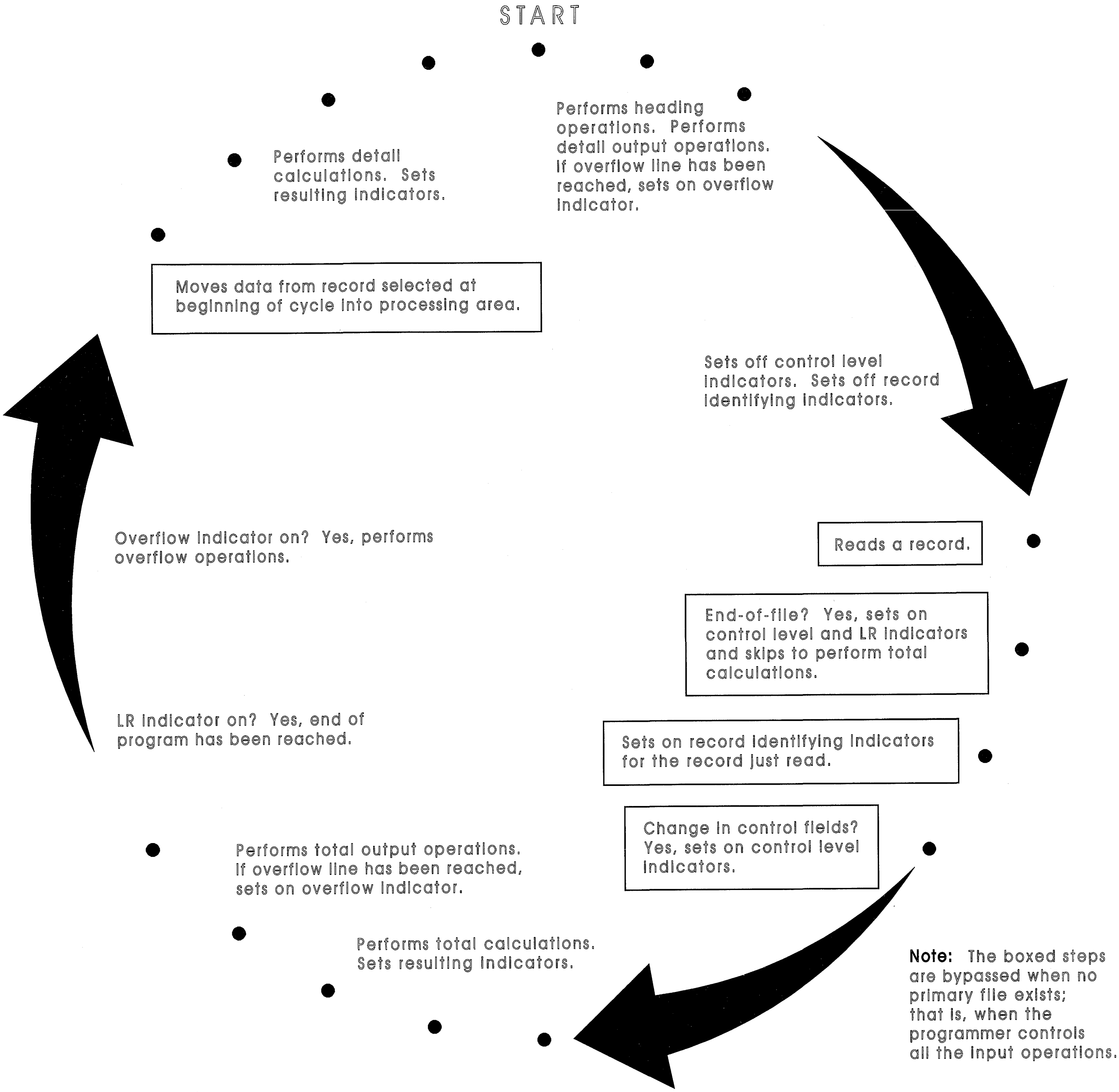
- By setting the LR indicator on
- By setting the RT indicator on
- By setting an H1 through H9 indicator on
- By specifying the RETURN operation code

The LR, RT, H1 through H9 indicators, and the RETURN operation code, can be used in conjunction with each other.

Program Control of File Processing

Specify a full procedural file (F in position 18 of the file description specifications) to control all or partial input of a program. A full procedural file indicates that *input* is controlled by program-specified calculation operations (for example, READ, CHAIN). When both full procedural files and a primary file (P in position 18 of the file description specifications) are specified in a program, some of the input is controlled by the program, and other input is controlled by the cycle. The program cycle exists when a full procedural file is specified; however, file processing occurs at detail or total calculation time for the full procedural file.

The file operation codes can be used for program control of input. These file operation codes are discussed in Chapter 22, "Operation Codes" on page 281.


START
Performs heading operations. Performs detail output operations. If overflow line has been reached, sets on overflow indicator.
Sets off control level indicators. Sets off record identifying indicators.
Reads a record.
End-of-file? Yes, sets on control level and LR indicators and skips to perform total calculations.
Sets on record identifying indicators for the record just read.
Change in control fields? Yes, sets on control level indicators.
Performs total calculations. Sets resulting indicators.
Performs total output operations. If overflow line has been reached, sets on overflow indicator.
LR indicator on? Yes, end of program has been reached.
Overflow indicator on? Yes, performs overflow operations.
Moves data from record selected at beginning of cycle into processing area.
Performs detail calculations. Sets resulting indicators.

Note: The boxed steps are bypassed when no primary file exists; that is, when the programmer controls all the input operations.

Figure 7. Programmer Control of Input Operation within the Program-Cycle

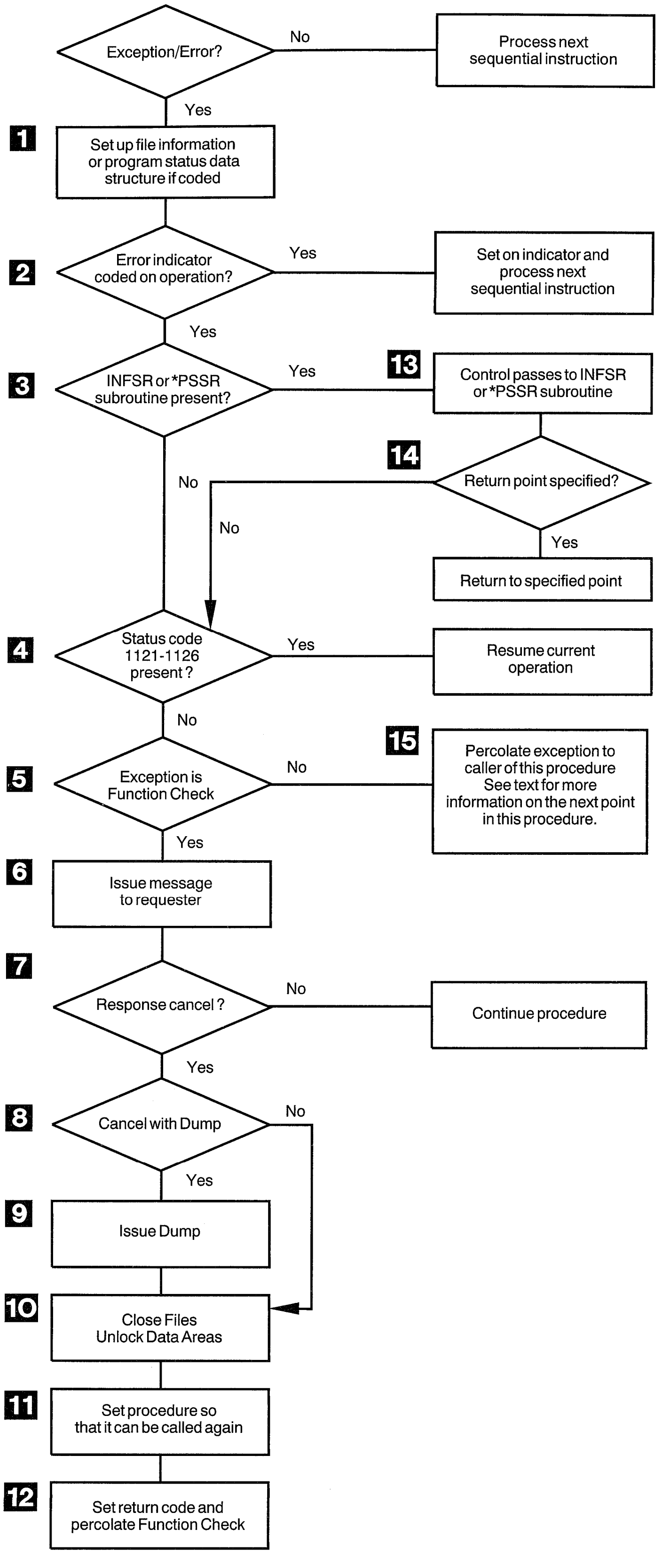

Exception/Error?
No
Process next sequential instruction
Yes
1
Set up file information or program status data structure if coded
2
Error indicator coded on operation?
Yes
Set on indicator and process next sequential instruction
Yes
3
INFSR or *PSSR subroutine present?
Yes
13
Control passes to INFSR or *PSSR subroutine
14
Return point specified?
No
No
Yes
Return to specified point
4
Status code 1121-1126 present ?
Yes
Resume current operation
No
15
Percolate exception to caller of this procedure See text for more information on the next point in this procedure.
5
Exception is Function Check
No
Yes
6
Issue message to requester
7
Response cancel ?
No
Continue procedure
Yes
8
Cancel with Dump
No
Yes
9
Issue Dump
10
Close Files Unlock Data Areas
11
Set procedure so that it can be called again
12
Set return code and percolate Function Check

Figure 8. Detail Flow of RPG IV Exception/Error Handling Routine

RPG IV Exception/Error Handling Routine

Figure 8 on page 26 shows the specific steps in the RPG IV exception/error handling routine. The item numbers in the following description refer to the numbers in the figure.

1 Set up the file information or procedure status data structure, if specified, with status information.

2 If the exception/error occurred on an operation code that has an indicator specified in positions 73 and 74, the indicator is set on, and control returns to the next sequential instruction in the calculations.

3 If the appropriate exception/error subroutine (INFSR or *PSSR) is present in the procedure, the procedure branches to step 13; otherwise, the procedure continues with step 4.

4 If the Status code is 1121-1126 (see "Status Codes" on page 75), control returns to the current instruction in the calculations. If not, the procedure continues with step 5.

5 If the exception is a function check, the procedure continues with step 6. If not, it branches to step 15.

6 An inquiry message is issued to the requester. For an interactive job, the message goes to the requester. For a batch job, the message goes to QSYSOPR. If QSYSOPR is not in break mode, a default response is issued.

7 If the user's response is to cancel the procedure, the procedure continues with step 8. If not, the procedure continues.

8 If the user's response is to cancel with a dump, the procedure continues with step 9. If not, the procedure branches to step 10.

9 A dump is issued.

10 All files are closed and data areas are unlocked

11 The procedure is set so that it can be called again.

12 The return code is set and the function check is percolated.

13 Control passes to the exception/error subroutine (INFSR or *PSSR).

14 If a return point is specified in factor 2 of the ENDSR operation for the exception/error subroutine, the procedure goes to the specified return point. If a return point is not specified, the procedure goes to step 4. If a field name is specified in factor 2 of the ENDSR operation and the content is not one of the RPG IV-defined return points (such as *GETIN or *DETC), the procedure goes to step 6. No error is indicated, and the original error is handled as though the factor 2 entry were blank.

15 If no invocation handles the exception, then it is promoted to function check and the procedure branches to step 5. Otherwise, depending on the action taken by the handler, control resumes in this procedure either at step 10 or at the next machine instruction after the point at which the exception occurred.

Chapter 4. RPG IV Indicators

An indicator is a one byte character field which contains either on ('1') or off ('0'). It is generally used to indicate the result of an operation or to condition (or control) the processing of an operation.

Indicators are defined either by an entry on the specification or by the RPG IV program itself. The positions on the specification in which you define an indicator determine how the indicator is used. An indicator that has been defined can then be used to condition calculation and output operations.

The RPG IV program sets and resets certain indicators at specific times during the program cycle. In addition, the state of most indicators can be changed by calculation operations. All indicators except MR, 1P, KA through KN, and KP through KY can be set on with the SETON operation code; all indicators except MR and 1P can be set off with the SETOFF operation code.

This chapter is divided into the following topics:

- Indicators defined on the RPG IV specifications
- Indicators not defined on the RPG IV specifications
- Using indicators
- Indicators referred to as data.

Indicators Defined on RPG IV Specifications

You can specify the following indicators on the RPG IV specifications:

- Overflow indicator (the OFLIND keyword on the file description specifications).
- Record identifying indicator (positions 21 and 22 of the input specifications).
- Control level indicator (positions 63 and 64 of the input specifications).
- Field indicator (positions 69 through 74 of the input specifications).
- Resulting indicator (positions 71 through 76 of the calculation specifications).
- *IN array, *IN(xx) array element or *INxx field (See “Indicators Referred to As Data” on page 56 for a description of how an indicator is defined when used with one of these reserved words.).

The defined indicator can then be used to condition operations in the program.

Overflow Indicators

An overflow indicator is defined by the OFLIND keyword on the file description specifications. It is set on when the last line on a page has been printed or passed. Valid indicators are *INOA through *INOG, *INOV, and *IN01 through *IN99. A defined overflow indicator can then be used to condition calculation and output operations. A description of the overflow indicator and fetch overflow logic is given in “Overflow Routine” on page 23.

Record Identifying Indicators

A record identifying indicator is defined by an entry in positions 21 and 22 of the input specifications and is set on when the corresponding record type is selected for processing. That indicator can then be used to condition certain calculation and output operations. Record identifying indicators do not have to be assigned in any particular order.

The valid record identifying indicators are:

01-99
H1-H9
L1-L9
LR
U1-U8
RT

For an externally described file, a record identifying indicator is optional, but, if you specify it, it follows the same rules as for a program described file.

Generally, the indicators 01 through 99 are used as record identifying indicators. However, the control level indicators (L1 through L9) and the last record indicator (LR) can be used. If L1 through L9 are specified as record identifying indicators, lower level indicators are not set on.

When you select a record type for processing, the corresponding record identifying indicator is set on. All other record identifying indicators are off except when a file operation code is used at detail and total calculation time to retrieve records from a file (see below). The record identifying indicator is set on after the record is selected, but before the input fields are moved to the input area. The record identifying indicator for the new record is on during total time for the old record; therefore, calculations processed at total time using the fields of the old record cannot be conditioned by the record identifying indicator of the old record. You can set the indicators off at any time in the program cycle; they are set off before the next primary or secondary record is selected.

If you use a file operation code on the calculation specifications to retrieve a record, the record identifying indicator is set on as soon as the record is retrieved from the file. The record identifying indicator is not set off until the appropriate point in the RPG IV cycle. (See Figure 7 on page 25.) Therefore, it is possible to have several record identifying indicators for the same file, as well as record-not-found indicators, set on concurrently if several operations are issued to the same file within the same RPG IV program cycle.

Rules for Assigning Record Identifying Indicators

When you assign record identifying indicators to records in a program described file, remember the following:

- You can assign the same indicator to two or more different record types if the same operation is to be processed on all record types. To do this, you specify the record identifying indicator in positions 21 and 22, and specify the record identification codes for the various record types in an OR relationship.
- You can associate a record identifying indicator with an AND relationship, but it must appear on the first line of the group. Record identifying indicators cannot be specified on AND lines.

- An undefined record (a record in a program described file that was not described by a record identification code in positions 23 through 46) causes the program to halt.
- A record identifying indicator can be specified as a record identifying indicator for another record type, as a field indicator, or as a resulting indicator. No diagnostic message is issued, but this use of indicators may cause erroneous results.

When you assign record identifying indicators to records in an externally described file, remember the following:

- AND/OR relationships cannot be used with record format names; however, the same record identifying indicator can be assigned to more than one record.
- The record format name, rather than the file name, must be specified in positions 7 through 16.

For an example of record identifying indicators, see Figure 9.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
I*
I*Record identifying indicator 01 is set on if the record read
I*contains an S in position 1 or an A in position 1.
IINPUT1    NS  01    1 CS
I         OR         1 CA
I                                  1   25  FLD1
I* Record identifying indicator 02 is set on if the record read
I* contains XYZA in positions 1 through 4.
I          NS  02    1 CX    2 CY    3 CZ
I         AND        4 CA
I                                  1   15  FLDA
I                                 16   20  FLDB
I* Record identifying indicator 95 is set on if any record read
I* does not meet the requirements for record identifying indicators
I* 01 or 02.
I          NS  95
```

Figure 9 (Part 1 of 2). Examples of Record Identifying Indicators

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IRcdname+++....Ri.....................................................
I*
I* For an externally described file, record identifying indicator 10
I* is set on if the ITMREC record is read and record identifying
I* indicator 20 is set on if the SLSREC or COMREC records are read.
IITMREC       10
ISLSREC       20
ICOMREC       20
```

Figure 9 (Part 2 of 2). Examples of Record Identifying Indicators

Control Level Indicators (L1-L9)

A control level indicator is defined by an entry in positions 63 and 64 of the input specifications, designating an input field as a control field. It can then be used to condition calculation and output operations. The valid control level indicator entries are L1 through L9.

A control level indicator designates an input field as a control field. When a control field is read, the data in the control field is compared with the data in the same control field from the previous record. If the data differs, a control break occurs, and the control level indicator assigned to the control field is set on. You can then use control level indicators to condition operations that are to be processed only when all records with the same information in the control field have been read. Because the indicators stay on for both total time and the first detail time, they can also be used to condition total printing (last record of a control group) or detail printing (first record in a control group). Control level indicators are set off before the next record is read.

A control break can occur after the first record containing a control field is read. The control fields in this record are compared to an area in storage that contains hexadecimal zeros. Because fields from two different records are not being compared, total calculations and total output operations are bypassed for this cycle.

Control level indicators are ranked in order of importance with L1 being the lowest and L9 the highest. All lower level indicators are set on when a higher level indicator is set on as the result of a control break. However, the lower level indicators can be used in the program only if they have been defined. For example, if L8 is set on by a control break, L1 through L7 are also set on. The LR (last record) indicator is set on when the input files are at end of file. LR is considered the highest level indicator and forces L1 through L9 to be set on.

You can also define control level indicators as record identifying or resulting indicators. When you use them in this manner, the status of the lower level indicators is not changed when a higher level indicator is set on. For example, if L3 is used as a resulting indicator, the status of L2 and L1 would not change if L3 is set on.

The importance of a control field in relation to other fields determines how you assign control level indicators. For example, data that demands a subtotal should have a lower control level indicator than data that needs a final total. A control field containing department numbers should have a higher control level indicator than a control field containing employee numbers if employees are to be grouped within departments (see Figure 10 on page 34).

Rules for Control Level Indicators

When you assign control level indicators, remember the following:

- You can specify control fields only for primary or secondary files.
- You cannot specify control fields for full procedural files, binary format fields, or look-ahead fields.
- You cannot use control level indicators when an array name is specified in positions 49 through 62 of the input specifications; however, you can use control level indicators with an array element.
- Control level compare operations are processed for records in the order in which they are found, regardless of the file from which they come.
- If you use the same control level indicator in different record types or in different files, the control fields associated with that control level indicator must be the same length (see Figure 10 on page 34) except for date, time, and timestamp fields which need only match in type (that is, they can be different formats).
- The control level indicator field length is the length of a control level indicator in a record. For example, if L1 has a field length of 10 bytes in a record, the control level indicator field length for L1 is 10 positions.

  The control level indicator field length for split control fields is the sum of the lengths of all fields associated with a control level indicator in a record. If L2 has a split control field consisting of 3 fields of length: 12 bytes, 2 bytes and 4 bytes; then the control level indicator field length for L2 is 18 positions.

  If multiple records use the same control level indicator, then the control level indicator field length is the length of only one record, not the sum of all the lengths of the records.

  Within a program, the sum of the control level indicator field lengths of all control level indicators cannot exceed 256 positions.
- Record positions in control fields assigned different control level indicators can overlap in the same record type (see Figure 11 on page 35). For record types that require control or match fields, the total length of the control or match field must be less than or equal to 256. For example, in Figure 11 on page 35, 15 positions have been assigned to control levels.
- Field names are ignored in control level operations. Therefore, fields from different record types that have been assigned the same control level indicator can have the same name.
- Control levels need not be written in any sequence. An L2 entry can appear before L1. All lower level indicators need not be assigned.
- If different record types in a file do not have the same number of control fields, unwanted control breaks can occur.

Figure 12 on page 35 shows an example of how to avoid unwanted control breaks.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
A* EMPLOYEE MASTER FILE -- EMPMSTL
A          R EMPREC                    PFILE(EMPMSTL)
A            EMPLNO         6
A            DEPT           3
A            DIVSON         1
A*
A*                 (ADDITIONAL FIELDS)
A*
A          R EMPTIM                    PFILE(EMPMSTP)
A            EMPLNO         6
A            DEPT           3
A            DIVSON         1
A*
A*                 (ADDITIONAL FIELDS)
```

Figure 10 (Part 1 of 2). Control Level Indicators (Two Record Types)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
I*
I*  In this example, control level indicators are defined for three
I*  fields. The names of the control fields (DIVSON, DEPT, EMPLNO)
I*  give an indication of their relative importance.
I*  The division (DIVSON) is the most important group.
I*  It is given the highest control level indicator used (L3).
I*  The department (DEPT) ranks below the division;
I*  L2 is assigned to it.  The employee field (EMPLNO) has
I*  the lowest control level indicator (L1) assigned to it.
I*
IEMPREC        10
I                                          EMPLNO        L1
I                                          DIVSON        L3
I                                          DEPT          L2
I*
I*  The same control level indicators can be used for different record
I*  types.  However, the control fields having the same indicators must
I*  be the same length.  For records in an externally described file,
I*  the field attributes are defined in the external description.
I*
IEMPTIM        20
I                                          EMPLNO        L1
I                                          DEPT          L2
I                                          DIVSON        L3
```

Figure 10 (Part 2 of 2). Control Level Indicators (Two Record Types)

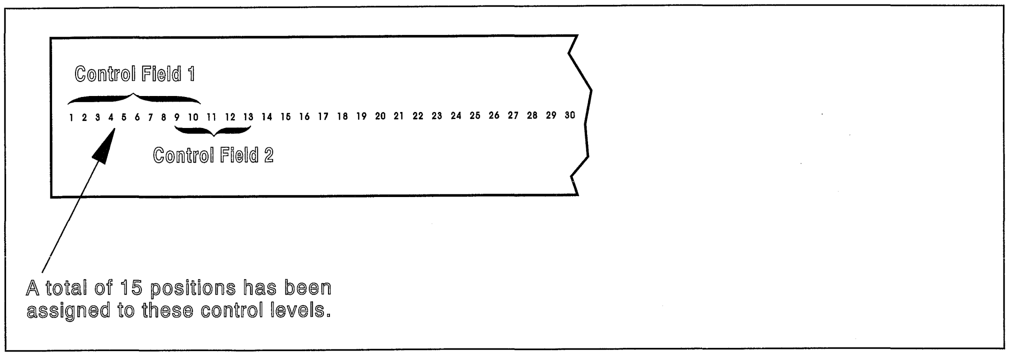


Figure 11. Overlapping Control Fields

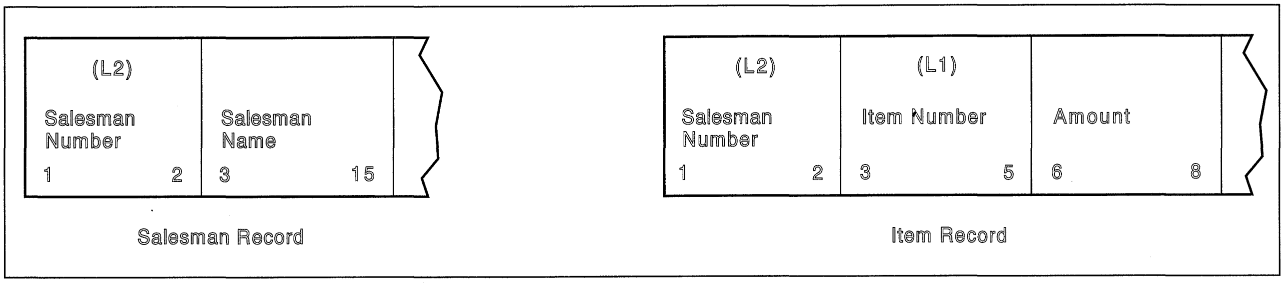


Figure 12 (Part 1 of 4). How to Avoid Unwanted Control Breaks

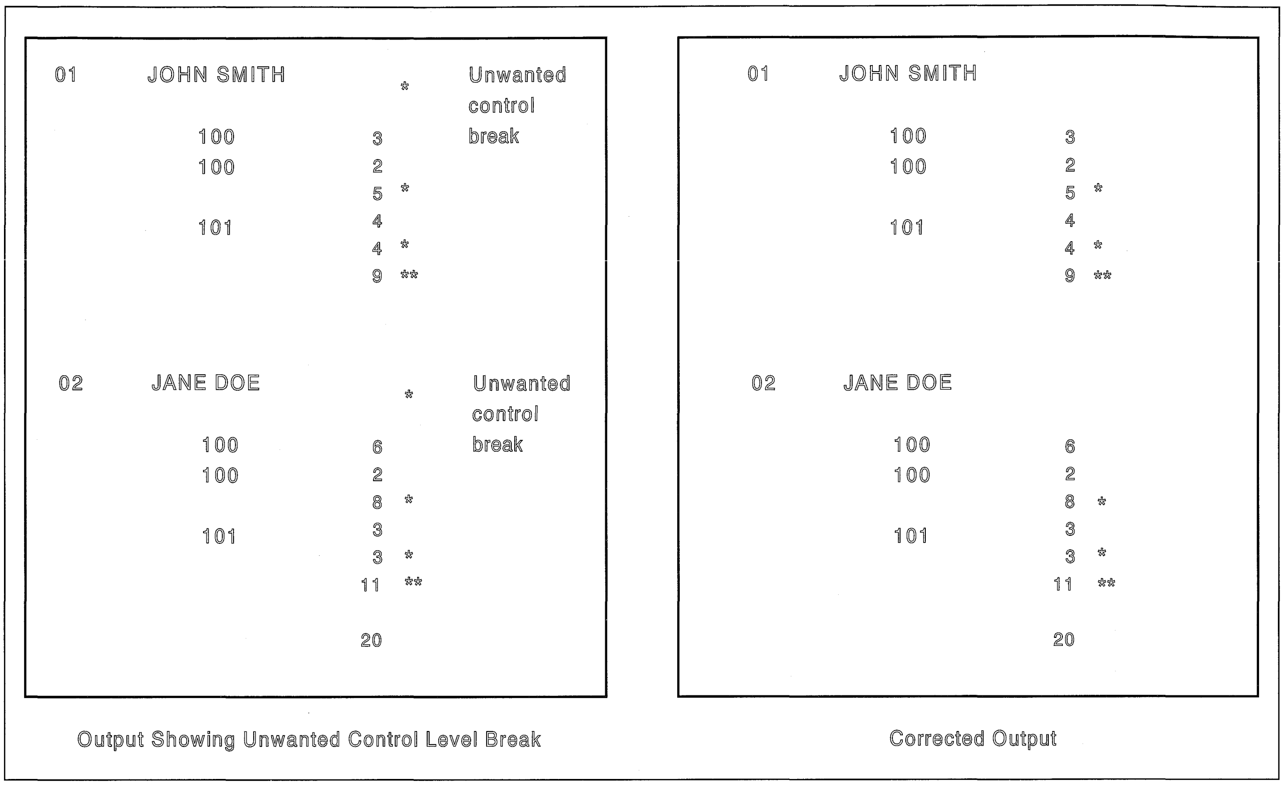
```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
ISALES         01
I                                  1    2  L2FLD         L2
I                                  3   15  NAME
IITEM          02
I                                  1    2  L2FLD         L2
I                                  3    5  L1FLD         L1
I                                  6    8  AMT
```

Figure 12 (Part 2 of 4). How to Avoid Unwanted Control Breaks

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*  Indicator 11 is set on when the salesman record is read.
C*
C   01              SETON                                        11
C*
C*  Indicator 11 is set off when the item record is read.
C*  This allows the normal L1 control break to occur.
C*
C   02              SETOFF                                       11
C   02AMT           ADD       L1TOT         L1TOT             5 0
CL1   L1TOT         ADD       L2TOT         L2TOT             5 0
CL2   L2TOT         ADD       LRTOT         LRTOT             5 0
C*

*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
OPRINTER   D    01                  1  1
O                       L2FLD                5
O                       NAME                25
O          D    02                     1
O                       L1FLD               15
O                       AMT           Z     15
O*
O*  When the next item record causes an L1 control break, no total
O*  output is printed if indicator 11 is on.  Detail calculations
O*  are then processed for the item record.
O*
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O          T    L1N11               1
O                     L1TOT           ZB    25
O                                           27 '*'
O          T    L2                  1
O                     L2TOT           ZB    25
O                                           28 '**'
O          T    LR                  1
O                     LRTOT           ZB    25
```

Figure 12 (Part 3 of 4). How to Avoid Unwanted Control Breaks



```
01      JOHN SMITH          *    Unwanted
                                 control
            100         3        break
            100         2
                        5  *
            101         4
                        4  *
                        9  **



02      JANE DOE            *    Unwanted
                                 control
            100         6        break
            100         2
                        8  *
            101         3
                        3  *
                       11  **

                       20
```

Output Showing Unwanted Control Level Break

```
01      JOHN SMITH

            100         3
            100         2
                        5  *
            101         4
                        4  *
                        9  **



02      JANE DOE

            100         6
            100         2
                        8  *
            101         3
                        3  *
                       11  **

                       20
```

Corrected Output

Figure 12 (Part 4 of 4). How to Avoid Unwanted Control Breaks

Different record types normally contain the same number of control fields. However, some applications require a different number of control fields in some records.

The salesman records contain only the L2 control field. The item records contain both L1 and L2 control fields. With normal RPG IV coding, an unwanted control break is created by the first item record following the salesman record. This is recognized by an L1 control break immediately following the salesman record and results in an asterisk being printed on the line below the salesman record.

- Numeric control fields are compared in zoned decimal format. Packed numeric input fields lengths can be determined by the formula:

  $$d = 2n - 1$$

  Where d = number of digits in the field and n = length of the input field. The number of digits in a packed numeric field is always odd; therefore, when a packed numeric field is compared with a zoned decimal numeric field, the zoned field must have an odd length.
- When numeric control fields with decimal positions are compared to determine whether a control break has occurred, they are always treated as if they had no decimal positions. For instance, 3.46 is considered equal to 346.
- If you specify a field as numeric, only the positive numeric value determines whether a control break has occurred; that is, a field is always considered to be positive. For example, -5 is considered equal to +5.
- Date and time fields are converted to *ISO format before being compared
- Graphic data is compared by hexadecimal value

Split Control Field

A split control field is formed when you assign more than one field in an input record the same control level indicator. For a program described file, the fields that have the same control level indicator are combined by the program in the order specified in the input specifications and treated as a single control field (see Figure 13). The first field defined is placed in the high-order (leftmost) position of the control field, and the last field defined is placed in the low-order (rightmost) position of the control field.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IMASTER        01
I                                 28   31  CUSNO         L4
I                                 15   20  ACCTNO        L4
I                                 50   52  REGNO         L4
```

Figure 13. Split Control Fields

For an externally described file, fields that have the same control level indicator are combined in the order in which the fields are described in the data description specifications (DDS), not in the order in which the fields are specified on the input specifications. For example, if these fields are specified in DDS in the following order:

EMPNO
DPTNO
REGNO

and if these fields are specified with the same control level indicator in the following order on the input specifications:

REGNO L3

DPTNO L3

EMPNO L3

the fields are combined in the following order to form a split control field: EMPNO DPTNO REGNO.

Some special rules for split control fields are:

- For one control level indicator, you can split a field in some record types and not in others if the field names are different. However, the length of the field, whether split or not, must be the same in all record types.
- You can vary the length of the portions of a split control field for different record types if the field names are different. However, the total length of the portions must always be the same.
- A split control field can be made up of a combination of packed decimal fields and zoned decimal fields so long as the field lengths (in digits or characters) are the same.
- You must assign all portions of a split control field in one record type the same field record relation indicator and it must be defined on consecutive specification lines.

- When a split control field contains a date, time, or timestamp field than all fields in the split control field must be of the same type.

Figure 14 shows examples of the preceding rules.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IDISK      BC  91   95 C1
I         OR   92   95 C2
I         OR   93   95 C3
I
I* All portions of the split control field must be assigned the same
I* control level indicator and all must have the same field record
I* relation entry.
I                                   1    5  FLD1A         L1
I                                  46   50  FLD1B         L1
I                                  11   13  FLDA          L2
I                                  51   60  FLD2A         L3
I                                  31   40  FLD2B         L3
I                                  71   75  FLD3A         L4  92
I                                  26   27  FLD3B         L4  92
I                                  41   45  FLD3C         L4  92
I                                  61   70  FLDB              92
I                                  21   25  FLDC              92
I                                   6   10  FLD3D         L4  93
I                                  14   20  FLD3E         L4  93
```

Figure 14. Split Control Fields–Special Rules

The record identified by a '1' in position 95 has two split control fields:

1. FLD1A and FLD1B
2. FLD2A and FLD2B

The record identified with a '2' in position 95 has three split control fields:

1. FLD1A and FLD1B
2. FLD2A and FLD2B
3. FLD3A, FLD3B, and FLD3C

The third record type, identified by the 3 in position 95, also has three split control fields:

1. FLD1A and FLD1B
2. FLD2A and FLD2B
3. FLD3D and FLD3E

Field Indicators

A field indicator is defined by an entry in positions 69 and 70, 71 and 72, or 73 and 74 of the input specifications. The valid field indicators are:

01-99
H1-H9
U1-U8
RT

You can use a field indicator to determine if the specified field or array element is greater than zero, less than zero, zero, or blank. Positions 69 through 72 are valid for numeric fields only; positions 73 and 74 are valid for numeric or character fields. An indicator specified in positions 69 and 70 is set on when the numeric input field is greater than zero; an indicator specified in positions 71 and 72 is set on when the numeric input field is less than zero; and an indicator specified in positions 73 and 74 is set on when the numeric input field is zero or when the character input field is blank. You can then use the field indicator to condition calculation or output operations.

A field indicator is set on when the data for the field or array element is extracted from the record and the condition it represents is present in the input record. This field indicator remains on until another record of the same type is read and the condition it represents is not present in the input record, or until the indicator is set off as the result of a calculation.

You can use halt indicators (H1 through H9) as field indicators to check for an error condition in the field or array element as it is read into the program.

Rules for Assigning Field Indicators

When you assign field indicators, remember the following:

- Indicators for plus, minus, zero, or blank are set off at the beginning of the program. They are not set on until the condition (plus, minus, zero, or blank) is satisfied by the field being tested on the record just read.
- Field indicators cannot be used with entire arrays or with look-ahead fields. However, an entry can be made for an array element.
- A numeric input field can be assigned two or three field indicators. However, only the indicator that signals the result of the test on that field is set on; the others are set off.
- If the same field indicator is assigned to fields in different record types, its state (on or off) is always based on the last record type selected.
- When different field indicators are assigned to fields in different record types, a field indicator remains on until another record of that type is read. Similarly, a field indicator assigned to more than one field within a single record type always reflects the status of the last field defined.
- The same field indicator can be specified as a field indicator on another input specification, as a resulting indicator, as a record identifying indicator, or as a field record relation indicator. No diagnostic message is issued, but this use of indicators could cause erroneous results, especially when match fields or level control is involved.
- If the same indicator is specified in all three positions, the indicator is always set on when the record containing this field is selected.

Resulting Indicators

A resulting indicator is defined by an entry in positions 71 through 76 of the calculation specifications. The purpose of the resulting indicators depends on the operation code specified in positions 26 through 35. (See the individual operation code in Chapter 22, “Operation Codes” for a description of the purpose of the resulting indicators.) For example, resulting indicators can be used to test the result field after an arithmetic operation, to identify a record-not-found condition, to indicate an exception/error condition for a file operation, or to indicate an end-of-file condition.

The valid resulting indicators are:

01-99
H1-H9
OA-OG, OV
L1-L9
LR
U1-U8
KA-KN, KP-KY (valid only with SETOFF)
RT

You can specify resulting indicators in three places (positions 71-72, 73-74, and 75-76) of the calculation specifications. The positions in which the resulting indicator is defined determine the condition to be tested.

In most cases, when a calculation is processed, the resulting indicators are set off, and, if the condition specified by a resulting indicator is satisfied, that indicator is set on. However, there some exceptions to this rule, notably "LOOKUP (Look Up a Table or Array Element)" on page 385, "SETOFF (Set Indicator Off)" on page 458, and "SETON (Set Indicator On)" on page 459. A resulting indicator can be used as a conditioning indicator on the same calculation line or in other calculations or output operations. When you use it on the same line, the prior setting of the indicator determines whether or not the calculation is processed. If it is processed, the result field is tested and the current setting of the indicator is determined (see Figure 15 on page 42).

Rules for Assigning Resulting Indicators

When assigning resulting indicators, remember the following:

- Resulting indicators cannot be used when the result field refers to an entire array.
- If the same indicator is used to test the result of more than one operation, the last operation processed determines the setting of the indicator.
- When L1 through L9 indicators are used as resulting indicators and are set on, lower level indicators are not set on. For example, if L8 is set on, L1 through L7 are not set on.
- If H1 through H9 indicators are set on when used as resulting indicators, the program halts unless the halt indicator is set off prior to being checked in the program cycle. (See Chapter 3, "Program Cycle" on page 13).
- The same indicator can be used to test for more than one condition depending on the operation specified.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  Two resulting indicators are used to test for the different
C*  conditions in a subtraction operation.  These indicators are
C*  used to condition the calculations that must be processed for
C*  a payroll job.  Indicator 10 is set on if the hours worked (HRSWKD)
C*  are greater than 40 and is then used to condition all operations
C*  necessary to find overtime pay.  If Indicator 20 is not on
C*  (the employee worked 40 or more hours), regular pay based on a
C*  40-hour week is calculated.
C*
C     HRSWKD        SUB       40            OVERTM            3 01020
C*
C  N20PAYRAT        MULT (H)  40            PAY               6 2
C   10OVERTM        MULT (H)  OVRRAT        OVRPAY            6 2
C   10OVRPAY        ADD       PAY           PAY
C*
C*  If indicator 20 is on (employee worked less than 40 hours), pay
C*  based on less than a 40-hour week is calculated.
C   20PAYRAT        MULT (H)  HRSWKD        PAY
C*
```

Figure 15. Resulting Indicators Used to Condition Operations

Indicators Not Defined on the RPG IV Specifications

Not all indicators that can be used as conditioning indicators in an RPG IV program are defined on the specification forms. External indicators (U1 through U8) are defined by a CL command or by a previous RPG IV program. Internal indicators (1P, LR, MR, and RT) are defined by the RPG IV program cycle itself.

External Indicators

The external indicators are U1 through U8. These indicators can be set in a CL program or in an RPG IV program. In a CL program, they can be set by the SWS (switch-setting) parameter on the CL commands CHGJOB (Change Job) or CRTJOBD (Create Job Description). In an RPG IV program, they can be set as a resulting indicator or field indicator.

The status of the external indicators can be changed in the program by specifying them as resulting indicators on the calculation specifications or as field indicators on the input specifications. However, changing the status of the OS/400 job switches with a CL program during processing of an RPG IV program has no effect on the copy of the external indicators used by the RPG IV program. Setting the external indicators on or off in the program has no effect on file operations. File operations function according to the status of the U1 through U8 indicators when the program is initialized. However, when a program ends normally with LR on, the external indicators are copied back into storage, and their status reflects their last status in the RPG IV program. The current status of the external indicators can then be used by other programs.

Note: When using "RETURN (Return to Caller)" on page 444 with the LR indicator off, you are specifying a return without an end and, as a result, no external indicators are updated.

Internal Indicators

Internal indicators include:

- First page indicator
- Last record indicator
- Matching record indicator
- Return Indicator.

First Page Indicator (1P)

The first page (1P) indicator is set on by the RPG IV program when the program starts running and is set off by the RPG IV program after detail time output. The first record will be processed after detail time output. The 1P indicator can be used to condition heading or detail records that are to be written at 1P time. Do not use the 1P indicator to condition output fields that require data from input records because the input data will not be available.

The 1P indicator cannot be used to condition total or exception output lines and should not be used in an AND relationship with control level indicators. The 1P indicator cannot be specified as a resulting indicator.

Last Record Indicator (LR)

In a program that contains a primary file, the last record indicator (LR) is set on after the last record from a primary/secondary file has been processed, or it can be set on by the programmer.

The LR indicator can be used to condition calculation and output operations that are to be done at the end of the program. When the LR indicator is set on, all other control level indicators (L1 through L9) are also set on. If any of the indicators L1 through L9 have not been defined as control level indicators, as record identifying indicators, as resulting indicators, or by *INxx, the indicators are set on when LR is set on, but they cannot be used in other specifications.

In a program that does not contain a primary file, you can set the LR indicator on as one method to end the program. (For more information on how to end a program without a primary file, see Chapter 3, “Program Cycle” on page 13.) To set the LR indicator on, you can specify the LR indicator as a record identifying indicator or a resulting indicator. If LR is set on during detail calculations, all other control level indicators are set on at the beginning of the next cycle. LR and the record identifying indicators are both on throughout the remainder of the detail cycle, but the record identifying indicators are set off before LR total time.

Matching Record Indicator (MR)

The matching record indicator (MR) is associated with the matching field entries M1 through M9. It can only be used in a program when Match Fields are defined in the primary and at least one secondary file.

The MR indicator is set on when all the matching fields in a record of a secondary file match all the matching fields of a record in the primary file. It remains on during the complete processing of primary and secondary records. It is set off when all total calculations, total output, and overflow for the records have been processed.

At detail time, MR always indicates the matching status of the record just selected for processing; at total time, it reflects the matching status of the previous record. If all primary file records match all secondary file records, the MR indicator is always on.

Use the MR indicator as a field record relation indicator, or as a conditioning indicator in the calculation specifications or output specifications to indicate operations that are to be processed only when records match. The MR indicator cannot be specified as a resulting indicator.

For more information on Match Fields and multifile processing, see Chapter 6, "General File Considerations."

Return Indicator (RT)

You can use the return indicator (RT) to indicate to the internal RPG IV logic that control should be returned to the calling program. The test to determine if RT is on is made after the test for the status of LR and before the next record is read. If RT is on, control returns to the calling program. RT is set off when the program is called again.

Because the status of the RT indicator is checked after the halt indicators (H1 through H9) and LR indicator are tested, the status of the halt indicators or the LR indicator takes precedence over the status of the RT indicator. If both a halt indicator and the RT indicator are on, the halt indicator takes precedence. If both the LR indicator and RT indicator are on, the program ends normally.

RT can be set on as a record identifying indicator, a resulting indicator, or a field indicator. It can then be used as a conditioning indicator for calculation or output operations.

For a description of how RT can be used to return control to the calling program, see "Communicating with Other Objects" in the *ILE RPG/400 Programmer's Guide*.

Using Indicators

Indicators that you have defined as overflow indicators, control level indicators, record identifying indicators, field indicators, resulting indicators, *IN, *IN(xx), *INxx, or those that are defined by the RPG IV language can be used to condition files, calculation operations, or output operations. An indicator must be defined before it can be used as a conditioning indicator. The status (on or off) of an indicator is not affected when it is used as a conditioning indicator. The status can be changed only by defining the indicator to represent a certain condition.

File Conditioning

The file conditioning indicators are specified by the EXTIND keyword on the file description specifications. Only the external indicators U1 through U8 are valid for file conditioning. (The USROPN keyword can be used to specify that no implicit OPEN should be done.)

If the external indicator specified is off when the program is called, the file is not opened and file operations for that file are ignored while the program is running. Primary and secondary input files are processed as if they were at end-of-file. The end-of-file indicator is set on for all READ operations to that file. Input, calculation, and output specifications for the file need not be conditioned by the external indicator.

Rules for File Conditioning

When you condition files, remember the following:

- A file conditioning entry can be made for input, output, update, or combined files.
- A file conditioning entry cannot be made for table or array input.
- Output files for tables can be conditioned by U1 through U8. If the indicator is off, the table is not written.
- A record address file can be conditioned by U1 through U8, but the file processed by the record address file cannot be conditioned by U1 through U8.
- If the indicator conditioning a primary file with matching records is off, the MR indicator is not set on.
- Output does not occur for an output, an update, or a combined file if the indicator conditioning the file is off.
- If the indicator conditioning an input, an update, or a combined file is off, the file is considered to be at end of file. The end-of-file indicator is set on for READ, READC, READE, READPE, and READP operations. CHAIN, EXFMT, SETGT, and SETLL operations are ignored and no indicators are set.

Field Record Relation Indicators

Field record relation indicators are specified in positions 67 and 68 of the input specifications. The valid field record relation indicators are:

01-99
H1-H9
MR
RT
L1-L9
U1-U8

Field record relation indicators cannot be specified for externally described files.

You use field record relation indicators to associate fields with a particular record type when that record type is one of several in an OR relationship. The field described on the specification line is available for input only if the indicator specified in the field record relation entry is on or if the entry is blank. If the entry is blank, the field is common to all record types defined by the OR relationship.

Assigning Field Record Relation Indicators

You can use a record identifying indicator (01 through 99) in positions 67 and 68 to relate a field to a particular record type. When several record types are specified in an OR relationship, all fields that do not have a field record relation indicator in positions 67 and 68 are associated with all record types in the OR relationship. To relate a field to just one record type, you enter the record identifying indicator assigned to that record type in positions 67 and 68 (see Figure 16 on page 47).

An indicator (01 through 99) that is not a record identifying indicator can also be used in positions 67 and 68 to condition movement of the field from the input area to the input fields.

Control fields, which you define with an L1 through L9 indicator in positions 63 and 64 of the input specifications, and match fields, which are specified by a match value (M1 through M9) in positions 65 and 66 of the input specifications, can also be related to a particular record type in an OR relationship if a field record relation indicator is specified. Control fields or match fields in the OR relationship that do not have a field record relation indicator are used with all record types in the OR relationship.

If two control fields have the same control level indicator or two match fields have the same matching level value, a field record relation indicator can be assigned to just one of the match fields. In this case, only the field with the field record relation indicator is used when that indicator is on. If none of the field record relation indicators are on for that control field or match field, the field without a field record relation indicator is used. Control fields and match fields can only have entries of 01 through 99 or H1 through H9 in positions 67 and 68.

You can use positions 67 and 68 to specify that the program accepts and uses data from a particular field only when a certain condition occurs (for example, when records match, when a control break occurs, or when an external indicator is on). You can indicate the conditions under which the program accepts data from a field by specifying indicators L1 through L9, MR, or U1 through U8 in positions 67 and 68. Data from the field named in positions 49 through 62 is accepted only when the field record relation indicator is on.

External indicators are primarily used when file conditioning is specified with the "EXTIND(*INUx)" on page 190 keyword on the file description specifications. However, they can be used even though file conditioning is not specified.

A halt indicator (H1 through H9) in positions 67 and 68 relates a field to a record that is in an OR relationship and also has a halt indicator specified in positions 21 and 22.

Remember the following points when you use field record relation indicators:

- Control level (positions 63 and 64) and matching fields (positions 65 and 66) with the same field record relation indicator must be grouped together.
- Fields used for control level (positions 63 and 64) and matching field entries (positions 65 and 66) without a field record relation indicator must appear before those used with a field record relation indicator.
- Control level (positions 63 and 64) and matching fields (positions 65 and 66) with a field record relation indicator (positions 67 and 68) take precedence, when the indicator is on, over control level and matching fields of the same level without an indicator.
- Field record relations (positions 67 and 68) for matching and control level fields (positions 63 through 66) must be specified with record identifying indicators (01 through 99 or H1 through H9) from the main specification line or an OR relation line to which the matching field refers. If multiple record types are specified in an OR relationship, an indicator that specifies the field relation can be used to relate matching and control level fields to the pertinent record type.
- Noncontrol level (positions 63 and 64) and matching field (positions 65 and 66) specifications can be interspersed with groups of field record relation entries (positions 67 and 68).

- The MR indicator can be used as a field record relation indicator to reduce processing time when certain fields of an input record are required only when a matching condition exists.
- The number of control levels (L1 through L9) specified for different record types in the OR relationship can differ. There can be no control level for certain record types and a number of control levels for other record types.
- If all matching fields (positions 65 and 66) are specified with field record relation indicators (positions 67 and 68), each field record relation indicator must have a complete set of matching fields associated with it.
- If one matching field is specified without a field record relation indicator, a complete set of matching fields must be specified for the fields without a field record relation indicator.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC...............................
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IREPORT    AA  14    1 C5
I        OR    16    1 C6
I                                20   30  FLDB
I                                 2   10  FLDA              07
I*
I*  Indicator 07 was specified elsewhere in the program.
I*
I                                40   50  FLDC              14
I                                60   70  FLDD              16
```

Figure 16. Field Record Relation

The file contains two different types of records, one identified by a 5 in position 1 and the other by a 6 in position 1. The FLDC field is related by record identifying indicator 14 to the record type identified by a 5 in position 1. The FLDD field is related to the record type having a 6 in position 1 by record identifying indicator 16. This means that FLDC is found on only one type of record (that identified by a 5 in position 1) and FLDD is found only on the other type. FLDA is conditioned by indicator 07, which was previously defined elsewhere in the program. FLDB is found on both record types because it is not related to any one type by a record identifying indicator.

Function Key Indicators

You can use function key indicators in a program that contains a WORKSTN device if the associated function keys are specified in data description specifications (DDS). Function keys are specified in DDS with the CFxx or CAxx keyword. For an example of using function key indicators with a WORKSTN file, see the WORKSTN chapter in the *ILE RPG/400 Programmer's Guide*.

Function Key Indicator	Corresponding Function Key	Function Key Indicator	Corresponding Function Key
KA	1	KM	13
KB	2	KN	14
KC	3	KP	15
KD	4	KQ	16
KE	5	KR	17
KF	6	KS	18
KG	7	KT	19
KH	8	KU	20
KI	9	KV	21
KJ	10	KW	22
KK	11	KX	23
KL	12	KY	24

The function key indicators correspond to function keys 1 through 24. Function key indicator KA corresponds to function key 1, KB to function key 2 . . . KY to function key 24.

Function key indicators that are set on can then be used to condition calculation or output operations. Function key indicators can be set off by the SETOFF operation.

Halt Indicators (H1-H9)

You can use the halt indicators (H1 through H9) to indicate errors that occur during the running of a program. The halt indicators can be set on as record identifying indicators, field indicators, or resulting indicators.

The halt indicators are tested at the *GETIN step of the RPG IV cycle (see Chapter 3, "Program Cycle" on page 13). If a halt indicator is on, a message is issued to the user. The following responses are valid:

- Set off the halt indicator and continue the program.
- Issue a dump and end the program.
- End the program with no dump.

If a halt indicator is on when a RETURN operation is processed or when the LR indicator is on, the called program ends abnormally. The calling program is informed that the called program ended with a halt indicator on.

For a detailed description of the steps that occur when a halt indicator is on, see the detailed flowchart of the RPG IV cycle in Chapter 3, "Program Cycle" on page 13.

Indicators Conditioning Calculations

Indicators that are used to specify the conditions under which a calculation is done are to be defined elsewhere in the program. Indicators to condition calculations can be specified in positions 7 and 8 and/or in positions 9 through 11.

Positions 7 and 8

You can specify control level indicators (L1 through L9 and LR) in positions 7 and 8 of the calculation specifications.

If positions 7 and 8 are blank, the calculation is processed at detail time, is a statement within a subroutine, or is a declarative statement. If indicators L1 through L9 are specified, the calculation is processed at total time only when the specified indicator is on. If the LR indicator is specified, the calculation is processed during the last total time.

Note: An L0 entry can be used to indicate that the calculation is a total calculation that is to be processed on every program cycle.

Positions 9-11

You can use positions 9 through 11 of the calculation specifications to specify indicators that control the conditions under which an operation is processed. You can specify N is position 9 to indicate that the indicator should be tested for the value of off ('0') The valid entries for positions 10 through 11 are:

01-99
H1-H9
MR
OA-OG, OV
L1-L9
LR
U1-U8
KA-KN, KP-KY
RT

Any indicator that you use in positions 9 through 11 must be previously defined as one of the following types of indicators:

- Overflow indicators (file description specifications "OFLIND(*INxx)" on page 192)
- Record identifying indicators (input specifications, positions 21 and 22)
- Control level indicators (input specifications, positions 63 and 64)
- Field indicators (input specifications, positions 69 through 74)
- Resulting indicators (calculation specifications, positions 71 through 76)
- External indicators
- Indicators are set on, such as LR and MR
- *IN array, *IN(xx) array element, or *INxx field (see "Indicators Referred to As Data" on page 56 for a description of how an indicator is defined when used with one of these reserved words).

If the indicator must be off to condition the operation, place an N in positions 9. The indicators in grouped AND/OR lines, plus the control level indicators (if specified in positions 7 and 8), must all be exactly as specified before the operation is done as in Figure 17 on page 50.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C   25
CAN L1              SUB       TOTAL         TOTAL        A
CL2 10
CANNL3TOTAL         MULT      05            SLSTAX       B
C*
```

Figure 17. Conditioning Operations (Control Level Indicators)

Assume that indicator 25 represents a record type and that a control level 2 break occurred when record type 25 was read. L1 and L2 are both on. All operations conditioned by the control level indicators in positions 7 and 8 are done before operations conditioned by control level indicators in positions 9 through 11. Therefore, the operation in **B** occurs before the operation in **A**. The operation in **A** is done on the first record of the new control group indicated by 25, whereas the operation in **B** is a total operation done for all records of the previous control group.

The operation in **B** can be done when the L2 indicator is on provided the other conditions are met: Indicator 10 must be on; the L3 indicator must not be on.

The operation conditioned by both L2 and NL3 is done only when a control level 2 break occurs. These two indicators are used together because this operation is not to be done when a control level 3 break occurs, even though L2 is also on.

Some special considerations you should know when using conditioning indicators in positions 9 through 11 are as follows:

- With externally described work station files, the conditioning indicators on the calculation specifications must be either defined in the RPG program or be defined in the DDS source for the workstation file.
- With program described workstation files, the indicators used for the workstation file are unknown at compile time of the RPG program. Thus indicators 01-99 are assumed to be declared and they can be used to condition the calculation specifications without defining them.
- Halt indicators can be used to end the program or to prevent the operation from being processed when a specified error condition is found in the input data or in another calculation. Using a halt indicator is necessary because the record that causes the halt is completely processed before the program stops. Therefore, if the operation is processed on an error condition, the results are in error. A halt indicator can also be used to condition an operation that is to be done only when an error occurs.
- If LR is specified in positions 9 through 11, the calculation is done after the last record has been processed or after LR is set on.
- If a control level indicator is used in positions 9 through 11 and positions 7 and 8 are not used (detail time), the operation conditioned by the indicator is done only on the record that causes a control break or any higher level control break.

- If a control level indicator is specified in positions 7 and 8 (total time) and MR is specified in positions 9 through 11, MR indicates the matching condition of the previous record and not the one just read that caused the control break. After all operations conditioned by control level indicators in positions 7 and 8 are done, MR then indicates the matching condition of the record just read.
- If positions 7 and 8 and positions 9 through 11 are blank, the calculation specified on the line is done at detail calculation time.

Figure 18 through Figure 19 show examples of conditioning indicators.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...*
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
I*
I*  Field indicators can be used to condition operations.  Assume the
I*  program is to find weekly earnings including overtime.  The over-
I*  time field is checked to determine if overtime was entered.
I*  If the employee has worked overtime, the field is positive and -
I*  indicator 10 is set on.  In all cases the weekly regular wage
I*  is calculated.  However, overtime pay is added only if
I*  indicator 10 is on.
I*
ITIME      AB  01
I                                  1    7  EMPLNO
I                                  8   10 0OVERTM              10
I                                 15   20 2RATE
I                                 21   25 2RATEOT
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++
C*
C*  Field indicator 10 was assigned on the input specifications.
C*  It is used here to condition calculation operations.
C*
C                   EVAL (H)  PAY = RATE * 40
C   10              EVAL (H)  PAY = PAY + (OVERTM * RATEOT)
```

Figure 18. Conditioning Operations (Field Indicators)

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC...............................
I......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
I*
I*  A record identifying indicator is used to condition an operation.
I*  When a record is read with a T in position 1, the 01 indicator is
I*  set on.  If this indicator is on, the field named SAVE is added
I*  to SUM.  When a record without T in position 1 is read, the 02
I*  indicator is set on.  The subtract operation, conditioned by 02,
I*  then performed instead of the add operation.
I*
IFILE      AA  01    1 CT
I         OR   02    1NCT
I                                 10   15 2SAVE
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  Record identifying indicators 01 and 02 are assigned on the input
C*  specifications.  They are used here to condition calculation
C*  operations.
C*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C   01              ADD       SAVE          SUM               8 2
C   02              SUB       SAVE          SUM               8 2
```

Figure 19. Conditioning Operations (Record Identifying Indicators)

Indicators Used in Expressions

Indicators can be used as booleans in expressions in the extended factor 2 field of the calculation specification. They must be referred to as data (that is, using *IN or *INxx). The following examples demonstrate this.

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C* In these examples, the IF structure is performed only if 01 is on.
C* *IN01 is treated as a boolean with a value of on or off.

C* In the first example, the value of the indicator ('0' or '1') is
C* checked.
C                   IF        *IN01

C* In the second example, the logical expression B < A is evaluated.
C* If true, 01 is set on.  If false 01 is set off.  This is analogous
C* to using COMP with A and B and placing 01 in the appropriate
C* resulting indicator position.
C                   EVAL      *IN01 = B < A
```

Figure 20. Indicators Used in Expressions

See the expressions chapter and the operation codes chapter in this document for more examples and further details.

Indicators Conditioning Output

Indicators that you use to specify the conditions under which an output record or an output field is written must be previously defined in the program. Indicators to condition output are specified in positions 21 through 29. All indicators are valid for conditioning output.

The indicators you use to condition output must be previously defined as one of the following types of indicators:

- Overflow indicators (file description specifications, "OFLIND(*INxx)" on page 192)
- Record identifying indicators (input specifications, positions 21 and 22)
- Control level indicators (input specifications, positions 63 and 64)
- Field indicators (input specifications, positions 69 through 74)
- Resulting indicators (calculation specifications, positions 71 through 76)
- Indicators set by the RPG IV program such as 1P and LR
- External indicators set prior to or during program processing
- *IN array, *IN(xx) array element, or *INxx field (see "Indicators Referred to As Data" on page 56 for a description of how an indicator is defined when used with one of these reserved words).

If an indicator is to condition an entire record, you enter the indicator on the line that specifies the record type (see Figure 21 on page 54). If an indicator is to condition when a field is to be written, you enter the indicator on the same line as the field name (see Figure 21 on page 54).

Conditioning indicators are not required on output lines. If conditioning indicators are not specified, the line is output every time that type of record is checked for output. If you specify conditioning indicators, one indicator can be entered in each of the three separate output indicator fields (positions 22 and 23, 25 and 26, and 28 and 29). If these indicators are on, the output operation is done. An N in the position preceding each indicator (positions 21, 24, or 27) means that the output operation is done only if the indicator is not on (a negative indicator). No output line should be conditioned by all negative indicators; at least one of the indicators should be positive. If all negative indicators condition a heading or detail operation, the operation is done at the beginning of the program cycle when the first page (1P) lines are written.

You can specify output indicators in an AND/OR relationship by specifying AND/OR in positions 16 through 18. An unlimited number of AND/OR lines can be used. AND/OR lines can be used to condition output records, but they cannot be used to condition fields. However, you can condition a field with more than three indicators by using the EVAL operation in calculations. The following example illustrates this.

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C* Indicator 20 is set on only if indicators 10, 12, 14,16, and 18
C* are set on.
C                   EVAL      *IN20 = *IN10 AND *IN12 AND *IN14
C                              AND *IN16 AND *IN18
OFilename++DAddN01N02N03Excnam++++.......................................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O* OUTFIELD is conditioned by indicator 20, which effectively
O* means it is conditioned by all the indicators in the EVAL
O* operation.
OPRINTER   E
O                20      OUTFIELD
```

Other special considerations you should know about for output indicators are as follows:

- The first page indicator (1P) allows output on the first cycle before the primary file read, such as printing on the first page. The line conditioned by the 1P indicator must contain constant information used as headings or fields for reserved words such as PAGE and UDATE. The constant information is specified in the output specifications in positions 53 through 80. If 1P is used in an OR relationship with an overflow indicator, the information is printed on every page (see Figure 22 on page 55). Use the 1P indicator only with heading or detail output lines. It cannot be used to condition total or exception output lines or should not be used in an AND relationship with control level indicators.
- If certain error conditions occur, you might not want output operation processed. Use halt indicators to prevent the data that caused the error from being used (see Figure 23 on page 55).
- To condition certain output records on external conditions, use external indicators to condition those records.

See the Printer File section in the *ILE RPG/400 Programmer's Guide* for a discussion of the considerations that apply to assigning overflow indicators on the output specifications.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+...........................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O*
O*  One indicator is used to condition an entire line of printing.
O*  When 44 is on, the fields named INVOIC, AMOUNT, CUSTR, and SALSMN
O*  are all printed.
O*
OPRINT     D    44                     1
O                       INVOIC              10
O                       AMOUNT              18
O                       CUSTR               65
O                       SALSMN              85
O*
O*  A control level indicator is used to condition when a field should
O*  be printed.  When indicator 44 is on, fields INVOIC, AMOUNT, and
O*  CUSTR are always printed.  However, SALSMN is printed for the
O*  first record of a new control group only if 44 and L1 are on.
O*
OPRINT     D    44                  1
O                       INVOIC              10
O                       AMOUNT              18
O                       CUSTR               65
O                  L1   SALSMN              85
```

Figure 21. Output Indicators

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O*
O*  The 1P indicator is used when headings are to be printed
O*  on the first page only.
O*
OPRINT     H      1P                    3
0                                            8 'ACCOUNT'
O*
O*  The 1P indicator and an overflow indicator can be used to print
O*  headings on every page.
O*
OPRINT     H     1P                     3  1
0         OR    OF
0                                            8 'ACCOUNT'
```

Figure 22. 1P Indicator

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
I*
I*  When an error condition (zero in FIELDB) is found, the halt
I*  indicator is set on.
I*
IDISK      AA  01
I                                   1   3  FIELDA         L1
I                                   4   8 0FIELDB                  H1
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  When H1 is on, all calculations are bypassed.
C*
C   H1               GOTO      END
C                     :
C                     :               Calculations
C                     :
C     END            TAG
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O*
O*  FIELDA and FIELDB are printed only if H1 is not on.
O*  Use this general format when you do not want information that
O*  is in error to be printed.
O*
OPRINT     H    L1                  0  2 01
0                                           50 'HEADING'
0          D    01NH1                  1  0
0                       FIELDA                5
0                       FIELDB        Z     15
```

Figure 23. Preventing Fields from Printing

Indicators Referred to As Data

An alternative method of referring to and manipulating RPG IV indicators is provided by the RPG IV reserved words *IN and *INxx.

*IN

The array *IN is a predefined array of 99 one-position, character elements representing the indicators 01 through 99. The elements of the array should contain only the character values '0' (zero) or '1' (one).

The specification of the *IN array or the *IN(xx) variable-index array element as a field in an input record, as a result field, or as factor 1 in a PARM operation defines indicators 01 through 99 for use in the program.

The operations or references valid for an array of single character elements are valid with the array *IN except that the array *IN cannot be specified as a subfield in a data structure, or as a result field of a PARM operation.

*INxx

The field *INxx is a predefined one-position character field where xx represents any one of the RPG IV indicators except 1P or MR.

The specification of the *INxx field or the *IN(n) fixed-index array element (where n = 1 - 99) as a field in an input record, as a result field, or as factor 1 in a PARM operation defines the corresponding indicator for use in the program.

You can specify the field *INxx wherever a one-position character field is valid except that *INxx cannot be specified as a subfield in a data structure, as the result field of a PARM operation, or in a SORTA operation.

Additional Rules

Remember the following rules when you are working with the array *IN, the array element *IN(xx) or the field *INxx:

- Moving a character '0' (zero) or *OFF to any of these fields sets the corresponding indicator off.
- Moving a character '1' (one) or *ON to any of these fields sets the corresponding indicator on.
- Do not move any value, other than '0' (zero) or '1' (one), to *INxx. Any subsequent normal RPG IV indicator tests may yield unpredictable results.

See Figure 24 on page 57 for some examples of indicators referred to as data.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  When this program is called, a single parameter is passed to
C*  control some logic in the program.  The parameter sets the value
C*  of indicator 50.  The parameter must be passed with a character
C*  value of 1 or 0.
C*
C     *ENTRY        PLIST
C     *IN50         PARM                    SWITCH            1
C*
C*
C*  Subroutine SUB1 uses indicators 61 through 68.  Before the
C*  subroutine is processed, the status of these indicators used in
C*  the mainline program is saved.  (Assume that the indicators are
C*  set off in the beginning of the subroutine.)  After the subroutine
C*  is processed, the indicators are returned to their original state.
C*
C*
C                   MOVEA     *IN(61)        SAV8             8
C                   EXSR      SUB1
C                   MOVEA     SAV8          *IN(61)
```

Figure 24 (Part 1 of 2). Examples of Indicators Referred to as Data

```
C*
C*  A code field (CODE) contains a numeric value of 1 to 5 and is
C*  used to set indicators 71 through 75.  The five indicators are set
C*  off.  Field X is calculated as 70 plus the CODE field.  Field X is
C*  then used as the index into the array *IN.  Different subroutines
C*  are then used based on the status of indicators 71 through 75.
C*
C                   MOVEA     '00000'      *IN(71)
C   70              ADD       CODE         X               3 0
C                   MOVE      *ON          *IN(X)
C   71              EXSR      CODE1
C   72              EXSR      CODE2
C   73              EXSR      CODE3
C   74              EXSR      CODE4
C   75              EXSR      CODE5
```

Figure 24 (Part 2 of 2). Examples of Indicators Referred to as Data

Summary of Indicators

Table 1 and Table 2 show a summary of where indicators are defined, what the valid entries are, where the indicators are used, and when the indicators are set on and off. Table 2 on page 59 indicates the primary condition that causes each type of indicator to be set on and set off by the RPG IV program. "Function Key Indicators" on page 47 lists the function key indicators and the corresponding function keys.

Table 1. Indicator Entries and Uses

	Where Defined/Used	01-99	1P	H1-H9	L1-L9	LR	MR	OA-OG OV	U1-U8	KA-KN KP-KY	RT
User Defined	Overflow indicator, file description specifications, OFLIND keyword	X						X			
	Record identifying indicator input specifications, positions 21-22	X		X	X	X			X		X
	Control level, input specifications, positions 63-64				X						
	Field level, input specifications, positions 69-74	X		X					X		X
	Resulting indicator, calculation specifications, positions 71-76	X		X	X	X		X[1]	X	X[2]	X
RPG Defined	Internal Indicator		X			X	X				X
	External Indicator								X		
Used	File conditioning, file description specifications								X		
	File record relation, input specifications 67-68[3]	X		X	X		X		X		X
	Control level, calculation specifications, positions 7-8				X	X					
	Conditioning indicators, calculation specifications, positions 9-11	X		X	X	X	X	X	X	X	X
	Output indicators, output specifications, positions 21-29	X	X[4]	X	X	X	X	X	X	X	X

[1] The overflow indicator must be defined on the file description specification first.

[2] KA through KN and KP through KY can be used as resulting indicators only with the SETOFF operation.

[3] Only a record identifying indicator from a main or OR record can be used to condition a control or match field. L1 or L9 cannot be used to condition a control or match field.

[4] The 1P indicator is allowed only on heading and detail lines.

Table 2 (Page 1 of 2). When Indicators Are Set On and Off by the RPG IV Logic Cycle

Type of Indicator	Set On	Set Off
Overflow	When printing on or spacing or skipping past the overflow line.	OA-OG, OV: After the following heading and detail lines are completed. 01-99: By the user.
Record identifying	When specified primary / secondary record has been read and before total calculations are processed; immediately after record is read from a full procedural file.	Before the next primary/secondary record is read during the next processing cycle.
Control level	When the value in a control field changes. All lower level indicators are also set on.	At end of following detail cycle.
Field indicator	By blank or zero in specified fields, by plus in specified field, or by minus in specified field.	Before this field status is to be tested the next time.

Table 2 (Page 2 of 2). When Indicators Are Set On and Off by the RPG IV Logic Cycle

Type of Indicator	Set On	Set Off
Resulting	When the calculation is processed and the condition that the indicator represents is met.	The next time a calculation is processed for which the same indicator is specified as a resulting indicator and the specified condition is not met.
Function key	When the corresponding function key is pressed for WORKSTN files and at subsequent reads to associated subfiles.	By SETOFF or move fields logic for a WORKSTN file.
External U1-U8	By CL command prior to beginning the program, or when used as a resulting or a field indicator.	By CL command prior to beginning the program, or when used as a resulting or when used as a resulting or a field indicator.
H1-H9	As specified by programmer.	When the continue option is selected as a response to a message, or by the programmer.
RT	As specified by programmer.	When the program is called again.
Internal Indicators 1P	At beginning of processing before any input records are read.	Before the first record is read.
LR	After processing the last primary/secondary record of the last file or by the programmer.	At the beginning of processing, or by the programmer.
MR	If the match field contents of the record of a secondary file correspond to the match field contents of a record in the primary file.	When all total calculations and output are completed for the last record of the matching group.

Chapter 5. Exception/Error Data Structures and Subroutines

File Exception/Errors

File Information Data Structure

A file information data structure (INFDS) can be defined for each file to make file exception/error and file feedback information available to the program. The file information data structure must be unique for each file. The file information data structure contains the following feedback information:

- File Feedback (length is 80)
- Open Feedback (length is 160)
- Input/Output Feedback (length is 126)
- Device Specific Feedback (length is variable)
- Get Attributes Feedback (length is variable)

Note: The get attributes feedback uses the same positions in the INFDS as the input/output feedback and device specific feedback. This means that if you have a get attributes feedback, you cannot have input/output feedback or device feedback, and vise versa.

The length of the INFDS depends on what fields you have declared in your INFDS. The minimum length of the INFDS is 80.

File Feedback Information

The file feedback information starts in position 1 and ends in position 80 in the file information data structure. The file feedback information contains data about the file which is specific to RPG. This includes information about the error/exception that identify:

- The name of the file for which the exception/error occurred
- The record being processed when the exception/error occurred or the record that caused the exception/error
- The last operation being processed when the exception/error occurred
- The status code
- The RPG IV routine in which the exception/error occurred.

The fields from position 1 to position 66 in the file feedback section of the INFDS are always provided and updated even if INFDS is not specified in the program. The fields from position 67 to position 80 of the file feedback section of the INFDS are only updated after a POST operation to a specific device.

If INFDS is not specified, the information in the file feedback section of the INFDS can be output using the DUMP operation. For more information see “DUMP (Program Dump)” on page 358.

Overwriting the file feedback section of the INFDS may cause unexpected results in subsequent error handling and is not recommended.

The location of some of the more commonly used the subfields in the file feedback section of the INFDS is defined by special keywords. The contents of the file feedback section of the INFDS along with the special keywords and their descriptions can be found in the following tables:

Table 3 (Page 1 of 2). Contents of the File Feedback Information Available in the File Information Data Structure (INFDS)

From (Positions 26-32)	To (Positions 33-39)	Format	Length	Keyword	Information
1	8	Character	8	*FILE	The first 8 characters of the file name.
9	9	Character	1		Open indication (1 = open).
10	10	Character	1		End of file (1 = end of file)
11	15	Zoned decimal	5,0	*STATUS	Status code. For a description of these codes, see “Status Codes” on page 75.
16	21	Character	6	*OPCODE	Operation code The first five positions (left-adjusted) specify the type of operation by using the character representation of the calculation operation codes. For example, if a READE was being processed, READE is placed in the leftmost five positions. If the operation was an implicit operation (for example, a primary file read or update on the output specifications), the equivalent operation code is generated (such as READ or UPDAT) and placed in location *OPCODE. Operation codes which have 6 letter names will be shortened to 5 letters. **DELETE** DELET **EXCEPT** EXCPT **READPE** REDPE **UNLOCK** UNLCK **UPDATE** UPDAT The remaining position contains one of the following: **F** The last operation was specified for a file name. **R** The last operation was specified for a record. **I** The last operation was an implicit file operation.
22	29	Character	8	*ROUTINE	First 8 characters of the procedure name or zero if the call is by procedure pointer.
30	37	Character	8		RPG IV source listing line number.
38	42	Zoned decimal	5,0		User-specified reason for error on SPECIAL file.

Table 3 (Page 2 of 2). Contents of the File Feedback Information Available in the File Information Data Structure (INFDS)

From (Positions 26-32)	To (Positions 33-39)	Format	Length	Keyword	Information
38	45	Character	8	*RECORD	For a program described file the record identifying indicator is placed left-adjusted in the field; the remaining six positions are filled with blanks. For an externally described file, the first 8 characters of the name of the record being processed when the exception/error occurred.
46	52	Character	7		Machine or system message number.
53	66	Character	14		Unused.

Table 4. Contents of the File Feedback Information Available in the File-Information Data Structure (INFDS) Valid after a POST

From (Positions 26-32)	To (Positions 33-39)	Format	Length	Keyword	Information
67	70	Zoned decimal	4,0	*SIZE	Screen size (product of the number of rows and the number of columns on the device screen).
71	72	Zoned decimal	2,0	*INP	The display's keyboard type. Set to 00 if the keyboard is alphanumeric or katakana. Set to 10 if the keyboard is ideographic.
73	74	Zoned decimal	2,0	*OUT	The display type. Set to 0O if the display is alphanumeric or katakana. Set to 10 if the display is ideographic. Set to 20 if the display is DBCS.
75	76	Zoned decimal	2,0	*MODE	Always set to 00.

INFDS File Feedback Example: To specify an INFDS which contains fields in the file feedback section, you can make the following entries:

- Specify the INFDS keyword on the file description specification with the name of the file information data structure
- Specify the file information data structure and the subfields you wish to use on a definition specification.
- Specify special keywords left-adjusted, in the FROM field (positions 26-32) on the definition specification, or specify the positions of the fields in the FROM field (position 26-32) and the TO field (position 33-39).

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    IF   E             DISK    INFDS(FILEFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DFILEFBK          DS
D FILE              *FILE                                                  * File name
D OPEN_IND                9      9                                         * File open?
D EOF_IND                10     10                                         * File at eof?
D STATUS            *STATUS                                                * Status code
D OPCODE            *OPCODE                                                * Last opcode
D ROUTINE           *ROUTINE                                               * RPG Routine
D LIST_NUM               30     37                                         * Listing line
D SPCL_STAT              38     42S 0                                      * SPECIAL status
D RECORD            *RECORD                                                * Record name
D MSGID                  46     52                                         * Error MSGID
D SCREEN            *SIZE                                                  * Screen size
D NLS_IN            *INP                                                   * NLS Input?
D NLS_OUT           *OUT                                                   * NLS Output?
D NLS_MODE          *MODE                                                  * NLS Mode?
```

Figure 25. Example of Coding an INFDS with File Feedback Information

Note: The keywords are not labels and cannot be used to access the subfields. Short entries are padded on the right with blanks.

Open Feedback Information

Positions 81 through 240 in the file information data structure contain open feedback information. The contents of the file open feedback area are copied by RPG to the open feedback section of the INFDS whenever the file associated with the INFDS is opened. This includes members opened as a result of a read operation on a multi-member processed file.

A description of the contents of the open feedback area, and what file types the fields are valid for, can be found in *Data Management*

INFDS Open Feedback Example: To specify an INFDS which contains fields in the open feedback section, you can make the following entries:

- Specify the INFDS keyword on the file description specification with the name of the file information data structure
- Specify the file information data structure and the subfields you wish to use on a definition specification.
- Use information in &dtamgmt to determine which fields you wish to include in the INFDS. To calculate the From and To positions (positions 26 through 32 and 33 through 39 of the definition specifications) that specify the subfields of the open feedback section of the INFDS, use the Offset, Data Type, and Length given in *Data Management* and do the following calculations:

  ```
  From = 81 + Offset
  To = From - 1 + Character_Length
  Character_Length = Length (in bytes)
  ```

  For example, for overflow line number of a printer file, *Data Management* gives:

```
   Offset = 107
   Data Type is binary
   Length = 2

Therefore,

   From = 81 + 107 = 188,
   To = 188 - 1 + 2 = 189.

   See subfield OVERFLOW in example below
```

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    O    F  132        PRINTER INFDS(OPNFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DOPNFBK                DS
D ODP_TYPE                   81     82                                         * ODP Type
D FILE_NAME                  83     92                                         * File name
D LIBRARY                    93    102                                         * Library name
D SPOOL_FILE                103    112                                         * Spool file name
D SPOOL_LIB                 113    122                                         * Spool file lib
D SPOOL_NUM                 123    124B 0                                      * Spool file num
D RCD_LEN                   125    126B 0                                      * Max record len
D KEY_LEN                   127    128B 0                                      * Max key len
D MEMBER                    129    138                                         * Member name
D TYPE                      147    148B 0                                      * File type
D ROWS                      152    153B 0                                      * Num PRT/DSP rows
D COLIMNS                   154    155B 0                                      * Num PRT/DSP cols
D NUM_RCDS                  156    159B 0                                      * Num of records
D ACC_TYPE                  160    161                                         * Access type
D DUP_KEY                   162    162                                         * Duplicate key?
D SRC_FILE                  163    163                                         * Source file?
D VOL_OFF                   184    185B 0                                      * Vol label offset
D BLK_RCDS                  186    187B 0                                      * Max rcds in blk
D OVERFLOW                  188    189B 0                                      * Overflow line
D BLK_INCR                  190    191B 0                                      * Blk increment
D FLAGS1                    196    196                                         * Misc flags
D REQUESTER                 197    206                                         * Requester name
D OPEN_COUNT                207    208B 0                                      * Open count
D BASED_MBRS                211    212B 0                                      * Num based mbrs
D FLAGS2                    213    213                                         * Misc flags
D OPEN_ID                   214    215                                         * Open identifier
D RCDFMT_LEN                216    217B 0                                      * Max rcd fmt len
D CCSID                     218    219B 0                                      * Database CCSID
D FLAGS3                    220    220                                         * Misc flags
D NUM_DEVS                  227    229B 0                                      * Num devs defined
```

Figure 26. Example of Coding an INFDS with Open Feedback Information

Input/Output Feedback Information

Positions 241 through 366 in the file information data structure are used for input/output feedback information. The contents of the file common input/output feedback area are copied by RPG to the input/output feedback section of the INFDS:

- On every input/output operation if a POST for the file, with factor 1 blank has not been specified anywhere in your program.
- Only after a POST for the file if a POST for the file, with factor 1 blank has been specified anywhere in your program.

For more information see “POST (Post)” on page 428.

A description of the contents of the input/output feedback area can be found in *Data Management*

INFDS Input/Output Feedback Example: To specify an INFDS which contains fields in the open feedback section, you can make the following entries:

- Specify the INFDS keyword on the file description specification with the name of the file information data structure
- Specify the file information data structure and the subfields you wish to use on a definition specification.
- Use information in &dtamgmt to determine which fields you wish to include in the INFDS. To calculate the From and To positions (positions 26 through 32 and 33 through 39 of the definition specifications) that specify the subfields of the input/output feedback section of the INFDS, use the Offset, Data Type, and Length given in *Data Management* and do the following calculations:

  ```
  From = 241 + Offset
  To = From - 1 + Character_Length
  Character_Length = Length (in bytes)
  ```

  For example, for device class of a file, *Data Management* gives:

  ```
  Offset = 30
  Data Type is character
  Length = 2
  ```

  ```
  Therefore,

     From = 241 + 30 = 271,
     To = 271 - 1 + 2 = 272.

     See subfield DEV_CLASS in example below
  ```

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    IF   E             DISK    INFDS(MYIOFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++Comments++++++++++
DMYIOFBK          DS
D                                                                         * 241-242 not used
D WRITE_CNT             243    246B 0                                     * Write count
D READ_CNT              247    250B 0                                     * Read count
D WRTRD_CNT             251    254B 0                                     * Write/read count
D OTHER_CNT             255    258B 0                                     * Other I/O count
D OPERATION             260    260                                        * Cuurent operation
D IO_RCD_FMT            261    270                                        * Rcd format name
D DEV_CLASS             271    272                                        * Device class
D IO_PGM_DEV            273    282                                        * Pgm device name
D IO_RCD_LEN            283    286B 0                                     * Rcd len of I/O
```

Figure 27. Example of Coding an INFDS with Input/Output Feedback Information

Device Specific Feedback Information

The device specific feedback information in the file information data structure starts at position 367 in the INFDS, and contains input/output feedback information specific to a device.

The length of the INFDS when device specific feedback information is required, depends on two factors: the device type of the file, and on whether DISK files are keyed or not. The minimum length is 528; but some files require a longer INFDS.

- For WORKSTN files, the INFDS is long enough to hold the device-specific feedback information for any type of display or ICF file starting at position 241. For example, if the longest device-specific feedback information requires 390 bytes, the INFDS for WORKSTN files is 630 bytes long (240+390=630).
- For externally-described DISK files, the INFDS is at least long enough to hold the longest key in the file beginning at position 401.

More information on the contents and length of the device feedback for database file, printer files, ICF and display files can be found in *Data Management*.

The contents of the device specific input/output feedback area of the file are copied by RPG to the device specific feedback section of the INFDS:

- On every input/output operation if a POST for the file, with factor 1 blank has not been specified anywhere in your program.
- Only after a POST for the file if a POST for the file, with factor 1 blank has been specified anywhere in your program.

For more information see “POST (Post)” on page 428.

INFDS Device Specific Feedback Examples: To specify an INFDS which contains fields in the open feedback section, you can make the following entries:

- Specify the INFDS keyword on the file description specification with the name of the file information data structure
- Specify the file information data structure and the subfields you wish to use on a definition specification.
- Use information in &dtamgmt to determine which fields you wish to include in the INFDS. To calculate the From and To positions (positions 26 through 32 and 33 through 39 of the definition specifications) that specify the subfields of the input/output feedback section of the INFDS, use the Offset, Data Type, and Length given in *Data Management* and do the following calculations:

  ```
  From = 367 + Offset
  To = From - 1 + Character_Length
  Character_Length = Length (in bytes)
  ```

  For example, for relative record number of a data base file, *Data Management* gives:

```
      Offset = 30
      Data Type is binary
      Length = 4

   Therefore,

      From = 367 + 30 = 397,
      To = 397 - 1 + 4 = 272.

      See subfield DB_RRN in DBFBK data structure in example below
```

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    O    F 132         PRINTER INFDS(PRTFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DPRTFBK           DS
D CUR_LINE              367    368B 0                                      * Current line num
D CUR_PAGE              369    372B 0                                      * Current page cnt
D PRT_MAJOR             401    402                                         * Major ret code
D PRT_MINOR             403    404                                         * Minor ret code
```

Figure 28. Example of Coding an INFDS with Printer Specific Feedback Information

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    IF   E             DISK    INFDS(DBFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DDBFBK            DS
D FDBK_SIZE             367    370B 0                                      * Size of DB fdbk
D JOIN_BITS             371    374B 0                                      * JFILE bits
D LOCK_RCDS             377    378B 0                                      * Nbr locked rcds
D POS_BITS              385    385                                         * File pos bits
D DLT_BITS              384    384                                         * Rcd deleted bits
D NUM_KEYS              387    388B 0                                      * Num keys (bin)
D KEY_LEN               393    394B 0                                      * Key length
D MBR_NUM               395    395B 0                                      * Member number
D DB_RRN                397    400B 0                                      * Relative-rcd-num
D KEY                   401   2400                                         * Key value (max
D                                                                          *   size 2000)
```

Figure 29. Example of Coding an INFDS with Database Specific Feedback Information

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    CF   E             WORKSTN INFDS(DSPFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DDSPFBK           DS
D DSP_FLAG1             367    368                                         * Display flags
D DSP_AID               369    369                                         * AID byte
D CURSOR                370    371                                         * Cursor location
D DATA_LEN              372    375B 0                                      * Actual data len
D SF_RRN                376    377B 0                                      * Subfile rrn
D MIN_RRN               378    379B 0                                      * Subfile min rrn
D NUM_RCDS              380    381B 0                                      * Subfile num rcds
D ACT_CURS              382    383                                         * Active window
D                                                                          *  cursor location
D DSP_MAJOR             401    402                                         * Major ret code
D DSP_MINOR             403    404                                         * Minor ret code
```

Figure 30. Example of Coding an INFDS with Display Specific Feedback Information

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    CF   E             WORKSTN INFDS(ICFFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DICFFBK           DS
D ICF_AID               369    369                                         * AID byte
D ICF_LEN               372    375B 0                                      * Actual data len
D ICF_MAJOR             401    402                                         * Major ret code
D ICF_MINOR             403    404                                         * Minor ret code
D SNA_SENSE             405    412                                         * SNA sense rc
D SAFE_IND              413    413                                         * Safe indicator
D RQSWRT                415    415                                         * Request write
D RMT_FMT               416    425                                         * Remote rcd fmt
D ICF_MODE              430    437                                         * Mode name
```

Figure 31. Example of Coding an INFDS with ICF Specific Feedback Information

Get Attributes Feedback Information

The get attributes feedback information in the file information data structure starts at position 241 in the INFDS, and contains information about a display device or ICF session (a device associated with a WORKSTN file). The end position of the get attributes feedback information depends on the length of the data returned by a get attributes data management operation. The get attributes data management operation is performed when a POST with a program device specified for factor 1 is used.

More information about the contents and the length of the get attributes data can be found in *Data Management*.

INFDS Get Attributes Feedback Example: To specify an INFDS which contains fields in the get attributes feedback section, you can make the following entries:

- Specify the INFDS keyword on the file description specification with the name of the file information data structure
- Specify the file information data structure and the subfields you wish to use on a definition specification.
- Use information in &dtamgmt to determine which fields you wish to include in the INFDS. To calculate the From and To positions (positions 26 through 32 and 33 through 39 of the definition specifications) that specify the subfields of

the get attributes feedback section of the INFDS, use the Offset, Data Type, and Length given in *Data Management* and do the following calculations:

```
From = 241 + Offset
To = From - 1 + Character_Length
Character_Length = Length (in bytes)
```

For example, for device type of a file, *Data Management* gives:

```
   Offset = 31
   Data Type is character
   Length = 6

Therefore,

   From = 241 + 31 = 272,
   To = 272 - 1 + 6 = 277.

   See subfield DEV_TYPE in example below
```

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    CF   E             WORKSTN INFDS(DSPATRFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DDSPATRFBK        DS
D PGM_DEV               241    250                                         * Program device
D DEV_DSC               251    260                                         * Dev description
D USER_ID               261    270                                         * User ID
D DEV_CLASS             271    271                                         * Device class
D DEV_TYPE              272    277                                         * Device type
D REQ_DEV               278    278                                         * Requester?
D ACQ_STAT              279    279                                         * Acquire status
D INV_STAT              280    280                                         * Invite status
D DATA_AVAIL            281    281                                         * Data available
D NUM_ROWS              282    283B 0                                      * Number of rows
D NUM_COLS              284    285B 0                                      * Number of cols
D BLINK                 286    286                                         * Allow blink?
D LINE_STAT             287    287                                         * Online/offline?
D DSP_LOC               288    288                                         * Display location
D DSP_TYPE              289    289                                         * Display type
D KBD_TYPE              290    290                                         * Keyboard type
D CTL_INFO              342    342                                         * Controller info
D COLOR_DSP             343    343                                         * Color capable?
D GRID_DSP              344    344                                         * Grid line dsp?
D
D* Following fields apply to ISDN...
D ISDN_LEN              385    386B 0                                      * Rmt number len
D ISDN_TYPE             387    388                                         * Rmt number type
D ISDN_PLAN             389    390                                         * Rmt number plan
D ISDN_NUM              391    430                                         * Rmt number
D ISDN_SLEN             435    436B 0                                      * Rmt sub-address
D                                                                          *   length
D ISDN_STYPE            437    438                                         * Rmt sub-address
D                                                                          *   type
D ISDN_SNUM             439    478                                         * Rmt sub-address
D ISDN_CON              480    480                                         * Connection
D ISDN_RLEN             481    482B 0                                      * Rmt address len
D ISDN_RNUM             483    514                                         * Rmt address
D ISDN_ELEN             519    520                                         * Extension len
D ISDN_ETYPE            521    521                                         * Extension type
D ISDN_ENUM             522    561                                         * Extension num
D ISDN_XTYPE            566    566                                         * X.25 call type
D
```

Figure 32. Example of Coding an INFDS with Display file Get Attributes Feedback Information

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    CF   E             WORKSTN INFDS(ICFATRFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DICFATRFBK        DS
D PGM_DEV                241    250                                       * Program device
D DEV_DSC                251    260                                       * Dev description
D USER_ID                261    270                                       * User ID
D DEV_CLASS              271    271                                       * Device class
D DEV_TYPE               272    272                                       * Device type
D REQ_DEV                278    278                                       * Requester?
D ACQ_STAT               279    279                                       * Acquire status
D INV_STAT               280    280                                       * Invite status
D DATA_AVAIL             281    281                                       * Data available
D SES_STAT               291    291                                       * Session status
D SYNC_LVL               292    292                                       * Synch level
D CONV_TYPE              293    293                                       * Conversation typ
D RMT_LOC                294    301                                       * Remote location
D LCL_LU                 302    309                                       * Local LU name
D LCL_NETID              310    317                                       * Local net ID
D RMT_LU                 318    325                                       * Remote LU
D RMT_NETID              326    333                                       * Remote net ID
D APPC_MODE              334    341                                       * APPC Mode
D LU6_STATE              345    345                                       * LU6 conv state
D LU6_COR                346    353                                       * LU6 conv
D                                                                         *    correlator
D* Following fields apply to ISDN...
D ISDN_LEN               385    386B 0                                    * Rmt number len
D ISDN_TYPE              387    388                                       * Rmt number type
D ISDN_PLAN              389    390                                       * Rmt number plan
D ISDN_NUM               391    430                                       * Rmt number
D ISDN_SLEN              435    436B 0                                    * sub-addr len
D ISDN_STYPE             437    438                                       * sub-addr type
D ISDN_SNUM              439    478                                       * Rmt sub-address
D ISDN_CON               480    480                                       * Connection
D ISDN_RLEN              481    482B 0                                    * Rmt address len
D ISDN_RNUM              483    514                                       * Rmt address
D ISDN_ELEN              519    520                                       * Extension len
D ISDN_ETYPE             521    521                                       * Extension type
D ISDN_ENUM              522    561                                       * Extension num
D ISDN_XTYPE             566    566                                       * X.25 call type
```

Figure 33 (Part 1 of 2). Example of Coding an INFDS with ICF file Get Attributes Feedback Information

```
D
D* Following info is available only when program was started
D* as result of a received program start request... (P_ stands for protected)
D TRAN_PGM                567    630                                 * Trans pgm name
D P_LUWIDLN               631    631                                 * LUWID fld len
D P_LUNAMELN              632    632                                 * LU-NAME len
D P_LUNAME                633    649                                 * LU-NAME
D P_LUWIDIN               650    655                                 * LUWID instance
D P_LUWIDSEQ              656    657B 0                              * LUWID seq num
D
D* Following info is available only when a protected conversation
D* is started on remote system...  (U_ stands for unprotected)
D U_LUWIDLN               658    658                                 * LUWID fld len
D U_LUNAMELN              659    659                                 * LU-NAME len
D U_LUNAME                660    676                                 * LU-NAME
D U_LUWIDIN               677    682                                 * LUWID instance
D U_LUWIDSEQ              683    684B 0                              * LUWID seq num
```

Figure 33 (Part 2 of 2). Example of Coding an INFDS with ICF file Get Attributes Feedback Information

Blocking Considerations

The fields of the input/output specific feedback in the INFDS and in most cases the fields of the device specific feedback information section of the INFDS, are not updated for each operation to the file in which the records are blocked and unblocked. The feedback information is updated only when a block of records is transferred between RPG IV system and the OS/400 system. However, if you are doing blocked input on a data base file, the relative record number and the key value in the data base feedback section of the INFDS are updated:

- On every input/output operation if a POST for the file, with factor 1 blank has not been specified anywhere in your program.
- Only after a POST for the file if a POST for the file, with factor 1 blank has been specified anywhere in your program.

You can obtain valid updated feedback information by using the CL command OVRDBF (Override with Database File) with SEQONLY(*NO) specified. If you use a file override command, the RPG IV language does not block or unblock the records in the file.

For more information on blocking and unblocking of records in RPG see *ILE RPG/400 Programmer's Guide*.

File Exception/Error Subroutine (INFSR)

To identify the user-written RPG IV subroutine that may receive control following file exception/errors, specify the INFSR keyword on the File Description specification with the name of the subroutine that receives control when exception/errors occur on this file. The subroutine name can be *PSSR, which indicates that the program exception/error subroutine is given control for the exception/errors on this file.

A file exception/error subroutine (INFSR) receives control when an exception/error occurs on an implicit (primary or secondary) file operation or on an explicit file operation that does not have an indicator specified in positions 73 and 74. The file exception/error subroutine can also be run by the EXSR operation code. Any of

the RPG IV operations can be used in the file exception/error subroutine. Factor 1 of the BEGSR operation and factor 2 of the EXSR operation must contain the name of the subroutine that receives control (same name as specified with the INFSR keyword on the file description specifications).
The ENDSR operation must be the last specification for the file exception/error subroutine and should be specified as follows:

Position **Entry**

6 C

7-11 Blank

12-25 Can contain a label that is used in a GOTO specification within the subroutine.

26-35 ENDSR

36-49 Optional entry to designate where control is to be returned following processing of the subroutine. The entry must be a 6-position character field, literal, or array element whose value specifies one of the following return points.

Note: If the return points are specified as literals, they must be enclosed in apostrophes. If they are specified as named constants, the constants must be character and must contain only the return point with no leading blanks. If they are specified in fields or array elements, the value must be left-adjusted in the field or array element.

***DETL** Continue at the beginning of detail lines.

***GETIN** Continue at the get input record routine.

***TOTC** Continue at the beginning of total calculations.

***TOTL** Continue at the beginning of total lines.

***OFL** Continue at the beginning of overflow lines.

***DETC** Continue at the beginning of detail calculations.

***CANCL** Cancel the processing of the program.

Blanks Return control to the RPG IV default error handler. This applies when factor 2 is a value of blanks and when factor 2 is not specified. If the subroutine was called by the EXSR operation and factor 2 is blank, control returns to the next sequential instruction. Blanks are only valid at runtime.

50-76 Blank.

Remember the following when specifying the file exception/error subroutine:

- The programmer can explicitly call the file exception/error subroutine by specifying the name of the subroutine in factor 2 of the EXSR operation.
- After the ENDSR operation of the file exception/error subroutine is run, the RPG IV language resets the field or array element specified in factor 2 to blanks. Thus, if the programmer does not place a value in this field during the processing of the subroutine, the RPG IV default error handler receives control following processing of the subroutine unless the subroutine was called by the EXSR operation. Because factor 2 is set to blanks, the programmer can specify the return point within the subroutine that is best suited for the exception/error that occurred. If the subroutine was called by the EXSR operation and factor 2 of the ENDSR operation is blank, control returns to the next sequential instruction following the EXSR operation. A file exception/error subroutine can handle errors in more than one file.

- If a file exception/error occurs during the start or end of a program, control passes to the RPG IV default error handler, and not to the user-written file exception/error or subroutine (INFSR).
- Because the file exception/error subroutine may receive control whenever a file exception/error occurs, an exception/error could occur while the subroutine is running if an I/O operation is processed on the file in error. If an exception/error occurs on the file already in error while the subroutine is running, the subroutine is called again; this will result in a program loop unless the programmer codes the subroutine to avoid this problem. One way to avoid such a program loop is to set a first-time switch in the subroutine. If it is not the first time through the subroutine, set on a halt indicator and issue the RETURN operation as follows:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* If INFSR is already handling the error, exit.
C     ERRRTN        BEGSR
C     SW            IFEQ      '1'
C                   SETON                                        H1
C                   RETURN
C* Otherwise, flag the error handler.
C                   ELSE
C                   MOVE      '1'           SW
C                     :
C                     :
C                     :
C                   ENDIF
C* End error processing.
C                   MOVE      '0'           SW
C                   ENDSR
```

Note: It may not be possible to continue processing the file after an I/O error has occurred. To continue, it may be necessary to issue a CLOSE operation and then an OPEN operation to the file.

Status Codes

File Status Codes

Any code placed in the subfield location *STATUS that is greater than 99 is considered to be an exception/error condition. If the status code is greater than 99, the error indicator, if specified in positions 73 and 74, is set on or the file exception/error subroutine receives control. Location *STATUS is updated after every file operation.

The codes in the following tables are placed in the subfield location *STATUS for the file information data structure:

Table 5. Normal Codes

Code	Device[1]	RC[2]	Condition
00000			No exception/error.
00002	W	n/a	Function key used to end display.
00011	W,D,SQ	11xx	End of file on a read (input).
00012	W,D,SQ	n/a	No-record-found condition on a CHAIN, SETLL, and SETGT operations.
00013	W	n/a	Subfile is full on WRITE operation.

Note: [1]"Device" refers to the devices for which the condition applies. The following abbreviations are used: P = PRINTER; D = DISK; W = WORKSTN; SP = SPECIAL; SQ = Sequential. The major/minor return codes under column RC apply only to WORKSTN files. [2]The formula mmnn is used to described major/minor return codes: mm is the major and nn the minor.

Table 6 (Page 1 of 2). Exception/Error Codes

Code	Device[1]	RC[2]	Condition
01011	W,D,SQ	n/a	Undefined record type (input record does not match record identifying indicator).
01021	W,D,SQ	n/a	Tried to write a record that already exists (file being used has unique keys and key is duplicate, or attempted to write duplicate relative record number to a subfile).
01022	D	n/a	Referential constraint error detected on file member.
01031	W,D,SQ	n/a	Match field out of sequence.
01041	n/a	n/a	Array/table load sequence error.
01042	n/a	n/a	Array/table load sequence error. Alternate collating sequence used.
01051	n/a	n/a	Excess entries in array/table file.
01071	W,D,SQ	n/a	Numeric sequence error.
01121[4]	W	n/a	No indicator on the DDS keyword for Print key.
01122[4]	W	n/a	No indicator on the DDS keyword for Roll Up key.
01123[4]	W	n/a	No indicator on the DDS keyword for Roll Down key.
01124[4]	W	n/a	No indicator on the DDS keyword for Clear key.
01125[4]	W	n/a	No indicator on the DDS keyword for Help key.
01126[4]	W	n/a	No indicator on the DDS keyword for Home key.
01201	W	34xx	Record mismatch detected on input.
01211	all	n/a	I/O operation to a closed file.
01215	all	n/a	OPEN issued to a file already opened.
01216[3]	all	yes	Error on an implicit OPEN/CLOSE operation.
01217[3]	all	yes	Error on an explicit OPEN/CLOSE operation.
01218	D,SQ	n/a	Record already locked.
01221	D,SQ	n/a	Update operation attempted without a prior read.
01222	D,SQ	n/a	Record cannot be allocated due to referential constraint error
01231	SP	n/a	Error on SPECIAL file.
01235	P	n/a	Error in PRTCTL space or skip entries.

Table 6 (Page 2 of 2). Exception/Error Codes

Code	Device[1]	RC[2]	Condition
01241	D,SQ	n/a	Record number not found. (Record number specified in record address file is not present in file being processed.)
01251	W	80xx 81xx	Permanent I/O error occurred.
01255	W	82xx 83xx	Session or device error occurred. Recovery may be possible.
01261	W	n/a	Attempt to exceed maximum number of acquired devices.
01271	W	n/a	Attempt to acquire unavailable device
01281	W	n/a	Operation to unacquired device.
01282	W	0309	Job ending with controlled option.
01284	W	n/a	Unable to acquire second device for single device file
01285	W	0800	Attempt to acquire a device already acquired.
01286	W	n/a	Attempt to open shared file with SAVDS or IND options.
01287	W	n/a	Response indicators overlap IND indicators.
01299	W,D,SQ	yes	Other I/O error detected.
01331	W	0310	Wait time exceeded for READ from WORKSTN file.

Note: [1]"Device" refers to the devices for which the condition applies. The following abbreviations are used: P = PRINTER; D = DISK; W = WORKSTN; SP = SPECIAL; SQ = Sequential. The major/minor return codes under column RC apply only to WORKSTN files. [2]The formula mmnn is used to described major/minor return codes: mm is the major and nn the minor. [3]Any errors that occur during an open or close operation will result in a *STATUS value of 1216 or 1217 regardless of the major/minor return code value. [4]See Figure 8 on page 26 for special handling.

The following table shows the major/minor return code to *STATUS value mapping for errors that occur to AS/400 programs using WORKSTN files only. See the *Data Management* for more information on Major/Minor return codes.

Major	Minor	*STATUS
00,02	all	00000
03	all (except 09,10)	00000
03	09	01282
03	10	01331
04	all	01299
08	all	01285[1]
11	all	00011
34	all	01201
80,81	all	01251
82,83	all	01255

Note: [1]The return code field will not be updated for a *STATUS value of 1285, 1261, or 1281 because these conditions are detected before calling data management. To monitor for these errors, you must check for the *STATUS value and not for the corresponding major/minor return code value.

Program Exception/Errors

Some examples of program exception/errors are: division by zero, SQRT of a negative number, invalid array index, an error on a CALL, an error return from a called program, and a start position or length out of range for a string operation. They can be handled in one of the following ways:

- An indicator can be specified in positions 73 and 74 of the calculation specifications for an operation code. This indicator is set on if an exception/error occurs during the processing of the specified operation. The optional program status data structure is updated with the exception/error information. You can determine the action to be taken by testing the indicator.
- You can create a user defined ILE exception handler which will take control when an exception occurs. For more information see *ILE RPG/400 Programmer's Guide*.
- A program exception/error subroutine can be specified. You enter *PSSR in factor 1 of a BEGSR operation to specify this subroutine. Information regarding the program exception/error is made available through a program status data structure that is specified with an S in position 23 of the data structure statement on the definition specifications.
- If the indicator or the program exception/error subroutine is not present, program exception/errors are handled by the RPG IV default error handler.

Program Status Data Structure

A program status data structure can be defined to make program exception/error information available to an RPG IV program.

A data structure is defined as a program status data structure by an S in position 23 of the data structure statement. A program status data structure contains predefined subfields that provide you with information about the program exception/error that occurred. The location of the subfields in the program status data structure is defined by special keywords or by predefined From and To positions. In order to access the subfields, you assign a name to each subfield. The keywords must be specified, left-adjusted in positions 26 through 39.

Information from the program status data structure is also provided in a formatted dump. However, a formatted dump might not contain information for fields in the PSDS if the PSDS is not coded, or the length of the PSDS does not include those fields. For example, if the PSDS is only 275 bytes long, the time and date or program running will appear as *N/A*. in the dump, since this information starts at byte 276. For more information see “DUMP (Program Dump)” on page 358.

Note: Call performance with LR on will be greatly improved by having no PSDS, or a PSDS no longer than 80 bytes, since some of the information to fill the PSDS after 80 bytes is costly to obtain.

Table 7 on page 79 provides the layout of the subfields of the data structure and the predefined From and To positions of its subfields that can be used to access information in this data structure.

Table 7 (Page 1 of 3). Contents of the Program Status Data Structure

From (Positions 26-32)	To (Positions 33-39)	Format	Length	Keyword	Information
1	10	Character	10	*PROC	Procedure name
11	15	Zoned decimal	5,0	*STATUS	Status code
16	20	Zoned decimal	5,0		Previous status code.
21	28	Character	8		RPG IV source listing line number. number.
29	36	Character	8	*ROUTINE	Name of the RPG IV routine in which the exception or error occurred. This subfield is updated at the beginning of an RPG IV routine or after a program call only when the *STATUS subfield is updated with a nonzero value. The following names identify the routines: *INIT Program initialization *DETL Detail lines *GETIN Get input record *TOTC Total calculations *TOTL Total lines *DETC Detail calculations *OFL Overflow lines *TERM Program ending *ROUTINE Name of program or procedure called (first 8 characters). **Note:** *ROUTINE is not valid unless you use the normal RPG IV cycle. Logic that takes the program out of the normal RPG IV cycle may cause *ROUTINE to reflect an incorrect value.
37	39	Zoned decimal	3,0	*PARMS	Number of parameters passed to this program from a calling program
40	42	Character	3		Exception type (CPF for a OS/400 system exception or MCH for a machine exception).
43	46	Character	4		Exception number. For a CPF exception, this field contains a CPF message number. For a machine exception, it contains a machine exception number.
47	50	Character	4		Reserved
51	80	Character	30		Work area for messages. This area is only meant for internal use by the RPG IV compiler. The organization of information will not always be consistent. It can be displayed by the user.
81	90	Character	10		Name of library in which the program is located.
91	170	Character	80		Retrieved exception data. CPF messages are placed in this subfield when location *STATUS contains 09999.

Table 7 (Page 2 of 3). Contents of the Program Status Data Structure

From (Positions 26-32)	To (Positions 33-39)	Format	Length	Keyword	Information
171	174	Character	4		Identification of the exception that caused RNX9001 exception to be signaled.
175	190	Character	16		Unused.
191	198	Character	8		Date (*DATE format) the job entered system. In the case of batch jobs submitted for overnight processing, those run after midnight will carry the next day's date.
199	200	Zoned decimal	2,0		First 2 digits of a 4-digit year. The same as the first 2 digits of *YEAR.
201	208	Character	8		Name of file on which the last file operation occurred (updated only when an error occurs).
209	243	Character	35		Status information on the last file used. This information includes the status code, the RPG IV opcode, the RPG IV routine name, the source listing line number, and record name. It is updated only when an error occurs. **Note:** The opcode name is in the same form as *OPCODE in the INFDS
244	253	Character	10		Job name.
254	263	Character	10		User name from the user profile.
264	269	Zoned decimal	6,0		Job number.
270	275	Zoned decimal	6,0		Date (in UDATE format) the job was entered in the system (UDATE is derived from this date). In the case of batch jobs submitted for overnight processing, those run after midnight will carry the next day's date.
276	281	Zoned decimal	6,0		Date of program running (the system date in UDATE format).
282	287	Zoned decimal	6 (zero decimal positions)		Time of program running in the format hhmmss.
288	293	Character	6		Date (in UDATE format) the program was compiled.
294	299	Character	6		Time (in the format hhmmss) the program was compiled.
300	303	Character	4		Level of the compiler.
304	313	Character	10		Source file name.

Table 7 (Page 3 of 3). Contents of the Program Status Data Structure

From (Positions 26-32)	To (Positions 33-39)	Format	Length	Keyword	Information
314	323	Character	10		Source library name.
324	333	Character	10		Source file member name.
334	343	Character	10		Program containing procedure.
344	353	Character	10		Module containing procedure.
354	429	character	76		Unused.

Program Status Codes

Any code placed in the subfield location *STATUS that is greater than 99 is considered to be an exception/error condition. If the status code is greater than 99, the error indicator, if specified in positions 73 and 74, is set on, or the program exception/error subroutine receives control. Location *STATUS is updated when an exception/error occurs.

The following codes are placed in the subfield location *STATUS for the program status data structure:

Normal Codes

Code	Condition
00000	No exception/error occurred
00001	Called program returned with the LR indicator on.

Exception/Error Codes

Code	Condition
00100	Value out of range for string operation
00101	Negative square root
00102	Divide by zero
00103	An intermediate result is not large enough to contain the result.
00112	Invalid Date, Time or Timestamp value.
00113	Date overflow or underflow. (For example, when the result of a Date calculation results in a number greater than *Hival or less than *Loval.)
00114	Date mapping errors, where a Date is mapped from a 4 character year to a 2 character year and the date range is not 1940-2039.
00120	Table or array out of sequence.
00121	Array index not valid
00122	OCCUR outside of range
00123	Reset attempted during initialization step of program
00202	Called program or procedure failed; halt indicator (H1 through H9) not on
00211	Error calling program or procedure
00221	Called program tried to use a parameter not passed to it.

00222	Pointer or parameter error
00231	Called program or procedure returned with halt indicator on
00232	Halt indicator on in this program
00233	Halt indicator on when RETURN operation run
00299	RPG IV formatted dump failed
00333	Error on DSPLY operation
00401	Data area specified on IN/OUT not found
00402	*PDA not valid for non-prestart job
00411	Data area type or length does not match
00412	Data area not locked for output
00413	Error on IN/OUT operation
00414	User not authorized to use data area
00415	User not authorized to change data area
00421	Error on UNLOCK operation
00431	Data area previously locked by another program
00432	Data area locked by program in the same process
00450	Character field not entirely enclosed by shift-out and shift-in characters
00501	Failure to retrieve sort sequence.
00502	Failure to convert sort sequence.
00802	Commitment control not active.
00803	Rollback operation failed.
00804	Error occurred on COMMIT operation
00805	Error occurred on ROLBK operation
00907	Decimal data error (digit or sign not valid)
00970	The level number of the compiler used to generate the program does not agree with the level number of the RPG IV run-time subroutines.
09998	Internal failure in RPG IV compiler or in run-time subroutines
09999	Program exception in system routine.

PSDS Example: To specify a PSDS in your program, you code the program status data structure and the subfields you wish to use on a definition specification.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DMYPSDS          SDS
D PROC_NAME         *PROC                                                  * Procedure name
D PGM_STATUS        *STATUS                                                * Status code
D PRV_STATUS             16     20S 0                                      * Previous status
D LINE_NUM               21     28                                         * Src list line num
D ROUTINE           *ROUTINE                                               * Routine name
D PARMS             *PARMS                                                 * Num passed parms
D EXCP_TYPE              40     42                                         * Exception type
D EXCP_NUM               43     46                                         * Exception number
D*
D PGM_LIB                81     90                                         * Program library
D EXCP_DATA              91    170                                         * Exception data
D EXCP_ID               171    174                                         * Exception Id
D DATE                  191    198                                         * Date (*DATE fmt)
D YEAR                  199    200S 0                                      * Year (*YEAR fmt)
D LAST_FILE             201    208                                         * Last file used
D FILE_INFO             209    243                                         * File error info
D JOB_NAME              244    253                                         * Job name
D USER                  254    263                                         * User name
D JOB_NUM               264    269S 0                                      * Job number
D JOB_DATE              270    275S 0                                      * Date (UDATE fmt)
D RUN_DATE              276    281S 0                                      * Run date (UDATE)
D RUN_TIME              282    287S 0                                      * Run time (UDATE)
D CRT_DATE              288    293                                         * Create date
D CRT_TIME              294    299                                         * Create time
D CPL_LEVEL             300    303                                         * Compiler level
D SRC_FILE              304    313                                         * Source file
D SRC_LIB               314    323                                         * Source file lib
D SRC_MBR               324    333                                         * Source file mbr
D PROC_PGM              334    343                                         * Pgm Proc is in
D PROC_MOD              344    353                                         * Mod Proc is in
```

Figure 34. Example of Coding a PSDS

Note: The keywords are not labels and cannot be used to access the subfields. Short entries are padded on the right with blanks.

Program Exception/Error Subroutine

To identify the user-written RPG IV subroutine that is to receive control when a program exception/error occurs, specify *PSSR in factor 1 of the subroutine's BEGSR operation. If an indicator is not specified in positions 73 and 74 for the operation code or if an exception occurs that is not expected for the operation code (ie. an array indexing error during a SCAN operation), control is transferred to this subroutine when a program exception/error occurs. In addition, the subroutine can also be called by the EXSR operation. *PSSR can be specified on the INFSR keyword on the file description specifications and receives control if a file exception/error occurs.

Any of the RPG IV operation codes can be used in the program exception/error subroutine. The ENDSR operation must be the last specification for the subroutine, and the factor 2 entry on the ENDSR operation specifies the return point following the running of the subroutine. For a discussion of the valid entries for factor 2, see "File Exception/Error Subroutine (INFSR)" on page 73.

Remember the following when specifying a program exception/error subroutine:

- You can explicitly call the *PSSR subroutine by specifying *PSSR in factor 2 of the EXSR operation.
- After the ENDSR operation of the *PSSR subroutine is run, the RPG IV language resets the field, subfield, array element, or array element specified in factor 2 to blanks. This allows you to specify the return point within the subroutine that is best suited for the running/error that occurred. If factor 2 contains blanks at the end of the subroutine, the RPG IV default error handler receives control; if the subroutine was called by an EXSR or CASxx operation, control returns to the next sequential instruction following the EXSR or ENDCS.
- Because the program exception/error subroutine may receive control whenever a non-file exception/error occurs, an exception/error could occur while the subroutine is running. If an exception/error occurs while the subroutine is running, the subroutine is called again; this will result in a program loop unless the programmer codes the subroutine to avoid this problem.
- If you have used the OPTIMIZE(*FULL) option on either the CRTBNDRPG or CRTRPGMOD commands, you have to declare all fields that you refer to during exception handling with the NOOPT keyword in the definition specification for the field. This will ensure that when you run your program, the fields referred to during exception handling will have current values.

Chapter 6. General File Considerations

This chapter contains a more detailed explanation of:

- Multi-file Processing
- Match fields
- Alternate collating sequence
- File translation.

Primary/Secondary Multi-file Processing

In an RPG IV program, the processing of a primary input file and one or more secondary input files, with or without match fields, is termed multi-file processing. Selection of records from more than one file based on the contents of match fields is known as multi-file processing by matching records. Multi-file processing can be used with externally described or program described input files that are designated as primary/secondary files.

Multi-file Processing with No Match Fields

When no match fields are used in multi-file processing, records are selected from one file at a time. When the records from one file are all processed, the records from the next file are selected. The files are selected in this order:

1. Primary file, if specified
2. Secondary files in the order in which they are described on the file description specifications.

Multi-file Processing with Match Fields

When match fields are used in multi-file processing, the program selects the records for processing according to the contents of the match fields. At the beginning of the first cycle, the program reads one record from every primary/secondary input file and compares the match fields in the records. If the records are in ascending order, the program selects the record with the lowest match field. If the records are in descending order, the program selects the record with the highest match field.

When a record is selected from a file, the program reads the next record from that file. At the beginning of the next program cycle, the new record is compared with the other records in the read area that are waiting for selection, and one record is selected for processing.

Records without match fields can also be included in the files. Such records are selected for processing before records with match fields. If two or more of the records being compared have no match fields, selection of those records is determined by the priority of the files from which the records came. The priority of the files is:

1. Primary file, if specified
2. Secondary files in the order in which they are described on the file description specifications.

When the primary file record matches one or more of the secondary records, the MR (matching record) indicator is set on. The MR indicator is on for detail time processing of a matching record through the total time that follows the record. This indicator can be used to condition calculation or output operations for the record that is selected. When one of the matching records must be selected, the selection is determined by the priority of the files from which the records came.

Figure 6 on page 22 shows the logic flow of multifile processing.

A program can be written where only one input file is defined with match fields and no other input files have match fields. The files without the match fields are then processed completely according to the previously mentioned priority of files. The file with the match fields is processed last, and sequence checking occurs for that file.

Assigning Match Field Values (M1-M9)

When assigning match field values (M1 through M9) to fields on the input specifications in positions 65 and 66, consider the following:

- Sequence checking is done for all record types with match field specifications. All match fields must be in the same order, either all ascending or all descending. The contents of the fields to which M1 through M9 are assigned are checked for correct sequence. An error in sequence causes the RPG IV exception/error handling routine to receive control. When the program continues processing, the next record from the same file is read.
- Not all files used in the program must have match fields. Not all record types within one file must have match fields either. However, at least one record type from two files must have match fields if files are ever to be matched.
- The same match field values must be specified for all record types that are used in matching. See Figure 35 on page 87.
- Date, time, and timestamp match fields with the same match field values (M1 through M9) must be the same type (for example, all date) but can be different formats.
- All character or numeric match fields with the same match field values (M1 through M9) should be the same length and type If the match field contains packed data, the zoned decimal length (two times packed length - 1) is used as the length of the match field. It is valid to match a packed field in one record against a zoned decimal field in another if the digit lengths are identical. The length must always be odd because the length of a packed field is always odd.
- Record positions of different match fields can overlap, but the total length of all fields must not exceed 256 characters.
- If more than one match field is specified for a record type, all the fields are combined and treated as one continuous field (see Figure 35 on page 87). The fields are combined according to descending sequence (M9 to M1) of matching field values.
- Match fields values cannot be repeated in a record.
- All match fields given the same matching field value (M1 through M9) are considered numeric if any one of the match fields is described as numeric.
- When numeric fields having decimal positions are matched, they are treated as if they had no decimal position. For instance 3.46 is considered equal to 346.
- Only the digit portions of numeric match fields are compared. Even though a field is negative, it is considered to be positive because the sign of the numeric field is ignored. Therefore, a -5 matches a +5.
- Date and time fields are converted to *ISO format for comparisons
- Graphic data is compared hexadecimally

- Whenever more than one matching field value is used, all match fields must match before the MR indicator is set on. For example, if match field values M1, M2, and M3 are specified, all three fields from a primary record must match all three match fields from a secondary record. A match on only the fields specified by M1 and M2 fields will not set the MR indicator on (see Figure 35).
- Matching fields cannot be used for lookahead fields, and arrays.
- Field names are ignored in matching record operations. Therefore, fields from different record types that are assigned the same match level can have the same name.
- If an alternate collating sequence or a file translation is defined for the program, character fields are matched according to the alternate sequence specified.
- A field specified as binary (B in position 36 of the input specifications) cannot be assigned a match field value. However, a field specified as packed (P in position 36 of the input specifications) can be assigned a match field value.
- Match fields that have no field record relation indicator must be described before those that do. When the field record relation indicator is used with match fields, the field record relation indicator should be the same as a record identifying indicator for this file, and the match fields must be grouped according to the field record relation indicator.
- When any match value (M1 through M9) is specified for a field without a field record relation indicator, all match values used must be specified once without a field record relation indicator. If all match fields are not common to all records, a dummy match field should be used. Field record relation indicators are invalid for externally described files. (see Figure 36 on page 89).
- Match fields are independent of control level indicators (L1 through L9).
- If multi-file processing is specified and the LR indicator is set on, the program bypasses the multi-file processing routine.

Figure 37 on page 90 is an example of how match fields are specified.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
FMASTER        IP    E           K DISK
FWEEKLY        IS    E           K DISK
```

The files in this example are externally described (E in position 22) and are to be processed by keys (K in position 34).

Figure 35 (Part 1 of 2). Match Fields in Which All Values Match

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IRcdname+++....Ri......................................................
I.............Ext-field+.................Field+++++++++L1M1..PlMnZr....
I*                           MASTER FILE
IEMPMAS        01
I                                        EMPLNO          M1
I                                        DIVSON          M3
I                                        DEPT            M2
IDEPTMS        02
I                                        EMPLNO          M1
I                                        DEPT            M2
I                                        DIVSON          M3
I*                           WEEKLY FILE
IWEEKRC        03
I                                        EMPLNO          M1
I                                        DIVSON          M3
I                                        DEPT            M2
```

Figure 35 (Part 2 of 2). Match Fields in Which All Values Match

Three files are used in matching records. All the files have three match fields specified, and all use the same values (M1, M2, M3) to indicate which fields must match. The MR indicator is set on only if all three match fields in either of the files EMPMAS and DEPTMS are the same as all three fields from the WEEKRC file.

The three match fields in each file are combined and treated as one match field organized in the following descending sequence:

DIVSON	M3
DEPT	M2
EMPLNO	M1

The order in which the match fields are specified in the input specifications does not affect the organization of the match fields.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC................................
I......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IDISK     AB  01    1 C1
I         OR  02    1 C2
I         OR  03    1 C3
I                             1   10 0EMPNO          M1
I                            11   15 0DUMMY          M2
I                            11   15 0DEPT           M202
I                            16   20 0DEPT           M203
```

Figure 36 (Part 1 of 2). Match Fields with Dummy Match Field

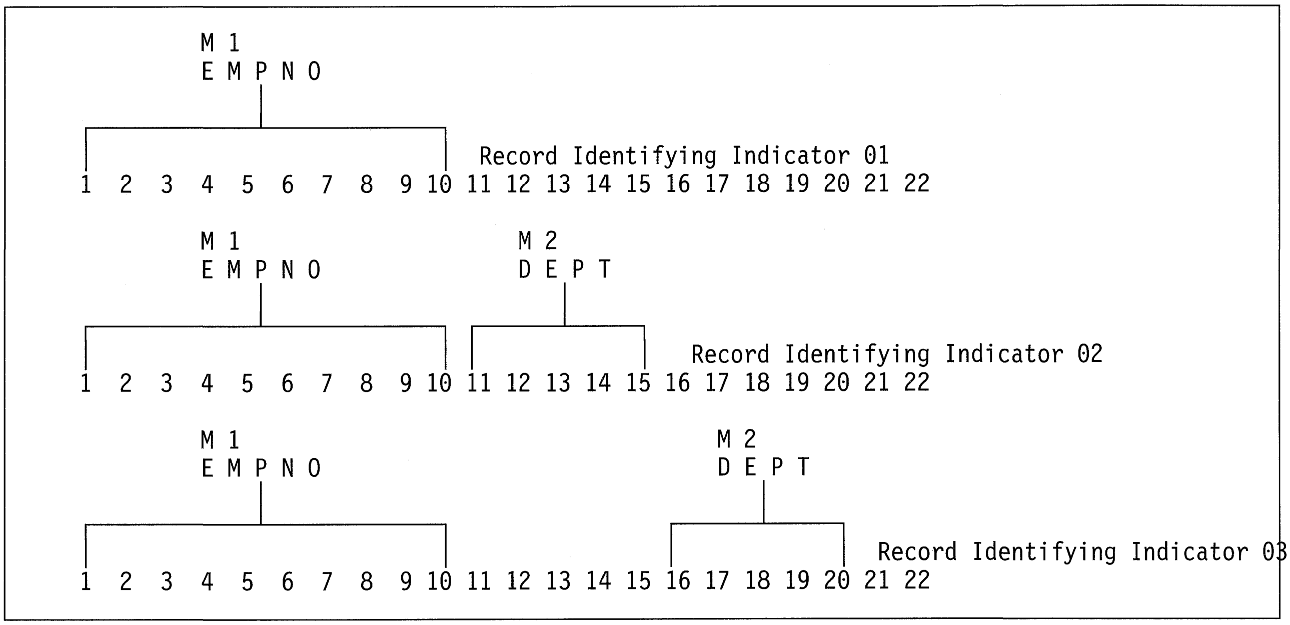


Figure 36 (Part 2 of 2). Match Fields with a Dummy M2 Field

Three different record types are found in the input file. All three contain a match field in positions 1 through 10. Two of them have a second match field. Because M1 is found on all record types, it can be specified without a field record relation entry in positions 67 and 68. If one match value (M1 through M9) is specified without field record relation entries, all match values must be specified once without field record relation entries. Because the value M1 is specified without field record relationship, an M2 value must also be specified once without field record relationship. The M2 field is not on all record types; therefore a dummy M2 field must be specified next. The dummy field can be given any unique name, but its specified length must be equal to the length of the true M2 field. The M2 field is then related to the record types on which it is found by field record relation entries.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
FPRIMARY   IPEA F   64        DISK
FFIRSTSEC  IS A F   64        DISK
FSECSEC    IS A F   64        DISK

*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IPRIMARY   AA  01    1 CP    2NC
I                                  2    3  MATCH           M1
I*
I          BB  02    1 CP    2 C
I                                  2    3  NOM
I*
IFIRSTSEC  AB  03    1 CS    2NC
I                                  2    3  MATCH           M1
I*
I          BC  04    1 CS    2 C
I                                  2    3  NOM
I*
ISECSEC    AC  05    1 CT    2NC
I                                  2    3  MATCH           M1
I*
I          BD  06    1 CT    2 C
I                                  2    3  NOM
```

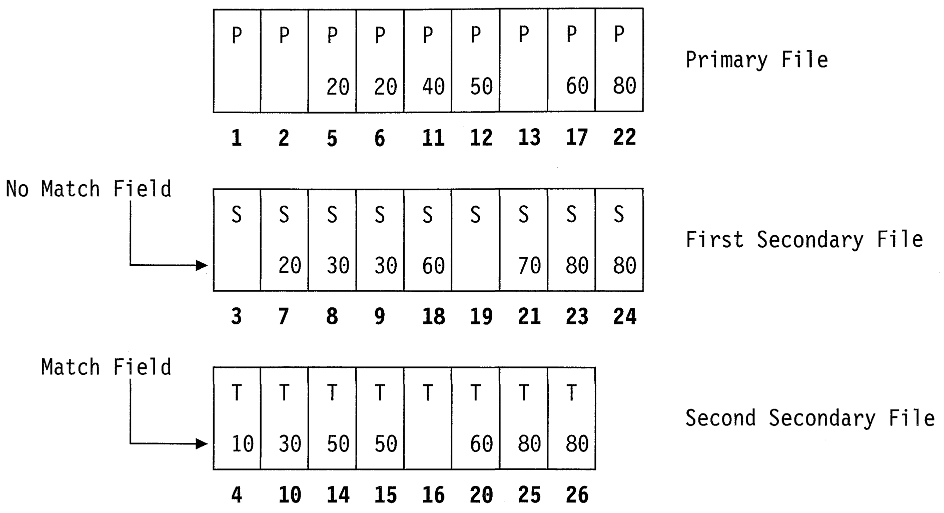
Figure 37. Match Field Specifications for Three Disk Files

Processing Matching Records

Matching records for two or more files are processed in the following manner:

- Whenever a record from the primary file matches a record from the secondary file, the primary file is processed first. Then the matching secondary file is processed. The record identifying indicator that identifies the record type just selected is on at the time the record is processed. This indicator is often used to control the type of processing that takes place.
- Whenever records from ascending files do not match, the record having the lowest match field content is processed first. Whenever records from descending files do not match, the record having the highest match field content is processed first.
- A record type that has no match field specification is processed immediately after the record it follows. The MR indicator is off. If this record type is first in the file, it is processed first even if it is not in the primary file.
- The matching of records makes it possible to enter data from primary records into their matching secondary records because the primary record is processed before the matching secondary record. However, the transfer of data from secondary records to matching primary records can be done only when look-ahead fields are specified.

Figure 38 through Figure 39 show how records from three files are selected for processing.



The records from the three disk files above are selected in the order indicated by the dark numbers.

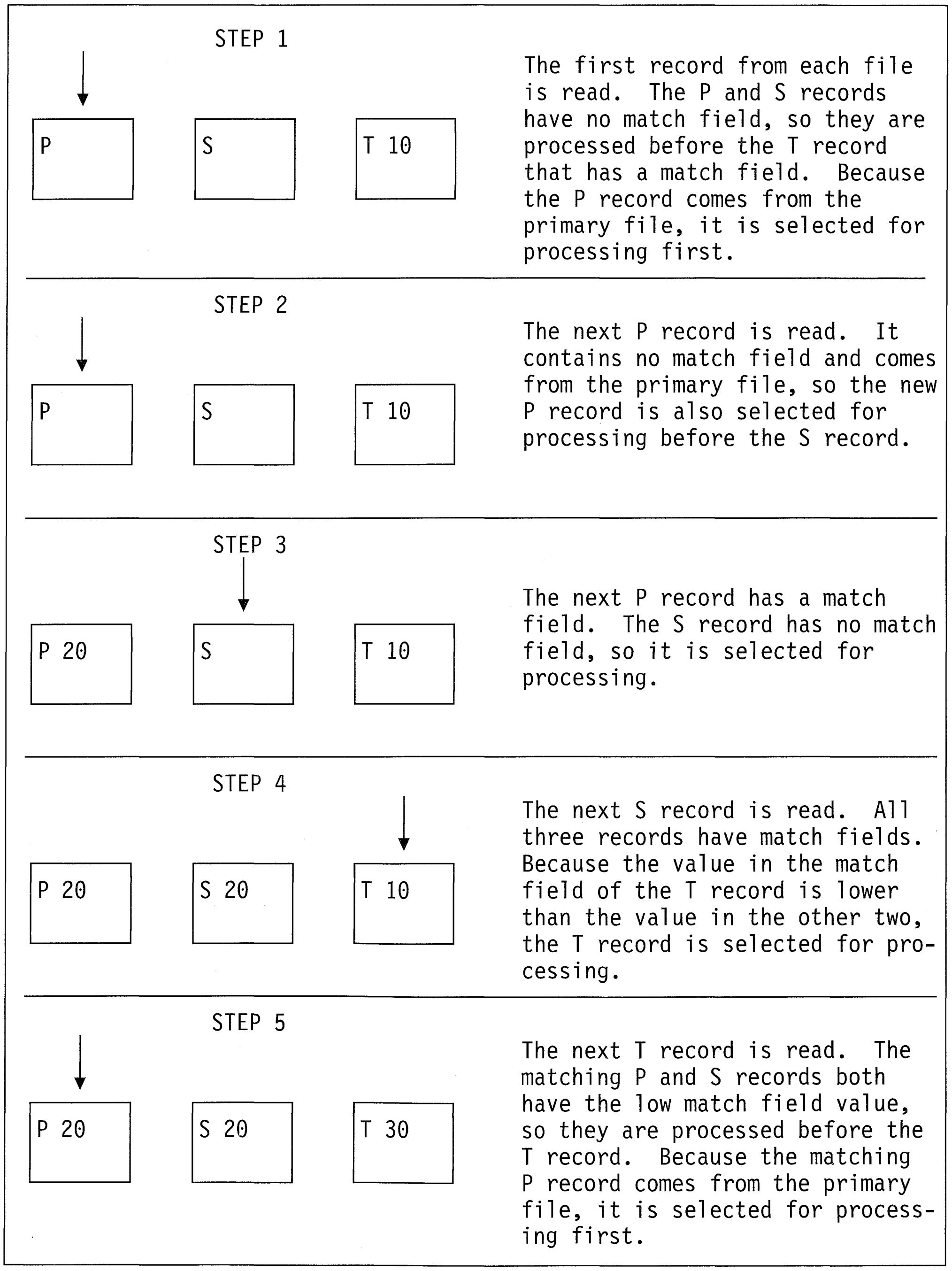
Figure 38. Normal Record Selection from Three Disk Files

Table 8 (Page 1 of 2). Normal Record Selection from Three Disk Files

Cycle	File Processed	Indicators On	Reason for Setting Indicator
1	PRIMARY	02	No match field specified
2	PRIMARY	02	No match field specified
3	FIRSTSEC	04	No match field specified
4	SECSEC	05	Second secondary low; no primary match
5	PRIMARY	01, MR	Primary matches first secondary
6	PRIMARY	01, MR	Primary matches first secondary
7	FIRSTSEC	03, MR	First secondary matches primary
8	FIRSTSEC	03	First secondary low; no primary match
9	FIRSTSEC	03	First secondary low; no primary match
10	SECSEC	05	Second secondary low; no primary match
11	PRIMARY	01	Primary low; no secondary match
12	PRIMARY	01, MR	Primary matches second secondary
13	PRIMARY	02	No match field specified
14	SECSEC	05, MR	Second secondary matches primary
15	SECSEC	05, MR	Second secondary matches primary
16	SECSEC	06	No match field specified
17	PRIMARY	01, MR	Primary matches both secondary files
18	FIRSTSEC	03, MR	First secondary matches primary
19	FIRSTSEC	04	No match field specified
20	SECSEC	05, MR	Second secondary matches primary
21	FIRSTSEC	03	First secondary low; no primary match

Table 8 (Page 2 of 2). Normal Record Selection from Three Disk Files

Cycle	File Processed	Indicators On	Reason for Setting Indicator
22	PRIMARY	01, MR	Primary matches both secondary files
23	FIRSTSEC	03, MR	First secondary matches primary
24	FIRSTSEC	02, MR	First secondary matches primary
25	SECSEC	05, MR	Second secondary matches primary
26	SECSEC	05, MR	Second secondary matches primary



STEP 1

The first record from each file is read. The P and S records have no match field, so they are processed before the T record that has a match field. Because the P record comes from the primary file, it is selected for processing first.

STEP 2

The next P record is read. It contains no match field and comes from the primary file, so the new P record is also selected for processing before the S record.

STEP 3

The next P record has a match field. The S record has no match field, so it is selected for processing.

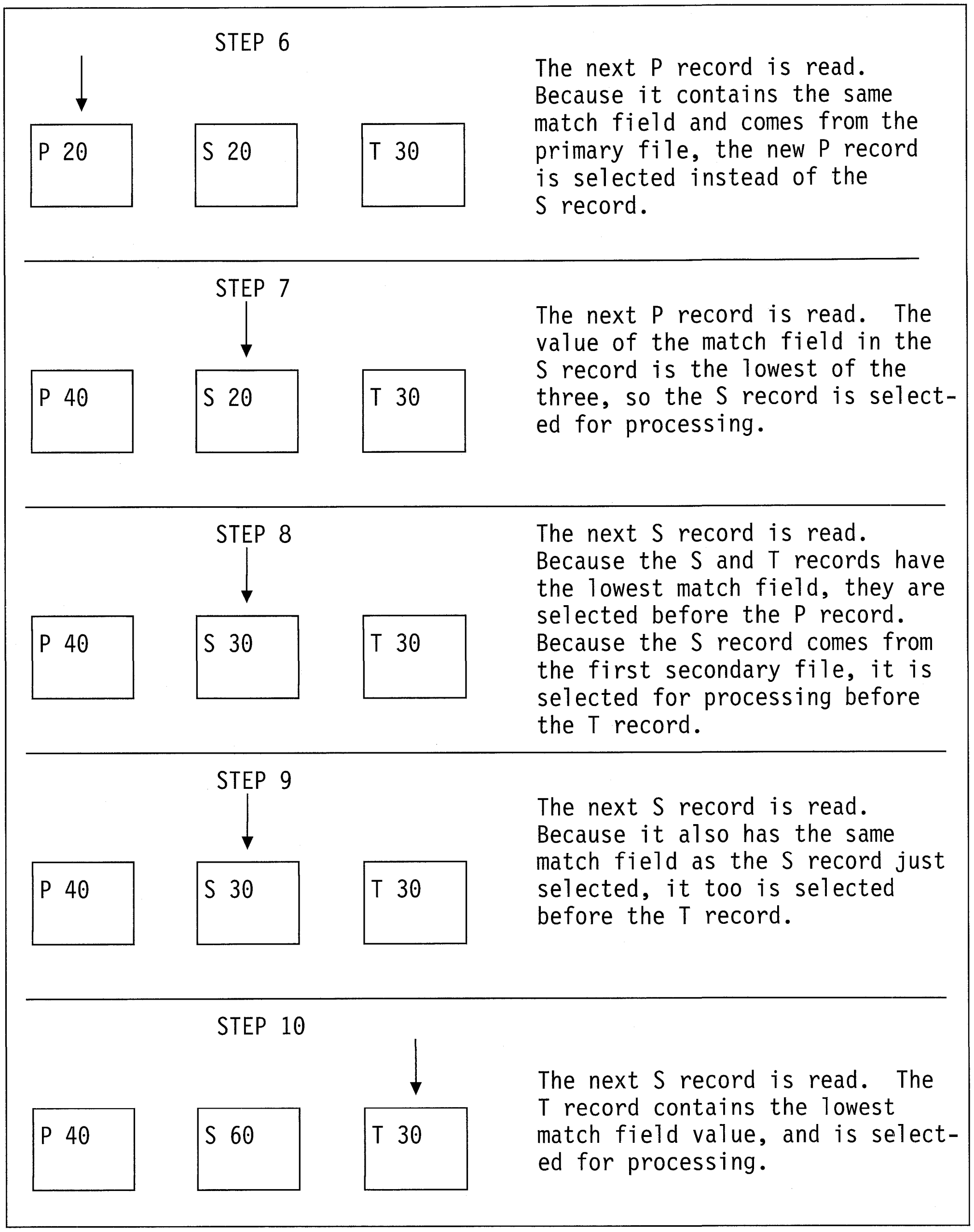
STEP 4

The next S record is read. All three records have match fields. Because the value in the match field of the T record is lower than the value in the other two, the T record is selected for processing.

STEP 5

The next T record is read. The matching P and S records both have the low match field value, so they are processed before the T record. Because the matching P record comes from the primary file, it is selected for processing first.

Figure 39 (Part 1 of 2). Normal Record Selection from Three Disk Files



The next P record is read. Because it contains the same match field and comes from the primary file, the new P record is selected instead of the S record.

The next P record is read. The value of the match field in the S record is the lowest of the three, so the S record is selected for processing.

The next S record is read. Because the S and T records have the lowest match field, they are selected before the P record. Because the S record comes from the first secondary file, it is selected for processing before the T record.

The next S record is read. Because it also has the same match field as the S record just selected, it too is selected before the T record.

The next S record is read. The T record contains the lowest match field value, and is selected for processing.

Figure 39 (Part 2 of 2). Normal Record Selection from Three Disk Files

Alternate Collating Sequence

Each character is represented internally by a hexadecimal value, which governs the order (ascending or descending sequence) of the characters and is known as the normal collating sequence. The alternate collating sequence function can be used to alter the normal collating sequence. This function also can be used to allow two or more characters to be considered equal.

Changing the Collating Sequence

Using an alternate collating sequence means modifying the collating sequence for character match fields (file selection) and character compares. You specify that an alternate collating sequence will be used by specifying the ALTSEQ keyword on the control specification. The calculation operations affected by the alternate collating sequence are ANDxx, COMP, CABxx, CASxx, DOU, DOUxx, DOW, DOWxx, IF, IFxx, ORxx, and WHENxx. This does not apply to graphic compare operations. LOOKUP and SORTA are affected only if you specify ALTSEQ(*EXT). The characters are not permanently changed by the alternate collating sequence, but are temporarily altered until the matching field or character compare operation is completed.

Changing the collating sequence does not affect the LOOKUP and SORTA operations (unless you specify ALTSEQ(*EXT)) or the hexadecimal values assigned to the figurative constants *HIVAL and *LOVAL. However, changing the collating sequence can affect the order of the values of *HIVAL and *LOVAL in the collating sequence. Therefore, if you specify an alternate collating sequence in your program and thereby cause a change in the order of the values of *HIVAL and *LOVAL, undesirable results may occur.

Using an External Collating Sequence

To specify that the values in the SRTSEQ and LANGID command parameters should be used to determine the alternate collating sequence, specify ALTSEQ(*EXT) on the control specificiation. For example, if ALTSEQ(*EXT) is used, and the CRTBNDRPG command specified SRTSEQ(*LANGIDSHR) LANGID(*JOBRUN), then when the program is run, the shared-weight table for the user running the program will be used as the alternate collating sequence.

Since the LOOKUP and SORTA operations are affected by the alternate collating sequence when ALTSEQ(*EXT) is specified, character compile-time arrays and tables are sequence-checked using the alternate collating sequence. If the actual collating sequence is not known until runtime, the array and table sequence cannot be checked until runtime. This means that you could get a runtime error saying that a compile-time array or table is out of sequence.

Pre-run arrays and tables are also sequence-checked using the alternate collating sequence when ALTSEQ(*EXT) is specified.

Specifying an Alternate Collating Sequence in Your Source

To specify that an alternate collating sequence is to be used, use the ALTSRC(*SRC) keyword on the control specification. If you use the **ALTSEQ, **CTDATA, and **FTRANS keywords, the sequence data may be entered in any sequence following the source records. If you do not use those keywords, the sequence data must follow the source records, and the file translation records but precede any compile-time array data.

If a character is to be inserted between two consecutive characters, you must specify every character that is altered by this insertion. For example, if the dollar sign ($) is to be inserted between A and B, specify the changes for character B onward.

See Appendix B, “EBCDIC Collating Sequence” on page 501 for the EBCDIC character set.

Formatting the Alternate Collating Sequence Records

The changes to the collating sequence must be transcribed into the correct record format so that they can be entered into the system. The alternate collating sequence must be formatted as follows:

Record Position	Entry
1-6	ALTSEQ (This indicates to the system that the normal sequence is being altered.)
7-10	Leave these positions blank.
11-12	Enter the hexadecimal value for the character whose normal sequence is being changed.
13-14	Enter the hexadecimal value of the character replacing the character whose normal sequence is being changed.
15-18 19-22 23-26 ... 77-80	All groups of four beginning with position 15 are used in the same manner as positions 11 through 14. In the first two positions of a group enter the hexadecimal value of the character to be replaced. In the last two positions enter the hexadecimal value of the character that replaces it.

The records that describe the alternate collating sequence must be preceded by a record with **ƀ (ƀ = blank) in positions 1 through 3. The remaining positions in this record can be used for comments.

File Translation

The file translation function translates any of the 8-bit codes used for characters into another 8-bit code. The use of file translation indicates one or both of the following:

- A character code used in the input data must be translated into the system code.
- The output data must be translated from the system code into a different code. The translation on input data occurs before any field selection has taken place. The translation on output data occurs after any editing taken place.

Remember the following when specifying file translation:

- File translation can be specified for data in array or table files (T in position 18 of the file description specifications).
- File translation can be used with data in combined, input, or update files that are translated at input and output time according to the file translation table provided. If file translation is used to translate data in an update file, each record must be written before the next record is read.
- For any I/O operation that specifies a search argument in factor 1 (such as CHAIN, READE, READPE, SETGT, or SETLL) for files accessed by keys, the search argument is translated before the file is accessed.
- If file translation is specified for both a record address file and the file being processed (if the file being processed is processed sequentially within limits), the records in the record address file are first translated according to the file translation specified for that file, and then the records in the file being processed are translated according to the file translation specified for that file.
- File translation applies only on a single byte basis.

- Every byte in the input and output record is translated

Specifying File Translation

To specify file translation, use the "FTRANS{(*NONE *SRC)}" keyword on the control specification. The translations must be transcribed into the correct record format for entry into the system. These records, called the file translation table records, must precede any alternate collating sequence records, or arrays and tables loaded at compile time. They must be preceded by a record with **ƀ (ƀ = blank) in positions 1 through 3 or **FTRANS in positions 1 through 8. The remaining positions in this record can be used for comments.

Translating One File or All Files

File translation table records must be formatted as follows:

Record Position	Entry
1-8 (to translate all files)	Enter *FILESƀƀ (ƀ represents a blank) to indicate that all files are to be translated. Complete the file translation table record beginning with positions 11 and 12. If *FILESƀƀ is specified, no other file translation table can be specified in the program.
1-8 (to translate a specific file)	Enter the name of the file to be translated. Complete the file translation table record beginning with positions 11 and 12. The *FILESƀƀ entry is *not* made in positions 1 through 8 when a specific file is to be translated.
9-10	Blank
11-12	Enter the hexadecimal value of the character to be translated from on input or to be translated to on output.
13-14	Enter the hexadecimal equivalent of the internal character the RPG IV language works with. It will replace the character in positions 11 and 12 on input and be replaced by the character in positions 11 and 12 on output.
15-18 19-22 23-26 ... 77-80	All groups of four beginning with position 15 are used in the same manner as positions 11 through 14. In the first two positions of a group, enter the hexadecimal value of the character to be replaced. In the last two positions, enter the hexadecimal value of the character that replaces it.

The first blank entry ends the record. There can be one or more records per file translation table. When multiple records are required in order to define the table, the same file name must be entered on all records. A change in file name is used to separate multiple translation tables. An *FILES record causes all files, including tables and arrays specified by a T in position 18 of the file description specifications, to be translated by the same table.

Translating More Than One File

If the same file translation table is needed for more than one file but not for all files, two types of records must be specified. The first record type specifies the file using the tables, and the second record type specifies the table. More than one record for each of these record types can be specified. A change in file names is used to separate multiple translation tables.

```
     HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     H* In this example all the files are translated
     H FTRANS
     FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++
     FFILE1     IP   F   10        DISK
     FFILE2     IS   F   10        DISK
     FFILE3     IS   F   10        DISK
     FFILE4     IS   F   10        DISK
**FTRANS
*FILES    81C182C283C384C4
```

```
     HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     H* In this example different translate tables are used and
     H* FILE3 is not translated.
     H FTRANS
     FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++
     FFILE1     IP   F   10        DISK
     FFILE2     IS   F   10        DISK
     FFILE3     IS   F   10        DISK
     FFILE4     IS   F   10        DISK
**FTRANS
FILE1     8182
FILE2     C1C2
FILE4     81C182C283C384C4
```

Specifying the Files

File translation table records must be formatted as follows:

Record Position	Entry
1-7	*EQUATE
8-10	Leave these positions blank.
11-80	Enter the name(s) of file(s) to be translated. If more than one file is to be translated, the file names must be separated by commas.

Additional file names are associated with the table until a file name not followed by a comma is encountered. A file name cannot be split between two records; a comma following a file name must be on the same record as the file name. You can create only one file translation table by using *EQUATE.

Specifying the Table

File translation table records must be formatted as follows:

Record Position	Entry
1-7	*EQUATE

Record Position	Entry
8-10	Leave these positions blank.
11-12	Enter the hexadecimal value of the character to be translated from on input or to be translated to on output.
13-14	Enter the hexadecimal equivalent of the internal character the RPG IV language works with. It will replace the character in positions 11 and 12 on input and be replaced by the character in positions 11 and 12 on output.
15-18 19-22 23-26 ... 77-80	All groups of four beginning with position 15 are used the same way as positions 11 through 14. In the first two positions of a group, enter the hexadecimal value of the character to be replaced. In the last two positions, enter the hexadecimal value of the character that replaces it.

The first blank record position ends the record. If the number of entries exceeds 80 positions, duplicate positions 1 through 10 on the next record and continue as before with the translation pairs in positions 11 through 80. All table records for one file must be kept together.

The records that describe the file translation tables must be preceded by a record with **ƀ (ƀ = blank) in positions 1 through 3 or with **FTRANS. The remaining positions in this record can be used for comments.

```
     HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     H* In this example several files are translated with the
     H* same translation table.  FILE2 is not translated.
     H FTRANS
     FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++
     FFILE1     IP   F   10        DISK
     FFILE2     IS   F   10        DISK
     FFILE3     IS   F   10        DISK
     FFILE4     IS   F   10        DISK
**FTRANS
*EQUATE   FILE1,FILE3,FILE4
*EQUATE   81C182C283C384C485C586C687C788C889C98ACA8BCB8CCC8DCD8ECE8F
*EQUATE   91D192D2
```

Data

This section provides information on the various aspects of RPG IV data. It describes:

- Data types and data formats
- Literals and named constants
- Data structures
- Arrays and tables
- Editing numeric fields
- Data initialization

Also included is information on default values, *HIVAL, and *LOVAL.

Chapter 7. Data Types and Data Formats

This chapter describes the data types supported by RPG IV and their special characteristics. The supported data types are

- Character
- Numeric
- Graphic
- Date
- Time
- Timestamp
- Basing Pointer
- Procedure Pointer

Character Data Type

Character data may be one or more bytes in length. Operation codes which operate on strings accept character data.

The default initialization value for nonindicator character fields is blanks.

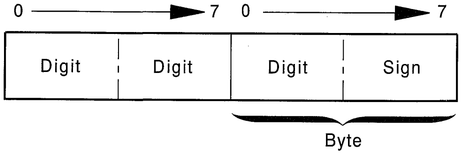
Indicators are a special type of character data. Indicator data consists of the RPG IV indicators and the field specified with the COMMIT keyword on the file description specification. Indicators are all one byte long and can only contain the character values '0' and '1'. The default value of indicators is '0'.

Numeric Data Type

Numeric data consists of any data defined as having zero or more decimal positions. The default initialization value for numeric fields is zeroes. Numeric data has three possible formats: zoned-decimal, packed-decimal, and binary.

Packed-Decimal Format

Packed-decimal format means that each byte of storage (except for the low order byte) can contain two decimal numbers. The low-order byte contains one digit in the leftmost portion and the sign (positive or negative) in the rightmost portion. The standard signs are used: hexadecimal F for positive numbers and hexadecimal D for negative numbers. The packed-decimal format looks like this:



The sign portion of the low-order byte indicates whether the numeric value represented in the digit portions is positive or negative. Figure 40 on page 105 shows what the decimal number 8191 looks like in packed-decimal format.

For a program-described file, you specify packed-decimal input, output, and array or table fields with the following entries:

Packed-decimal input field: Specify P in position 36 of the input specifications.

Packed-decimal output field: Specify P in position 52 of the output specifications. This position must be blank if editing is specified.

Packed-decimal array or table field: Specify P in position 40 of the definition specifications. The external format for compile-time arrays and tables cannot be packed-decimal format.

For an externally described file, the data format is specified in the data description specifications.

Determining the Digit Length of a Packed-Decimal Field

Use the following formula to find the length in digits of a packed-decimal field:

```
Number of digits = 2n – 1,

...where n = number of packed input record positions used.
```

This formula gives you the maximum number of digits you can represent in packed-decimal format; the upper limit is 30.

Packed fields can be up to 16 bytes long. Table 9 shows the packed equivalents for zoned-decimal fields up to 30 digits long:

Table 9. Packed Equivalents for Zoned-Decimal Fields up to 30 Digits Long

Zoned-Decimal Length in Digits	**Number of Bytes Used in Packed-Decimal Field**
1	1
2, 3	2
4, 5	3

. .
. .
. .
	.
	.
	.

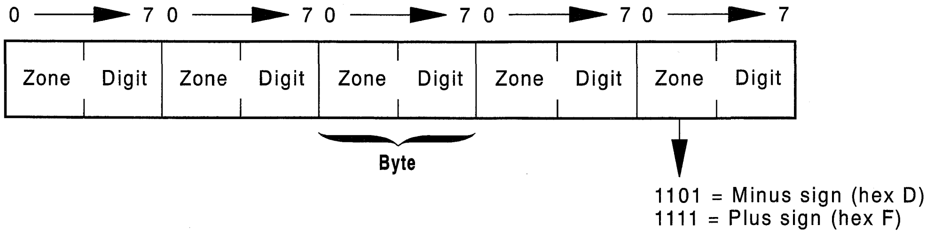
28, 29	15
30, 31	16

Note: Only 30 digits are allowed. If you use positional notation for 16-byte packed fields, you must use the PACKEVEN keyword or otherwise define the field as having 30 digits.

For example, an input field read in packed-decimal format has a length of five bytes (as specified on the input or data description specifications). The number of digits in this field equals 2(5) – 1 or 9. Therefore, when the field is used in the calculation specifications, the result field must be nine positions long. The "PACKEVEN" keyword on the definition specification can be used to indicate which of the two possible sizes you want when you specify a packed field using from and to positions rather than number of digits.

Zoned-Decimal Format

Zoned-decimal format means that each byte of storage can contain one digit or one character. In the zoned-decimal format, each byte of storage is divided into two portions: a 4-bit zone portion and a 4-bit digit portion. The zoned-decimal format looks like this:


0 → 7 0 → 7 0 → 7 0 → 7 0 → 7
Zone Digit Zone Digit Zone Digit Zone Digit Zone Digit
Byte
1101 = Minus sign (hex D)
1111 = Plus sign (hex F)

The zone portion of the low-order byte indicates the sign (positive or negative) of the decimal number. The standard signs are used: hexadecimal F for positive numbers and hexadecimal D for negative numbers. In zoned-decimal format, each digit in a decimal number includes a zone portion; however, only the low-order zone portion serves as the sign. Figure 40 on page 105 shows what the number 8191 looks like in zoned-decimal format.

You must consider the change in field length when coding the end position in positions 40 through 43 of the output specifications and the field is to be output in packed format. To find the length of the field after it has been packed, use the following formula:

$$\textbf{Field length} = \frac{n}{2} + 1$$

...where n = number of digits in the zoned decimal field.

(Any remainder from the division is ignored.)

For a program-described file, zoned-decimal format is specified by a blank in position 36 of the input specifications, in position 52 of the output specifications, or in position 40 of the definition specifications. For an externally described file, the data format is specified in position 35 of the data description specifications.

You can specify an alternative sign format for zoned-decimal format. In the alternative sign format, the numeric field is immediately preceded or followed by a + or – sign. A plus sign is a hexadecimal 4E, and a minus sign is a hexadecimal 60.

When an alternative sign format is specified, the field length (specified on the input specification) must include an additional position for the sign. For example, if a field is 5 digits long and the alternative sign format is specified, a field length of 6 positions must be specified.

Binary Format

Binary format means that the sign (positive or negative) is in the leftmost bit of the field and the integer value is in the remaining bits of the field. Positive numbers have a zero in the sign bit; negative numbers have a one in the sign bit and are in twos complement form. In binary format, each field must be either 2 or 4 bytes long.

Program-Described File

Every input field read in binary format is assigned a field length (number of digits) by the compiler. A length of 4 is assigned to a 2-byte binary field; a length of 9 is assigned to a 4-byte binary field, if the field is not defined elsewhere in the program. Because of these length restrictions, the highest decimal value that can be assigned to a 2-byte binary field is 9999 and the highest decimal value that can be assigned to a 4-byte binary field is 999 999 999. In general, a binary field of n digits can have a maximum value of n 9s. This discussion assumes zero decimal positions.

For program-described files, specify binary input, binary output, and binary array or table fields with the following entries:

- *Binary input field:* Specify B in position 36 of the input specifications.
- *Binary output field:* Specify B in position 52 of the output specifications. This position must be blank if editing is specified.

  The length of a field to be written in binary format cannot exceed nine digits. If the length of the field is from one to four digits, the compiler assumes a binary field length of 2 bytes. If the length of the field is from five to nine digits, the compiler assumes a binary field length of 4 bytes.

  Because a 2-byte field in binary format is converted by the compiler to a decimal field with 1 to 4 digits, the input value may be too large. If it is, the leftmost digit of the number is dropped. For example, if a four digit binary input field has a binary value of hexadecimal 6000. The compiler converts this to 24 576 in decimal. The 2 is dropped and the result is 4576.
- *Binary array or table field:* Specify B in position 40 of the definition specifications. The external format for compile-time arrays and tables must not be binary. format.

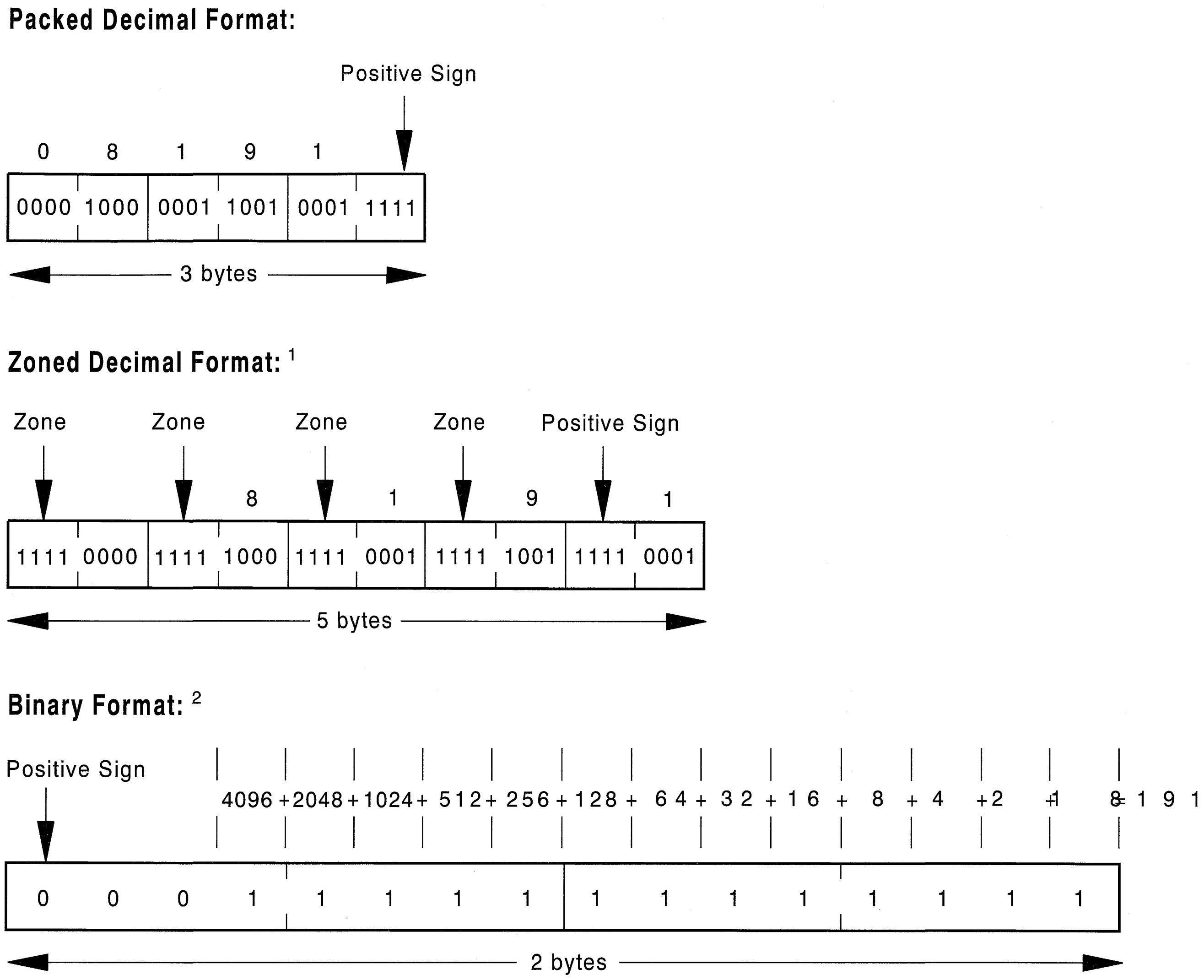
Externally Described File

For an externally-described file, the data format is specified in position 35 of the data description specifications. The number of digits in the field is exactly the same as the length in the DDS description. For example, if you define a binary field in your DDS specification as having 7 digits and 0 decimal positions, the RPG IV compiler handles the data like this:

1. The field is defined as a 4-byte binary field in the input specification
2. A Packed(7,0) field is generated for the field in the RPG IV program.

If you want to retain the complete binary field information, redefine the field as a binary subfield in a data structure or as a binary stand-alone field.

Figure 40 on page 105 shows what the decimal number 8191 looks like in various formats.



[1] If 8191 is read into storage as a zoned-decimal field, it occupies 4 bytes. If it is converted to packed-decimal format, it occupies 3 bytes. When it is converted back to zoned-decimal format, it occupies 5 bytes.

[2] To obtain the numeric value of a positive binary number add the values of the bits that are on (1), do not include the sign bit. To obtain the numeric value of a negative binary number, add the values of the bits that are off (0) plus one (the sign bit is not included).

Figure 40. Binary, Packed, and Zoned-Decimal Representation of the Number 8191

Date Data

Date fields have a predetermined size and format. They can be defined on the definition specification. Leading and trailing zeros are required for all date data.

Date constants or variables used in comparisons or assignments do not have to be in the same format or use the same separators. Also dates used for I/O operations such as input fields, output fields or key fields are also converted (if required) to the necessary format for the operation..

The default internal format for date variables is *ISO. This default internal format can be overridden globally by the control specification keyword DATFMT and individually by the definition specification keyword DATFMT.

The hierarchy used when determining the internal date format and separator for a date field is

1. From the DATFMT keyword specified on the definition specification
2. From the DATFMT keyword specified on the control specification
3. *ISO

There are two kinds of date data formats, 2-digit year formats and 4-digit year. For 2-digit year formats, years in the range 1940 to 2039 can be represented. This leads to the possibility of a date overflow condition occurring when converting from a 4-digit year format to a 2-digit year format.

The following table lists the formats for date data.

Table 10. Date formats for Date data type

Format name	Description	Format	Length	Example
*MDY	Month/Day/Year	mm/dd/yy	8	01/15/91
*DMY	Day/Month/Year	dd/mm/yy	8	15/01/91
*YMD	Year/Month/Day	yy/mm/dd	8	91/01/15
*JUL	Julian	yy/ddd	6	91/015
*ISO	International Standards Organization	yyyy-mm-dd	10	1991-01-15
*USA	IBM USA Standard	mm/dd/yyyy	10	01/15/1991
*EUR	IBM European Standard	dd.mm.yyyy	10	15.01.1991
*JIS	Japanese Industrial Standard Christian Era	yyyy-mm-dd	10	1991-01-15

The following table lists the *LOVAL, *HIVAL, and default values for all the date formats.

Table 11. Date Values

Format name	Description	*LOVAL	*HIVAL	Default Value
*MDY	Month/Day/Year	01/01/40	12/31/39	01/01/01
*DMY	Day/Month/Year	01/01/40	31/12/39	01/01/01
*YMD	Year/Month/Day	40/01/01	39/12/31	01/01/01
*JUL	Julian	40/001	39/365	01/001
*ISO	International Standards Organization	0001-01-01	9999-12-31	0001-01-01
*USA	IBM USA Standard	01/01/0001	12/31/9999	01/01/0001
*EUR	IBM European Standard	01.01.0001	31.12.9999	01.01.0001
*JIS	Japanese Industrial Standard Christian Era	0001-01-01	9999-12-31	0001-01-01

Time Data

Time fields have a predetermined size and format. They can be defined on the definition specification. Leading and trailing zeros are required for all time data.

Time constants or variables used in comparisons or assignments do not have to be in the same format or use the same separators. Also, times used for I/O operations such as input fields, output fields or key fields are also converted (if required) to the necessary format for the operation.

The default internal format for time variables is *ISO. This default internal format can be overridden globally by the control specification keyword TIMFMT and individually by the definition specification keyword TIMFMT.

The hierarchy used when determining the internal time format and separator for a time field is

1. From the TIMFMT keyword specified on the definition specification
2. From the TIMFMT keyword specified on the control specification
3. *ISO

The following table lists the formats for time data.

Table 12. Time formats for Time data type

Format name	Description	Format	Length	Example
*HMS	Hours:Minutes:Seconds	hh:mm:ss	8	14:00:00
*ISO	International Standards Organization	hh.mm.ss	8	14.00.00
*USA	IBM USA Standard. AM and PM can be any mix of upper and lower case.	hh:mm AM or hh:mm PM	8	02:00 PM
*EUR	IBM European Standard	hh.mm.ss	8	14.00.00
*JIS	Japanese Industrial Standard Christian Era	hh:mm:ss	8	14:00:00

The following table lists the *LOVAL, *HIVAL, and default values for all the date formats.

Table 13. Time Values

Format name	Description	*LOVAL	*HIVAL	Default Value
*HMS	Hours:Minutes:Seconds	00:00:00	24:00:00	00:00:00
*ISO	International Standards Organization	00.00.00	24.00.00	00.00.00
*USA	IBM USA Standard. AM and PM can be any mix of upper and lower case.	00:00 AM	12:00 AM	00:00 AM
*EUR	IBM European Standard	00.00.00	24.00.00	00.00.00
*JIS	Japanese Industrial Standard Christian Era	00:00:00	24:00:00	00:00:00

Timestamp Data

Timestamp fields have a predetermined size and format. They can be defined on the definition specification. Timestamp data must be in the format

```
yyyy-mm-dd-hh.mm.ss.mmmmmm (length 26).
```

Microseconds (.mmmmmm) are optional for timestamp literals and if not provided will be padded on the right with zeroes. Leading zeros are required for all timestamp data.

The default initialization value for a timestamp is midnight of January 1, 0001 (0001-01-01-00.00.00.000000). The *HIVAL value for a timestamp is 9999-12-31-24.00.00.000000. Similarly, the *LOVAL value for timestamp is 0001-01-01-00.00.00.00000.

Graphic Data Type

The graphic data type is a character string where each character is represented by 2 bytes. Fields defined as graphic data do not contain shift-out (SO) or shift-in (SI) characters. The default initialization value for graphic data is X'4040'. The value of *HIVAL is X'FFFF' and the value of *LOVAL is X'0000' The difference between single byte and graphic data is shown in the following figure:

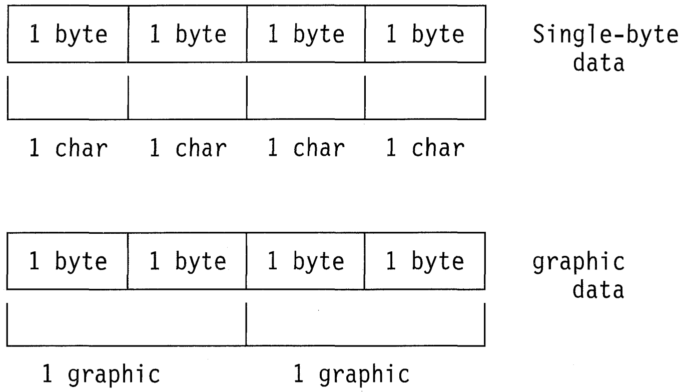


Figure 41. Comparing Single-byte and graphic data

The length of a graphic field, in bytes, is two times the number of graphic characters in the field.

If you add a record to the database file and graphic fields are not specified for output, the ILE RPG/400 compiler will place double-byte blanks in the fields for output. The following conditions will result in blanks being placed in your output fields:

- The fields are not specified for output on the output specification.
- Conditioning indicators are not satisfied for the field.

Basing Pointer Data Type

Basing pointers are used to point to data in storage. The storage is accessed by defining a field as based on a particular basing pointer variable and setting the basing pointer field to point to the required storage location. Basing pointers can be defined on the definition specification with the “BASED(basing_pointer_name)” keyword.

The length of the basing pointer field must be 16 bytes long and must be aligned on a 16 byte boundary. This requirement for boundary alignment can cause a pointer subfield of a data structure not to follow the preceding field directly, and can cause multiple occurrence data structures to have noncontiguous occurrences. The default initialization value for basing pointers is *NULL.

Examples

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
 *
 *  Define a based data structure, array and field.
 *  If PTR1 is not defined, it will be implicitly defined
 *  by the compiler.
 *
 *  Note that before these based fields or structures can be used,
 *  the basing pointer must be set to point to the correct storage
 *  location.
 *
D DSbased         DS                  BASED(PTR1)
D    Field1                    1 16A
D     Field2                     2
D
D ARRAY           S             20A   DIM(12) BASED(PRT2)
D
D Temp_fld        S               *   BASED(PRT3)
D
D PTR2                            *   INZ
D PTR3                            *   INZ(*NULL)
```

Figure 42. Defining based structures and fields

Figure 43 illustrates the use of pointers, based structures and system APIs. This program does the following:

1. Receives the Library and File name you wish to process
2. Creates a User space using the QUSCRTUS API
3. Calls an API (QUSLMBR) to list the members in the requested file
4. Gets a pointer to the User space using the QUSPTRUS API
5. Displays a message with the number of members and the name of the first and last member in the file

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
D NAME            S              20    INZ('LISTSPACE YOURLIB  ')
D ATTRIBUTE       S              10    INZ('ABC')
D INIT_SIZE       S               9B 0 INZ(9999999)
D AUTHORITY       S              10    INZ('*CHANGE')
D TEXT            S              50    INZ('File member space')
D SPACE           DS                   BASED(PTR)
D SP1                         32767
 *
 * ARR is used with OFFSET to set the pointer to array
 *
D ARR                             1    OVERLAY(SP1) DIM(32767)
 *
 * Offset is pointing to start of array
 *
D OFFSET                          9B 0 OVERLAY(SP1:125)
 *
 * Size has number of member names retrieved
 *
D SIZE                            9B 0 OVERLAY(SP1:133)
D MBRPTR          S               *
D MBRARR          S              10    BASED(MBRPTR) DIM(32767)
D PTR             S               *
D LIB_FILE        S              20
D  LIB            S              10
D  FILE           S              10
D WHAT            S              10    INZ('*ALL      ')
D X               S               7  0 INZ(1)
D OVERRIDE        S               1    INZ('1')
D FIRST_LAST      S              50    INZ('    members, +
D                                      First =          , +
D                                      Last =          ')
```

Figure 43 (Part 1 of 2). Example of using pointers and based structures with an API

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CL0N01++++++++++++++Opcode(E)+Extended Factor 2+++++++++++++++++++++++++++++++++
 *
 * Receive Library and file you want to process
 *
C     *ENTRY        PLIST
C                   PARM                    LIB
C                   PARM                    FILE
C                   EVAL      LIB_FILE = LIB + FILE
 *
 * Create the user space
 *
C                   CALL      'QUSCRTUS'                           99
C                   PARM                    NAME
C                   PARM                    ATTRIBUTE
C                   PARM                    INIT_SIZE
C                   PARM      ' '           INIT_VALUE        1
C                   PARM                    AUTHORITY
C                   PARM                    TEXT
 *
 * CAll the API to list the members in the requested file
 *
C                   CALL      'QUSLMBR'
C                   PARM                    NAME
C                   PARM      'MBRL0100'    MBR_LIST          8
C                   PARM                    LIB_FILE
C                   PARM                    WHAT
C                   PARM                    OVERRIDE
 *
 * Get a pointer to the user-space
 *
C                   CALL      'QUSPTRUS'
C                   PARM                    NAME
C                   PARM                    PTR
 *
 * Set the basing pointer for the member array
 *
C                   EVAL      MBRPTR = %ADDR(ARR(OFFSET))
C                   Move      size          charsize          3
C                   EVAL      %Subst(First_Last:1:3)  = charsize
C                   EVAL      %subst(First_last:23:10) = mbrarr(1)
C                   EVAL      %subst(First_last:41:10) = mbrarr(size)
C     FIRST_LAST    DSPLY
C                   EVAL      *INLR = '1'
```

Figure 43 (Part 2 of 2). Example of using pointers and based structures with an API

Procedure Pointer Data Type

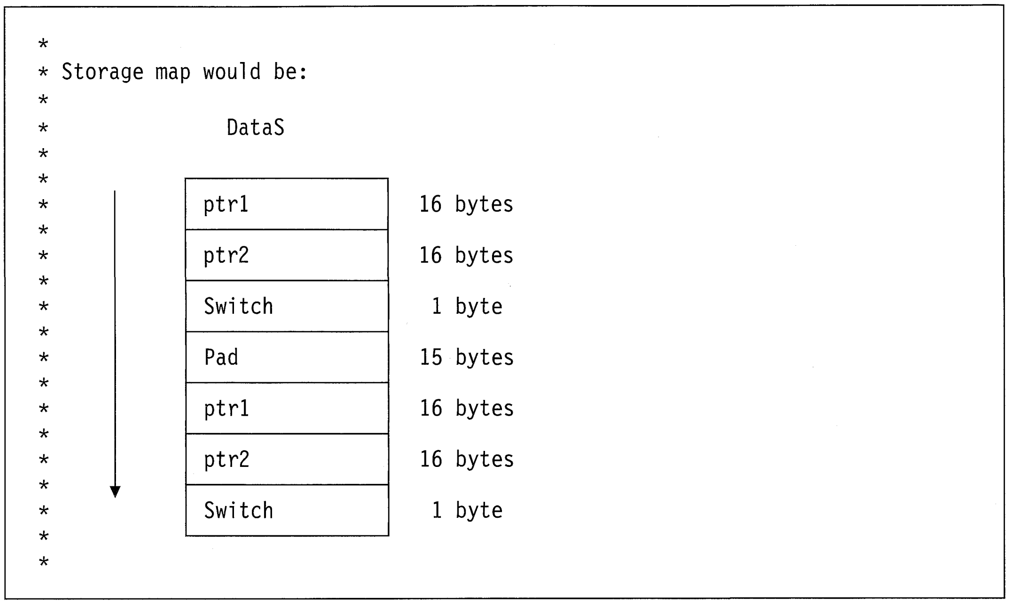
Procedure pointers are used to point to procedures or functions. A procedure pointer points to an entry point that is bound into the program. Procedure pointers are defined on the definition specification.

The length of the procedure pointer field must be 16 bytes long and must be aligned on a 16 byte boundary. This requirement for boundary alignment can cause a pointer subfield of a data structure not to follow the preceding field directly, and can cause multiple occurrence data structures to have noncontiguous occurrences. The default initialization value for procedure pointers is *NULL.

Examples

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
 *
 * Define a basing pointer field and initialize to the address of the
 * data structure My_Struct.
 *
D  My_struct       DS
D   My_array                      10    DIM(50)
D
D Ptr1             S              16*   INZ(%ADDR(My_Struct))
 *
 * Or equivalently, defaults to length 16 if length not defined
 *
D Ptr1             S                *   INZ(%ADDR(My_Struct))
 *
 * Define a procedure pointer field and initialize to NULL
 *
D Ptr1             S              16*   PROCPTR INZ(*NULL)
 *
 * Define a procedure pointer field and initialize to the address
 * of the procedure My_Proc.
 *
D Ptr1             S              16*   PROCPTR INZ(%PADDR(My_Proc))
 *
 * Define pointers in a multiple occurence data structure and map out
 * the storage.
 *
DDataS            DS                    OCCURS(2)
D ptr1                              *
D ptr2                              *
D Switch                           1A
```

Figure 44 (Part 1 of 2). Defining pointers



```
 *
 * Storage map would be:
 *
 *              DataS
 *
 *
 *                  ptr1        16 bytes
 *
 *                  ptr2        16 bytes
 *
 *                  Switch       1 byte
 *
 *                  Pad         15 bytes
 *
 *                  ptr1        16 bytes
 *
 *                  ptr2        16 bytes
 *
 *                  Switch       1 byte
 *
 *
```

Figure 44 (Part 2 of 2). Defining pointers

Unsupported Database Data-Types

The ILE RPG/400 compiler tolerates null-capable fields and variable-length fields.

Null Values/Null Capable Fields

Null-capable fields containing null values in a database file can be read into your ILE RPG/400 program if you specify the *YES value on the ALWNULL keyword of the CRTRPGMOD or CRTBNDRPG commands. Currently, null value support only applies to externally described input-only files (files with no addition specified on the file specification).

When an externally described file contains null-capable fields and *NO is specified on the ALWNULL keyword, the following conditions apply:

- A record containing null values retrieved from an input or update file will cause a data mapping error and an error message will be issued.
- Data in the record is not accessible and none of the fields in the record can be updated with the values from the input record containing null values.
- The ILE RPG/400 compiler is not able to place null values in null-capable fields for updating or adding a record. If you want to place null values in null-capable fields, you can use SQL/400 or other products which have full support of null values.

When an externally described input-only file contains null-capable fields and *YES is specified on the ALWNULL keyword, the following conditions apply:

- When a record is retrieved from a database file and there are some fields containing null values in the record, database default values for the null-capable fields will be placed into those fields containing null values. The default value will be the user defined DDS defaults or system defaults.

- Control-level indicators, match-field entries and field indicators are not allowed on an input specification if the input field is a null-capable field from an externally described input-only file.
- Keyed operations are not allowed when factor 1 on a keyed input calculation operation corresponds to a null-capable key field in an externally described input-only file.
- Sequential-within-limits processing is not allowed when a file contains null-capable key fields.

Note: For a program-described file, a null value in the record always causes a data mapping error, regardless of the value specified on the ALWNULL keyword.

Variable-Length Fields

By specifying *VARCHAR (for variable length character fields) or *VARGRAPHIC (for variable length graphic fields) on the CVTOPT keyword of the CRTRPGMOD or CRTBNDRPG commands, the ILE RPG/400 compiler will internally define variable-length fields from an externally described file or data structure as an ILE RPG/400 fixed-length character field. When *VARCHAR or *VARGRAPHIC is not specified, variable-length fields are ignored and inaccessible in ILE RPG/400 programs. For more information, see the CVTOPT keyword description in the *ILE RPG/400 Programmer's Guide*.

The following conditions apply when *VARCHAR or *VARGRAPHIC is specified on the CRTRPGMOD or CRTBNDRPG command:

- If a variable-length field is extracted from an externally described file or an externally described data structure, it is declared in an ILE RPG/400 program as a fixed-length character field.
- For single-byte character fields, the length of the declared ILE RPG/400 field is the length of the DDS field plus 2 bytes.
- For DBCS-graphic data fields, the length of the declared RPG/400 field is two times the length of the DDS field plus 2 bytes.
- The two extra bytes in the ILE RPG/400 field contain a binary number which represents the current length of the variable-length field. Figure 45 on page 115 shows the ILE RPG/400 field length of variable-length fields.
- For variable-length graphic fields defined as fixed-length character fields, the length is measured in double bytes.

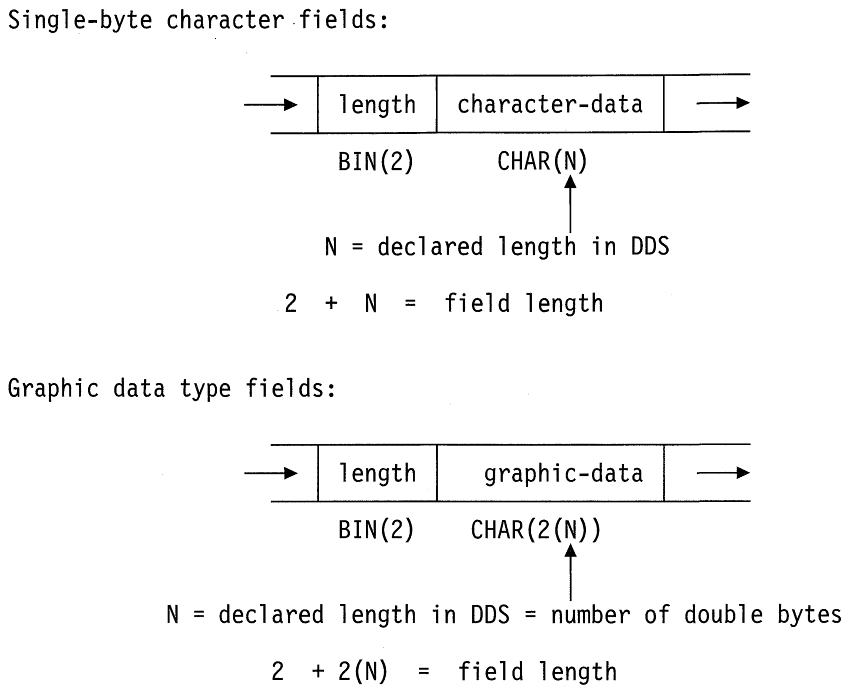

Single-byte character fields:
length
character-data
BIN(2)
CHAR(N)
N = declared length in DDS
2 + N = field length
Graphic data type fields:
length
graphic-data
BIN(2)
CHAR(2(N))
N = declared length in DDS = number of double bytes
2 + 2(N) = field length

Figure 45. ILE RPG/400 Field Length of Variable-Length Fields

- Your ILE RPG/400 program can perform any valid character calculation operations on the declared fixed-length field. However, because of the structure of the field, the first two bytes of the field must contain valid binary data. An I/O exception error will occur for an output operation if the first two bytes of the field contain invalid field length data.
- Control-level indicators, match field entries, and field indicators are not allowed on an input specification if the input field is a variable-length field from an externally described input file.
- Sequential-within-limits processing is not allowed when a file contains variable-length key fields.
- Keyed operations are not allowed when factor 1 of a keyed operation corresponds to a variable-length key field in an externally described file.
- If you choose to selectively output certain fields in a record and the variable-length field is not specified on the output specification, or if the variable-length field is ignored in the ILE RPG/400 program, the ILE RPG/400 compiler will place a default value in the output buffer of the newly-added record. The default is 0 in the first two bytes and blanks in all of the remaining bytes.
- If you want to change variable-length fields, ensure that the current field length is correct. One way to do this is:
  1. Define a data structure with the variable-length field name as a subfield name.
  2. Define a 4-digit binary subfield overlaying the beginning of the field, and define an N-byte character subfield overlaying the field starting at position 3.
  3. Update the field.

Alternatively, you can move another variable-length field left-aligned into the field. An example of how to change a variable-length field in an ILE RPG/400 program follows.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
A*
A*   File MASTER contains a variable length field
A*
AAN01N02N03T.Name++++++Rlen++TDpBLinPosFunctions++++++++++++++++++++
A*
A          R REC
A            FLDVAR       100          VARLEN

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
F*
F*   Externally described file name is MASTER.
F*   Compile the RPG/400 program with CVTOPT(*VARCHAR).
F*
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
F*
FMASTER    UF   E             DISK

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
D*
D*   FLDVAR is a variable-length field defined in DDS with
D*   a DDS length of 100.  Notice that the RPG field length
D*   is 102.
D*
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
D                 DS
D FLDVAR                 1    102
D   FLDLEN                      4B 0 OVERLAY(FLDVAR:1)
D   FLDCHR                    100    OVERLAY(FLDVAR:3)

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C* A character value is moved to the variable length field FLDCHR.
C* After the CHECKR operation, FLDLEN has a value of 5.
C                   READ      MASTER                                 LR
C                   MOVEL     'SALES'       FLDCHR
C     ' '           CHECKR    FLDCHR        FLDLEN
C  NLR              UPDAT     REC
```

Figure 46. Changing a Variable-Length Field in an ILE RPG/400 Program

If variable-length graphic fields are required, you can code a 2-byte binary field to hold the length, and a 2(N) length subfield to hold the data portion of the field.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
D*
D*   The variable-length graphic field VGRAPH is declared in the
D*   DDS as length 3.  This means the maximum length of the field
D*   is 3 double bytes, or 6 bytes.  The total length of the field,
D*   counting the length portion, is 8 bytes.
D*
D*   Compile the ILE RPG/400 program with CVTOPT(*VARGRAPHIC).
D*
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
D                 DS
DVGRAPH                          8
D  VLEN                          4B 0 OVERLAY(VGRAPH:1)
D  VDATA                         3G   OVERLAY(VGRAPH:3)
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..*
C*
C*   Assume GRPH is a fixed length graphic field of length 2
C*   double bytes.  Copy GRPH into VGRAPH and set the length of
C*   VGRAPH to 2.
C*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C                   MOVEL     GRPH          VDATA
C                   Z-ADD     2             VLEN
```

Figure 47. Using a Variable-Length Graphic Field in an ILE RPG/400 Program

Error Handling for Database Data Mapping Errors

For any input or output operation, a data mapping error will cause a severe error message to be issued. For blocked output, if one or more of the records in the block contains data mapping errors and the file is closed before reaching the end of the block, a severe error message is issued and a system dump is created.

Chapter 8. Literals and Named Constants

Literals and named constants are types of constants. Constants can be specified in factor 1 or factor 2 of certain operations and in the constant field of output specifications. Constants can also be used with keywords on the definition specification.

Literals

A literal is a self-defining constant that can be referred to in a program. A literal can belong to any of the RPG IV data types.

Character Literals

The following are the rules for specifying a character literal:

- Any combination of characters can be used in a character literal. This includes DBCS characters. DBCS characters must be enclosed by shift-out and shift-in characters and must be an even number of bytes. Embedded blanks are valid.
- Character literals must be enclosed in apostrophes (').
- An apostrophe required as part of a literal is represented by two apostrophes. For example, the literal O'CLOCK is coded as 'O''CLOCK'.
- Character literals are compatible only with character data

Hexadecimal Literals

The following are the rules for specifying a hexadecimal literal:

- Hexadecimal literals take the form:

  ```
  X'x1x2...xn'
  ```

- Where X'x1x2...xn' can only contain the characters A-F, a-f, and 0-9.
- The literal coded between the apostrophes must be of even length.
- Each pair of characters defines a single byte.
- Hexadecimal literals are allowed anywhere that character literals are supported except as factor 2 of ENDSR and as edit words.
- Except when used in the bit operations BITON, BITOFF, and TESTB, a hexadecimal literal has the same meaning as the corresponding character literal. For the bit operations, factor 2 may contain a hexadecimal literal representing 1 byte. The rules and meaning are the same for hexadecimal literals as for character fields.
- If the hexadecimal literal contains the hexadecimal value for a single quote, it does not have to be specified twice, unlike character literals. For example, the literal

  ```
  A'B
  ```

  is specified as

  ```
  'A''B'
  ```

  but the hexadecimal version is X'C17DC2' not X'C17D7DC2'.
- Hexadecimal literals are compatible only with character data

Numeric Literals

The following are the rules for specifying a numeric literal:

- A numeric literal consists of any combination of the digits 0 through 9. A decimal point or a sign can be included.
- The sign (+ or -), if present, must be the leftmost character. An unsigned literal is treated as a positive number.
- Blanks cannot appear in a numeric literal.
- Numeric literals are not enclosed in apostrophes (').
- Numeric literals are used in the same way as a numeric field, except that values cannot be assigned to numeric literals.
- The decimal separator may be either a comma or a period

Date Literals

Date literals take the form D'xxxxxx' where:

- D indicates that the literal is of type date
- xxxxxx is a valid date in the format specified on the control specification
- xxxxxx is enclosed by apostrophes

Time Literals

Time literals take the form T'xxxxxx' where:

- T indicates that the literal is of type time
- xxxxxx is a valid time in the format specified on the control specification
- xxxxxx is enclosed by apostrophes

Timestamp Literals

Timestamp literals take the form Z'yyyy-mm-dd-hh.mm.ss.mmmmmm' where:

- Z indicates that the literal is of type timestamp
- yyyy-mm-dd is a valid date (year-month-day)
- hh.mm.ss.mmmmmm is a valid time (hours.minutes.seconds.microseconds)
- yyyy-mm-dd-hh.mm.ss.mmmmmm is enclosed by apostrophes
- Microsecond are optional and if not specified will default to zeros

Graphic Literals

Graphic literals take the form G'oK1K2i' where:

- G indicates that the literal is of type graphic
- o is a shift-out character
- K1K2 is an even number of bytes and does not contain a shift-out or shift-in character
- i is a shift-in character
- oK1K2i is enclosed by apostrophes

Named Constants

A named constant is a symbolic name assigned to a literal. Named constants are defined on definition specifications. The value of a named constant follows the rules specified for literals.

Named Constants

You can give a name to a constant. This name represents a specific value which cannot be changed when the program is running.

Rules for Named Constants

- Named constants can be specified in factor 1, factor 2, and extended factor 2 on the calculation specifications, as parameters to keywords on the control specification, as parameters to built-in functions, and in the Field Name, Constant, or Edit Word fields in the output specifications. They can also be used as array indexes and as the format name in a WORKSTN output specification or with keywords on the definition specification.
- Numeric named constants have no predefined precision. Actual precision is defined by the context that is specified.
- The named constant can be defined anywhere on the definition specifications.

Example of Defining a Named Constant

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.................................Keywords+++++++++++++++++++++++++++++
 *
 *  Define a date field and initialize it to the 3rd of September
 *  1988.
 *
D DateField      S                D   INZ(D'1988-09-03')
 *
 *  Define a binary 9,5 field and initialize it to 0.
 *
D BIN9_5         S               9B 5 INZ
 *
 *  Define a named constant whose value is the lower case alphabet.
 *
D Lower          C                    CONST('abcdefghijklmnop-
D                                     qrstuvwxyz')
 *
 *  Define a named constant without explicit use of the keyword CONST.
 *
D Upper          C                    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

Figure 48. Defining named constants

Figurative Constants

The figurative constants *BLANK/*BLANKS, *ZERO/*ZEROS, *HIVAL, *LOVAL, *NULL, *ALL'x..', *ALLG'oK1K2i' *ALLX'x1..' and *ON/*OFF are implied literals that can be specified without a length, because the implied length and decimal positions of a figurative constant are the same as those of the associated field. (For exceptions, see the following section, “Rules for Figurative Constants” on page 122.)

Figurative constants can be specified in factor 1 and factor 2 of the calculation specifications. The following shows the reserved words and implied values for figurative constants:

Reserved Words	Implied Values
*BLANK/*BLANKS	All blanks. Valid only for character or graphic fields
*ZERO/*ZEROS	Character/numeric fields: All zeros.
*HIVAL	Character or graphic fields: The highest collating character for the system (hexadecimal FFs). Numeric fields: All nines with a positive sign. Date, time and timestamp fields: See “Date Data” on page 105, “Time Data” on page 107 and “Timestamp Data” on page 108 for *HIVAL values for date, time, and timestamp data.
*LOVAL	Character or graphic fields: The lowest collating character for the system (hexadecimal zeros). Numeric fields: All nines with a negative sign. Date, time and timestamp fields: See “Date Data” on page 105, “Time Data” on page 107 and “Timestamp Data” on page 108 for *LOVAL values for date, time, and timestamp data.
*ALL'x..'	Character/numeric fields: Character string x . . is cyclically repeated to a length equal to the associated field. If the field is a numeric field, all characters within the string must be numeric (0 through 9). No sign or decimal point can be specified when *ALL'x..' is used as a numeric constant.
*ALLG'oK1K2i'	Graphic fields: The graphic string K1K2 is cyclically repeated to a length equal to the associated field.
*ALLX'x1..'	Character fields: The hexadecimal literal X'x1..' is cyclically repeated to a length equal to the associated field.
*NULL	A null value valid for basing pointers or procedure pointers
*ON/*OFF	*ON is all ones. *OFF is all zeros. Both are only valid for character fields.

Rules for Figurative Constants

Remember the following rules when using figurative constants:

- MOVE and MOVEL operations allow moving a character figurative constant to a numeric field. The figurative constant is first expanded as a zoned numeric with the size of the numeric field, converted to packed or binary numeric if needed, and then stored in the target numeric field. The digit portion of each character in the constant must be valid. If not, a decimal data error will occur.
- Figurative constants are considered elementary items. Except for MOVEA, figurative constants act like a field if used in conjunction with an array. For example: MOVE *ALL'XYZ' ARR.

  If ARR has 4-byte character elements, then each element will contain 'XYZX'.
- MOVEA is considered to be a special case. The constant is generated with a length equal to the portion of the array specified. For example:
  - MOVEA *BLANK ARR(X)

    Beginning with element X, the remainder of ARR will contain blanks.
  - MOVEA *ALL'XYZ' ARR(X)

    ARR has 4-byte character elements. Element boundaries are ignored, as is always the case with character MOVEA operations. Beginning with element X, the remainder of the array will contain 'XYZXYZXYZ...'.

Note that the results of MOVEA are different from those of the MOVE example above.

- After figurative constants are set/reset to their appropriate length, their normal collating sequence can be altered if an alternate collating sequence is specified.
- The move operations MOVE and MOVEL produce the same result when moving the figurative constants *ALL'x..', *ALLG'oK1K2i', and *ALLX'x1..'. The string is cyclically repeated character by character (starting on the left) until the length of the associated field is the same as the length of the string.
- Figurative constants can be used in compare operations as long as one of the factors is not a figurative constant.
- The figurative constants, *BLANK/*BLANKS, are moved as zeros to a numeric field in a MOVE operation.

Chapter 9. Data Structures

The RPG IV program allows you to define an area in storage and the layout of the fields, called subfields, within the area. This area in storage is called a **data structure**. You define a data structure by specifying DS in positions 24 through 25 on a definition specification.

You can use a data structure to:

- Define the same internal area multiple times using different data formats
- Operate on a field and change its contents
- Operate on all the subfields as a group using the name of the data structure
- Define a data structure and its subfields in the same way a record is defined
- Define multiple occurrences of a set of data
- Group non-contiguous data into contiguous internal storage locations.

In addition, there are three special data structures, each with a specific purpose:

- A data area data structure (identified by a U in position 23 of the definition specification)
- A file information data structure (identified by the keyword INFDS on a file description specifications)
- A program-status data structure (identified by an S in position 23 of the definition specification).

Data structures can be program-described or externally-described.

A program-described data structure is identified by a blank in position 22 of the definition specification. The subfield definitions for a program-described data structure must immediately follow the data structure definiton.

An externally-described data structure, identified by an E in position 22 of the definition specification, has subfield descriptions contained in an externally-described file. At compile time, the ILE RPG/400 compiler uses the external name to locate and extract the external description of the data structure subfields. You specify the name of the external description either in positions 7 through 21, or as a parameter for the keyword EXTNAME.

An external subfield name can be renamed in the program using the keyword EXTFLD. The keyword PREFIX can be used to add a prefix to the external subfield names. Additional subfields can be added to an externally described data structure by specifying program-described subfields immediately after the list of external subfields.

Special Data Structures

Special data structures include:

- Data area data structures
- File information data structures (INFDS)
- Program-status data structures.

Data Area Data Structure

A data area data structure, identified by a U in position 23 of the definition specification, indicates to the RPG IV program that it should read in and lock the data area of the same name at program initialization and should write out and unlock the same data area at the end of the program. Data area data structures, as in all other data structures, have the type character. A data area read into a data area data structure must also be character. The data area and data area data structure must have the same name unless you rename the data area within the RPG IV program by using the *DTAARA DEFINE operation code or the DTAARA keyword.

You can specify the data area operations (IN, OUT, and UNLOCK) for a data area that is implicitly read in and written out. Before you use a data area data structure with these operations, you must specify that data area in the result field of the *DTAARA DEFINE operation or with the DTAARA keyword.

A data area data structure cannot be specified in the result field of a PARM operation in the *ENTRY PLIST.

If you specify blanks for the data area data structure (positions 7 through 21 of the definition specification that contains a U in position 23), the RPG IV program uses the local data area. To provide a name for the local data area, use the *DTAARA DEFINE operation, with *LDA in factor 2 and the name in the result field or DTAARA(*LDA) on the definition specification.

File Information Data Structure

You can specify a file information data structure (defined by the keyword INFDS on a file description specifications) for each file in the program. This provides you with status information on the file exception/error that occurred. The file information data structure name must be unique for each file. A file information data structure contains predefined subfields that provide information on the file exception/error that occurred. For a discussion of file information data structures and their subfields, see “File Information Data Structure” on page 61.

Program-Status Data Structure

A program-status data structure, identified by an S in position 23 of the definition specification, provides program exception/error information to the program. For a discussion of program-status data structures and their predefined subfields, see “Program Status Data Structure” on page 78.

Data Structure Examples

The following examples show various uses for data structures and how to define them.

Example	Description
Figure 49	Using a data structure to subdivide a field
Figure 50 on page 128	Using a data structure to group fields
Figure 51 on page 129	Data structure with absolute and length notation
Figure 52 on page 129	Rename and initialize an externally described data structure
Figure 53 on page 130	Using PREFIX to rename all fields in an external data structure
Figure 54 on page 130	Defining a *LDA data area data structure
Figure 55 on page 131	Defining a multiple occurrence data structure
Figure 56 on page 131	Using data area data structures (1)
Figure 57 on page 132	Using data area data structures (2)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
 * Use length notation to define the data structure subfields.
 * You can refer to the entire data structure by using Partno, or by
 * using the individual subfields Manufactr, Drug, Strength or Count.
 *
D Partno          DS
D  Manufactr                     4
D  Drug                          6
D  Strength                      3
D  Count                         3  0
D

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..................................
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr......
 *
 *  Records in program described file FILEIN contain a field, Partno,
 *  which needs to be subdivided for processing in this program.
 *  To achieve this, the field Partno is described as a data structure
 *  using the above Definition specification
 *
IFILEIN    NS  01   1 CA    2 CB
I                                  3   18  Partno
I                                 19   29  Name
I                                 30   40  Patno
```

Figure 49. Using a Data structure to subdivide a field

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.................................Keywords+++++++++++++++++++++++++++++
 *
 *  When you use a data structure to group fields, fields from
 *  non-adjacent locations on the input record can be made to occupy
 *  adjacent internal locations. The area can then be referred to by
 *  the data structure name or individual subfield name.
 *
D Partkey          DS
D  Location                        4
D  Partno                          8
D  Type                            4
D

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC................................
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr......
 *
 *  Fields from program described file TRANSACTN need to be
 *  compared to the field retrieved from an Item_Master file
 *
ITRANSACTN NS  01   1 C1    2 C2
I                                     3   10  Partno
I                                    11   16 0Quantity
I                                    17   20  Type
I                                    21   21  Code
I                                    22   25  Location
I

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  Use the data structure name Partkey, to compare to the field
 *  Item_Nbr
 *
C                    :
C     Partkey        IFEQ      Item_Nbr                                 99
C                    :
C*
```

Figure 50. Using a data structure to group fields

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.................................Keywords+++++++++++++++++++++++++++++
 *
 *  Define a program described data structure called FRED
 *  The data structure is composed of 5 fields:
 *   1.   An array with element length 10 and dimension 70(Field1)
 *   2.   A field of length 30 (Field2)
 *   3/4. Divide Field2 in 2 equal length fields (Field3 and Field4)
 *   5.   Define a binary field over the 3rd field
 *  Note the indentation to improve readability
 *
 *
 *  Absolute notation:
 *
 *  The compiler will determine the array element length (Field1)
 *  by dividing the total length (700) by the dimension (70)
 *
D FRED            DS
D  Field1                 1    700    DIM(70)
D  Field2               701    730
D   Field3              701    715
D    Field5             701    704B 2
D   Field4              716    730
 *
 *  Length notation:
 *
 *  The OVERLAY keyword is used to subdivide Field2
 *
D FRED            DS
D  Field1                       10    DIM(70)
D  Field2                       30
D   Field3                      15    OVERLAY(Field2)
D    Field5                      4B 2 OVERLAY(Field3)
D   Field4                      15    OVERLAY(Field2:16)
```

Figure 51. Data structure with absolute and length notation

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.................................Keywords+++++++++++++++++++++++++++++
 *
 *  Define an externally described data structure with internal name
 *  FRED and external name EXTDS and rename field CUST to CUSTNAME
 *  Initialize CUSTNAME to 'GEORGE' and PRICE to 1234.89.
 *  Assign to subfield ITMARR (defined in the external decription as a
 *  100 byte character field) the DIM keyword
 *
D Fred          E DS                  EXTNAME(EXTDS)
D   CUSTNAME    E                     EXTFLD(CUST) INZ('GEORGE')
D   PRICE       E                     INZ(1234.89)
D   ITMARR      E                     DIM(10)
```

Figure 52. Rename and initialize an externally described data structure

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
D
D extds1        E DS                  EXTNAME (CUSTDATA)
D                                     PREFIX (CU_)
D   Name        E                     INZ ('Joe''s Garage')
D   Custnum     E                     EXTFLD (NUMBER)
D
 *
 *  The previous data structure will expand as follows:
 *  -- All externally described fields are included in the data
 *     structure
 *  -- Renamed subfields keep their new names
 *  -- Subfields that are not renamed are prefixed with the
 *     prefix string
 *
 *  Expanded data structure:
 *
D EXTDS1        E DS
D   CU_NAME     E               20A   EXTFLD (NAME)
D                                     INZ ('Joe''s Garage')
D   CU_ADDR     E               50A   EXTFLD (ADDR)
D   CUSTNUM     E                9S0  EXTFLD (NUMBER)
D   CU_SALESMN  E                7P0  EXTFLD (SALESMN)
```

Figure 53. Using PREFIX to rename all fields in an external data structure

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
 *  Define a data area data structure based on the *LDA. The information
 *  in the data structure is composed of 2 fields.  The first is a
 *  graphic array (30) of graphic character length 10 and the
 *  second is a Date field.  The Date field is in *ISO format.
 *  The data structure is shown in absolute and length notation.
 *
 *
 *  Absolute notation:
 *
D DS_IO          UDS
D  Graf_fld               1    600G   DIM(30)
D  Date_fld             601    610D
 *
 *  Length notation:
 *
D DS_IO          UDS                  DTAARA(*LDA)
D  Graf_fld                     10G   DIM(30)
D  Date_fld                       D
```

*Figure 54. Defining a *LDA data area data structure*

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
 *  Define a Multiple Occurrence data structure of 20 elements with:
 *  -- 3 fields of character 20
 *  -- A 4th field of character 10 which overlaps the 2nd
 *     field starting at the second position.
 *
 *  Named constant 'twenty' is used to define the occurrence
 *
 *  Absolute notation (using begin/end positions)
 *
D twenty          C                   CONST(20)
D
DDataStruct       DS                  OCCURS (twenty)
D field1                  1     20
D field2                 21     40
D  field21               22     31
D field3                 41     60
 *
 *  Mixture of absolute and length notation
 *
D DataStruct      DS                  OCCURS(twenty)
D  field1                       20
D  field2                       20
D   field21              22     31
D  field3                41     60
```

Figure 55. Defining a multiple occurrence data structure

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DFTNAME(Program1)
H
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
 *  This program uses a data area data structure to accumulate
 *  a series of totals.
 *
D  Totals        UDS
D    Tot_amount                  8 2
D    Tot_gross                  10 2
D    Tot_netto                  10 2

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2++++++++++++++++++++++++++++++++++++++
 *
C                   :
C                   EVAL      Tot_amount = Tot_amount + amount
C                   EVAL      Tot_gross  = Tot_gross  + gross
C                   EVAL      Tot_netto  = Tot_netto  + netto
```

Figure 56. Using data area data structures (program 1)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DFTNAME(Program2)
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
 *  This program processes the totals accumulated in Program1.
 *  Program2 then uses the total in the subfields to do calculations.
 *
D  Totals          UDS
D    Tot_amount                    8 2
D    Tot_gross                    10 2
D    Tot_netto                    10 2

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
C                   :
C                   EVAL      *IN91 =  (Amount2 <> Tot_amount)
C                   EVAL      *IN92 =  (Gross2 <> Tot_gross)
C                   EVAL      *IN93 =  (Netto2 <> Tot_Netto)
C                   :
```

Figure 57. Using data area data structures (program 2)

Chapter 10. Using Arrays and Tables

Arrays and tables are both collections of data fields (elements) of the same:

- Field length
- Data type
  - Character
  - Numeric
  - Date
  - Time
  - Timestamp
  - Graphic
  - Basing Pointer
  - Procedure Pointer
- Format
- Number of decimal positions (if numeric)

Arrays and tables differ in that:

- You can refer to a specific array element by its position
- You cannot refer to specific table elements by their position
- An array name by itself refers to all elements in the array
- A table name always refers to the element found in the last "LOOKUP (Look Up a Table or Array Element)" operation

The next section describes how to code an array, how to specify the initial values of the array elements, how to change the values of an array, and the special considerations for using an array. The section after next describes the same information for tables.

Arrays

There are three types of arrays:

- The *run-time array* is loaded by your program while it is running.
- The *compile-time array* is loaded when your program is created. The initial data becomes a permanent part of your program.
- The *prerun-time array* is loaded from an array file when your program begins running, before any input, calculation, or output operations are processed.

The essentials of defining and loading an array are described for a run-time array. For defining and loading compile-time and prerun-time arrays you use these essentials and some additional specifications.

Array Name and Index

You refer to an entire array using the array name alone. You refer to the individual elements of an array using (1) the array name, followed by (2) a left parenthesis, followed by (3) an index, followed by (4) a right parenthesis -- for example: AR(IND). The index indicates the position of the element within the array (starting from 1) and is either a number or a field containing a number.

The following rules apply when you specify an array name and index:

- The array name must be a unique symbolic name.

- The index must be a numeric field or constant greater than zero and with zero decimal positions
- If the array is specified within an expression in the extended factor 2 field, the index may be an expression returning a numeric value with zero decimal positions
- At run time, if your program refers to an array using an index with a value that is zero, negative, or greater than the number of elements in the array, then the error/exception routine takes control of your program.

The Essential Array Specifications

You define an array on a definition specification. Here are the essential specifications for all arrays:

- Specify the array name in positions 7 through 21
- Specify the number of entries in the array using the DIM keyword
- Specify length, data format, and decimal positions as you would any scalar fields. You may specify explicit From- and To-position entries (if defining a subfield), or an explicit Length-entry; or you may define the array attributes using the LIKE keyword; or the attributes may be specified elsewhere in the program.
- If you need to specify a sort sequence, use the ASCEND or DESCEND keywords.

Figure 58 shows an example of the essential array specifications.

Coding a Run-Time Array

If you make no further specifications beyond the essential array specifications, you have defined a *run-time array*. Note that the keywords ALT, CTDATA, EXTFMT, FROMFILE, PERRCD, and TOFILE cannot be used for a run-time array.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
DARC              S              3A   DIM(12)
```

Figure 58. The Essential Array Specifications to Define a Run-Time Array

Loading a Run-Time Array

You can assign initial values for a run-time array using the INZ keyword on the definition specification. You can also assign initial values for a run-time array through input or calculation specifications. This second method may also be used to put data into other types of arrays.

For example, you may use the calculation specifications for the MOVE operation to put 0 in each element of an array (or in selected elements).

Using the input specifications, you may fill an array with the data from a file. The following sections provide more details on retrieving this data from the records of a file.

Note: Date and time runtime data must be in the same format and use the same separators as the date or time array being loaded.

Loading a Run-Time Array in One Source Record

If the array information is contained in one record, the information can occupy consecutive positions in the record or it can be scattered throughout the record.

If the array elements are consecutive on the input record, the array can be loaded with a single input specification. Figure 59 shows the specifications for loading an array, INPARR, of six elements (12 characters each) from a single record from the file ARRFILE.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
DINPARR           S            12A    DIM(6)
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IARRFILE   AA  01
I                                 1   72  INPARR
```

Figure 59. Using a Run-Time Array with Consecutive Elements

If the array elements are scattered throughout the record, they can be defined and loaded one at a time, with one element described on a specification line. Figure 60 shows the specifications for loading an array, ARRX, of six elements with 12 characters each, from a single record from file ARRFILE; a blank separates each of the elements from the others.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
DARRX             S            12A    DIM(6)
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IARRFILE   AA  01
I                                 1   12  ARRX(1)
I                                14   25  ARRX(2)
I                                27   38  ARRX(3)
I                                40   51  ARRX(4)
I                                53   64  ARRX(5)
I                                66   77  ARRX(6)
```

Figure 60. Defining a Run-Time Array with Scattered Elements

Loading a Run-Time Array Using Multiple Source Records

If the array information is in more than one record, you may use various methods to load the array. The method to use depends on the size of the array and whether or not the array elements are consecutive in the input records. The RPG IV program processes one record at a time. Therefore the entire array is not processed until all the records containing the array information are read and the information is moved into the array fields. It may be necessary to suppress calculation and output operations until the entire array is read into the program.

Sequencing Run-Time Arrays

Run-time arrays are not sequence checked. If you process a SORTA (sort an array) operation, the array is sorted into the sequence specified on the definition specification (the ASCEND or DESCEND keywords) defining the array. If the sequence is not specified, the array is sorted into ascending sequence. When the high (positions 71 and 72 of the calculation specifications) or low (positions 73 and 74 of the calculation specifications) indicators are used in the LOOKUP operation, the array sequence must be specified.

Coding a Compile-Time Array

A compile-time array is specified using the essential array specifications plus the keyword CTDATA. You can specify the number of array entries in an input record using the PERRCD keyword on the definition specification. If you do not specify the PERRCD keyword, the number of entries defaults to 1. You can specify the external data format using the "EXTFMT(code)" keyword. See the specifications in Figure 61 on page 137.

Loading a Compile-Time Array

For a *compile-time array*, enter array source data into records in the program source member. If you use the **ALTSEQ, **CTDATA, and **FTRANS keywords, the array data may be entered in anywhere following the source records. If you do not use those keywords, the array data must follow the source records, and any alternate collating sequence or file translation records in the order in which the compile-time arrays and tables were defined on the definition specifications. This data is loaded into the array when the program is compiled. Until the program is recompiled with new data, the array will always initially have the same values each time you call the program unless the previous call ended with LR off.

Arrays can be described separately or in alternating format (with the ALT keyword). Alternating format means that the elements of one array are intermixed on the input record with elements of another array.

Rules for Array Source Records

The rules for array source records are:

- The first array entry for each record must begin in position 1.
- All elements must be the same length and follow each other with no intervening spaces
- An entire record need not be filled with entries. If it is not, blanks or comments can be included after the entries (see Figure 61 on page 137).
- If the number of elements in the array as specified on the definition specification is greater than the number of entries provided, the remaining elements are filled with the default values for the data type specified.

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....*
      DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++
      DARC                              3A   DIM(12) PERRCD(5) CTDATA
**CTDATA ARC
48K16343J64044HComments can be placed here
12648A47349K346Comments can be placed here
50B125         Comments can be placed here
```

48K	163	43J	640	44H	126	48A	473	49K	346	50B	125

This is the compile-time array, ARC.

Figure 61. Array Source Record with Comments

- Each record, except the last, must contain the number of entries specified with the PERRCD keyword on the definition specifications. In the last record, unused entries must be blank and comments can be included after the unused entries.
- Each entry must be contained entirely on one record. An entry cannot be split between two records; therefore, the length of a single entry is limited to the maximum length of 100 characters (size of source record). If arrays are used and are described in alternating format, corresponding elements must be on the same record; together they cannot exceed 100 characters.
- For date and time compile-time arrays the data must be in the same format and use the same separators as the date or time array being loaded.
- Array data may be specified in one of two ways:
  1. **CTDATA arrayname: The data for the array may be specified anywhere in the compile-time data section.
  2. **b: (b=blank) The data for the arrays must be specified in the same order in which they are specified in the Definition specifications.

  Only one of these techniques may be used in one program.
- Arrays can be in ascending(ASCEND keyword), descending (DESCEND keyword), or no sequence (no keyword specified).
- For ascending or descending character arrays when ALTSEQ(*EXT) is specified on the control specification, the alternate collating sequence is used for the sequence checking. If the actual collating sequence is not known at compile time (for example, if SRTSEQ(*JOBRUN) is specified on the command parameter) the alternate collating sequence table will be retrieved at runtime and the checking will occur during initialization at *INIT. Otherwise, the checking will be done at compile time.
- Graphic arrays will be sorted by hexadecimal values, regardless of the alternate collating sequence.
- If L or R is specified on the EXTFMT keyword on the definition specification, each element must include the sign (+ or -). For example, an array with an element size of 2 with L specified would require 3 positions in the source data (+37-38+52-63).
- Graphic data must be enclosed in shift-out and shift-in characters. If several elements of graphic data are included in a single record (without intervening nongraphic data) only one set of shift-out and shift-in characters is required for the record. If a graphic array is defined in alternating format with a nongraphic array, the shift-in and shift-out characters must surround the graphic data. If

two graphic arrays are defined in alternating format, only one set of shift-in and shift-out characters is required for each record.

Coding a Prerun-Time Array

On the definition specifications, in addition to the essential array specifications, you can specify the name of the file with the array input data, using the FROMFILE keyword. You can use the TOFILE keyword to specify the name of a file to which the array is written at the end of the program. If the file is a combined file (specified by a C in position 17 of the file description specifications), the parameter for the FROMFILE and TOFILE keywords must be the same. You can use the PERRCD keyword to specify the number of elements per input record.

On the EXTFMT keyword, specify a P if the array data is in packed format, B if the data is in binary format, L to indicate a sign on the left of a data element, or R to indicate a sign on the right of a data element.

Specify a T in position 18 of the file description specifications for the file with the array input data.

Compare the coding of two prerun-time arrays, a compile-time array, and a run-time array in Figure 62 on page 139.

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....*
     HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     H DATFMT(*USA) TIMFMT(*HMS)
     DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++
     D* Run-time array.  ARI has 10 elements of type date. They are
     D* initialized to September 15, 1994.  This is in month, day,
     D* year format using a slash as a separator as defined on the
     D* control specification.
     DARI              S               D   DIM(10) INZ(D'09/15/1994')
     D* Compile-time arrays in alternating format.  Both arrays have
     D* eight elements (three elements per record).  ARC is a character
     D* array of length 15, and ARD is a time array with a predefined
     D* length of 8.
     DARC              S             15    DIM(8) PERRCD(3)
     D                                     CTDATA
     DARD              S               T   DIM(8) ALT(ARC)
     D*
     D* Prerun-time array.  ARE, which is to be read from file DISKIN,
     D* has 250 character elements (12 elements per record).  Each
     D* element is five positions long.  The size of each record
     D* is 60 (5*12). The elements are arranged in ascending sequence.
     DARE              S              5A   DIM(250) PERRCD(12) ASCEND
     D                                     FROMFILE(DISKIN)
     D*
     D*
     D* Prerun-time array specified as a combined file.  ARH is written
     D* back to the same file from which it is read when the program
     D* ends normally with LR on.  ARH has 250  character elements
     D* (12 elements per record).  Each elements is five positions long.
     D* The elements are arranged in ascending sequence.
     DARH              S              5A   DIM(250) PERRCD(12) ASCEND
     D                                     FROMFILE(DISKOUT)
     D                                     TOFILE(DISKOUT)
**CTDATA ARC
Toronto        12:15:00Winnipeg       13:23:00Calgary        15:44:00
Sydney         17:24:30Edmonton       21:33:00Saskatoon      08:40:00
Regina         12:33:00Vancouver      13:20:00
```

Figure 62. Definition Specifications for Different Types of Arrays

Figure 62 shows the definition specifications required for several types of arrays.

For compile-time arrays, the TOFILE keyword can be used to specify a file to which the array is to be written when the program ends with LR on. For prerun-time arrays, the ALT keyword can be used to specify arrays in alternating format.

Loading a Prerun-Time Array

For a *prerun-time array*, enter array input data into a file. The file must be a sequential program described file. When you call a program, but before any input, calculation, or output operations are processed the array is loaded with initial values from the file. By modifying this file, you can alter the array's initial values on the next call to the program, without recompiling the program. The file is read in arrival sequence. The rules for prerun-time array data are the same as for compile-time array data, except there are no restrictions on the length of each record. See "Rules for Array Source Records" on page 136.

Sequence Checking for Character Arrays

When sequence checking for character arrays occurs depends on whether and how ALTSEQ has been specified and whether the array is compile time or prerun time. The following table indicates when sequence checking occurs.

Control Specification Entry	ALTSEQ Used for SORTA, LOOKUP and Sequence Checking	When Sequence Checked for Compile Time Array	When Sequence Checked for Prerun Time Array
ALTSEQ(*NONE)	No	Compile time	Run time
ALTSEQ(*SRC)	No	Compile time	Run time
ALTSEQ(*EXT) (known at compile time)	Yes	Compile time	Run time
ALTSEQ(*EXT) (known only at run time)	Yes	Run time	Run time

Note: For compatibility with RPG III, SORTA and LOOKUP do not use the alternate collating sequence with ALTSEQ(*SRC). If you want these operations to be performed using the alternate collating sequence, you can define a table on the system (object type *TBL), containing your alternate sequence. Then you can change ALTSEQ(*SRC) to ALTSEQ(*EXT) on your control specification and specify the name of your table on the SRTSEQ parameter of the create command.

Initializing Arrays

Run-Time Arrays

To initialize each element in a run-time array to the same value, specify the INZ keyword on the definition specification. If the array is defined as a data structure subfield, the normal rules for data structure initialization overlap apply (the initialization is done in the order that the fields are declared within the data structure).

Compile-Time and Prerun-Time Arrays

The INZ keyword cannot be specified for a compile-time or prerun-time array, because their initial values are assigned to them through other means (compile-time data or data from an input file). If a compile-time or prerun-time array appears in a globally initialized data structure, it is not included in the global initialization.

Note:

Compile-time arrays are initialized in the order in which the data is declared after the program, and prerun-time arrays are initialized in the order of declaration of their initialization files, regardless of the order in which these arrays are declared in the data structure. Pre-run time arrays are initialized after compile-time arrays.

If a subfield initialization overlaps a compile-time or prerun-time array, the initialization of the array takes precedence; that is, the array is initialized after the subfield, regardless of the order in which fields are declared within the data structure.

Defining Related Arrays

You can load two compile-time arrays or two prerun-time arrays in *alternating format* by using the ALT keyword on the definition of the alternating array. You specify the name of the primary array as the parameter for the ALT keyword. The records for storing the data for such arrays have the first element of the first array followed by the first element of the second array, the second element of the first array followed by the second element of the second array, the third element of the first array followed by the third element of the second array, and so on. Corresponding elements must appear on the same record. The PERRCD keyword on the main array definition specifies the number of corresponding pairs per record, each pair of elements counting as a single entry. You can specify EXTFMT on both the main and alternating array.

Figure 63 shows two arrays, ARRA and ARRB, in alternating format.

ARRA (Part Number)	ARRB (Unit Cost)
345126	373
38A437	498
39K143	1297
40B125	93
41C023	3998
42D893	87
43K823	349
44H111	697
45P673	898
46C732	47587

Arrays ARRA and ARRB can be described as two separate array files or as one array file in alternating format.

Figure 63. Arrays in Alternating and Nonalternating Format

The records for ARRA and ARRB look like the records below when described as two separate array files.

This record contains ARRA entries in positions 1 through 60.

ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry
1.....	7.....	13....	19....	25....	31....	37....	43....	49....	55....

Figure 64. Arrays Records for Two Separate Array Files

This record contains ARRB entries in positions 1 through 50.

ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry
1.....	6.....	11....	16....	21....	26....	31....	36....	41....	46....

Figure 65. Arrays Records for One Array File

The records for ARRA and ARRB look like the records below when described as one array file in alternating format. The first record contains ARRA and ARRB entries in alternating format in positions 1 through 55. The second record contains ARRA and ARRB entries in alternating format in positions 1 through 55.

ARRA entry	ARRB entry	ARRA entry	ARRB entry	ARRA entry	ARRB entry	ARRA entry	ARRB entry	ARRA entry	ARRB entry
1.....	1.....	7.....	6.....	13....	11....	19....	16....	25....	21....

Figure 66. Arrays Records for One Array File in Alternating Format

```
*....+....1....+....2....+....3....+....4....+....5....+....6....+....*
     DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++
     DARRA             S              6A   DIM(6) PERRCD(1) CTDATA
     DARRB             S              5  0 DIM(6) ALT(ARRA)
     DARRGRAPHIC       S              3G   DIM(2) PERRCD(2) CTDATA
     DARRC             S              3A   DIM(2) ALT(ARRGRAPHIC)
     DARRGRAPH1        S              3G   DIM(2) PERRCD(2) CTDATA
     DARRGRAPH2        S              3G   DIM(2) ALT(ARRGRAPH1)
**CTDATA ARRA
345126  373
38A437  498
39K143 1297
40B125   93
41C023 3998
42D893   87
**CTDATA ARRGRAPHIC
ok1k2k3iabcok4k5k6iabc
**CTDATA ARRGRAPH1
ok1k2k3k4k5k6k1k2k3k4k5k6i
```

Searching Arrays

The LOOKUP operation can be used to search arrays. See "LOOKUP (Look Up a Table or Array Element)" on page 385 for a description of the LOOKUP operation.

Searching an Array Without an Index

When searching an array without an index, use the status (on or off) of the resulting indicators to determine whether a particular element is present in the array. Searching an array without an index can be used for validity checking of input data to determine if a field is in a list of array elements. Generally, an equal LOOKUP is requested.

In factor 1 in the calculation specifications, specify the search argument (data for which you want to find a match in the array named) and place the array name factor 2.

In factor 2 specify the name of the array to be searched. At least one resulting indicator must be specified. Entries must not be made in both high and low for the same LOOKUP operation. The resulting indicators must *not* be specified in high or low if the array is not in sequence (ASCEND or DESCEND keywords). Control level and conditioning indicators (specified in positions 7 through 11) can also be used. The result field cannot be used.

The search starts at the beginning of the array and ends at the end of the array or when the conditions of the lookup are satisfied. Whenever an array element is found that satisfies the type of search being made (equal, high, low), the resulting indicator is set on.

Figure 67 shows an example of a LOOKUP on an array without an index.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++
FARRFILE   IT   F    5        DISK
F*
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DDPTNOS           S              5S 0 DIM(50) FROMFILE(ARRFILE)
D*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* The LOOKUP operation is processed and, if an element of DPTNOS equal
C* to the search argument (DPTNUM) is found, indicator 20 is set on.
C     DPTNUM        LOOKUP    DPTNOS                                 20
```

Figure 67. LOOKUP Operation for an Array without an Index

ARRFILE, which contains department numbers, is defined in the file description specifications as an input file (I in position 17) with an array file designation (T in position 18). The file is program described (F in position 22), and each record is 5 positions in length (5 in position 27).

In the definition specifications, ARRFILE is defined as containing the array DPTNOS. The array contains 50 entries (DIM(50)). Each entry is 5 positions in length (positions 33-39) with zero decimal positions (positions 41-42). One department number can be contained in each record (PERRCD defaults to 1).

Searching an Array with an Index

To find out which element satisfies a LOOKUP search, start the search at a particular element in the array. To do this type of search, make the entries in the calculation specifications as you would for an array without an index. However, in factor 2, enter the name of the array to be searched, followed by a parenthesized numeric field (with zero decimal positions) containing the number of the element at which the search is to start. This numeric constant or field is called the index because it points to a certain element in the array. The index is updated with the element number which satisfied the search or is set to 0 if the search failed.

You can use a numeric constant as the index to test for the existence of an element that satisfies the search starting at an element other than 1.

All other rules that apply to an array without an index apply to an array with an index.

Figure 68 shows a LOOKUP on an array with an index.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
FARRFILE   IT   F   25           DISK
F*
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
DDPTNOS           S              5S 0 DIM(50) FROMFILE(ARRFILE)
DDPTDSC           S             20A   DIM(50) ALT(DPTNOS)
D*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* The Z-ADD operation begins the LOOKUP at the first element in DPTNOS.
C                   Z-ADD     1             X                 3 0
C* At the end of a successful LOOKUP, when an element has been found
C* that contains an entry equal to the search argument DPTNUM,
C* indicator 20 is set on and the MOVE operation places the department
C* description, corresponding to the department number, into DPTNAM.
C     DPTNUM        LOOKUP    DPTNOS(X)                              20
C* If an element is not found that is equal to the search argument,
C* element X of DPTDSC is moved to DPTNAM.
C                   IF        NOT *IN20
C                   MOVE      DPTDSC(X)     DPTNAM           20
C                   ENDIF
```

Figure 68. LOOKUP Operation on an Array with an Index

This example shows the same array of department numbers, DPTNOS, as Figure 67 on page 143. However, an alternating array of department descriptions, DPTDSC, is also defined. Each element in DPTDSC is 20 positions in length. If there is insufficient data in the file to initialize the entire array, the remaining elements in DPTNOS are filled with zeros and the remaining elements in DPTDSC are filled with blanks.

Using Arrays

Arrays can be used in input, output, or calculation specifications.

Specifying an Array in Calculations

An entire array or individual elements in an array can be specified in calculation specifications. You can process individual elements like fields.

A noncontiguous array defined with the OVERLAY keyword cannot be used with the MOVEA operation or in the result field of a PARM operation.

To specify an entire array, use only the array name, which can be used as factor 1, factor 2, or the result field. The following operations can be used with an array name: ADD, Z-ADD, SUB, Z-SUB, MULT, DIV, SQRT, ADDDUR, SUBDUR, EVAL, EXTRCT, MOVE, MOVEL, MOVEA, MLLZO, MLHZO, MHLZO, MHHZO, DEBUG, XFOOT, LOOKUP, SORTA, PARM, DEFINE, CLEAR, RESET, CHECK, CHECKR, and SCAN.

Several other operations can be used with an array element only but not with the array name alone. These operations include but are not limited to: BITON, BITOFF, COMP, CABxx, TESTZ, TESTN, TESTB, MVR, DO, DOUxx, DOWxx, DOU, DOW, IFxx, WHENxx, WHEN, IF, SUBST, and CAT.

When specified with an array name without an index or with an asterisk as the index (for example, ARRAY or ARRAY(*)) certain operations are repeated for each element in the array. These are ADD, Z-ADD, EVAL, SUB, Z-SUB, ADDDUR, SUBDUR, EXTRCT, MULT, DIV, SQRT, MOVE, MOVEL, MLLZO, MLHZO, MHLZO and MHHZO. The following rules apply to these operations when an array name without an index is specified:

- When factors 1 and 2 and the result field are arrays with the same number of elements, the operation uses the first element from every array, then the second element from every array until all elements in the arrays are processed. If the arrays do not have the same number of entries, the operation ends when the last element of the array with the fewest elements has been processed. When factor 1 is not specified for the ADD, SUB, MULT, and DIV operations, factor 1 is assumed to be the same as the result field.
- When one of the factors is a field, a literal, or a figurative constant and the other factor and the result field are arrays, the operation is done once for every element in the shorter array. The same field, literal, or figurative constant is used in all of the operations.
- The result field must always be an array.
- If an operation code uses factor 2 only (for example, Z-ADD, Z-SUB, SQRT, ADD, SUB, MULT, or DIV may not have factor 1 specified) and the result field is an array, the operation is done once for every element in the array. The same field or constant is used in all of the operations if factor 2 is not an array.
- Resulting indicators (positions 71 through 76) cannot be used because of the number of operations being processed.

Note:

When used in an EVAL operation %ADDR(arr) and %ADDR(arr(*)) do not have the same meaning. See “%ADDR (Get Address of Variable)” on page 264 for more detail.

Sorting Arrays

You can sort arrays using the “SORTA (Sort an Array)” on page 461 operation code. The array is sorted into sequence (ascending or descending), depending on the sequence specified for the array on the definition specification.

Sorting using part of the array as a key

You can use the OVERLAY keyword to overlay one array over another. For example, you can have a base array which contains names and salaries and two overlay arrays (one for the names and one for the salaries). You could then sort the base array by either name or salary by sorting on the appropriate overlay array.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D                 DS
D Emp_Info                     50     DIM(500) ASCEND
D   Emp_Name                    45    OVERLAY(Emp_Info:1)
D   Emp_Salary                   9P 2 OVERLAY(Emp_Info:46)
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C* The following SORTA sorts Emp_Info by employee name.
C* The sequence of Emp_Name is used to determine the order of the
C* elements of Emp_Info.
C                   SORTA     Emp_Name
C* The following SORTA sorts Emp_Info by employee salary
C* The sequence of Emp_Salary is used to determine the order of the
C* elements of Emp_Info.
C                   SORTA     Emp_Salary
```

Figure 69. SORTA Operation with OVERLAY

Array Output

Entire arrays can be written out under RPG IV control only at end of program when the LR indicator is on. To indicate that an entire array is to be written out, specify the name of the output file with the TOFILE keyword on the definition specifications. This file must be described as a sequentially organized output or combined file in the file description specifications.
If the file is a combined file and is externally described as a physical file, the information in the array at the end of the program replaces the information read into the array at the start of the program. Logical files may give unpredictable results.

If an entire array is to be written to an output record (using output specifications), describe the array along with any other fields for the record:

- Positions 30 through 43 of the output specifications must contain the array name used in the definition specifications.
- Positions 47 through 51 of the output specifications must contain the record position where the last element of the array is to end. If an edit code is specified, the end position must include blank positions and any extensions due to the edit code (see “Editing Entire Arrays” listed next in this chapter).

Output indicators (positions 21 through 29) can be specified. Zero suppress (position 44), blank-after (position 45), and data format (position 52) entries pertain to every element in the array.

Editing Entire Arrays

When editing is specified for an entire array, all elements of the array are edited. If different editing is required for various elements, refer to them individually.

When an edit code is specified for an entire array (position 44), two blanks are automatically inserted between elements in the array: that is, there are blanks to the left of every element in the array except the first. When an edit word is specified, the blanks are not inserted. The edit word must contain all the blanks to be inserted.

Tables

The explanation of arrays applies to tables except for the following differences:

Activity **Differences**
Defining A table name must be a unique symbolic name that begins with the letters TAB.
Loading Tables can be loaded only at compilation time and prerun-time.
Using and Modifying table elements Only one element of a table is active at one time. The table name is used to refer to the active element. An index cannot be specified for a table.
Searching The LOOKUP operation is specified differently for tables.

LOOKUP with One Table

When a single table is searched, factor 1, factor 2, and at least one resulting indicator must be specified. Conditioning indicators (specified in positions 7 through 11) can also be used.

Whenever a table element is found that satisfies the type of search being made (equal, high, low), that table element is made the current element for the table If the search is not successful, the previous current element remains the current element.

Before a first successful LOOKUP, the first element is the current element.

Resulting indicators reflect the result of the search. If the indicator is on, reflecting a successful search, the element satisfying the search is the current element.

LOOKUP with Two Tables

When two tables are used in a search, only one is actually searched. When the search condition (high, low, equal) is satisfied, the corresponding elements are made available for use.

Factor 1 must contain the search argument, and factor 2 must contain the name of the table to be searched. The result field must name the table from which data is also made available for use. A resulting indicator must also be used. Control level and conditioning indicators can be specified in positions 7 through 11, if needed.

The two tables used should have the same number of entries. If the table that is searched contains more elements than the second table, it is possible to satisfy the search condition. However, there might not be an element in the second table that corresponds to the element found in the search table. Undesirable results can occur.

Note: If you specify a table name in an operation other than LOOKUP before a successful LOOKUP occurs, the table is set to its first element.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* The LOOKUP operation searches TABEMP for an entry that is equal to
C* the contents of the field named EMPNUM. If an equal entry is
C* found in TABEMP, indicator 09 is set on, and the TABEMP entry and
C* its related entry in TABPAY are made the current elements.
C     EMPNUM        LOOKUP    TABEMP        TABPAY                   09
C* If indicator 09 is set on, the contents of the field named
C* HRSWKD are multiplied by the value of the current element of
C* TABPAY.
C                   IF        *IN09
C     HRSWKD        MULT(H)   TABPAY        AMT               6 2
C                   ENDIF
```

Figure 70. Searching for an Equal Entry

Specifying the Table Element Found in a LOOKUP Operation

Whenever a table name is used in an operation other than LOOKUP, the table name actually refers to the data retrieved by the last successful search. Therefore, when the table name is specified in this fashion, elements from a table can be used in calculation operations.

If the table is used as factor 1 in a LOOKUP operation, the current element is used as the search argument. In this way an element from a table can itself become a search argument.

The table can also be used as the result field in operations other than the LOOKUP operation. In this case the value of the current element is changed by the calculation specification. In this way the contents of the table can be modified by calculation operations (see Figure 71).

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C     ARGMNT        LOOKUP    TABLEA                                 20
C* If element is found multiply by 1.5
C* If the contents of the entire table before the MULT operation
C* were 1323.5, -7.8, and 113.4  and the value of ARGMNT is -7.8,
C* then the second element is the current element.
C* After the MULT operation, the entire table now has the
C* following value:  1323.5, -11.7, and 113.4.
C* Note that only the second element has changed since that was
C* the current element, set by the LOOKUP.
C                   IF        *IN20
C     TABLEA        MULT      1.5           TABLEA
C                   ENDIF
```

Figure 71. Specifying the Table Element Found in LOOKUP Operations

Chapter 11. Editing Numeric Fields

Editing provides a means of punctuating numeric fields, including the printing of currency symbols, commas, periods, minus sign, and floating minus. It also provides for field sign movement from the rightmost digit to the end of the field, blanking zero fields, spacing in arrays, date field editing, and currency symbol or asterisk protection. A field can be edited by edit codes, or edit words.

When you print fields that are not edited, the fields appear exactly as they are internally represented. The following examples show why you may want to edit numeric output fields.

Type of Field	Field in the Computer	Printing of Unedited Field	Printing of Edited Field
Alphanumeric	JOHN T SMITH	JOHN T SMITH	JOHN T SMITH
Numeric (positive)	0047652	0047652	47652
Numeric (negative)	004765K	004765K	47652-

The unedited alphanumeric field and the unedited positive numeric field are easy to read when printed, but the unedited negative numeric field is confusing because it contains a K, which is not numeric. The K is a combination of the digit 2 and the negative sign for the field. They are combined so that one of the positions of the field does not have to be set aside for the sign. The combination is convenient for storing the field in the computer, but it makes the output hard to read. Therefore, numeric fields need to be edited before they are printed.

Edit Codes

Edit codes provide a means of editing numeric fields according to a predefined pattern. They are divided into three categories: simple (X, Y, Z), combination (1 through 4, A through D, J through Q), and user-defined (5 through 9). You enter the edit code in position 44 of the output specifications for the field to be edited.

Simple Edit Codes

You can use simple edit codes to edit numeric fields without having to specify any punctuation. These codes and their functions are:

- The X edit code ensures a hexadecimal F sign for positive fields. However, because the system does this, you normally do not have to specify this code. Leading zeros are not suppressed. The X edit code does not modify negative numbers.
- The Y edit code is normally used to edit a 3- to 9-digit date field. It suppresses the leftmost zeros of date fields, up to but not including the digit preceding the first separator. Slashes are inserted to separate the day, month, and year. The “DATEDIT(fmt{separator})” and “DECEDIT('value')” keywords on the control specification can be used to alter edit formats.
- The Y edit code is not valid for *YEAR, *MONTH, and *DAY.

- The Z edit code removes the sign (plus or minus) from and suppresses the leading zeros of a numeric field. The decimal point is not placed in the field and is not printed.

Combination Edit Codes

The combination edit codes (1 through 4, A through D, J through Q) punctuate a numeric field.

The DECEDIT keyword on the control specification determines what character is used for the decimal separator and whether leading zeroes are suppressed. The decimal position of the source field determines whether and where a decimal point is printed. If decimal positions are specified for the source field and the zero balance is to be suppressed, the decimal separator prints *only* if the field is not zero. If a zero balance is not to be printed, a zero field prints as blanks.

When a zero balance is to be printed and the field is equal to zero, either of the following is printed:

- A decimal separator followed by n zeros, where n is the number of decimal places in the field
- A zero in the units position of a field if no decimal places are specified.

You can use a floating currency symbol or asterisk protection with any of the 12 combination edit codes. To specify a floating currency symbol, code the currency symbol in positions 53-55 on the output specification, along with an edit code in position 44 for the field to be edited. The floating currency symbol appears to the left of the first significant digit. The floating currency symbol does not print on a zero balance when an edit code is used that suppresses the zero balance. (A dollar sign ($) is used as the currency symbol unless a currency symbol is specified with the CURSYM keyword on the control specification.)

An asterisk constant coded in positions 53 through 55 of the output specifications ('*'), along with an edit code for the field to be edited causes an asterisk to be printed for each zero suppressed. A complete field of asterisks is printed on a zero balance source field.

Asterisk fill and the floating currency symbol *cannot* be used with the simple (X, Y, Z) or with the user-defined (5 through 9) edit codes.

A currency symbol can appear before the asterisk fill (fixed currency symbol). This requires two output specifications with the following coding:

1. Place a currency symbol constant in position 53 of the first output specification. The end position specified in positions 47-51 should be one space before the beginning of the edited field.
2. In the second output specification, place the edit field in positions 30-43, an edit code in position 44, end position of the edit field in positions 47-51, and '*' in positions 53-55.

When an edit code is used to print an entire array, two blanks precede each element of the array (except the first element).

Table 14 summarizes the functions of the combination edit codes. The codes edit the field in the format listed on the left. A negative field can be punctuated with no sign, CR, a minus sign (-), or a floating minus sign as shown on the top of the figure.

Table 14. Combination Edit Codes

		Negative Balance Indicator			
Prints with Grouping Separator	**Prints Zero Balance**	**No Sign**	**CR**	**-**	**Floating Minus**
Yes	Yes	1	A	J	N
Yes	No	2	B	K	0
No	Yes	3	C	L	P
No	No	4	D	M	Q

User-Defined Edit Codes

IBM has predefined edit codes 5 through 9. You can use them as they are, or you can delete them and create your own. For a description of the IBM-supplied edit codes, see “Edit Descriptions” in Chapter 6 of the *Programming Reference Summary*.

The user-defined edit codes allow you to handle common editing problems that would otherwise require the use of an edit word. Instead of the repetitive coding of the same edit word, a user-defined edit code can be used. These codes are system defined by the CL command CRTEDTD (Create Edit Description).

When you edit a field defined to have decimal places, be sure to use an edit word that has an editing mask for both the fractional and integer portions of the field. Remember that when a user-defined edit code is specified in a program, any system changes made to that user-defined edit code are not reflected until the program is recompiled. For further information on CRTEDTD, see the *Programming: Control Language Reference*.

Editing Considerations

Remember the following when you specify any of the edit codes:

- Edit fields of a non-printer file with caution. If you do edit fields of a non-printer file, be aware of the contents of the edited fields and the effects of any operations you do on them. For example, if you use the file as input, the fields written out with editing must be considered character fields, not numeric fields.
- Consideration should be given to data added by the edit operation. The amount of punctuation added increases the overall length of the output field. If these added characters are not considered, the output fields may overlap.
- The end position specified for output is the end position of the edited field. For example, if any of the edit codes J through M are specified, the end position is the position of the minus sign (or blank if the field is positive).

Summary of Edit Codes

Table 15 summarizes the edit codes and the options they provide. A simplified version of this table is printed above positions 45 through 70 on the output specifications. Table 16 on page 153 shows how fields look after they are edited.

Table 17 on page 154 shows the effect that the different edit codes have on the same field with a specified end position for output.

Table 15 (Page 1 of 2). Edit Codes

				DECEDIT Keyword Parameter				
Edit Code	**Commas**	**Decimal Point**	**Sign for Negative Balance**	**'.'**	**','**	**'0,'**	**'0.'**	**Zero Suppress**
1	Yes	Yes	No Sign	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes
2	Yes	Yes	No Sign	Blanks	Blanks	Blanks	Blanks	Yes
3		Yes	No Sign	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes
4		Yes	No Sign	Blanks	Blanks	Blanks	Blanks	Yes
5-9[1]								
A	Yes	Yes	CR	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes
B	Yes	Yes	CR	Blanks	Blanks	Blanks	Blanks	Yes
C		Yes	CR	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes
D		Yes	CR	Blanks	Blanks	Blanks	Blanks	Yes
J	Yes	Yes	- (minus)	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes
K	Yes	Yes	- (minus)	Blanks	Blanks	Blanks	Blanks	Yes
L		Yes	- (minus)	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes
M		Yes	- (minus)	Blanks	Blanks	Blanks	Blanks	Yes
N	Yes	Yes	- (floating minus)	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes
O	Yes	Yes	- (floating minus)	Blanks	Blanks	Blanks	Blanks	Yes
P		Yes	- (floating minus)	.00 or 0	,00 or 0	0,00 or 0	0.00 or 0	Yes

[1] These are the user-defined edit codes.

[2] The X edit code ensures a hexadecimal F sign for positive values. Because the system does this for you, normally you do not have to specify this code.

[3] The Y edit code suppresses the leftmost zeros of date fields, up to but not including the digit preceding the first separator. The Y edit code also inserts slashes (/) between the month, day, and year according to the following pattern:

```
   nn/n
   nn/nn
   nn/nn/n
   nn/nn/nn
  nnn/nn/nn
   nn/nn/nnnn
  nnn/nn/nnnn
 nnnn/nn/nn
nnnnn/nn/nn
```

[4] The Z edit code removes the sign (plus or minus) from a numeric field and suppresses leading zeros.

Table 15 (Page 2 of 2). Edit Codes

				DECEDIT Keyword Parameter				
Edit Code	**Commas**	**Decimal Point**	**Sign for Negative Balance**	**'.'**	**','**	**'0,'**	**'0.'**	**Zero Sup-press**
Q		Yes	- (floating minus)	Blanks	Blanks	Blanks	Blanks	Yes
X[2]								Yes
Y[3]								Yes
Z[4]								Yes

[1] These are the user-defined edit codes.

[2] The X edit code ensures a hexadecimal F sign for positive values. Because the system does this for you, normally you do not have to specify this code.

[3] The Y edit code suppresses the leftmost zeros of date fields, up to but not including the digit preceding the first separator. The Y edit code also inserts slashes (/) between the month, day, and year according to the following pattern:

```
    nn/n
    nn/nn
    nn/nn/n
    nn/nn/nn
   nnn/nn/nn
    nn/nn/nnnn
   nnn/nn/nnnn
  nnnn/nn/nn
 nnnnn/nn/nn
```

[4] The Z edit code removes the sign (plus or minus) from a numeric field and suppresses leading zeros.

Table 16 (Page 1 of 2). Examples of Edit Code Usage

Edit Codes	**Positive Number-Two Decimal Positions**	**Positive Number-No Decimal Positions**	**Negative Number-Three Decimal Positions**	**Negative Number-No Decimal Positions**	**Zero Balance-Two Decimal Positions**	**Zero Balance-No Decimal Positions**
Unedited	1234567	1234567	00012ƀ[5]	00012ƀ[5]	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		
5-9[1]						
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	1234567	.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

Table 16 (Page 2 of 2). Examples of Edit Code Usage

Edit Codes	Positive Number-Two Decimal Positions	Positive Number-No Decimal Positions	Negative Number-Three Decimal Positions	Negative Number-No Decimal Positions	Zero Balance-Two Decimal Positions	Zero Balance-No Decimal Positions
M	12345.67	1234567	.120-	120-		
N	12,345.67	1,234,567	-.120	-120	.00	0
O	12,345,67	1,234,567	-.120	-120		
P	12345.67	1234567	-.120	-120	.00	0
Q	12345.67	1234567	-.120	-120		
X[2]	1234567	1234567	00012ƀ[5]	00012ƀ[5]	000000	000000
Y[3]			0/01/20	0/01/20	0/00/00	0/00/00
Z[4]	1234567	1234567	120	120		

[1] These edit codes are user-defined.

[2] The X edit code ensures a hex F sign for positive values. Because the system does this for you, normally you do not have to specify this code.

[3] The Y edit code suppresses the leftmost zeros of date fields, up to but not including the digit preceding the first separator. The Y edit code also inserts slashes (/) between the month, day, and year according to the following pattern:

```
    nn/n
    nn/nn
    nn/nn/n
    nn/nn/nn
   nnn/nn/nn
    nn/nn/nnnn  Format used with M, D or blank in position 19
   nnn/nn/nnnn  Format used with M, D or blank in position 19
  nnnn/nn/nn    Format used with Y in position 19
 nnnnn/nn/nn    Format used with Y in position 19
```

[4] The Z edit code removes the sign (plus or minus) from a numeric field and suppresses leading zeros of a numeric field.

[5] The ƀ represents a blank. This may occur if a negative zero does not correspond to a printable character.

Table 17 (Page 1 of 2). Effects of Edit Codes on End Position

	Negative Number, 2 Decimal Positions. End Position Specified as 10.								
	Output Print Positions								
Edit Code	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Unedited				0	0	4	1	K[1]	
1					4	.	1	2	

[1]K represents a negative 2.

[2]These are user-defined edit codes.

Table 17 (Page 2 of 2). Effects of Edit Codes on End Position

	Negative Number, 2 Decimal Positions. End Position Specified as 10.								
	Output Print Positions								
Edit Code	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
2					4	.	1	2	
3					4	.	1	2	
4					4	.	1	2	
5-9[2]									
A			4	.	1	2	C	R	
B			4	.	1	2	C	R	
C			4	.	1	2	C	R	
D			4	.	1	2	C	R	
J				4	.	1	2	-	
K				4	.	1	2	-	
L				4	.	1	2	-	
M				4	.	1	2	-	
N				-	4	.	1	2	
O				-	4	.	1	2	
P				-	4	.	1	2	
Q				-	4	.	1	2	
X				0	0	4	1	K[1]	
Y			0	/	4	1	/	2	
Z						4	1	2	

[1] K represents a negative 2.

[2] These are user-defined edit codes.

Edit Words

If you have editing requirements that cannot be met by using the edit codes described above, you can use an edit word. An edit word is a character literal or a named constant specified in positions 53 - 80 of the output specification. It describes the editing pattern for an numeric and allows you to directly specify:

- Blank spaces
- Commas and decimal points, and their position
- Suppression of unwanted zeros
- Leading asterisks
- The currency symbol, and its position
- Addition of constant characters
- Output of the negative sign, or CR, as a negative indicator.

The edit word is used as a template, which the system applies to the source data to produce the output.

The edit word may be specified directly on an output specification or may be specified as a named constant with a named constant name appearing in the edit word field of the output specification.

Named constants, used as edit words, are limited to 115 characters.

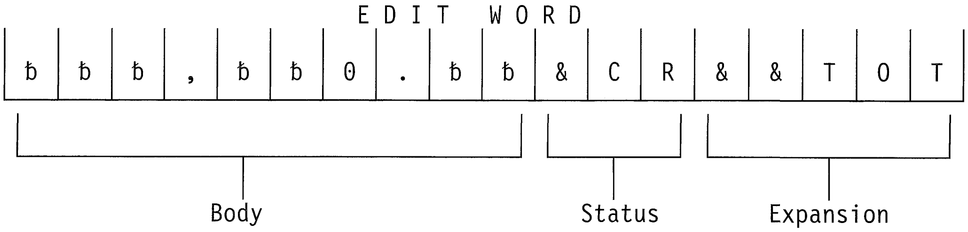
How to Code an Edit Word

To use an edit word, code the output specifications as shown below:

Position	**Entry**
21-29	Can contain conditioning indicators.
30-43	Contains the name of the numeric field from which the data that is to be edited is taken.
44	*Edit code*. Must be blank, if you are using an edit word to edit the source data.
45	A “B” in this position indicates that the source data is to be set to zero or blanks after it has been edited and output. Otherwise the source data remains unchanged.
47-51	Identifies the end (rightmost) position of the field in the output record.
53-80	*Edit word*. Can be up to 26 characters long and must be enclosed by apostrophes, unless it is a named constant. Enter the leading apostrophe, or begin the named constant name in column 53. The edit word, unless a named constant, must begin in column 54.

Parts of an Edit Word

An edit word (coded into positions 53 to 80 of the output specifications) consists of three parts: the body, the status, and the expansion. The following shows the three parts of an edit word:



The *body* is the space for the digits transferred from the source data field to the output record. The body begins at the leftmost position of the edit word. The number of blanks (plus one zero or an asterisk) in the edit word body must be equal to or greater than the number of digits of the source data field to be edited. The body ends with the rightmost character that can be replaced by a digit.

The *status* defines a space to allow for a negative indicator, either the two letters CR or a minus sign (-). The negative indicator specified is output only if the source data is negative. All characters in the edit word between the last replaceable character (blank, zero suppression character) and the negative indicator are also output with the negative indicator only if the source data is negative; if the source data is positive, these status positions are replaced by blanks. Edit words without the CR or - indicators have no status positions.

The status must be entered after the last blank in the edit word. If more than one CR follows the last blank, only the first CR is treated as a status; the remaining CRs are treated as constants. For the minus sign to be considered as a status, it must be the last character in the edit word.

The *expansion* is a series of ampersands and constant characters entered after the status. Ampersands are replaced by blank spaces in the output; constants are output as is. If status is not specified, the expansion follows the body.

Forming the Body of an Edit Word

The following characters have special meanings when used in the body of an edit word:

Blank: Blank is replaced with the character from the corresponding position of the source data field specified by the field name in positions 30 through 43 of the output specifications. A blank position is referred to as a digit position.

Decimals and Commas: Decimals and commas are in the same relative position in the edited output field as they are in the edit word unless they appear to the left of the first significant digit in the edit word. In that case, they are blanked out or replaced by an asterisk.

In the following examples below, all the leading zeros will be suppressed (default) and the decimal point will not print unless there is a significant digit to its left.

Edit Word	Source Data	Appears in Output Record as:
'ƀƀƀƀƀƀƀ'	0000072	ƀƀƀƀƀ72
'ƀƀƀƀƀƀƀ.ƀƀ'	000000012	ƀƀƀƀƀƀƀƀ12
'ƀƀƀƀƀƀƀ.ƀƀ'	000000123	ƀƀƀƀƀƀ1.23

Zeros: The first zero in the body of the edit word is interpreted as an end-zero-suppression character. This zero is placed where zero suppression is to end. Subsequent zeros put into the edit word are treated as constants (see "Constants" below).

Any leading zeros in the source data are suppressed up to and including the position of the end-zero-suppression character. Significant digits that would appear in the end-zero-suppression character position, or to the left of it, are output.

Edit Word	Source Data	Appears in Output Record as:
'ƀƀƀ0ƀƀƀƀƀƀ'	00000004	ƀƀƀƀ000004
'ƀƀƀ0ƀƀƀƀƀƀ'	012345	ƀƀƀƀ012345
'ƀƀƀ0ƀƀƀƀƀƀ'	012345678	ƀƀ12345678

If the leading zeros include, or extend to the right of, the end-zero-suppression character position, that position is replaced with a blank. This means that if you wish the same number of leading zeros to appear in the output as exist in the source data, the edit word body must be wider than the source data.

Edit Word	Source Data	Appears in Output Record as:
'0ƀƀƀ'	0156	ƀ156
'0ƀƀƀƀ'	0156	ƀ0156

Constants (including commas and decimal point) that are placed to the right of the end-zero-suppression character are output, even if there is no source data. Constants to the left of the end-zero-suppression character are only output if the source data has significant digits that would be placed to the left of these constants.

Edit Word	Source Data	Appears in Output Record as:
'ƀƀƀƀƀƀ0.ƀƀ'	000000001	ƀƀƀƀƀƀƀ.01
'ƀƀƀƀƀƀ0.ƀƀ'	000000000	ƀƀƀƀƀƀƀ.00
'ƀƀƀ,ƀ0ƀ.ƀƀ'	00000012	ƀƀƀƀƀƀ0.12
'ƀƀƀ,ƀ0ƀ.ƀƀ'	00000123	ƀƀƀƀƀƀ1.23
'ƀ0ƀ,ƀƀƀ.ƀƀ'	00000123	ƀƀ0,001.23

Asterisk: The first asterisk in the body of an edit word also ends zero suppression. Subsequent asterisks put into the edit word are treated as constants (see "Constants" below). Any zeros in the edit word following this asterisk are also treated as constants. There can be only one end-zero-suppression character in an edit word, and that character is the first asterisk *or* the first zero in the edit word.

If an asterisk is used as an end-zero-suppression character, all leading zeros that are suppressed are replaced with asterisks in the output. Otherwise, the asterisk suppresses leading zeros in the same way as described above for "Zeros".

Edit Word	Source Data	Appears in Output Record as:
'*ƀƀƀƀƀƀ.ƀƀ'	000000123	*ƀƀƀƀƀ1.23
'ƀƀƀƀƀ*ƀ.ƀƀ'	000000000	******0.00
'ƀƀƀƀƀ*ƀ.ƀƀ**'	000056342	****563.42**

Note that leading zeros appearing after the asterisk position are output as leading zeros. Only the suppressed leading zeros, including the one in the asterisk position, are replaced by asterisks.

Currency Symbol: A currency symbol followed directly by a first zero in the edit word (end-zero-suppression character) is said to float. All leading zeros are suppressed in the output and the currency symbol appears in the output immediately to the left of the most significant digit.

Edit Word	Source Data	Appears in Output Record as:
'ƀƀ,ƀƀƀ,ƀ$0.ƀƀ'	000000012	ƀƀƀƀƀƀƀƀƀ$.12
'ƀƀ,ƀƀƀ,ƀ$0.ƀƀ'	000123456	ƀƀƀƀ$1,234.56

If the currency symbol is put into the first position of the edit word, then it will always appear in that position in the output. This is called a fixed currency symbol.

Edit Word	Source Data	Appears in Output Record as:
'$ƀ,ƀƀƀ,ƀƀ0.ƀƀ'	000123456	$ƀƀƀƀ1,234.56
'$ƀƀ,ƀƀƀ,0ƀ0.ƀƀ'	000000000	$ƀƀƀƀƀƀƀƀ00.00
'$ƀ,ƀƀƀ,*ƀƀ.ƀƀ'	000123456	$****1,234.56

A currency symbol anywhere else in the edit word and not immediately followed by a zero end-suppression-character is treated as a constant (see "Constants" below).

Ampersand: Causes a blank in the edited field. The example below might be used to edit a telephone number. Note that the zero in the first position is required to print the constant AREA.

Edit Word	Source Data	Appears in Output Record as:
'0AREA&ƀƀƀ&NO.&ƀƀƀ-ƀƀƀƀ'	4165551212	ƀAREAƀ416ƀNO.ƀ555-1212

Constants: All other characters entered into the body of the edit word are treated as constants. If the source data is such that the output places significant digits or leading zeros to the left of any constant, then that constant appears in the output. Otherwise, the constant is suppressed in the output. Commas and the decimal point follow the same rules as for constants. Notice in the examples below, that the presence of the end-zero-suppression character as well as the number of significant digits in the source data, influence the output of constants.

The following edit words could be used to print cheques. Note that the second asterisk is treated as a constant, and that, in the third example, the constants preceding the first significant digit are not output.

Edit Word	Source Data	Appears in Output Record as:
'$ƀƀƀƀƀƀ**DOLLARS&ƀƀ&CTS'	000012345	$****123*DOLLARSƀ45ƀCTS
'$ƀƀƀƀƀƀ**DOLLARS&ƀƀ&CTS'	000000006	$********DOLLARSƀ06ƀCTS
'$ƀƀƀƀƀƀƀ&DOLLARS&ƀƀ&CTS'	000000006	$ƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀ6ƀCTS

A date could be printed by using either edit word:

Edit Word	Source Data	Appears in Output Record as:
'ƀƀ/ƀƀ/ƀƀ'	010388	ƀ1/03/88
'0ƀƀ/ƀƀ/ƀƀ'	010389	ƀ01/03/89

Note that any zeros or asterisks following the first occurrence of an edit word are treated as constants. The same is true for - and CR:

Edit Word	Source Data	Appears in Output Record as:
'ƀƀ0.ƀƀ000'	01234	ƀ12.34000
'ƀƀ*.ƀƀ000'	01234	*12.34000

Forming the Status of an Edit Word

The following characters have special meanings when used in the status of an edit word:

Ampersand: Causes a blank in the edited output field. An ampersand cannot be placed in the edited output field.

CR or minus symbol: If the sign in the edited output is plus (+), these positions are blanked out. If the sign in the edited output field is minus (–), these positions remain undisturbed.

The following example adds a negative value indication. The minus sign will print only when the value in the field is negative. A CR symbol fills the same function as a minus sign.

Edit Word	Source Data	Appears in Output Record as:
'ƀƀƀƀƀƀƀ.ƀƀ-'	000000123-	ƀƀƀƀƀƀ1.23-
'ƀƀƀƀƀƀƀ.ƀƀ-'	000000123	ƀƀƀƀƀƀ1.23ƀ

Constants between the last replaceable character and the - or CR symbol will print only if the field is negative; otherwise, blanks will print in these positions. Note the use of ampersands to represent blanks:

Edit Word	Source Data	Appears in Output Record as:
'ƀ,ƀƀƀ,ƀƀ0.ƀƀ&30&DAY&CR'	000000123-	ƀƀƀƀƀƀƀƀƀ1.23ƀ30ƀDAYƀCR
'ƀ,ƀƀƀ,ƀƀ0.ƀƀ&30&DAY&CR'	000000123	ƀƀƀƀƀƀƀƀƀ1.23ƀƀƀƀƀƀƀƀƀƀ

Formatting the Expansion of an Edit Word

The characters in the expansion portion of an edit word are always written. The expansion cannot contain blanks. If a blank is required in the edited output field, specify an ampersand in the body of the edit word.

Constants may be added to print on every line:

Edit Word	Source Data	Appears in Output Record as:
'ƀ,ƀƀ0.ƀƀ&CR&NET'	000123-	ƀƀƀƀ1.23ƀCRƀNET
'ƀ,ƀƀ0.ƀƀ&CR&NET'	000123	ƀƀƀƀ1.23ƀƀƀƀNET

Note that the CR in the middle of a word may be detected as a negative field value indication. If a word such as SECRET is required, use the coding in the example below.

Edit Word	Source Data	Appears in Output Record as:
'ƀƀ0.ƀƀ&SECRET'	12345-	123.45ƀSECRET
'ƀƀ0.ƀƀ&SECRET'	12345	123.45ƀƀƀƀƀET
'ƀƀ0.ƀƀ&CR&&SECRET'	12345	123.45ƀƀƀƀƀSECRET

Summary of Coding Rules for Edit Words

The following rules apply to edit words:

- Position 44 (edit codes) must be blank.
- Positions 30 through 43 (field name) must contain the name of a numeric field.
- An edit word must be enclosed in apostrophes, unless it is a named constant. Enter the leading apostrophe or begin a named constant name in position 53. The edit word itself must begin in position 54.
- The edit word can contain more digit positions (blanks plus the initial zero or asterisk) than the field to be edited, but must not contain less. If there are more digit positions in the edit word than there are digits in the field to be edited, leading zeros are added to the field before editing.
- If leading zeros from the source data are desired, the edit word must contain one more position than the field to be edited, and a zero must be placed in the high-order position of the edit word.
- In the body of the edit word only blanks and the zero-suppression stop characters (zero and asterisk) are counted as digit positions. The floating currency symbol is not counted as a digit position.
- When the floating currency symbol is used, the sum of the number of blanks and the zero-suppression stop character (digit positions) contained in the edit word must be equal to or greater than the number of positions in the field to be edited.
- Any zeros or asterisks following the leftmost zero or asterisk are treated as constants; they are not replaceable characters.

Editing Externally Described Files

Edit codes must be specified in data description specifications (DDS), instead of the RPG IV language, to edit output for externally described files. See the &ddsguidl. for information on how to specify edit codes in the data description specifications. However, if an externally described file, which has an edit code specified, is to be written out as a program described output file, you must specify editing in the output specifications. In this case, any edit codes in the data description specifications are ignored.

Chapter 12. Initialization of Data and Initialization Subroutine

This chapter describes how data is initialized by ILE RPG/400.

Initialization

The initialization support provided by the RPG IV compiler consists of three parts: the initialization subroutine, the CLEAR and RESET operation codes, and data initialization.

Initialization Subroutine (*INZSR)

The initialization subroutine allows you to process calculation specifications before 1P output. It is declared like any other subroutine, but with the special name *INZSR in factor 1. This subroutine will be automatically invoked at the end of the initialization step in the RPG IV program before 1P output. You can enter any calculations that you want in this subroutine except RESET, and it can also be called explicitly by using an EXSR or CASxx operation code.

CLEAR and RESET Operation Codes

The CLEAR operation code sets a variable or all variables in a structure to its default value. If you specify a structure (record format, data structure or array) all fields in that structure are cleared in the order in which they are declared.

The RESET operation code sets a variable or all variables in a structure to their initial value. The initial value for a variable is the value it had at the end of the initialization step in the RPG IV cycle, after the initialization subroutine has been invoked. You can use data structure initialization to assign initial values to subfields, and then change the values during the running of the program, and use the RESET operation code to set the field values back to their initial values. Because the initial value is the value the variable had after the initialization subroutine is executed, you can use the initialization subroutine to assign initial values to a variable and then later use RESET to set the variable back to this initial value. This applies only to the initialization subroutine when it is run automatically as a part of the initialization step.

Data Initialization

Data is initialized with the "INZ{(constant)}" keyword on the definition specification. You can specify an initial value as a parameter on the INZ keyword, or specify the keyword without a parameter and use the default initial values. Default initial values for the various data types are described in Chapter 7, "Data Types and Data Formats." See Chapter 10, "Using Arrays and Tables" for information on initializing arrays.

Specifications

This section describes the RPG IV specifications. First, information common to several specifications, such as keyword syntax and continuation rules is described. Next, the specifications are described in the order in which they must be entered in your program. Each specification description lists all the fields on the specification and explains all the possible entries.

Chapter 13. General Information

RPG IV code is coded on a variety of specifications. Each specification has a specific set of functions.

The following illustration describes the specifications.

Note

The RPG IV source program must be entered into the system in the order shown. Any of the specification types can be absent, but at least one must be present.

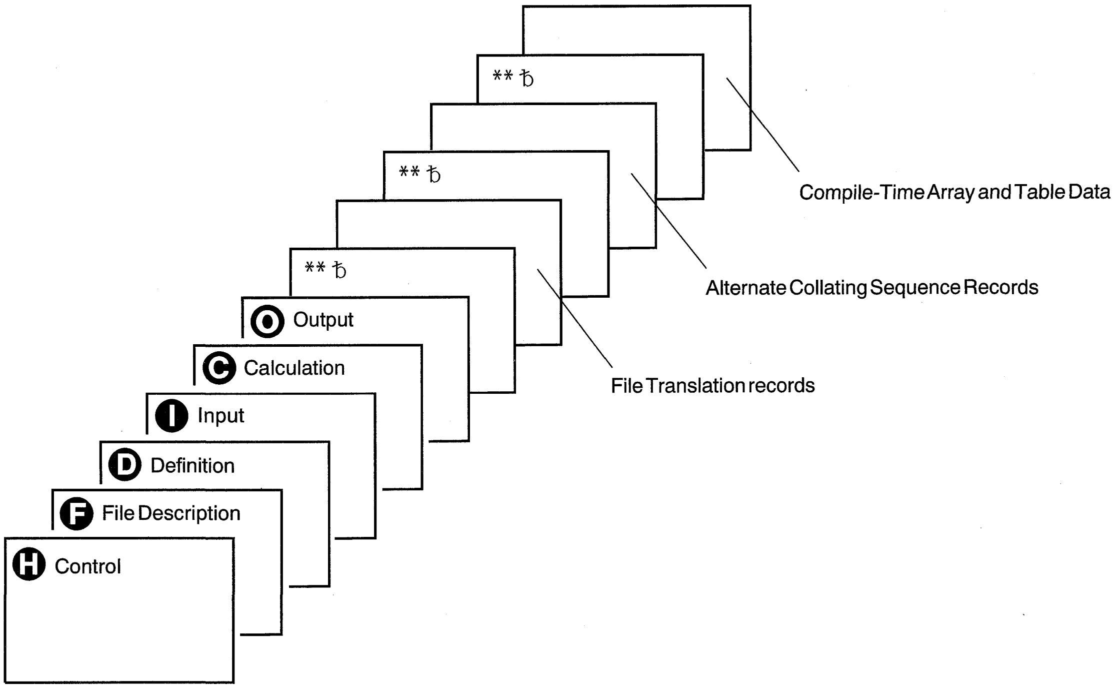


Figure 72. Order of the Types of Specifications in an RPG IV Source Program

H Control (Header) specifications provide information about program generation and running of the compiled program. Refer to Chapter 14, “Control Specifications” for a description of the entries on this specification.

F File description specifications define all files in the program. Refer to Chapter 15, “File Description Specifications” for a description of the entries on this specification.

D Definition specifications define data used in your program. Arrays, tables, data structures, subfields, constants, and stand-alone fields are defined on this specification. Refer to Chapter 16, “Definition Specification” for a description of the entries on this specification.

I Input specifications describe records, and fields in the input files and indicate how the records and fields are used by the program. Refer to Chapter 17, “Input Specifications” for a description of the entries on this specification.

C Calculation specifications describe calculations to be done by the program and indicate the order in which they are done. Calculation specifications can control certain input and output operations. Refer to Chapter 18, "Calculation Specifications" for a description of the entries on this specification.

O Output specifications describe the records and fields and indicate when they are to be written by the program. Refer to Chapter 19, "Output Specifications" for a description of the entries on this specification.

The RPG IV language consists of a mixture of position-dependent code and free form code. Those specifications which support keywords (Control, File Description, and Definition) allow free format in the keyword fields. The Calculation specification allows free format with those operation codes which support extended factor 2. Otherwise, RPG IV entries are position specific. To represent this, each illustration of RPG IV code will be in listing format with a scale drawn across the top.

This reference contains a detailed description of the individual RPG IV specifications. Each field and its possible entries are described. Chapter 22, "Operation Codes" describes the operation codes that are coded on the Calculation specification, which is described in Chapter 18, "Calculation Specifications."

Common Entries

The following entries are common to all RPG specifications:

- Positions 1-5 can be used for comments.
- Specification type (position 6). The following letter codes can be used:

Entry	**Specification Type**
H	Control
F	File description
D	Definition
I	Input
C	Calculation
O	Output

- Comment Statements
  - Position 7 contains an '*'. This will denote the line as a comment line regardless of any other entry on the specification.
  - Position 6 - 80 is blank
- Positions 7 - 80 are blank and position 6 contains a valid specification. This is a valid line, not a comment, and sequence rules are enforced.

Syntax of keywords

The notation for keywords is as follows:

```
Keyword{(parameter {: parameter...})}
```

Keywords may have no parameters, optional parameters, required parameters, or a combination of required and optional parameters.

Parameter(s) are enclosed in parentheses ().

The curly brackets {} are used to indicate optional parameters in the examples in this document. They are not part of the keyword syntax.

The colon(:) is used to separate parameters.

Continuation rules

The fields which may be continued are:

- The keywords field on the control specification
- The keywords field on the file description specification
- The keywords field on the definition specification
- The Extended factor-2 field on the calculation specification
- The constant/editword field on the output specification

General rules for continuation are as follows:

- The continuation line must be a valid line for the specification being continued (H, F, D, C, or O in position 6)
- Only blank lines, empty specification lines or comment lines are allowed between continued lines
- The continuation can occur after a complete token. Tokens are
  - Names (for example, keywords, file names, field names)
  - Parentheses
  - The separator character (:)
  - Expression operators
  - Built-in functions
  - Special words
  - Literals
- A continuation can also occur within a literal
  - For character, date, time, and timestamp literals
    - A hyphen (-) indicates continuation is in the first available position in the continued field
    - A plus (+) indicates continuation with the first nonblank character in or past the first position in the continued field
  - For graphic literals
    - Either the hyphen (-) or plus (+) can be used to indicate a continuation.
    - Each segment of the literal must be enclosed by shift-out and shift-in characters.
    - When the a graphic literal is assembled, only the first shift-out and the last shift-in character will be included.
    - Regardless of which continuation character is used for a graphic literal, the literal continues with the first character after the shift-out character on the continuation line. Spaces preceding the shift-out character are ignored.
  - For numeric literals
    - No continuation character is used
    - A numeric literal continues with a numeric character or decimal point on the continuation line in the continued field
  - For hexadecimal literals
    - Either a hyphen (-) or a plus (+) can be used to indicate a continuation
    - The literal will be continued with the first nonblank character on the next line

Control specification keyword field

The rule for continuation on the control specification is:

- The specification continues on or past position 6 of the next control specification

Example

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DATFMT(
H      *MDY&
H        )
```

File description specification keyword field

The rules for continuation on the file description specification are:

- The specification continues on or past position 44 of the next file description specification
- Positions 7-43 of the continuation line must be blank

Example

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
F                                     EXTIND
F                                     (
F                                     *INU1
F                                     )
```

Definition specification keyword field

The rules for continuation on the Definition specification are:

- The specification continues on or past position 44 of the next Definition specification dependent on the continuation character specified
- Positions 7-43 of the continuation line must be blank

Example

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D                                     Keywords-cont++++++++++++++++++++++++

DMARY             C                   CONST(
D                                     'Mary had a little lamb, its -
D* Only a comment or a completely blank line is allowed in here
D                                     fleece was white as snow.'
D                                     )
D* Numeric literal, continues with the first non blank in/past position 44
D*
DNUMERIC          C                   12345
D                                       67
D* Graphic named constant, must have shift-out in/past position 44
DGRAF             C                  G'oAABBCCDDi+
D                                      oEEFFGGi'
```

Calculation specification Extended Factor-2

The rules for continuation on the Calculation specification are:

- The specification continues on or past position 36 of the next Calculation specification
- Positions 7-35 of the continuation line must be blank

Example

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++++
C                             Extended-factor2-++++++++++++++++++++++++++++

C                   EVAL      MARY='Mary had a little lamb, its +
C* Only a comment or a completely blank line is allowed in here
C                                fleece was white as snow.'
C*
C*  Continuation of arithmetic expression, NOT a continuation
C*  character
C
C                   EVAL      A = (B*D)/ C +
C                             24
C*  The first use of '+' in this example is the concatenation
C*  operator.  The second use is the character literal continuation.
C                   EVAL      ERRMSG = NAME +
C                                      ' was not found +
C                                      in the file.'
```

Output specification constant/editword field

The rules for continuation on the Output specification are:

- The specification continues on or past position 53 of the next Output specification
- Positions 7-52 of the continuation line must be blank

Example

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8

O............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat+++
O                                            Continue Constant/editword+++

O                                         80 'Mary had a little lamb, its-
O* Only a comment or a completely blank line is allowed in here
O                                             fleece was white as snow.'
```

Chapter 14. Control Specifications

The control specification statements, identified by an H in column 6, provides information about generating and running programs. However, there are three different ways in which this information can be provided to the compiler and the compiler searches for this information in the following order:

1. A control specification included in your source
2. A data area named RPGLEHSPEC in *LIBL
3. A data area named DFTLEHSPEC in QRPGLE

The search is stopped once one of these sources is found. If none of them is found, the control specification keywords are assigned their default values.

See the description of the individual entries for their default values.

Using a Data Area as a Control Specification

Use the CL command CRTDTAARA (Create Data Area) to create a data area defined as type *CHAR. (See the *Programming: Control Language Reference* for a description of the Create Data Area command.) Specify as the initial value of the data area the entries for the control specification that are to be used. For example, if the DUMP operation is to be used for all RPG IV programs, place the keyword DEBUG(*YES) in the data area. The data area can be whatever size is required to accommodate the keywords specified. The data area must not have an H in position 6.

Control Specification Statement

The control specification consists solely of keywords. The keywords can be placed anywhere between positions 7 and 80. Positions 81-100 can be used for comments.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++++
```

Figure 73. Control Specification Layout

The following is an example of a control specification.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H ALTSEQ(*EXT) CURSYM('$') DATEDIT(*MDY) DATFMT(*MDY/) DEBUG(*YES)
H DECEDIT('.') FORMSALIGN(*YES) FTRANS(*SRC) DFTNAME(name)
H TIMFMT(*ISO)
```

Position 6 (Form Type)

An H must appear in position 6 to identify this line as the control specification.

Positions 7-80 (Keywords)

The control specification keywords are used to determine how the program will deal with devices and how certain types of information will be represented.

ALTSEQ{(*NONE *SRC *EXT)}

If this keyword is not specified or specified with *NONE, the normal collating sequence is used. If the keyword is specified without parameters or with the parameter *SRC, the alternate collating sequence table specified in the program is used. If specified with the parameter *EXT, the alternate collating sequence table is specified outside the program. The collating sequence table is dependent on the SRTSEQ and LANGID parameters. If the keyword is not specified or specified with *NONE or *EXT, an alternate collating sequence table must not be specified in the program.

CURSYM('sym')

This keyword specifies a character used as a currency symbol in editing. The symbol must be a single character enclosed in quotes. Any character in the RPG character set (see Chapter 1, "Symbolic Names and Reserved Words" on page 3) may be used except:

- 0 (zero)
- * (asterisk)
- , (comma)
- & (ampersand)
- . (period)
- – (minus sign)
- C (letter C)
- R (letter R)
- Blank

If the keyword is not specified, $ (dollar sign) will be used as the currency symbol.

DATEDIT(fmt{separator})

This keyword specifies the format of numeric fields when using the Y edit code. The separator character is optional. The value (fmt) can be *DMY, *MDY, or *YMD. The default separator is /. A separator character of & (ampersand) may be used to specify a blank separator.

DATFMT(fmt{separator})

Specifies the date format for date literals and the default format for date fields within the program. The default is *ISO format. Table 18 on page 175 describes the various date formats and their separators.

Table 18. External Date formats for Date data type

RPG name	Description	Format (Default Separator)	Valid Separators	Length	Example
*MDY	Month/Day/Year	mm/dd/yy	/ - . , '&'	8	01/15/91
*DMY	Day/Month/Year	dd/mm/yy	/ - . , '&'	8	15/01/91
*YMD	Year/Month/Day	yy/mm/dd	/ - . , '&'	8	91/01/15
*JUL	Julian	yy/ddd	/ - . , '&'	6	91/015
*ISO	International Standards Organization	yyyy-mm-dd	-	10	1991-01-15
*USA	IBM USA Standard	mm/dd/yyyy	/	10	01/15/1991
*EUR	IBM European Standard	dd.mm.yyyy	.	10	15.01.1991
*JIS	Japanese Industrial Standard Christian Era	yyyy-mm-dd	-	10	1991-01-15

DEBUG{(*NO *YES)}

If this keyword is not specified or specified with *NO, DUMP operations are not performed. If specified, optionally with *YES, DUMP operations will be performed.

DECEDIT('value')

This keyword specifies the character used as the decimal point for edited decimal numbers. Leading zeros are printed when the absolute value of the number is less than 1. The possible values are

'.'	Decimal point is period; leading zero not printed (.123)
','	Decimal point is comma; leading zero not printed (,123)
'0.'	Decimal point is period; leading zero printed (0.123)
'0,'	Decimal point is comma; leading zero printed (0,123)

The default value is '.' (period).

DFTNAME(rpg_name)

When used this keyword specifies the default program or module name. If not specified, the default name will be RPGPGM or RPGMOD respectively for a program or module. This name can be overridden outside the program. The RPG rules for names (see “Symbolic Names” on page 3) apply.

FORMSALIGN{(*NO *YES)}

When this keyword is specified, the first line of an output file conditioned with the 1P indicator can be printed repeatedly, allowing you to align the printer. If not specified or specified with *NO, no alignment will be performed. If specified, optionally with *YES, first page forms alignment will occur.

Rules for Forms Alignment:

a. The records specified on Output Specifications for a file with a device entry for a printer type device conditioned by the first page indicator (1P) may be written as many times as desired. The line will print once. The operator will then have the option to print the line again or continue with the rest of the program.

b. All spacing and skipping specified will be performed each time the line is printed.

c. When the option to continue with the rest of the program is selected, the line will not be reprinted.

d. The function may be performed for all printer files.

e. If a page field is specified, it will be incremented only the first time the line is printed.

f. When the continue option is selected, the line count will be the same as if the function were performed only once when line counter is specified.

FTRANS{(*NONE *SRC)}

If this keyword is specified, optionally with *SRC, file translation will take place and the translate table must be specified in the program. If not specified or specified with *NONE, no file translation will take place and the translate table must not be present.

TIMFMT(fmt{separator})

This keyword specifies the format of time literals and the default format for time fields in the program. The default is *ISO.

The following table shows the time formats supported and their separators:

Table 19. External Time formats for Time data type

RPG name	Description	Format (Default Separator)	Valid Separators	Length	Example
*HMS	Hours:Minutes:Seconds	hh:mm:ss	: . , &	8	14:00:00
*ISO	International Standards Organization	hh.mm.ss	.	8	14.00.00
*USA	IBM USA Standard. AM and PM can be any mix of upper and lower case.	hh:mm AM or hh:mm PM	:	8	02:00 PM
*EUR	IBM European Standard	hh.mm.ss	.	8	14.00.00
*JIS	Japanese Industrial Standard Christian Era	hh:mm:ss	:	8	14:00:00

Chapter 15. File Description Specifications

File description specifications identify each file used by a program. Each file in a program must have a corresponding file description specification statement.

A file can be either **program-described** or **externally-described**. In program-described files, record and field descriptions are included within the RPG program (using input and output specifications). Externally-described files have their record and field descriptions defined externally using DDS, IDDU, or SQL/400 commands.

The following limitations apply per program:

- Only one primary file can be specified. The presence of a primary file is not required.
- Only one record-address file.
- A maximum of eight PRINTER files.
- There is no limit for the maximum number of files allowed.

File Description Specification Statement

The general layout for the file description specification is as follows:

- the file description specification type (F) is entered in position 6
- the non-commentary part of the specification extends from position 7 to position 80
  - the fixed-format entries extend from positions 7 to 42
  - the keyword entries extend from positions 44 to 80
- the comments section of the specification extends from position 81 to position 100

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++++
```

Figure 74. File Description Specification Layout

File Description Continuation Line

If additional space is required for keywords, the keywords field can be continued on subsequent lines as follows:

- position 6 of the continuation line must contain an F
- positions 7 to 43 of the continuation line must be blank
- the specification continues on or past position 44

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
F.....................................Keywords+++++++++++++++++++++++++++++Comments++++++++++++
```

Figure 75. File Description Continuation Line Layout

Position 6 (Form Type)

An F must be entered in this position for file-description specifications.

Positions 7-16 (File Name)

Entry	**Explanation**
A valid file name	Every file used in a program must have a unique name. The file name can be from 1 to 10 characters long, and must begin in position 7.

The file name specified in positions 7 through 16 must be an existing file name that has been defined to the OS/400 system. However, one of the OS/400 system override commands can be used to associate the RPG IV file name to the file name defined to the OS/400 system.

For an externally-described file, the file must exist at both compilation time and at run time. For a program-described file, the file need exist only at run time.

When the files are opened at run time, they are opened in the reverse order to that specified in the file-description specifications. The RPG IV device name defines the operations that can be processed on the associated file.

Program Described File

For program-described files, the file name entered in positions 7 through 16 must also be entered on:

- Input specifications if the file is a primary, secondary, or full procedural file
- Output specifications or an output calculation operation line if the file is an output, update, or combined file, or if file addition is specified for the file
- Definition specifications if the file is a table or array file.
- Calculation specifications if the file name is required for the operation code specified

Externally Described File

For externally described files, the file name entered in positions 7 through 16 is the name used to locate the record descriptions for the file. The following rules apply to externally described files:

- Input and output specifications for externally described files are optional. They are required only if you are adding RPG IV functions, such as control fields or record identifying indicators, to the external description retrieved.
- When an external description is retrieved, the record definition can be referred to by its record format name on the input, output, or calculation specifications.
- A record format name must be a unique symbolic name.
- RPG IV does not support an externally-described logical file with two record formats of the same name. However, such a file can be accessed if it is program described.

Position 17 (File Type)

Entry	Explanation
I	Input file
O	Output file
U	Update file
C	Combined (input/output) file.

Input Files

An input file is one from which a program reads information. It can contain data records, arrays, or tables, or it can be a record-address file.

Output Files

An output file is a file to which information is written.

Update Files

An update file is an input file whose records can be read and updated. Updating alters the data in one or more fields of any record contained in the file and writes that record back to the same file from which it was read. If records are to be deleted, the file must be specified as an update file.

Combined Files

A combined file is both an input file and an output file. When a combined file is processed, the output record contains only the data represented by the fields in the output record. This differs from an update file, where the output record contains the input record modified by the fields in the output record.

A combined file is valid for a SPECIAL or WORKSTN file. A combined file is also valid for a DISK or SEQ file if position 18 contains T (an array or table replacement file).

Position 18 (File Designation)

Entry	Explanation
Blank	Output file
P	Primary file
S	Secondary file
R	Record address file
T	Array or table file
F	Full procedural file

Primary File

When several files are processed by cycle processing, one must be designated as the primary file. In multi-file processing, processing of the primary file takes precedence. Only one primary file is allowed per program.

Secondary File

When more than one file is processed by the RPG cycle, the additional files are specified as secondary files. Secondary files must be input capable (input, update, or combined file type). The processing of secondary files is determined by the order in which they are specified in the file-description specifications and by the rules of multi-file logic.

Record Address File (RAF)

A record-address file is a sequentially organized file used to select records from another file. Only one file in a program can be specified as a record-address file. This file is described on the file-description specification and not on the input specifications. A record-address file must be program-described; however, a record-address file can be used to process a program described file or an externally described file.

The file processed by the record-address file must be a primary, secondary, or full-procedural file, and must also be specified as the parameter to the RAFDATA keyword on the file-description specification of the record-address file.

You cannot specify a record-address file for the device SPECIAL.

A record-address file that contains relative-record numbers must also have a T specified in position 35 and an F in position 22.

Array or Table File

Array and table files specified by a T in position 18 are loaded at program initialization time. The array or table file can be input or combined. Leave this entry blank for array or table output files. You cannot specify SPECIAL as the device for array and table input files. You cannot specify an externally described file as an array or table file.

If T is specified in position 18, you can specify a file type of combined (C in position 17) for a DISK or SEQ file. A file type of combined allows an array or table file to be read from or written to the same file (an array or table replacement file). In addition to a C in position 17, the filename in positions 7-16 must also be specified as the parameter to the TOFILE keyword on the definition specification.

Full Procedural File

A full procedural file is not processed by the RPG cycle: input is controlled by calculation operations. File operation codes such as CHAIN or READ are used to do input functions.

Position 19 (End of File)

Entry	Explanation
E	All records from the file must be processed before the program can end. This entry is not valid for files processed by a record-address file. All records from all files which use this option must be processed before the LR indicator is set on by the RPG cycle to end the program.
Blank	If position 19 is blank for all files, all records from all files must be processed before end of program (LR) can occur. If position 19 is not blank for all files, all records from this file may or may not be processed before end of program occurs in multi-file processing.

Use position 19 to indicate whether the program can end before all records from the file are processed. An E in position 19 applies only to input, update, or combined files specified as primary, secondary, or record-address files.

If the records from all primary and secondary files must be processed, position 19 must be blank for all files or must contain E's for all files. For multiple input files, the end-of-program (LR) condition occurs when all input files for which an E is specified in position 19 have been processed. If position 19 is blank for all files, the end-of-program condition occurs when all input files have been processed.

When match fields are specified for two or more files and an E is specified in position 19 for one or more files, the LR indicator is set on after:

- The end-of-file condition occurs for the last file with an E specified in position 19.
- The program has processed all the records in other files that match the last record processed from the primary file.
- The program has processed the records in those files without match fields up to the next record with non-matching match fields.

When no file or only one file contains match field specifications, no records of other files are processed after end of file occurs on all files for which an E is specified in position 19.

Position 20 (File Addition)

Position 20 indicates whether records are to be added to an input or update file. For output files, this entry is ignored.

Entry	**Explanation**
Blank	No records can be added to an input or update file (I or U in position 17).
A	Records are added to an input or update file when positions 18 through 20 of the output record specifications for the file contain "ADD", or when the WRITE operation code is used in the calculation specification.

See Table 20 for the relationship between position 17 and position 20 of the file-description specifications and positions 18 through 20 of the output specifications.

Table 20. Processing Functions for Files

Function	**Specification**		
	File Description		**Output**
	Position 17	**Position 20**	**Positions 18-20**
Create new file[1] or Add records to existing file	O O	Blank A	Blank ADD
Process file	I	Blank	Blank
Process file and add records to the existing file	I	A	ADD
Process file and update the records (update or delete)	U	Blank	Blank
Process file and add new records to an existing file	U	A	ADD
Process file and delete an existing record from the file	U	Blank	DEL

[1] Within RPG, the term *create a new file* means to add records to a newly created file. Thus, the first two entries in this table perform the identical function. Both are listed to show that there are two ways to specify that function.

Position 21 (Sequence)

Entry	**Explanation**
A or blank	Match fields are in ascending sequence.
D	Match fields are in descending sequence.

Position 21 specifies the sequence of input fields used with the match fields specification (positions 65 and 66 of the input specifications). Position 21 applies only to input, update, or combined files used as primary or secondary files. Use positions

65 and 66 of the input specifications to identify the fields containing the sequence information.

If more than one input file with match fields is specified in the program, a sequence entry in position 21 can be used to check the sequence of the match fields and to process the file using the matching record technique. The sequence need only be specified for the first file with match fields specified. If sequence is specified for other files, the sequence specified must be the same; otherwise, the sequence specified for the first file is assumed.

If only one input file with match fields is specified in the program, a sequence entry in position 21 can be used to check fields of that file to ensure that the file is in sequence. By entering one of the codes M1 through M9 in positions 65 and 66 of the input specifications, and by entering an A, blank, or D in position 21, you specify sequence checking of these fields.

Sequence checking is required when match fields are used in the records from the file. When a record from a matching input file is found to be out of sequence, the RPG IV exception/error handling routine is given control.

Position 22 (File Format)

Entry	**Explanation**
F	Program described file
E	Externally described file

An F in position 22 indicates that the records for the file are described within the RPG IV program on input/output specifications (except for array/table files and record-address files).

An E in position 22 indicates that the record descriptions for the file are external to the RPG IV source program. The compiler obtains these descriptions at compilation time and includes them in the source program.

Positions 23-27 (Record Length)

Use positions 23 through 27 to indicate the length of the logical records contained in a program-described file. The maximum record size that can be specified is 32766; however, record-size constraints of any device may override this value. This entry must be blank for externally described files.

If the file being defined is a record-address file and the record length specified is 3, it is assumed that each record in the file consists of a 3-byte binary field for the relative-record numbers starting at offset 0. If the record length is 4 or greater, each relative-record number in the record-address file is assumed to be a 4-byte field starting at offset 1. If the record length is left blank, the actual record length is retrieved at run time to determine how to handle the record-address file.

If the file opened at run time has a primary record length of 3, then 3-byte relative-record numbers (one per record) are assumed; otherwise, 4-byte relative-record numbers are assumed. This support can be used to allow RPG IV programs to use System/36 environment SORT files as record-address files.

Table 21. Valid Combinations for a Record Address File containing Relative Record Numbers (RAFRRN)

Record Length Positions 23-27	RAF Length Positions 29-33	Type of Support
Blank	Blank	Support determined at run time.
3	3	System/36 support.
> = 4	4	Native support.

Position 28 (Limits Processing)

Entry	Explanation
L	Sequential-within-limits processing by a record-address file
Blank	Sequential or random processing

Use position 28 to indicate whether the file is processed by a record-address file that contains limits records.

A record-address file used for limits processing contains records that consist of upper and lower limits. Each record contains a set of limits that consists of the lowest record key and the highest record key from the segment of the file to be processed. Limits processing can be used for keyed files specified as primary, secondary, or full procedural files.

The L entry in position 28 is valid only if the file is processed by a record-address file containing limits records. Random and sequential processing of files is implied by a combination of positions 18 and 34 of the file-description specifications, and by the calculation operation specified.

The operation codes "SETLL (Set Lower Limit)" on page 455 and "SETGT (Set Greater Than)" on page 451 can be used to position a file; however, the use of these operation codes does not require an L in this position.

For more information on limits processing, refer to the *ILE RPG/400 Programmer's Guide*.

Positions 29-33 (Length of Key or Record Address)

Entry **Explanation**

1-2000 The number of positions required for the key field in a program described file or the length of the entries in the record-address file (which must be a program-described file).

If the program-described file being defined uses keys for record identification, enter the number of positions occupied by each record key. This entry is required for indexed files.

If the keys are packed, the key field length should be the packed length; this is the number of digits in DDS divided by 2 plus 1 and ignoring any fractions.

If the file being defined is a record-address file, enter the number of positions that each entry in the record-address file occupies.

If the keys are graphic, the key field length should be specified in bytes (for example, 3 graphic characters requires 6 bytes).

Blank These positions must be blank for externally described files. (The key length is specified in the external description.) For a program-described file, a blank entry indicates that keys are not used. Positions 29-33 can also be blank for a record-address file with a blank in positions 23-27 (record length).

Position 34 (Record Address Type)

Entry **Explanation**

Blank Relative record numbers are used to process the file.

Records are read consecutively.

Record address file contains relative-record numbers.

For limits processing, the record-address type (position 34) is the same as the type of the file being processed.

A Character keys (valid only for program-described files specified as indexed files or as a record-address-limits file).

P Packed keys (valid only for program-described files specified as indexed files or as a record-address-limits file).

G Graphic keys (valid only for program-described files specified as indexed files or as a record-address-limits file).

K Key values are used to process the file. This entry is valid only for externally described files.

D Date keys are used to process the file. This entry is valid only for program-described files specified as indexed files or as a record-address-limits file.

T Time keys are used to process the file. This entry is valid only for program-described files specified as indexed files or as a record-address-limits file.

Z Timestamp Keys are used to process the file. This entry is valid only for program-described files specified as indexed files or as a record-address-limits file.

Blank = Non-keyed Processing

A blank indicates that the file is processed without the use of keys, that the record-address file contains relative-record numbers (a T in position 35), or that the keys in a record-address-limits file are in the same format as the keys in the file being processed.

A file processed without keys can be processed consecutively or randomly by relative-record number.

Input processing by relative-record number is determined by a blank in position 34 and by the use of the CHAIN, SETLL, or SETGT operation code. Output processing by relative-record number is determined by a blank in position 34 and by the use of the RECNO keyword on the file description specifications.

A = Character Keys

The indexed file (I in position 35) defined on this line is processed by character-record keys. (A numeric field used as the search argument is converted to zoned decimal before chaining.) The A entry must agree with the data format of the field identified as the key field (length in positions 29 to 33 and starting position specified as the parameter to the KEYLOC keyword).

The record-address-limits file (R in position 18) defined on this line contains character keys. The file being processed by this record address file can have an A, P, or K in position 34.

P = Packed Keys

The indexed file (I in position 35) defined on this line is processed by packed-decimal-numeric keys. The P entry must agree with the data format of the field identified as the key field (length in positions 29 to 33 and starting position specified as the parameter to the KEYLOC keyword).

The record-address-limits file defined on this line contains record keys in packed decimal format. The file being processed by this record address file can have an A, P, or K in position 34.

G = Graphic Keys

The indexed file (I in position 35) defined on this line is processed by graphic keys. Since each graphic character requires two bytes, the key length must be an even number. The record-address file which is used to process this indexed file must also have a 'G' specified in position 34 of its file description specification, and its key length must also be the same as the indexed file's key length (positions 29-33).

K = Key

A K entry indicates that the externally described file is processed on the assumption that the access path is built on key values. If the processing is random, key values are used to identify the records.

If this position is blank for a keyed file, the records are retrieved in arrival sequence.

D = Date Keys

The indexed file (I in position 35) defined on this line is processed by date keys. The D entry must agree with the data format of the field identified as the key field (length in positions 29 to 33 and starting position specified as the parameter to the KEYLOC keyword).

The hierarchy used when determining the format and separator for the date key is:

1. From the DATFMT keyword specified on the file description specification
2. From the DATFMT keyword specified in the control specification
3. *ISO

T = Time Keys

The indexed file (I in position 35) defined on this line is processed by time keys. The T entry must agree with the data format of the field identified as the key field (length in positions 29 to 33 and starting position specified as the parameter to the KEYLOC keyword).

The hierarchy used when determining the format and separator for the time key is:

1. From the TIMFMT keyword specified on the file description specification
2. From the TIMFMT keyword specified in the control specification
3. *ISO

Z = Timestamp Keys

The indexed file (I in position 35) defined on this line is processed by timestamp keys. The Z entry must agree with the data format of the field identified as the key field (length in positions 29 to 33 and starting position specified as the parameter to the KEYLOC keyword).

For more information on record address type, refer to the *ILE RPG/400 Programmer's Guide*.

Position 35 (File Organization)

Entry	**Explanation**
Blank	The program-described file is processed without keys, or the file is externally described.
I	Indexed file (valid only for program-described files).
T	Record address file that contains relative-record numbers (valid only for program-described files).

Use position 35 to identify the organization of program described files.

Blank = Non-keyed Program-Described File

A program-described file that is processed without keys can be processed:

- Randomly by relative-record numbers, positions 28 and 34 must be blank.
- Entry Sequence, positions 28 and 34 must be blank.
- As a record-address file, position 28 must be blank.

I = Indexed File

An indexed file can be processed:

- Randomly or sequentially by key
- By a record-address file (sequentially within limits). Position 28 must contain an L.

T = Record Address File

A record-address file (indicated by an R in position 18) that contains relative-record numbers must be identified by a T in position 35. (A record-address file must be program described.) Each record retrieved from the file being processed is based on the relative record number in the record-address file. (Relative record numbers cannot be used for a record-address-limits file.)

Each relative-record number in the record-address file is a 4-byte binary field; therefore, each 4-byte unit of a record-address file contains a relative-record number. A minus one (-1 or hexadecimal FFFFFFFF) relative-record number value causes the record to be skipped. End of file occurs when all record-address file records have been processed.

For more information on how to handle record-address files, see the *ILE RPG/400 Programmer's Guide*.

Positions 36-42 (Device)

Entry	**Explanation**
PRINTER	File is a printer file, a file with control characters that can be sent to a printer.
DISK	File is a disk file. This device supports sequential and random read/write functions. These files can be accessed on a remote system by Distributed Data Management (DDM).
WORKSTN	File is a workstation file. Input/output is through a display or ICF file.
SPECIAL	This is a special file. Input or output is on a device that is accessed by a user-supplied program. The name of the program must be specified as the parameter for the PGMNAME keyword. A parameter list is created for use with this program, including an option code parameter and a status code parameter. The file must be a fixed unblocked format. See "PLIST(Plist_name)" on page 193 and "PGMNAME(program_name)" on page 193 for more information.
SEQ	File is a sequentially organized file. The actual device is specified in a CL command or in the file description, which is accessed by the file name.

Use positions 36 through 42 to specify the RPG IV device name to be associated with the file. The RPG IV device name defines the RPG IV functions that can be done on the associated file. Certain functions are valid only for a specific RPG IV device name (such as the EXFMT operation for WORKSTN). The file name specified in positions 7 through 16 can be overridden at run time, allowing you to change the input/output device used in the program.

Note that the RPG IV device names are not the same as the system device names.

Position 43 (Reserved)

Position 43 must be blank.

Positions 44-80 (Keywords)

Positions 44 to 80 are provided for file-description-specification keywords. Keywords are used to provide additional information about the file being defined. See "File Description Keywords" on page 189.

File Description Keywords

File-description-specification keywords can have no parameters, have optional parameters, or required parameters. The syntax for keywords is as follows:

```
Keyword{(parameter{:parameter....})}
```

- when one or more parameters are specified, the parameters are enclosed in a set of parentheses ()
- multiple parameters are separated by a colon ':'
- parameters which are shown enclosed in curly brackets {} are optional parameters (the curly brackets are shown for notational purposes only and must not be entered)

If additional space is required for keywords, the keyword field can be continued on subsequent lines. See "File Description Continuation Line" on page 178.

COMMIT{(rpg_name)}

COMMIT allows the user the option of processing files under commitment control. An optional parameter, rpg_name, may be specified. The parameter is implicitly defined as a field of type indicator (that is, a character field of length one), and is initialized by RPG to '0'.

By specifying the optional parameter, the programmer can control at run-time whether or not commitment control is enabled. If the parameter contains a '1', the file will be opened with the COMMIT indication on, otherwise the file will be opened without COMMIT. The parameter must be set prior to opening the file. If the file is opened at program initialization, the parameter can be passed in through a parameter or defined as an external indicator. If the file is opened explicitly, using the OPEN operation in the calculation specifications, it can be set prior to the OPEN operation.

Use the COMMIT and ROLBK operation codes to group changes to this file and other files currently under commitment control so that changes all happen together, or do not happen at all.

For shared opens, if the file is already open, this keyword has no effect.

DATFMT(format{separator})

DATFMT allows the specification of a default date format and a default separator (which is optional). If the file, on which this keyword is specified, is indexed and the keyfield is a date, then this also provides the default format for the keyfield.

For date Input fields this specifies the default external date format/separator (Input specification positions 31-35).

For date Output fields this specifies the default external date format/separator (Output specification positions 53-57).

For a Record-Address file this specifies the date format of date limits keys read from the RAF file.

See Table 18 on page 175 for valid formats and separators.

DEVID(fieldname)

The fieldname parameter contains the name of the program device that supplied the record processed in the file. The field is updated each time a record is read from a file. Also, you may move a program device name into this field to direct an output or device-specific input operation (other than a READ-by-file-name or an implicit cycle read) to a different device.

The fieldname is implicitly defined as a 10-character alphanumeric field. The device name specified in the field must be left-justified and padded with blanks. Initially, the field is blank. A blank field indicates the requester device. If the requester device is not acquired for your file, you must not use a blank field.

The DEVID field is maintained for each call to a program. If you call program B from within program A, the DEVID field for program A is not affected. Program B uses a separate DEVID field. When you return to program A, its DEVID field has the same value as it had before you called program B. If program B needs to know which devices are acquired to program A, program A must pass this information (as a parameter list) when it calls program B.

When you specify the DEVID keyword but not the MAXDEV keyword the RPG IV program assumes a multiple device file (MAXDEV with a parameter of *FILE).

To determine the name of the requester device, you may look in the appropriate area of the file information data structure (see "File Information Data Structure" on page 61). Or, you may process one of the input or output operations described above with the fieldname blank. After the operation, the fieldname has the name of the requester device.

EXTIND(*INUx)

This keyword indicates to the compiler that the file should only be a candidate for OPEN if the external indicator is set in the job.

EXTIND lets the programmer control the operation of input, output, update, and combined files at run time. If the specified indicator is on at program initialization, the file is opened. If the indicator is not on, the file is not opened and is ignored during processing. The *INU1 through *INU8 indicators can be set as follows:

- By the OS/400 control language.
- When used as a resulting indicator for a calculation operation or as field indicators on the input specifications. Setting the *INU1 through *INU8 indicators in this manner has no effect on file conditioning.

  See also "USROPN" on page 197.

FORMLEN(number)

Use the FORMLEN keyword to specify the form length of a PRINTER file. The form length must be greater than or equal to 1 and less than or equal to 255. The parameter specifies the exact number of lines available on the form or page to be used.

Changing the form length does not require recompiling the program. The override to the compiled value can be specified by an OS/400 system override command (PAGESIZE parameter on the override with printer file (OVRPRTF) command).

When the FORMLEN keyword is specified, the FORMOFL keyword must also be specified.

FORMOFL(number)

The FORMOFL keyword is used to specify the overflow line number. The overflow line number must be less than or equal to the form length. When the line that is specified as the overflow line is printed, the overflow indicator is set on.

In the OS/400 system, changing the overflow line does not require recompiling the program. The override to the compiled value can be specified by an OS/400 system override command (OVRFLW parameter on the override with printer file (OVRPRTF) command).

When the FORMOFL keyword is specified, the FORMLEN keyword must also be specified.

IGNORE(recformat{:recformat...})

This keyword lets you ignore a record format from an externally described file. The external name of the record format to be ignored is specified as the parameter recformat. One or more record formats can be specified, separated by colons (:). The program runs as if the specified record format(s) did not exist. All other record formats contained in the file will be included.

When the IGNORE keyword is specified for a file, the INCLUDE keyword cannot be specified.

INCLUDE(recformat{:recformat...})

This keyword can be used to include only those record format names specified on the INCLUDE keyword. All other record formats contained in the file will be excluded. For WORKSTN files, the record formats specified using the SFILE keyword are also included in the program, they need not be specified twice. Multiple record formats can be specified, separated by colons (:).

When the INCLUDE keyword is specified for a file, the IGNORE keyword cannot be specified.

INFDS(DSname)

This keyword lets you define and name a data structure to contain the feedback information associated with the file. The data structure name is specified as the parameter for INFDS. If INFDS is specified for more than one file, each associated data structure must have a unique name.

For additional information on file information data structures, see “File Information Data Structure” on page 61.

INFSR(SUBRname)

The file exception/error subroutine specified as the parameter to this keyword may receive control following file exception/errors. The subroutine name may be *PSSR, which indicates the user defined program exception/error subroutine is to be given control for errors on this file.

KEYLOC(number)

Use the KEYLOC keyword to specify the record position in which the key field for a program-described indexed-file begins. The parameter must be between 1 and 32766.

The key field of a record contains the information that identifies the record. The key field must be in the same location in all records in the file.

MAXDEV(*ONLY/*FILE)

MAXDEV specifies the maximum number of devices defined for the WORKSTN file. The default, *ONLY, indicates a single device file. If *FILE is specified, the maximum number of devices (defined for the WORKSTN file on the create-file command) is retrieved at file open, and SAVEIND and SAVEDS space allocation will be done at runtime.

With a shared file, the MAXDEV value is not used to restrict the number of acquired devices.

When you specify DEVID, SAVEIND, or SAVEDS but not MAXDEV, the RPG IV program assumes the default of a multiple device file (MAXDEV with a parameter of *FILE).

OFLIND(*INxx)

Use OFLIND to specify an overflow indicator to condition which lines in the PRINTER file will be printed when overflow occurs. This entry is valid only for a PRINTER device. Default overflow processing (that is, automatic page eject at overflow) is done if the OFLIND keyword is not specified.

Valid Parameters:

*INOA-*INOG, *INOV: Specified overflow indicator conditions the lines to be printed when overflow occurs on a program described printer file.

*IN01-*IN99: Set on when a line is printed on the overflow line, or the overflow line is reached or passed during a space or skip operation.

Note: Indicators *INOA through *INOG, and *INOV are not valid for externally described files.

Only one overflow indicator can be assigned to a file. If more than one PRINTER file in a program is assigned an overflow indicator, that indicator must be unique for each file.

PASS(*NOIND)

Specify PASS(*NOIND) on the file description specification line for a program described WORKSTN file if you are taking responsibility for passing indicators on input and output.

With PASS(*NOIND), the RPG IV language does not pass indicators to data management on output and does not receive them on input. Indicators are passed by describing them as fields (in the form *INxx, *IN(xx), or *IN) in the input or output record. They must be specified in the sequence required by the data description specifications (DDS). You can use the DDS listing to determine this sequence.

If this keyword is not specified the indicators are passed based on the DDS keyword INDARA. If the INDARA file keyword is specified, indicators are passed in a separate indicator area, otherwise they are passed in the buffer preceding the data fields.

Note: Either the PASS(*NOIND) keyword on the file definition specification or the INDARA keyword on the DDS may be specified - not both. If PASS(*NOIND) is specified, then the INDARA keyword cannot be specified in the DDS. Alternatively, if PASS(*NOIND) is not specified in the RPG program, the INDARA keyword must be specified on the DDS.

PGMNAME(program_name)

When SPECIAL is the device entry (positions 36 through 42), the program specified as the parameter to PGMNAME handles the support for the special I/O device.

Note: The parameter must be a valid program name and not a bound procedure name.

See “Positions 36-42 (Device)” on page 188 and “PLIST(Plist_name)” for more information.

PLIST(Plist_name)

PLIST supplies, as its parameter, the name of the parameter list to be passed to the program for the SPECIAL file. The program is specified using the PGMNAME keyword, see “PGMNAME(program_name).” This entry is valid only when the device specified (positions 36 to 42) in the file-description line is SPECIAL. The parameters identified by this entry are added to the end of the parameter list passed by the program.

PREFIX(prefix_name)

This keyword is allowed for an externally-described file. The characters specified as 'prefix_name' are prefixed to the names of all fields defined in all records of the file specified in positions 7-16.

Rules:

- Fields that are explicitly renamed on the Input specifications are not affected by this keyword.
- The total length of the name after applying the prefix must not exceed the maximum length of an RPG field name.

PRTCTL(data_struct{:*COMPAT})

The PRTCTL keyword specifies the use of dynamic printer control. The data structure specified as the parameter data_struct refers to the forms control information and line count value. The PRTCTL keyword is valid only for a program described file.

The optional parameter *COMPAT indicates that the data structure layout is compatible with RPG III. The default, *COMPAT not specified, will require the use of the extended length data structure.

Extended Length PRTCTL Data Structure

A minimum of 15 bytes is required for this data structure. Layout of the PRTCTL data structure is as follows:

Data Structure Positions	Subfield Contents
1-3	A three-position character field that contains the space-before value (valid entries: blank or 0-255)
4-6	A three-position character field that contains the space-after value (valid entries: blank or 0-255)
7-9	A three-position character field that contains the skip-before value (valid entries: blank or 1-255)
10-12	A three-position character field that contains the skip-after value (valid entries: blank or 1-255)
13-15	A three-digit numeric (zoned decimal) field with zero decimal positions that contains the current line count value.

*COMPAT PRTCTL Data Structure

Data Structure Positions	Subfield Contents
1	A one-position character field that contains the space-before value (valid entries: blank or 0-3)
2	A one-position character field that contains the space-after value (valid entries: blank or 0-3)
3-4	A two-position character field that contains the skip-before value (valid entries: blank, 1-99, A0-A9 for 100-109, B0-B2 for 110-112)
5-6	A two-position character field that contains the skip-after value (valid entries: blank, 1-99, A0-A9 for 100-109, B0-B2 for 110-112)
7-9	A three-digit numeric (zoned decimal) field with zero decimal positions that contains the current line count value.

The values contained in the first four subfields of the extended length data structure are the same as those allowed in positions 40 through 51 (space and skip entries) of the output specifications. If the space and skip entries (positions 40 through 51) of the output specifications are blank, and if subfields 1 through 4 are also blank, the default is to space 1 after. If the PRTCTL option is specified, it is used only for the output records that have blanks in positions 40 through 51. You can control the space and skip value (subfields 1 through 4) for the PRINTER file by changing the values in these subfields while the program is running.

Subfield 5 contains the current line count value. The RPG IV compiler does not initialize subfield 5 until after the first output line is printed. The RPG IV compiler then changes subfield 5 after each output operation to the file.

RAFDATA(filename)

If a file is defined as a Record Address File (RAF) (an R in position 18) the RAFDATA keyword is used to specify the name of the input or update file that contains the data records to be processed. (See “Record Address File (RAF)” on page 180 for further information.)

RECNO(fieldname)

This keyword is optional for DISK files to be processed by relative-record number. The RECNO keyword must be specified for output files processed by relative-record number, output files that are referenced by a random WRITE calculation operation, or output files that are used with ADD on the output specifications.

The RECNO keyword can be specified for input/update files. The relative-record number of the record retrieved is placed in the 'fieldname', for all operations that reposition the file (such as READ, SETLL, or OPEN). It must be defined as numeric with zero decimal positions. The field length must be sufficient to contain the longest record number for the file.

The compiler will not open a SEQ or DISK file for blocking or unblocking records if the RECNO keyword is specified for the file.

Note: When the RECNO keyword is specified for input or update files with file-addition ('A' in position 20), the value of the fieldname parameter must refer to a relative-record number of a deleted record, for the output operation to be successful.

RENAME(Ext_format:Int_format)

This keyword allows you to rename record formats in an externally described file. The external name of the record format that is to be renamed is entered as the Ext_format parameter. The Int_format parameter is the name of the record as it is used in the program. The external name is replaced by this name in the program.

SAVEDS(DSname)

The SAVEDS keyword allows the specification of the data structure saved and restored for each device. Before an input operation, the data structure for the device operation is saved. After the input operation, the data structure for the device associated with this current input operation is restored. This data structure cannot be a data area data structure, file information data structure, or program status data structure, and it cannot contain a compile-time array or prerun-time array.

If the SAVEDS keyword is not specified, no saving and restoring is done. SAVEDS must not be specified for shared files.

When you specify SAVEDS but not MAXDEV, the RPG IV program assumes a multiple device file (MAXDEV with a parameter of *FILE).

SAVEIND(number)

Indicators from 01 to the number specified are saved and restored for each device attached to a mixed or multiple device file. Before an input operation, the indicators for the device associated with the previous input or output operation are saved. After the input operation, the indicators for the device associated with this current input operation are restored. Specify a number from 1 through 99, as the parameter to the SAVEIND keyword. No indicators are saved and restored if the SAVEIND keyword is not specified, or if the MAXDEV keyword is not specified or specified with the parameter *ONLY.

If you specified the DDS keyword INDARA, the number you specify for the SAVEIND keyword must be less than any response indicator you use in your DDS. For example, if you specify INDARA and CF01(55) in your DDS, the maximum value for the SAVEIND keyword is 54. The SAVEIND keyword must not be used with shared files.

When you specify the SAVEIND keyword but not the MAXDEV keyword, the RPG IV program assumes a multiple device file.

SFILE(recformat:rrnfield)

If the main file-description line contains E in position 22 and WORKSTN in positions 36 through 42, the SFILE keyword must be used to define any subfiles to be used in the file. The recformat parameter must specify the RPG IV name of the record format to be processed as a subfile.

The rrnfield parameter must specify the name of the relative-record number field for this subfile. The relative-record number of any record retrieved by a READC or CHAIN operation is placed into the field named as the rrnfield. This field is also used to specify the record number that RPG IV uses for a WRITE operation to the subfile or for output operations that use ADD. The field name specified as the parameter rrnfield must be defined as numeric with zero decimal positions. The field must have enough positions to contain the largest record number for the file. (See the SFLSIZ keyword in the *DDS Reference*.)

Relative record number processing is implicitly defined as part of the SFILE definition. If multiple subfiles are defined, each subfile requires the specification of the SFILE keyword.

Do not use the SFILE keyword with the SLN keyword.

SLN(number)

The SLN (Start Line Number) keyword determines where a record format is written to a display file. The main file-description line must contain WORKSTN in positions 36 through 42 and a C or O in positions 17. The DDS for the file must specify the keyword SLNO(*VAR) for one or more record formats. When you specify the SLN keyword, the parameter will automatically be defined in the program as a numeric field with length of 2 and with 0 decimal positions.

Do not use the SLN keyword with the SFILE keyword.

TIMFMT(format{separator})

The TIMFMT keyword allows the specification of a default time format and a default separator (which is optional). If the file, on which this keyword is specified, is indexed and the keyfield is a time, then the time format specified also provides the default format for the keyfield.

For time Input fields this specifies the default external time format/separator (Input specification positions 31-35).

For time Output fields this specifies the default external time format/separator (Output specification positions 53-57).

For a Record-Address file this specifies the time format of time limits keys read from the RAF file.

See Table 19 on page 176 for valid format and separators.

USROPN

The USROPN keyword causes the file not to be opened at program initialization. This gives the programmer control of the file's first open. The file will have to be explicitly opened using the OPEN operation in the calculation specifications. This keyword is not valid for input files designated as primary, secondary, table, or record-address files, or for output files conditioned by the 1P (first page) indicator.

The USROPN keyword is required for programmer control of only the first file opening. For example, if a file is opened and later closed by the CLOSE operation, the programmer can reopen the file (using the OPEN operation) without having specified the USROPN keyword on the file description specification.

See also “EXTIND(*INUx)” on page 190.

File Types and Processing Methods

Table 22 shows the valid entries for positions 28, 34, and 35 of the file-description specifications for the various file types and processing methods. The methods of disk file processing include:

- Relative-record-number processing
- Consecutive processing
- Sequential-by-key processing
- Random-by-key processing
- Sequential-within-limits processing.

Table 22. Processing Methods for DISK Files

Access	Method	Opcode	Position 28	Position 34	Position 35	Explanation
Random	RRN	CHAIN	Blank	Blank	Blank	Access by physical order of records
Seq	Key	READ READE READP READPE cycle	Blank	Blank	I	Access by key sequentially
Seq	Within Limits	READ READE READP READPE cycle	L	A, P, G, D, T, or Z	I	Access by key sequentially controlled by record-address-limits file
Seq	RRN	READ cycle	Blank	Blank	T	Access sequentially restricted to RRN numbers in RAF file

For further information on the various file processing methods, see the section entitled "Methods for Processing Disk Files", in the chapter "Accessing Database Files" in the *ILE RPG/400 Programmer's Guide*.

Chapter 16. Definition Specification

Definition Specifications can be used to define data structures, data-structure subfields, standalone fields, and named constants.

On the definition specification **arrays and tables** can be defined as either a data-structure subfield or a standalone field. For additional information on defining and using arrays and tables, see also Chapter 10, “Using Arrays and Tables” on page 133.

Built in functions (BIF) can be specified on definition specifications in the keyword field (as a parameter to a keyword). A built in function is allowed on the definition specification only if the values of all arguments are known at compile-time. When specified as parameters for the definition specification keywords DIM, OCCURS, OVERLAY, and PERRCD, all arguments for a BIF must be defined earlier in the specifications. For further information on using built in functions, see Chapter 20, “Built-in Functions” on page 261

For further information on data structures, constants, data types, and data formats, see also “Data” on page 99.

Definition Specification Statement

The general layout for the definition specification is as follows:

- the definition specification type (D) is entered in position 6
- the non-commentary part of the specification extends from position 7 to position 80
  - the fixed-format entries extend from positions 7 to 42
  - the keyword entries extend from positions 44 to 80
- the comments section of the specification extends from position 81 to position 100

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++++
```

Figure 76. Definition Specification Layout

Definition-Specification Continuation Line

If additional space is required for keywords, the keywords field can be continued on subsequent lines as follows:

- position 6 of the continuation line must contain a D
- positions 7 to 43 of the continuation line must be blank
- the specification continues on or past position 44

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
D.....................................Keywords+++++++++++++++++++++++++++++Comments++++++++++++
```

Figure 77. Definition-Specification Continuation-Line Layout

Position 6 (Form Type)

A D must be entered in this position for definition specifications.

Positions 7-21 (Name)

Entry	Explanation
Name	The name of the data structure, data-structure subfield, standalone field, or named constant to be defined.
Blank	Specifies filler fields in data-structure subfield definitions, or an unnamed data structure in data-structure definitions.

Use positions 7-21 to specify the name of the data item being defined. The normal rules for RPG IV symbolic names apply; reserved words cannot be used (see “Symbolic Names” on page 3). The name can begin in any position in the space provided. Thus, indenting can be used to indicate the shape of data in data structures.

For an externally-described subfield, a name specified here replaces the external-subfield name specified on the EXTFLD keyword.

Position 22 (External Description)

This position is used to identify a data structure or data-structure subfield as externally described. If a data structure or subfield is not being defined on this specification, then this field must be left blank.

Entry **Explanation for Data Structures**

E Identifies a data structure as externally described: subfield definitions are defined externally. If the EXTNAME keyword is not specified, positions 7-21 must contain the name of the externally described file containing the data structure definition.

Blank Program described: subfield definitions for this data structure follow this specification.

Entry **Explanation for Subfields**

E Identifies a data-structure subfield as externally described. The specification of an externally-described subfield is necessary only when keywords such as EXTFLD and INZ are required.

Blank Program described: the data-structure subfield is defined on this specification line.

Position 23 (Type of Data Structure)

This entry is used to identify the type of data structure being defined. If a data structure is not being defined, this entry must be left blank.

Entry **Explanation**

Blank The data structure being defined is not a program status or data-area data structure; or a data structure is not being defined on this specification

S Program status data structure. Only one data structure may be designated as the program status data structure.

U Data-area data structure.

RPG IV retrieves the data area at initialization and rewrites it at end of program.

- If the DTAARA keyword is specified, the parameter to the DTAARA keyword is used as the name of the external data area.
- If the DTAARA keyword is not specified, the name in positions 7-21 is used as the name of the external data area.
- If a name is not specified either by the DTAARA keyword, or by positions 7-21, *LDA (the local data area) is used as the name of the external data area.

Positions 24-25 (Type of Definition)

Entry **Explanation**

Blank The specification defines a data structure subfield.

DS The specification defines a data structure.

C (in column 24)
The specification defines a constant. Position 25 must be blank.

S (in column 24)
The specification defines a standalone field, array or table. Standalone fields allow you to define individual work fields, without requiring the definition of a data structure. The following is allowed for standalone fields:

- A standalone field has a specifiable internal data type.
- A standalone field may be defined as an array, table or field.
- Only length notation is allowed.

Named constant and standalone-field definition specifications may not be included within definition specifications for a data structure and its subfields.

For a list of valid keywords, grouped according to type of definition, please refer to Table 24 on page 217.

Positions 26-32 (From Position)

Positions 26-32 may only contain an entry if the location of a subfield within a data structure is being defined.

Entry **Explanation**

Blank A blank FROM position indicates that the value in the TO/LENGTH field specifies the length of the subfield, or that a subfield is not being defined on this specification line.

nnnnnnn Absolute starting position of the subfield within a data structure. The value specified must be from 1 to 32767 for a named data structure (and from 1 to 9999999 for an unnamed data structure), and right-justified in these positions.

Reserved Words
Reserved words for the program status data structure or for a file information data structure are allowed (left-justified) in the FROM-TO/LENGTH fields (positions 26-39). These special reserved words define the location of the subfields in the data structures. Reserved words for the program status data structure are *STATUS, *PROC, *PARM, and *ROUTINE. Reserved words for the file information data structure (INFDS) are *FILE, *RECORD, *OPCODE, *STATUS, and *ROUTINE.

Positions 33-39 (To Position / Length)

Entry **Explanation**

Blank If positions 33-39 are blank:

- a named constant is being defined on this specification line, or
- the standalone field or subfield is being defined LIKE another field, or
- the standalone field or subfield is of a type where a length is implied, or,
- the subfield's attributes are defined elsewhere, or

- a data structure is being defined. The length of the data structure is the maximum value of the subfield To-Positions.

nnnnnnn Positions 33-39 may contain a (right-justified) numeric value, from 1 to 32767 for a named data structure (and from 1 to 9999999 for an unnamed data structure), as follows:

- If the From field (position 26-32) contains a numeric value, then a numeric value in this field specifies the absolute end position of the subfield within a data structure.
- If the From field is blank, a numeric value in this field specifies :
  - the length of the entire data structure, or
  - the length of the standalone field, or
  - the length of the subfield. Within the data structure, this subfield is positioned such that its starting position is greater than the maximum to-position of all previously defined subfields in the data structure. Padding is inserted if the subfield is defined with type basing pointer or procedure pointer to ensure that the subfield is aligned properly.

Note: For graphic fields, the number specified here is the number of graphic characters, NOT the number of bytes (1 graphic character = 2 bytes). For numeric fields, the number specified here is the number of digits (for packed and zoned numeric fields: 1-30; for binary numeric fields: 1-9).

+|-nnnnn This entry is valid for standalone fields or subfields defined using the LIKE keyword. The length of the standalone field or subfield being defined on this specification line is determined by adding or subtracting the value entered in these positions to the length of the field specified as the parameter to the LIKE keyword.

Note: For graphic fields, the number specified here is the number of graphic characters, NOT the number of bytes (1 graphic character = 2 bytes). For numeric fields, the number specified here is the number of digits.

Reserved Words

If positions 26-32 are used to enter special reserved words, this field becomes an extension of the previous one, creating one large field (positions 26-39). This allows for reserved words, with names longer than 7 characters in length, to extend into this field. See "Positions 26-32 (From Position)" on page 202, 'Reserved Words'.

Position 40 (Internal Data Type)

This entry allows you to specify how a standalone field or data-structure subfield is stored internally. This entry pertains strictly to the internal representation of the data item being defined, regardless of how the data item is stored externally (that is, if it is stored externally).

Entry	Explanation
Blank	If the LIKE keyword is not specified: the item is being defined as character if the decimal positions entry is blank. If the decimal positions entry is not blank, the item is defined as packed numeric if it is a standalone field, or as zoned numeric if it is a subfield. **Note:** The entry must be blank when the LIKE keyword is specified.
A	Character
G	Graphic
T	Time
D	Date
Z	Timestamp
P	Numeric (Packed decimal format)
B	Numeric (Binary format)
S	Numeric (Zoned format)
*****	Basing pointer or procedure pointer

Positions 41-42 (Decimal Positions)

Positions 41-42 are used to indicate the number of decimal positions in a numeric subfield or standalone field. If the field is numeric, there must always be an entry in these positions; if there are no decimal positions, enter a 0.

Entry	Explanation
Blank	The value is not numeric or has been defined with the LIKE keyword.
0-30	Decimal positions: the number of positions to the right of the decimal in a numeric field.

This entry can only be supplied in combination with the TO/Length field. If the TO/Length field is blank, the value of this entry is defined somewhere else in the program (for example, through an externally described data base file).

Position 43 (Reserved)

Position 43 must be blank.

Positions 44-80 (Keywords)

Positions 44 to 80 are provided for definition-specification keywords. Keywords are used to describe and define data and its attributes. See "Definition-Specification Keywords" on page 205 for a description of each keyword.

Use this area to specify any keywords necessary to fully define the field.

Definition-Specification Keywords

Definition-specification keywords can have no parameters, optional parameters, or required parameters. The syntax for keywords is as follows:

```
Keyword{(parameter{:parameter....})}
```

- when one or more parameters are specified, the parameters are enclosed in a set of parentheses ()
- multiple parameters are separated by a colon ':'
- parameters which are shown enclosed in curly brackets {} are optional parameters (the curly brackets are shown for notational purposes only and must not be entered)

If additional space is required for keywords, the keyword field can be continued on subsequent lines. See "Definition-Specification Continuation Line" on page 200.

ALT(array_name)

Keyword indicating that the compile-time or pre-runtime array or table is in alternating format.

The array defined with the ALT keyword is the alternating array and the array name specified as the parameter is the main array. The alternate array definition may proceed or follow the main array definition.

The keywords on the main array define the loading for both arrays. The initialization data is in alternating order, beginning with the main array, as follows: main/alt/main/alt/...

In the alternate array definition, the PERRCD, FROMFILE, TOFILE, and CTDATA keywords are not valid.

ASCEND

Used to describe the sequence of the data in an array or table loaded at prerun-time or compile time. See also "DESCEND" on page 207.

Ascending sequence means that the array or table entries must start with the lowest data entry (according to the collating sequence) and go to the highest. Items with equal value are allowed.

A prerun-time array or table is checked for the specified sequence at the time the array or table is loaded with data. If the array or table is out of sequence, control passes to the RPG IV exception/error handling routine. A run-time array (loaded by input and/or calculation specifications) is not sequence checked.

When ALTSEQ(*EXT) is specified, the alternate collating sequence is used when checking the sequence of compile-time arrays or tables. If the alternate sequence is not known until run-time, the sequence is checked at run-time; if the array or table is out of sequence, control passes to the RPG IV exception/error handling routine.

A sequence (ascending or descending) must be specified if the LOOKUP operation is used to search an array or table for an entry to determine whether the entry is high or low compared to the search argument.

If the SORTA operation code is used with an array, and no sequence is specified, an ascending sequence is assumed.

BASED(basing_pointer_name)

When the BASED keyword is specified for a data structure or standalone field, a **basing pointer** is created using the name specified as the keyword parameter. This basing pointer holds the address (storage location) of the **based** data structure or standalone field being defined. In other words, the name specified in positions 7-21 is used to refer to the data stored at the location contained in the basing pointer.

Note: Before the based data structure or standalone field can be used, the basing pointer must be assigned a valid address.

If an array is defined as a based standalone field it must be a *runtime* array.

CONST(constant)

The CONST keyword is used to specify the value of a named constant. This keyword is optional (the constant value can be specified with or without the CONST keyword), and is only valid for named constant definitions (C in position 24).

The parameter must be a literal, figurative constant, or built-in-function. The constant may be continued on subsequent lines by adhering to the appropriate continuation rules (see “Continuation rules” on page 169 for further details).

If a named constant is used as a parameter for the keywords DIM, OCCURS, PERRCD, or OVERLAY, the named constant must be defined prior to its use.

CTDATA

The CTDATA keyword indicates that the array or table is loaded using compile-time data. The data is specified at the end of the program following the ** or **CTDATA(array/table name) specification.

When an array or table is loaded at compilation time, it is compiled along with the source program and included in the program. Such an array or table does not need to be loaded separately every time the program is run.

DATFMT(format{separator})

Specifies the internal date format for a Date field and optionally the separator character. This keyword will be automatically generated for an externally described data structure subfield of type Date and determined at compile time.

See Table 18 on page 175 for valid formats and separators.

The hierarchy used when determining the internal format and separator for a date array or field is:

1. From the DATFMT keyword specified on the definition specification
2. From the DATFMT keyword specified in the control specification
3. *ISO

DESCEND

Used to describe the sequence of the data in an array or table loaded at prerun-time or compile time. See also “ASCEND” on page 205.

Descending sequence means that the array or table entries must start with the highest data entry (according to the collating sequence) and go to the lowest. Items with equal value are allowed.

A prerun-time array or table is checked for the specified sequence at the time the array or table is loaded with data. If the array or table is out of sequence, control passes to the RPG IV exception/error handling routine. A run-time array (loaded by input and/or calculation specifications) is not sequence checked.

When ALTSEQ(*EXT) is specified, the alternate collating sequence is used when checking the sequence of compile-time arrays or tables. If the alternate sequence is not known until run-time, the sequence is checked at run-time; if the array or table is out of sequence, control passes to the RPG IV exception/error handling routine.

A sequence (ascending or descending) must be specified if the LOOKUP operation is used to search an array or table for an entry to determine whether the entry is high or low compared to the search argument.

If the SORTA operation code is used with an array, and no sequence is specified, an ascending sequence is assumed.

DIM(numeric_constant)

Used to define the number of elements in an array or table.

The numeric constant must have zero (0) decimal positions. It can be a literal, a named constant or a built-in function.

DTAARA{(data_area_name)}

The DTAARA keyword is used to associate a standalone field, data structure, data-structure subfield or data-area data structure with an external data area. The DTAARA keyword has the same function as the *DTAARA DEFINE operation code (see “*DTAARA DEFINE” on page 342).

If data_area_name is not specified then the name specified in positions 7-21 is also the name of the external data area. If data_area_name is specified then it must be a valid AS/400 data area name, including *LDA (for the local data area), and *PDA (for the program initialization parameters data area).

If the parameter is not specified and the data-structure name is not specified, then the default is *LDA.

When the DTAARA keyword is specified, the IN, OUT, and UNLOCK operation codes can be used on the data area.

EXPORT

The specification of the EXPORT keyword allows a data structure or standalone field defined within a module to be used by another module in the program. The storage for the data item is allocated in the module containing the EXPORT definition.

Note: The initialization for the storage occurs when the program entry procedure (of the program containing the module) is first called. RPG IV will not do any further initialization on this storage, even if the procedure ended with LR on, or ended abnormally on the previous call.

The following restrictions apply when EXPORT is specified:

- only one module may define the data item as exported
- you cannot export a field that is specified in the Result-Field entry of a PARM in the *ENTRY PLIST.
- unnamed data structures cannot be exported
- BASED data items cannot be exported
- Both IMPORT and EXPORT cannot be specified for the same data item.

For a multiple-occurrence data structure or table, each module will contain its own copy of the occurrence number or table index. An OCCUR or LOOKUP operation in any module will have a local impact since the occurrence number or index is local to each module.

See also “IMPORT” on page 209.

EXTFLD(field_name)

The EXTFLD keyword is used to rename a subfield in an externally described data structure. Enter the external name of the subfield as the parameter to the EXTFLD keyword, and specify the name to be used in the program in the Name field (positions 7-21).

The keyword is optional. If not specified, the name extracted from the external definition is used as the data-structure subfield name.

If the PREFIX keyword is specified for the data structure, the prefix will not be applied to fields renamed with EXTFLD.

EXTFMT(code)

Use the EXTFMT keyword to specify the external data type for compile-time and prerun-time numeric arrays and tables. The external data type is the format of the data in the records in the file. This entry has no effect on the format used for internal processing (internal data type) of the array or table in the program.

The possible values for the parameter are:

S The data for the array or table is in zoned decimal format.

P The data for the array or table is in packed decimal format.

B The data for the array or table is in binary format.

L The data for a numeric array or table element has a preceding (left) plus or minus sign.

R The data for a numeric array or table element has a following (right) plus or minus sign.

Notes:

1. If the EXTFMT keyword is not specified, the external format defaults to 'S'.
2. For compile-time arrays and tables, the only values allowed are S, L, and R.

EXTNAME(file_name{:format_name})

The EXTNAME keyword is used to specify the name of the file which contains the field descriptions used as the subfield description for the data structure being defined.

The file_name parameter is required. Optionally a format name may be specified to direct the compiler to a specific format within a file. If format_name parameter is not specified the first record format is used.

If the data structure definition contains an E in column 22, and the EXTNAME keyword is not specified, the name specified in positions 7-21 is used.

The compiler will generate the following Definition specification entries for all fields of the externally described data structure:

- Subfield name (Name will be the same as the external name, unless renamed by keyword EXTFLD or the PREFIX keyword is used to apply a prefix).
- Subfield length
- Subfield internal datatype (will be the same as the External type, unless the CVTOPT compile option is specified for the type. In that case the datatype will be character).

All data structure keywords are allowed with the EXTNAME keyword.

FROMFILE(file_name)

The FROMFILE keyword is used to specify the file with input data for the prerun-time array or table being defined. The FROMFILE keyword must be specified for every prerun-time array or table used in the program.

See also “TOFILE(file_name)” on page 215.

IMPORT

The IMPORT keyword specifies that storage for the data item being defined is allocated in another module, but may be accessed in this module.

If a name is defined as imported but no module in the program contains an exported definition of the name, an error will occur at link time. See “EXPORT” on page 208.

The following restrictions apply when IMPORT is specified:

- The data item may not be initialized (the INZ keyword is not allowed). The exporting module manages all initialization for the data.

- An imported field cannot be defined as a compile-time or prerun-time array or table, or as a data area. (Keywords CTDATA, FROMFILE, TOFILE, EXTFMT, PERRCD, and DTAARA are not allowed.)
- An imported field may not be specified as an argument to the RESET operation code since the initial value is defined in the exporting module.
- You cannot specify an imported field in the Result-Field entry of a PARM in the *ENTRY PLIST.
- You cannot define an imported field as based (the keyword BASED is not allowed).
- This keyword is not allowed for unnamed data structures.
- The only other keywords allowed are DIM, EXTNAME, LIKE, OCCURS, and PREFIX.

For a multiple-occurrence data structure or table, each module will contain its own copy of the occurrence number or table index. An OCCUR or LOOKUP operation in any module will have a local impact since the occurrence number or index is local to each module.

INZ{(constant)}

This keyword initializes the standalone field, data structure or data-structure subfield to the default value for its data type or, optionally, to the constant specified in parentheses. When used to initialize a data structure, the constant parameter is not allowed.

The constant specified must be consistent with the type being initialized. The constant can be a literal, named constant, figurative constant or built-in function. When initializing Date or Time data type fields or named constants with Date or Time values, the format of the literal must be consistent with the default format as derived from the Control specification, regardless of the actual format of the date or time field.

A data structure, data-structure subfield, or standalone field defined with the INZ keyword cannot be specified as a parameter on an *ENTRY PLIST.

Note: When the parameter is not specified:

- standalone fields and subfields of initialized data structures are initialized to their default initial values (for example, blanks for character, 0 for numeric).
- subfields of uninitialized data structures (INZ not specified on the definition specification for the data structure) are initialized to blanks (regardless of their data type).

This keyword is not valid in combination with BASED or IMPORT.

LIKE(RPG_name)

When the LIKE keyword is specified, the data being defined takes on the attributes and the length of the variable specified as the parameter. Standalone fields and data-structure subfields may be defined using this keyword. The data type entry (position 40) must be blank. This keyword is functionally equivalent to the *LIKE DEFINE operation code (see "*LIKE DEFINE" on page 342).

When using LIKE to define character fields, the number specified in the To/Length entry is the number of additional (or fewer) characters. When using LIKE to define graphic fields, the number specified in the To/Length entry is the number of additional (or fewer) graphic characters (1 graphic character = 2 bytes). When using LIKE to define numeric fields, the number specified in the To/Length entry is the number of additional (or fewer) digits.

When LIKE is used to define an array, the DIM keyword is still required to define the array dimensions. However, DIM(%elem(array)) can be used to define an array exactly like another array.

When using LIKE to define Date, Time, Timestamp, Basing Pointer, or Procedure Pointer fields, the To/Length entry (positions 33-39) must be blank.

The following are examples of defining data using the LIKE keyword.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
 *  Define a field like another with a length increase of 5 characters.
 *
D  Name           S             20
D  Long_name      S             +5    LIKE(Name)
 *
 *  Define a data structure subfield array with DIM(20) like another
 *  field and initialize each array element with the value *ALL'X'.
 *  Also, declare another subfield of type pointer immediately
 *  following the first subfield. Pointer is implicitly defined
 *  with a length of 16 bytes
 *
D  Struct         DS
D   Dim20                              LIKE(Name) DIM(20) INZ(*ALL'X')
D   Pointer                       *
 *
 *  Define a field which is based on the *LDA.  Take the length and type
 *  of the field from the field 'Name'.
 *
D  Lda_fld        S                    LIKE(Name) DTAARA(*LDA)
```

Figure 78. Defining fields LIKE other fields

NOOPT

No optimization is to be performed on the standalone field or data structure for which this keyword is specified. This insures that the content of the data item is the latest assigned value. This may be necessary for those fields whose values are used in exception handling.

Note: The optimizer may keep some values in registers and restore them only to storage at predefined points during normal program execution. Exception handling may break this *normal* execution sequence, and consequently program variables contained in registers may not be returned to their assigned storage locations. As a result, when those variables are used in exception handling, they may not contain the latest assigned value. The NOOPT keyword will ensure their currency.

All keywords allowed for standalone field definitions or data structure definitions are allowed with NOOPT.

OCCURS(numeric_constant)

The OCCURS keyword allows the specification of the number of occurrences of a multiple occurrence data structure.

The numeric_constant parameter must be a value greater than 0 with no decimal positions. It can be a numeric literal, a built-in function returning a numeric value, or a numeric constant. If the parameter is a named constant, it must be defined prior to this specification.

This keyword is not valid for a program status data structure, a file information data structure, or a data area data structure.

If a multiple occurrence data structure contains pointer subfields, the distance between occurrences must be an exact multiple of 16 because of system storage restrictions for pointers. This means that the distance between occurrences may be greater than the length of each occurrence.

The following is an example showing the storage allocation of a multiple occurrence data structure with pointer subfields.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D DS1             DS                  OCCURS(2)
D   POINTER                     16*
D   FLD5                         5
D DS2             DS                  OCCURS(2)
D   CHAR16                      16
D   CHR5                         5
```

Allocation of fields in storage. The occurrences of DS1 are 32 bytes apart, while the occurrences of DS2 are 21 bytes apart.

DS1 OCCURRENCE 1			DS1 OCCURRENCE 2		
POINTER	FLD5	(fill)	POINTER	FLD5	(fill)

DS2 OCCURRENCE 1		DS2 OCCURRENCE 2	
CHAR16	CHR5	CHAR16	CHR5

Figure 79. Storage Allocation of Multiple Occurrence Data Structure with Pointer Subfields

OVERLAY(name{:pos})

This keyword is allowed only for data structure subfields. The subfield overlays the storage of the subfield specified by the parameter name at the position specified by the parameter pos. If pos is not specified, it defaults to 1.

The following rules apply to keyword OVERLAY:

1. The name parameter must be the name of a subfield defined previously in the current data structure.
2. The pos parameter (if specified) must be a value greater than 0 with no decimal positions. It can be a numeric literal, a built-in function returning a numeric value, or a numeric constant. If pos is a named constant, it must be defined prior to this specification.
3. The OVERLAY keyword is not allowed when the From-Position entry is not blank.
4. The subfield being defined must be contained completely within the subfield specified by the name parameter.
5. If the subfield specified as the first parameter for the OVERLAY keyword is an array, the OVERLAY keyword applies to each element of the array. That is, the field being defined is defined as an array with the same number of elements. The first element of this array overlays the first element of the overlayed array, the 2nd element of this array overlays the 2nd element of the overlayed array, etcetera. No array keywords may be specified for the subfield with the OVERLAY keyword in this situation. (Refer to Figure 80) See also "SORTA (Sort an Array)" on page 461.

   If the subfield name, specified as the first parameter for the OVERLAY keyword, is an array and its element length is longer than the length of the subfield being defined, the array elements of the subfield being defined are not stored contiguously. Such an array is not allowed as the Result Field of a PARM operation or in Factor 2 or the Result Field of a MOVEA operation.

Note: The pos parameter is in units of bytes, regardless of the types of the subfields.

Examples

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D DataStruct      DS
D   A                           10    DIM(5)
D     B                          5    OVERLAY(A)
D     C                          5    OVERLAY(A:6)
```

Allocation of fields in storage:

A(1)		A(2)		A(3)		A(4)		A(5)	
B(1)	C(1)	B(2)	C(2)	B(3)	C(3)	B(4)	C(4)	B(5)	C(5)

Figure 80. Storage Allocation of Subfields with Keywords DIM and OVERLAY

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D DataStruct      DS
D  A                          5
D    B                        1    OVERLAY(A) DIM(4)
```

Allocation of fields in storage:

A				
B(1)	B(2)	B(3)	B(4)	

Figure 81. Storage Allocation of Subfields with Keywords DIM and OVERLAY

PACKEVEN

The PACKEVEN keyword indicates that the packed field or array has an even number of digits. The keyword is only valid for packed program-described data-structure subfields defined using FROM/TO positions. For a field or array element of length N, if the PACKEVEN keyword is not specified, the number of digits is 2N - 1; if the PACKEVEN keyword is specified, the number of digits is 2(N-1).

PERRCD(numeric_constant)

The PERRCD keyword allows you to specify the number of elements per record for a compile-time or a prerun-time array or table. If the PERRCD keyword is not specified, the number of elements per record defaults to one (1).

The numeric_constant parameter must be a value greater than 0 with no decimal positions. It can be a numeric literal, a built-in function returning a numeric value, or a numeric constant. If the parameter is a named constant, it must be defined prior to this specification.

The PERRCD keyword is valid only when the keyword FROMFILE, TOFILE, or CTDATA is specified.

PREFIX(prefix_string)

The PREFIX keyword allows the specification of a string which is to be prefixed to the subfield names of the externally-described data structure being defined.

The following rules apply:

- Subfields that are explicitly renamed using the EXTFLD keyword are not affected by this keyword.
- The total length of a name after applying the prefix must not exceed the maximum length of an RPG field name.

PROCPTR

The PROCPTR keyword defines an item as a procedure pointer. The Internal Data-Type field (position 40) must contain a *.

TIMFMT(format{separator})

The TIMFMT keyword allows the specification of a time format, and optionally the time separator, for a standalone field or data-structure subfield, of type Time. This keyword will be automatically generated for an externally-described data-structure subfield of type Time.

See Table 19 on page 176 for valid formats and separators.

The hierarchy used when determining the internal format and separator for a time array or field is:

1. From the TIMFMT keyword specified on the definition specification
2. From the TIMFMT keyword specified in the control specification
3. *ISO

TOFILE(file_name)

The TOFILE keyword allows the specification of a target file to which a prerun-time or compile-time array or table is to be written.

If an array or table is to be written, specify the file name of the output or combined file as the keyword parameter. This file must also be defined in the file description specifications. An array or table can be written to only one output device.

If an array or table is assigned to an output file, it is automatically written if the LR indicator is on at program termination. The array or table is written after all other records are written to the file.

If an array or table is to be written to the same file from which it was read, the same file name that was specified as the FROMFILE parameter must be specified as the TOFILE parameter. This file must be defined as a combined file (C in position 17 on the file-description specification).

Summary According to Definition Specification Type

See Table 23 for a listing of required and allowed entries for each definition-specification type.

See Table 24 on page 217 for a listing of the keywords allowed for each definition-specification type.

Table 23. Required/Allowed Entries for each Definition Specification Type

Type	Pos. 7-21 Name	Pos. 22 External	Pos. 23 DS Type	Pos. 24-25 Defn. Type	Pos. 26-32 From	Pos. 33-39 To / Length	Pos. 40 Data-type	Pos. 41-42 Decimal Pos.	Pos. 44-80 Key-words
Data Structure	A	A	A	X		A			A
Data Structure Subfield	A				A	A	A	A	A
External Subfield	A	X							A
Standalone Field	X			X		A	A	A	A
Named Constant	X			X					X

Note:
X = An entry in these positions is required.
A = An entry in these positions is allowed.

Table 24. Keywords Supported for Each Definition Specification Type

Keyword	Data Structure	Data Structure Subfield	External Subfield	Standalone Field	Named Constant
ALT		X	X	X	
ASCEND		X	X	X	
BASED	X			X	
CONST					X
CTDATA		X	X	X	
DATFMT		X		X	
DESCEND		X	X	X	
DIM		X	X	X	
DTAARA	X	X		X	
EXPORT	X			X	
EXTFLD			X		
EXTFMT		X	X	X	
EXTNAME	X				
FROMFILE		X	X	X	
IMPORT	X			X	
INZ	X	X	X	X	
LIKE		X		X	
NOOPT	X			X	
OCCURS	X				
OVERLAY		X			
PACKEVEN		X			
PERRCD		X	X	X	
PREFIX	X				
PROCPTR		X		X	
TIMFMT		X		X	
TOFILE		X	X	X	

Chapter 17. Input Specifications

For a program described input file, input specifications describe the types of records within the file, the sequence of the types of records, the fields within a record, the data within the field, indicators based on the contents of the fields, control fields, fields used for matching records, and fields used for sequence checking. For an externally described file, input specifications are optional and can be used to add RPG IV functions to the external description.

Detailed information for the input specifications is given in:

- Entries for program described files
- Entries for externally described files

Input Specification Statement

The general layout for the Input specification is as follows:

- the input specification type (I) is entered in position 6
- the non-commentary part of the specification extends from position 7 to position 80
- the comments section of the specification extends from position 81 to position 100

Program Described

For program described files, entries on input specifications are divided into the following categories:

- Record identification entries (positions 7 through 46), which describe the input record and its relationship to other records in the file.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..................................Comments++++++++++++
I.........And..RiPos1+NCCPos2+NCCPos3+NCC..................................Comments++++++++++++
```

Figure 82. Program Described Record Layout

- Field description entries (positions 31 through 74), which describe the fields in the records. Each field is described on a separate line, below its corresponding record identification entry.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
I........................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr......Comments++++++++++++
```

Figure 83. Program Described Field Layout

Externally Described

For externally described files, entries on input specifications are divided into the following categories:

- Record identification entries (positions 7 through 16, and 21 through 22), which identify the record (the externally described record format) to which RPG IV functions are to be added.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
IRcdname+++....Ri..........................................................Comments++++++++++++
```

Figure 84. Externally Described Record Layout

- Field description entries (positions 21 through 30, 49 through 66, and 69 through 74), which describe the RPG IV functions to be added to the fields in the record. Field description entries are written on the lines following the corresponding record identification entries.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
I..............Ext-field+..................Field+++++++++L1M1..PlMnZr......Comments++++++++++++
```

Figure 85. Program Described Field Layout

Program Described Files

Position 6 (Form Type)

An I must appear in position 6 to identify this line as an input specification statement.

Record Identification Entries

Record identification entries (positions 7 through 46) for a program described file describe the input record and its relationship to other records in the file.

Positions 7-16 (File Name)

Entry	**Explanation**
A valid file name	Same file name that appears on the file description specifications for the input file.

Enter the name of the file to be described in these positions. This name must be the same name defined for the file on the file description specifications. This file must be an input file, an update file, or a combined file. The file name must be entered on the first record identification line for each file and can be entered on subsequent record identification lines for that file. All entries describing one input file must appear together; they cannot be mixed with entries for other files.

Positions 16-18 (Logical Relationship)

Entry	**Explanation**
AND	More than three identification codes are used.
OR	Two or more record types have common fields.

An unlimited number of AND/OR lines can be used. For more information see “AND Relationship” on page 226 and “OR Relationship” on page 226.

Positions 17-18 (Sequence)

Entry	**Explanation**
Any two alphabetic characters	The program does not check for special sequence.
Any two-digit number	The program checks for special sequence within the group.

The numeric sequence entry combined with the number (position 19) and option (position 20) entries causes the program to check the sequence of input records within a file. If the sequence is not correct, control passes to the RPG IV exception/error handling routine. If AND or OR lines are specified, the sequence entry is made on the main record line of the group, not on the AND or OR lines.

Alphabetic and numeric entries can be made for different records (different record identification lines) in the same file, but records with alphabetic entries must be specified before records with numeric entries.

Alphabetic Entries

Enter any two alphabetic characters in these positions when no sequence checking is to be done. It is common programming practice to specify these codes in a sequence that aids in program documentation. However, it is not necessary to use unique alphabetic entries.

Numeric Entries

Enter a unique numeric code in positions 17 and 18 if one record type must be read before another record type in a file. Numeric entries must be in ascending order, starting with 01, but need not be consecutive. When a numeric entry is used, the appropriate entries must be made in positions 19 and 20.

To specify sequence checking, each record type must have a record identification code, and the record types must be numbered in the order in which they should appear. This order is checked as the records are read. If a record type is out of sequence, control passes to the RPG IV exception/error handling routine.

Sequence numbers ensure only that all records of each record type precede the records of higher sequence-numbered record types. The sequence numbers do not ensure that records within a record type are in any certain order. Sequence numbers are unrelated to control levels and do not provide for checking data in fields of a record for a special sequence. Use positions 65 and 66 (matching fields) to indicate that data in fields of a record should be checked for a special sequence.

Position 19 (Number)

Entry	Explanation
Blank	The program does not check record types for a special sequence (positions 17 and 18 have alphabetic entries).
1	Only one record of this type can be present in the sequenced group.
N	One or more records of this type can be present in the sequenced group.

This entry must be used when a numeric entry is made in positions 17 and 18. If an alphabetic entry is made in positions 17 and 18, this entry must be blank.

Position 20 (Option)

Entry	Explanation
Blank	The record type must be present if sequence checking is specified.
O	The record type is optional (that is, it may or may not be present) if sequence checking is specified.

This entry must be blank if positions 17 and 18 contain an alphabetic entry.

Sequence checking of record types has no meaning when all record types within a file are specified as optional (alphabetic entry in positions 17 and 18 or O entry in position 20).

Positions 21-22 (Record Identifying Indicator, or **)

Entry	Explanation
Blank	No indicator is used.
01-99	General indicator.
L1-L9 or LR	Control level indicator used for a record identifying indicator.
H1-H9	Halt indicator.
U1-U8	External indicator.
RT	Return indicator.
**	Lookahead field (not an indicator). Lookahead can be used only with a primary or secondary file.

The indicators specified in these positions are used in conjunction with the record identification codes (positions 23 through 46).

Indicators

Positions 21 and 22 associate an indicator with the record type defined on this line. The normal entry is one of the indicators 01 to 99; however, the control level indicators L1 through L9 and LR can be used to cause certain total steps to be processed. If a control level indicator is specified, lower control level indicators are not set on. The halt indicators H1 through H9 can be used to stop processing. The return indicator (RT) is used to return to the calling program.

When a record is selected for processing and satisfies the conditions indicated by the record identification codes, the appropriate record identifying indicator is set on. This indicator can be used to condition calculation and output operations. Record identifying indicators can be set on or set off by the programmer. However, at the end of the cycle, all record identifying indicators are set off before another record is selected.

Lookahead Fields

The entry of ** is used for the lookahead function. This function lets you look at information in the next record in a file. You can look not only at the file currently selected for processing but also at other files present but not selected during this cycle.

Field description lines must contain From and To entries in the record, a field name, and decimal positions if the field is numeric. Note that a lookahead field may not be specified as a field name on input specifications or as a data structure name on definition specifications or as a Result Field on Calculation Specifications.

Positions 17 and 18 must contain an alphabetic entry. The lookahead fields are defined in positions 49 through 62 of the lines following the line containing ** in positions 21 and 22. Positions 63 through 80 must be blank.

Any or all of the fields in a record can be defined as lookahead fields. This definition applies to all records in the file, regardless of their type. If a field is used both as a lookahead field and as a normal input field, it must be defined twice with different names.

The lookahead function can be specified only for primary and secondary files and can be specified only once for a file. It cannot be used for full procedural files (identified by an F in position 18 of the file description specifications), or with AND or OR lines.

When a record is being processed from a combined file or an update file, the data in the lookahead field is the same as the data in the record being processed, not the data in the next record.

The lookahead function causes information in the file information data structure to be updated with data pertaining to the lookahead record, not to the current primary record.

If an array element is specified as a lookahead field, the entire array is classified as a lookahead field.

Lookahead fields are filled with nines when all records in the file have been processed so that the end of the file can be recognized.

Positions 23-46 (Record Identification Codes)

Entries in positions 23 through 46 identify each record type in the input file. One to three identification codes can be entered on each specification line. More than three record identification codes can be specified on additional lines with the AND/OR relationship. If the file contains only one record type, the identification codes can be left blank; however, a record identifying indicator entry (positions 21 and 22) and a sequence entry (positions 17 and 18) must be made.

Note: Record identification codes are not applicable for graphic data type processing: record identification is done on single byte positions only.

Three sets of entries can be made in positions 23 through 46: 23 through 30, 31 through 38, and 39 through 46. Each set is divided into four groups: position, not, code part, and character.

The following table shows which categories use which positions in each set.

Category	23-30	31-38	39-46
Position	23-27	31-35	39-43
Not	28	36	44
Code Part	29	37	45
Character	30	38	46

Entries in these sets need not be in sequence. For example, an entry can be made in positions 31 through 38 without requiring an entry in positions 23 through 30. Entries for record identification codes are not necessary if input records within a file are of the same type. An input specification containing no record identification code defines the last record type for the file, thus allowing the handling of any record types that are undefined. If no record identification codes are satisfied, control passes to the RPG IV exception/error handling routine.

Positions 23-27, 31-35, and 39-43 (Position)

Entry	Explanation
Blank	No record identification code is present.
1-32766	The position that contains the record identification code in the record.

In these positions enter the position that contains the record identification code in each record. The position containing the code must be within the record length specified for the file. This entry must be right-adjusted, but leading zeros can be omitted.

Positions 28, 36, and 44 (Not)

Entry	Explanation
Blank	Record identification code must be present.
N	Record identification code must not be present.

Enter an N in this position if the code described must not be present in the specified record position.

Positions 29, 37, and 45 (Code Part)

Entry	Explanation
C	Entire character
Z	Zone portion of character
D	Digit portion of character.

This entry specifies what part of the character in the record identification code is to be tested.

Character (C): The C entry indicates that the complete structure (zone and digit) of the character is to be tested.

Zone (Z): The Z entry indicates that the zone portion of the character is to be tested. The zone entry causes the four high-order bits of the character entry to be compared with the zone portion of the character in the record position specified in the position entry. The following three special cases are exceptions:

- The hexadecimal representation of an & (ampersand) is 50. However, when an ampersand is coded in the character entry, it is treated as if its hexadecimal representation were C0, that is, as if it had the same zone as A through I. An ampersand in the input data satisfies two zone checks: one for a hexadecimal 5 zone, the other for a hexadecimal C zone.
- The hexadecimal representation of a - (minus sign) is 60. However, when a minus sign is coded in the character entry, it is treated as if its hexadecimal representation were D0, that is, as if it had the same zone as J through R. A minus sign in the input data satisfies two zone checks: one for a hexadecimal 6 zone, the other for a hexadecimal D zone.
- The hexadecimal representation of a blank is 40. However, when a blank is coded in the character entry, it is treated as if its hexadecimal representation were F0, that is, as if it had the same zone as 0 through 9. A blank in the input data satisfies two zone checks: one for a hexadecimal 4 zone, the other for a hexadecimal F zone.

Digit (D): The D entry indicates that the digit portion of the character is to be tested. The four low-order bits of the character are compared with the character specified by the position entry.

Positions 30, 38, and 46 (Character)

In this position enter the identifying character that is to be compared with the character in the position specified in the input record.

The check for record type always starts with the first record type specified. If data in a record satisfies more than one set of record identification codes, the first record type satisfied determines the record types.

When more than one record type is specified for a file, the record identification codes should be coded so that each input record has a unique set of identification codes.

AND Relationship

The AND relationship is used when more than three record identification codes identify a record.

To use the AND relationship, enter at least one record identification code on the first line and enter the remaining record identification codes on the following lines with AND coded in positions 16 through 18 for each additional line used. Positions 7 through 15, 19 through 20, and 46 through 80 of each line with AND in positions 16 through 18 must be blank. Sequence, and record-identifying-indicator entries are made in the first line of the group and cannot be specified in the additional lines.

An unlimited number of AND/OR lines can be used on the input specifications.

OR Relationship

The OR relationship is used when two or more record types have common fields.

To use the OR relationship, enter OR in positions 16 and 17. Positions 7 through 15, 18 through 20, and 46 through 80 must be blank. A record identifying indicator can be entered in positions 21 and 22. If the indicator entry is made and the record identification codes on the OR line are satisfied, the indicator specified in positions 21 and 22 on that line is set on. If no indicator entry is made, the indicator on the preceding line is set on.

An unlimited number of AND/OR lines can be used on the input specifications.

Field Description Entries

The field description entries (positions 31 through 74) must follow the record identification entries (positions 7 through 46) for each file.

Position 6 (Form Type)

An I must appear in position 6 to identify this line as an input specification statement.

Positions 7-30 (Reserved)

Positions 7-30 must be blank.

Positions 31-34 (Date/Time External Format)

Positions 31-34 specify the external format for a date or time field. See Table 10 on page 106 and Table 12 on page 107 for valid date and time formats.

The hierarchy used when determining the external date/time format and separator for date and time fields is:

1. The date format and separator specified in positions 31-35
2. From the DATFMT/TIMFMT keyword specified for the current file
3. From the DATFMT/TIMFMT keyword specified in the control specification
4. *ISO

Date and time fields will be converted from the external date/time format determined above to the internal format of the date/time field.

Position 35 (Date/Time Separator)

Position 35 specifies a separator character to be used for date/time fields. The & (ampersand) can be used to specify a blank separator. See Table 10 on page 106 and Table 12 on page 107 for date and time formats and their default separators.

For an entry to be made in this field, an entry must also be made in positions 31-34 (date/time external format).

Position 36 (Data Format)

Entry	Explanation
Blank	The input field is in zoned decimal format or is a character field.
P	The input field is in packed decimal format.
B	The input field is in binary format.
L	The numeric input field has a preceding (left) plus or minus sign and is in zoned decimal format.
R	The number input field has a following (right) plus or minus sign and is in zoned decimal format.
D	Date field
T	Time field
Z	Timestamp field
A	Character field
S	Zoned decimal field
G	Graphic field

The entry in position 36 specifies the format of the data in the records in the file. This entry has no effect on the format used for internal processing of the input field in the program.

See Chapter 9 in the *ILE RPG/400 Programmer's Guide* for information on internal field formats.

Positions 37-46 (Field Location)

Entry	Explanation
Two 1- to 5-digit numbers	Beginning of a field (from) and end of a field (to).

This entry describes the location and size of each field in the input record. Positions 37 through 41 specify the location of the field's beginning position; positions 42 through 46 specify the location of the field's end position. To define a single-position field, enter the same number in positions 37 through 41 and in positions 42 through 46. Numeric entries must be right-adjusted; leading zeros can be omitted.

The maximum number of positions in the input record for each type of field is as follows:

Number of Positions	Type of Field
30	Zoned decimal numeric (30 digits)
16	Packed numeric (30 digits)
4	Binary (9 digits)
32767	Character (32767 characters)
32766	Graphic (16383 graphic characters)
31	Numeric with leading or trailing sign (30 digits)
32767	Data structure.

For arrays, enter the beginning position of the array in positions 37 through 41 and the ending position in positions 42 through 46. The array length must be an integral multiple of the length of an element. The From-To position does not have to account for all the elements in the array. The placement of data into the array starts with the first element.

Positions 47-48 (Decimal Positions)

Entry	Explanation
Blank	Character, graphic, date, time, or timestamp field
0-30	Number of decimal positions in numeric field.

This entry, used with the data format entry in position 36, describes the format of the field. An entry in this field identifies the input field as numeric, that is, if the field is numeric, an entry must be made. The number of decimal positions specified for a numeric field cannot exceed the length of the field.

Positions 49-62 (Field Name)

Entry	Explanation
Symbolic name	Field name, data structure name, data structure subfield name, array name, array element, PAGE, PAGE1-PAGE7, *IN, *INxx, or *IN(xx).

These positions name the fields of an input record that are used in an RPG IV program. This name must follow the rules for symbolic names.

To refer to an entire array on the input specifications, enter the array name in positions 49 through 62. If an array name is entered in positions 49 through 62, control level (positions 63-64), matching fields (positions 65 and 66), and field indicators (positions 67 through 68) must be blank.

To refer to an element of an array, specify the array name, followed by an index enclosed within parentheses. The index is either a numeric field with zero decimal positions or the actual number of the array element to be used. The value of the index can vary from 1 to n, where n is the number of elements within the array.

Positions 63-64 (Control Level)

Entry	Explanation
Blank	This field is not a control field. Control level indicators cannot be used with full procedural files.
L1-L9	This field is a control field.

Positions 63 and 64 indicate the fields that are used as control fields. A change in the contents of a control field causes all operations conditioned by that control level indicator and by all lower level indicators to be processed.

A split control field is a control field that is made up of more than one field, each having the same control level indicator. The first field specified with that control level indicator is placed in the high-order position of the split control field, and the last field specified with the same control level indicator is placed in the low-order position of the split control field.

Positions 65-66 (Matching Fields)

Entry	Explanation
Blank	This field is not a match field.
M1-M9	This field is a match field.

This entry is used to match the records of one file with those of another or to sequence check match fields within one file. Match fields can be specified only for fields in primary and secondary files.

Match fields within a record are designated by an M1 through M9 code entered in positions 65 and 66 of the appropriate field description specification line. A maximum of nine match fields can be specified.

The match field codes M1 through M9 can be assigned in any sequence. For example, M3 can be defined on the line before M1, or M1 need not be defined at all.

When more than one match field code is used for a record, all fields can be considered as one large field. M1 or the lowest code used is the rightmost or low-order position of the field. M9 or the highest code used is the leftmost or high-order position of the field.

The ALTSEQ (alternate collating sequence) and FTRANS (file translation) keywords on the control specification can be used to alter the collating sequence for match fields.

If match fields are specified for only a single sequential file (input, update, or combined), match fields within the file are sequence checked. The MR indicator is not set on and cannot be used in the program. An out-of-sequence record causes the RPG IV exception/error handling routine to be given control.

In addition to sequence checking, match fields are used to match records from the primary file with those from secondary files.

Positions 67-68 (Field Record Relation)

Entry	Explanation
Blank	The field is common to all record types.
01-99	General indicators.
L1-L9	Control level indicators.
MR	Matching record indicator.
U1-U8	External indicators.
H1-H9	Halt indicators.
RT	Return indicator.

Field record relation indicators are used to associate fields within a particular record type when that record type is one of several in an OR relationship. This entry reduces the number of lines that must be written.

The field described on a line is extracted from the record by the RPG IV program only when the indicator coded in positions 67 and 68 is on or when positions 67 and 68 are blank. When positions 67 and 68 are blank, the field is common to all record types defined by the OR relationship.

Field record relation indicators can be used with control level fields (positions 63 and 64) and matching fields (positions 65 and 66).

Positions 69-74 (Field Indicators)

Entry	Explanation
Blank	No indicator specified
01-99	General indicators
H1-H9	Halt indicator
U1-U8	External indicators
RT	Return indicator.

Entries in positions 69 through 74 test the status of a field or of an array element as it is read into the program. Field indicators are specified on the same line as the field to be tested. Depending on the status of the field (plus, minus, zero, or blank), the appropriate indicator is set on and can be used to condition later specifications. The same indicator can be specified in two positions, but it should not be used for all three positions. Field indicators cannot be used with arrays that are not indexed or look-ahead fields.

Positions 69 and 70 (plus) and positions 71 and 72 (minus) are valid for numeric fields only. Positions 73 and 74 can be used to test a numeric field for zeros or a character or graphic field for blanks.

The field indicators are set on if the field or array element meets the condition specified when the record is read. Each field indicator is related to only one record type; therefore, the indicators are not reset (on or off) until the related record is read again or until the indicator is defined in some other specification.

Externally Described Files

Position 6 (Form Type)

An I must appear in position 6 to identify this line as an input specifications statement.

Record Identification Entries

When the description of an externally described file is retrieved by the compiler, the record definitions are also retrieved. To refer to the record definitions, specify the record format name in the input, calculation, and output specifications of the program. Input specifications for an externally described file are required if:

- Record identifying indicators are to be specified.
- A field within a record is to be renamed for the program.
- Control level or matching field indicators are to be used.
- Field indicators are to be used.

The field description specifications must immediately follow the record identification specification for an externally described file.

A record line for an externally described file defines the beginning of the override specifications for the record. All specifications following the record line are part of the record override until another record format name or file name is found in positions 7 through 16 of the input specifications. All record lines that pertain to an externally described file must appear together; they cannot be mixed with entries for other files.

Positions 7-16 (Record Name)

Enter one of the following:

- The external name of the record format. (The file name cannot be used for an externally described file.)
- The RPG IV name specified by the RENAME keyword on the file description specifications if the external record format was renamed. A record format name can appear only once in positions 7 through 16 of the input specifications for a program.

Positions 17-20 (Reserved)

Positions 17 through 20 must be blank.

Positions 21-22 (Record Identifying Indicator)

The specification of record identifying indicators in these positions is optional but, if present, follows the rules as described under "Program Described Files" on page 221 earlier in this chapter, except for look-ahead specifications, which are not allowed for an externally described file.

Positions 23-80 (Reserved)

Positions 23-80 must be blank.

Field Description Entries

The field description specifications for an externally described file can be used to rename a field within a record for a program or to specify control level, field indicator, and match field functions. The field definitions (attributes) are retrieved from the externally described file and cannot be changed by the program. If the attributes of a field are not valid to an RPG IV program (such as numeric length greater than 30 digits), the field cannot be used. Diagnostic checking is done on fields contained in an external record format in the same way as for source statements.

Positions 7-20 (Reserved)

Positions 7 through 20 must be blank.

Positions 21-30 (External Field Name)

If a field within a record in an externally described file is to be renamed, enter the external name of the field in these positions. A field may have to be renamed because the name is the same as a field name specified in the program and two different names are required.

Positions 31-48 (Reserved)

Positions 31 through 48 must be blank.

Positions 49-62 (Field Name)

The field name entry is made only when it is required for the RPG IV function (such as control levels) added to the external description. The field name entry contains one of the following:

- The name of the field as defined in the external record description (if 10 characters or less).
- The name specified to be used in the program that replaced the external name specified in positions 21 through 30.

The field name must follow the rules for using symbolic names.

Positions 63-64 (Control Level)

This entry indicates whether the field is to be used as a control field in the program.

Entry	Explanation
Blank	This field is not a control field.
L1-L9	This field is a control field.

Note: For externally described files, split control fields are combined in the order in which the fields are specified on the data description specifications (DDS), not in the order in which the fields are specified on the input specifications.

Positions 65-66 (Matching Fields)

This entry indicates whether the field is to be used as a match field.

Entry	Explanation
Blank	This field is not a match field.
M1-M9	This field is a match field.

See "Positions 65-66 (Matching Fields)" on page 229 for more information on match fields.

Positions 67-68 (Reserved)

Positions 67 and 68 must be blank.

Positions 69-74 (Field Indicators)

Entry	Explanation
Blank	No indicator specified
01-99	General indicators
H1-H9	Halt indicators
U1-U8	External indicators
RT	Return indicator.

See "Positions 69-74 (Field Indicators)" on page 230 for more information.

Positions 75-80 (Reserved)

Positions 75 through 80 must be blank.

Chapter 18. Calculation Specifications

Calculation specifications indicate the operations done on the data in a program.

Calculation entries must be grouped in the following order:

- Detail calculations
- Total calculations
- Subroutines.

Calculations within the groups must be specified in the order in which they are to be done.

The calculation specifications are entered on the RPG IV Calculation Specifications. See Chapter 22, “Operation Codes” on page 281 for details on how these positions must be specified for individual operation codes.

The calculation specification can also be used to enter SQL statements into an RPG IV program. See ILE RPG/400 Programmer’s Guide and DB2/400 SQL Reference for more information.

Calculation Specification Statement

The general layout for the calculation specification is as follows:

- the calculation specification type (C) is entered in position 6
- the non-commentary part of the specification extends from position 7 to position 80. These positions are divided into three parts that specify the following:
  - *When calculations are done:*
    The control level indicator and the conditioning indicators specified in positions 7 through 11 determine when and under what conditions the calculations are to be done.
  - *What kind of calculations are done:*
    The entries specified in positions 12 through 70 (12 through 80 for operations that use extended factor 2, see "Calculation Extended Factor 2 Specification Statement" on page 241 and "Operations Using Expressions" on page 294) specify the kind of calculations done, the data (such as fields or files) upon which the operation is done, and the field that contains the results of the calculation.
  - *What tests are done on the results of the operation:*
    Indicators specified in positions 71 through 76 are used to test the results of the calculations and can condition subsequent calculations or output operations. The resulting indicator positions have various uses, depending on the operation code. For the uses of these positions, see the individual operation codes in Chapter 22, "Operation Codes" on page 281.
- the comments section of the specification extends from position 81 to position 100

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments++++++++++++
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++++Comments++++++++++++
```

Figure 86. Calculation Specification Layout

Calculation-Specification Extended Factor-2 Continuation Line

The Extended Factor-2 field can be continued on subsequent lines as follows:

- position 6 of the continuation line must contain a C
- positions 7 to 35 of the continuation line must be blank
- the specification continues on or past position 36

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
C.............................Extended-factor2-continuation++++++++++++++++Comments++++++++++++
```

Figure 87. Calculation-Specification Extended Factor-2 Continuation Line

Position 6 (Form Type)

A C must appear in position 6 to identify this line as a calculation specification statement.

Positions 7-8 (Control Level)

Entry	Explanation
Blank	The calculation operation is done at detail calculation time for each program cycle if the indicators in positions 9 through 11 allow it; or the calculation is part of a subroutine. Blank is also used for declarative operation codes.
L0	The calculation operation is done at total calculation time for each program cycle.
L1-L9	The calculation operation is done at total calculation time when the control level indicator is on. The indicator is set on either through a level break or as the result of an input or calculation operation.
LR	The calculation operation is done after the last record has been processed or after the LR indicator has been set on.
SR	The calculation operation is part of an RPG IV subroutine. A blank entry is also valid for calculations that are part of a subroutine.
AN, OR	Indicators on more than one line condition the calculation.

Control Level Indicators

The L0 entry is used in positions 7 and 8 to indicate that the calculation is always done during total calculation time.

If indicators L1 through L9 are specified in positions 7 and 8, the calculation is processed at total calculation time only when the specified indicator is on. Remember that, if L1 through L9 are set on by a control break, all lower level indicators are also set on. If positions 7 and 8 are blank, the calculation is done at detail time calculation, is a statement within a subroutine, is a declarative statement, or is a continuation line.

The following operations can be specified within total calculations with positions 7 and 8 blank: PLIST, PARM, KLIST, KFLD, TAG, DEFINE, and ELSE. (Conditioning indicators in positions 9 through 11 are not allowed with these operations.) In addition, all the preceding operations except TAG and ELSE can be specified anywhere within the calculations, even between an ENDSR operation of one subroutine and the BEGSR operation of the next subroutine or after the ENDSR operation for the last subroutine.

Last Record Indicator

The LR Indicator, if specified in positions 7 and 8, causes the calculation to be done during the last total calculation time.

If there is a primary file but no secondary files in the program, the LR indicator is set on after the last input record has been read, the calculations specified for the record have been done, and the detail output for the last record read has been completed.

If there is more than one input file (primary and secondary), the RPG IV programmer determines which files are to be checked for end-of-file by entering an E in position 19 of the file description specifications. LR is set on when all files with an end-of-file specification have been completely read, when detail output for the last record in these files has been completed, and after all matching secondary records have been processed.

When the LR indicator is set on after the last input record has been read, all control indicators L1 through L9 defined to the program are also set on.

Subroutine Identifier

An SR entry in positions 7 and 8 may optionally be used for operations within subroutines as a documentation aid. Subroutine lines must appear after the total calculation specifications. The operation codes BEGSR and ENDSR serve as delimiters for a subroutine.

AND/OR Lines Identifier

Positions 7 and 8 can contain AN or OR to define additional indicators (positions 9 through 11) for a calculation.

The entry in positions 7 and 8 of the line immediately preceding an AND/OR line or a group of AND/OR lines determines when the calculation is to be processed. The entry in positions 7 and 8 on the first line of a group applies to all AND/OR lines in the group. A control level indicator (L1 through L9, L0, or LR) is entered for total calculations, an SR or blanks for subroutines, and a blank for detail calculations.

Positions 9-11 (Indicators)

Entry	**Explanation**
Blank	The operation is processed on every record
01-99	General indicators.
KA-KN, KP-KY	Function key indicators.
L1-L9	Control level indicators.
LR	Last record indicator.
MR	Matching record indicator.
H1-H9	Halt indicators.
RT	Return indicator.
U1-U8	External indicators.
OA-OG, OV	Overflow indicator.

Positions 10 and 11 contain an indicator that is tested to determine if a particular calculation is to be processed. A blank in position 9 designates that the indicator must be on for a calculation to be done. An N in positions 9 designates that the associated indicator must be off for a calculation to be done.

Positions 12-25 (Factor 1)

Factor 1 names a field or gives actual data (literals) on which an operation is done, or contains a RPG IV special word (for example, *LOCK) which provides extra information on how an operation is to be done. The entry must begin in position 12. The entries that are valid for factor 1 depend on the operation code specified in positions 26 through 35. For the specific entries for factor 1 for a particular operation code, see Chapter 22, “Operation Codes” on page 281. With some operation codes, two operands may be specified separated by a colon.

Positions 26-35 (Operation and Extender)

Positions 26 through 35 specify the kind of operation to be done using factor 1, factor 2, and the result field entries. The operation code must begin in position 26. For further information on the operation codes, see Chapter 22, "Operation Codes" on page 281.

Operation Extender

Entry	Explanation
Blank	No operation extension supplied
H	Half adjust (round) result of numeric operation
N	Record is read but not locked
P	Pad the result field with blanks
D	Operational descriptors or Date field
T	Time field
Z	Timestamp field

The operation extenders provide additional attributes to the operations that they accompany. Operation extenders are specified in positions 26-35 of calculation specifications. They must begin to the right of the operation code and be contained within parentheses -- for example, MULT(H). Blanks can be used for readability -- for example, READE (N) or

CAT (P)

An H indicates whether the contents of the result field are to be half adjusted (rounded). Resulting indicators are set according to the value of the result field after half-adjusting has been done.

An N in a READ, READE, READP, READPE, or CHAIN operation on an update disk file indicates that a record is to be read, but not locked. If no value is specified, the default action of locking occurs.

A P indicates that, the result field is padded after executing the instruction if the result field is longer than the result of the operation.

A D when specified on the CALLB operation code indicates that operational descriptors are included.

The D, T, and Z extenders can be used with the TEST operation code to indicate a date, time, or timestamp field.

Positions 36-49 (Factor 2)

Factor 2 names a field, record format or file, or gives actual data on which an operation is to be done, or contains a special word (for example, *ALL) which gives extra information about the operation to be done. The entry must begin in position 36. The entries that are valid for factor 2 depend on the operation code specified in positions 26 through 35. With some operation codes, two operands may be specified separated by a colon. For the specific entries for factor 2 for a particular operation code, see Chapter 22, "Operation Codes" on page 281.

Positions 50-63 (Result Field)

The result field names the field or record format that contains the result of the calculation operation specified in positions 26 through 35. The field specified must be modifiable. For example, it cannot be a lookahead field or a user date field. With some operation codes, two operands may be specified separated by a colon. See Chapter 22, "Operation Codes" on page 281 for the result field rules for individual operation codes.

Positions 64-68 (Field Length)

Entry	Explanation
1-30	Numeric field length.
1-32767	Character field length.
Blank	The result field is defined elsewhere or a field cannot be defined using this operation code

Positions 64 through 68 specify the length of the result field. This entry is optional, but can be used to define a numeric or character field not defined elsewhere in the program. These definitions of the field entries are allowed if the result field contains a field name. Other data types must be defined on the definition specification or on the calculation specification using the *LIKE DEFINE operation.

The entry specifies the number of positions to be reserved for the result field. The entry must be right-adjusted. The unpacked length (number of digits) must be specified for numeric fields.

If the result field is defined elsewhere in the program, no entry is required for the length. However, if the length is specified, and if the result field is defined elsewhere, the length must be the same as the previously defined length.

Positions 69-70 (Decimal Positions)

Entry	Explanation
Blank	The result field is character data, has been defined elsewhere in the program, or no field name has been specified.
0-30	Number of decimal positions in a numeric result field.

Positions 69-70 indicate the number of positions to the right of the decimal in a numeric result field. If the numeric result field contains no decimal positions, enter a '0' (zero). This position must be blank if the result field is character data or if no field length is specified. The number of decimal positions specified cannot exceed the length of the field.

Positions 71-76 (Resulting Indicators)

These positions can be used, for example, to test the value of a result field after the completion of an operation, or to indicate conditions like end-of-file, error, or record-not-found. For some operations, you can control the way the operation is performed by specifying different combinations of the three resulting indicators (for example, LOOKUP). The resulting indicator positions have different uses, depending on the operation code specified. See the individual operation codes in Chapter 22, "Operation Codes" on page 281 for a description of the associated resulting indicators. For arithmetic operations, the result field is tested only after the field is truncated and half-adjustment is done (if specified). The setting of indicators depends on the results of the tests specified.

Entry	Explanation
Blank	No resulting indicator specified
01-99	General indicators
KA-KN, KP-KY	Function key indicators
H1-H9	Halt indicators
L1-L9	Control level indicators
LR	Last record indicator
OA-OG, OV	Overflow indicators
U1-U8	External indicators
RT	Return indicator.

Resulting indicators cannot be used when the result field uses a non-indexed array.

If the same indicator is used as a resulting indicator on more than one calculation specification, the most recent specification processed determines the status of that indicator.

Remember the following points when specifying resulting indicators:

- When the calculation operation is done, the specified resulting indicators are set off, and, if a condition specified by a resulting indicator is satisfied, that indicator is set on.
- When a control level indicator (L1 through L9) is set on, the lower level indicators are not set on.
- When a halt indicator (H1 through H9) is set on, the program ends abnormally at the next *GETIN point in the cycle, or when a RETURN operation is processed, unless the halt indicator is set off before the indicator is tested.

Calculation Extended Factor 2 Specification Statement

Certain operation codes allow an expression to be used in the extended factor 2 field.

Positions 7-8 (Control Level)

See "Positions 7-8 (Control Level)" on page 237.

Positions 9-11 (Indicators)

See "Positions 9-11 (Indicators)" on page 238.

Positions 12-25 (Factor 1)

Factor 1 must be blank.

Positions 26-35 (Operation and Extender)

Positions 26 through 35 specify the kind of operation to be done using the expression in the extended factor 2 field. The operation code must begin in position 26. For further information on the operation codes, see Chapter 22, "Operation Codes" on page 281.

The program processes the operations in the order specified on the calculation specifications form.

Operation Extender

Entry	Explanation
Blank	No operation extension supplied.
H	Half adjust (round) result of numeric operation

Half adjust may be specified on arithmetic EVAL operations.

Positions 36-80 (Extended Factor 2)

A free form syntax is used in this field. It consists of combinations of operands and operators, and may optionally span multiple lines. If specified across multiple lines, the continuation lines must be blank in positions 7-35. See “Operations Using Expressions” on page 294 or the specific operation codes for more information. See “Continuation rules” on page 169 for more information on coding continuation lines.

Chapter 19. Output Specifications

Output specifications describe the record and the format of fields in a program described output file and when the record is to be written. Output specifications are optional for an externally described file. Output specifications can be divided into two categories: record identification and control (positions 7 through 51), and field description and control (positions 21 through 80). These specifications are entered on the RPG IV Output Specifications.

Detailed information for the output specifications is given in:

- Entries for program described files
- Entries for externally described files

Output Specification Statement

The general layout for the Output specification is as follows:

- the output specification type (O) is entered in position 6
- the non-commentary part of the specification extends from position 7 to position 80
- the comments section of the specification extends from position 81 to position 100

Program Described

For program described files, entries on the output specifications can be divided into two categories:

- Record identification and control (positions 7 through 51)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................Comment+++++++++++++
OFilename++DAddN01N02N03Excnam++++.........................................Comment+++++++++++++
O.........And..N01N02N03Excnam++++.........................................Comment+++++++++++++
```

Figure 88. Program Described Record Layout

- Field description and control (positions 21 through 80). Each field is described on a separate line, below its corresponding record identification entry.

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++Comment+++++++++++++
O..............................................Constant/editword-ContinutioComment+++++++++++++
```

Figure 89. Program Described Field Layout

Externally Described

For externally described files, entries on output specifications are divided into the following categories:

- Record identification and control (positions 7 through 39)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
ORcdname+++D...N01N02N03Excnam++++.........................................Comment+++++++++++++
ORcdname+++DAddN01N02N03Excnam++++.........................................Comment+++++++++++++
O.........And..N01N02N03Excnam++++.........................................Comment+++++++++++++
```

Figure 90. Externally Described Record Layout

- Field description and control (positions 21 through 43, and 45).

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 10
O..............N01N02N03Field+++++++++.B...................................Comment+++++++++++++
```

Figure 91. Externally Described Field Layout

Program Described Files

Position 6 (Form Type)

An O must appear in position 6 to identify this line as an output specifications statement.

Record Identification and Control Entries

Entries in positions 7 through 51 identify the output records that make up the files, provide the correct spacing on printed reports, and determine under what conditions the records are to be written.

Positions 7-16 (File Name)

Entry	Explanation
A valid file name	Same file name that appears on the file description specifications for the output file.

Specify the file name on the first line that defines an output record for the file. The file name specified must be the same file name assigned to the output, update, or combined file on the file description specifications. If records from files are interspersed on the output specifications, the file name must be specified each time the file changes.

For files specified as output, update, combined or input with ADD, at least one output specification is required unless an explicit file operation code with a data structure name specified in the result field is used in the calculations. For example, a WRITE operation does not require output specifications.

Positions 16-18 (Logical Relationship)

Entry	Explanation
AND or OR	AND/OR indicates a relationship between lines of output indicators. AND/OR lines are valid for output records, but not for fields.

Positions 16 through 18 specify AND/OR lines for output operations. To specify this relationship, enter AND/OR in positions 16 through 18 on each additional line following the line containing the file name. At least one indicator must be specified on each AND line. For an AND relationship and fetch overflow position 18 must be specified on the first line only (file name line). A fetch overflow entry is required on OR lines for record types requiring the fetch overflow routine.

Positions 7 through 15 must be blank when AND/OR is specified.

An unlimited number of AND/OR lines can be specified on the output specifications.

Position 17 (Type)

Entry	Explanation
H or D	Detail records usually contain data that comes directly from the input record or that is the result of calculations processed at detail time. Heading records usually contain constant identifying information such as titles, column headings, page number, and date. No distinction is made between heading and detail records. The H/D specifications are available to help the programmer document the program.
T	Total records usually contain data that is the end result of specific calculations on several detail records.
E	Exception records are written during calculation time. Exception records can be specified only when the operation code EXCEPT is used. See Chapter 22, "Operation Codes" for further information on the EXCEPT operation code.

Position 17 indicates the type of record to be written. Position 17 must have an entry for every output record. Heading (H) and detail (D) lines are both processed as detail records. No special sequence is required for coding the output records; however, lines are handled at separate times within the program cycle based on their record type. See Figure 4 on page 14 and Figure 5 on page 16 for more information on when in the cycle output is performed.

Positions 18-20 (Record Addition/Deletion)

Entry	Explanation
ADD	Add a record to the file or subfile.
DEL	Delete the last record read from the file. The deleted record cannot be retrieved; the record is deleted from the system.

An entry of ADD is valid for input, output, or update files. DEL is valid for update DISK files only.

Note: The file description specifications for a file using the ADD function for DISK files must have an A in position 20.

This entry must appear on the same line that contains the record type (H, D, T, E) specification (position 17). If an AND/OR line is used following an ADD or DEL entry, this entry applies to the AND/OR line also.

Position 18 (Fetch Overflow/Release)

Entry	Explanation
Blank	Must be blank for all files except printer files (PRINTER specified in positions 36 through 42 of the file description specifications). If position 18 is blank for printer files, overflow is not fetched.
F	Fetch overflow.
R	Release a device (workstation) after output.

Fetch Overflow

An F in position 18 specifies fetch overflow for the printer file defined on this line. This file must be a printer file that has overflow lines. Fetch overflow is processed only when an overflow occurs and when all conditions specified by the indicators in positions 21 through 29 are satisfied. An overflow indicator cannot be specified on the same line as fetch overflow.

If an overflow indicator has not been specified with the OFLIND keyword on the file description specifications for a printer file, the compiler assigns one to the file. An overflow line is generated by the compiler for the file, except when no other output records exist for the file or when the printer uses externally described data. This compiler-generated overflow can be fetched.

Overflow lines can be written during detail, total, or exception output time. When the fetch overflow is specified, only overflow output associated with the file containing the processed fetch is output. The fetch overflow entry (F) is required on each OR line for record types that require the overflow routine. The fetch overflow routine does not automatically advance forms. For detailed information on the overflow routine see “Overflow Routine” on page 23 and Figure 6 on page 22

The form length and overflow line can be specified using the FORMLEN and OFLIND keywords on the file description specifications, in the printer device file, or through an OS/400 override command.

Release

After an output operation is complete, the device used in the operation is released if you have specified an R in position 18 of the corresponding output specifications. See the “REL (Release)” on page 440 operation for further information on releasing devices.

Positions 21-29 (Output Conditioning Indicators)

Entry	Explanation
Blank	The line or field is output every time the record (heading, detail, total, or exception) is checked for output.
01-99	A general indicator that is used as a resulting indicator, field indicator, or record identifying indicator.
KA-KN, KP-KY	Function key indicators.
L1-L9	Control level indicators.
H1-H9	Halt indicators.
U1-U8	External indicator set before running the program or set as a result of a calculation operation.
OA-OG, OV	Overflow indicator previously assigned to this file.
MR	Matching record indicator.
LR	Last record indicator.
RT	Return indicator.
1P	First-page indicator. Valid only on heading or detail lines.

Conditioning indicators are not required on output lines. If conditioning indicators are not specified, the line is output every time that record is checked for output. Up to three indicators can be entered on one specification line to control when a record or a particular field within a record is written. The indicators that condition the output are coded in positions 22 and 23, 25 and 26, and 28 and 29. When an N is entered in positions 21, 24, or 27, the indicator in the associated position must be off for the line or field to be written. Otherwise, the indicator must be on for the line or field to be written. See “PAGE, PAGE1-PAGE7” on page 251 for information on how output indicators affect the PAGE fields.

If more than one indicator is specified on one line, all indicators are considered to be in an AND relationship.

If the output record must be conditioned by more than three indicators in an AND relationship, enter the letters AND in positions 16 through 18 of the following line and specify the additional indicators in positions 21 through 29 on that line.

For an AND relationship, fetch overflow (position 18) can only be specified on the first line. Positions 40 through 51 (spacing and skipping) must be blank for all AND lines.

An overflow indicator must be defined on the file description specifications with the OFLIND keyword before it can be used as a conditioning indicator. If a line is to be conditioned as an overflow line, the overflow indicator must appear on the main specification line or on the OR line. If an overflow indicator is used on an AND line, the line is *not* treated as an overflow line, but the overflow indicator is checked before the line is written. In this case, the overflow indicator is treated like any other output indicator.

If the output record is to be written when any one of two or more sets of conditions exist (an OR relationship), enter the letters OR in positions 16-18 of the following specification line, and specify the additional OR indicators on that line.

When an OR line is specified for a printer file, the skip and space entries (positions 40 through 51) can all be blank, in which case the space and skip entries of the preceding line are used. If they differ from the preceding line, enter space and skip entries on the OR line. If fetch overflow (position 18) is used, it must be specified on each OR line.

Positions 30-39 (EXCEPT Name)

When the record type is an exception record (indicated by an E in position 17), a name can be placed in these positions of the record line. The EXCEPT operation can specify the name assigned to a group of the records to be output. This name is called an EXCEPT name. An EXCEPT name must follow the rules for using symbolic names. A group of any number of output records can use the same EXCEPT name, and the records do not have to be consecutive records.

When the EXCEPT operation is specified without an EXCEPT name, only those exception records without an EXCEPT name are checked and written if the conditioning indicators are satisfied.

When the EXCEPT operation specifies an EXCEPT name, only the exception records with that name are checked and written if the conditioning indicators are satisfied.

The EXCEPT name is specified on the main record line and applies to all AND/OR lines.

If an exception record with an EXCEPT name is conditioned by an overflow indicator, the record is written only during the overflow portion of the RPG IV cycle or during fetch overflow. The record is not written at the time the EXCEPT operation is processed.

An EXCEPT operation with no fields can be used to release a record lock in a file. The UNLOCK operation can also be used for this purpose. In Figure 92 on page 249, the record lock in file RCDA is released by the EXCEPT operation. For more information, see *ILE Application Development Example*, SC41-3602.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C     KEY           CHAIN     RCDA
C                   EXCEPT    RELESE
ORcdname+++D...N01N02N03Excnam++++....................................
O
O*
ORCDA      E             RELESE
O*                     (no fields)
```

Figure 92. Record Lock in File Released by EXCEPT Operation

Positions 40-51 (Space and Skip)

Use positions 40 through 51 to specify line spacing and skipping for a printer file. Spacing refers to advancing one line at a time, and skipping refers to jumping from one print line to another.

If spacing and skipping are specified for the same line, the spacing and skipping operations are processed in the following sequence:

Skip before
Space before
Print a line
Skip after
Space after.

If the PRTCTL (printer control option) keyword is not specified on the file description specifications, an entry must be made in one of the following positions when the device is PRINTER: 40-42 (space before), 43-45 (space after), 46-48 (skip before), or 49-51 (skip after). If a space/skip entry is left blank, the particular function with the blank entry (such as space before or space after) does not occur. If entries are made in positions 40-42 (space before) or in positions 46-51 (skip before and skip after) and no entry is made in positions 43 - 45 (space after), no space occurs after printing. When PRTCTL is specified, it is used only on records with blanks specified in positions 40 through 51.

If a skip before or a skip after a line on a new page is specified, but the printer is on that line, the skip does not occur.

Positions 40-42 (Space Before)

Entry	Explanation
0 or Blank	No spacing
1-255	Spacing values

Positions 43-45 (Space After)

Entry	Explanation
0 or Blank	No spacing
1-255	Spacing values

Positions 46-48 (Skip Before)

Entry	Explanation
Blank	No skipping occurs.
1-255	Skipping values

Positions 49-51 (Skip After)

Entry	Explanation
Blank	No skipping occurs.
1-255	Skipping values

Field Description and Control Entries

These entries determine under what conditions and in what format fields of a record are to be written.

Each field is described on a separate line. Field description and control information for a field begins on the line following the record identification line.

Positions 21-29 (Output Indicators)

Indicators specified on the field description lines determine whether a field is to be included in the output record, except for PAGE reserved fields. See “PAGE, PAGE1-PAGE7” on page 251 for information on how output indicators affect the PAGE fields. The same types of indicators can be used to control fields as are used to control records, see “Positions 21-29 (Output Conditioning Indicators)” on page 247. Indicators used to condition field descriptions lines cannot be specified in an AND/OR relationship. Conditioning indicators cannot be specified on format name specifications (see “Positions 53-80 (Constant, Edit Word, Date/Time Format, Format Name)” on page 254) for program described WORKSTN files.

Positions 30-43 (Field Name)

In positions 30 through 43, use one of the following entries to specify each field that is to be written out:

- A field name
- Blanks if a constant is specified in positions 53 through 80
- A table name, array name, or array element
- A named constant
- The RPG IV reserved words PAGE, PAGE1 through PAGE7, *PLACE, UDATE, *DATE, UDAY, *DAY, UMONTH, *MONTH, UYEAR, *YEAR, *IN, *INxx, or *IN(xx)
- A data structure name or data structure subfield name.

Note: A pointer field is not a valid output field—that is, pointer fields cannot be written.

Field Names, Blanks, Tables and Arrays

The field names used must be defined in the program. Do not enter a field name if a constant or edit word is used in positions 53-80. If a field name is entered in positions 30 through 43, positions 7 through 20 must be blank.

Fields can be specified in any order because the sequence in which they appear on the output records is determined by the entry in positions 47 through 51. If fields overlap, the last field specified is the only field completely written.

When a non-indexed array name is specified, the entire array is written. An array name with a constant index or variable index causes one element to be written. When a table name is specified, the element last found in a "LOOKUP (Look Up a Table or Array Element)" on page 385 operation is written. The first element of a table is written if no successful LOOKUP operation was done.

The conditions for a record and the field it contains must be satisfied before the field is written out.

PAGE, PAGE1-PAGE7

To use automatic page numbering, code PAGE in positions 30 through 43 as the name of the output field. Indicators specified in positions 21 though 29 condition the resetting of the resetting of the PAGE field, not whether it prints. The PAGE field is always incremented by 1 and printed. If the conditioning indicators are met, it is reset to zero before being incremented by 1 and printed. If page numbers are needed for several output files (or for different numbering within one file), the entries PAGE1 through PAGE7 can be used. The PAGE fields are automatically zero-suppressed by the Z edit code.

For more information on the PAGE reserved words, see "RPG IV Words with Special Functions/Reserved Words."

*PLACE

*PLACE is an RPG IV reserved word that is used to repeat data in an output record. Fields or constants that have been specified on previous specification lines can be repeated in the output record without having the field and end positions named on a new specification line. When *PLACE is coded in positions 30 through 43, all data between the first position and the highest end position previously specified for a field in that output record is repeated until the end position specified in the output record on the *PLACE specification line is reached. The end position specified on the *PLACE specification line must be at least twice the highest end position of the group of fields to be duplicated. *PLACE can be used with any type of output. Blank after (position 45), editing (positions 44, 53 through 80), data format (position 52), and relative end positions cannot be used with *PLACE.

User Date Reserved Words

The user date reserved words (UDATE, *DATE, UDAY, *DAY, UMONTH, *MONTH, UYEAR, *YEAR) allow the programmer to supply a date for the program at run time. For more information on the user date reserved words, see "Rules for User Date."

*IN, *INxx, *IN(xx)

The reserved words *IN, *INxx and *IN(xx) allow the programmer to refer to and manipulate RPG IV indicators as data.

Position 44 (Edit Codes)

Entry	Explanation
Blank	No edit code is used.
1-9, A-D, J-Q, X, Y, Z	Numeric fields are zero-suppressed and punctuated according to a predefined pattern without the use of edit words.

Position 44 is used to specify edit codes that suppress leading zeros in a numeric field or to punctuate a numeric field without using an edit word. Allowable entries are 1 through 9, A through D, J through Q, X, Y, Z, and blank.

For more information on edit codes see Chapter 11, “Editing Numeric Fields” on page 149.

Edit codes 5 through 9 are user-defined edit codes and are defined externally by an OS/400 function. The edit code is determined at compilation time. Subsequent changes to a user-defined edit code will not affect the editing by the RPG IV compiler unless the program is recompiled.

Position 45 (Blank After)

Entry	Explanation
Blank	The field is not reset.
B	The field specified in positions 30 through 43 is reset to blank, zero, or the default date/time/timestamp value after the output operation is complete.

Position 45 is used to reset a numeric field to zeros or a character or graphic field to blanks. Date, time, and timestamp fields are reset to their default values.

If the field is conditioned by indicators in positions 21 through 29, the blank after is also conditioned. This position must be blank for look-ahead, user date reserved words, *PLACE, named constants, and literals.

Resetting fields to zeros may be useful in total output when totals are accumulated and written for each control group in a program. After the total is accumulated and written for one control group, the total field can be reset to zeros before accumulation begins on the total for the next control group.

If blank after (position 45) is specified for a field to be written more than once, the B should be entered on the last line specifying output for that field, or else the field named will be printed as the blank-after value for all lines after the one doing the blank after.

Positions 47-51 (End Position)

Entry	Explanation
1-n	End position
K1-K10	Length of format name for WORKSTN file.

Positions 47 through 51 define the end position of a field or constant on the output record, or define the length of the data description specifications record format name for a program described WORKSTN file.

The K identifies the entry as a length rather than an end position, and the number following the K indicates the length of the record format name. For example, if the format name is CUSPMT, the entry in positions 50 and 51 is K6. Leading zeros are permitted following the K, and the entry must be right-adjusted.

Valid entries for end positions are blanks, +nnnn, -nnnn, and nnnnn. All entries in these positions must end in position 51. Enter the position of the rightmost character of the field or constant. The end position must not exceed the record length for the file.

If an entire array is to be written, enter the end position of the last element in the array in positions 47 through 51. If the array is to be edited, be careful when specifying the end position to allow enough positions to write all edited elements. Each element is edited according to the edit code or edit word.

The +nnnn or -nnnn entry specifies the placement of the field or constant relative to the end position of the previous field. The sign must be in position 47. The number (nnnn) must be right-adjusted, but leading zeros are not required. To calculate the end position, use these formulas :

```
EP = PEP +nnnn + FL

EP = PEP −nnnn + FL
```

EP is the calculated end position. PEP is the previous end position. For the first field specification in the record, PEP is equal to zero. FL is the length of the field after editing, or the length of the constant specified in this specification. The use of +nnnn is equivalent to placing nnnn positions between the fields. A -nnnn causes an overlap of the fields by nnnn positions. For example, if the previous end position (PEP) is 6, the number of positions to be placed between the fields (nnnn) is 5, and the field length (FL) is 10, the end position (EP) equals 21.

When *PLACE is used, an actual end position must be specified; it cannot be blank or a displacement.

An entry of blank is treated as an entry of +0000. No positions separate the fields.

Position 52 (Data Format)

Entry	Explanation
Blank	• For numeric fields the data is to be written in zoned decimal format. • For graphic fields the data is to be written with SO/SI brackets. • For date, time, and timestamp fields the data is to be written without format conversion performed. • For character fields the data is to be written as it is stored.
P	The numeric field is to be written in packed decimal format.
B	The numeric field is to be written in binary format.
L	The numeric field is to be written with a preceding (left) plus or minus sign.
R	The numeric field is to be written with a following (right) plus or minus sign.
D	Date field— the date field will be converted to the format specified in positions 53-80 or to the default file date format.

T Time field— the time field will be converted to the format specified in positions 53-80 or to the default file time format.
Z Valid for Timestamp fields only.
A Valid for Character fields only.
S Valid for Zoned decimal fields only.
G The Graphic field will be written without SO/SI brackets.

This position must be blank if editing is specified.

The entry in position 52 specifies the format of the data in the records in the file. This entry has no effect on the format used for internal processing of the output field in the program.

A 'G' or blank must be specified for a graphic field in a program-described file. If 'G' is specified, then, the data will be output without SO/SI. If this column is blank for program described output, then SO/SI brackets will be placed around the field in the output record by the compiler if the field is of type graphic. You must ensure that there is sufficient room in the output record for both the data and the SO/SI characters.

Positions 53-80 (Constant, Edit Word, Date/Time Format, Format Name)

Positions 53 through 80 are used to specify a constant, a format name, or an edit word for a program described file.

Constants

Constants consist of character data (literals) that does not change from one processing of the program to the next. A constant is the actual data used in the output record rather than a name representing the location of the data.

A constant can be placed in positions 53 through 80. The constant must begin in position 54 (apostrophe in position 53), and it must end with an apostrophe even if it contains only numeric characters. Any apostrophe used within the constant must be entered twice; however, only one apostrophe appears when the constant is written out. The field name (positions 30 through 43) must be blank. Constants can be continued (see “Continuation rules” on page 169 for continuation rules). Instead of entering a constant, you can use a named constant.

Graphic literals or named constants are not allowed as edit words, but may be specified as constants.

Edit Words

An edit word specifies the punctuation of numeric fields, including the printing of dollar signs, commas, periods, and sign status. See “Parts of an Edit Word” on page 156 for details.

Edit words must be character literals or named constants. Graphic and hexadecimal literals and named constants are not allowed.

Date/Time Format

Specify the external format for date/time fields. See “DATFMT(fmt{separator})” on page 174 and “TIMFMT(fmt{separator})” on page 176 for valid date/time formats.

The hierarchy used when determining the external date/time format and separator for date and time fields is:

1. The date format and separator specified in positions 53-58 (or 53-57).
2. From the DATFMT/TIMFMT keyword specified for the current file
3. From the DATFMT/TIMFMT keyword specified in the control specification
4. *ISO

Date and time fields will be converted from their internal date/time format to the external format determined above.

Record Format Name

The name of the data description specifications record format that is used by a program described WORKSTN file must be specified in positions 53 through 62. One format name is required for each output record for the WORKSTN file; specifying more than one format name per record is not allowed. Conditioning indicators cannot be specified on format name specifications for program described WORKSTN files. The format name must be enclosed in apostrophes. You must also enter Kn in positions 47 through 51, where n is the length of the format name. For example, if the format name is ‘CUSPMT’, enter K6 in positions 50 and 51. A named constant can also be used.

Externally Described Files

Position 6 (Form Type)

An O must appear in position 6 to identify this line as an output specifications statement.

Record Identification and Control Entries

Output specifications for an externally described file are optional. Entries in positions 7 through 39 of the record identification line identify the record format and determine under what conditions the records are to be written.

Positions 7-16 (Record Name)

Entry	**Explanation**
A valid record format name	A record format name must be specified for an externally described file.

Positions 16-18 (Logical Relationship)

Entry	**Explanation**
AND or OR	AND/OR indicates a relationship between lines of output indicators. AND/OR lines are valid for output records, but not for fields.

See “Positions 16-18 (Logical Relationship)” on page 245 for more information.

Position 17 (Type)

Entry	Explanation
H or D	Detail records
T	Total records
E	Exception records.

Position 17 indicates the type of record to be written. See "Position 17 (Type)" on page 245 for more information.

Position 18 (Release)

Entry	Explanation
R	Release a device after output.

See "Release" on page 247 for more information.

Positions 18-20 (Record Addition)

Entry	Explanation
ADD	Add a record to a file.
DEL	Delete an existing record from the file.

For more information on record addition, see "Positions 18-20 (Record Addition/Deletion)" on page 246.

Positions 21-29 (Output Indicators)

Output indicators for externally described files are specified in the same way as those for program described files. The overflow indicators OA-OG, OV are not valid for externally described files. For more information on output indicators, see "Positions 21-29 (Output Conditioning Indicators)" on page 247.

Positions 30-39 (EXCEPT Name)

An EXCEPT name can be specified in these positions for an exception record line. See "Positions 30-39 (EXCEPT Name)" on page 248 for more information.

Field Description and Control Entries

For externally described files, the only valid field descriptions are output indicators (positions 21 through 29), field name (positions 30 through 43), and blank after (position 45).

Positions 21-29 (Output Indicators)

Indicators specified on the field description lines determine whether a field is to be included in the output record. The same types of indicators can be used to control fields as are used to control records. See "Positions 21-29 (Output Conditioning Indicators)" on page 247 for more information.

Positions 30-43 (Field Name)

Entry	Explanation
Valid field name	A field name specified for an externally described file must be present in the external description unless the external name was renamed for the program.
*ALL	Specifies the inclusion of all the fields in the record.

For externally described files, only the fields specified are placed in the output record. *ALL can be specified to include all the fields in the record. If *ALL is specified, no other field description lines can be specified for that record. In particular, you cannot specify a B (blank after) in position 45.

For an update record, only those fields specified in the output field specifications and meeting the conditions specified by the output indicators are placed in the output record to be rewritten. The values that were read are used to rewrite all other fields.

For the creation of a new record (ADD specified in positions 18-20), the fields specified are placed in the output record. Those fields not specified or not meeting the conditions specified by the output indicators are written as zeros or blanks, depending on the data format specified in the external description.

Position 45 (Blank After)

Entry	Explanation
Blank	The field is not reset.
B	The field specified in positions 30 through 43 is reset to blank, zero, or the default date/time/timestamp value after the output operation is complete.

Position 45 is used to reset a numeric field to zeros or a character or graphic field to blanks. Date, time, and timestamp fields are reset to their default values.

If the field is conditioned by indicators in positions 21 through 29, the blank after is also conditioned. This position must be blank for look-ahead, user date reserved words, *PLACE, named constants, and literals.

Resetting fields to zeros may be useful in total output when totals are accumulated and written for each control group in a program. After the total is accumulated and written for one control group, the total field can be reset to zeros before accumulation begins on the total for the next control group.

If blank after (position 45) is specified for a field to be written more than once, the B should be entered on the last line specifying output for that field, or else the field named will be printed as the blank-after value for all lines after the one doing the blank after.

Built-in Functions, Expressions, and Operation Codes

This section describes the various ways in which you can manipulate data or devices. First, there is a description of built-in functions and their use on definition specifications and calculation specifications. Next, expressions and the rules governing them are described. Finally, there are two chapters dealing with operation codes. The first groups the operation codes functionally and provides information common to operation codes within those groups. The second and final chapter, describes the operation codes in detail in alphabetical order.

Chapter 20. Built-in Functions

Built-in functions are similar to operation codes in that they perform operations on data you specify. All built-in functions have the percent symbol (%) as their first character. The syntax of built-in functions is

```
function-name(argument{:argument...})
```

Arguments for the function may be variables, constants, expressions, or other built-in functions. An expression argument can include a built-in function. The following example illustrates this.

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C*
C* This example shows a complex expression with multiple
C* nested built-in functions.
C*
C* %TRIM takes as its argument a string.  In this example, the
C* argument is the concatenation of string A and the string
C* returned by the %SUBST built-in function.  %SUBST will return
C* a substring of string B starting at position 11 and continuing
C* for the length returned by %SIZE minus 20.  %SIZE will return
C* the length of string B.
C*
C* If A is the string '     Toronto,' and B is the string
C* '  Ontario, Canada            ' then the argument for %TRIM will
C* be '     Toronto, Canada            ' and RES will have the value
C* 'Toronto, Canada'.
C*
C                   EVAL      RES = %TRIM(A + %SUBST(B:11:%SIZE(B) - 20))
```

Figure 93. Built-in Function Arguments Example

See the individual built-in function descriptions for details on what arguments are allowed. Unlike operation codes, built-in functions return a value rather than placing a value in a result field. The following example illustrates this difference.

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* In the following example, CITY contains the string
C* 'Toronto, Ontario'.  The SCAN operation is used to locate the
C* separating blank, position 9 in this illustration.  SUBST
C* places the string 'Ontario' in field TCNTRE.
C*
C* Next, TCNTRE is compared to the literal 'Ontario' and
C* 1 is added to CITYCNT.
C*
C     ' '           SCAN      CITY          C
C                   ADD       1             C
C                   SUBST     CITY:C        TCNTRE
C     'Ontario'     IFEQ      TCNTRE
C                   ADD       1             CITYCNT
C                   ENDIF
C*
C* In this example, CITY contains the same value, but the
C* variable TCNTRE is not necessary since the %SUBST built-in
C* function returns the appropriate value.  In addition, the
C* intermediary step of adding 1 to C is simplified since
C* %SUBST accepts expressions as arguments.
C*
C     ' '           SCAN      CITY          C
C                   IF        %SUBST(CITY:C+1) = 'Ontario'
C                   EVAL      CITYCNT = CITYCNT+1
C                   ENDIF
```

Figure 94. Built-in Function Example

Note that the arguments used in this example (the variable CITY and the expression C+1) are analogous to the factor values for the SUBST operation. The return value of the function itself is analogous to the result. In general, the arguments of the built-in function are similar to the factor 1 and factor 2 fields of an operation code.

Another useful feature of built-in functions is that they can simplify maintenance of your code when used on the definition specification. The following example demonstrates this feature.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D*
D* In this example, CUSTNAME is a field in the
D* externally described data structure CUSTOMER.
D* If the length of CUSTNAME is changed, the attributes of
D* both TEMPNAME and NAMEARRAY would be changed merely by
D* recompiling.  The use of the %SIZE built-in function means
D* no changes to your code would be necessary.
D*
D CUSTOMER      E DS
D                 DS
D TEMPNAME                            LIKE(CUSTNAME)
D NAMEARRAY                       1   OVERLAY(TEMPNAME)
D                                     DIM(%SIZE(TEMPNAME))
```

Figure 95. Simplified Maintenance with Built-in Functions

Built-in functions can be used in expressions on the extended factor 2 calculation specification and with keywords on the definition specification. When used with definition specification keywords, the value of the built-in function must be known at compile time and the argument cannot be an expression.

The following table lists the built-in functions, their arguments, and the value they return.

Built-in Function Name	Argument(s)	Value Returned
%ADDR	variable name	address of variable
%ELEM	array, table, or multiple occurrence data structure name	number of elements or occurrences
%PADDR	procedure name	address of procedure
%SIZE	variable, array, or literal{:*ALL}	size of variable or literal
%SUBST	string:start{:length}	substring
%TRIM	string	string with left and right blanks trimmed
%TRIML	string	string with left blanks trimmed
%TRIMR	string	string with right blanks trimmed

For more information see

- Chapter 16, "Definition Specification" on page 199
- "Calculation Extended Factor 2 Specification Statement" on page 241
- "Operations Using Expressions" on page 294
- "DOU (Do Until)" on page 349
- "DOW (Do While)" on page 352
- "EVAL (Evaluate expression)" on page 362
- "IF (If)" on page 374
- "WHEN (When True Then Select)" on page 485

Built-in Functions Alphabetically

%ADDR (Get Address of Variable)

```
%ADDR(variable)
%ADDR(variable(index))
%ADDR(variable(expression))
```

%ADDR returns a value of type basing pointer. The value is the address of the specified variable. It may only be compared with and assigned to items of type basing pointer.

If %ADDR with an array index parameter is specified as parameter for definition specification keywords INZ or CONST, the array index must be known at compile-time. The index must be either a numeric literal or a numeric constant.

In an EVAL operation where the result of the assignment is an array with no index, %ADDR on the right hand side of the assignment operator has a different meaning depending on the argument for the %ADDR. If the argument for %ADDR is an array name without an index and the result is an array name, each element of the result array will contain the address of the beginning of the argument array. If the argument for %ADDR is an array name with an index of (*), then each element of the result array will contain the address of the corresponding element in the argument array. This is illustrated in “%ADDR Example”

If the variable specified as parameter is a table, multiple occurrence data structure, or subfield of a multiple occurrence data structure, the address will be the address of the current table index or occurrence number.

If the variable is based, %ADDR returns the value of the basing pointer for the variable. If the variable is a subfield of a based data structure, the value of %ADDR is the value of the basing pointer plus the offset of the subfield.

If the variable is specified as a PARM of the *ENTRY PLIST, %ADDR returns the address passed to the program by the caller.

%ADDR Example

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
D* The following set of definitions is valid since the array
D* index has a compile-time value
D*
D   ARRAY         S             20A   DIM (100)
D* Set the pointer to the address of the seventh element of the array.
D   PTR           S               *   INZ (%ADDR(ARRAY(SEVEN)))
D   SEVEN         C                   CONST (7)
D*
D DS1             DS                  OCCURS (100)
D                               20A
D   SUBF                        10A
D                               30A
D CHAR10          S             10A   BASED (P)
D PARRAY          S               *   DIM(100)
```

Figure 96 (Part 1 of 2). %ADDR Example

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C*
C     23             OCCUR     DS1
C                    EVAL      SUBF = *ALL'abcd'
C                    EVAL      P = %ADDR (SUBF)
C
C                    IF        CHAR10 = SUBF
C*                      This condition is true
C                    ENDIF
C
C                    IF        %ADDR (CHAR10) = %ADDR (SUBF)
C*                      This condition is also true
C                    ENDIF
C
C* The following statement also changes the value of SUBF
C                    EVAL      CHAR10 = *ALL'efgh'
C
C                    IF        CHAR10 = SUBF
C*                      This condition is still true
C                    ENDIF
 *---------------------------------------------------------------
C           24       OCCUR     DS1
C                    IF        CHAR10 = SUBF
C*                      This condition is no longer true
C                    ENDIF
 *---------------------------------------------------------------
C* The address of an array element is taken using an expression
C* as the array index
C*
C                    EVAL      P = %ADDR (ARRAY (X + 10))
 *---------------------------------------------------------------
C* Each element of the array PARRAY contains the address of the
C* first element of the array ARRAY.
C                    EVAL      PARRAY = %ADDR(ARRAY)
C* Each element of the array PARRAY contains the address of the
C* corresponding element of the array ARRAY
C                    EVAL      PARRAY = %ADDR(ARRAY(*))
```

Figure 96 (Part 2 of 2). %ADDR Example

%ELEM (Get Number of Elements)

```
%ELEM(table_name)
%ELEM(array_name)
%ELEM(multiple_occurrence_data_structure_name)
```

%ELEM returns the number of elements in the specified array, table, or multiple-occurrence data structure. It may be specified anywhere a numeric constant is allowed in the definition specification or in an expression in the extended factor 2 field.

The parameter must be the name of an array, table, or multiple occurrence data structure.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D
D arr1d           S             20    DIM(10)
D table           S             10    DIM(20) ctdata
D mds             DS            20    occurs(30)
D like_array      S                   like(arr1d) dim(%elem(arr1d))
D array_dims      C                   const (%elem (arr1d))
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C
C* In the following examples num will be equal to 10, 20, and 30.
C*
C                   EVAL      num = %elem (arr1d)
C                   EVAL      num = %elem (table)
C                   EVAL      num = %elem (mds)
```

Figure 97. %ELEM Example

%PADDR (Get Procedure Address)

```
%PADDR(string)
```

%PADDR returns a value of type procedure pointer. The value is the address of the entry point specified as the argument.

%PADDR may be compared with and assigned to only items of type procedure pointer.

The parameter to %PADDR must be a character or hexadecimal literal or a constant name that represents a character or hexadecimal literal. The entry point name specified by the character string must be found at program bind time and must be in the correct case.

%PADDR Examples

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D
D PROC            S               *   PROCPTR
D                                     INZ (%PADDR ('FIRSTPROG'))
D PROC1           S               *   PROCPTR
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C*
C*  The following statement calls procedure 'FIRSTPROG'.
C*
C                   CALLB     PROC
 *------------------------------------------------------------------
C* The following statements call procedure 'NextProg'.
C* This a C procedure and is in mixed case.  Note that
C* the procedure name is case sensitive.
C*
C                   EVAL      PROC1 = %PADDR ('NextProg')
C                   CALLB     PROC1
```

%SIZE

```
%SIZE(variable)
%SIZE(literal)
%SIZE(array{:*ALL})
%SIZE(table{:*ALL})
%SIZE(multiple occurrence data structure{:*ALL})
```

%SIZE returns the number of bytes occupied by the constant or field. The argument may be a literal, a named constant, a data structure, a data structure subfield, a field, an array or a table name.

For a graphic literal, the size is the number of bytes occupied by the graphic characters, not including leading and trailing shift characters. For a graphic field, the size returned is the single byte length.

If the argument is an array name, table name, or multiple occurrence data structure name, the value returned is the size of one element or occurrence. If *ALL is specified as the second parameter for %SIZE, the value returned is the storage taken up by all elements or occurrences. For a multiple occurrence data structure containing pointer subfields, the size may be greater than the size of one occurrence times the number of occurrences. This is possible because the system requires that pointers be placed in storage at addresses evenly divisible by 16. This means that the length of each occurrence may have to be increased enough to make the length an exact multiple of 16 so that the pointer subfields will be positioned correctly in storage for every occurrence.

%SIZE may be specified anywhere that a numeric constant is allowed on the definition specification and in an expression in the extended factor 2 field of the calculation specification.

%SIZE Examples

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D
D arr1            S             10    DIM(4)
D table1          S              5    DIM(20)
D field1          S             10
D field2          S              9B 0
D field3          S              5P 2
D mds             DS            20    occurs(10)
D mds_size        C                   const (%size (mds: *all))
D mds_ptr         DS            20    OCCURS(10)
D   pointer                       *
d
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C                                                                      Result
C                   eval      num = %SIZE(field1)                          10
C                   eval      num = %SIZE('HH')                             2
C                   eval      num = %SIZE(123.4)                            4
C                   eval      num = %SIZE(-03.00)                           4
C                   eval      num = %SIZE(arr1)                            10
C                   eval      num = %SIZE(arr1:*ALL)                       40
C                   eval      num = %SIZE(table1)                           5
C                   eval      num = %SIZE(table1:*ALL)                    100
C                   eval      num = %SIZE(mds)                             20
C                   eval      num = %SIZE(mds:*ALL)                       200
C                   EVAL      num = %SIZE(mds_ptr)                         20
C                   EVAL      num = %SIZE(mds_ptr:*ALL)                   320
C                   eval      num = %SIZE(field2)                           4
C                   eval      num = %SIZE(field3)                           3
```

%SUBST

```
%SUBST(string:start{:length})
```

%SUBST returns a portion of argument string. It may also be used as the result of an assignment with the EVAL operation code.

The start parameter represents the starting position of the substring.

The length parameter represents the length of the substring. If it is not specified, the length is the length of the string parameter less the start value plus one.

The string must be character or graphic data. Starting position and length may be any numeric value or numeric expression with zero decimal positions. The starting position must be greater than zero. The length may be greater than or equal to zero.

When specified as a parameter for a definition specification keyword, the parameters must be literals or named constants representing literals. When specified on a free-form calculation specification, the parameters may be any expression.

%SUBST Used for its Value

%SUBST returns a substring from the contents of the specified string. The string may be any character or graphic field or expression. Unindexed arrays are allowed for string, start, and length. The substring begins at the specified starting position in the string and continues for the length specified. If length is not specified then the substring continues to the end of the string. For example:

```
The value of  %subst('Hello World': 5+2) is  'World'
The value of  %subst('Hello World':5+2:10-7) is 'Wor'
The value of  %subst('abcd' + 'efgh':4:3) is 'def'
```

For graphic characters the start position and length is consistent with the 2-byte character length (position 3 is the third 2-byte character and length 3 represents 3 2-byte characters to be operated on).

%SUBST Used as the Result of an Assignment

When used as the result of an assignment this built-in function refers to certain positions of the argument string. Unindexed arrays are not allowed for start and length.

The result begins at the specified starting position in the variable and continues for the length specified. If length is not specified or it refers to characters beyond the end of the string then the string is referenced to its end.

When %SUBST is used as the result of an assignment, the first parameter must refer to a storage location. That is, the first parameter of the %SUBST operation must be one of the following.

- Field
- Data Structure
- Data Structure Subfield
- Array Name

- Array Element
- Table Element

Any valid expressions are permitted for the the second and third parameters of %SUBST when it appears as the result of an assignment with an EVAL operation.

%SUBST Example

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C*
C* In this example, CITY contains 'Toronto, Ontario'
C* %SUBST returns the value 'Ontario'.
C*
C     ' '           SCAN      CITY          C
C                   IF        %SUBST(CITY:C+1) = 'Ontario'
C                   EVAL      CITYCNT = CITYCNT+1
C                   ENDIF
C*
C* Before the EVAL, A has the value 'abcdefghijklmno'.
C* After the EVAL A has the value 'ab****ghijklmno'
C*
C                   EVAL      %SUBST(A:3:4) = '****'
```

%TRIM

`%TRIM(string)`

%TRIM returns the given string less any leading and trailing blanks.

The string can be character or graphic data.

When specified as a parameter for a definition specification keyword, the string parameter must be a constant.

%TRIM Examples

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D
D LOCATION        S             16A
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C*
C* LOCATION will have the value 'Toronto, Ontario'.
C*
C                   EVAL      LOCATION = %TRIM('  Toronto, Ontario  ')
C*
C* Name will have the value 'Chris Smith'.
C*
C                   MOVE(P)   'Chris'       FIRSTNAME        10
C                   MOVE(P)   'Smith'       LASTNAME         10
C                   EVAL      NAME =
C                             %TRIM(FIRSTNAME) +' '+ %TRIM(LASTNAME)
```

%TRIML

```
%TRIML(string)
```

%TRIML returns the given string less any leading blanks.

The string can be character or graphic data.

When specified as a parameter for a definition specification keyword, the string parameter must be a constant.

%TRIML Examples

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C*
C* LOCATION will have the value 'Toronto, Ontario  '.
C*
C                   EVAL      LOCATION = %TRIML('  Toronto, Ontario  ')
```

%TRIMR

```
%TRIMR(string)
```

%TRIMR returns the given string less any trailing blanks.

The string can be character or graphic data.

When specified as a parameter for a definition specification keyword, the string parameter must be a constant.

%TRIMR Examples

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D
D LOCATION        S             18A
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C*
C* LOCATION will have the value '  Toronto, Ontario'.
C*
C                   EVAL      LOCATION = %TRIMR('  Toronto, Ontario  ')
C*
C* Name will have the value 'Chris Smith'.
C*
C                   MOVEL(P)  'Chris'       FIRSTNAME        10
C                   MOVEL(P)  'Smith'       LASTNAME         10
C                   EVAL      NAME =
C                             %TRIMR(FIRSTNAME) +' '+ %TRIMR(LASTNAME)
```

Chapter 21. Expressions

Expressions are simply groups of operands and operators. For example,

```
A + B * C
STRINGA + STRINGC
D = %ELEM(ARRAYNAME)
```

Expressions can be used on the extended factor 2 calculation specification.

Expression Operators and Operands

Expression operators may be any of the following:

- Unary
  - +
  - -
  - NOT (negation of indicator)
- Binary
  - + (addition for numerics or concatenation for character or graphic data)
  - - (subtraction)
  - * (multiplication)
  - ** (exponentiation)
  - / (division)
  - = (equal)
  - >= (greater than or equal)
  - > (greater than)
  - <= (less than or equal)
  - < (less than)
  - <> (not equal)
  - AND (logical and)
  - OR (logical or)
- Built-in functions

This list indicates the precedence of operators from highest to lowest:

1. ()
2. built-in functions
3. unary +, unary -, NOT
4. **
5. *,/
6. binary +, binary -
7. =,>=,>,<=,<,<>
8. AND
9. OR

Precedence rules determine the order in which operations are performed within expressions. High precedence operations are performed before lower precedence operations. Since parentheses have the highest precedence, operations within parentheses are always performed first. Operators of the same precedence are evaluated in left to right order. The following examples illustrate this point.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Extended-factor2+++++++++++++++++++++++++++
C*
C* The following two operations produce different
C* results although the order of operands and operators
C* is the same.  If PRICE = 100, DISCOUNT = 10, and
C* TAXRATE = 0.15 the first EVAL would result in a TAX of
C* 98.5.  Since multiplication has a higher precedence than
C* subtraction, DISCOUNT * TAXRATE is the first operation
C* performed.  The result of that operation (1.5) is then
C* subtracted from PRICE.
C*
C* The second EVAL would result in a TAX of 13.50.  Since
C* parentheses have the highest precedence the operation
C* within parenthesis is performed first and the result
C* of that operation (90) is then multiplied by TAXRATE.
C*
C                   EVAL      TAX = PRICE - DISCOUNT * TAXRATE
C                   EVAL      TAX = (PRICE - DISCOUNT) * TAXRATE
```

Figure 98. Expression Examples

Any data field, named constant, or literal can be used as an operand.

Table 25 describes the type of operand allowed for each unary operator and the type of the result. Table 26 describes the type of operands allowed for each binary operator and the type of the result. Table 27 on page 277 describes the type of operands allowed for each built-in function and the type of the result.

Table 25. Types Supported for Unary Operations

Operation	Operand Type	Result Type
- (negation)	Numeric	Numeric
+	Numeric	Numeric
NOT	Character (indicator)	Character (indicator)

Table 26 (Page 1 of 2). Operands Supported for Binary Operations

Operator	Operand 1 Type	Operand 2 Type	Result Type
+ (addition)	Numeric	Numeric	Numeric
- (subtraction)	Numeric	Numeric	Numeric
* (multiplication)	Numeric	Numeric	Numeric
/ (division)	Numeric	Numeric	Numeric
** (exponentiation)	Numeric	Numeric	Numeric
+ (concatenation)	Character	Character	Character
+ (concatenation)	Graphic	Graphic	Graphic
Note: For the following operations the operands may be of any type, but the two operands must be of the same type.			
= (equal to)	Any	Any	Character (indicator)
>= (greater than or equal to)	Any	Any	Character (indicator)

Table 26 (Page 2 of 2). Operands Supported for Binary Operations

Operator	Operand 1 Type	Operand 2 Type	Result Type
> (greater than)	Any	Any	Character (indicator)
<= (less than or equal to)	Any	Any	Character (indicator)
< (less than)	Any	Any	Character (indicator)
<> (not equal to)	Any	Any	Character (indicator
AND (logical and)	Character (indicator)	Character (indicator)	Character (indicator)
OR (logical or)	Character (indicator)	Character (indicator)	Character (indicator)

Table 27. Types Supported for Built-in Functions

Operation	Operand 1 Type	Operand 2 Type	Operand 3 Type	Result Type
%SUBST	Character	Numeric	{Numeric}	Character
%SUBST	Graphic	Numeric	{Numeric}	Graphic
%TRIM	Character			Character
%TRIM	Graphic			Graphic
%TRIML	Character			Character
%TRIML	Graphic			Graphic
%TRIMR	Character			Character
%TRIMR	Graphic			Graphic
Note: For the following built-in functions, arguments must be literals, named constants or variables.				
%ELEM	Any			Numeric
%PADDR	Character			Procedure pointer
%SIZE	Any	{*ALL}		Numeric
Note: For the following built-in functions, arguments must be literals, named constants or variables. However, if an array index is specified, it may be any valid numeric expression.				
%ADDR	Any			Basing pointer

The following example demonstrates how expressions can be used.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++
C*
C* The operations within the DOU group will iterate until the
C* logical expression is true.  That is, either COUNTER is less
C* than MAXITEMS or indicator 03 is on.
C*
C                   DOU       COUNTER < MAXITEMS OR *IN03
C*
C* The operations controlled by the IF operation will occur if
C* DUEDATE (a date variable) is an earlier date than
C* December 31, 1994.
C*
C                   IF        DUEDATE < D'12-31-94'
C*
C* In this numeric expression, COUNTER is assigned the value
C* of COUNTER plus 1.
C*
C                   EVAL      COUNTER = COUNTER + 1
C*
C* This numeric expression uses a built-in function to assign
C* the number of elements in the array ARRAY to the variable
C* ARRAYSIZE.
C*
C                   EVAL      ARRAYSIZE = %ELEM(ARRAY)
C*
C* This expression calculates interest and performs half
C* adjusting on the result which is placed in the variable
C* INTEREST.
C*
C                   EVAL (H)  INTEREST = BALANCE * RATE
C*
C* This character expression concatenates the elements of a
C* phone number and places them in the character variable
C* PHONE.  If AREACODE is '416', EXCHANGE is '448', and NUMBER
C* is '3616', PHONE will be '(416) 448-3616'.  Note that no
C* continuation character is needed to continue the
C* expression.  The final + on the first line is a
C* concatenation operator, not a continuation character.
C* Note that all the variables are character.  You cannot
C* concatenate numeric data.
C*
C                   EVAL      PHONE = '(' + AREACODE + ')' +
C                             ' ' + EXCHANGE + '-' + NUMBER
```

Figure 99. Expression Examples

Expression Rules

The following are general rules which apply to all expressions:

- An expression can be coded in the Extended-Factor 2 field
- It can be continued as required
- Blanks (like parentheses) are required only to resolve ambiguity. However, they may be used to enhance readability. It is the programmer's responsibility to ensure that the expression will be interpreted as expected. For example, X**DAY is X multiplied by *DAY while X** DAY is X raised to the power of DAY.

Precision Rules

The precision of intermediate results are based on the attributes of the operands.

Table 28. Precision of Intermediate Results

Operation	Result Precision
Note: The following operations produce a numeric result. Ln is the length of the operand in digits where n is either r for result or a numeric representing the operand. Dn is the number of digits to the right of the decimal point where n is either r for result or a numeric representing the operand. T is the temporary value. Note that if any operand has a floating point representation (for example, it is the result of the exponentiation operator), the result also is a floating point value and the precision rules no longer apply. A floating point value has the precision available from double precision floating point representation.	

N1+N2	T=min (max (L1-D1, L2-D2)+1, 30)
	Dr=min (max (D1,D2), 30-t)
	Lr=t+Dr

N1-N2	T=min (max (L1-D1, L2-D2)+1, 30)
	Dr=min (max (D1,D2), 30-t)
	Lr=t+Dr

N1*N2	Lr=min (L1+L2, 30)
	Dr=min (D1+D2, 30-min ((L1-D1)+(L2-D2), 30))

N1/N2	Lr=30
	Dr=max (30-((L1-D1)+D2), 0)

N1**N2	Double float
Note: The following operations produce a character result. Ln represents the length of the operand in number of characters.	
C1+C2	Lr=min(L1+L2,32767)
Note: The following operations produce a a DBCS result. Ln represents the length of the operand in number of DBCS characters.	
D1+D2	Lr=min(L1+L2,16383)
Note: The following operations produce a result of type character with subtype indicator. The result is always an indicator value (1 character).	
V1=V2	1 (indicator)
V1>=V2	1 (indicator)
V1>V2	1 (indicator)
V1<=V2	1 (indicator)
V1<V2	1 (indicator)
V1<>V2	1 (indicator)
V1 AND V2	1 (indicator)
V1 OR V2	1 (indicator)

Chapter 22. Operation Codes

The RPG IV programming language allows you to do many different types of operations on your data. Operation codes, which are entered on the calculation specifications, indicate the operation to be done. Usually they are abbreviations of the name of the operation.

Operation codes can be categorised by function. The first part of this chapter includes general information about these categories. The latter part of the chapter describes each operation code in alphabetical order and shows one or more examples for most of the operations.

The tables on the next few pages are a summary of the specifications for each operation code.

- An empty column indicates that the field must be blank.
- All underlined fields are required.
- An underscored space denotes that there is no resulting indicator in that position.
- Symbols

(H)	Half adjust (round the numeric result).
(N)	Do not lock record.
(P)	Pad the result with blanks or zeros.
(D)	Operational descriptors or Date field.
(T)	Time field.
(Z)	Timestamp field.
+	Plus.
–	Minus.
BL	Blank(s).
BN	Blank(s) then numeric.
BOF	Beginning of the file.
EOF	End of the file.
EQ	Equal.
ER	Error.
FD	Found.
HI	Greater than.
IN	Indicator.
LO	Less than.
LR	Last record.
NR	No record was found.
NU	Numeric.

OF	Off.
ON	On.
Z	Zero.
ZB	Zero or Blank.

Table 29 (Page 1 of 5). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators		
				71-72	73-74	75-76
ACQ	Device name	WORKSTN file			ER	
ADD (H)	Addend	Addend	Sum	+	–	Z
ADDDUR	Date/Time	Duration:Duration Code	Date/Time		ER	
ANDxx	Comparand	Comparand				
BEGSR	Subroutine name					
BITOFF		Bit numbers	Character field			
BITON		Bit numbers	Character field			
CABxx	Comparand	Comparand	Label	HI	LO	EQ
CALL		Program name	Plist name		ER	LR
CALLB (D)		Procedure name or Procedure pointer	Plist name		ER	LR
CASxx	Comparand	Comparand	Subroutine name	HI	LO	EQ
CAT (P)	Source string 1	Source string 2:number of blanks	Target string			
CHAIN (N)	Search argument	File name	Data structure	NR	ER	
CHECK[2]	Comparator String	Base String:start	Left-most Position(s)		ER	FD
CHECKR[2]	Comparator String	Base String:start	Right-most Position(s)		ER	FD
CLEAR	*NOKEY	*ALL	Structure or Variable or Record format			
CLOSE		File name			ER	
COMMIT	Boundary				ER	
COMP[1]	Comparand	Comparand		HI	LO	EQ
DEFINE	*LIKE	Referenced field	Defined field			
DEFINE	*DTAARA	External data area	Internal field			
DELETE	Search argument	File name		NR	ER	
DIV (H)	Dividend	Divisor	Quotient	+	–	Z
DO	Starting value	Limit value	Index value			
DOU		Indicator expression				
DOUxx	Comparand	Comparand				

[1] At least 1 resulting indicator is required.

[2] A found indicator is required if the result field is not specified.

[3] You must specify factor 2 or the result field. You may specify both.

[4] You must specify factor 1 or the result field. You may specify both.

Table 29 (Page 2 of 5). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators		
				71-72	73-74	75-76
DOW		Indicator expression				
DOWxx	Comparand	Comparand				
DSPLY[4]	Message identifier	Output queue	Response		ER	
DUMP	Identifier					
ELSE						
END		Increment value				
ENDCS						
ENDDO		Increment value				
ENDIF						
ENDSL						
ENDSR	Label	Return point				
EVAL		Result = Expression				
EXCEPT		EXCEPT name				
EXFMT		Record format name			ER	
EXSR		Subroutine name				
EXTRCT		Date/Time:Duration Code	Target Field		ER	
FEOD		File name			ER	
FORCE		File name				
GOTO		Label				
IF		Indicator expression				
IFxx	Comparand	Comparand				
IN	*LOCK	Data area name			ER	
ITER						
KFLD			Key field			
KLIST	KLIST name					
LEAVE						
LOOKUP[1] **(array)**	Search argument	Array name		HI	LO	EQ
LOOKUP[1] **(table)**	Search argument	Table name	Table name	HI	LO	EQ
MHHZO		Source field	Target field			
MHLZO		Source field	Target field			
MLHZO		Source field	Target field			
MLLZO		Source field	Target field			

[1] **At least 1 resulting indicator is required.**

[2] **A found indicator is required if the result field is not specified.**

[3] **You must specify factor 2 or the result field. You may specify both.**

[4] **You must specify factor 1 or the result field. You may specify both.**

Table 29 (Page 3 of 5). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators		
				71-72	73-74	75-76
MOVE (P)	Date/Time Format	Source field	Target field	+	–	ZB
MOVEA (P)		Source	Target	+	–	ZB
MOVEL (P)	Date/Time Format	Source field	Target field	+	–	ZB
MULT (H)	Multiplicand	Multiplier	Product	+	–	Z
MVR			Remainder	+	–	Z
NEXT	Program device	File name			ER	
OCCUR	Occurrence value	Data structure	Occurrence value		ER	
OPEN		File name			ER	
ORxx	Comparand	Comparand				
OTHER						
OUT	*LOCK	Data area name			ER	
PARM	Target field	Source field	Parameter			
PLIST	PLIST name					
POST[3]	Program device	File name	INFDS name		ER	
READ (N)		File name, Record name	Data structure		ER	EOF
READC		Record name			ER	EOF
READE (N)	Search argument	File name, Record name	Data structure		ER	EOF
READP (N)		File name, Record name	Data structure		ER	BOF
READPE (N)	Search argument	File name, Record name	Data structure		ER	BOF
REL	Program device	File name			ER	
RESET	*NOKEY	*ALL	Structure or Variable or Record format		ER	
RETURN						
ROLBK					ER	
SCAN[2]	Comparator string:length	Base string:start	Left-most position(s)		ER	FD
SELECT						
SETGT	Search argument	File name		NR	ER	
SETLL	Search argument	File name		NR	ER	EQ

[1] At least 1 resulting indicator is required.

[2] A found indicator is required if the result field is not specified.

[3] You must specify factor 2 or the result field. You may specify both.

[4] You must specify factor 1 or the result field. You may specify both.

Table 29 (Page 4 of 5). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators		
				71-72	73-74	75-76
SETOFF[1]				OF	OF	OF
SETON[1]				ON	ON	ON
SHTDN				ON		
SORTA		Array name				
SQRT (H)		Value	Root			
SUB (H)	Minuend	Subtrahend	Difference	+	–	Z
SUBDUR (duration)	Date/Time	Date/Time	Duration: Duration Code		ER	
SUBDUR (new date)	Date/Time	Duration:Duration Code	Date/Time		ER	
SUBST (P)	Length to extract	Base string:start	Target string		ER	
TAG	Label					
TEST (D)	Date Format		Date or Character field or Numeric field		ER	
TEST (T)	Time Format		Time or Character field or Numeric field		ER	
TEST (Z)			Timestamp or Character field or Numeric field		ER	
TESTB[1]		Bit numbers	Character field	OF	ON	EQ
TESTN[1]			Character field	NU	BN	BL
TESTZ[1]			Character field			
TIME			Numeric field			
UNLOCK		Data area, record, or file name			ER	
UPDATE		File name	Data structure		ER	
WHEN		Indicator expression				
WHENxx	Comparand	Comparand				

[1] At least 1 resulting indicator is required.

[2] A found indicator is required if the result field is not specified.

[3] You must specify factor 2 or the result field. You may specify both.

[4] You must specify factor 1 or the result field. You may specify both.

Table 29 (Page 5 of 5). Operation Code Specifications Summary

Codes	Factor 1	Factor 2	Result Field	Resulting Indicators		
				71-72	73-74	75-76
WRITE		File name	Data structure		ER	EOF
XFOOT (H)		Array name	Sum	+	–	Z
XLATE (P)	From:To	String:start	Target String		ER	
Z-ADD (H)		Addend	Sum	+	–	Z
Z-SUB (H)		Subtrahend	Difference	+	–	Z

[1] **At least 1 resulting indicator is required.**

[2] **A found indicator is required if the result field is not specified.**

[3] **You must specify factor 2 or the result field. You may specify both.**

[4] **You must specify factor 1 or the result field. You may specify both.**

Arithmetic Operations

The arithmetic operations are:

- “ADD (Add)” on page 308
- “DIV (Divide)” on page 346
- “MULT (Multiply)” on page 411
- “MVR (Move Remainder)” on page 412
- “SQRT (Square Root)” on page 463
- “SUB (Subtract)” on page 464
- “XFOOT (Summing the Elements of an Array)” on page 491
- “Z-ADD (Zero and Add)” on page 494
- “Z-SUB (Zero and Subtract)” on page 495.

For examples of arithmetic operations, see Figure 100 on page 289.

Remember the following when specifying arithmetic operations:

- Arithmetic operations can be done only on numerics (including numeric subfields, numeric arrays, numeric array elements, numeric table elements, numeric named constants, numeric figurative constants, and numeric literals).
- Decimal alignment is done for all arithmetic operations. Even though truncation can occur, the position of the decimal point in the result field is not affected.
- The length of any field specified in an arithmetic operation cannot exceed 30 digits. If the result exceeds 30 digits, digits are dropped from either or both ends, depending on the location of the decimal point.
- Digits are dropped from either or both ends, depending on the location of the decimal point.
- The TRUNCNBR option (on the CRTBNDRPG and CRTRPGMOD commands) determines whether truncation on the left occurs with numeric overflow or a runtime error is generated.
- An arithmetic operation does not change factor 1 and factor 2 unless they are the same as the result field.
- The result of an arithmetic operation replaces the data that was in the result field.

- If you use conditioning indicators with DIV and MVR, it is your responsibility to ensure that the DIV operation occurs immediately before the MVR operation. If conditioning indicators on DIV cause the MVR operation to be executed when the immediately preceding DIV was not executed, then undesirable results may occur.
- Half-adjusting is done by adding 5 (-5 if the field is negative) one position to the right of the last specified decimal position in the result field. The half adjust entry is allowed only with arithmetic operations, but not with an MVR operation or with a DIV operation followed by the MVR operation. Half adjust only affects the result if the number of decimal positions in the calculated result is greater than the number of decimal positions in the result field. Half adjusting occurs after the operation but before the result is placed in the result field. Resulting indicators are set according to the value of the result field after half-adjusting has been done.

For arithmetic operations in which all three fields are used:

- Factor 1, factor 2, and the result field can be three different fields.
- Factor 1, factor 2, and the result field can all be the same field.
- Factor 1 and factor 2 can be the same field but different from the result field.
- Either factor 1 or factor 2 can be the same as the result field.

For the ADD, SUB, MULT, and DIV operations, factor 1 is not required. If factor 1 is not specified, the operation is done as though factor 1 and the result field were the same field.

For information on using arrays with arithmetic operations, see “Specifying an Array in Calculations” on page 145.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments
C*
C*   In the following example, the initial field values are:
C*
C*                         A = 1
C*                         B = 10.0
C*                         C = 32
C*                         D = -20
C*                         E = 6
C*                         F = 10.0
C*                         G = 2.77
C*                         H = 70
C*                         J = .6
C*                         K = 25
C*                         L = 1.0, 1.7, -1.1                              Result:
C*
C                   ADD       1             A                 3 0          A = 002
C     B             ADD       C             V                 5 2          V = 042.00
C     B             ADD       D             V                              V = -10.00
C                   Z-ADD     C             V                              V = 032.00
C                   SUB       1             E                 3 0          E = 005
C     C             SUB       B             W                 5 1          W = 0022.0
C     C             SUB       D             W                              W = 0052.0
C                   Z-SUB     C             W                              W = -0032.0
C                   MULT      E             F                 3 0          F = 060
C     B             MULT      G             X                 8 4          X = 0027.7000
C     B             MULT      D             X                              X = -200.0000
C                   DIV       B             H                 3 0          H = 007
C     C             DIV       J             Y                 6 2          Y = 0053.33
C                   MVR                     Z                 5 3          Z = 00.002
C                   SQRT      K             Z                              Z = 05.000
C                   XFOOT     L             Z                              Z = 01.600
```

Figure 100. Summary of Arithmetic Operations

Array Operations

The array operations are:

- "LOOKUP (Look Up a Table or Array Element)" on page 385
- "MOVEA (Move Array)" on page 399
- "SORTA (Sort an Array)" on page 461
- "XFOOT (Summing the Elements of an Array)" on page 491.

While many operations work with arrays, these operations perform specific array functions. See each operation for an explanation of its function.

Bit Operations

The bit operations are:

- "BITOFF (Set Bits Off)" on page 313
- "BITON (Set Bits On)" on page 314
- "TESTB (Test Bit)" on page 474.

The BITOFF and BITON operations allow you to turn off and on specific bits in a field specified in the result field. The result field must be a one-position character field.

The TESTB operation compares the bits identified in factor 2 with the corresponding bits in the field named as the result field.

The bits in a byte are numbered from left to right. The left most bit is bit number 0. In these operations, factor 2 specifies the bit pattern (bit numbers) and the result field specifies a one-byte character field on which the operation is performed. To specify the bit numbers in factor 2, a 1-byte hexadecimal literal or a 1-byte character field is allowed. The bit numbers are indicated by the bits that are turned on in the literal or the field. Alternatively, a character literal which contains the bit numbers can also be specified in factor 2.

Branching Operations

The branching operations are:

- “CABxx (Compare and Branch)” on page 316
- “GOTO (Go To)” on page 373
- “ITER (Iterate)” on page 379
- “LEAVE (Leave a Do Group)” on page 384
- “TAG (Tag)” on page 471.

The GOTO operation (when used with a TAG operation) allows branching. When a GOTO operation occurs, the program branches to the specified label. The label can be specified before or after the GOTO operation. The label is specified by the TAG or ENDSR operation.

The TAG operation names the label that identifies the destination of a GOTO or CABxx operation.

The ITER operation transfers control from within a DO-group to the ENDDO statement of the DO-group.

The LEAVE operation is similar to the ITER operation; however, LEAVE transfers control to the statement ***following*** the ENDDO operation.

See each operation for an explanation of its function.

Call Operations

The call operations are:

- “CALL (Call a Program)” on page 318
- “CALLB (Call a Bound Procedure)” on page 321
- “PARM (Identify Parameters)” on page 424
- “PLIST (Identify a Parameter List)” on page 426
- “RETURN (Return to Caller)” on page 444.

The CALL and CALLB operations allow an RPG IV procedure to transfer control to other programs or procedures. The RETURN operation transfers control back to the calling program or procedure. The PLIST and PARM operations can be used to allow the calling and called programs or procedures to address the same data.

See each operation for an explanation of its function.

Compare Operations

The compare operations are:

- "ANDxx (And)" on page 311
- "COMP (Compare)" on page 341
- "CABxx (Compare and Branch)" on page 316
- "CASxx (Conditionally Invoke Subroutine)" on page 322
- "DOU (Do Until)" on page 349
- "DOUxx (Do Until)" on page 350
- "DOW (Do While)" on page 352
- "DOWxx (Do While)" on page 353
- "IF (If)" on page 374
- "IFxx (If)" on page 375
- "ORxx (Or)" on page 420
- "WHEN (When True Then Select)" on page 485
- "WHENxx (When True Then Select)" on page 486

In the ANDxx, CABxx, CASxx, DOUxx, DOWxx, IFxx, ORxx, and WHENxx operations, xx can be:

xx	**Meaning**
GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 is not equal to factor 2.
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.
Blanks	Unconditional processing (CASxx or CABxx).

The compare operations test fields for the conditions specified in the operations. These operations do not change the values of the fields. For COMP, CABXX, and CASXX, the resulting indicators assigned in postions 71 and 76 are set according to the results of the operation. All data types may be compared to fields of the same data type.

Remember the following when using the compare operations:

- If numeric fields are compared, fields of unequal length are aligned at the implied decimal point. The fields are filled with zeros to the left and/or right of the decimal point making the field lengths and number of decimal positions equal for comparison.

- All numeric comparisons are algebraic. A plus (+) value is always greater than a minus (-) value.

- Blanks within zoned numeric fields are assumed to be zeros, if the FIXNBR(*ZONED) compiler option is used in the compilation of the program.

- All graphic comparisons are done using the hexadecimal representation of the graphic characters. The alternate sequence is not used.

- If character or graphic fields are compared, fields of unequal length are aligned to their leftmost character. The shorter field is filled with blanks to equal the length of the longer field so that the field lengths are equal for comparison.

- If an alternate collating sequence (using the “ALTSEQ{(*NONE *SRC *EXT)}” keyword on the Control specification) has been specified for the comparison of character fields, the comparands are converted to the alternate sequence and then compared. If *HIVAL and *LOVAL are used to set up the comparison, the alternate collating sequence may alter the value before the compare operation.
- Date fields are converted to a common format when being compared.
- Time fields are converted to a common format when being compared.
- When basing pointer fields are compared for anything except equality or inequality, the results will be unpredictable unless the pointers point to addresses within contiguous storage (for example, all point to positions within the same data structure, array, or standalone field).
- When procedure pointer fields are compared for anything except equality or inequality, the results will be unpredictable.
- An array name cannot be specified in a compare operation, but an array element may be specified.
- The ANDxx and ORxx operations can be used following DOUxx, DOWxx, IFxx, and WHENxx.

Data-Area Operations

The data-area operations are:

- “IN (Retrieve a Data Area)” on page 377
- “OUT (Write a Data Area)” on page 423
- “UNLOCK (Unlock a Data Area or Release a Record)” on page 481.

The IN and OUT operations allow you to retrieve and write one or all data areas in a program, depending on the factor 2 entry.

The IN and OUT operations also allow you to control the locking or unlocking of a data area. When a data area is locked, it can be read but not updated by other programs or procedures.

The following lock states are used:

- For an IN operation with *LOCK specified, an exclusive allow read lock state is placed on the data area.
- For an OUT operation with *LOCK the data area remains locked after the write operation
- For an OUT operation with blank the data area is unlocked after it is updated
- UNLOCK is used to unlock data areas and release record locks, the data areas and/or records are not updated.

During the actual transfer of data into or out of a data area, there is a system-internal lock on the data area. If several users are contending for the same data area, a user may get an error message indicating that the data area is not available.

Remember the following when using the IN, OUT, and UNLOCK operations:

- A data-area operation cannot be done on a data area that is not defined to the operating system.

- Before the IN, OUT, and UNLOCK operations can be done on a data area, you must specify the DTAARA keyword on the definition specification for the data area, or specify the data area in the result field of an *DTAARA DEFINE statement. (For further information on the DEFINE statement, see "DEFINE (Field Definition)" on page 342.)
- A locked data area cannot be updated or locked by another RPG program; however, the data area can be retrieved by an IN operation with factor 1 blank.
- A data-area name cannot be the name of a multiple-occurrence data structure, an input record field, an array, an array element, or a table.
- A data area cannot be the subfield of a multiple occurrence data structure, a data-area data structure, a program-status data structure, a file-information data structure (INFDS), or a data structure that appears on an *DTAARA DEFINE statement.

A data structure defined with a U in position 23 of the definition specifications indicates that the data structure is a data area. You may specify the DTAARA keyword for a data area data structure, if specified you can use the IN, OUT and UNLOCK operation codes to specify further operations for the data area. The data area is automatically read and locked at program initialization time, and the contents of the data structure are written to the data area when the program ends with LR on.

To define the local data area (*LDA) you may specify the DTAARA(*LDA) keyword on the definition specification for the data area, or specify *LDA in factor 2 of a *DTAARA DEFINE statement.

To define the *PDA you may specify the DTAARA(*PDA) keyword on the definition specification for the data area, or specify *PDA in factor 2 of a *DTAARA DEFINE statement.

Date Operations

Date operations allow you to perform date and time arithmetic, extract portions of a date, time or timestamp field; or test for valid fields. They operate on Date, Time, and Timestamp fields, and character and numeric fields representing dates, times and timestamps. The date operations are:

- "ADDDUR (Add Duration)" on page 309
- "EXTRCT (Extract Date/Time/Timestamp)" on page 370
- "SUBDUR (Subtract Duration)" on page 465
- "TEST (Test Date/Time/Timestamp)" on page 472

With "ADDDUR (Add Duration)" you can add a duration to a date or time. With "SUBDUR (Subtract Duration)" you can subtract a duration from a date or time, or calculate the duration between 2 dates, times or timestamps. With "EXTRCT (Extract Date/Time/Timestamp)" you can extract part of a date, time or timestamp. With "TEST (Test Date/Time/Timestamp)" you can test for a valid date, time, or timestamp field. The valid duration codes (and their short forms) are:

- *YEARS for the year (*Y)
- *MONTHS for the month (*M)
- *DAYS for the day (*D)
- *HOURS for the hours (*H)
- *MINUTES for the minutes (*MN)
- *SECONDS for the seconds (*S)
- *MSECONDS for the microseconds (*MS).

Declarative Operations

The declarative operations do not cause an action to occur (except PARM with optional factor 1 or 2); they can be specified anywhere within calculations. They are used to declare the properties of fields or to mark parts of a program. The control level entry (positions 7 and 8) can be blank or can contain an entry to group the statements within the appropriate section of the program. The declarative operations are:

- "DEFINE (Field Definition)" on page 342
- "KFLD (Define Parts of a Key)" on page 381
- "KLIST (Define a Composite Key)" on page 382
- "PARM (Identify Parameters)" on page 424
- "PLIST (Identify a Parameter List)" on page 426
- "TAG (Tag)" on page 471.

The DEFINE operation either defines a field based on the attributes (length and decimal positions) of another field or defines a field as a data area.

The KLIST and KFLD operations are used to indicate the name by which a composite key field may be referred and the fields that compose the composite key. A *composite key* is a key that contains a list of key fields. It is built from left to right, with the first KFLD specified being the leftmost (high-order) field of the composite key.

The PLIST and PARM operations are used with the CALL and CALLB operations to allow a called program or procedure access to parameters from a calling program or procedure.

The TAG operation names the destination of a branching operation such as GOTO or CABxx.

Operations Using Expressions

The operations using expressions are:

- "DOU (Do Until)" on page 349
- "DOW (Do While)" on page 352
- "EVAL (Evaluate expression)" on page 362
- "IF (If)" on page 374
- "WHEN (When True Then Select)" on page 485

The IF, DOU, DOW and WHEN operations are analogous to the IFxx, DOUxx, DOWxx, and WHENxx operations. The EVAL operation is a form of assignment. In all these cases, what distinguishes the operation is the use of free form expressions in the extended factor 2 field.

File Operations

The file operation codes are:

- "ACQ (Acquire)" on page 307
- "CHAIN (Random Retrieval from a File)" on page 327
- "CLOSE (Close Files)" on page 339
- "COMMIT (Commit)" on page 340

- "DELETE (Delete Record)" on page 345
- "EXCEPT (Calculation Time Output)" on page 364
- "EXFMT (Write/Then Read Format)" on page 366
- "FEOD (Force End of Data)" on page 371
- "FORCE (Force a Certain File to Be Read Next Cycle)" on page 372
- "NEXT (Next)" on page 413
- "OPEN (Open File for Processing)" on page 418
- "POST (Post)" on page 428
- "READ (Read a Record)" on page 430
- "READC (Read Next Changed Record)" on page 432
- "READE (Read Equal Key)" on page 434
- "READP (Read Prior Record)" on page 436
- "READPE (Read Prior Equal)" on page 438
- "REL (Release)" on page 440
- "ROLBK (Roll Back)" on page 445
- "SETGT (Set Greater Than)" on page 451
- "SETLL (Set Lower Limit)" on page 455
- "UNLOCK (Unlock a Data Area or Release a Record)" on page 481
- "UPDATE (Modify Existing Record)" on page 483
- "WRITE (Create New Records)" on page 489.

Most file operations can be used with both program described and externally described files (F or E respectively in position 22 of the file description specifications).

When an externally described file is used with certain file operations, a record format name, rather than a file name, can be specified in factor 2. Thus, the processing operation code retrieves and/or positions the file at a record format of the specified type according to the rules of the calculation operation code used.

When the OVRDBF (override with data base file) command is used with the MBR (*ALL) parameter specified, the SETLL, SETGT and CHAIN operations only process the current open file member. For more information, refer to the *Programming: Data Base Guide*.

The WRITE and UPDATE operations that specify a program described file name in factor 2 *must* have a data structure name specified in the result field. The CHAIN, READ, READE, READP, and READPE operations that specify a program described file name in factor 2 *may* have a data structure name specified in the result field. With the CHAIN, READ, READE, READP, and READPE operations, data is transferred directly between the file and the data structure, without processing the input specifications for the file. Thus, no record identifying or field indicators are set on as a result of an input operation to a data structure. If all input and output operations to the file have a data structure specified in the result field, input and output specifications are not required.

If an input operation (CHAIN, EXFMT, READ, READC, READE, READP, READPE) does not retrieve a record because no record was found, because an error occurred in the operation, or because the last record was already retrieved (end of file), then no data is extracted and all fields in the program remain unchanged.

If you specify N as the operation extender of a CHAIN, READ, READE, READP, or READPE operation for an update disk file, a record is read without locking. If no operation extender is specified, the record is locked if the file is an update disk file.

Exception/errors that occur during file operations can be handled by the programmer (by coding an error indicator or specifying a file-error subroutine), or by the RPG IV error handler.

Indicator-Setting Operations

The indicator setting operations are

- "SETOFF (Set Indicator Off)" on page 458
- "SETON (Set Indicator On)" on page 459

The "SETON (Set Indicator On)" and "SETOFF (Set Indicator Off)" operations set (on or off) indicators specified in positions 71 through 76. At least one resulting indicator must be specified in these positions. Remember the following when setting indicators:

- The 1P, MR, KA through KN, and KP through KY indicators cannot be set on by the SETON operation.
- The 1P and MR indicators cannot be set off by the SETOFF operation.
- Setting L1 through L9 on or off with a SETON or SETOFF operation does not set any lower control level indicators.

Information Operations

The information operations are:

- "DUMP (Program Dump)" on page 358
- "SHTDN (Shut Down)" on page 460
- "TIME (Time of Day)" on page 479.

The DUMP operation provides a dump of all indicators, fields, data structures, arrays, and tables used in a program.

The SHTDN operation allows the program to determine whether the system operator has requested shutdown. If so, the resulting indicator that must be specified in positions 71 and 72 is set on.

The TIME operation allows the program to access the system time of day and system date at any time during program running.

Initialization Operations

The initialization operations provide run-time clearing and resetting of all elements in a structure (record format, data structure, array, or table) or a variable (field, subfield, or indicator).

The initialization operations are:

- "CLEAR (Clear)" on page 335
- "RESET (Reset)" on page 441.

The CLEAR operation sets all elements in a structure or variable to their default value depending on the field type (numeric, character, graphic, indicator, pointer, or date/time/timestamp).

The RESET operation sets all elements in a structure or variable to their initial values (the values they had at the end of the initialization step in the program cycle).

The RESET operation is used with data structure initialization and the initialization subroutine (*INZSR). You can use both data structure initialization and the *INZSR to set the initial value of a variable. The initial value will be used to set the variable if it appears in the result field of a RESET operation.

When these operation codes are applied to record formats, only fields which are output are affected (if factor 2 is blank) or all fields (if factor 2 is *ALL). The factor 1 entry of *NOKEY prevents key fields from being cleared or reset.

*ALL may be specified in factor 2 if the result field contains a table name, or multiple occurrence data structure or record format. If *ALL is specified all elements or occurrences will be cleared or reset. See "CLEAR (Clear)" on page 335 and "RESET (Reset)" on page 441 for more detail.

For more information see Chapter 7, "Data Types and Data Formats" on page 101.

Message Operation

The message operation

- "DSPLY (Display Function)" on page 355

allows interactive communication between the program and the operator or between the program and the display workstation that requested the program.

Move Operations

Move operations transfer all or part of factor 2 to the result field.

The source and target of the move operation can be of the same or different types, but some restrictions apply:

- For pointer moves, source and target must be the same type, either both basing pointers or both procedure pointers.
- When using MOVEA, both the source and target must be of the same type.
- MOVEA is not allowed for Date, Time or Timestamp fields

The move operations are:

- "MOVE (Move)" on page 392
- "MOVEA (Move Array)" on page 399
- "MOVEL (Move Left)" on page 406.

Factor 2 remains unchanged. Resulting indicators can be specified in positions 71 through 76, except for the MOVE and MOVEL operations if the result field is an array, or for the MOVEA operation if the result field is an array or array element.

Moving Character, Graphic, and Numeric Data

When a character field is moved into a numeric result field, the digit portion of each character is converted to its corresponding numeric character and then moved to the result field. Blanks are transferred as zeros. For the MOVE operation, the zone portion of the rightmost character is converted to its corresponding sign and moved to the rightmost position of the numeric result field. It becomes the sign of the field. (See Figure 148 on page 393 for an example.) For the MOVEL operation, the zone portion of the rightmost character of factor 2 is converted and used as the sign of the result field (unless factor 2 is shorter than the result field) whether or not the rightmost character is included in the move operation. (See Figure 153 on page 408 for an example.)

If move operations are specified between numeric fields, the decimal positions specified for the factor 2 field are ignored. For example, if 1.00 is moved into a three-position numeric field with one decimal position, the result is 10.0.

Factor 2 may contain the figurative constants *ZEROS for moves to character or numeric fields. To achieve the same function for graphic fields, the user should code *ALLG'oXXi' (where 'XX' represents graphic zeros).

When moving data from a character source to graphic fields, if the source is a character literal, named constant, or *ALL, the compiler will check to make sure it is entirely enclosed by one pair of shift-out shift-in characters (SO/SI). The compiler also checks that the character source is of even length and at least 4 bytes (SO/SI plus one graphic character). When moving from a hexadecimal literal or *ALLX to graphic field, the first byte and last byte of the hexadecimal literal or the pattern within *ALLX must not be 0E (shift out) and 0F (shift in). But the hexadecimal literal (or pattern) should still represent an even number of bytes.

When a character field is involved in a move from/to a graphic field, the compiler will check that the character field is of even length and at least 4 bytes long. At runtime, the compiler checks the content of the character field to make sure it is entirely enclosed by only one pair of SO/SI.

When moving from a graphic field to a character field, if the length of the character field is greater than the length of the graphic field (in bytes) plus 2 bytes, the SO/SI are added immediately before and after the graphic data. This may cause unbalanced SO/SI in the character field due to residual data in the character field, which will not be diagnosed by the compiler.

When move operations are used to move data from character fields to graphic fields, shift-out and shift-in characters are removed. When moving data from graphic fields to character fields, shift-out and shift-in characters are inserted in the target field.

If you specify operation extender P for a move operation, the result field is padded from the right for MOVEL and MOVEA and from the left for MOVE. The pad characters are blank for character, double-byte blanks for graphic, 0 for numeric, and '0' for indicator. The padding takes place after the operation. If you use MOVE or MOVEL to move a field to an array, each element of the array will be padded. If you use these operations to move an array to an array and the result contains more elements than the factor 2 array, the same padding takes place but the extra elements are not affected. A MOVEA operation with an array name in the result field will pad the last element affected by the operation plus all subsequent elements.

When resulting indicators are specified for move operations, the result field determines which indicator is set on. If the result field is a character field or graphic field, only the resulting indicator in positions 75 and 76 can be specified. This indicator is set on if the result field is all blanks. When the result field is numeric, all three resulting indicator positions may be used. These indicators are set on as follows:

High (71-72)	Set on if the result field is greater than 0.
Low (73-74)	Set on if the result field is less than 0.
Equal (75-76)	Set on if the result field is equal to 0.

Moving Date-Time Data

The MOVE and MOVEL operation codes can be used to move Date, Time and Timestamp data type fields.

The following combinations are allowed for the MOVE and MOVEL operation codes:

- Date to Date
- Time to Time
- Timestamp to Timestamp
- Date to Timestamp
- Time to Timestamp (sets micro-seconds to 000000)
- Timestamp to Date
- Timestamp to Time
- Date to Character or Numeric
- Time to Character or Numeric
- Timestamp to Character or Numeric
- Character or Numeric to Date
- Character or Numeric to Time
- Character or Numeric to Timestamp

Factor 1 is optional for MOVE and MOVEL and is used to specify the format of the character or numeric field if it is the source or target of the operation. Any valid format may be specified. If factor 1 is blank, the format of the Date-Time field is used.

Time format *USA is not allowed for movement between Time and numeric classes.

A special value (*JOBRUN) can be specified in factor 1 if the format of the source or target character or numeric field is based on the run-time job attributes (DATFMT, DATSEP and TIMSEP). If *JOBRUN is used for formatting a Time field and the Result field is character, the resulting format will be HHMMSS using the TIMSEP from the job. If the field specified in Factor 2 and the result field both are of class Date, Time or Timestamp, the entry in Factor 1 must be blank.

A two digit year format (*MDY, *DMY, *YMD, and *JUL) can only represent dates in the range 1940 through 2039. An error will be issued if conversion to a two-digit year format is requested for dates outside this range.

Factor 2 is required and may contain a character or numeric, Date, Time or Timestamp field; array; array element; a literal; or a named constant, to be converted.

If the field specified in factor 2 is character, it must have separator characters, for example 09/06/91.

Separator characters must be valid for the specified format.

If factor 2 is not a valid representation of a date or time or its format does not match the format specified in factor 1, an error is generated.

The result field must be a Date, Time or Timestamp field or a numeric or character class field, array or array element. The date or time is placed in the result field according to its defined format or the format code specified in factor 1.

When moving from a Date to a Timestamp field, the time and microsecond portion of the timestamp are unaffected, however the entire timestamp is checked and an error will be generated if it is not valid.

When moving from a Time to a Timestamp field, the microseconds part of the timestamp is set to 000000. The date portion remains unaffected, but the entire timestamp will be checked and an error will be generated when it is not valid.

The P operation extender can only be specified if the result field is character or numeric.

If the result field is numeric, separator characters will be removed, prior to the operation. The length used is the length after removing the separator characters.

Example of Converting a Character Field to a Date Field

The following example shows how to convert from a character field in the form CYYMMDD to a date field in *ISO format. This is particularly useful when using command parameters of type *DATE.

```
             CMD        PROMPT('Use DATE parameter')
             PARM       KWD(DATE) TYPE(*DATE)
```

Figure 101. Source for a command using a date parameter.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*----------------------------------------------------------
D* Use a data structure for the input parameter DateParm.
D* The first byte is either '0' or '1' to indicate the
D* century.  Bytes 2-7 are the YYMMDD part of the input
D* date.
D*----------------------------------------------------------
D DateParm        DS
D   YYMMDD                    2      7S 0
D*----------------------------------------------------------
D* Declare a date type with date format *ISO.
D*----------------------------------------------------------
D ISO_date        S               D   DATFMT(*ISO)
D*----------------------------------------------------------
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C     *ENTRY        PLIST
C                   PARM                    DateParm
C*----------------------------------------------------------
C* The format of the DATE parameter is CYYMMDD, so the
C* last 6 digits are YYMMDD.  Use MOVE to convert the
C* date from the YYMMDD zoned value to the *ISO date.
C*----------------------------------------------------------
C     *YMD          MOVE      YYMMDD        ISO_Date
```

Figure 102. Part of RPG IV command processing program for this command.

Move Zone Operations

The move zone operations are:

- “MHHZO (Move High to High Zone)” on page 388
- “MHLZO (Move High to Low Zone)” on page 389
- “MLHZO (Move Low to High Zone)” on page 390
- “MLLZO (Move Low to Low Zone)” on page 391.

The move zone operations move only the zone portion of a character.

Characters J through R have D zones and can be used to obtain a negative value:

Whenever the word *high* is used in a move zone operation, the field involved must be a character field; whenever *low* is used, the field involved can be either a character or a numeric field.

`(J = hexadecimal D1, ..., R = hexadecimal D9).`

Note: While you may see this usage in old programs, your code will be clearer if you use hexadecimal literals for this purpose. Use X'F0' to obtain a positive zone and X'D0' to obtain a negative zone.

Note: The character (-) is represented by a hexadecimal 60, and cannot be used to obtain a negative result, since it has a zone of 6, and a negative result requires a zone of "D".

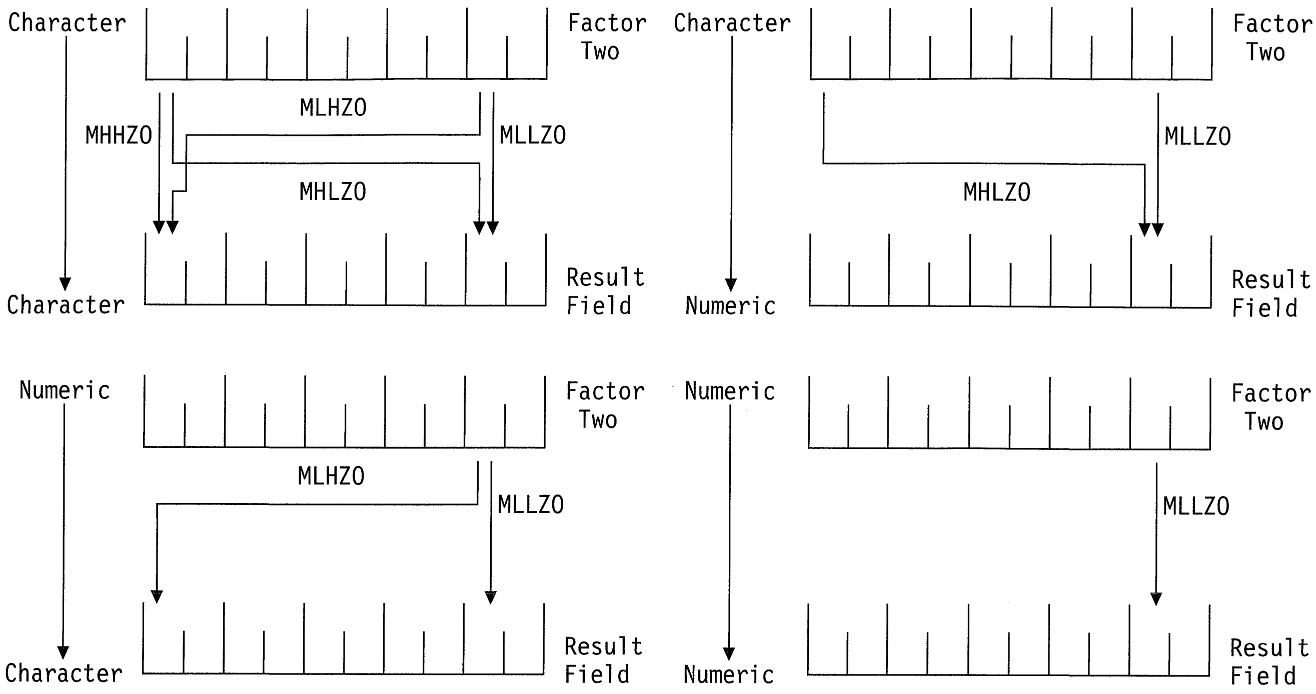


Figure 103. Function of MOVE Zone Operations

String Operations

The string operations include concatenation, scanning, substringing, translation, and verification. String operations can only be used on character or graphic fields.

The string operations are:

- "CAT (Concatenate Two Strings)" on page 324
- "CHECK (Check Characters)" on page 330
- "CHECKR (Check Reverse)" on page 333
- "SCAN (Scan String)" on page 446
- "SUBST (Substring)" on page 468
- "XLATE (Translate)" on page 492.

The CAT operation concatenates two strings to form one.

The CHECK and CHECKR operations verify that each character or graphic character in factor 2 is among the valid characters in factor 1. CHECK verifies from left to right and CHECKR from right to left.

The SCAN operation scans the base string in factor 2 for occurrences of another string specified in factor 1.

The SUBST operation extracts a specified string from a base string in factor 2. The extracted string is placed in the result field.

The XLATE operation translates characters in factor 2 according to the FROM and TO strings in factor 1.

Note: Figurative constants cannot be used in the factor 1, factor 2, or result fields. No overlapping in a data structure is allowed for factor 1 and the result field, or factor 2 and the result field.

In the string operations, factor 1 and factor 2 may have two parts. If both parts are specified, they must be separated by a colon. This option applies to all but the CAT, CHECK, CHECKR, and SUBST operations (where it applies only to factor 2).

If you specify P as the operation extender for the CAT, SUBST, or XLATE operations, the result field is padded from the right with blanks after the operation.

See each operation for a more detailed explanation.

When using string operations on graphic fields, all data in factor 1, factor 2 and result field must be graphic. When numeric values are specified for length, start position, and number of blanks for graphic characters, the values represent double byte characters.

When using string operations on the graphic part of character data, the start position, length and number of blanks represent single byte characters. Preserving data integrity is the user's responsibility.

Structured Programming Operations

The structured programming operations are:

- "ANDxx (And)" on page 311
- "DO (Do)" on page 347
- "DOUxx (Do Until)" on page 350
- "DOWxx (Do While)" on page 353
- "ELSE (Else)" on page 359
- "ENDyy (End a Structured Group)" on page 360
- "IFxx (If)" on page 375
- "ITER (Iterate)" on page 379
- "LEAVE (Leave a Do Group)" on page 384
- "ORxx (Or)" on page 420
- "OTHER (Otherwise Select)" on page 421
- "SELECT (Begin a Select Group)" on page 449
- "WHENxx (When True Then Select)" on page 486.

The DO operation allows the processing of a group of calculations zero or more times starting with the value in factor 1, incrementing each time by a value on the associated ENDDO operation until the limit specified in factor 2 is reached.

The DOUxx operation allows the processing of a group of calculations one or more times based on the results of comparing factor 1 and factor 2. The end of a DOUxx operation is indicated by an ENDDO operation.

The DOWxx operation allows the processing of a group of calculations zero or more times based on the results of comparing factor 1 and factor 2. The end of a DOWxx operation is indicated by an ENDDO operation.

The LEAVE operation interrupts control flow prematurely and transfers control to the statement following the ENDDO operation of an iterative structured group. The ITER operation causes the next loop iteration to occur immediately.

An IFxx operation allows the processing of a group of calculations based on the results of comparing factor 1 and factor 2. The ELSE operation allows you to specify a group of calculations to be processed if the IFxx condition is not satisfied. The end of an IFxx group is indicated by ENDIF.

The SELECT, WHENxx, and OTHER group of operations are used to conditionally process one of several alternative sequences of operations. The beginning of the select group is indicated by the SELECT operation. The WHENxx operations are used to choose the operation sequence to process. The OTHER operation is used to indicate an operation sequence that is processed when none of the WHENxx conditions are fulfilled. The end of the select group is indicated by the ENDSL operation.

The ANDxx and ORxx operations are used with the DOUxx, DOWxx, WHENxx, and IFxx operations to specify a more complex condition than the comparison of a single factor 1 and factor 2 pair. The ANDxx operation has higher precedence than the ORxx operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* In the following example, indicator 25 will be set on only if the
C* first two conditions are true or the third condition is true.
C*
C* As an expression, this would be written:
C* EVAL *IN25 = ((FIELDA > FIELDB) AND (FIELDA >= FIELDC)) OR (FIELDA < FIELDD)
C*
C*
C     FIELDA        IFGT      FIELDB
C     FIELDA        ANDGE     FIELDC
C     FIELDA        ORLT      FIELDD
C                   SETON                                        25
C                   ELSE
C                   SETOFF                                       25
C                   ENDIF
```

Figure 104. Example of AND/OR Precedence

A DO, DOUxx, DOWxx, IFxx, or SELECT operation (with or without ANDxx or ORxx operations), and an ENDyy operation, delimit a structured group. The ENDDO operation ends each DO, DOUxx, and DOWxx group or causes the structured group to be reprocessed until the specified ending conditions are met. The SELECT must end with an ENDSL. An IFxx operation and an IFxx operation with an ELSE operation must end with an ENDIF operation. Using END gives you the same results as using ENDIF, ENDSL, or ENDDO.

The rules for making the comparison on the ANDxx, DOUxx, DOWxx, IFxx, ORxx and WHENxx operation codes are the same as those given under "Compare Operations" on page 291.

In the ANDxx, DOUxx, DOWxx, IFxx, ORxx, and WHENxx operations, xx can be:

xx	**Meaning**
GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 is not equal to factor 2.

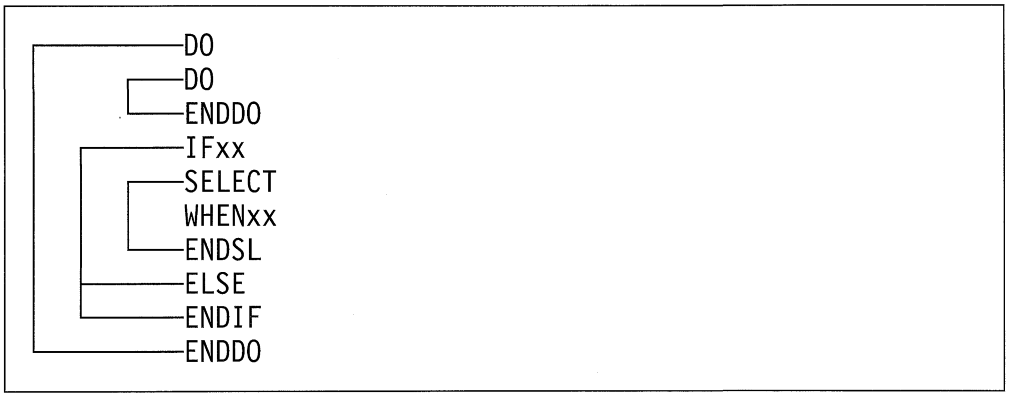
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.

In the ENDyy operation, yy can be:

yy	**Meaning**
CS	End for CASxx operation.
DO	End for DO, DOUxx, and DOWxx operation.
IF	End for IFxx operation.
SL	End for SELECT operation.
Blanks	End for any structured operation.

Note: The yy in the ENDyy operation is optional.

If a structured group, in this case a do group, contains another complete structured group, together they form a nested structured group. Structured groups can be nested to a maximum depth of 100 levels. The following is an example of nested structured groups, three levels deep:



```
DO
DO
ENDDO
IFxx
SELECT
WHENxx
ENDSL
ELSE
ENDIF
ENDDO
```

Remember the following when specifying structured groups:

- Each nested structured group must be completely contained within the outer level structured group.
- Each structured group must contain one of a DO, DOUxx, DOWxx, IFxx, or SELECT operation and its associated ENDyy operation.
- A structured group can be contained in detail, total, or subroutine calculations, but it cannot be split among them.
- Branching into a structured group from outside the structured group may cause undesirable results.

Subroutine Operations

An RPG IV subroutine is a group of calculation specifications in a program that can be processed several times in that program. The RPG IV subroutine operations are:

- “BEGSR (Beginning of Subroutine)” on page 312
- “ENDSR (End of Subroutine)” on page 361
- “EXSR (Invoke Subroutine)” on page 367
- “CASxx (Conditionally Invoke Subroutine)” on page 322.

RPG IV subroutine specifications must follow all other calculation operations that can be processed for a program; however, the PLIST, PARM, KLIST, KFLD, and DEFINE operations may be specified between an ENDSR operation (the end of one subroutine) and a BEGSR operation (the beginning of another subroutine) or after all subroutines. A subroutine can be called using an EXSR or CASxx operation anywhere in the calculation specifications. Subroutine lines can be identified by SR in positions 7 and 8. The only valid entries in positions 7 and 8 of a subroutine line are SR, AN, OR, or blanks.

For information on how to code a subroutine, see "Coding Subroutines" on page 368.

Test Operations

The test operations are:

- "TEST (Test Date/Time/Timestamp)" on page 472
- "TESTB (Test Bit)" on page 474
- "TESTN (Test Numeric)" on page 476
- "TESTZ (Test Zone)" on page 478.

The TESTx operations allow you to test fields specified in the result field. TEST tests for valid date, time, or timestamp data. TESTB tests the bit pattern of a result field. TESTN tests if the character field specified in the result field contain all numbers, or numbers with leading blanks, or all blanks. TESTZ tests the zone portion of the leftmost character of a character field specified in the result field. The result of these operations is indicated by the resulting indicators.

Chapter 23. Operation Codes Detail

This chapter describes, in alphabetical order, each operation code.

ACQ (Acquire)

Code	Factor 1	Factor 2	Result Field	Indicators		
ACQ	Device name	WORKSTN file		_	ER	_

The ACQ operation acquires the program device specified in factor 1 for the WORKSTN file specified in factor 2. If the device is available, ACQ attaches it to the file. If it is not available or is already attached to the file, an error occurs. If an indicator is specified in positions 73 and 74, the indicator is set on. If no indicator is specified, but the INFSR subroutine is specified, the INFSR receives control when an error/exception occurs. If no indicator or INFSR subroutine is specified, the default error/exception handler receives control when an error/exception occurs.

No input or output operation occurs when the ACQ operation is processed. ACQ may be used with a multiple device file or, for error recovery purposes, with a single device file. One program may acquire and have the device available to any called program which shares the file and allow the called program to release the device. See the section on "Multiple-Device Files" in the chapter about using WORKSTN files in the *ILE RPG/400 Programmer's Guide*.

ADD (Add)

Code	Factor 1	Factor 2	Result Field	Indicators		
ADD (H)	Addend	Addend	Sum	+	–	Z

If factor 1 is specified, the ADD operation adds it to factor 2 and places the sum in the result field. If factor 1 is not specified, the contents of factor 2 are added to the result field and the sum is placed in the result field. Factor 1 and factor 2 must be numeric and can contain one of: an array, array element, constant, field name, literal, subfield, or table name. For the rules for specifying an ADD operation, see "Arithmetic Operations" on page 287.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The value 1 is added to RECNO.
C                   ADD       1             RECNO
C* The contents of EHWRK are added to CURHRS.
C                   ADD       EHWRK         CURHRS
C* The contents of OVRTM and REGHRS are added together and
C* placed in TOTPAY.
C     OVRTM         ADD       REGHRS        TOTPAY
```

Figure 105. ADD Operations

ADDDUR (Add Duration)

Code	Factor 1	Factor 2	Result Field	Indicators		
ADDDUR	Date/Time	Duration:Duration Code	Date/Time	_	ER	_

The ADDDUR operation adds the duration specified in factor 2 to a date or time and places the resulting Date, Time or Timestamp in the result field.

Factor 1 is optional and may contain a Date, Time or Timestamp field, subfield, array, array element, literal or constant. If factor 1 contains a field name, array or array element then its data type must be the same data type as the field specified in the result field. If factor 1 is not specified the duration is added to the field specified in the result field.

Factor 2 is required and contains two subfactors. The first is a duration and may be a numeric field, array element or constant with zero decimal positions. If the duration is negative then it is subtracted from the date. The second subfactor must be a valid duration code indicating the type of duration.

The duration code must be consistent with the result field data type. You can add a year, month or day duration but not a minute duration to a date field).

The result field must be a date, time or timestamp data type field, array or array element. If Factor 1 is blank, the duration is added to the value in the result field. If the result field is an array, the value in factor 2 is added to each element of the array. If the result field is a time field, the result must always be a valid Time, (if the calculated result is >= 24:00:00, subtract 24 hours or a multiple of 24 hours until the time is valid). If the result is a Timestamp field, and the *MS or time portion is operated on, the date must be adjusted, each time the 24:00:00 boundary is crossed.

If the value of the Date, Time or Timestamp field in factor 1 is invalid, if factor 1 is blank and the value of the result field before the operation is invalid or if the result of the operation is invalid an error (status code 112) is signalled and, the error indicator (columns 73-74), if specified, will be set on, and the value of the result field will remain unchanged.

Note: The system places a 15 digit limit on durations. Adding a Duration with more than 15 significant digits will cause errors or truncation. These problems can be avoided by limiting the first subfactor in Factor 2 to 15 digits.

For more information see “Date Operations” on page 293.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H TIMFMT(*USA) DATFMT(*MDY&)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
DDateconst       C                   CONST(D'12 31 92')
D*
D* Define a Date field and initialize
D*
DLoandate        S                  D   DATFMT(*EUR) INZ(D'12 31 92')
DDuedate         S                  D   DATFMT(*ISO)
Dtimestamp       S                  Z
Danswer          S                  T

CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....

C* Determine a DUEDATE which is xx years, yy months, zz days later
C* than LOANDATE.
C     LOANDATE      ADDDUR    XX:*YEARS     DUEDATE
C                   ADDDUR    YY:*MONTHS    DUEDATE
C                   ADDDUR    ZZ:*DAYS      DUEDATE
C* Determine the date 23 days later
C*
C                   ADDDUR    23:*D         DUEDATE

C* Add a 1234 microseconds to a timestamp
C*
C                   ADDDUR    1234:*MS      timestamp

C* Add 12 HRS and 16 minutes to midnight
C*
C     T'00:00 am'   ADDDUR    12:*Hours     answer
C                   ADDDUR    16:*Minutes   answer

C* Subtract 30 days from a loan due date
C*
C                   ADDDUR    -30:*D        LOANDUE
```

Figure 106. ADDDUR Operations

ANDxx (And)

Code	Factor 1	Factor 2	Result Field	Indicators		
ANDxx	Comparand	Comparand				

This operation must immediately follow a ANDxx, DOUxx, DOWxx, IFxx, ORxx, or WHENxx operation. With ANDxx, you can specify a complex condition for the DOUxx, DOWxx, IFxx, and WHENxx operations. The ANDxx operation has higher precedence than the ORxx operation.

The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry must be the same as the control level entry for the associated DOUxx, DOWxx, IFxx, or WHENxx operation. Conditioning indicator entries (positions 9 through 11) are not permitted.

Factor 1 and factor 2 must contain a literal, a named constant, a figurative constant, a table name, an array element, a data structure name, or a field name. Factor 1 and factor 2 must be of the same type. For example, a character field cannot be compared with a numeric. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See “Compare Operations” on page 291.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* If ACODE is equal to A and indicator 50 is on, the MOVE
C* and WRITE operations are processed.
C     ACODE         IFEQ      'A'
C     *IN50         ANDEQ     *ON
C                   MOVE      'A'           ACREC
C                   WRITE     RCRSN
C* If the previous conditions were not met but ACODE is equal
C* to A, indicator 50 is off, and ACREC is equal to D, the
C* following MOVE operation is processed.
C                   ELSE
C     ACODE         IFEQ      'A'
C     *IN50         ANDEQ     *OFF
C     ACREC         ANDEQ     'D'
C                   MOVE      'A'           ACREC
C                   ENDIF
C                   ENDIF
```

Figure 107. ANDxx Operations

BEGSR (Beginning of Subroutine)

Code	Factor 1	Factor 2	Result Field	Indicators		
BEGSR	Subroutine name					

The BEGSR operation identifies the beginning of an RPG IV subroutine. Factor 1 contains the subroutine name. You may specify the same name in factor 2 of the EXSR operation referring to the subroutine, in the result field of the CASxx operation referring to the subroutine, or in the entry of an INFSR file specification keyword of the subroutine is a file-error subroutine. The control level entry (positions 7 and 8) can be SR or blank. Conditioning indicator entries are not permitted.

Every subroutine must have a unique symbolic name. The keyword *PSSR used in factor 1 specifies that this is a program exception/error subroutine to handle program-detected exception/errors. Only one subroutine can be defined by this keyword. *INZSR in factor 1 specifies a subroutine to be run during the initialization step. Only one subroutine can be defined *INZSR.

See Figure 138 on page 369 for an example of coding subroutines; see “Subroutine Operations” on page 305 for general information on subroutine operations.

BITOFF (Set Bits Off)

Code	Factor 1	Factor 2	Result Field	Indicators		
BITOFF		Bit numbers	Character field			

The BITOFF operation causes bits identified in factor 2 to be set off (set to 0) in the result field. Bits not identified in factor 2 remain unchanged. Therefore, when using BITOFF to format a character, you should use both BITON and BITOFF: BITON to specify the bits to be set on (=1), and BITOFF to specify the bits to be set off (=0). Unless you explicitly set on or off all the bits in the character, you might not get the character you want.

If you want to assign a particular bit pattern to a character field, use the "MOVE (Move)" on page 392 operation with a hexadecimal literal in factor 2.

Factor 2 can contain:

- *Bit numbers 0-7:* From 1 to 8 bits can be set off per operation. They are identified by the numbers 0 through 7. (0 is the leftmost bit.) Enclose the bit numbers in apostrophes, and begin the entry in position 33. For example, to set off bits 0, 2, and 5, enter '025' in factor 2.
- *Field name:* You can specify the name of a one-position character field, table element, or array element in factor 2. The bits that are on in the field, table element, or array element are set off in the result field; bits that are off are not affected.
- *Hexadecimal literal or named constant:* You can specify a 1-byte hexadecimal literal or hexadecimal named constant. Bits that are on in factor 2 are set off in the result field; bits that are off are not affected.
- *Named constant:* A character named constant up to eight positions long containing the bit numbers to be set off.

In the result field, specify a one-position character field. It can be an array element if each element in the array is a one-position character field.

See Figure 108 on page 315 for an example of the BITOFF operation.

BITON (Set Bits On)

Code	Factor 1	Factor 2	Result Field	Indicators		
BITON		Bit numbers	Character field			

The BITON operation causes bits identified in factor 2 to be set on (set to 1) in the result field. Bits not identified in factor 2 remain unchanged. Therefore, when using BITON to format a character, you should use both BITON and BITOFF: BITON to specify the bits to be set on (=1), and BITOFF to specify the bits to be set off (=0). Unless you explicitly set on or off all the bits in the character, you might not get the character you want.

If you want to assign a particular bit pattern to a character field, use the "MOVE (Move)" on page 392 operation with a hexadecimal literal in factor 2.

Factor 2 can contain:

- *Bit numbers 0-7:* From 1 to 8 bits can be set on per operation. They are identified by the numbers 0 through 7. (0 is the leftmost bit.) Enclose the bit numbers in apostrophes, and begin the entry in position 33. For example, to set bits 0, 2, and 5 on, enter '025' in factor 2.
- *Field name:* You can specify the name of a one-position character field, table element, or array element in factor 2. The bits that are on in the field, table element, or array element are set on in the result field; bits that are off are not affected.
- *Hexadecimal literal or named constant:* You can specify a 1-byte hexadecimal literal. Bits that are on in factor 2 are set on in the result field; bits that are off are not affected.
- *Named constant:* A character named constant up to eight positions long containing the bit numbers to be set on.

In the result field, specify a one-position character field. It can be an array element if each element in the array is a one-position character field.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D BITNC           C                   '01234567'
D HEXNC           C                   X'0F'
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The bit settings are:
C*  Before the operations:       After the operations:

C*          FieldA = 00000000    FieldA = 10001111
C*          FieldB = 00000000    FieldB = 00010000
C*          FieldC = 11111111    FieldC = 11111111
C*          FieldD = 11000000    FieldD = 11010000
C*          FieldE = 11000000    FieldE = 11000001
C*          FieldF = 10000001    FieldF = 00000001
C*          FieldG = 11111111    FieldG = 01111111
C*          FieldH = 00000000    FieldH = 00001110
C*          FieldI = 11001010    FieldI = 00001111
C*
C                   BITON     '04567'       FieldA
C                   BITON     '3'           FieldB
C                   BITON     '3'           FieldC
C                   BITON     '3'           FieldD
C                   BITON     FieldE        FieldF
C                   BITON     '01'          FieldH
C                   BITOFF    '0'           FieldG
C                   BITOFF    BITNC         FieldI
C                   BITON     HEXNC         FieldI
```

Figure 108. BITON and BITOFF Operations

CABxx (Compare and Branch)

Code	Factor 1	Factor 2	Result Field	Indicators		
CABxx	Comparand	Comparand	Label	HI	LO	EQ

The CABxx operation compares factor 1 with factor 2. If the condition specified by xx is true, the program branches to the TAG or ENDSR operation associated with the label specified in the result field. Otherwise, the program continues with the next operation in the sequence. If the result field is not specified, the resulting indicators (positions 71-76) are set accordingly, and the program continues with the next operation in the sequence.

You can specify conditioning indicators. Factor 1 and factor 2 must contain a literal, a named constant, a figurative constant, a table name, an array element, a data structure name, or a field name. Factor 1 and factor 2 must be of the same type.

The CABxx operation can specify a branch:

- To a previous or a succeeding specification line
- From a detail calculation line to another detail calculation line
- From a total calculation line to another total calculation line
- From a detail calculation line to a total calculation line
- From a subroutine to a detail calculation line or a total calculation line.

The CABxx operation cannot specify a branch from outside a subroutine to a TAG or ENDSR operation within that subroutine. Branching from one part of the RPG IV logic cycle to another may result in an endless loop. You must ensure that the logic of your program does not produce undesirable results. The label specified in the result field must be associated with a unique TAG operation and must be a unique symbolic name.

Resulting indicators are optional. When specified, they are set to reflect the results of the compare operation. For example, the HI indicator is set when F1>F2, LO is set when F1<F2, and EQ is set when F1=F2.

See “Compare Operations” on page 291 for the rules for comparing factor 1 with factor 2.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*         The field values are:
C*         FieldA = 100.00
C*         FieldB = 105.00
C*         FieldC = ABC
C*         FieldD = ABCDE
C*
C*          Branch to TAGX.
C     FieldA        CABLT     FieldB        TAGX
C*
C*          Branch to TAGX.
C     FieldA        CABLE     FieldB        TAGX
C*
C*          Branch to TAGX; indicator 16 is off.
C     FieldA        CABLE     FieldB        TAGX                      16
C*
C*          Branch to TAGX; indicator 17 is off, indicator 18 is on.
C     FieldA        CAB       FieldB        TAGX                  1718
C*
C*          Branch to TAGX; indicator 19 is on.
C     FieldA        CAB       FieldA        TAGX                       19
C*
C*          No branch occurs.
C     FieldA        CABEQ     FieldB        TAGX
C*
C*          No branch occurs; indicator 20 is on.
C     FieldA        CABEQ     FieldB        TAGX                    20
C*
C*          No branch occurs; indicator 21 is off.
C     FieldC        CABEQ     FieldD        TAGX                  21
C                   :
C     TAGX          TAG
```

Figure 109. CABxx Operations

CALL (Call a Program)

Code	Factor 1	Factor 2	Result Field	Indicators		
CALL		Program name	Plist name	_	ER	LR

The CALL operation passes control to the program specified in factor 2.

Factor 2 must contain a character entry specifying the name of the program to be called. If you specify the library name, it must be immediately followed by a slash and then the program name (for example, 'LIB/PROG'.). Factor 2 must contain the name of a field, a literal, a named constant, or an array element that contains the name of the program to be called and that optionally contains the name of the library in which the program is located. If a library is not specified, the library list is used to find the program.

The total length of a literal, including the slash, cannot exceed 12 characters. The total length of the non-blank data in a field or named constant, including the slash, cannot exceed 21 characters. If either the program or the library name exceeds 10 characters, it is truncated to 10 characters. The program name is used exactly as specified in the literal, field, named constant, or array element to determine the program to be called. Any leading or trailing blanks are ignored. If the first character in the entry is a slash, the library list is used to find the program. If the last character in the entry is a slash, a compile-time message will be issued. (Lowercase characters are not shifted to uppercase. A name enclosed in quotation marks, for example, “ABC”, always includes the quotation marks as part of the name of the program to be called.) *CURLIB is not supported.

Program references are grouped to avoid the overhead of resolving to the target program. All references (using a CALL operation) to a specific program using a named constant or literal are grouped so that the program is resolved to only once, and all subsequent references to that program (by way of named constant or literal only) do not cause a resolve to recur.

The references are grouped if both the program and the library name are identical. All program references by variable name are grouped by the variable name. When a program reference is made with a variable, its current value is compared to the value used on the previous program reference operation that used that variable. If the value did not change, no resolve is done. If it did change, a resolve is done to the new program specified. Note that this rule applies only to references using a variable name. References using a named constant or literal are never re-resolved, and they do not affect whether or not a program reference by variable is re-resolved. Figure 110 on page 319 illustrates the grouping of program references.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D Pgm_Ex_A        C                  'LIB1/PGM1'
D Pgm_Ex_B        C                  'PGM1'
D PGM_Ex_C        C                  'LIB/PGM2'
I*
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C                   CALL      Pgm_Ex_A
C*
C* The following two calls will be grouped together because both
C* have the same program name (PGM1) and the same library name
C* (none).  Note that these will not be grouped with the call using
C* Pgm_Ex_A above because Pgm_Ex_A has a different library
C* name specified (LIB1).
C*
C                   CALL      'PGM1'
C                   CALL      Pgm_Ex_B
C*
C* The following two program references will be grouped together
C* because both have the same program name (PGM2) and the same
C* library name (LIB).
C*
C                   CALL      'LIB/PGM2'
C                   CALL      Pgm_Ex_C
C*
```

Figure 110 (Part 1 of 2). Example of Grouping of Program References

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The first call in the program using CALLV below will result in
C* a resolve being done for the variable CALLV to the program PGM1.
C* This is independent of any calls by a literal or named constant
C* to PGM1 that may have already been done in the program.  The
C* second call using CALLV will not result in a resolve to PGM1
C* because the value of CALLV has not changed.
C*
C                   MOVE      'PGM1'        CALLV         21
C                   CALL      CALLV
C                   CALL      CALLV
```

Figure 110 (Part 2 of 2). Example of Grouping of Program References

In the result field, specify the name of a PLIST to communicate values between the calling program and the called program. The result field can be blank if the called program does not access parameters or if the PARM statements directly follow the CALL operation.

Positions 71 and 72 must be blank. Any valid resulting indicator can be specified in positions 73 and 74 to be set on for an error returned from the called program and in positions 75 and 76 to be set on if the called program is an RPG IV program that returns with the LR indicator on.

The DSPPGMREF command is a CL command that is used to display information about the external references made by a program. Since CALLB is used to call procedures, factor 2 of CALLB operations will not be found in DSPPGMREF; only factor 2 of CALL operations. Using DSPPGMREF, you can query the names of programs called by way of named constants or literals.

Figure 111 illustrates the use of the CALL operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*  The CALL operation calls PROGA and allows PROGA to access
C*  FieldA and FieldB, defined elsewhere. PROGA is run using the content
C*  of FieldA and FieldB.  When PROGA has completed, control
C*  returns to the statement following the last PARM statement.
C*
C*
C                   CALL      'PROGA'
C                   PARM                    FieldA
C                   PARM                    FieldB
```

Figure 111. CALL Operation

CALLB (Call a Bound Procedure)

Code	Factor 1	Factor 2	Result Field	Indicators		
CALLB (D)		Procedure name or procedure pointer	Plist name	_	ER	LR

The CALLB operation is used to call bound procedures written in any of the ILE languages.

Factor 1 must be blank. The operation extender D may be used to include operational descriptors.

Note: Operational descriptors provide the programmer with run-time resolution of the exact attributes of character or graphic strings passed (that is, length and type of string). For more details see the *Programmer's Guide*.

Factor 2 is required and must be a literal or constant containing the name of the procedure to be called, or a procedure pointer containing the address of the procedure to be called. All references must be able to be resolved at bind time. The procedure name provided is case sensitive and may contain more than 10 characters, but no more than 255. If the name is longer than 255, it will be truncated to 255. The result field is optional and may contain a PLIST name.

An indicator specified in positions 73-74 will be set on when the call ends in error.

An indicator specified in positions 75-76 will be set on when the call ends with LR set on.

Note: When the literal or named constant specified on the CALLB starts with "CEE" or an underscore ('_'), the compiler will treat this as a bindable API. For more information on APIs see the SPI reference. To avoid confusion with system provided APIs, you should not name your procedures starting with "CEE".

For a CALLB with a procedure pointer field in factor 2, *ROUTINE in the PSDS will contain the literal '*N'. For a CALLB with a literal or named constant in factor 2, *ROUTINE in the PSDS will contain the first 8 characters of the procedure name.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D* Define a procedure pointer
D
D ProcPtr         S               *   PROCPTR INZ(%PADDR('Create_Space'))
D Extern          S             10
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C* The following call linkage would be STATIC
C                   CALLB     'BOUNDPGM'
C* The following call linkage would be DYNAMIC
C                   CALL      Extern
C* The following call linkage would be STATIC, using a procedure pointer
C                   CALLB     ProcPtr
```

Figure 112. CALLB Example

CASxx (Conditionally Invoke Subroutine)

Code	Factor 1	Factor 2	Result Field	Indicators		
CASxx	Comparand	Comparand	Subroutine name	HI	LO	EQ

The CASxx operation allows you to conditionally select a subroutine for processing. The selection is based on the relationship between factor 1 and factor 2, as specified by xx. If the relationship denoted by xx exists between factor 1 and factor 2, the subroutine specified in the result field is processed.

You can specify conditioning indicators. Factor 1 and factor 2 can contain a literal, a named constant, a figurative constant, a field name, a table name, an array element, a data structure name, or blanks (blanks are valid only if xx is blank and no resulting indicators are specified in positions 71 through 76). If factor 1 and factor 2 are not blanks, both must be of the same data type. In a CASƀƀ operation, factor 1 and factor 2 are required only if resulting indicators are specified in positions 71 through 76.

The result field must contain the name of a valid RPG IV subroutine, including *PSSR, the program exception/error subroutine, and *INZSR, the program initialization subroutine. If the relationship denoted by xx exists between factor 1 and factor 2, the subroutine specified in the result field is processed. If the relationship denoted by xx does not exist, the program continues with the next CASxx operation in the CAS group. A CAS group can contain only CASxx operations. An ENDCS operation must follow the last CASxx operation to denote the end of the CAS group. After the subroutine is processed, the program continues with the next operation to be processed following the ENDCS operation, unless the subroutine passes control to a different operation.

The CASƀƀ operation with no resulting indicators specified in positions 71 through 76 is functionally identical to an EXSR operation, because it causes the unconditional running of the subroutine named in the result field of the CASƀƀ operation. Any CASxx operations that follow an unconditional CASƀƀ operation in the same CAS group are never tested. Therefore, the normal placement of the unconditional CASƀƀ operation is after all other CASxx operations in the CAS group.

You cannot use conditioning indicators on the ENDCS operation for a CAS group.

See “Compare Operations” on page 291 for further rules for the CASxx operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The CASGE operation compares FieldA with FieldB.  If FieldA is
C*  greater than or equal to FieldB, Subr01 is processed and the
C*  program continues with the operation after the ENDCS operation.
C*
C     FieldA        CASGE     FieldB        Subr01
C*
C*  If FieldA is not greater than or equal to FieldB, the program
C*  next compares FieldA with FieldC.  If FieldA is equal to FieldC,
C*  SUBR02 is processed and the program continues with the operation
C*  after the ENDCS operation.
C*
C     FieldA        CASEQ     FieldC        Subr02
C*
C*  If FieldA is not equal to FieldC, the CAS operation causes Subr03
C*  to be processed before the program continues with the operation
C*  after the ENDCS operation.
C*  The CAS statement is used to provide a subroutine if none of
C*  the previous CASxx operations have been met.
C*
C                   CAS                     Subr03
C*
C*  The ENDCS operation denotes the end of the CAS group.
C*
C                   ENDCS
```

Figure 113. CASxx Operation

CAT (Concatenate Two Strings)

Code	Factor 1	Factor 2	Result Field	Indicators		
CAT (P)	Source string 1	Source string 2: number of blanks	Target string			

The CAT operation concatenates the string specified in factor 2 to the end of the string specified in factor 1 and places it in the result field. The source and target strings must all be of the same type, either all character or all graphic. If no factor 1 is specified, factor 2 is concatenated to the end of the result field string.

Factor 1 can contain a string, which can be one of: a field name, array element, named constant, data structure name, table name, or literal. If factor 1 is not specified, the result field is used. In the following discussion, references to factor 1 apply to the result field if factor 1 is not specified.

Factor 2 must contain a string, and may contain the number of blanks to be inserted between the concatenated strings. Its format is the string, followed by a colon, followed by the number of blanks. If graphic strings are being concatenated, the blanks are double-byte blanks. The string portion can contain one of: a field name, array element, named constant, data structure name, table name, literal, or data structure subfield name. The number of blanks portion must be numeric with zero decimal positions, and can contain one of: a named constant, array element, literal, table name, or field name.

If a colon is specified, the number of blanks must be specified. If no colon is specified, concatenation occurs with the trailing blanks, if any, in factor 1, or the result field if factor 1 is not specified.

If the number of blanks, N, is specified, factor 1 is copied to the result field left-justified. If factor 1 is not specified the result field string is used. Then N blanks are added following the last nonblank character. Then factor 2 is appended to this result. Leading blanks in factor 2 are not counted when N blanks are added to the result; they are just considered to be part of factor 2. If the number of blanks is not specified, the trailing and leading blanks of factor 1 and factor 2 are included in the result.

The result field must be a string and can contain one of: a field name, array element, data structure name, or table name. Its length should be the length of factor 1 and factor 2 combined plus any intervening blanks; if it is not, truncation occurs from the right.

A P operation extender indicates that the result field should be padded on the right with blanks after the concatenation occurs if the result field is longer than the result of the operation. If padding is not specified, only the leftmost part of the field is affected.

At run time, if the number of blanks is fewer than zero, the compiler defaults the number of blanks to zero.

Note: Figurative constants cannot be used in the factor 1, factor 2, or result fields. No overlapping is allowed in a data structure for factor 1 and the result field, or for factor 2 and the result field.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* CAT concatenates LAST to NAME and inserts one blank as specified
C* in factor 2.  TEMP contains 'Mr.ƀSmith'.
C                   MOVE      'Mr.   '      NAME              6
C                   MOVE      'Smith '      LAST              6
C     NAME          CAT       LAST:1        TEMP              9
C*
C* CAT concatenates 'RPG' to STRING and places 'RPG/400' in TEMP.
C                   MOVE      '/400'        STRING            4
C     'RPG'         CAT      STRING         TEMP              7
C*
C* The following example is the same as the previous example except
C* that TEMP is defined as a 10 byte field.  P operation extender
C* specifies that blanks will be used in the rightmost positions
C* of the result field that the concatenation result, 'RPG/400',
C* does not fill.  As a result, TEMP contains 'RPG/400ƀƀƀ'
C* after concatenation.
C                   MOVE      *ALL'*'       TEMP             10
C                   MOVE      '/400'        STRING            4
C     'RPG'         CAT(P)    STRING        TEMP
C*
C* After this CAT operation, the field TEMP contains 'RPG/4'.
C* Because the field TEMP was not large enough, truncation occurred.
C                   MOVE      '/400'        STRING            4
C     'RPG'         CAT       STRING        TEMP              5
C*
C* Note that the trailing blanks of NAME are not included because
C* NUM=0.  The field TEMP contains 'RPGIVƀƀƀƀƀ'.
C                   MOVE      'RPG  '       NAME              5
C                   MOVE      'IV   '       LAST              5
C                   Z-ADD     0             NUM               1 0
C     NAME          CAT(P)    LAST:NUM      TEMP             10
```

Figure 114. CAT Operation with leading blanks

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The following example shows leading blanks in factor 2.  After
C* the CAT, the RESULT contains 'MR.ƀSMITH'.
C*
C                   MOVE      'MR.'         NAME              3
C                   MOVE      ' SMITH'      FIRST             6
C     NAME          CAT       FIRST         RESULT            9
C*
C*  The following example shows the use of CAT without factor 1.
C*  FLD2 is a 9 character string.  Prior to the concatenation, it
C*  contains 'ABCƀƀƀƀƀƀ'; FLD1 contains 'XYZ'.
C*  After the concatenation, FLD2 contains 'ABCƀƀXYZƀ'.
C*
C                   MOVEL(P) 'ABC'          FLD2              9
C                   MOVE     'XYZ'          FLD1              3
C                   CAT      FLD1:2         FLD2
```

Figure 115. CAT Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*
D* The following example shows the use of graphic strings
D*
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
D*      Value of Graffld  is 'AACCBBGG'.
D*      Value of Graffld2 after CAT 'aa    AACCBBGG      '
D*      Value of Graffld3 after CAT 'AABBCCDDEEFFGGHHAACC'
D*
D Graffld                        4G   INZ(G'oAACCBBGGi')
D Graffld2                      10G   INZ
D Graffld3                      10G   INZ(G'oAABBCCDDEEFFGGHHi')

CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.

C* The value 2 represents 2 graphic blanks as separators
C     G'oaai'       cat       Graffld:2     Graffld2
C                   cat       Graffld       Graffld3
```

Figure 116. CAT Operation with Graphic data

CHAIN (Random Retrieval from a File)

Code	Factor 1	Factor 2	Result Field	Indicators		
CHAIN (N)	Search argument	File name or Format name	Data structure	NR	ER	_

The CHAIN operation retrieves a record from a full procedural file (F in position 18 of the file description specifications), sets a record identifying indicator on (if specified on the input specifications), and places the data from the record into the input fields.

Factor 1, the search argument, must contain the key or relative record number used to retrieve the record. If access is by key, factor 1 can be a field name, a named constant, a figurative constant, or a literal. In addition, a KLIST name can be specified in factor 1 for an externally described file. If access is by relative record number, factor 1 must contain an integer literal or a numeric field with zero decimal positions.

Factor 2 specifies the file or record format name that is to be read. A record format name is valid with an externally described file. If factor 2 is a file name and access is by key, the CHAIN operation retrieves the first record that matches the search argument.

If factor 2 is a record format name and access is by key, the CHAIN operation retrieves the first record of the specified record type whose key matches the search argument. If no record is found of the specified record type that matches the search argument, a no-record-found condition exists.

You can specify a data-structure name in the result field only if the file named in factor 2 is a program described file (identified by an F in position 22 of the file description specification). When you specify a data-structure name in the result field, the CHAIN operation retrieves the first record whose record identifier matches the search argument in factor 1 and places it in the data structure. See “File Operations” on page 294 for information on transferring data between the file and the data structure.

For a WORKSTN file, the CHAIN operation retrieves a subfile record.

For a multiple device file, you must specify a record format in factor 2. Data is read from the program device identified by the field name specified in the “DEVID(fieldname)” keyword in the file specifications for the device file. If the keyword is not specified, data is read from the device for the last successful input operation to the file.

If the file is specified as an input DISK file, all records are read without locks no operation extender can be specified. If the file is specified as UPDATE, all records are locked if the N operation extender is not specified.

If you are reading from an update disk file, you can specify an N operation extender to indicate that no lock should be placed on the record when it is read (e.g. CHAIN (N)). See the *ILE RPG/400 Programmer's Guide* for more information.

Positions 71 and 72 must contain an indicator that is set on if no record in the file matches the search argument. Positions 73 and 74 can contain an indicator to be set on if the CHAIN operation is not completed successfully. Positions 75 and 76 must be blank.

When the CHAIN operation is successful, the file specified in factor 2 is positioned such that a subsequent read operation retrieves the record logically following or preceding the retrieved record. When the CHAIN operation is not completed successfully (for example, an error occurs or no record is found), the file specified in factor 2 must be repositioned (for example, by a CHAIN or SETLL operation) before a subsequent read operation can be done on that file.

If an update (on the calculation or output specifications) is done on the file specified in factor 2 immediately after a successful CHAIN operation to that file, the last record retrieved is updated.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The CHAIN operation retrieves the first record from the file,
C*  FILEX, that has a key field with the same value as the search
C*  argument KEY (factor 1).
C*
C     KEY           CHAIN     FILEX                              60
C*
C*
C*  If a record with a key value equal to the search argument is not
C*  found, indicator 60 is set on and the EXSR operation conditioned
C*  by indicator 60 is processed.  If a record is found with a key
C*  value equal to the search argument, the program continues with
C*  the calculations after the EXSR operation.
C*
C   60              EXSR      Not_Found
```

Figure 117. CHAIN Operation with a File Name in Factor 2

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The CHAIN operation uses the value contained in the search
C*  argument KEY to retrieve a record of the record type REC1 from
C*  an externally described file. If no record is found of the
C*  specified type that has a key field equal to the search
C*  argument, indicator 72 is set on.  A complex key with a KLIST is
C*  used to retrieve records from files that have a composite key.
C*  If a record of the specified type is found that has a key field
C*  equal to the search argument, indicator 72 is set off and therefore
C*  the UPDATE operation is processed.
C*
C     KEY           CHAIN     REC1                                 72
C     KEY           KLIST
C                   KFLD                    Field1
C                   KFLD                    Field2
C                   IF        NOT *IN72
C*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The UPDATE operation modifies all the fields in the REC1 record.
C*
C                   UPDATE    REC1
C                   ENDIF
C*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The following example shows a CHAIN with no lock.
C*
C                   MOVE 3                  Rec_No
C     Rec_No        CHAIN (N) INPUT                                99
```

Figure 118. CHAIN Operation with a Record Format Name and with No Lock

CHECK (Check Characters)

Code	Factor 1	Factor 2	Result Field	Indicators		
CHECK	Comparator string	Base string:start	Left-position	_	ER	FD

The CHECK operation verifies that each character in the base string (factor 2) is among the characters indicated in the comparator string (factor 1). The base string and comparator string must be of the same type, either both character or both graphic. Verifying begins at the leftmost character of factor 2 and continues character by character, from left to right. Each character of the base string is compared with the characters of factor 1. If a match for a character in factor 2 exists in factor 1, the next base string character is verified. If a match is not found, an integer value is placed in the result field to indicate the position of the incorrect character.

You can specify a start position in factor 2, separating it from the base string by a colon. The start position is optional and defaults to 1. If the start position is greater than 1, the value in the result field is relative to the leftmost position in the base string, regardless of the start position.

The operation stops checking when it finds the first incorrect character or when the end of the base string is encountered. If no incorrect characters are found, the result field is set to zero.

If the result field is an array, the operation continues checking after the first incorrect character is found for as many occurrences as there are elements in the array. If there are more array elements than incorrect characters, all of the remaining elements are set to zeros.

Factor 1 must be a string, and can contain one of: a field name, array element, named constant, data structure name, data structure subfield, literal, or table name.

Factor 2 must contain either the base string or the base string, followed by a colon, followed by the start location. The base string portion of factor 2 can contain: a field name, array element, named constant, data-structure name, literal, or table name. The start location portion of factor 2 must be numeric with no decimal positions, and can be a named constant, array element, field name, literal, or table name. If no start location is specified, a value of 1 is used.

The result field can be a numeric variable, numeric array element, numeric table name, or numeric array. Define the field or array specified with no decimal positions. The result field is an optional field; if you do not specify it, you must specify the found indicator in position 75-76. If graphic data is used, the result field will contain graphic character positions (that is, position 3, the 3rd graphic character, will be character position 5).

Note: Figurative constants cannot be used in the factor 1, factor 2, or result fields. No overlapping is allowed in a data structure for factor 1 and the result field or for factor 2 and the result field.

Any valid indicator can be specified in positions 7 to 11.

The indicator in positions 73-74 is turned on if an error occurs. The indicator in positions 75-76 is turned on if any incorrect characters are found.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D* After the following example, N=6 and the found indicator 90
D* is on.  Because the start position is 2, the first nonnumeric
D* character found is the '.'.
D*
D
D Digits          C                   '0123456789'
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C
C                   MOVE      '$2000.'      Salary
C     Digits        CHECK     Salary:2      N                         90
C*
C* Because factor 1 is a blank, CHECK indicates the position
C* of the first nonblank character.  If STRING contains 'ƀƀƀthe',
C* NUM will contain the value 4.
C*
C
C           ' '     CHECK     String        Num               2 0
```

Figure 119. CHECK Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D* The following example checks that FIELD contains only the letters
D* A to J.  As a result, ARRAY=(136000) after the CHECK operation.
D* Indicator 90 turns on.
D*
D
D Letter          C                   'ABCDEFGHIJ'
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C
C                   MOVE      '1A=BC*'      Field             6
C     Letter        CHECK     Field         Array                     90
C
C*
C* In the following example, because FIELD contains only the
C* letters A to J, ARRAY=(000000).  Indicator 90 turns off.
C*
C
C                   MOVE      'FGFGFG'      Field             6
C     Letter        CHECK     Field         Array                     90
C
C
```

Figure 120. CHECK Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D
D* The following example checks a DBCS field for valid graphic
D* characters starting at graphic position 2 in the field.
D
D*       Value of Graffld  is 'DDBBCCDD'.
D*       The value of num after the CHECK is 4, since this is the
D*       first character 'DD' which is not contained in the string.
D
D Graffld                        4G   INZ(G'oDDBBCCDDi')
D Num                            5 0
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
C
C
C     G'oAABBCCi'   check     Graffld:2     Num
```

Figure 121. CHECK Operation with graphic data

CHECKR (Check Reverse)

Code	Factor 1	Factor 2	Result Field	Indicators		
CHECKR	Comparator string	Base string:start	Right-position	_	ER	FD

The CHECKR operation verifies that each character in the base string (factor 2) is among the characters indicated in the comparator string (factor 1). The base string and comparator string must be of the same type, either both character or both graphic. Verifying begins at the rightmost character of factor 2 and continues character by character, from right to left. Each character of the base string is compared with the characters of factor 1. If a match for a character in factor 2 exists in factor 1, the next source character is verified. If a match is not found, an integer value is placed in the result field to indicate the position of the incorrect character. Although checking is done from the right, the position placed in the result field will be relative to the left.

You can specify a start position in factor 2, separating it from the base string by a colon. The start position is optional and defaults to the length of the string. The value in the result field is relative to the leftmost position in the source string, regardless of the start position.

If the result field is not an array, the operation stops checking when it finds the first incorrect character or when the end of the base string is encountered. If no incorrect characters are found, the result field is set to zero.

If the result field is an array, the operation continues checking after the first incorrect character is found for as many occurrences as there are elements in the array. If there are more array elements than incorrect characters, all of the remaining elements are set to zeros.

Factor 1 must be a string and can contain one of: a field name, array element, named constant, data structure name, data structure subfield, literal, or table name.

Factor 2 must contain either the base string or the base string, followed by a colon, followed by the start location. The base string portion of factor 2 can contain: a field name, array element, named constant, data structure name, data structure subfield name, literal, or table name. The start location portion of factor 2 must be numeric with no decimal positions, and can be a named constant, array element, field name, literal, or table name. If no start location is specified, the length of the string is used.

The result field can be a numeric variable, numeric array element, numeric table name, or numeric array. Define the field or array specified with no decimal positions. If graphic data is used, the result field will contain graphic character positions (that is, position 3, the 3rd graphic character, will be character position 5). The result field is an optional field; if you do not specify it, you must specify the found indicator in position 75-76.

Note: Figurative constants cannot be used in the factor 1, factor 2, or result fields. No overlapping is allowed in a data structure for factor 1 and the result field, or for factor 2 and the result field.

Any valid indicator can be specified in positions 7 to 11.

The indicator in positions 73-74 is turned on if an error occurs. The indicator in positions 75-76 is turned on if any incorrect characters are found.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* Because factor 1 is a blank character, CHECKR indicates the
C* position of the first nonblank character.  This use of CHECKR
C* allows you to determine the length of a string.  If STRING
C* contains 'ABCDEF   ', NUM will contain the value 6.
C
C     ' '           CHECKR    String        Num                  20
C
```

Figure 122. CHECKR Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
D* After the following example, N=1 and the found indicator 90
D* is on. Because the start position is 5, the operation begins
D* with the rightmost 0 and the first nonnumeric found is the '$'.
D*
D Digits          C                   '0123456789'
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C                   MOVE      '$2000.'      Salary           6
C     Digits        CHECKR    Salary:5      N                        90
C
```

Figure 123. CHECKR Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*
D* The following example checks that FIELD contains only the letters
D* A to J.  As a result, ARRAY=(876310) after the CHECKR operation.
D* Indicator 90 turns on.
D
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D Array           S              1    DIM(6)
D Letter          C                   'ABCDEFGHIJ'
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C                   MOVE      '1A=BC***'    Field             8
C     Letter        CHECKR    Field         Array                    90
C
```

Figure 124. CHECKR Operation

CLEAR (Clear)

Code	Factor 1	Factor 2	Result Field	Indicators		
CLEAR	*NOKEY	*ALL	Structure or Variable			

The CLEAR operation sets elements in a structure (record format, data structure, array, or table) or a variable (field, subfield, array element or indicator), to their default value depending on field type (numeric, character, graphic, indicator, pointer, or date/time/timestamp). It allows you to clear structures on a global basis, as well as element by element, during run time.

Factor 1 must be blank unless the result field contains a record format name from a DISK file, in which case it can contain *NOKEY to indicate that key fields are not to be cleared.

The Result Field contains the structure or variable that is to be cleared. It can contain: a record-format name, data-structure name, array name, table name, field name, subfield, array element, or indicator name. If you specify a record-format name or data structure, all fields are cleared in the order they are defined within the structure. Fields in a data structure will be cleared according to their data types. If you have partially overlapping fields of different definitions, data that is not valid could exist in non-character fields. With a multiple-occurrence data structure, only those fields in the current occurrence are cleared. If you specify a table name, the current table element is cleared; if an array name, the entire array is cleared. If you specify an array element (including indicators) in the result field using an array index, only the element specified is cleared.

Factor 2 may be blank or can contain *ALL. If *ALL is specified, and the result field contains a multiple occurrence data structure or a table name, all occurrences or table elements will be cleared and the occurrence level will be set to 1.

When the CLEAR operation is applied to a record format name, and factor 2 contains *ALL and factor 1 is blank, all fields in the record format are cleared. If factor 1 contains *NOKEY, all fields for the record format except the key fields are cleared.

When the CLEAR operation is applied to a record-format name, and factor 2 is blank, only output fields in the record format are affected. For WORKSTN file record formats, only fields with a usage of output or both are affected. All field-conditioning indicators are affected by this operation. Fields in DISK, SEQ, or PRINTER file record formats are affected only if those record formats are output in the program. Input-only fields are not affected by the CLEAR operation. By definition, they assume new values at the next input operation.

Please see Chapter 7, “Data Types and Data Formats” for their default values.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D*
D
D DS1             DS
D  Num                       2    5 0
D  Char                     20  30A
D
D MODS            DS                  OCCURS(2)
D  Fld1                      1    5
D  Fld2                      6   10 0
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  In the following example, CLEAR sets all subfields in the data
C*  structure DS1 to their defaults, CHAR to blank, NUM to zero.
C
C                   CLEAR                   DS1
C
C*
C*  In the following example, CLEAR sets all occurrences for the
C*  multiple occurrence data structure MODS to their default values
C*  Fld1 to blank, Fld2 to zero.
C
C                   CLEAR     *ALL          MODS
C
```

Figure 125. CLEAR Operation

Figure 126 on page 337 shows an example of the field initialization for the CLEAR record format.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
A* Field2 and Field3 are defined as output capable fields and can be
A* affected by the CLEAR operation.  Indicator 10 can also be
A* changed by the CLEAR operation even though it conditions an
A* input only field because field indicators are all treated
A* as output fields. The reason for this is that *ALL was not specified
A* on the CLEAR operation
A*
AAN01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++*
A          R FMT01
A  10        Field1        10A  I  2 30
A            Field2        10A  O  3 30
A            Field3        10A  B  4 30
A*
A*  End of DDS source
A*


FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
F
FWORKSTN   CF   E             WORKSTN INCLUDE(FMT01)
F
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D
D IN              C                   'INPUT DATA'
D
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C                   CLEAR                   FMT01
C                   WRITE     FMT01
C
C*
C* The program will loop until PF03 is pressed.
C*
C     *IN03         DOWEQ     '0'
C                   READ      FMT01                                  LR
C*
C* PF04 will transfer input fields to output fields.
C
C     *IN04         IFEQ      '1'
C                   MOVEL     Field3        Field2
C                   MOVEL     Field1        Field3
C                   CLEAR                   *IN04
C                   ENDIF
C                   MOVEL     IN            Field1
C
```

Figure 126 (Part 1 of 2). Field Initialization for the CLEAR Record Format

```
C* When PF11 is pressed, all the fields in the record format
C* defined as output or both will be reset to the values they
C* held after the initialization step.
C*
C     *IN11         IFEQ      '1'
C                   RESET     FMT01
C                   CLEAR                   *IN11
C                   ENDIF
C* When PF12 is pressed, all the fields in the record
C* format defined as output or both will be cleared.
C
C     *IN12         IFEQ      '1'
C                   CLEAR                   FMT01
C                   CLEAR                   *IN12
C                   ENDIF
C  N03              WRITE     FMT01
C                   ENDDO
C                   SETON                                        LR
C
```

Figure 126 (Part 2 of 2). Field Initialization for the CLEAR Record Format

CLOSE (Close Files)

Code	Factor 1	Factor 2	Result Field	Indicators		
CLOSE		File name		_	ER	_

The explicit CLOSE operation closes one or more files or devices and disconnects them from the program. The file cannot be used again in the program unless you specify an explicit OPEN for that file. A CLOSE operation to an already closed file does not produce an error.

Factor 2 names the file to be closed. You can specify the keyword *ALL in factor 2 to close all the files at once. You cannot specify an array or table file (identified by a T in position 18 of the file description specifications) in factor 2.

You can specify a resulting indicator in positions 73 and 74 to be set on if the CLOSE operation is not completed successfully. Positions 71, 72, 75, and 76 must be blank.

If an array or table is to be written to an output file (specified using the TOFILE keyword) the array or table dump does not occur at LR time if the file is closed by a CLOSE operation). If the file is closed, it must be reopened for the dump to occur.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The explicit CLOSE operation closes FILEB.
C
C                   CLOSE     FILEB
C
C*  The CLOSE *ALL operation closes all files in the
C*  program. You must specify an explicit OPEN for any file that
C*  you wish to use again.  If the CLOSE operation is not completed
C*  completed successfully, indicator 17 is set on.
C
C                   CLOSE     *ALL                                 17
C
```

Figure 127. CLOSE Operation

COMMIT (Commit)

Code	Factor 1	Factor 2	Result Field	Indicators		
COMMIT	Boundary			_	ER	_

The COMMIT operation:

- Makes all the changes to your files, opened for commitment control, that have been specified in output operations since the previous commit or rollback "ROLBK (Roll Back)" operation (or since the beginning of operations under commitment control if there has been no previous commit or rollback operation). You specify a file to be opened for commit by specifying the COMMIT keyword on the file specification.
- Releases all the record locks for files you have under commitment control.

The file changes and the record-lock releases apply to all the files you have under commitment control, whether the changes have been requested by the program issuing the COMMIT operation, or by another program in the same activation group or job, dependent on the commit scope specified on the STRCMTCTL command. The program issuing the COMMIT operation does not need to have any files under commitment control. The COMMIT operation does not change the file position.

Commitment control starts when the CL command STRCMTCTL is executed. See the chapter on "Commitment Control" in the *ILE RPG/400 Programmer's Guide* for more information.

In factor 1, you can specify a constant or variable (of any type except pointer) to identify the boundary between the changes made by this COMMIT operation and subsequent changes. If you leave factor 1 blank, the identifier is null.

The optional indicator in positions 73 and 74 is set on if the operation is not completed successfully. For example, the indicator is set on if commitment control is not active.

COMP (Compare)

Code	Factor 1	Factor 2	Result Field	Indicators		
COMP	Comparand	Comparand		HI	LO	EQ

The COMP operation compares factor 1 with factor 2. Factor 1 and factor 2 can contain a literal, a named constant, a field name, a table name, an array element, a data structure, or a figurative constant. Factor 1 and factor 2 must have the same data type. As a result of the comparison, indicators are set on as follows:

High: (71-72) Factor 1 is greater than factor 2.
Low: (73-74) Factor 1 is less than factor 2.
Equal: (75-76) Factor 1 equals factor 2.

You must specify at least one resulting indicator in positions 71 through 76. Do not specify the same indicator for all three conditions. When specified, the resulting indicators are set on or off (for each cycle) to reflect the results of the compare.

For further rules for the COMP operation, see “Compare Operations” on page 291.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  Initial field values are:
C*                    FLDA = 100.00
C*                    FLDB = 105.00
C*                    FLDC = 100.00
C*                    FLDD = ABC
C*                    FLDE = ABCDE
C*
C*  Indicator 12 is set on; indicators 11 and 13 are set off.
C     FLDA          COMP      FLDB                               111213
C*
C*  Indicator 15 is set on; indicator 14 is set off.
C     FLDA          COMP      FLDB                               141515
C*
C*  Indicator 18 is set on; indicator 17 is set off.
C     FLDA          COMP      FLDC                               171718
C*
C*  Indicator 21 is set on; indicators 20 and 22 are set off
C     FLDD          COMP      FLDE                               202122
```

Figure 128. COMP Operation

DEFINE (Field Definition)

Code	Factor 1	Factor 2	Result Field	Indicators		
DEFINE	*LIKE	Referenced field	Defined field			
DEFINE	*DTAARA	External data area	Internal field			

Depending on the factor 1 entry, the declarative DEFINE operation can do either of the following:

- Define a field based on the attributes (length and decimal positions) of another field.
- Define a field as a data area.

You can specify the DEFINE operation anywhere within calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, the LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry is used for documentation only. Conditioning indicator entries (positions 9 through 11) are not permitted.

*LIKE DEFINE

The "DEFINE (Field Definition)" operation with *LIKE in factor 1 defines a field based upon the attributes (length and decimal positions) of another field.

Factor 2 must contain the name of the field being referenced, and the result field must contain the name of the field being defined. The field specified in factor 2 , which can be defined in the program or externally, provides the attributes for the field being defined. Factor 2 cannot be a literal or a named constant. If factor 2 is an array, an array element, or a table name, the attributes of an element of the array or table are used to define the field. The result field cannot be an array, an array element, a data structure, or a table name.

You can use positions 64 through 68 (field length) to make the result field entry longer or shorter than the factor 2 entry. A plus sign (+) preceding the number indicates a length increase; a minus sign (-) indicates a length decrease. Positions 65-68 can contain the increase or decrease in length (right-adjusted) or can be blank. If positions 64 through 68 are blank, the result field entry is defined with the same length as the factor 2 entry. You cannot change the number of decimal positions for the field being defined. The field length entry is allowed only for graphic, numeric, and character fields.

For graphic fields the field length difference is calculated in double byte characters.

See Figure 129 on page 343 for examples of *LIKE DEFINE.

*DTAARA DEFINE

The "DEFINE (Field Definition)" operation with *DTAARA in factor 1 associates a field, a data structure, a data-structure subfield, or a data-area data structure (within your RPG IV program) with an AS/400 data area (outside your RPG IV program).

In factor 2, specify the external name of a data area. Use *LDA for the name of the local data area or use *PDA for the Program Initialization Parameters (PIP) data area. If you leave factor 2 blank, the result field entry is both the RPG IV name and the external name of the data area.

In the result field, specify the name of one of the following that you have defined in your program: a field, a data structure, a data structure subfield, or a data-area data structure. You use this name with the IN and OUT operations to retrieve data from and write data to the data area specified in factor 2. When you specify a data-area data structure in the result field, the RPG IV program implicitly retrieves data from the data area at program start and writes data to the data area when the program ends.

The result field entry must not be the name of a program-status data structure, a file-information data structure (INFDS), a multiple-occurrence data structure, an input record field, an array, an array element, or a table. It cannot be the name of a subfield of a multiple-occurrence data structure, of a data area data structure, of a program-status data structure, of a file-information data structure (INFDS), or of a data structure that already appears on a *DTAARA DEFINE statement, or has already been defined as a data area using the DTAARA keyword on a definition specification.

In positions 64 through 70, you can define the length and number of decimal positions for the entry in the result field. These specifications must match those for the external description of the data area specified in factor 2. The local data area is character data of length 1024, but within your program you can access the local data area as if it has a length of 1024 or less.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
*
C*  FLDA is a 7-position character field.
C*  FLDB is a 5-digit field with 2 decimal positions.
C*
C*
C*  FLDP is a 7-position character field.
C     *LIKE        DEFINE    FLDA          FLDP
C*
C*  FLDQ is a 9-position character field.
C     *LIKE        DEFINE    FLDA          FLDQ             +2
C*
C*  FLDR is a 6-position character field.
C     *LIKE        DEFINE    FLDA          FLDR            - 1
C*
C*  FLDS is a 5-position numeric field with 2 decimal positions.
C     *LIKE        DEFINE    FLDB          FLDS
C*
C*  FLDT is a 6-position numeric field with 2 decimal positions.
C     *LIKE        DEFINE    FLDB          FLDT            + 1
C*
C*  FLDU is a 3-position numeric field with 2 decimal positions.
C     *LIKE        DEFINE    FLDB          FLDU            - 2
C*
C*  FLDX is a 3-position numeric field with 2 decimal positions.
C     *LIKE        DEFINE    FLDU          FLDX
```

Figure 129 (Part 1 of 2). DEFINE Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The attributes (length and decimal positions) of
C*  the data area (TOTGRS) must be the same as those for the
C*  external data area.
C
C     *DTAARA       DEFINE                  TOTGRS           10 2
C
C*
C*  The result field entry (TOTNET) is the name of the data area to
C*  be used within the RPG IV program.  The factor 2 entry (TOTAL)
C*  is the name of the data area as defined to the system.
C
C     *DTAARA       DEFINE    TOTAL         TOTNET
C
C*
C*  The result field entry (SAVTOT) is the name of the data area to
C*  be used within the RPG IV program.  The factor 2 entry (*LDA)
C*  indicates the use of the local data area.
C
C     *DTAARA       DEFINE    *LDA          SAVTOT
```

Figure 129 (Part 2 of 2). DEFINE Operation

DELETE (Delete Record)

Code	Factor 1	Factor 2	Result Field	Indicators		
DELETE	Search argument	File name		NR	ER	_

The DELETE operation deletes a record from a database file. The file must be an update file (identified by a U in position 17 of the file description specifications) The deleted record can never be retrieved.

If factor 1 contains no entry, the DELETE operation deletes the current record (the last record retrieved). The record must have been locked by a previous input operation (for example, CHAIN or READ).

Factor 1, the search argument, can contain a key or relative record number that identifies the record to be deleted. If access is by key, factor 1 can be a field name, a named constant, or a literal. In addition, a KLIST name can be specified in factor 1 for an externally described file. If duplicate records exist for the key, only the first of the duplicate records is deleted from the file. If access is by relative record number, factor 1 must contain a numeric constant or variable with zero decimal positions.

Factor 2 must contain the name of the update file or the name of a record format in the file from which a record is to be deleted. A record format name is valid only with an externally described file. If factor 1 is not specified, the record format name must be the name of the last record read from the file; otherwise, an error occurs.

If factor 1 has an entry, you must specify a resulting indicator in positions 71 and 72. If factor 1 does not have an entry, leave these positions blank. This indicator is set on if the record to be deleted is not found in the file. You can specify a resulting indicator in positions 73 and 74; it is set on if the DELETE operation is not completed successfully. (For example, you lack the authority on the file to delete records) Leave positions 75 and 76 blank.

Under the OS/400 operating system, if a read operation is done on the file specified in factor 2 after a successful DELETE operation to that file, the next record after the deleted record is obtained.

DIV (Divide)

Code	Factor 1	Factor 2	Result Field	Indicators		
DIV (H)	Dividend	Divisor	Quotient	+	–	Z

If factor 1 is specified, the DIV operation divides factor 1 by factor 2; otherwise, it divides the result field by factor 2. The quotient (result) is placed in the result field. If factor 1 is 0, the result of the divide operation is 0. Factor 2 cannot be 0. If it is, an error occurs and the RPG IV exception/error handling routine receives control. When factor 1 is not specified, the result field (dividend) is divided by factor 2 (divisor), and the result (quotient) is placed in the result field. Factor 1 and factor 2 must be numeric; each can contain one of: an array, array element, field, figurative constant, literal, named constant, subfield, or table name.

Any remainder resulting from the divide operation is lost unless the move remainder (MVR) operation is specified as the next operation. If you use conditioning indicators, you must ensure that the DIV operation is processed immediately before the MVR operation. If the MVR operation is processed before the DIV operation, undesirable results occur. If move remainder is the next operation, the result of the divide operation cannot be half-adjusted (rounded).

For further rules for the DIV operation, see “Arithmetic Operations” on page 287.

Figure 100 on page 289 shows examples of the DIV operation.

DO (Do)

Code	Factor 1	Factor 2	Result Field	Indicators		
DO	Starting value	Limit value	Index value			

The DO operation begins a group of operations and indicates the number of times the group will be processed. To indicate the number of times the group of operations is to be processed, specify an index field, a starting value, and a limit value. An associated ENDDO statement marks the end of the group. For further information on DO groups, see "Structured Programming Operations" on page 303.

In factor 1, specify a starting value with zero decimal positions, using a numeric literal, named constant, or field name. If you do not specify factor 1, the starting value is 1.

In factor 2, specify the limit value with zero decimal positions, using a numeric field name, literal, or named constant. If you do not specify factor 2, the limit value is 1.

In the result field, specify a numeric field name that will contain the current index value. The result field must be large enough to contain the limit value plus the increment. If you do not specify an index field, one is generated for internal use. Any value in the index field is replaced by factor 1 when the DO operation begins.

Factor 2 of the associated ENDDO operation specifies the value to be added to the index field. It can be a numeric literal or a numeric field with no decimal positions. If it is blank, the value to be added to the index field is 1.

In addition to the DO operation itself, the conditioning indicators on the DO and ENDDO statements control the DO group. The conditioning indicators on the DO statement control whether or not the DO operation begins. These indicators are checked only once, at the beginning of the DO loop. The conditioning indicators on the associated ENDDO statement control whether or not the DO group is repeated another time. These indicators are checked at the end of each loop.

The DO operation follows these 7 steps:

1. If the conditioning indicators on the DO statement line are satisfied, the DO operation is processed (step 2). If the indicators are not satisfied, control passes to the next operation to be processed following the associated ENDDO statement (step 7).
2. The starting value (factor 1) is moved to the index field (result field) when the DO operation begins.
3. If the index value is greater than the limit value, control passes to the calculation operation following the associated ENDDO statement (step 7). Otherwise, control passes to the first operation after the DO statement (step 4).
4. Each of the operations in the DO group is processed.
5. If the conditioning indicators on the ENDDO statement are not satisfied, control passes to the calculation operation following the associated ENDDO statement (step 7). Otherwise, the ENDDO operation is processed (step 6).
6. The ENDDO operation is processed by adding the increment to the index field. Control passes to step 3. (Note that the conditioning indicators on the DO statement are not tested again (step 1) when control passes to step 3.)

7. The statement after the ENDDO statement is processed when the conditioning indicators on the DO or ENDDO statements are not satisfied (step 1 or 5), or when the index value is greater than the limit value (step 3).

Remember the following when specifying the DO operation:

- The index, increment, limit value, and indicators can be modified within the loop to affect the ending of the DO group.
- A DO group cannot span both detail and total calculations.

See “LEAVE (Leave a Do Group)” and “ITER (Iterate)” for information on how those operations affect a DO operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The DO group is processed 10 times when indicator 17 is on;
C*  it stops running when the index value in field X, the result
C*  field, is greater than the limit value (10) in factor 2.  When
C*  the DO group stops running, control passes to the operation
C*  immediately following the ENDDO operation.  Because factor 1
C*  in the DO operation is not specified, the starting value is 1.
C*  Because factor 2 of the ENDDO operation is not specified, the
C*  incrementing value is 1.
C
C   17              DO        10            X                 3 0
C                   :
C                   ENDDO
C*
C*  The DO group can be processed 10 times.  The DO group stops
C*  running when the index value in field X is greater than
C*  the limit value (20) in factor 2, or if indicator 50 is not on
C*  when the ENDDO operation is encountered.  When indicator 50
C*  is not on, the ENDDO operation is not processed; therefore,
C*  control passes to the operation following the ENDDO operation.
C*  The starting value of 2 is specified in factor 1 of the DO
C*  operation, and the incrementing value of 2 is specified in
C*  factor 2 of the ENDDO operation.
C*
C     2             DO        20            X                 3 0
C                   :
C                   :
C                   :
C   50              ENDDO     2
```

Figure 130. DO Operation

DOU (Do Until)

Code	Factor 1	Extended Factor 2
DOU	Blank.	Expression

The DOU operation code precedes a group of operations which you want to execute at least once and possibly more than once. Its function is similar to that of the DOUxx operation code. An associated ENDDO statement marks the end of the group. It differs in that the logical condition is expressed by an indicator valued expression in the extended-Factor 2 entry. The operations controlled by the DOU operation are performed until the expression in the extended factor 2 field is true. See Chapter 21, “Expressions” for details on expressions.

Level and conditioning indicators are valid. Factor 1 must be blank. Extended factor 2 contains the expression to be evaluated.

```
CL0N01Factor1+++++++OpcodeE+Extended-factor2+++++++++++++++++++++++++++++
C                             Extended-factor2-continuation++++++++++++++++
C* In this example, the do loop will be repeated until the F3
C* is pressed.
C                   DOU     *INKC
C                   :
C                   :
C                   ENDDO
C
C* The following do loop will be repeated until *In01 is on
C* or until FIELD2 is greater than FIELD3
C
C                   DOU     *IN01 OR (Field2 > Field3)
C                   :
C                   :
C                   ENDDO
C* The following loop will be repeated until X is greater than the number
C* of elements in Array
C
C                   DOU     X > %elem(Array)
C                   EVAL    Total = Total + Array(x)
C                   EVAL    X = X + 1
C                   ENDDO
C
C*
```

Figure 131. DOU Example

DOUxx (Do Until)

Code	Factor 1	Factor 2	Result Field	Indicators		
DOUxx	Comparand	Comparand				

The DOUxx operation code precedes a group of operations which you want to execute at least once and possibly more than once. An associated ENDDO statement marks the end of the group. For further information on DO groups and the meaning of xx, see "Structured Programming Operations" on page 303.

Factor 1 and factor 2 must contain a literal, a named constant, a field name, a table name, an array element, a figurative constant, or a data structure name. Factor 1 and factor 2 must be the same data type.

On the DOUxx statement, you indicate a relationship xx. To specify a more complex condition, immediately follow the DOUxx statement with ANDxx or ORxx statements. The operations in the DO group are processed once, and then the group is repeated while the relationship exists between factor 1 and factor 2 or the condition specified by a combined DOUxx, ANDxx, or ORxx operation exists. The group is always processed at least once even if the condition is not true at the start of the group.

In addition to the DOUxx operation itself, the conditioning indicators on the DOUxx and ENDDO statements control the DO group. The conditioning indicators on the DOUxx statement control whether or not the DOUxx operation begins. The conditioning indicators on the associated ENDDO statement can cause a DO loop to end prematurely.

The DOUxx operation follows these steps:

1. If the conditioning indicators on the DOUxx statement line are satisfied, the DOUxx operation is processed (step 2). If the indicators are not satisfied, control passes to the next operation that can be processed following the associated ENDDO statement (step 6).
2. The DOUxx operation is processed by passing control to the next operation that can be processed (step 3). The DOUxx operation does not compare factor 1 and factor 2 or test the specified condition at this point.
3. Each of the operations in the DO group is processed.
4. If the conditioning indicators on the ENDDO statement are not satisfied, control passes to the next calculation operation following the associated ENDDO statement (step 6). Otherwise, the ENDDO operation is processed (step 5).
5. The ENDDO operation is processed by comparing factor 1 and factor 2 of the DOUxx operation or testing the condition specified by a combined operation. If the relationship xx exists between factor 1 and factor 2 or the specified condition exists, the DO group is finished and control passes to the next calculation operation after the ENDDO statement (step 6). If the relationship xx does not exist between factor 1 and factor 2 or the specified condition does not exist, the operations in the DO group are repeated (step 3).
6. The statement after the ENDDO statement is processed when the conditioning indicators on the DOUxx or ENDDO statements are not satisfied (steps 1 or 4), or when the relationship xx between factor 1 and factor 2 or the specified condition exists at step 5.

See "LEAVE (Leave a Do Group)" and "ITER (Iterate)" for information on how those operations affect a DOUxx operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The DOUEQ operation runs the operation within the DO group at
C*  least once.
C
C     FLDA          DOUEQ     FLDB
C
C*
C*  At the ENDDO operation, a test is processed to determine whether
C*  FLDA is equal to FLDB.  If FLDA does not equal FLDB, the
C*  preceding operations are processed again.  This loop continues
C*  processing until FLDA is equal to FLDB.  When FLDA is equal to
C*  FLDB, the program branches to the operation immediately
C*  following the ENDDO operation.
C
C                   SUB       1             FLDA
C                   ENDDO
C
C*
C*  The combined DOUEQ ANDEQ OREQ operation processes the operation
C*  within the DO group at least once.
C
C     FLDA          DOUEQ     FLDB
C     FLDC          ANDEQ     FLDD
C     FLDE          OREQ      100
C
C*
C*  At the ENDDO operation, a test is processed to determine whether
C*  the specified condition, FLDA equal to FLDB and FLDC equal to
C*  FLDD, exists.  If the condition exists, the program branches to
C*  the operation immediately following the ENDDO operation.  There
C*  is no need to test the OREQ condition, FLDE equal to 100, if the
C*  DOUEQ and ANDEQ conditions are met.  If the specified condition
C*  does not exist, the OREQ condition is tested.  If the OREQ
C*  condition is met, the program branches to the operation
C*  immediately following the ENDDO.  Otherwise, the operations
C*  following the OREQ operation are processed and then the program
C*  processes the conditional tests starting at the second DOUEQ
C*  operation.  If neither the DOUEQ and ANDEQ condition nor the
C*  OREQ condition is met, the operations following the OREQ
C*  operation are processed again.
C
C                   SUB       1             FLDA
C                   ADD       1             FLDC
C                   ADD       5             FLDE
C                   ENDDO
```

Figure 132. DOUxx Operations

DOW (Do While)

Code	Factor 1	Factor 2
DOW	Blank.	Expression

The DOW operation code precedes a group of operations which you want to process when a given condition exists. Its function is similar to that of the DOWxx operation code. An associated ENDDO statement marks the end of the group. It differs in that the logical condition is expressed by an indicator valued expression in the extended-Factor 2 entry. The operations controlled by the DOW operation are performed while the expression in the extended factor 2 field is true.

Level and conditioning indicators are valid. Factor 1 must be blank. Factor 2 contains the expression to be evaluated.

```
CL0N01Factor1+++++++OpcodeE+Extended-factor2++++++++++++++++++++++++++++++
C                           Extended-factor2-continuation+++++++++++++++++
C* In this example, the do loop will be repeated until the condition
C* is false. That is when A > 5 and/or B+C are not equal to zero.
C
C                   DOW     A <= 5 AND B+C = 0
C                   :
C                   :
C                   ENDDO
C
```

Figure 133. DOW Example

DOWxx (Do While)

Code	Factor 1	Factor 2	Result Field	Indicators		
DOWxx	Comparand	Comparand				

The DOWxx operation code precedes a group of operations which you want to process when a given condition exists. To specify a more complex condition, immediately follow the DOWxx statement with ANDxx or ORxx statements. An associated ENDDO statement marks the end of the group. For further information on DO groups and the meaning of xx, see “Structured Programming Operations” on page 303.

Factor 1 and factor 2 must contain a literal, a named constant, a figurative constant, a field name, a table name, an array element, or a data structure name. Factor 1 and factor 2 must be of the same data type. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See “Compare Operations” on page 291.

In addition to the DOWxx operation itself, the conditioning indicators on the DOWxx and ENDDO statements control the DO group. The conditioning indicators on the DOWxx statement control whether or not the DOWxx operation is begun. The conditioning indicators on the associated ENDDO statement control whether the DOW group is repeated another time.

The DOWxx operation follows these steps:

1. If the conditioning indicators on the DOWxx statement line are satisfied, the DOWxx operation is processed (step 2). If the indicators are not satisfied, control passes to the next operation to be processed following the associated ENDDO statement (step 6).
2. The DOWxx operation is processed by comparing factor 1 and factor 2 or testing the condition specified by a combined DOWxx, ANDxx, or ORxx operation. If the relationship xx between factor 1 and factor 2 or the condition specified by a combined operation does not exist, the DO group is finished and control passes to the next calculation operation after the ENDDO statement (step 6). If the relationship xx between factor 1 and factor 2 or the condition specified by a combined operation exists, the operations in the DO group are repeated (step 3).
3. Each of the operations in the DO group is processed.
4. If the conditioning indicators on the ENDDO statement are not satisfied, control passes to the next operation to run following the associated ENDDO statement (step 6). Otherwise, the ENDDO operation is processed (step 5).
5. The ENDDO operation is processed by passing control to the DOWxx operation (step 2). (Note that the conditioning indicators on the DOWxx statement are not tested again at step 1.)
6. The statement after the ENDDO statement is processed when the conditioning indicators on the DOWxx or ENDDO statements are not satisfied (steps 1 or 4), or when the relationship xx between factor 1 and factor 2 of the specified condition does not exist at step 2.

See “LEAVE (Leave a Do Group)” and “ITER (Iterate)” for information on how those operations affect a DOWxx operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The DOWLT operation allows the operation within the DO group
C*  to be processed only if FLDA is less than FLDB.  If FLDA is
C*  not less than FLDB, the program branches to the operation
C*  immediately following the ENDDO operation.  If FLDA is less
C*  than FLDB, the operation within the DO group is processed.
C
C     FLDA          DOWLT     FLDB
C
C*
C*  The ENDDO operation causes the program to branch to the first
C*  DOWLT operation where a test is made to determine whether FLDA
C*  is less than FLDB.  This loop continues processing until FLDA
C*  is equal to or greater than FLDB; then the program branches
C*  to the operation immediately following the ENDDO operation.
C
C                   MULT      2.08          FLDA
C                   ENDDO
C
C*  In this example, multiple conditions are tested.  The combined
C*  DOWLT ORLT operation allows the operation within the DO group
C*  to be processed only while FLDA is less than FLDB or FLDC. If
C*  neither specified condition exists, the program branches to
C*  the operation immediately following the ENDDO operation. If
C*  either of the specified conditions exists, the operation after
C*  the ORLT operation is processed.
C
C     FLDA          DOWLT     FLDB
C     FLDA          ORLT      FLDC
C
C*  The ENDDO operation causes the program to branch to the second
C*  DOWLT operation where a test determines whether specified
C*  conditions exist.  This loop continues until FLDA is equal to
C*  or greater than FLDB and FLDC; then the program branches to the
C*  operation immediately following the ENDDO operation.
C
C                   MULT      2.08          FLDA
C                   ENDDO
```

Figure 134. DOWxx Operations

DSPLY (Display Function)

Code	Factor 1	Factor 2	Result Field	Indicators		
DSPLY	"Value" or "message" to display	Output queue	Response	_	ER	_

The DSPLY operation allows the program to communicate with the display work station that requested the program. Either factor 1, the result field or both must be specified. The operation can display a message and accept a response.

The value in factor 1 and possibly the result field are used to create the message to be displayed. If factor 1 is specified, it can contain a field name, a literal, a named constant, a table name, or an array element whose value is used to create the message to be displayed. Factor 1 can also contain *M, followed by a message identifier that identifies the message to be retrieved from the message file, QUSERMSG. Use the OVRMSGF CL command to use a different message file. QUSERMSG must be in a library in the library list of the job receiving the message.

The message identifier must be 7 characters in length consisting 3 alphabetic characters and four numeric characters (for example, *MUSR0001, this means message USR0001 is used).

If specified, factor 2 can contain a character field, a literal, a named constant, a table name, or an array element whose value is the symbolic name of the object meant to receive the message and from which the optional response can be sent. Any queue name except a program message queue name can be the value contained in the factor 2 entry. The queue must be declared to the OS/400 system before it can be used during program execution. (For information on how to create a queue, see the *CL Programming*.) There are two predefined queues:

Queue **Value**

QSYSOPR The message is sent to the system operator. Note that the QSYSOPR message queue severity level must be zero (00) to enable the DSPLY operation to immediately display a message to the system operator.

*EXT The message is sent to the external message queue.

Note: For a batch job, if factor 2 is blank, the default is QSYSOPR. For an interactive job, if factor 2 is blank, the default is *EXT.

The result field is optional. If it is specified, the response is placed in it. It can contain a field name, a table name, or an array element in which the response is placed. If no data is entered, the result field is unchanged.

Positions 73 and 74 can contain an indicator to be set on if an error occurs on the operation. If an error occurs when the workstation user enters the response, the message is displayed again a maximum of five times. After the fifth display, the indicator in positions 73 and 74, if specified, is set on. If this indicator is not specified, the exception/error handling routine receives control.

When you specify the DSPLY operation *with no message identifier in factor 1*, the operation functions as follows:

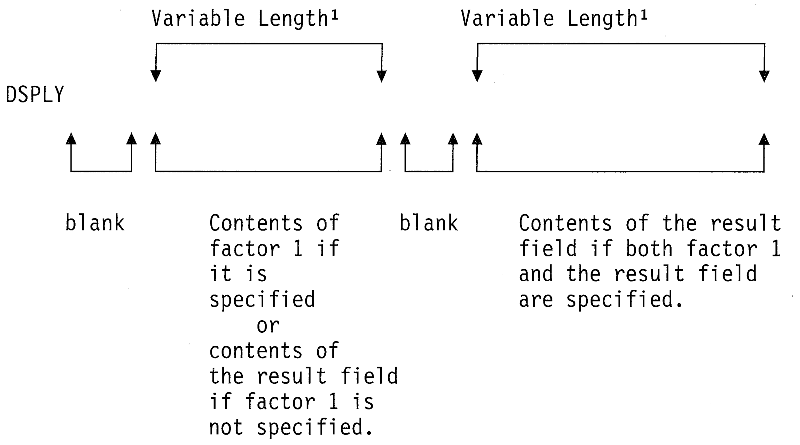
- When factor 1 contains an entry and the result field is blank, the contents of the factor 1 entry are displayed. The program does not wait for a response unless a display file with the parameter RSTDSP (*NO) specified was used to display a format at the workstation. Then the program waits for the user to press Enter.

- When factor 1 is blank and the result field contains an entry, the contents of the result field entry are displayed and the program waits for the user to enter data for the response. The reply is placed in the result field.
- When both factor 1 and the result field contain entries, the contents of the factor 1 and result field entries are combined and displayed. The program waits for the user to enter data for the response. The response is placed in the result field.
- If you request help on the message, you can find the type and attributes of the data that is expected and the number of invalid replies that have been entered.

  The maximum length of information that can be displayed is 52 bytes.

The format of the record written by the DSPLY operation with no message identifier in factor 1 follows:

[1]*The maximum length of information that can be displayed is 52 bytes.*


Variable Length[1]
Variable Length[1]
DSPLY
blank
Contents of factor 1 if it is specified or contents of the result field if factor 1 is not specified.
blank
Contents of the result field if both factor 1 and the result field are specified.

When you specify the DSPLY operation *with a message identifier in factor 1*, the operation functions as follows: the message identified by the factor 1 entry is retrieved from QUSERMSG, the message is displayed, and the program waits for the user to respond by entering data if the result field is specified. The response is placed in the result field. For information on how to format the display, see the *Data Management*.

When replying to a message, remember the following:

- Numeric fields sent to the display are right-adjusted and zero-suppressed.
- If a numeric field is entered with a length greater than the number of digits in the result field and the rightmost character is not a minus sign (-), an error is detected and a second wait occurs. The user must key in the field again.
- If graphic or character data is entered, the length must be equal or less than the receiving field length.
- If a Date, Time or Timestamp field is entered, the format and separator must match the format and separator of the result field. If the format or separator do not match, or the value is not valid (for example a date of 1999/99/99), an error is detected and a second wait occurs. The user must key in the field again.

- After 5 bad tries the DSPLY operation fails.
- To enter a null response to the system operator queue (QSYSOPR), the user must enter the characters *N and then press Enter.
- Character fields are padded on the right with blanks after all characters are entered.
- Numeric fields are right-adjusted and padded on the left with zeros after all characters are entered.
- Lowercase characters are not converted to uppercase.
- If factor 1 or the result field is of graphic data type, they will be bracketed by SO/SI when displayed. The SO/SI will be stripped from the value to be assigned to the graphic result field on input.

DUMP (Program Dump)

Code	Factor 1	Factor 2	Result Field	Indicators		
DUMP	Identifier					

The DUMP operation provides a dump (all fields, all files, indicators, data structures, arrays, and tables defined) of the program. It can be used independently or in combination with the OS/400 testing and debugging functions. In order for DUMP to be performed, the DBGVIEW(*NONE) compiler option must not be specified, as in this case the observability information required for DUMP is not included in the object. When the OPTIMIZE(*FULL) compiler option is selected, the field values shown in the dump may not reflect the actual content due to the effects of optimization.

The contents of the optional factor 1 identify the DUMP operation. It will replace the default heading on the dump listing if specified. It must contain a character or graphic entry that can be one of: a field name, literal, named constant, table name, or array element whose contents identify the dump. If factor 1 contains a graphic entry it is limited to 64 double byte characters. Factor 1 cannot contain a figurative constant.

The program continues processing the next calculation statement following the DUMP operation.

The DUMP operation is performed only if the DEBUG keyword is specified on the control specification. If the keyword is not specified, the DUMP operation is checked for errors and the statement is printed on the listing, but the DUMP operation is not processed.

When dumping files, the DUMP will dump the File Feedback Information section of the INFDS, but not the Open Feedback Information or the Input/Output Feedback Information sections of the INFDS. DUMP will instead dump the actual Open Feedback, and Device Feedback Information for the file.

Note that if the INFDS you have declared is not large enough to contain the Open Feedback, or Input/Output Feedback Information, then you do not have to worry about doing a POST before DUMP since the File Feedback Information in the INFDS is always up to date.

ELSE (Else)

Code	Factor 1	Factor 2	Result Field	Indicators		
ELSE						

The ELSE operation is an optional part of the IFxx and IF operations. If the IFxx comparison is met, the calculations before ELSE are processed; otherwise, the calculations after ELSE are processed.

Within total calculations, the control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry is for documentation purposes only. Conditioning indicator entries (positions 9 through 11) are not permitted.

To close the IFxx/ELSE group use an ENDIF operation.

Figure 142 on page 376 shows an example of an ELSE operation with an IFxx operation.

ENDyy (End a Structured Group)

Code	Factor 1	Factor 2	Result Field	Indicators		
END		Increment value				
ENDCS						
ENDDO		Increment value				
ENDIF						
ENDSL						

The ENDyy operation ends a CASxx, DO, DOU, DOW, DOUxx, DOWxx, IF, IFxx, or SELECT group of operations.

The ENDyy operations are listed below:

END End a CASxx, DO, DOU, DOUxx, DOW, DOWxx, IF, IFxx, or SELECT group
ENDCS End a CASxx group
ENDDO End a DO, DOU, DOUxx, DOW, or DOWxx group
ENDIF End an IF or IFxx group
ENDSL End a SELECT group

Factor 2 is allowed only on an ENDyy operation that delimits a DO group. It contains the incrementing value of the DO group. It can be positive or negative, must have zero decimal positions, and can be one of: an array element, table name, data structure, field, named constant, or numeric literal. If factor 2 is not specified on the ENDDO, the increment defaults to 1. If factor 2 is negative, the DO group will never end.

Conditioning indicators are optional for ENDDO and not allowed for ENDCS, ENDIF, and ENDSL.

Resulting indicators are not allowed. Factor 1, factor 2, and the result field must all be blank for ENDCS, ENDIF, and ENDSL.

If one ENDyy form is used with a different operation group (for example, ENDIF with a structured group), an error results at compilation time.

See the CASxx, DO, DOUxx, DOWxx, IFxx, and DOU, DOW, IF, and SELECT operations for examples that use the ENDyy operation.

ENDSR (End of Subroutine)

Code	Factor 1	Factor 2	Result Field	Indicators		
ENDSR	Label	Return point				

The ENDSR operation defines the end of an RPG IV subroutine and the return point to the main program. ENDSR must be the last statement in the subroutine. Factor 1 can contain a label that can be used as a point to which a GOTO operation within the subroutine can branch. The control level entry (positions 7 and 8) can be SR or blank. Conditioning indicator entries are not allowed.

The ENDSR operation ends a subroutine and causes a branch back to the statement immediately following the EXSR or CASxx operation unless the subroutine is a program exception/error subroutine (*PSSR) or a file exception/error subroutine (INFSR). For these subroutines, factor 2 of the ENDSR operation can contain an entry that specifies where control is to be returned following processing of the subroutine. This entry can be a field name that contains a reserved keyword or a literal or named constant that is a reserved keyword. If a return point that is not valid is specified, the RPG IV error handler receives control.

See "File Exception/Error Subroutine (INFSR)" on page 73 for more detail on return points.

See Figure 138 on page 369 for an example of coding an RPG IV subroutine.

EVAL (Evaluate expression)

Code	Factor 1	Factor 2
EVAL	Blank.	Assignment Statement

The EVAL operation code evaluates an assignment statement of the form result=expression. The expression is evaluated and the result placed in **result**. Therefore, **result** cannot be a literal or constant but must be a field name, array name, array element, data structure, data structure subfield, or a string using the %SUBST builtin function. The expression may yield any of the RPG data types. The type of the expression must be the same as the type of the result. A character or graphic result will be left justified and padded with blanks or truncated as required.

If the result represents an unindexed array or an array specified as array(*), the value of the expression is assigned to each element of the result, according to the rules described in the array chapter. Otherwise, the expression is evaluated once and the value is placed into each element of the array or sub-array. For numeric expressions, the half-adjust operation code extender is allowed. The rules for half adjusting are equivalent to those for the arithmetic operations.

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++

C*                   Assume FIELD1 = 10
C*                          FIELD2 =  9
C*                          FIELD3 =  8
C*                          FIELD4 =  7
C*                          ARR is defined with DIM(10)
C*                          *IN01 = *ON
C*                          A = 'abcdefghijklmno' (define as 15 long)
C*                          CHARFIELD1 = 'There'  (define as  5 long)
C*   The content of RESULT after the operation is 20
C                    EVAL     RESULT=FIELD1 + FIELD2+(FIELD3-FIELD4)
C*   The indicator *IN03  will be set to *TRUE
C                    EVAL     *IN03 = *IN01 OR (FIELD2 > FIELD3)
C*   Each element of array ARR will be assigned the value 72
C                    EVAL     ARR(*) = FIELD2 * FIELD3
C*   After the operation, the content of A = 'Hello There   '
C                    EVAL     A = 'Hello ' + CHARFIELD1
C*   After the operation the content of A = 'HelloThere    '
C                    EVAL     A = %TRIMR('Hello ') + %TRIML(CHARFIELD1)
C*   Date in assignment
C                    EVAL     ISODATE = DMYDATE
C*   Relational expression

C*   After the operation the value of *IN03 = *ON
C                    EVAL     *IN03 = FIELD3 > FIELD2
C*   Date in Relational expression
C                    EVAL     *IN05 = Date1  > Date2
C*   After the EVAL the original value of A contains 'ab****ghijklmno'
C                    EVAL     %SUBST(A:3:4) = '****'
C:
C*   After the EVAL PTR has the address of variable CHARFIELD1
C                    EVAL     PTR = %ADDR(CHARFIELD1)
C:
C*   An example to show that the result of a logical expression is
C*   Compatible with the character data type
C*   The following EVAL statement consisting of 3 logical expressions
C*   whose results are concatenated using the '+' operator
C*   The resulting value of the character field RES is '010'
C:
C                    EVAL     RES = (FIELD1<10) + *in01 + (field2 >= 17)
```

Figure 135. EVAL Operations

EXCEPT (Calculation Time Output)

Code	Factor 1	Factor 2	Result Field	Indicators		
EXCEPT		EXCEPT name				

The EXCEPT operation allows one or more records to be written during either detail calculations or total calculations.

See Figure 136 on page 365 for examples of the EXCEPT operation.

When specifying the EXCEPT operation remember:

- The exception records that are to be written during calculation time are indicated by an E in position 17 of the output specifications. An EXCEPT name, which is the same name as specified in factor 2 of an EXCEPT operation, can be specified in positions 30 through 39 of the output specifications of the exception records.
- Only exception records, not heading, detail, or total records, can contain an EXCEPT name.
- When the EXCEPT operation with a name in factor 2 is processed, only those exception records with the same EXCEPT name are checked and written if the conditioning indicators are satisfied.
- When factor 2 is blank, only those exception records with no name in positions 30 through 39 of the output specifications are checked and written if the conditioning indicators are satisfied.
- If an exception record is conditioned by an overflow indicator on the output specification, the record is written only during the overflow portion of the RPG IV cycle or during fetch overflow. The record is not written at the time the EXCEPT operation is processed.
- If an exception output is specified to a format that contains no fields, the following occurs:
  - If an output file is specified, a record is written with default values.
  - If a record is locked, the system treats the operation as a request to unlock the record. This is the alternative form of requesting an unlock. The preferred method is with the UNLOCK operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  When the EXCEPT operation with HDG specified in factor 2 is
C*  processed, all exception records with the EXCEPT name HDG are
C*  written.  In this example, UDATE and PAGE would be printed
C*  and then the printer would space 2 lines.
C*  The second HDG record would print a line of dots and then the
C*  printer would space 3 lines.
C*
C                   EXCEPT    HDG
C*
C*  When the EXCEPT operation with no entry in factor 2 is
C*  processed, all exception records that do not have an EXCEPT
C*  name specified in positions 30 through 39 are written if the
C*  conditioning indicators are satisfied.  Any exception records
C*  without conditioning indicators and without an EXCEPT name
C*  are always written by an EXCEPT operation with no entry in
C*  factor 2.  In this example,  if indicator 10 is on, TITLE and
C*  AUTH would be printed and then the printer would space 1 line.
C*
C                   EXCEPT
O*
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O
O          E    10                     1
O                       TITLE
O                       AUTH
O          E            HDG            2
O                       UDATE
O                       PAGE
O          E            HDG            3
O                                              '...............'
O                                              '...............'
O          E            DETAIL         1
O                       AUTH
O                       VERSNO
```

Figure 136. EXCEPT Operation with/without Factor 2 Specified

EXFMT (Write/Then Read Format)

Code	Factor 1	Factor 2	Result Field	Indicators		
EXFMT		Record format name		_	ER	_

The EXFMT operation is a combination of a WRITE followed by a READ to the same record format. EXFMT is valid only for a WORKSTN file defined as a full procedural (F in position 18 of the file description specifications) combined file (C in position 17 of the file description specifications) that is externally described (E in position 22 of the file description specifications)

Factor 2 must contain the name of the record format to be written and then read. A resulting indicator can be specified in positions 73 and 74 to be set on if the EXFMT operation is not completed successfully. When the indicator is set on, the read portion of the operation is not processed (record identifying indicators and fields are not modified). Positions 71, 72, 75, and 76 must be blank.

For the use of EXFMT with multiple device files, see the descriptions of the READ (by format name) and WRITE operations.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IPEASFRLen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
F*
F* PROMTD is a WORKSTN file which prompts the user for an option.
F* Based on what user enters, this program executes different
F* subroutines to add, delete, or change a record.
F*
FPROMTD     CF  E               WORKSTN
F*
F*
F* If user enters F3 function key, indicator *IN03 is set on and the
F* do while loop is exited.
F*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C                   DOW       *in03 = *off
C
C* EXFMT writes out the prompt to the screen and expects user to
C* enter an option. SCR1 is a record format name defined in the
C* WORKSTN file and OPT is a field defined in the record.
C
C                   EXFMT     SCR1
C                   SELECT
C                   WHEN      OPT = 'A'
C                   EXSR      ADDREC
C                   WHEN      OPT = 'D'
C                   EXSR      DELREC
C                   WHEN      OPT = 'C'
C                   EXSR      CHGREC
C                   ENDSL
C                   ENDDO
                      :
                      :
```

Figure 137. Example of EXFMT

EXSR (Invoke Subroutine)

Code	Factor 1	Factor 2	Result Field	Indicators		
EXSR		Subroutine name				

The EXSR operation causes the RPG IV subroutine named in factor 2 to be processed. The subroutine name must be a unique symbolic name and must appear as factor 1 of a BEGSR operation. The EXSR operation can appear anywhere in the calculation specifications. Whenever it appears, the subroutine that is named is processed. After operations in the subroutine are processed, the statement following the EXSR operation is processed except when a GOTO within the subroutine is given to a label outside the subroutine or when the subroutine is an exception/error subroutine with an entry in factor 2 of the ENDSR operation.

*PSSR used in factor 2 specifies that the program exception/error subroutine is to be processed. *INZSR used in factor 2 specifies that the program initialization subroutine is to be processed.

Coding Subroutines

An RPG IV subroutine can be processed from any point in the calculation operations. All RPG IV operations can be processed within a subroutine, and these operations can be conditioned by any valid indicators in positions 9 through 11. SR or blanks can appear in positions 7 and 8. Control level indicators (L1 through L9) cannot be used in these positions. However, AND/OR lines within the subroutine can be indicated in positions 7 and 8.

Fields used in a subroutine can be defined either in the subroutine or in the rest of the program. In either instance, the fields can be used by both the main program and the subroutine.

A subroutine cannot contain another subroutine. One subroutine can call another subroutine; that is, a subroutine can contain an EXSR or CASxx. However, an EXSR or CASxx specification within a subroutine cannot directly call itself. Indirect calls to itself through another subroutine should not be performed, because unpredictable results will occur. Use the GOTO and TAG operation codes if you want to branch to another point within the same subroutine.

Subroutines do not have to be specified in the order they are used. Each subroutine must have a unique symbolic name and must contain a BEGSR and an ENDSR statement.

The use of the GOTO (branching) operation is allowed within a subroutine. GOTO can specify the label on the ENDSR operation associated with that subroutine; it cannot specify the name of a BEGSR operation. A GOTO cannot be issued to a TAG or ENDSR within a subroutine unless the GOTO is in the same subroutine as the TAG or ENDSR." A GOTO within a subroutine can be issued to a TAG within either detail or total calculations.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* For a subroutine, positions 7 and 8 can be blank or contain SR.
C*
C                         :
C                         :
C                   EXSR      SUBRTB
C                         :
C                         :
C                         :
CL2                 EXSR      SUBRTA
C                         :
C                         :
C                         :
C     SUBRTA        BEGSR
C                         :
C                         :
C                         :
C*
C*  One subroutine can call another subroutine.
C*
C                   EXSR      SUBRTC
C                         :
C                         :
C                         :
C                   ENDSR
C     SUBRTB        BEGSR
C                         :
C                         :
C                         :
C*
```

Figure 138 (Part 1 of 2). Example of Coding Subroutines

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  GOTO and TAG operations can be used within a subroutine.
C*
C     START         TAG
C                         :
C                         :
C                         :
C   23              GOTO      END
C                         :
C                         :
C                         :
C   24              GOTO      START
C     END           ENDSR
C     SUBRTC        BEGSR
C                         :
C                         :
C                         :
C                   ENDSR
C*
```

Figure 138 (Part 2 of 2). Example of Coding Subroutines

EXTRCT (Extract Date/Time/Timestamp)

Code	Factor 1	Factor 2	Result Field	Indicators		
EXTRCT		Date/Time:Duration Code	Target	_	ER	_

The EXTRCT operation code will return one of:

- the year, month or day part of a date or timestamp field
- the hours, minutes or seconds part of a time or timestamp field
- the microseconds part of the timestamp field

to the field specified in the result field.

The Date, Time or Timestamp from which the information is required, is specified in factor 2, followed by the duration code. The entry specified in Factor-2 can be a field, subfield, array, array element, or a table. The duration code must be consistent with the Data type of Factor 2. See "Date Operations" on page 293 for valid duration codes.

Factor 1 must be blank.

The result field can be any numeric or character field, subfield, array/table element. The result field is cleared before the extracted data is assigned. For a character result field, the data is put left adjusted into the result field.

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....

C* Extract the month from a timestamp field to a 2-digit field
C* that is used as an index into a character array containing
C* the names of the months.  Then extract the day from the
C* timestamp to a 2-byte character field which can be used in
C* an EVAL concatenation expression to form a string containing
C* for example "March 13"AR
C
C                   EXTRCT    LOGONTIME:*M  LOGMONTH          2 0
C                   EXTRCT    LOGONTIME:*D  LOGDAY            2
C                   EVAL      DATE_STR = %TRIMR(MONTHS(LOGMONTH)
C                                        + ' ' + LOGDAY
```

Figure 139. EXTRCT Operations

FEOD (Force End of Data)

Code	Factor 1	Factor 2	Result Field	Indicators		
FEOD		File name		_	ER	_

The FEOD operation signals the logical end of data for a primary, secondary, or full procedural file. The FEOD function differs, depending on the file type and device. (For an explanation of how FEOD differs per file type and device, see the *DB2/400 Database Programming*, SC41-3701.)

FEOD differs from the CLOSE operation: the program is not disconnected from the device or file; the file can be used again for subsequent file operations without an explicit OPEN operation being specified to the file.

You can specify conditioning indicators. Factor 2 names the file to which FEOD is specified. You can specify a resulting indicator in positions 73 and 74 to be set on if the operation is not completed successfully.

To process any further sequential operations to the file after the FEOD operation (for example, READ or READP), you must reposition the file.

FORCE (Force a Certain File to Be Read Next Cycle)

Code	Factor 1	Factor 2	Result Field	Indicators		
FORCE		File name				

The FORCE operation allows selection of the file from which the next record is to be read. It can be used only for primary or secondary files.

Factor 2 must contain the name of a file from which the next record is to be selected.

If the FORCE operation is processed, the record is read at the start of the next program cycle. If more than one FORCE operation is processed during the same program cycle, all but the last is ignored. FORCE must be issued at *detail* time, not total time.

FORCE operations override the multifile processing method by which the program normally selects records. However, the first record to be processed is always selected by the normal method. The remaining records can be selected by FORCE operations. For information on how the FORCE operation affects match-field processing, see Figure 5 on page 16.

If FORCE is specified for a file that is at end of file, no record is retrieved from the file. The program cycle determines the next record to be read.

GOTO (Go To)

Code	Factor 1	Factor 2	Result Field	Indicators		
GOTO		Label				

The GOTO operation allows calculation operations to be skipped by instructing the program to go to (or branch to) another calculation operation in the program. A TAG operation names the destination of a GOTO operation. The TAG can either precede or follow the GOTO. Use a GOTO operation to specify a branch: A "TAG (Tag)" operation names the destination of a GOTO operation. Use a GOTO operation to specify a branch:

- From a detail calculation line to another detail calculation line
- From a total calculation line to another total calculation line
- From a detail calculation line to a total calculation line
- From a subroutine to a TAG or ENDSR within the same subroutine
- From a subroutine to a detail calculation line or to a total calculation line.

Branching from one part of the RPG IV logic cycle to another may result in an endless loop. You are responsible for ensuring that the logic of your program does not produce undesirable results.

Factor 2 must contain the label to which the program is to branch. This label is entered in factor 1 of a TAG or ENDSR operation. The label must be a unique symbolic name.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  If indicator 10, 15, or 20 is on, the program branches to
C*  the TAG label specified in the GOTO operations.
C*  A branch within detail calculations.
C   10              GOTO      RTN1
C*
C*  A branch from detail to total calculations.
C   15              GOTO      RTN2
C*
C     RTN1          TAG
C*
C                      :
C                      :
C:
C   20              GOTO      END
C*
C                      :
C                      :
C                      :
C     END           TAG
C*  A branch within total calculations.
CL1                 GOTO      RTN2
CL1                 :
CL1   RTN2          TAG
```

Figure 140. GOTO and TAG Operations

IF (If)

Code	Factor 1	Factor 2
IF	Blank	Expression

The IF operation code allows a series of operation codes to be processed if a condition is met. Its function is similar to that of the IFxx operation code. It differs in that the logical condition is expressed by an indicator valued expression in the extended-Factor 2 entry. The operations controlled by the IF operation are performed when the expression in the extended factor 2 field is true.

```
CL0N01Factor1+++++++OpcodeE+Extended-factor2++++++++++++++++++++++++++++++++
C                           Extended-factor2-continuation+++++++++++++++++++
C* The operations controlled by the IF operation are performed
C* when the expression is true.  That is A is greater than 10 and
C* indicator 20 is on.
C
C                   IF      A>10 AND *IN(20)
C                   :
C                   ENDIF
C*
C* The operations controlled by the IF operation are performed
C* when Date1 represents a later date then Date2
C
C                   IF      Date1 > Date2
C                   :
C                   ENDIF
C*
```

Figure 141. IF Operations

IFxx (If)

Code	Factor 1	Factor 2	Result Field	Indicators		
IFxx	Comparand	Comparand				

The IFxx operation allows a group of calculations to be processed if a certain relationship, specified by xx, exists between factor 1 and factor 2. When "ANDxx (And)" and "ORxx (Or)" operations are used with IFxx, the group of calculations is performed if the condition specified by the combined operations exists. (For the meaning of xx, see "Structured Programming Operations" on page 303.)

You can use conditioning indicators. Factor 1 and factor 2 must contain a literal, a named constant, a figurative constant, a table name, an array element, a data structure name, or a field name. Both the factor 1 and factor 2 entries must be of the same type.

If the relationship specified by the IFxx and any associated ANDxx or ORxx operations does not exist, control passes to the calculation operation immediately following the associated ENDIF operation. If an "ELSE (Else)" operation is specified as well, control passes to the first calculation operation that can be processed following the ELSE operation.

Conditioning indicator entries on the ENDIF operation associated with IFxx must be blank.

An ENDIF statement must be used to close an IFxx group. If an IFxx statement is followed by an ELSE statement, an ENDIF statement is required after the ELSE statement but not after the IFxx statement.

You have the option of indenting DO statements, IF-ELSE clauses, and SELECT-WHENxx-OTHER clauses in the compiler listing for readability. See the section on "Structured Programming" in the *ILE RPG/400 Programmer's Guide* for an explanation of how to indent statements in the source listing.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  If FLDA equals FLDB, the calculation after the IFEQ operation
C*  is processed.  If FLDA does not equal FLDB, the program
C*  branches to the operation immediately following the ENDIF.
C
C     FLDA          IFEQ      FLDB
C                       :
C                       :
C                       :
C                   ENDIF
C
C*  If FLDA equals FLDB, the calculation after the IFEQ operation
C*  is processed and control passes to the operation immediately
C*  following the ENDIF statement.  If FLDA does not equal FLDB,
C*  control passes to the ELSE statement and the calculation
C*  immediately following is processed.
C
C     FLDA          IFEQ      FLDB
C                       :
C                       :
C                       :
C                   ELSE
C                       :
C                       :
C                       :
C                   ENDIF
```

Figure 142 (Part 1 of 2). IFxx/ENDIF and IFxx/ELSE/ENDIF Operations

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  If FLDA is equal to FLDB and greater than FLDC, or, if FLDD
C*  is equal to FLDE and greater than FLDF, the calculation
C*  after the ANDGT operation is processed.  If neither of the
C*  specified conditions exists, the program branches to the
C*  operation immediately following the ENDIF statement.
C
C     FLDA          IFEQ      FLDB
C     FLDA          ANDGT     FLDC
C     FLDD          OREQ      FLDE
C     FLDD          ANDGT     FLDF
C                       :
C                       :
C                       :
C                   ENDIF
```

Figure 142 (Part 2 of 2). IFxx/ENDIF and IFxx/ELSE/ENDIF Operations

IN (Retrieve a Data Area)

Code	Factor 1	Factor 2	Result Field	Indicators		
IN	*LOCK	Data area name		_	ER	_

The IN operation retrieves a data area and optionally allows you to specify whether the data area is to be locked from update by another program. For a data area to be retrieved by the IN operation, it must be specified in the result field of an *DTAARA DEFINE statement or using the DTAARA keyword on the Definition specification. (See “DEFINE (Field Definition)” on page 342 for information on *DTAARA DEFINE operation and the Definition Specification for information on the DTAARA keyword).

Factor 1 can contain the reserved word *LOCK or can be blank. *LOCK indicates that the data area cannot be updated or locked by another program until (1) an “UNLOCK (Unlock a Data Area or Release a Record)” operation is processed, (2) an “OUT (Write a Data Area)” operation with no factor 1 entry is processed, or (3) the RPG IV program implicitly unlocks the data area when the program ends.

Factor 1 must be blank when factor 2 contains the name of the local data area or the Program Initialization Parameters (PIP) data area.

You can specify a *LOCK IN statement for a data area that the program has locked. When factor 1 is blank, the lock status is the same as it was before the data area was retrieved: If it was locked, it remains locked; if unlocked, it remains unlocked.

Factor 2 must be either the name of the result field used when you retrieved the data area or the reserved word *DTAARA. When *DTAARA is specified, all data areas defined in the program are retrieved. If an error occurs on the retrieval of a data area (for example, a data area can be retrieved but cannot be locked), an error occurs on the IN operation and the RPG IV exception/error handling routine receives control. If a message is issued to the requester, the message identifies the data area in error.

You can specify a resulting indicator in positions 73 and 74 to be set on if an error occurs during the operation. Positions 71-72 and 75-76 must be blank.

For further rules for the IN operation, see “Data-Area Operations” on page 292.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  TOTAMT, TOTGRS, and TOTNET are defined as data areas.  The IN
C*  operation retrieves all the data areas defined in the program
C*  and locks them.  The program processes calculations, and at
C*  LR time it writes and unlocks all the data areas.
C*  The data areas can then be used by other programs.
C*
C     *LOCK         IN        *DTAARA
C                   ADD       AMOUNT        TOTAMT
C                   ADD       GROSS         TOTGRS
C                   ADD       NET           TOTNET
C
CLR                 OUT       *DTAARA
C
C*
C* Define Data areas
C*
C     *DTAARA       DEFINE                  TOTAMT            8 2
C     *DTAARA       DEFINE                  TOTGRS           10 2
C     *DTAARA       DEFINE                  TOTNET           10 2
```

Figure 143. IN and OUT Operations

ITER (Iterate)

Code	Factor 1	Factor 2	Result Field	Indicators		
ITER						

The ITER operation transfers control from within a do group to the ENDDO statement of the do group. It can be used in DO, DOU, DOUxx, DOW, and DOWxx loops to transfer control immediately to a loop ENDDO statement. It causes the next iteration of the loop to be executed immediately. ITER affects the innermost loop.

If conditioning indicators are present on the ENDDO statement to which control is passed, and the condition is not satisfied, processing continues with the statement following the ENDDO operation.

The “LEAVE (Leave a Do Group)” operation is similar to the ITER operation; however, LEAVE transfers control to the statement ***following*** the ENDDO operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The following example uses a DOU loop containing a DOW loop.
C* The IF statement checks indicator 01.  If indicator 01 is ON,
C* the LEAVE operation is executed, transferring control out of
C* the innermost DOW loop to the Z-ADD instruction.  If indicator
C* 01 is not ON, subroutine PROC1 is processed.  Then indicator
C* 12 is checked.  If it is OFF, ITER transfers control to the
C* innermost ENDDO and the condition on the DOW is evaluated
C* again.  If indicator 12 is ON, subroutine PROC2 is processed.
C
C                   DOU       FLDA = FLDB
C                   :
C     NUM           DOWLT     10
C                   IF        *IN01
C                   LEAVE
C                   ENDIF
C                   EXSR      PROC1
C     *IN12         IFEQ      *OFF
C                   ITER
C                   ENDIF
C                   EXSR      PROC2
C                   ENDDO
C                   Z-ADD     20            RSLT              2 0
C                   :
C                   ENDDO
C                   :
```

Figure 144 (Part 1 of 2). ITER Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The following example uses a DOU loop containing a DOW loop.
C* The IF statement checks indicator 1. If indicator 1 is ON, the
C* MOVE operation is executed, followed by the LEAVE operation,
C* transferring control from the innermost DOW loop to the Z-ADD
C* instruction.  If indicator 1 is not ON, ITER transfers control
C* to the innermost ENDDO and the condition on the DOW is
C* evaluated again.
C                   :
C     FLDA          DOUEQ     FLDB
C                   :
C     NUM           DOWLT     10
C     *IN01         IFEQ      *ON
C                   MOVE      'UPDATE'      FIELD           20
C                   LEAVE
C                   ELSE
C                   ITER
C                   ENDIF
C                   ENDDO
C                   Z-ADD     20            RSLT            2 0
C                   :
C                   ENDDO
C                   :
```

Figure 144 (Part 2 of 2). ITER Operation

KFLD (Define Parts of a Key)

Code	Factor 1	Factor 2	Result Field	Indicators		
KFLD			Key field			

The KFLD operation is a declarative operation that indicates that a field is part of a search argument identified by a KLIST name.

The KFLD operation can be specified anywhere within calculations, including total calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. Conditioning indicator entries (positions 9 through 11) are not permitted.

The result field must contain the name of a field that is to be part of the search argument. The result field cannot contain an array name or a table name. Each KFLD field must agree in length, data type, and decimal position with the corresponding field in the composite key of the record or file. However, each KFLD field need not have the same name as the corresponding field in the composite key. The order the KFLD fields are specified in the KLIST determines which KFLD is associated with a particular field in the composite key. For example, the first KFLD field following a KLIST operation is associated with the leftmost (high-order) field of the composite key.

Figure 145 on page 383 shows an example of the KLIST operation with KFLD operations.

KLIST (Define a Composite Key)

Code	Factor 1	Factor 2	Result Field	Indicators		
KLIST	KLIST name					

The KLIST operation is a declarative operation that gives a name to a list of KFLDs. This list can be used as a search argument to retrieve records from files that have a composite key.

You can specify a KLIST anywhere within calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. Conditioning indicator entries (positions 9 through 11) are not permitted. Factor 1 must contain a unique name.

Remember the following when specifying a KLIST operation:

- If a search argument is composed of more than one field (a composite key), you must specify a KLIST with multiple KFLDs.
- A KLIST name can be specified as a search argument only for externally described files.
- A KLIST and its associated KFLD fields can appear anywhere in calculations.
- A KLIST must be followed immediately by at least one KFLD.
- A KLIST is ended when a non-KFLD operation is encountered.
- A KLIST name can appear in factor 1 of a CHAIN, DELETE, READE, READPE, SETGT, or SETLL operation.
- The same KLIST name can be used as the search argument for multiple files, or it can be used multiple times as the search argument for the same file.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
A*  DDS source
A          R RECORD
A            FLDA           4
A            SHIFT          1  0
A            FLDB          10
A            CLOCK#         5  0
A            FLDC          10
A            DEPT           4
A            FLDD           8
A          K DEPT
A          K SHIFT
A          K CLOCK#
A*
A*  End of DDS source
A*
A***********************************************************************

*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The KLIST operation indicates the name, FILEKY, by which the
C*  search argument can be specified.
C*
C     FILEKY        KLIST
C                   KFLD                    DEPT
C                   KFLD                    SHIFT
C                   KFLD                    CLOCK#
```

The following diagram shows what the search argument looks like.
The fields DEPT, SHIFT, and CLOCK# are
key fields in this record.

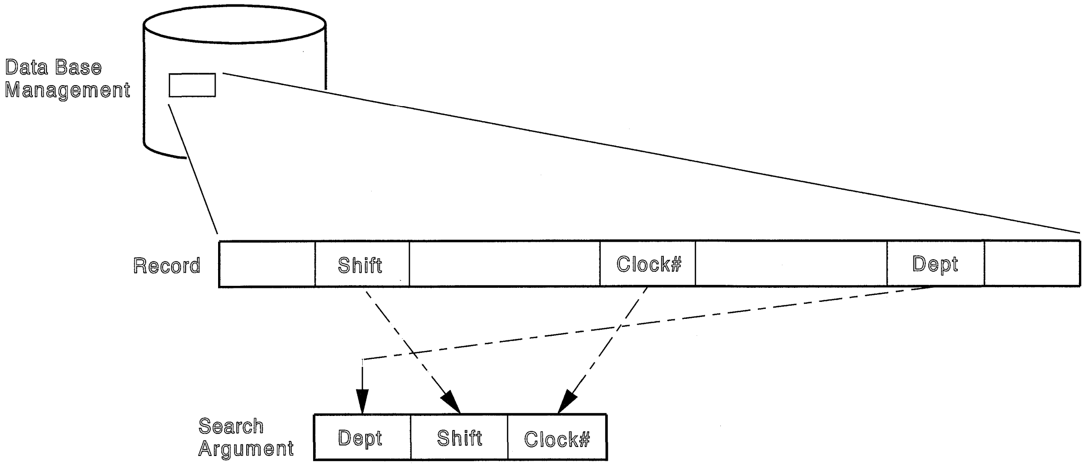


Figure 145. KLIST and KFLD Operations

LEAVE (Leave a Do Group)

Code	Factor 1	Factor 2	Result Field	Indicators		
LEAVE						

The LEAVE operation transfers control from within a do group to the statement following the ENDDO operation.

You can use LEAVE within a DO, DOU, DOUxx, DOW, or DOWxx loop to transfer control immediately from the innermost loop to the statement following the innermost loop's ENDDO operation. Using LEAVE to leave a do group does not increment the index.

In nested loops, LEAVE causes control to transfer "outwards" by one level only. LEAVE is not allowed outside a do group.

The "ITER (Iterate)" operation is similar to the LEAVE operation; however, ITER transfers control ***to*** the ENDDO statement.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The following example uses an infinite loop.  When the user
C* types 'q', control transfers to the LEAVE operation, which in
C* turn transfers control out of the loop to the Z-ADD operation.
C*
C     2             DOWNE     1
C                   :
C                   IF        ANSWER = 'q'
C                   LEAVE
C                   ENDIF
C                   :
C                   ENDDO
C                   Z-ADD     A             B
C*
C* The following example uses a DOUxx loop containing a DOWxx.
C* The IF statement checks indicator 1. If it is ON, indicator
C* 99 is turned ON, control passes to the LEAVE operation and
C* out of the inner DOWxx loop.
C*
C* A second LEAVE instruction is then executed because indicator 99
C* is ON, which in turn transfers control out of the DOUxx loop.
C*
C                   :
C     FLDA          DOUEQ     FLDB
C     NUM           DOWLT     10
C     *IN01         IFEQ      *ON
C                   SETON                                        99
C                   LEAVE
C                   :
C                   ENDIF
C                   ENDDO
C  99               LEAVE
C                   :
C                   ENDDO
C                   :
```

Figure 146. LEAVE Operation

LOOKUP (Look Up a Table or Array Element)

Code	Factor 1	Factor 2	Result Field	Indicators		
LOOKUP						
(array)	Search argument	Array name		HI	LO	EQ
(table)	Search argument	Table name	Table name	HI	LO	EQ

The LOOKUP operation causes a search to be made for a particular element in an array or table. Factor 1 is the search argument (data for which you want to find a match in the array or table named). It can be: a literal, a field name, an array element, a table name, a named constant, or a figurative constant. The nature of the comparison depends on the data type:

Character data If ALTSEQ(*EXT) is specified on the control specification, the alternate collating sequence is used for character LOOKUP. If ALTSEQ(*SRC) or no alternate sequence is specified, character LOOKUP does not use the alternate sequence.

Graphic data The comparison is hexadecimal; the alternate collating sequence is not used in any circumstance.

Numeric data Decimal alignment is not processed.

Other data types The considerations for comparison described in "Compare Operations" on page 291 apply to other types.

If a table is named in factor 1, the search argument used is the element of the table last selected in a LOOKUP operation, or it is the first element of the table if a previous LOOKUP has not been processed. The array or table to be searched is specified in factor 2.

For a table LOOKUP, the result field can contain the name of a second table from which an element (corresponding positionally with that of the first table) can be retrieved. The name of the second table can be used to reference the element retrieved. The result field must be blank if factor 2 contains an array name.

Resulting indicators specify the search condition for LOOKUP. One must be specified in positions 71 through 76 first to determine the search to be done and then to reflect the result of the search. Any specified indicator is set on only if the search is successful. No more than two indicators can be used. Resulting indicators can be assigned to equal and high or to equal and low. The program searches for an entry that satisfies either condition with equal given precedence; that is, if no equal entry is found, the nearest lower or nearest higher entry is selected.

Resulting indicators can be assigned to equal and low, or equal and high. The LOOKUP operation searches for an entry that satisfies either condition with equal given priority.

High (71-72): Instructs the program to find the entry that is nearest to, yet higher in sequence than, the search argument. The first higher entry found sets the indicator assigned to *high* on.
Low (73-74): Instructs the program to find the entry that is nearest to, yet lower in sequence than, the search argument. The first such entry found sets the indicator assigned to *low* on.
Equal (75-76): Instructs the program to find the entry equal to the search argument. The first equal entry found sets the indicator assigned to *equal* on.

When you use the LOOKUP operation, remember:

- The search argument and array or table must have the same type and length (except Time and Date fields which can have a different length).
- When LOOKUP is processed on an array and an index is used, the LOOKUP begins with the element specified by the index. The index value is set to the position number of the element located. An error occurs if the index is equal to zero or is higher than the number of elements in the array when the search begins. The index is set equal to one if the search is unsuccessful. If the index is a named constant, the index value will not change.
- A search can be made for high, low, high and equal, or low and equal only if a sequence is specified for the array or table on the definition specifications with the ASCEND or DESCEND keywords.
- No resulting indicator is set on if the search is not successful.
- If only an equal indicator (positions 75-76) is used, the LOOKUP operation will search the entire array or table. If your array or table is in ascending sequence and you want only an equal comparison, you can avoid searching the entire array or table by specifying a high indicator.
- The LOOKUP operation can produce unexpected results when the array is not in ascending or descending sequence.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  In this example, the programmer wants to know which element in
C*  ARY the LOOKUP operation locates.  The Z-ADD operation sets the
C*  field X to 1.  The LOOKUP starts at the element ARY that is
C*  indicated by field X and continues running until it finds the
C*  first element equal to SRCHWD.  The index value, X, is set to
C*  the position number of the element located.
C
C                   Z-ADD     1              X                 3 0
C     SRCHWD        LOOKUP    ARY(X)                                  26
C
C*  In this example, the programmer wants to know if an element
C*  is found that is equal to SRCHWD.  LOOKUP searches ARY until it
C*  finds the first element equal to SRCHWD.  When this occurs,
C*  indicator 26 is set on.
C
C     SRCHWD        LOOKUP    ARY                                     26
C
C*  The LOOKUP starts at a variable index number specified by field
C*  X.  Field X does not have to set to 1 before the LOOKUP
C*  operation.  When LOOKUP locates the first element in ARY equal
C*  to SRCHWD, indicator 26 is set on.  The index value, X, is
C*  set to the position number of the element located.
C*
C
C     SRCHWD        LOOKUP    ARY(X)                                  26
```

Figure 147. LOOKUP Operation with Arrays

MHHZO (Move High to High Zone)

Code	Factor 1	Factor 2	Result Field	Indicators		
MHHZO		Source field	Target field			

The MHHZO operation moves the zone portion of a character from the leftmost zone in factor 2 to the leftmost zone in the result field. Factor 2 and the result field must both be defined as character fields. For further information on the MHHZO operation, see “Move Zone Operations” on page 301.

The function of the MHHZO operation is shown in Figure 103 on page 302.

MHLZO (Move High to Low Zone)

Code	Factor 1	Factor 2	Result Field	Indicators		
MHLZO		Source field	Target field			

The MHLZO operation moves the zone portion of a character from the leftmost zone in factor 2 to the rightmost zone in the result field. Factor 2 must be defined as a character field. The result field can be character or numeric data. For further information on the MHLZO operation, see “Move Zone Operations” on page 301.

The function of the MHLZO operation is shown in Figure 103 on page 302.

MLHZO (Move Low to High Zone)

Code	Factor 1	Factor 2	Result Field	Indicators		
MLHZO		Source field	Target field			

The MLHZO operation moves the zone portion of a character from the rightmost zone in factor 2 to the leftmost zone in the result field. Factor 2 can be defined as a numeric field or as a character field, but the result field must be a character field. For further information on the MLHZO operation, see “Move Zone Operations” on page 301.

The function of the MLHZO operation is shown in Figure 103 on page 302.

MLLZO (Move Low to Low Zone)

Code	Factor 1	Factor 2	Result Field	Indicators		
MLLZO		Source field	Target field			

The MLLZO operation moves the zone portion of a character from the rightmost zone in factor 2 to the rightmost zone in the result field. Factor 2 and the result field can be either character data or numeric data. For further information on the MLLZO, see "Move Zone Operations" on page 301.

The function of the MLLZO operation is shown in Figure 103 on page 302.

MOVE (Move)

Code	Factor 1	Factor 2	Result Field	Indicators		
MOVE (P)	Date/Time Format	Source field	Target field	+	–	ZB

The MOVE operation transfers characters from factor 2 to the result field. Moving starts with the rightmost character of factor 2.

Factor 1 can contain a date or time format to specify the format of a character or numeric field that is the source or target of the operation.

When moving character, graphic or numeric data, if factor 2 is longer than the result field, the excess leftmost characters or digits of factor 2 are not moved. If the result field is longer than factor 2, the excess leftmost characters or digits in the result field are unchanged, unless padding is specified.

You cannot specify resulting indicators if the result field is an array; you can specify them if it is an array element, or a nonarray field.

If factor 2 is shorter than the length of the result field, a P specified in the operation extender position causes the result field to be padded on the left after the move occurs.

For further information on the MOVE operation, see “Move Operations” on page 297.

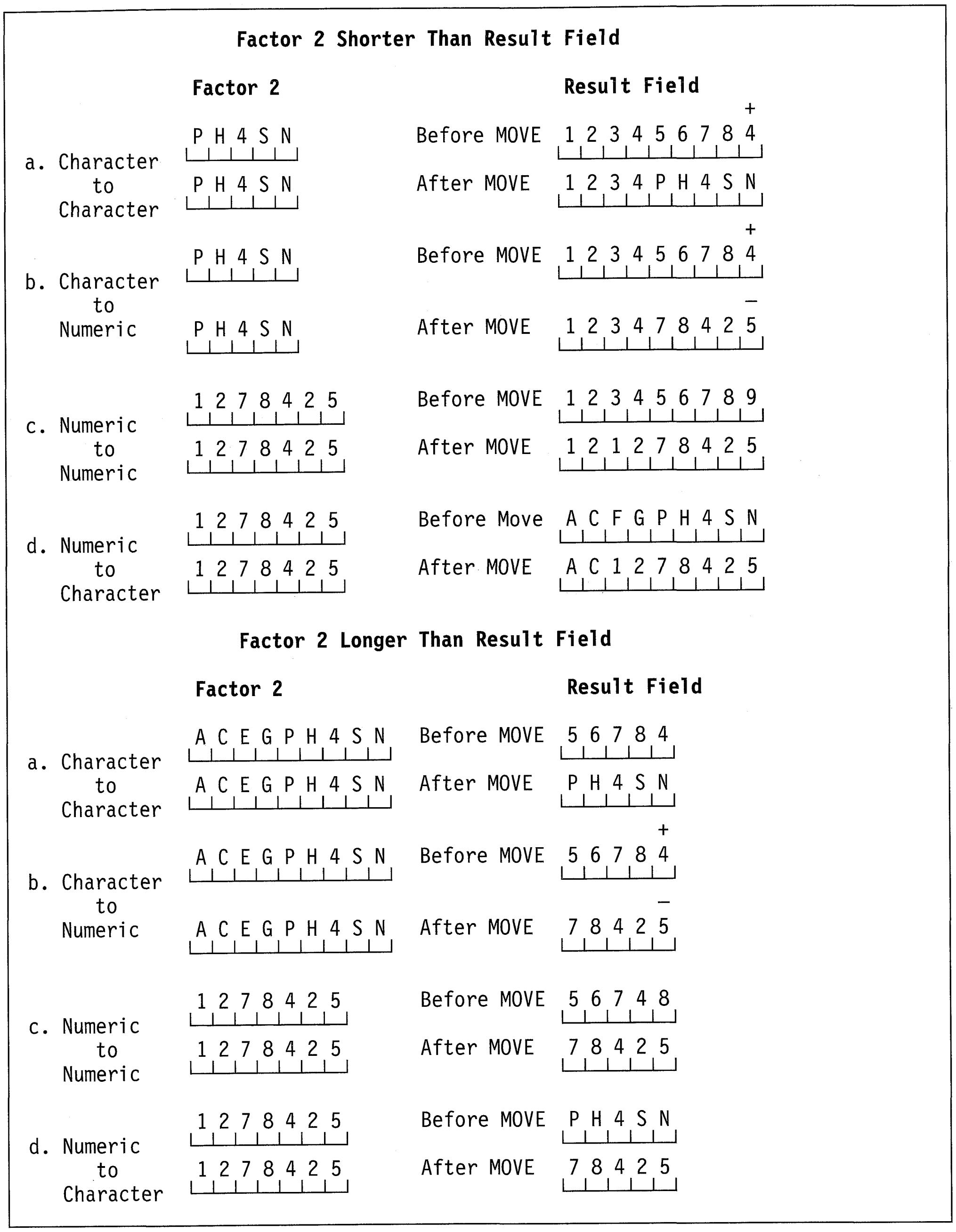


Figure 148 (Part 1 of 2). MOVE Operation

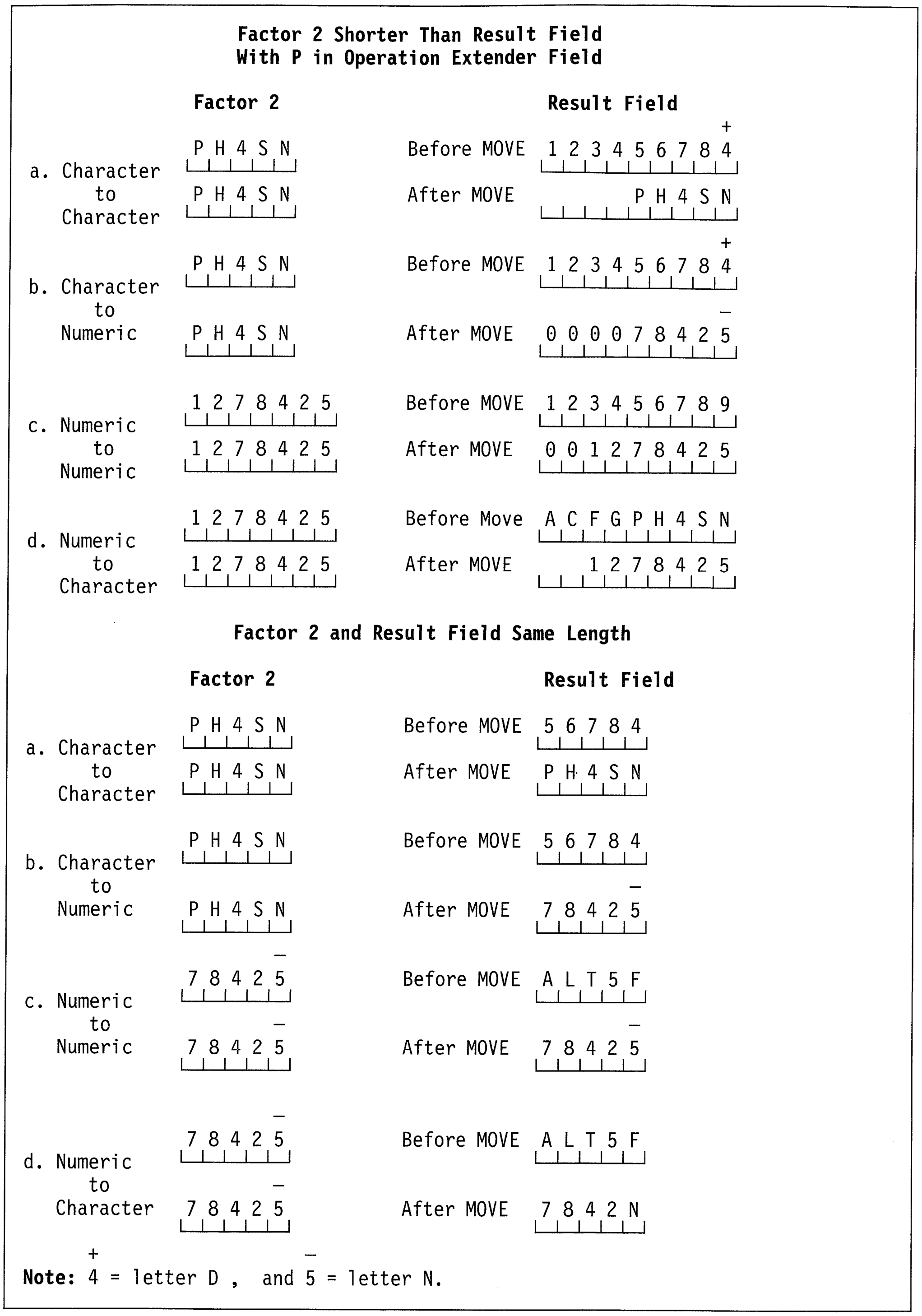


Figure 148 (Part 2 of 2). MOVE Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
H*  Control specification date format
H*
H DATFMT(*ISO)
H
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions+++++++++++++++++++++++++++
D
D DATE_ISO        S               D
D DATE_YMD        S               D   DATFMT(*YMD)
D                                     INZ(D'1992-03-24')
D DATE_JIS        S               D   DATFMT(*JIS)
D NUM_DATE        S              6P 0 INZ(210991)
D CHAR_DATE       S              8    INZ('02/01/53')
D DATE_EUR        S               D   DATFMT(*EUR)
D                                     INZ(D'1992-12-31')
D DATE_USA        S               D   DATFMT(*USA)
D

CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C* Move between Date fields. DATE_EUR will contain 24.03.1992
C*
C                   MOVE      DATE_YMD      DATE_EUR
C*
C* Convert numeric value in ddmmyy format into a *ISO Date.
C* DATE_ISO will contain 1991-09-21 after each of the 2 moves.
C*
C     *DMY          MOVE      210991        DATE_ISO
C     *DMY          MOVE      NUM_DATE      DATE_ISO
C*
C* Move a character value representing a *MDY date to a *JIS Date.
C* DATE_JIS will contain 1953-02-01 after each of the 2 moves.
C*
C     *MDY/         MOVE      '02/01/53'    DATE_JIS
C     *MDY/         MOVE      CHAR_DATE     DATE_JIS
C*
```

Figure 149 (Part 1 of 2). Move Date Operation

```
C*  Move a date field to a character field, using the
C*  date format and separators based on the jobattributes
C*
C     *JOBRUN       MOVE (P)  DATE_JIS      CHAR_DATE
C*
C*  Move a date field to a numeric field, using the
C*  date format based on the jobattributes
C*
C     *JOBRUN       MOVE (P)  DATE_JIS      NUM_DATE
C*
C*  DATE_USA will contain 12-31-9999
C*
C                   MOVE      *HIVAL        DATE_USA
C*
C*  Execution error, resulting in error code 114.  Year is not in
C*  1940-2039 date range.  DATE_YMD will be unchanged.
C*
C                   MOVE      DATE_USA      DATE_YMD
```

Figure 149 (Part 2 of 2). Move Date Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
H*  Control specification DATEDIT format
H*
H DATEDIT(*MDY)
H
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++
D
D*
D* Set the timestamp JOBSTART with the job start time and Date
D*
D Jobstart        S               Z
D Datestart       S               D
D Timestart       S               D
D Timebegin       S               T   inz(T'05:02:23')
D Datebegin       S               D   inz(D'1991-09-24')
D TmStamp         S               Z   inz
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C* stampchar will contain '1991-10-24-05.17.23.000000'
C* Factor 1 of the MOVE *DATE contains the same value specified
C* on the H-spec DATEDIT keyword
C*
C
C     *MDY          MOVE      *DATE         Datestart
C     *HMS          MOVE      *TIME         Timestart
C                   MOVE      Datestart     Jobstart
C                   MOVE      Timestart     Jobstart
C*
C* Assign a timestamp the value of a given time+15 minutes and
C* given date + 30 days. Move timestamp field to a character field
C* 'stampchar' will include all required separators
C
C                   ADDDUR    15:*minutes   Timebegin
C                   ADDDUR    30:*days      Datebegin
C                   MOVE      Timebegin     TmStamp
C                   MOVE      Datebegin     TmStamp
C                   MOVE      Stamp         stampchar   26
```

Figure 150. Example of MOVE with Timestamp

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions+++++++++++++++++++
D*
D* Example of MOVE between graphic and character fields
D*
D char_fld1      S             10A   inz('oK1K2K3  i')
D dbcs_fld1      S              4G
D char_fld2      S             10A   inz(*ALL'Z')
D dbcs_fld2      S              3G   inz(G'oK1K2K3i')
D*
C*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
C*
C* Value of dbcs_fld1 after MOVE operation is 'K1K2K3  '
C* Value of char_fld2 after MOVE oepration is 'ZZoK1K2K3i'
C*
C                   MOVE      char_fld1     dbcs_fld1
C                   MOVE      dbcs_fld2     char_fld2
```

Figure 151. MOVE between character and graphic fields

MOVEA (Move Array)

Code	Factor 1	Factor 2	Result Field	Indicators		
MOVEA (P)		Source	Target	+	–	ZB

The MOVEA operation transfers character, graphic or numeric values from factor 2 to the result field. (Certain restrictions apply when moving numeric values.) Factor 2 or the result field must contain an array. Factor 2 and the result field cannot specify the same array even if the array is indexed.

You can use MOVEA with a packed, binary, zoned, graphic or character array. You can:

- Move several contiguous array elements to a single field
- Move a single field to several contiguous array elements
- Move contiguous array elements to contiguous elements of another array.

Movement of data starts with the first element of an array if the array is not indexed or with the element specified if the array is indexed. The movement of data ends when the last array element is moved or filled. When the result field contains the indicator array, all indicators affected by the MOVEA operation are noted in the cross-reference listing.

The coding for and results of MOVEA operations are shown in Figure 152 on page 400.

Character and graphic MOVEA Operations

Both factor 2 and the result field must be defined as character or graphic.

On a character or graphic MOVEA operation, movement of data ends when the number of characters moved equals the shorter length of the fields specified by factor 2 and the result field; therefore, the MOVEA operation could end in the middle of an array element.

Numeric MOVEA Operations

Moves are only valid between fields and array elements with the same numeric length defined. Factor 2 and the result field entries can specify numeric fields, numeric array elements, or numeric arrays; at least one must be an array or array element. The numeric types can be binary, packed decimal, or zoned decimal but need not be the same between factor 2 and the result field.

Factor 2 can contain a numeric literal if the result field entry specifies a numeric array or numeric array-element:

- The numeric literal cannot contain a decimal point.
- The length of the numeric literal cannot be greater than the element length of the array or array element specified in the result field.

Decimal positions are ignored during the move and need not correspond. Numeric values are not converted to account for the differences in the defined number of decimal places.

The figurative constants *BLANK, *ALL, *ON and *OFF are not valid in factor 2 of a MOVEA operation on a numeric array.

General MOVEA Operations

If you need to use a MOVEA operation in your application, but restrictions on numeric MOVEA operations prevent you, you might be able to use character MOVEA operations. If the numeric array is in zoned decimal format:

- Define the numeric array as a subfield of a data structure
- Redefine the numeric array in the data structure as a character array.

If a figurative constant is specified with MOVEA, the length of the constant generated is equal to the portion of the array specified. For figurative constants in numeric arrays, the element boundaries are ignored except for the sign that is put in each array element. Examples are:

- MOVEA *BLANK ARR(X)

  Beginning with element X, the remainder of ARR will contain blanks.

- MOVEA *ALL'XYZ' ARR(X)

  ARR has 4-byte character elements. Element boundaries are ignored, as is always the case with character MOVEA. Beginning with element X, the remainder of the array will contain 'XYZXYZXYZXYZ. . .'.

For character, graphic and numeric MOVEA operations, you can specify a P operation extender to pad the result from the right.

For further information on the MOVEA operation, see "Move Operations" on page 297.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY
C*  Array-to-array move.  No indexing; different length array,
C*  same element length.
```

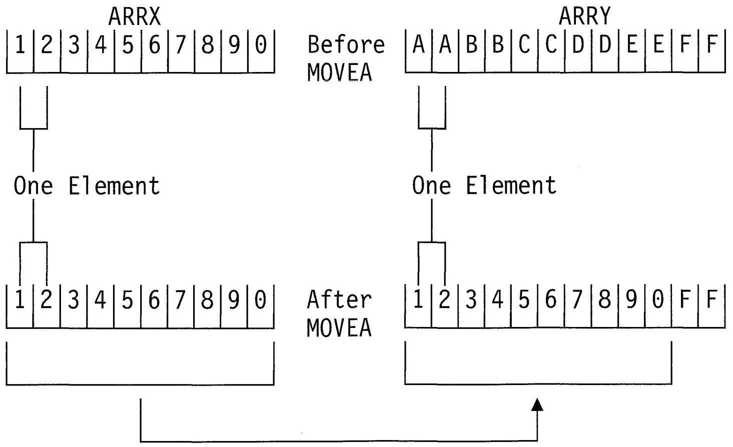


Figure 152 (Part 1 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY(3)
C*  Array-to-array move with index result field.
```

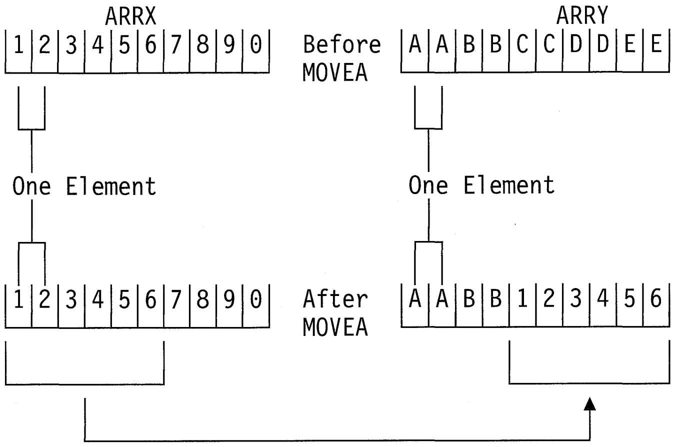


Figure 152 (Part 2 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY
C*  Array-to-array move, no indexing and different length array
C*  elements.
```

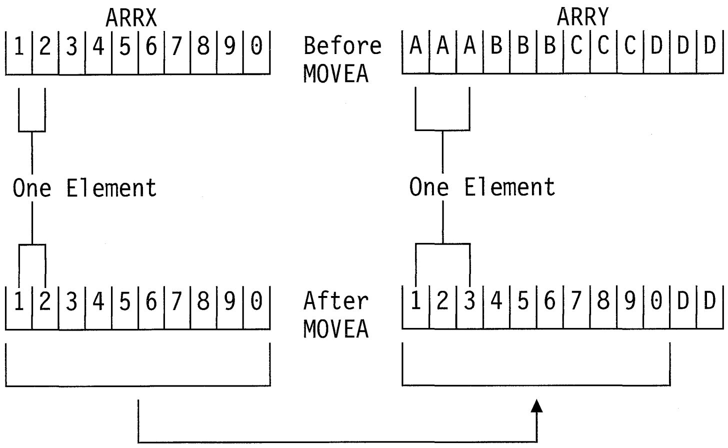


Figure 152 (Part 3 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX(4)       ARRY
C*  Array-to-array move, index factor 2 with different length array
C*  elements.
```

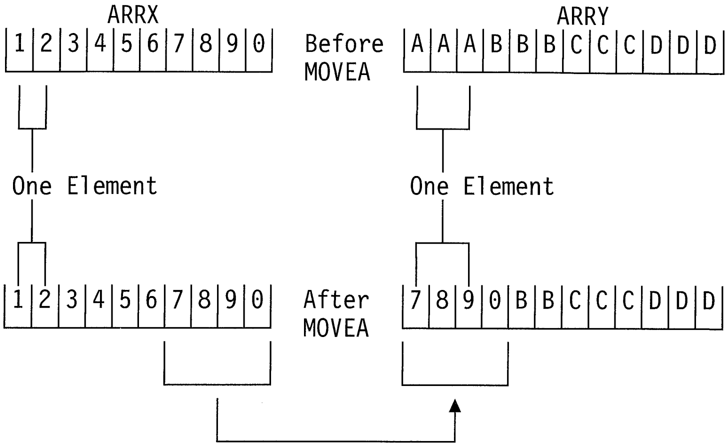


Figure 152 (Part 4 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     FIELDA        ARRY
C*  Field-to-array move, no indexing on array.
```

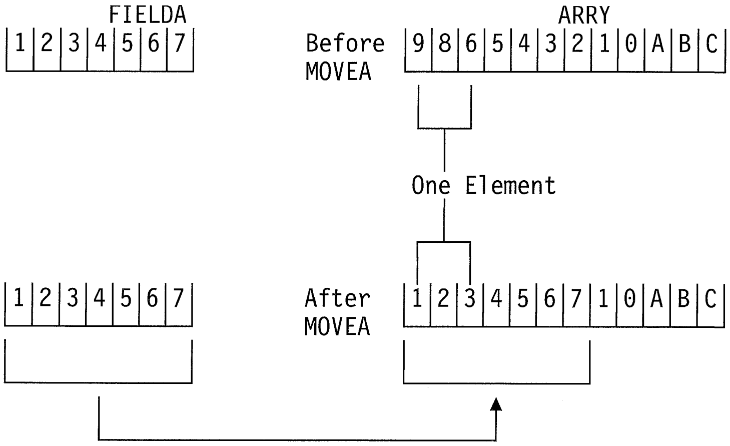


Figure 152 (Part 5 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* In the following example, N=3.  Array-to-field move with variable
C* indexing.
C                   MOVEA     ARRX(N)       FIELD
C*
```

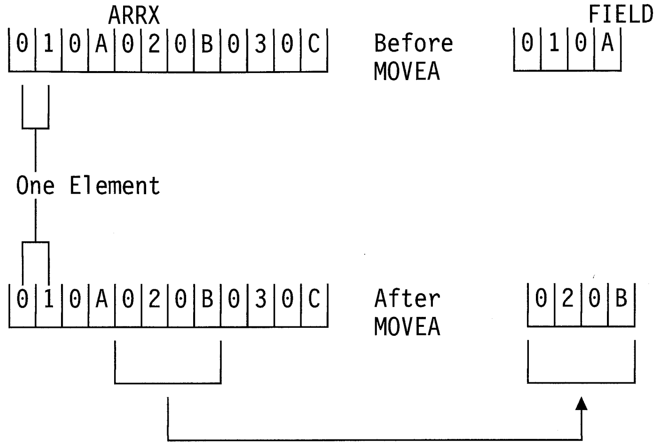


Figure 152 (Part 6 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRB          ARRZ
C*
C* An array-to-array move showing numeric elements.
```

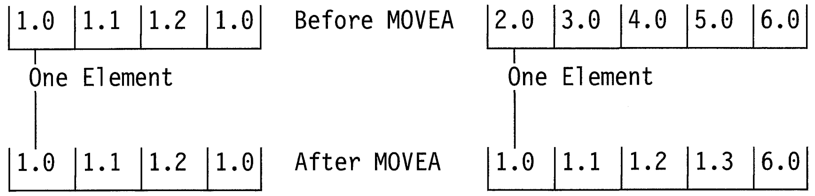


Figure 152 (Part 7 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRX          ARRY
C*  Array-to-array move with padding.  No indexing; different length
C*  array with same element length.
```

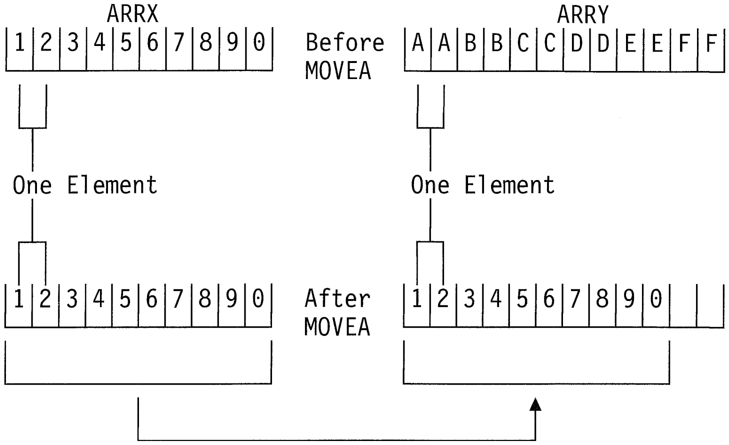


Figure 152 (Part 8 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRB          ARRZ
C*
C* An array-to-array move showing numeric elements with padding.
```

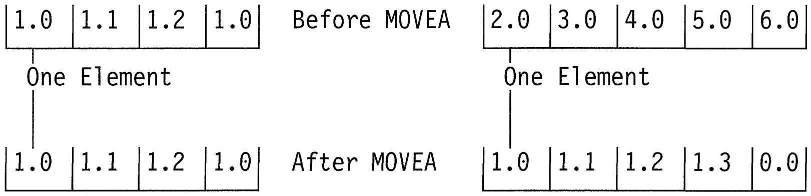


Figure 152 (Part 9 of 10). MOVEA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRX(3)       ARRY
C*  Array-to-array move with padding.  No indexing; different length
C*  array with different element length.
```

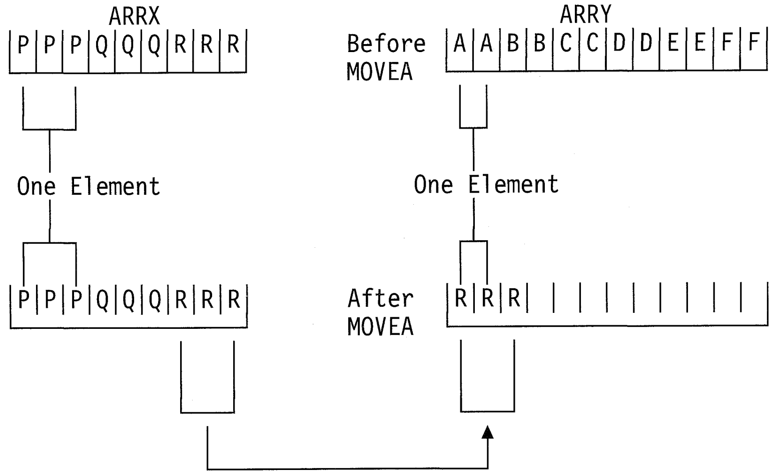


Figure 152 (Part 10 of 10). MOVEA Operation

MOVEL (Move Left)

Code	Factor 1	Factor 2	Result Field	Indicators		
MOVEL (P)	Date/Time Format	Source field	Target field	+	–	ZB

The MOVEL operation transfers characters from factor 2 to the result field. Moving begins with the leftmost character in factor 2. You cannot specify resulting indicators if the result field is an array. You can specify them if the result field is an array element, or a nonarray field.

When data is moved to a numeric field, the sign (+ or -) of the result field is retained except when factor 2 is as long as or longer than the result field. In this case, the sign of factor 2 is used as the sign of the result field.

Factor 1 can contain a date or time format to specify the format of a character or numeric field that is the source or target of the operation.

If factor 2 is longer than the result field, the excess rightmost characters of factor 2 are not moved. If the result field is longer than factor 2, the excess rightmost characters in the result field are unchanged, unless padding is specified.

The MOVEL operation is summarized in Figure 153 on page 408.

A summary of the rules for MOVEL operation for four conditions based on field lengths:

1. Factor 2 is the same length as the result field:
   a. If factor 2 and the result field are numeric, the sign is moved into the rightmost position.
   b. If factor 2 is numeric and the result field is character, the sign is moved into the rightmost position.
   c. If factor 2 is character and the result field is numeric, a minus zone is moved into the rightmost position of the result field if the zone from the rightmost position of factor 2 is a hexadecimal D (minus zone). However, if the zone from the rightmost position of factor 2 is not a hexadecimal D, a positive zone is moved into the rightmost position of the result field. Digit portions are converted to their corresponding numeric characters. If the digit portions are not valid digits, a data exception error occurs.
   d. If factor 2 and the result field are character, all characters are moved.
   e. If factor 2 and the result field are both graphic, all graphic characters are moved.
   f. If factor 2 is graphic and the result field is character, one graphic character will be lost, because 2 positions (bytes) in the character result field will be used to hold the SO/SI inserted by the compiler.
   g. If factor 2 is character and the result field is graphic, the factor 2 character data must be completely enclosed by one single pair of SO/SI. The SO/SI will be removed by the compiler before moving the data to the graphic result field.
2. Factor 2 is longer than the result field:
   a. If factor 2 and the result field are numeric, the sign from the rightmost position of factor 2 is moved into the rightmost position of the result field.
   b. If factor 2 is numeric and the result field is character, the result field contains only numeric characters.

c. If factor 2 is character and the result field is numeric, a minus zone is moved into the rightmost position of the result field if the zone from the rightmost position of factor 2 is a hexadecimal D (minus zone). However, if the zone from the rightmost position of factor 2 is not a hexadecimal D, a positive zone is moved into the rightmost position of the result field. Other result field positions contain only numeric characters.

d. If factor 2 and the result field are character, only the number of characters needed to fill the result field are moved.

e. If factor 2 and the result field are graphic, only the number of graphic characters needed to fill the result field are moved.

f. If factor 2 is graphic and the result field is character, the graphic data will be truncated and SO/SI will be inserted by the compiler.

g. If factor 2 is character and the result is graphic, the character data will be truncated. The character data must be completely enclosed by one single pair of SO/SI.

3. Factor 2 is shorter than the result field:

   a. If factor 2 is either numeric or character and the result field is numeric, the digit portion of factor 2 replaces the contents of the leftmost positions of the result field. The sign in the rightmost position of the result field is not changed.

   b. If factor 2 is either numeric or character and the result field is character data, the characters in factor 2 replace the equivalent number of leftmost positions in the result field. No change is made in the zone of the rightmost position of the result field.

   c. If factor 2 is graphic and the result field is character, the SO/SI are added immediately before and after the graphic data. This may cause unbalanced SO/SI in the character field due to residual data in the field, but this is users' responsibility.

   d. Notice that when moving from character to graphic field, the entire character field should be enclosed in SO/SI, e.g. if the character field length is 8, the character data in the field should be "oAABBƀƀi" (and not "oAABBiƀƀ").

4. Factor 2 is shorter than the result field and P is specified in the operation extender field:

   a. The move is performed as described above.

   b. The result field is padded from the right. See “Move Operations” on page 297 for more information on the rules for padding.

For further information on the MOVEL operation, see “Move Operations” on page 297.

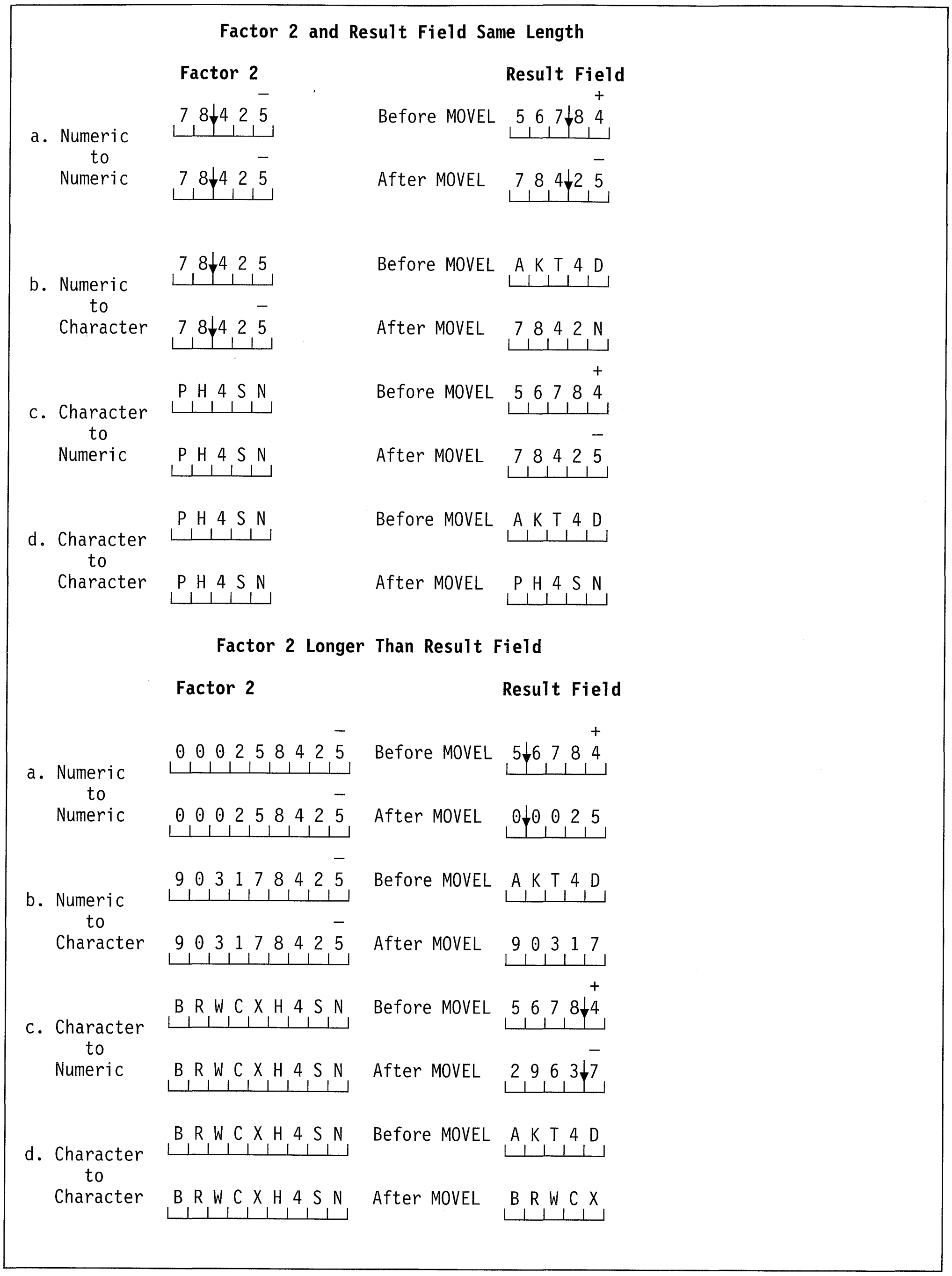
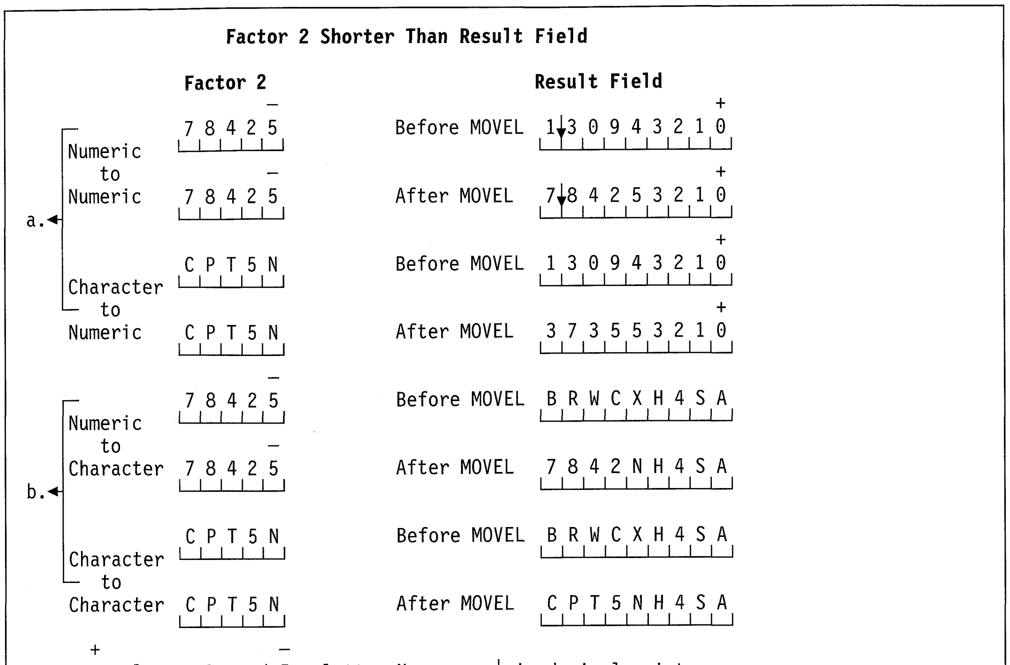
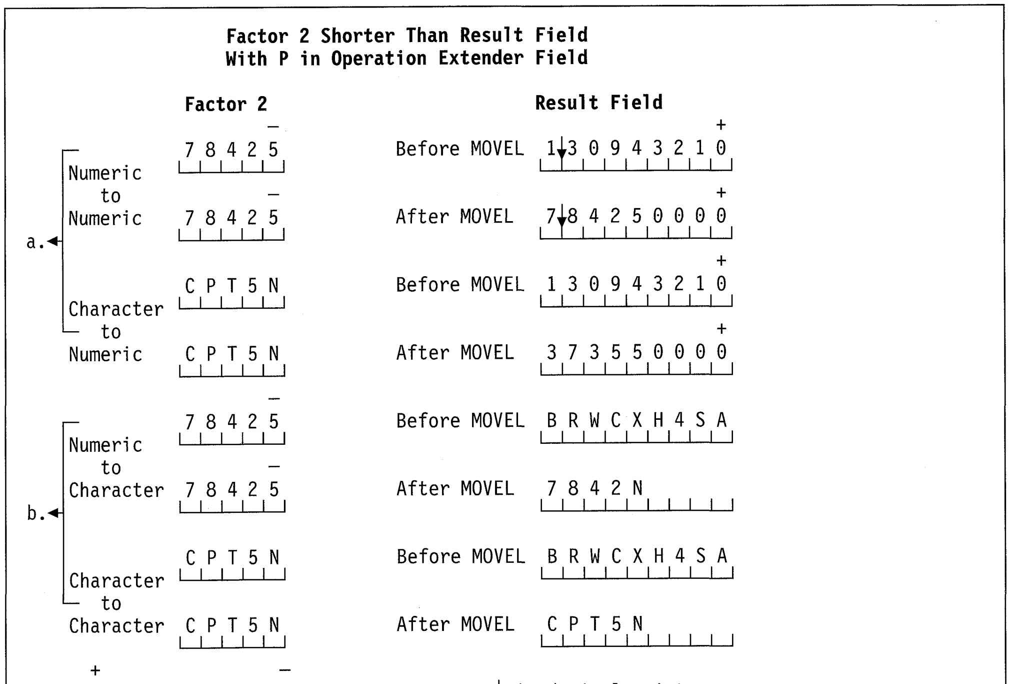


Figure 153 (Part 1 of 3). MOVEL Operation



Note: $\overset{+}{4}$ = letter D, and $\overset{-}{5}$ = letter N; arrow ↓ is decimal point.

Figure 153 (Part 2 of 3). MOVEL Operation



Note: $\overset{+}{4}$ = letter D, and $\overset{-}{5}$ = letter N; arrow ↓ is decimal point.

Figure 153 (Part 3 of 3). MOVEL Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D
D*
D* Example of MOVEL between graphic and character fields
D*
D char_fld1       S              8A   inz(' ')
D dbcs_fld1       S              4G   inz('oAABBCCDDi')
D char_fld2       S              4A   inz(' ')
D dbcs_fld2       S              3G   inz(G'oAABBCCi')
D char_fld3       S             10A   inz(*ALL'X')
D dbcs_fld3       S              3G   inz(G'oAABBCCi')
D char_fld4       S             10A   inz('oAABBCC  i')
D dbcs_fld4       S              2G
D*
C*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C*  The result field length is equal to the factor 2 length in bytes.
C*  One DBCS character is lost due to insertion of SO/SI.
C*  Value of char_fld1 after MOVEL operation is 'oAABBCCi'
C*
C                   MOVE      dbcs_fld1     char_fld1
C*
C*  Result field length shorter than factor 2 length. Truncation occurs.
C*  Value of char_fld2 after MOVEL operation is 'oAAi'
C*
C                   MOVE      dbcs_fld2     char_fld2
C*
C*  Result field length longer than factor 2 length. Example shows
C*  SO/SI are added immediately before and after graphic data.
C*  Before the MOVEL, Result Field contains 'XXXXXXXXXX'
C*  Value of char_fld3 after MOVEL operation is 'oAABBCCiXX'
C*
C                   MOVE      dbcs_fld3     char_fld3
C*
C*  Character to Graphic MOVEL
C*  Result Field shorter than Factor 2. Truncation occurs.
C*  Value of dbcs_fld4 after MOVEL operation is 'AABB'
C*
C                   MOVE      char_fld4     dbcs_fld4
```

Figure 154. MOVEL between character and graphic fields

MULT (Multiply)

Code	Factor 1	Factor 2	Result Field	Indicators		
MULT (H)	Multiplicand	Multiplier	Product	+	–	Z

If factor 1 is specified, factor 1 is multiplied by factor 2 and the product is placed in the result field. Be sure that the result field is large enough to hold it. Use the following rule to determine the maximum result field length: result field length equals the length of factor 1 plus the length of factor 2. If factor 1 is not specified, factor 2 is multiplied by the result field and the product is placed in the result field. Factor 1 and factor 2 must be numeric, and each can contain one of: an array, array element, field, figurative constant, literal, named constant, subfield, or table name. The result field must be numeric, but cannot be a named constant or literal. You can specify half adjust to have the result rounded.

For further information on the MULT operation, see "Arithmetic Operations" on page 287.

See Figure 100 on page 289 for examples of the MULT operation.

MVR (Move Remainder)

Code	Factor 1	Factor 2	Result Field	Indicators		
MVR			Remainder	+	–	Z

The MVR operation moves the remainder from the previous DIV operation to a separate field named in the result field. Factor 1 and factor 2 must be blank. The MVR operation must immediately follow the DIV operation. If you use conditioning indicators, ensure that the MVR operation is processed immediately after the DIV operation. If the MVR operation is processed before the DIV operation, undesirable results occur. The result field must be numeric and can contain one of: an array, array element, subfield, or table name.

Leave sufficient room in the result field if the DIV operation uses factors with decimal positions. The number of significant decimal positions is the greater of:

- The number of decimal positions in factor 1 of the previous divide operation
- The sum of the decimal positions in factor 2 and the result field of the previous divide operation.

The sign (+ or -) of the remainder is the same as the dividend (factor 1).

You cannot specify half adjust on a DIV operation that is immediately followed by an MVR operation.

The maximum number of whole number positions in the remainder is equal to the whole number of positions in factor 2 of the previous divide operation.

The MVR operation cannot be used if the previous divide operation has an array specified in the result field.

For further information on the MVR operation, see “Arithmetic Operations” on page 287.

See Figure 100 on page 289 for an example of the MVR operation.

NEXT (Next)

Code	Factor 1	Factor 2	Result Field	Indicators		
NEXT	Program device	File name		_	ER	_

The NEXT operation code forces the next input for a multiple device file to come from the program device specified in factor 1, providing the input operation is a cycle read or a READ-by-file-name. Any read operation, including CHAIN, EXFMT, READ, and READC, ends the effect of the previous NEXT operation. If NEXT is specified more than once between input operations, only the last operation is processed. The NEXT operation code can be used only for a multiple device file.

In factor 1, enter the name of a 10-character field that contains the program device name or a character literal or named constant that is the program device name. In factor 2, enter the name of the multiple device WORKSTN file for which the operation is requested.

You can specify an indicator in positions 73 and 74. It is set on if an exception/error occurs on the NEXT operation. If the INFSR subroutine is specified and positions 73 and 74 do not contain an indicator, the subroutine automatically receives control when an exception/error occurs. If the INFSR subroutine is not specified and positions 73 and 74 do not contain an indicator, the default error handler takes control.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....

C
C* Assume devices Dev1 and Dev2  are connected to the WORKSTN file
C* DEVICEFILE. The first READ reads data from DEV1, the second READ
C* reads data from DEV2. The NEXT operation will direct the program
C* to wait for data from the device specified in factor 1 (i.e. DEV1)
C* for the third READ
C
C                   READ      Devicefile                             99
C                   :
C                   READ      Devicefile                             99
C                   :
C     'DEV1'        NEXT
C                   :
C                   READ      Devicefile                             99
```

Figure 155. NEXT Operations

OCCUR (Set/Get Occurrence of a Data Structure)

Code	Factor 1	Factor 2	Result Field	Indicators		
OCCUR	Occurrence value	Data structure	Occurrence value	_	ER	_

The OCCUR operation code specifies the occurrence of the data structure that is to be used next within an RPG IV program.

The OCCUR operation establishes which occurrence of a multiple occurrence data structure is used next in a program. Only one occurrence can be used at a time. If a data structure with multiple occurrences or a subfield of that data structure is specified in an operation, the first occurrence of the data structure is used until an OCCUR operation is specified. After an OCCUR operation is specified, the occurrence of the data structure that was established by the OCCUR operation is used.

Factor 1 is optional; if specified, it can contain a numeric, zero decimal position literal, field name, named constant, or a data structure name. Factor 1 is used during the OCCUR operation to set the occurrence of the data structure specified in factor 2. If factor 1 is blank, the value of the current occurrence of the data structure in factor 2 is placed in the result field during the OCCUR operation.

If factor 1 is a data structure name, it must be a multiple occurrence data structure. The current occurrence of the data structure in factor 1 is used to set the occurrence of the data structure in factor 2.

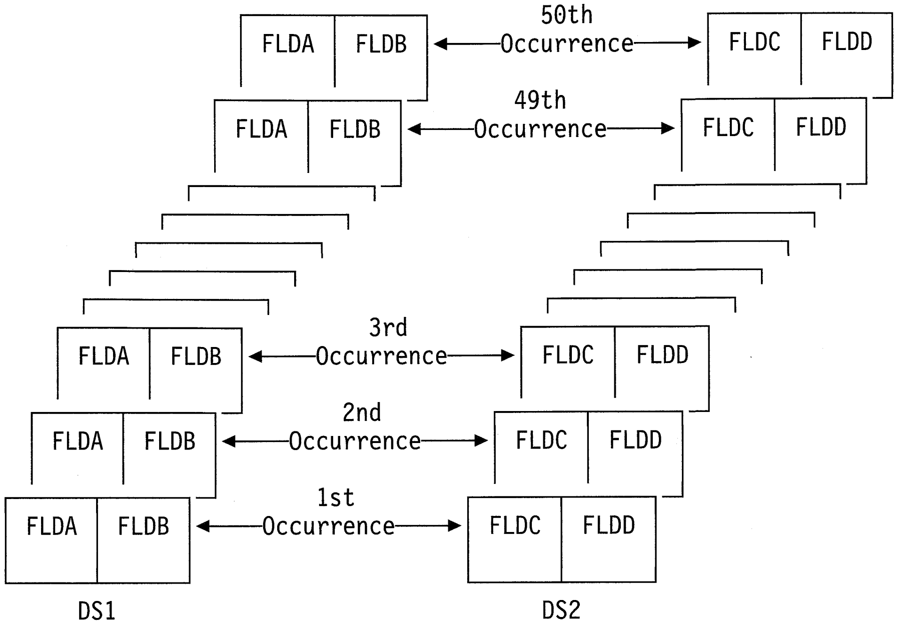
Factor 2 is required and must be the name of a multiple occurrence data structure.

The result field is optional; if specified, it must be a numeric field name with no decimal positions. During the OCCUR operation, the value of the current occurrence of the data structure specified in factor 2, after being set by any value or data structure that is optionally specified in factor 1, is placed in the result field.

At least one of factor 1 or the result field must be specified.

You can specify a resulting indicator in positions 73 and 74 to be set on if the occurrence specified is outside the valid range set for the data structure. If the occurrence is outside the valid range, the occurrence of the data structure in factor 2 remains the same as before the OCCUR operation was processed.

When a multiple-occurrence data structure is imported or exported, the information about the current occurrence is not imported or exported. See the “EXPORT” and “IMPORT” keywords for more information.



```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D*
D*  DS1 and DS2 are multiple occurrence data structures.
D*  Each data structure has 50 occurrences.
D DS1             DS                  OCCURS(50)
D  FLDA                   1      5
D  FLDB                   6     80
D*
D DS2             DS                  OCCURS(50)
D  FLDC                   1      6
D  FLDD                   7     11
```

Figure 156 (Part 1 of 2). Uses of the OCCUR Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*  DS1 is set to the third occurrence.  The subfields FLDA
C*  and FLDB of the third occurrence can now be used.  The MOVE
C*  and Z-ADD operations change the contents of FLDA and FLDB,
C*  respectively, in the third occurrence of DS1.
C
C     3             OCCUR     DS1
C                   MOVE      'ABCDE'       FLDA
C                   Z-ADD     22            FLDB
C*
C*  DS1 is set to the fourth occurrence.  Using the values in
C*  FLDA and FLDB of the fourth occurrence of DS1, the MOVE
C*  operation places the contents of FLDA in the result field,
C*  FLDX, and the Z-ADD operation places the contents of FLDB
C*  in the result field, FLDY.
C
C     4             OCCUR     DS1
C                   MOVE      FLDA          FLDX
C                   Z-ADD     FLDB          FLDY
C*
C*  DS1 is set to the occurrence specified in field X.
C*  For example, if X = 10, DS1 is set to the tenth occurrence.
C     X             OCCUR     DS1
C*
C*  DS1 is set to the current occurrence of DS2.  For example, if
C*  the current occurrence of DS2 is the twelfth occurrence, DSI
C*  is set to the twelfth occurrence.
C     DS2           OCCUR     DS1
C*
C*  The value of the current occurrence of DS1 is placed in the
C*  result field, Z.  Field Z must be numeric with zero decimal
C*  positions.  For example, if the current occurrence of DS1
C*  is 15, field Z contains the value 15.
C                   OCCUR     DS1           Z
C
C*  DS1 is set to the current occurrence of DS2.  The value of the
C*  current occurrence of DS1 is then moved to the result field,
C*  Z.  For example, if the current occurrence of DS2 is the fifth
C*  occurrence, DS1 is set to the fifth occurrence.  The result
C*  field, Z, contains the value 5.
C
C     DS2           OCCUR     DS1           Z
C*
C*  DS1 is set to the current occurrence of X.  For example, if
C*  X = 15, DS1 is set to the fifteenth occurrence.  If X equals
C*  0 or is greater than 50, an error occurs and indicator 20 is
C*  set on.  If indicator 20 is on, the LR indicator is set on.
C
C     X             OCCUR     DS1                                  20
C
C     *IN20         IFEQ      *ON
C                   SETON                                        LR
C                   END
```

Figure 156 (Part 2 of 2). Uses of the OCCUR Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D*
D* Procedure P1 exports a multiple occurrence data structure.
D* Since the information about the current occurrence is
D* not exported, P1 can communicate this information to
D* other procedures using parameters, but in this case it
D* communicates this information by exporting the current
D* occurrence.
D*
D EXP_DS          DS                  OCCURS(50) EXPORT
D  FLDA                   1      5
D NUM_OCCUR       C                   %ELEM(EXP_DS)
D EXP_DS_CUR      S              5P 0 EXPORT
D*
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
C*
C* Loop through the occurrences.  For each occurrence, call
C* procedure P2 to process the occurrence. Since the occurrence
C* number EXP_DS_CUR is exported, P2 will know which occurrence
C* to process.
C*
C                   DO        NUM_OCCUR     EXP_DS_CUR
C     EXP_DS_CUR    OCCUR     EXP_DS
C                     :
C                   CALLB     'P2'
C                   ENDDO
C                     :
```

Figure 157 (Part 1 of 2). Exporting a Multiple Occurrence DS

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D*
D* Procedure P2 imports the multiple occurrence data structure.
D* The current occurrence is also imported.
D*
D EXP_DS          DS                  OCCURS(50) IMPORT
D  FLDA                   1      5
D EXP_DS_CUR      S              5P 0 IMPORT
D*
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
C*
C* Set the imported multiple-occurrence data structure using
C* the imported current occurrence.
C*
C     EXP_DS_CUR    OCCUR     EXP_DS
C*
C* Process the current occurrence.
C                     :
```

Figure 157 (Part 2 of 2). Exporting a Multiple Occurrence DS

OPEN (Open File for Processing)

Code	Factor 1	Factor 2	Result Field	Indicators		
OPEN		File name		_	ER	_

The explicit OPEN operation opens the file named in factor 2. The factor 2 entry cannot be designated as a primary, secondary, or table file. You can specify a resulting indicator in positions 73 and 74 to be set on if the OPEN operation is not successful. If no indicator is specified, but the INFSR subroutine is specified, the INFSR automatically receives control when an error/exception occurs. If no indicator or INFSR subroutine is specified, the default error/exception handler receives control when an error/exception occurs.

To open the file specified in factor 2 for the first time in a program with an explicit OPEN operation, specify the USROPN keyword on the file description specifications. (See Chapter 15, “File Description Specifications” for restrictions when using the USROPN keyword.)

If a file is opened and later closed by the CLOSE operation in the program, the programmer can reopen the file with the OPEN operation and the USROPN keyword on the file description specification is not required. When the USROPN keyword is not specified on the file description specification, the file is opened at program initialization. If an OPEN operation is specified for a file that is already open, an error occurs.

Multiple OPEN operations in a program to the same file are valid as long as the file is closed when the OPEN operation is issued to it.

When you open a file with the DEVID keyword specified (on the file description specifications), the fieldname specified as a parameter on the DEVID keyword is set to blanks. See the description of the DEVID keyword, in Chapter 15, “File Description Specifications.”

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
F
FEXCEPTN   O    E             DISK    USROPN
FFILEX     F    E             DISK
F
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The explicit OPEN operation opens the EXCEPTN file for
C*  processing if indicator 97 is on and indicator 98 is off.
C*  Note that the EXCEPTN file on the file description
C*  specifications has the USROPN keyword specified.
C*  Indicator 99 will be set on if the OPEN operation fails
C*
C                   IF        *in97 and not *in98
C                   OPEN      EXCEPTN                               99
C                   IF        not *in99
C                   WRITE     ERREC
C                   ENDIF
C                   ENDIF
C*
C*  FILEX is opened at program initialization.  The explicit
C*  CLOSE operation closes FILEX before control is passed to RTNX.
C*  RTNX or another program can open and use FILEX.  Upon return,
C*  the OPEN operation reopens the file.  Because the USROPN
C*  keyword is not specified for FILEX, the file is opened at
C*  program initialization
C*
C                   CLOSE     FILEX
C                   CALL      'RTNX'
C                   OPEN      FILEX
```

Figure 158. OPEN Operation with CLOSE Operation

ORxx (Or)

Code	Factor 1	Factor 2	Result Field	Indicators		
ORxx	Comparand	Comparand				

The ORxx operation is optional with the DOUxx, DOWxx, IFxx, WHENxx, and ANDxx operations. ORxx is specified immediately following a DOUxx, DOWxx, IFxx, WHENxx, ANDxx or ORxx statement. Use ORxx to specify a more complex condition for the DOUxx, DOWxx, IFxx, and WHENxx operations.

The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry must be the same as the entry for the associated DOUxx, DOWxx, IFxx, or WHENxx operation. Conditioning indicator entries (positions 9 through 11) are not allowed.

Factor 1 and factor 2 must contain a literal, a named constant, a figurative constant, a table name, an array element, a data structure name, or a field name. Factor 1 and factor 2 must be of the same type. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See "Compare Operations" on page 291.

Figure 132 on page 351 shows an example of ORxx and ANDxx operations with a DOUxx operation.

OTHER (Otherwise Select)

Code	Factor 1	Factor 2	Result Field	Indicators		
OTHER						

The OTHER operation begins the sequence of operations to be processed if no WHENxx or "WHEN (When True Then Select)" condition is satisfied in a SELECT group. The sequence ends with the ENDSL or END operation.

Rules to remember when using the OTHER operation:

- The OTHER operation is optional in a SELECT group.
- Only one OTHER operation can be specified in a SELECT group.
- No WHENxx or WHEN operation can be specified after an OTHER operation in the same SELECT group.
- The sequence of calculation operations in the OTHER group can be empty; the effect is the same as not specifying an OTHER statement.
- Within total calculations, the control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry is for documentation purposes only. Conditioning indicator entries (positions 9 through 11) are not allowed.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* Example of a SELECT group with WHENxx and OTHER.  If X equals 1,
C* do the operations in sequence 1; if X does not equal 1 and Y
C* equals 2, do the operations in sequence 2.  If neither
C* condition is true, do the operations in sequence 3.
C*
C                   SELECT
C     X             WHENEQ    1
C*
C* Sequence 1
C*
C                      :
C                      :
C     Y             WHENEQ    2
C*
C* Sequence 2
C*
C                      :
C                      :
C                   OTHER
C*
C* Sequence 3
C*
C                      :
C                      :
C                   ENDSL
```

Figure 159. OTHER Operation

For more details and examples, see the SELECT and WHENxx operations.

OUT (Write a Data Area)

Code	Factor 1	Factor 2	Result Field	Indicators		
OUT	*LOCK	Data area name		_	ER	_

The OUT operation updates the data area specified in factor 2. To specify a data area in factor 2 of an OUT operation, you must ensure two things:

- The data area must also be specified in the result field of a *DTAARA DEFINE statement, or defined using the DTAARA keyword on the Definition specification.
- The data area must have been locked previously by a *LOCK IN statement or it must have been specified as a data area data structure by a U in position 23 of the definition specifications. (The RPG IV language implicitly retrieves and locks data area data structures at program initialization.)

Factor 1 can contain the reserved word *LOCK or can be blank. When factor 1 contains *LOCK, the data area remains locked after it is updated. When factor 1 is blank, the data area is unlocked after it is updated.

Factor 1 must be blank when factor 2 contains the name of the local data area or the Program Initialization Parameters (PIP) data area.

Factor 2 must be either the name of the result field used when you retrieved the data area or the reserved word *DTAARA. When *DTAARA is specified, all data areas defined in the program are updated. If an error occurs when one or more data areas are updated (for example, if you specify an OUT operation to a data area that has not been locked by the program), an error occurs on the OUT operation and the RPG IV exception/error handling routine receives control. If a message is issued to the requester, the message identifies the data area in error.

You can specify a resulting indicator in positions 73 and 74 to be set on if an error occurs during the operation. Positions 71-72 and 75-76 must be blank.

For further rules for the OUT operation, see “Data-Area Operations” on page 292.

See Figure 143 on page 378 for an example of the OUT operation.

PARM (Identify Parameters)

Code	Factor 1	Factor 2	Result Field	Indicators		
PARM	Target field	Source field	Parameter			

The declarative PARM operation defines the parameters that compose a parameter list (PLIST). PARM operations can appear anywhere in calculations as long as they immediately follow the PLIST, CALL, or CALLB operation they refer to. PARM statements must be in the order expected by the called program or procedure. One PARM statement, or as many as 255 for a CALL or 399 for a CALLB or PLIST are allowed.

The PARM operation can be specified anywhere within calculations, including total calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement in the appropriate section of the program. Conditioning indicator entries (positions 9 through 11) are not allowed.

Factor 1 and factor 2 entries are optional. If specified, the entries must be the same type as specified in the result field. A literal or named constant cannot be specified in factor 1. Factor 1 and factor 2 must be blank if the result field contains the name of a multiple-occurrence data structure.

The result field must contain the name of a:

- For all PARM statements:
  - Field
  - Data structure
  - Array
- For non-*ENTRY PLIST PARM statements it can also contain:
  - Array element
  - *OMIT (CALLB only)

    If *OMIT is specified, factor 1 and factor 2 must be blank, no length and decimal positions can be specified.

The Result-Field entry of a PARM operation in the *ENTRY PLIST cannot contain:

- *IN, *INxx, *IN(xx)
- *OMIT
- A label
- A literal
- A named constant
- A data-area name
- A data-area data structure name
- A globally initialized data structure
- A data structure with initialized subfields
- A data structure with a compile time array as a subfield
- A table name
- Fields or data structures defined with the keywords BASED, IMPORT, or EXPORT
- An array element
- A data-structure subfield name

- The name of a compile-time array
- The name of a program status or file information data structure (INFDS)

A field name can be specified only once in an *ENTRY PLIST.

If an array is specified in the result field, the area defined for the array is passed to the called program or procedure. When a data structure with multiple occurrences is passed to the called program or procedure, all occurrences of the data structure are passed as a single field. However, if a subfield of a multiple occurrence data structure is specified in the result field, only the current occurrence of the subfield is passed to the called program or procedure.

Each parameter field has only one storage location; it is in the calling program or procedure. The address of the storage location of the result field is passed to the called program or procedure on a PARM operation. If the called program or procedure changes the value of a parameter, it changes the data at that storage location. When control returns to the calling program or procedure, the parameter in the calling program or procedure (that is, the result field) has changed. Even if the called program or procedure ends in error after it changes the value of a parameter, the changed value exists in the calling program or procedure. To preserve the information passed to the called program or procedure for later use, specify in factor 2 the name of the field that contains the information you want to pass to the called program or procedure. Factor 2 is copied into the result field, and the storage address of the result field is passed to the called program or procedure.

Because the parameter fields are accessed by address, not field name, the calling and called parameters do not have to use the same field names for fields that are passed. The attributes of the corresponding parameter fields in the calling and called programs or procedures should be the same. If they are not, undesirable results may occur.

When a CALL or CALLB operation runs, the following occurs:

1. In the calling procedure, the contents of the factor 2 field of a PARM operation are copied into the result field (receiver field) of the same PARM operation.
2. In the case of a CALLB when the result field is *OMIT, a null address will be passed to the called procedure.
3. In the called procedure, after it receives control and after any normal program initialization, the contents of the result field of a PARM operation are copied into the factor 1 field (receiver field) of the same PARM operation.
4. In the called procedure, when control is returned to the calling procedure, the contents of the factor 2 field of a PARM operation are copied into the result field (receiver field) of the same PARM operation. This move does not occur if the called procedure ends abnormally.
5. Upon return to the calling procedure, the contents of the result field of a PARM operation in the calling procedure are copied into the factor 1 field (receiver field) of the same PARM operation. This move does not occur if the called procedure ends abnormally or if an error occurs on the call operation.

Note: The data is move in the same way as data is moved using the EVAL operation code. Strict type compatibility is enforced. For a discussion of how to call and pass parameters to a program through CL, see the *CL Programming*.

Figure 160 on page 426 illustrates the PARM operation.

PLIST (Identify a Parameter List)

Code	Factor 1	Factor 2	Result Field	Indicators		
PLIST	PLIST name					

The declarative PLIST operation defines a unique symbolic name for a parameter list to be specified in a CALL or CALLB operation.

You can specify a PLIST operation anywhere within calculations, including within total calculations and between subroutines. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement in the appropriate section of the program. The PLIST operation must be immediately followed by at least one PARM operation. Conditioning indicator entries (positions 9 through 11) are not allowed.

Factor 1 must contain the name of the parameter list. If the parameter list is the entry parameter list, factor 1 must contain *ENTRY. Only one *ENTRY parameter list can be specified in a program or procedure. A parameter list is ended when an operation other than PARM is encountered.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  In the calling program, the CALL operation calls PROG1 and
C*  allows PROG1 to access the data in the parameter list fields.
C                   CALL      'PROG1'       PLIST1
C*
C*  In the second PARM statement, when CALL is processed, the
C*  contents of factor 2, *IN27, are placed in the result field,
C*  BYTE.  When PROG1 returns control, the contents of the result
C*  field, BYTE, are placed in the factor 1 field, *IN30.  Note
C*  that factor 1 and factor 2 entries on a PARM are optional.
C*
C     PLIST1        PLIST
C                   PARM                    Amount            5 2
C     *IN30         PARM      *IN27         Byte              1
```

Figure 160 (Part 1 of 3). PLIST/PARM Operations

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   CALLB     'PROG2'
C*  In this example, the PARM operations immediately follow a
C*  CALLB operation instead of a PLIST operation.
C                   PARM                    Amount            5 2
C     *IN30         PARM      *IN27         Byte              1
```

Figure 160 (Part 2 of 3). PLIST/PARM Operations

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*  In the called procedure, PROG2,  *ENTRY in factor 1 of the
C*  PLIST statement identifies it as the entry parameter list.
C*  When control transfers to PROG2, the contents of the result
C*  fields (FieldC and FieldG) of the parameter list are placed in
C*  the factor 1 fields (FieldA and FieldD).  When the called procedure
C*  returns, the contents of the factor 2 fields of the parameter
C*  list (FieldB and FieldE) are placed in the result fields (FieldC
C*  and FieldG).  All of the fields are defined elsewhere in the
C*  procedure.
C     *ENTRY        PLIST
C     FieldA        PARM      FieldB        FieldC
C     FieldD        PARM      FieldE        FieldG
```

Figure 160 (Part 3 of 3). PLIST/PARM Operations

POST (Post)

Code	Factor 1	Factor 2	Result Field	Indicators		
POST	Program device	File name	INFDS name	_	ER	_

The POST operation puts information in an INFDS (file information data structure). This information contains the following:

- File Feedback Information specific to RPG I/O for the file
- Open Feedback Information for the file
- Input/Output Feedback Information and Device Dependent Feedback Information for the file OR Get Attribute Information

In factor 1, you can specify a program device name to get information about that specific program device. If you specify a program device in factor 1, the file must be defined as a WORKSTN file. If factor 1 does contain a program device, then the INFDS will contain Get Attribute Information following the Open Feedback Information. Use either a character field of length 10 or less, a character literal, or a character named constant. If you leave factor 1 blank, then the INFDS will contain Input/Output Feedback Information and Device Dependent Feedback Information following the Open Feedback Information.

In factor 2, specify the name of a file. Information for this file is posted in the INFDS associated with this file.

If you specify a file in factor 2, you can leave the result field blank. The INFDS associated with this file using the INFDS keyword in the file specification will be used. You can specify a file in factor 2 and its associated INFDS in the result field. If you leave factor 2 blank, you must specify the data structure name that has been used in the INFDS
keyword for the file specification in the result field; information from the associated file in the file specification will be posted.

In positions 73 and 74, you can specify an indicator that is set on if there is an error. If no indicator is specified, control passes to your INFSR subroutine (if you have specified one) or the default error/exception handler when an error/exception occurs.

Even when a POST operation code is not processed, its existence in your program can affect the way the RPG IV language operates. Usually, the INFDS is updated at each input and output operation or block of operations. However, if anywhere in your program, you have specified a POST operation with factor 1 blank, then RPG IV updates the I/O Feedback Information area and the Device Dependent Feedback Information area in the INFDS of any file only when you process a POST operation for that file. The File Dependent Information in the INFDS is updated on all Input/Output operations. If you have opened a file for multiple-member processing, the Open Feedback Information in the INFDS will be updated when an input operation (READ, READP, READE READPE) causes a new member to be opened.

Note that DUMP retrieves its information directly from the Open Data Path and not from the INFDS, so the file information sections of the DUMP do not depend on POST.

If a program has no POST operation code, or if it has only POST operation codes with factor 1 specified, the Input/Output Feedback and Device Dependent Feedback section is updated with each input/output operation or block of operations. If RPG is blocking records, most of the information in the INFDS will be valid only for the last complete block of records processed. When doing blocked input, from a data base file, RPG will update the relative record number and key information in the INFDS for each read, not just the last block of records processed. If you require more accurate information, do not use record blocking. See “File Information Data Structure” on page 61 for more information on record blocking. If you do not require feedback information after every input/output operation, you may be able to improve performance by using the POST operation only when you require the feedback information.

When a POST operation is processed, the associated file must be open. If you specify a program device on the POST operation, it does not have to be acquired by the file.

READ (Read a Record)

Code	Factor 1	Factor 2	Result Field	Indicators		
READ (N)		File name, Record name	Data structure	_	ER	EOF

The READ operation reads the record, currently pointed to, from a full procedural file (identified by an F in position 18 of the file description specifications).

Factor 2 must contain the name of a file. A record format name in factor 2 is allowed only with an externally described file (E in position 22 of the file description specifications). It may be the case that a READ-by-format-name operation will receive a different format than the one you specified in factor 2. If so, your READ operation ends in error.

The result field can contain the name of a data structure into which the record is read only if the file named in factor 2 is a program described file (identified by an F in position 22 of the file description specifications). See “File Operations” on page 294 for information on how data is transferred between the file and the data structure.

If a READ operation is successful, the file is positioned at the next record that satisfies the read. If either indicator is set on, you must reposition the file (using a CHAIN, SETLL or SETGT operation).

If the file from which you are reading is an update disk file, you can specify an N operation extender to indicate that no lock should be placed on the record when it is read. See the *ILE RPG/400 Programmer's Guide* for more information.

You can specify an indicator in positions 73 and 74 to be set on if the READ operation is not completed successfully. If an error occurs when no indicator is specified, control passes to your INFSR subroutine (if specified) or the default error/exception handler.

You must specify an indicator in positions 75 and 76 to signal whether end of file occurred on the READ operation. The file must be repositioned after the indicator is set on to process any further successful sequential operation (for example, READ or READP) to the file. This indicator is set on or off every time the READ operation is performed.

Figure 161 on page 431 illustrates the READ operation.

When you specify a multiple device file in factor 2, the READ operation does one of:

- Reads data from the device specified in the most recent NEXT operation (if such a NEXT operation has been processed).
- Accepts the first response from any device that has been acquired for the file, and that was specified for “invite status” with the DDS keyword INVITE. If there are no invited devices, the operation receives an end of file. The input is processed according to the corresponding format. If the device is a workstation, the last format written to it is used. If the device is a communications device, you can select the format.

Refer to the *ICF Programming* for more information on format selection processing for an ICF file. If you are using a BSC, CMN, or MXD file refer to the *System/38 CL Reference Manual,* for information on the FMTSLT parameter on the CRTBSCF, CRTCMNF, or ADDCMNDEVE command respectively.

The READ operation will stop waiting after a period of time in which no input is provided, or when one of the following CL commands has been entered with the controlled option specified:

- ENDJOB (End Job)
- ENDSBS (End Subsystem)
- PWRDWNSYS (Power Down System)
- ENDSYS (End System).

The error indicator specified in positions 73 and 74 is set on. See the *ICF Programming* for a discussion of the WAITRCD parameter on the commands to create or modify a file. This parameter controls the length of time the READ operation waits for input.

When you specify a format name in factor 2, and the format name is associated with a multiple device file, data is read from the device identified by the field specified in the DEVID keyword in file specifications. If there is no such entry, data is read from the device used in the last successful input operation.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  READ retrieves the next record from the file FILEA, which must
C*  be a full procedural file.  Indicator 61 is set on if end of
C*  file occurs on READ, or if end of file has occurred previously
C*  and the file has not been repositioned.  When indicator 61
C*  is set on, the program branches to the label (EOF) specified
C*  in the GOTO statement.
C                   READ      FILEA                                 61
C   61              GOTO      EOF
C*
C*  READ retrieves the next record of the type REC1 (factor 2)
C*  from an externally described file.  (REC1 is a record format
C*  name.)  Indicator 64 is set on if end of file occurs on READ, or
C*  or if it has occurred previously and the file has not been
C*  repositioned.  When indicator 64 is set on, the program
C*  branches to the label (EOF) specified in the GOTO statement.
C*  N operation code extender indicates that the record is not locked.
C*
C                   READ(N)   REC1                                  64
C   64              GOTO      EOF
C
C     EOF           TAG
```

Figure 161. READ Operation

READC (Read Next Changed Record)

Code	Factor 1	Factor 2	Result Field	Indicators		
READC		Record name		_	ER	EOF

The READC operation can be used only with an externally described WORKSTN file to obtain the next changed record in a subfile. Factor 2 is required and must be the name of a record format defined as a subfile by the SFILE keyword on the file description specifications. (See "SFILE(recformat:rrnfield)" on page 196 for information on the SFILE keyword.)

For a multiple device file, data is read from the subfile record associated with a program device; the program device is identified by the field specified in the DEVID keyword on the file specifications. If there is no such entry, data is read from the program device used for the last successful input operation.

You can specify a resulting indicator in positions 73 and 74 to be set on if an error occurs while the operation is running. A resulting indicator in positions 75 and 76 is required; it is set on when there are no more changed records in the subfile.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
F* CUSSCR is a WORKSTN file which displays a list of records from
F* the CUSINFO file. SFCUSR is the subfile name.
F*
FCUSINFO   UF   E             DISK
FCUSSCR    CF   E             WORKSTN SFILE(SFCUSR:RRN)
F
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C* After the subfile has been loaded with the records from the
C* CUSINFO file. It is written out to the screen using EXFMT with
C* the subfile control record, CTLCUS. If there are any changes in
C* any one of the records listed on the screen, the READC operation
C* will read the changed records one by one in the do while loop.
C* The corresponding record in the CUSINFO file will be located
C* with the CHAIN operation and will be updated with the changed
C* field.
C                   :
C                   EXFMT     CTLCUS
C                   :
C* SCUSNO, SCUSNAM, SCUSADR, and SCUSTEL are fields defined in the
C* subfile. CUSNAM, CUSADR, and CUSTEL are fields defined in a
C* record, CUSREC which is defined in the file CUSINFO.
C*
C                   READC     SFCUSR                                 27
C                   DOW       *IN27 = *OFF
C   SCUSNO          CHAIN     CUSINFO                             25
C* Update the record only if the record is found in the file.
C                   :
C                   IF        NOT *IN25
C                   EVAL      CUSNAM = SCUSNAM
C                   EVAL      CUSADR = SCUSADR
C                   EVAL      CUSTEL = SCUSTEL
C                   UPDATE    CUSREC
C                   ENDIF
C                   READC     SFCUSR                                 27
C                   ENDDO
```

Figure 162. READC example

READE (Read Equal Key)

Code	Factor 1	Factor 2	Result Field	Indicators		
READE (N)	Search argument	File name, Record name	Data structure	_	ER	EOF

The READE operation retrieves the next sequential record from a full procedural file (identified by an F in position 18 of the file description specifications) if the key of the record matches the search argument. If the key of the record does not match the search argument, the indicator that must be specified in positions 75 and 76 is set on, and the record is *not* returned to the program.

Factor 1, the search argument, is optional and identifies the record to be retrieved. It can be a field name, a literal, a named constant, or a figurative constant. You can also specify a KLIST name in factor 1 for an externally described file. If factor 1 is left blank and the full key of the next record is equal to that of the current record, the next record in the file is retrieved. The full key is defined by the record format or file used in factor 2.

Note: If the file being read is defined as update, a temporary lock on the next record is requested and the search argument is compared to the key of that record. If the record is already locked, the program must wait until the record is available before obtaining the temporary lock and making the comparison. If the comparison is unequal, the record-not-found indicator is turned on, and the temporary record lock is removed. If no lock (N operation extender) is specified, a temporary lock is not requested.

Factor 2 must contain the name of the file or record format to be retrieved. A record format name in factor 2 is allowed only with an externally described file (identified by an E in position 22 of the file description specifications).

The result field can contain the name of a data structure into which the record is read only if the file named in factor 2 is a program described file (identified by an F in position 22 of the file description specifications). See “File Operations” on page 294 for a description of the way data is transferred between the file and data structure.

If the file you are reading is an update disk file, you can specify an N operation extender to indicate that no lock should be placed on the record when it is read. See the *ILE RPG/400 Programmer’s Guide* for more information.

You can specify a resulting indicator in positions 73 and 74 to be set on if the operation does not complete successfully. You must specify a resulting indicator in positions 75 and 76. The indicator is set on if a record is not found with a key equal to the search argument or if end of file occurs. If a READE operation is not successful, you must reposition the file (for example, by a “CHAIN (Random Retrieval from a File),” “SETGT (Set Greater Than),” or “SETLL (Set Lower Limit)” operation).

If factor 1 is specified and you are processing a distributed data management (DDM) file, which was created before Version 3 Release 1 Modification 0, a key comparison cannot be done at the data management level. READE will do a key comparison using a hexadecimal collating sequence. This may give different results than expected when the content of the field on which the access path is built differs from the actual content of the fields that READE is using. The DDS features that cause the key comparison to differ are:

- ALTSEQ was specified for the file
- ABSVAL, ZONE, UNSIGNED or DIGIT keywords on key fields
- Variable length Date, Time or Timestamp key fields
- SRTSEQ for the file is not hexadecimal
- The sign is different from the system preferred sign

A READE (with factor 1 specified) that immediately follows an OPEN operation or an EOF condition retrieves the first record in the file if the key of the record matches the search argument. A READE (with *no* factor 1 specified) that immediately follows an OPEN operation or an EOF condition results in an error condition. The error indicator, if specified, in positions 73 and 74 is set on. No further I/O operations can be issued against the file until it is successfully closed and reopened.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  With Factor 1 Specified...
C*
C*  The READE operation retrieves the next record from the file
C*  FILEA and compares its key to the search argument, KEYFLD.
C*  Indicator 55 is set on if KEYFLD is not equal to the key of
C*  the record read or if end of file is encountered.
C*
C          KEYFLD        READE     FILEA                                55
C*
C*  The READE operation retrieves the next record of the type REC1
C*  from an externally described file and compares the key of the
C*  record read to the search argument, KEYFLD.  (REC1 is a record
C*  format name.)  Indicator 56 is set on if KEYFLD is not equal to
C*  the key of the record read or if end of file is encountered.
C          KEYFLD        READE     REC1                                 56
C*
C*  With No Factor 1 Specified...
C*
C*  The READE operation retrieves the next record in the access
C*  path from the file FILEA if the key value is equal to
C*  the key value of the record at the current cursor position.
C*  If the key values are not equal, indicator 55 is set on.
C                        READE     FILEA                                55
C*
C*  The READE operation retrieves the next record in the access
C*  path from the file FILEA if the key value equals the key value
C*  of the record at the current position.  REC1 is a record format
C*  name.  Indicator 56 is set on if the key values are unequal.
C*  N indicates that the record is not locked.
C                        READE(N)  REC1                                 56
```

Figure 163. READE Operation

READP (Read Prior Record)

Code	Factor 1	Factor 2	Result Field	Indicators		
READP (N)		File name, Record name	Data structure	_	ER	BOF

The READP operation reads the prior record from a full procedural file (identified by an F in position 18 of the file description specifications).

Factor 2 must contain the name of a file or record format to be read. A record format name in factor 2 is allowed only with an externally described file. If a record format name is specified in factor 2, the record retrieved is the first prior record of the specified type. Intervening records are bypassed.

The result field can contain the name of a data structure into which the record is read only if the file named in factor 2 is a program described file (identified by an F in position 22 of the file description specifications). See "File Operations" on page 294 for how data is transferred between the file and data structure.

If a READP operation is successful, the file is positioned at the next record that satisfies the read. If a READP operation is not successful, you must reposition the file (for example, by a CHAIN, SETLL or SETGT operation). You can specify an indicator in positions 73 and 74 to be set on if the READP operation is not completed successfully.

If the file from which you are reading is an update disk file, you can specify an N operation extender to indicate that no lock should be placed on the record when it is read. See the *ILE RPG/400 Programmer's Guide* for more information.

You must specify an indicator in positions 75 and 76 to be set on when no prior records exist in the file (beginning of file condition). If the file is not repositioned after this indicator is set on, the indicator is set for every subsequent READP operation to the file.

You must reposition the file, for example, by a CHAIN, SETLL or SETGT operation, after the beginning of file indicator in position 75-76 is set on to process any further successful sequential operations

Figure 164 on page 437 shows READP operations with a file name and record format name specified in factor 2.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The READP operation reads the prior record from FILEA.
C*  Indicator 71 is set on if beginning of file is encountered.
C*  When indicator 71 is set on, the program branches to the
C*  label BOF specified in the GOTO operation.
C                   READP     FILEA                                  71
C   71              GOTO      BOF
C*
C*  The READP operation reads the next prior record of the type
C*  REC1 from an externally described file.  (REC1 is a record
C*  format name.)  Indicator 72 is set on if beginning of file is
C*  encountered during processing of the READP operation.  When
C*  indicator 72 is set on, the program branches to the label BOF
C*  specified in the GOTO operation.
C                   READP     PREC1                                  72
C   72              GOTO      BOF
C*
C     BOF           TAG
```

Figure 164. READP Operation

READPE (Read Prior Equal)

Code	Factor 1	Factor 2	Result Field	Indicators		
READPE (N)	Search argument	File name, Record name	Data structure	_	ER	BOF

The READPE operation retrieves the next prior sequential record from a full procedural file if the key of the record matches the search argument. If the key of the record does not match the search argument, the indicator in positions 75-76 is set on and the record is not returned to the program.

Factor 1, the search argument, is optional and identifies the record to be retrieved. It can be a field name, a literal, a named constant, or a figurative constant. You can also specify a KLIST name in factor 1 for an externally defined file. If factor 1 is left blank and the full key of the next prior record is equal to that of the current record, the next prior record in the file is retrieved. The full key is defined by the record format or file used in factor 2.

Factor 2 must contain the name of the file or record format to be retrieved. A record format name in factor 2 is allowed only with an externally described file (identified by an E in position 22 of the file description specifications).

The result field can contain the name of a data structure into which the record is read only if the file named in factor 2 is a program described file (identified by an F in position 22 of the file description specifications). See “File Operations” on page 294 for a description of the way data is transferred between the file and data structure.

If the file from which you are reading is an update disk file, you can specify an N operation extender to indicate that no lock should be placed on the record when it is read. See the *ILE RPG/400 Programmer's Guide* for more information.

You can specify a resulting indicator in positions 73 and 74 to be set on if the operation is not completed successfully. You must specify a resulting indicator positions 75 and 76. The indicator is set on if a record is not found with a key equal to the search argument, or if beginning of file is encountered. If a READPE operation is not successful, you must reposition the file (for example, by a CHAIN, SETLL or SETGT operation).

Note: If the file being read is defined as update, a temporary lock on the prior record is requested and the search argument is compared to the key of that record. If the record is already locked, the program must wait until the record is available before obtaining the temporary lock and making the comparison. If the comparison is unequal, the record-not-found indicator is turned on, and the temporary record lock is removed. If no lock (N operation extender) is specified, a temporary lock is not requested.

If factor 1 is specified and you are processing a distributed data management (DDM) file, which was created before Version 3 Release 1 Modification 0, a key comparison cannot be done at the data management level. READPE will do a key comparison using a hexadecimal collating sequence. This may give different results than expected when the content of the field on which the access path is built differs from the actual content of the fields that READPE is using. The DDS features that cause the key comparison to differ are:

- ALTSEQ was specified for the file
- ABSVAL, ZONE, UNSIGNED or DIGIT keywords on key fields
- Variable length, Date, Time or Timestamp key fields
- SRTSEQ for the file is not hexadecimal
- The sign is different from the system preferred sign

A READPE (with factor 1 specified) that immediately follows an OPEN operation or a BOF condition returns BOF. A READPE (with *no* factor 1 specified) that immediately follows an OPEN operation or a BOF condition results in an error condition. The error indicator, if specified, in positions 73 and 74 is set on. The file *must* be repositioned using a CHAIN, SETLL, READ, READE or READP with factor 1 specified, prior to issuing a READPE operation with factor 1 blank. A SETGT operation code should not be used to position the file prior to issuing a READPE as this results in a record-not-found condition (because the record previous to the current record never has the same key as the current record after a SETGT is issued).

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* With Factor 1 Specified...
C*
C* The previous record is read and the key compared to FieldA.
C* Indicator 99 is set on if the record's key does not match
C* FieldA.
C     FieldA        READPE    FileA                                  99
C*
C* The previous record is read from FileB and the key compared
C* to FieldB.  The record is placed in data structure Ds1.  If
C* the record key does not match FieldB, indicator 99 is set on.
C     FieldB        READPE    FileB         Ds1                      99
C*
C* The previous record from record format RecA is read, and
C* the key compared to FieldC.  Indicator 88 is set on if the
C* operation is not completed successfully, and 99 is set on if
C* the record key does not match FieldC.
C     FieldC        READPE    RecA                                 8899
C*
C* With No Factor 1 Specified...
C*
C* The previous record in the access path is retrieved if its
C* key value equals the key value of the current record.
C* Indicator 99 is set on if the key values are not equal.
C                   READPE    FileA                                  99
C*
C* The previous record is retrieved from FileB if its key value
C* matches the key value of the record at the current position
C* in the file.  The record is placed in data structure Ds1.
C* Indicator 99 is set on if the key values are not equal.
C                   READPE    FileB         Ds1                      99
C*
C* The previous record from record format RecA is retrieved if
C* its key value matches the key value of the current record in
C* the access path.  Indicator 88 is set on if the operation is
C* not successful; 99 is set on if the key values are unequal.
C                   READPE    RecA                                 8899
```

Figure 165. READPE Operation

REL (Release)

Code	Factor 1	Factor 2	Result Field	Indicators		
REL	Program device	File name		_	ER	_

The REL operation releases the program device specified in factor 1 from the WORKSTN file specified in factor 2.

In factor 1, specify the program device name. Use either a character field of length 10 or less, a character literal, or a named constant. In factor 2, specify the file name.

You can specify an indicator in positions 73 and 74 that is set on when an error occurs. If you do not specify one and an error occurs, control passes to your INFSR subroutine (if specified) or the default error/exception handler.

When there are no program devices acquired to a WORKSTN file, the next READ-by-file-name or cycle-read gets an end-of-file condition. You must decide what the program does next. The REL operation may be used with a multiple device file or, for error recovery purpose, with a single device file.

Note: To release a record lock, use the UNLOCK operation. See the UNLOCK operation for more information about releasing record locks for update disk files.

RESET (Reset)

Code	Factor 1	Factor 2	Result Field	Indicators		
RESET	*NOKEY	*ALL	Structure or Variable	_	ER	_

The RESET operation sets elements in a structure (record format, data structure, array, or table), or a variable (field, subfield, or indicator) to its initial value. The initial value for a variable is the value the variable had at the end of the *INIT phase of the program. This value can be set using the “INZ{(constant)}” on page 210 page=no. keyword on the definition specification, or you can use the initialization subroutine to assign an initial value to the structure or variable. If RESET is specified for a structure or variable in the program, RPG will take a snapshot of the initial value of that variable or structure at the end of the *INIT phase after the *INZSR (initialization subroutine) is processed. See Figure 5 on page 16 for more information. This value is then used to reset the structure or variable.

Factor 1 must be blank unless the result field contains a record format name. In this case, factor 1 can contain *NOKEY, which indicates that key fields are not to be reset to their initial values.

Factor 2 may be blank or can contain *ALL. If *ALL is specified and the result field contains a multiple occurrence data structure or a table name, all occurrences or table elements will be reset and the occurrence level will be set to 1.

The result field contains the structure or variable to be reset to its initial value. It can contain one of: a record format, data structure name, array name, table name, field name, subfield, array element, or indicator name. If a record format name or a single occurrence data structure is specified, all fields are reset (in the order they are declared within the structure). In the case of a multiple-occurrence data structure, all fields in the current occurrence are reset. If a table name is specified, the current table element is reset; in the case of an array name, the entire array is reset. If an array element (including indicators) is specified in the result field using an array index, only the element specified is reset.

When the RESET operation is applied to a record format name and factor 2 contains *ALL and factor 1 is blank, all fields in the record format are reset. If factor 1 contains *NOKEY, all fields for the record format except the key fields are reset.

When the RESET operation is applied to a record format name and factor 2 contains blanks, only those fields that are output in that record format are affected. For WORKSTN (positions 36-42) file record formats, only those fields with a usage of output or both are affected. All field-conditioning indicators of the record format are affected by the operation. Fields in DISK, SEQ, or PRINTER file record formats are affected only if the record format is output in the program. Input-only fields are not affected by the RESET operation. By definition, they assume new values at the next input operation. You can use both data structure initialization and the *INZSR to set the initial value of a field or structure. The value is then used to reset the field or structure if it appears in the result field of a RESET operation. For example, you can use the *INZSR to set the values of several fields in a record format, and then later in the program use the RESET operation against the record format to reset the values of the fields. The snapshot of the field value for the RESET operation is taken at the end of the initialization step in the program, after the *INZSR is run. Any changes made to the values of variables in the *INZSR override any data structure initialization, and the value the variable has at the end of the initialization step is used to save the reset snapshot.

This operation results in an increase in the amount of storage required by the program. For any variable or structure that is reset, the storage requirement is doubled. Note that for multiple occurrence data structures, tables and arrays, the initial value of every occurrence or element is saved.

If a RESET occurs during the initialization routine of the program, an error message will be issued at run time. If a GOTO or CABxx is used to leave subroutine calculations during processing of the *INZSR, or if control passes to another part of the cycle as the result of error processing, the part of the initialization step which initializes the save areas will never be reached. In this case, an error message will be issued for all RESET operations in the program at run time.

For more information, see “Initialization” in Chapter 9 of the *ILE RPG/400 Programmer's Guide* and the “CLEAR (Clear)” operation code.

Note: RESET is not allowed for based variables and IMPORTed variables.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FEXTFILE   O    E             DISK
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D
D* The file EXTFILE contains one record format RECFMT containing
D* the character fields CHAR1 and CHAR2 and the numeric fields
D* NUM1 and NUM2.
D
D DS1             DS
D  DAY1                   1      8    INZ('MONDAY')
D  DAY2                   9     16    INZ('THURSDAY')
D  JDATE                 17     22
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The following operation sets DAY1, DAY2, and JDATE to blanks.
C
C                   CLEAR                   DS1
C
C* The following operation will set DAY1, DAY2, and JDATE to their
C* initial values of 'MONDAY', 'THURSDAY', and UDATE respectively.
C* The initial value of UDATE for JDATE is set in the *INZSR.
C
C                   RESET                   DS1
C
C* The following operation will set CHAR1 and CHAR2 to blanks and
C* NUM1 and NUM2 to zero.
C                   CLEAR                   RECFMT
C* The following operation will set CHAR1, CHAR2, NUM1, and
C* NUM2 to their initial values of 'NAME', 'ADDRESS', 1, and 2
C* respectively. These initial values are set in the *INZSR.
C*
C                   RESET                   RECFMT
C
C     *INZSR        BEGSR
C                   MOVEL     UDATE         JDATE
C                   MOVEL     'NAME    '    CHAR1
C                   MOVEL     'ADDRESS '    CHAR2
C                   Z-ADD     1             NUM1
C                   Z-ADD     2             NUM2
C                   ENDSR
C* The following operation sets all fields in the record format
C* to blanks, except the key fields.
C*
C     *NOKEY        RESET     *ALL          DBRECFMT
```

Figure 166. RESET Operation

RETURN (Return to Caller)

Code	Factor 1	Factor 2	Result Field	Indicators		
RETURN						

The RETURN operation causes a return to the caller as follows:

1. The halt indicators are checked. If a halt indicator is on, the procedure ends abnormally. (All open files are closed, an error return code is set to indicate to the calling routine that the procedure has ended abnormally, and control returns to the calling routine.)
2. If no halt indicators are on, the LR indicator is checked. If LR is on, the program ends normally. (Locked data area structures, arrays, and tables are written, and external indicators are reset.)
3. If no halt indicator is on and LR is not on, the procedure returns to the calling routine. Data is preserved for the next time the procedure is run. Files and data areas are not written out. See the Programmer's Guide for information on how running in a *NEW activation group affects the operation of RETURN.

ROLBK (Roll Back)

Code	Factor 1	Factor 2	Result Field	Indicators		
ROLBK				_	ER	_

The ROLBK operation:

- Eliminates all the changes to your files that have been specified in output operations since the previous COMMIT or ROLBK operation (or since the beginning of operations under commitment control if there has been no previous COMMIT or ROLBK operation).
- Releases all the record locks for the files you have under commitment control.
- Repositions the file to its position at the time of the previous COMMIT operation (or at the time of the file OPEN, if there has been no previous COMMIT operation.)

Commitment control starts when the CL command STRCMTCTL is executed. See the chapter on "Commitment Control" in the *ILE RPG/400 Programmer's Guide* for more information.

The file changes and the record-lock releases apply to all the files under commitment control in your activation group or job, whether the changes have been requested by the program issuing the ROLBK operation or by another program in the same activation group or job. The program issuing the ROLBK operation does not need to have any files under commitment control. For example, suppose program A calls program B and program C. Program B has files under commitment control, and program C does not. A ROLBK operation in program C still affects the files changed by program B.

The optional indicator in positions 73 and 74 is set on if the operation is not successfully completed.

SCAN (Scan String)

Code	Factor 1	Factor 2	Result Field	Indicators		
SCAN	Compare string:length	Base string:start	Left-most position	_	ER	FD

The SCAN operation scans a string (base string) contained in factor 2 for a substring (compare string) contained in factor 1. The scan begins at a specified location contained in factor 2 and continues for the length of the compare string which is specified in factor 1. The compare string and base string must both be of the same type, either character or graphic.

Factor 1 must contain either the compare string or the compare string, followed by a colon, followed by the length. The compare string portion of factor 1 can contain one of: a field name, array element, named constant, data structure name, literal, or table name. The length portion must be numeric with no decimal positions and can contain one of: a named constant, array element, field name, literal, or table name. If no length is specified, it is that of the compare string.

Factor 2 must contain either the base string or the base string, followed by a colon, followed by the start location of the SCAN. The base string portion of factor 2 can contain one of: a field name, array element, named constant, data structure name, literal, or table name. The start location portion of factor 2 must be numeric with no decimal positions and can be a named constant, array element, field name, literal, or table name.If graphic strings are used, the start position and length are measured in double bytes. If no start location is specified, a value of 1 is used.

The result field contains the numeric value of the leftmost position of the compare string in the base string, if found. It must be numeric with no decimal positions and can contain one of: a field name, array element, array name, or table name. If no result field is specified, a resulting indicator in positions 75 and 76 must be specified. The result field is set to 0 if the string is not found. If the result field contains an array, each occurrence of the compare string is placed in the array with the leftmost occurrence in element 1. The array elements following the element containing the rightmost occurrence are all zero. The result array should be as large as the field length of the base string specified in factor 2.

Notes:

1. The strings are indexed from position 1.
2. If the start position is greater than 1, the result field contains the position of the compare string relative to the beginning of the source string, not relative to the start position.
3. Figurative constants cannot be used in the factor 1, factor 2, or result fields.
4. No overlapping within data structures is allowed for factor 1 and the result field or factor 2 and the result field.

A resulting indicator in positions 75 and 76 can be specified to be set on if the string being scanned for is found. A resulting indicator in positions 73 and 74 can be specified to be set on if there is an error during the SCAN operation. An error occurs if the start position is greater than the length of factor 2 or if the value of factor 1 is too large. If no error indicator is specified and an error condition occurs, *PSSR, the error/exception handling subroutine runs (if it is specified in the program). If it is not specified, an error message is issued.

The SCAN begins at the leftmost character of factor 2 (as specified by the start location) and continues character by character, from left to right, comparing the characters in factor 2 to those in factor 1. If the result field is not an array, the SCAN operation will locate only the first occurrence of the compare string. To continue scanning beyond the first occurrence, use the result field from the previous SCAN operation to calculate the starting position of the next SCAN. If the result field is a numeric array, as many occurrences as there are elements in the array are noted. If no occurrences are found, the result field is set to zero; if the result field is an array, all its elements are set to zero.

Leading, trailing, or embedded blanks specified in the compare string are included in the SCAN operation.

The SCAN operation is case-sensitive. A compare string specified in lowercase will not be found in a base string specified in uppercase.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The SCAN operation finds the substring 'ABC' starting in
C* position 3 in factor 2; 3 is placed in the result field.
C* Indicator 90 is set on because the string is found. Because
C* no starting position is specified, the default of 1 is used.
C     'ABC'         SCAN      'XCABCD'      RESULT                   90
C*
C* This SCAN operation scans the string in factor 2 for an
C* occurrence of the string in factor 1 starting at position 3.
C* The 'Y' in position 1 of the base string is ignored because
C* the scan operation starts from position 3.
C* The operation places the values 5 and 6 in the first and
C* second elements of the array. Indicator 90 is set on.
C
C                   MOVE      'YARRYY'      FIELD1            6
C                   MOVE      'Y'           FIELD2            1
C     FIELD2        SCAN      FIELD1:3      ARRAY                    90
C
C* This SCAN operation scans the string in factor 2, starting
C* at position 2, for an occurrence of the string in factor 1
C* for a length of 4. Because 'TOOL' is not found in FIELD1,
C* INT is set to zero and indicator 90 is set off.
C
C                   MOVE      'TESTING'     FIELD1            7
C                   Z-ADD     2             X                 1 0
C                   MOVEL     'TOOL'        FIELD2            5
C     FIELD2:4      SCAN      FIELD1:X      INT90            20      90
C
```

Figure 167. SCAN Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
D*
D*       A Graphic SCAN example
D*
D*       Value of Graffld is graphic 'AACCBBGG'.
D*       Value of Number after the scan is 3 as the 3rd graphic
D*       character matches the value in factor 1
D
D Graffld                         4G   inz(G'oAACCBBGGi')

CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..

C* The SCAN operation scans the graphic string in factor 2 for
C* an occurrence of the graphic literal in factor 1. As this is a
C* graphic operation, the SCAN will operate on 2 bytes at a time

C
C     G'oBBi'       SCAN      Graffld:2     Number            5 0    90
C
```

Figure 168. SCAN Operation using graphic

SELECT (Begin a Select Group)

Code	Factor 1	Factor 2	Result Field	Indicators		
SELECT						

The select group conditionally processes one of several alternative sequences of operations. It consists of:

- A SELECT statement
- Zero or more WHENxx or WHEN groups
- An optional OTHER group
- ENDSL or END statement.

After the SELECT operation, control passes to the statement following the first WHENxx condition that is satisfied. All statements are then executed until the next WHENxx operation. Control passes to the ENDSL statement (only one WHENxx is executed). If no WHENxx condition is satisfied and an OTHER operation is specified, control passes to the statement following the OTHER operation. If no WHENxx condition is satisfied and no OTHER operation is specified, control transfers to the statement following the ENDSL operation of the select group.

Conditioning indicators can be used on the SELECT operation. If they are not satisfied, control passes immediately to the statement following the ENDSL operation of the select group. Conditioning indicators cannot be used on WHENxx, WHEN, OTHER and ENDSL operation individually.

The select group can be specified anywhere in calculations. It can be nested within IF, DO, or other select groups. The IF and DO groups can be nested within select groups.

If a SELECT operation is specified inside a select group, the WHENxx and OTHER operations apply to the new select group until an ENDSL is specified.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* In the following example, if X equals 1, do the operations in
C* sequence 1 (note that no END operation is needed before the
C* next WHENxx); if X does NOT equal 1, and if Y=2 and X<10, do the
C* operations in sequence 2.  If neither condition is true, do
C* the operations in sequence 3.
C*
C                   SELECT
C                   WHEN      X = 1
C                   Z-ADD     A             B
C                   MOVE      C             D
C*  Sequence 1
C                   :
C                   WHEN      ((Y = 2) AND (X < 10))
C*  Sequence 2
C                   :
C                   OTHER
C*  Sequence 3
C                   :
C                   ENDSL
C*
C* The following example shows a select group with conditioning
C* indicators.  After the CHAIN operation, if indicator 10 is on,
C* then control passes to the ADD operation.  If indicator 10 is
C* off, then the select group is processed.
C*
C     KEY           CHAIN     FILE                               10
C  N10              SELECT
C                   WHEN      X = 1
C*  Sequence 1
C                   :
C                   WHEN      Y = 2
C*  Sequence 2
C                   :
C                   ENDSL
C                   ADD       1             N
```

Figure 169. SELECT Operation

SETGT (Set Greater Than)

Code	Factor 1	Factor 2	Result Field	Indicators		
SETGT	Search argument	File name		NR	ER	_

The SETGT operation positions a file at the next record with a key or relative record number that is greater than the key or relative record number specified in factor 1. The file must be a full procedural file (identified by an F in position 18 of the file description specifications).

Factor 1 is required. If the file is accessed by key, factor 1 can be a field name, a named constant, a figurative constant, or a literal that is used as the search argument in positioning a file. You can also specify a KLIST name in factor 1 for an externally described file that is positioned by key. If the file is accessed by relative record number, factor 1 must be an integer literal, named constant, or field.

Factor 2 is required and must be either a file name or a record format name. A record format name in factor 2 is allowed only with an externally described file.

You can specify a resulting indicator in positions 71 and 72 to be set on if no record is found with a key or relative record number that is greater than the search argument specified in factor 1. You can specify any valid resulting indicator in positions 73 and 74 to be set on if an error occurs during processing of the operation.

If the SETGT operation is not successful (no-record-found condition), the file is positioned to the end of the file.

Figurative constants can also be used to position the file.

Note: The discussion and examples of using figurative constants which follow, assume that *LOVAL and *HIVAL are not used as actual keys in the file.

When used with a file with a composite key, figurative constants are treated as though each field of the key contained the figurative constant value. In most cases, *LOVAL positions the file so that the first read retrieves the record with the lowest key. In most cases, *HIVAL positions the file so that a READ receives an end-of-file indication; a subsequent READP retrieves the last record in the file. However, note the following cases for using *LOVAL and *HIVAL:

- With an externally described file that has a key in descending order, *HIVAL positions the file so that the first read operation retrieves the first record in the file (the record with the highest key), and *LOVAL positions the file so that a READP operation retrieves the last record in the file (the record with the lowest key).

- If a record is added or a key field is altered after a SETGT operation with either *LOVAL or *HIVAL, the file may no longer be positioned to the record with the lowest or highest key.
- *LOVAL for numeric represents a key value X‘99...9D’ and *HIVAL for numeric keys represents a key value X‘99...9F’. When a program described file has a packed decimal key specified in the file specifications but the actual file key field contains character data, records may have keys that are less than *LOVAL or greater than *HIVAL. When a key field contains unsigned binary data, *LOVAL may not be the lowest key.

When *LOVAL or *HIVAL are used with key fields with a Date or Time data type, the values are dependent of the Date-Time format used. For details on these values please see Chapter 7, “Data Types and Data Formats” on page 101.

Following the SETGT operation, a file is positioned so that it is immediately before the first record whose key or relative record number is greater than the search argument specified in factor 1. You retrieve this record by reading the file. Before you read the file, however, records may be deleted from the file by another job or through another file in your job. Thus, you may not get the record you expected. For information on preventing unexpected modification of your files, see the discussion of allocating objects in the *Programming: Control Language Reference*.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*  This example shows how to position the file so READ will read
C*  the next record. The search argument, KEY, specified for the
C*  SETGT operation has a value of 98; therefore, SETGT positions
C*  the file before the first record of file format FILEA that
C*  has a key field value greater than 98.  The file is positioned
C*  before the first record with a key value of 100.  The READ
C*  operation reads the record that has a value of 100 in its key
C*  field.
C
C     KEY           SETGT     FILEA
C                   READ      FILEA                                  64
C*
C*  This example shows how to read the last record of a group of
C*  records with the same key value and format from a program
C*  described file.  The search argument, KEY, specified for the
C*  SETGT operation positions the file before the first record of
C*  file FILEB that has a key field value greater than 70.
C*  The file is positioned before the first record with a key
C*  value  of 80.  The READP operation reads the last record that
C*  has a value of 70 in its key field.
C
C     KEY           SETGT     FILEB
C                   READP     FILEB                                  64
```

Figure 170 (Part 1 of 4). SETGT Operation

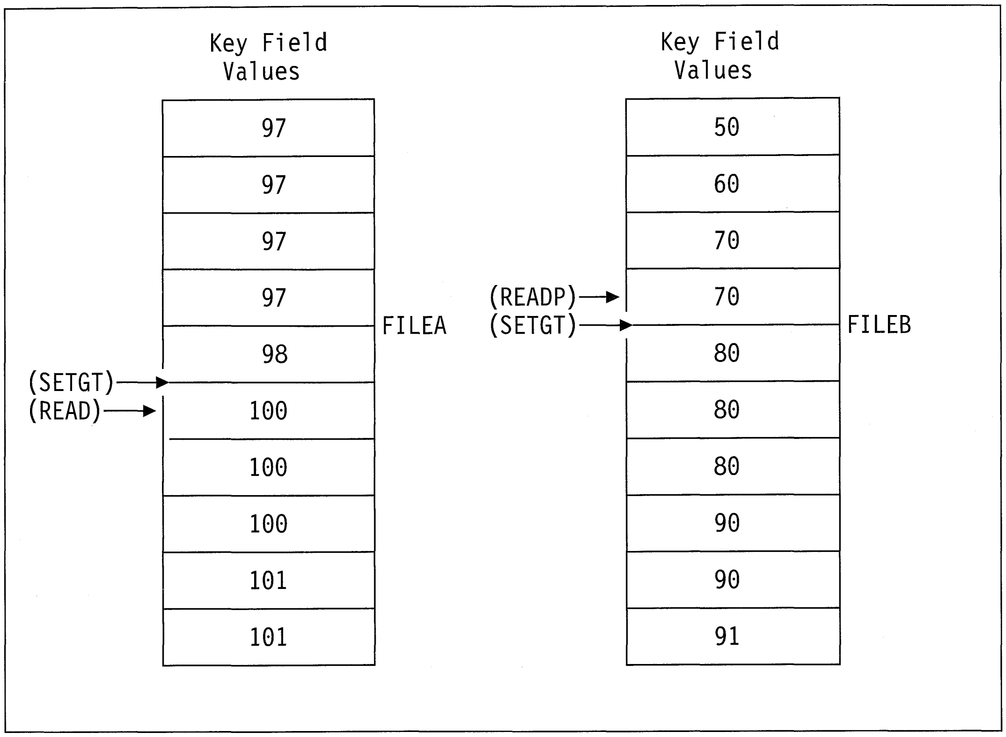


Figure 170 (Part 2 of 4). SETGT Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  This example shows the use of *LOVAL.  The SETLL operation
C*  positions the file before the first record of a file in
C*  ascending order.  The READ operation reads the first record
C*  (key value 97).
C
C     *LOVAL        SETLL     RECDA
C                   READ      RECDA                                  64
C
C*  This example shows the use of *HIVAL.  The SETGT operation
C*  positions the file after the last record of a file in ascending
C*  order. The READP operation reads the last record (key value 91).
C
C     *HIVAL        SETGT     RECDB
C                   READP     RECDB                                  64
```

Figure 170 (Part 3 of 4). SETGT Operation

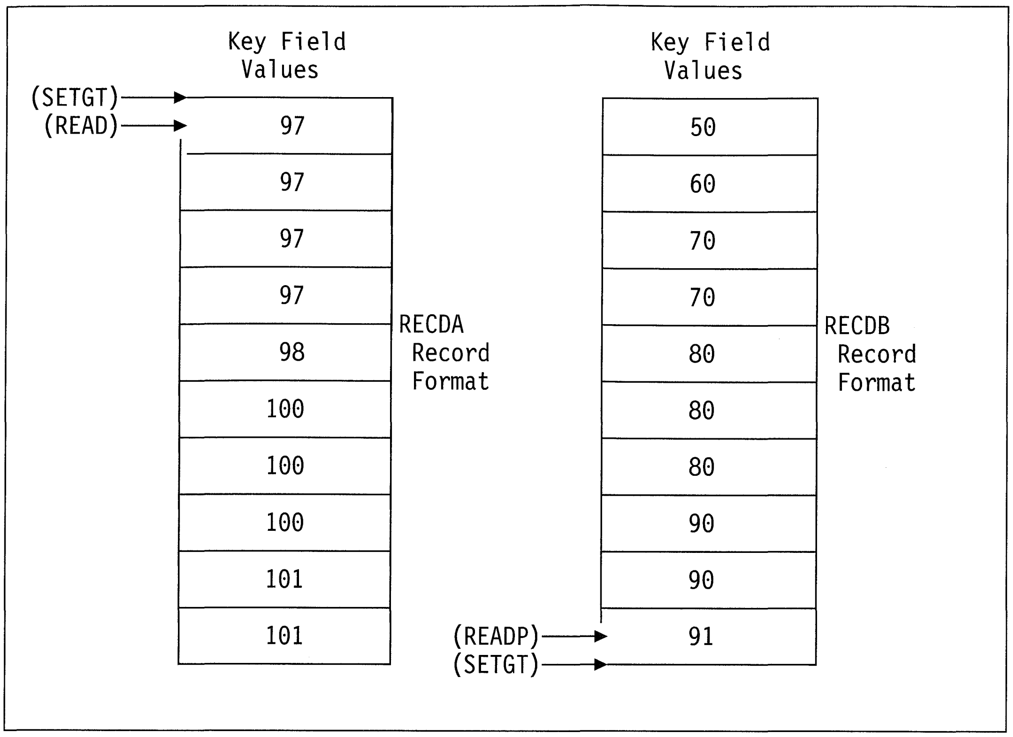


Figure 170 (Part 4 of 4). SETGT Operation

SETLL (Set Lower Limit)

Code	Factor 1	Factor 2	Result Field	Indicators		
SETLL	Search argument	File name		NR	ER	EQ

The SETLL operation positions a file at the next record that has a key or relative record number that is greater than or equal to the search argument (key or relative record number) specified in factor 1. The file must be a full procedural file (identified by an F in position 18 of the file description specifications).

Factor 1 is required. If the file is accessed by key, factor 1 can be a field name, a named constant, a figurative constant, or a literal that is used as the search argument in positioning the file. You can also specify a KLIST name in factor 1 for an externally described file that is positioned by key. If the file is accessed by relative record number, factor 1 must contain an integer literal, named constant, or numeric field with no decimal positions.

Factor 2 is required and can contain either a file name or a record format name. A record format name in factor 2 is allowed only with an externally described file.

The resulting indicators reflect the status of the operation. If an indicator is specified in positions 71 and 72, it is set on when the search argument is greater than the highest key or relative record number in the file. If an indicator is specified in positions 73 and 74, it is set on when an error occurs during running of the operation. If an indicator is specified in positions 75 and 76, it is set on when a record is present whose key or relative record number is equal to the search argument.

If factor 2 contains a file name for which the lower limit is to be set, the file is positioned at the first record with a key or relative record number equal to or greater than the search argument specified in factor 1.

If factor 2 contains a record format name for which the lower limit is to be set, the file is positioned at the first record of the specified type that has a key equal to or greater than the search argument specified in factor 1.

Figurative constants can be used to position the file.

Note: The discussion and examples of using figurative constants which follow, assume that *LOVAL and *HIVAL are not used as actual keys in the file.

When used with a file with a composite key, figurative constants are treated as though each field of the key contained the figurative constant value. Using SETLL with *LOVAL positions the file so that the first read retrieves the record with the lowest key. In most cases (when duplicate keys are not allowed), *HIVAL positions the file so that a READP retrieves the last record in the file, or a READ receives an end-of-file indication. However, note the following cases for using *LOVAL and *HIVAL:

- With an externally described file that has a key in descending order, *HIVAL positions the file so that the first read operation retrieves the first record in the file (the record with the highest key), and *LOVAL positions the file so that a READP operation retrieves the last record in the file (the record with the lowest key).

- If a record is added or a key field altered after a SETLL operation with either *LOVAL or *HIVAL, the file may no longer be positioned to the record with the lowest or highest key.
- *LOVAL for numeric keys represents a key value X'99...9D' and *HIVAL represents a key value X'99...9F'. When a program described file has a packed decimal key specified in the file specifications but the actual file key field contains character data, records may have keys that are less than *LOVAL or greater than *HIVAL. When a key field contains unsigned binary data, *LOVAL may not be the lowest key.

When *LOVAL or *HIVAL are used with key fields with a Date or Time data type, the values are dependent of the Date-Time format used. For details on these values please see Chapter 7, "Data Types and Data Formats" on page 101.

Figure 170 on page 452 (part 2 of 2) shows the use of figurative constants with the SETGT operation. Figurative constants are used the same way with the SETLL operation.

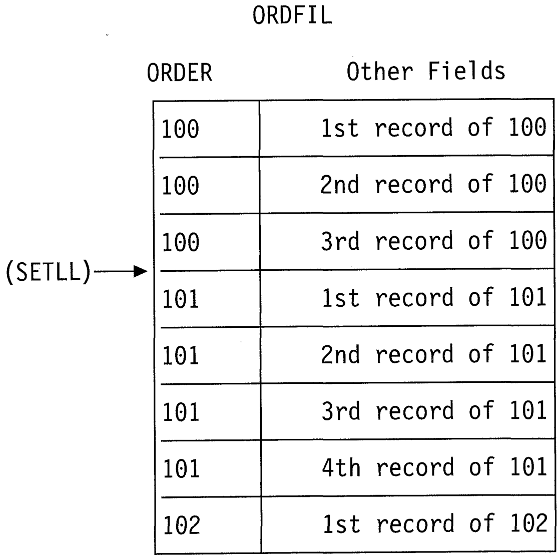
Remember the following when using the SETLL operation:

- If the SETLL operation is not successful (no records found condition), the file is positioned to the end of the file.
- When end of file is reached on a file being processed by SETLL, another SETLL can be issued to reposition the file.
- After a SETLL operation successfully positions the file at a record, you retrieve this record by reading the file. Before you read the file, however, records may be deleted from the file by another job or through another file in your job. Thus, you may not get the record you expected. Even if the resulting indicator in positions 75 and 76 is set on to indicate you found a matching record, you may not get that record. For information on preventing unexpected modification of your files, see the discussion of allocating objects in the *Programming: Control Language Reference*.
- SETLL does not cause the system to access a data record. If you are only interested in verifying that a key actually exists, SETLL with an equal indicator (positions 75-76) is a better performing solution than the CHAIN operation in most cases. Under special cases of a multiple format logical file with sparse keys, CHAIN can be a faster solution than SETLL.

In this example, the file ORDFIL contains order records. The key field is the order number (ORDER) field. There are multiple records for each order. ORDFIL looks like this in the calculation specifications:

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  All the 101 records in ORDFIL are to be printed.  The value 101
C*  has previously been placed in ORDER.  The SETLL operation
C*  positions the file at the first record with the key value 101
C*  and indicator 55 is set on.
C
C     ORDER         SETLL     ORDFIL                                 55
C
C*  The following DO loop processes all the records that have the
C*  same key value. The indicator 55 is used for conditioning
C*  the DO loop.
C
C   55              DOU       *IN56
C     ORDER         READE     ORDFIL                                 56
C  N56              EXCEPT    DETAIL
C                   ENDDO
C*  The READE operation reads the second, third, and fourth 101
C*  records in the same manner as the first 101 record was read.
C*  After the fourth 101 record is read, the READE operation is
C*  attempted.  Because the 102 record is not of the same group,
C*  indicator 56 is set on the EXCEPT operation is bypassed and
C*  the DOU loop ends.
```

Figure 171 (Part 1 of 2). SETLL Operation



ORDFIL

ORDER	Other Fields
100	1st record of 100
100	2nd record of 100
100	3rd record of 100
101	1st record of 101
101	2nd record of 101
101	3rd record of 101
101	4th record of 101
102	1st record of 102

Figure 171 (Part 2 of 2). SETLL Operation

SETOFF (Set Indicator Off)

Code	Factor 1	Factor 2	Result Field	Indicators		
SETOFF				OF	OF	OF

The SETOFF operation sets off any indicators specified in positions 71 through 76. You must specify at least one resulting indicator in positions 71 through 76. Entries of 1P and MR are not valid. Setting off L1 through L9 indicators does not automatically set off any lower control-level indicators.

Figure 172 on page 459 illustrates the SETOFF operation.

SETON (Set Indicator On)

Code	Factor 1	Factor 2	Result Field	Indicators		
SETON				ON	ON	ON

The SETON operation sets on any indicators specified in positions 71 through 76. You must specify at least one resulting indicator in positions 71 through 76. Entries of 1P, MR, KA through KN, and KP through KY are not valid. Setting on L1 through L9 indicators does not automatically set on any lower control-level indicators.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The SETON and SETOFF operations set from one to three indicators
C*  specified in positions 71 through 76 on and off.
C*  The SETON operation sets indicator 17 on.
C
C                   SETON                                        17
C
C*  The SETON operation sets indicators 17 and 18 on.
C
C                   SETON                                        1718
C
C*  The SETOFF operation sets indicator 21 off.
C
C                   SETOFF                                       21
```

Figure 172. SETON and SETOFF Operations

SHTDN (Shut Down)

Code	Factor 1	Factor 2	Result Field	Indicators		
SHTDN				ON	_	_

The SHTDN operation allows the programmer to determine whether the system operator has requested shutdown. If the system operator has requested shutdown, the resulting indicator specified in positions 71 and 72 is set on. Positions 71 and 72 must contain one of the following indicators: 01 through 99, L1 through L9, U1 through U8, H1 through H9, LR, or RT.

The system operator can request shutdown by specifying the *CNTRLD option on the following CL commands: ENDJOB (End Job), PWRDWNSYS (Power Down System), ENDSYS (End System), and ENDSBS (End Subsystem). For information on these commands, see the *Programming: Control Language Reference*.

Positions 73 through 76 must be blank.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  When the SHTDN operation is run, a test is made to determine
C*  whether the system operator has requested shutdown.  If so,
C*  indicator 27 is set on.
C
C                   SHTDN                                        27
C   27              EXSR      Term_appl
C                   :
C                   :
C     Term_appl     BEGSR
C                   CLOSE     *ALL
C                   :
C                   ENDSR
```

Figure 173. SHTDN Operation

SORTA (Sort an Array)

Code	Factor 1	Factor 2	Result Field	Indicators		
SORTA		Array name				

Factor 2 contains the name of an array to be sorted. The array is sorted into sequence (ascending or descending), depending on the sequence specified for the array on the definition specification. If no sequence is specified, the array is sorted into ascending sequence. The array *IN cannot be specified in factor 2 of a SORTA operation. If the array is defined as a compile-time or prerun-time array with data in alternating form, the alternate array is not sorted. Only the array specified in factor 2 is sorted.

If the array is defined with the “OVERLAY(name{:pos})” keyword, the base array will be sorted in the sequence defined by the OVERLAY array.

Graphic arrays will be sorted by the hexadecimal values of the array elements, disregarding the alternate collating sequence, in the order specified on the definition specification.

Note: Sorting an array does not preserve any previous order. For example, if you sort an array twice, using different overlay arrays, the final sequence will be that of the last sort. Elements that are equal in the sort sequence but have different hexadecimal values (for example, due to alternate collating sequence or the use of an overlay array to determine sequence), may not be in the same order after sorting as they were before.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
DARRY             S             1A   DIM(8) ASCEND
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The SORTA operation sorts ARRY into ascending sequence because
C*  the ASCEND keyword is specified.
C*  If the unsorted ARRY contents were GT1BA2L0, the sorted ARRY
C*  contents would be ABGLT012.
C
C                   SORTA     ARRY
```

Figure 174. SORTA Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D* In this example, the base array has the values aa44 bb33 cc22 dd11
D* so the overlaid array ARRO has the values 44 33 22 11.
D                 DS
D ARR                            4    DIM(4) ASCEND
D ARRO                           2    OVERLAY(ARR:3)
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C* After the SORTA operation, the base array has the values
C* dd11 cc22 bb33 aa44
C
C                   SORTA     ARRO
```

Figure 175. SORTA Operation with OVERLAY

SQRT (Square Root)

Code	Factor 1	Factor 2	Result Field	Indicators		
SQRT (H)		Value	Root			

The SQRT operation derives the square root of the field named in factor 2. The square root of factor 2 is placed in the result field.

Factor 2 must be numeric, and can contain one of: an array, array element, field, figurative constant, literal, named constant, subfield, or table name.

The result field must be numeric, and can contain one of: an array, array element, subfield, or table element.

An entire array can be used in a SQRT operation if factor 2 and the result field contain array names.

The number of decimal positions in the result field can be either less than or greater than the number of decimal positions in factor 2. However, the result field should not have fewer than half the number of decimal positions in factor 2.

If the value of the factor 2 field is zero, the result field value is also zero. If the value of the factor 2 field is negative, the RPG IV exception/error handling routine receives control.

For further rules on the SQRT operation, see “Arithmetic Operations” on page 287.

See Figure 100 on page 289 for an example of the SQRT operation.

SUB (Subtract)

Code	Factor 1	Factor 2	Result Field	Indicators		
SUB (H)	Minuend	Subtrahend	Difference	+	–	Z

If factor 1 is specified, factor 2 is subtracted from factor 1 and the difference is placed in the result field. If factor 1 is not specified, the contents of factor 2 are subtracted from the contents of the result field.

Factor 1 and factor 2 must be numeric, and each can contain one of: an array, array element, field, figurative constant, literal, named constant, subfield, or table name.

The result field must be numeric, and can contain one of: an array, array element, subfield, or table name.

For rules for the SUB operation, see “Arithmetic Operations” on page 287.

See Figure 100 on page 289 for examples of the SUB operation.

SUBDUR (Subtract Duration)

Code	Factor 1	Factor 2	Result Field	Indicators		
SUBDUR	Date/Time/Timestamp	Duration:Duration Code	Date/Time/ Timestamp	_	ER	_
SUBDUR	Date/Time/Timestamp	Date/Time/Timestamp	Duration: Duration code	_	ER	_

The SUBDUR operation has been provided to:

- Subtract a duration to establish a new Date, Time or Timestamp
- Calculate a duration

Subtract a duration: The SUBDUR operation can be used to subtract a duration specified in factor 2 from a field or constant specified in factor 1 and place the resulting Date, Time or Timestamp in the field specified in the Result field.

Factor 1 is optional and may contain a Date, Time or Timestamp field, array, array element, literal or constant. If factor 1 contains a field name, array or array element then its data type must be the same type as the field specified in the result field. If factor 1 is not specified then the duration is subtracted from the field specified in the result field.

Factor 2 is required and contains two subfactors. The first is a numeric field, array or constant with zero decimal positions. If the field is negative then the duration is added to the field. The second subfactor must be a valid duration code indicating the type of duration.

The duration code must be consistent with the result field data type. e.g. you can subtract a Year, Month or Day duration but not a Minute duration from a Date field.

The result field must be a Date, Time or Timestamp data type field, array or array element. If factor 1 is blank, the duration is subtracted from the value in the result field. If the result field is an array, the value in factor-2 is subtracted from each element in the array. If the result field is a time field, the result must always be a valid Time, (i.e. if the calculated result is < 24:00:00, add 24 hours or a multiple of 24 hours until the time is valid). If the result is a timestamp field, and the *MS or time portion is operated on, the date must be adjusted, each time the 24:00:00 boundary is crossed.

Note:

The system places a 15 digit limit on durations. Subtracting a Duration with more than 15 significant digits will cause errors or truncation. These problems can be avoided by limiting the first subfactor in Factor 2 to 15 digits.

Calculate a duration: The SUBDUR operation can also be used to calculate a duration between:

1. Two dates
2. A date and a timestamp
3. Two times

4. A time and a timestamp
5. Two timestamps

The data types in Factor 1 and Factor 2 must be compatible types as specified above.

Factor 1 is required and must contain a Date, Time or Timestamp field, subfield, array, array element, constant or literal.

Factor 2 is required and must also contain a Date, Time or Timestamp field, array, array element, literal or constant.

The result field consists of two subfactors. The first is the name of a zero decimal numeric field, array or array element in which the result of the operation will be placed. The second subfactor contains a duration code denoting the type of duration. The result field will be negative if the date in factor 1 is earlier than the date in factor 2.

The duration code must be consistent with factor 1 and factor 2, *Years(*Y), *Months(*M) and *Days(*D) if factor 1 and/or factor 2 is a Date or Timestamp, *Hours(*H), *Minutes(*MN) and *Seconds(*S) when factor 1 and/ or factor 2 is a Time or Timestamp.

Note: Calculating a micro-second Duration (*mseconds) can exceed the 15 digit system limit for Durations and cause errors or truncation. This situation will occur when there is more than a 32 year and 9 month difference between the Factor 1 and Factor 2 entries.

Possible error situations

1. For subtracting durations:
   - If the value of the Date, Time or Timestamp field in factor 1 is invalid
   - if factor 1 is blank and the value of the result field before the operation is invalid
   - or if the result of the operation is invalid
2. For calculating durations:
   - If the value of the Date, Time or Timestamp field in factor 1 or factor 2 is invalid
   - or if the result field is not large enough to hold the resulting duration.

In each of these cases an error will be signalled, the error indicator(columns 73-74), if specified, will be set on, and the value of the result field will remain unchanged.

If an error is detected an error will be generated with one of the following program status codes:

- 103: Result field not large enough to hold result
- 112: Date, Time or Timestamp value not valid
- 113: A Date overflow or underflow occurred (that is, the resulting Date is greater than *HIVAL or less than *LOVAL).

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....

C* Determine a LOANDATE which is xx years, yy months, zz days prior to
C* the DUEDATE.

C     DUEDATE       SUBDUR    XX:*YEARS     LOANDATE
C                   SUBDUR    YY:*MONTHS    LOANDATE
C                   SUBDUR    ZZ:*DAYS      LOANDATE

C* Add 30 days to a loan due date
C*
C                   SUBDUR    -30:*D        LOANDUE

C* Calculate the number or days between a LOANDATE and a DUEDATE.

C     LOANDATE      SUBDUR    DUEDATE       NUM_DAYS:*D        5 0

C* Determine the number of seconds between LOANDATE and DUEDATE.

C     LOANDATE      SUBDUR    DUEDATE       NUM_SECS:*S        5 0
```

Figure 176. SUBDUR Operations

SUBST (Substring)

Code	Factor 1	Factor 2	Result Field	Indicators		
SUBST (P)	Length to extract	Base string:start	Target string	_	ER	_

The SUBST operation returns a substring from factor 2, starting at the location specified in factor 2 for the length specified in factor 1, and places this substring in the result field. If factor 1 is not specified, the length of the string from the start position is used. For graphic strings, the start position is measured in double bytes. The base string and the target string must both be of the same type, either both character or both graphic.

Factor 1 can contain the length value of the string to be extracted from the string specified in factor 2. It must be numeric with no decimal positions and can contain one of: a field name, array element, table name, literal, or named constant.

Factor 2 must contain either the base string, or the base string followed by ':', followed by the start location. The base string portion can contain one of: a field name, array element, named constant, data structure name, table name, or literal. The start position must be numeric with zero decimal positions, and can contain one of the following: a field name, array element, table name, literal or named constant. If it is not specified, SUBST starts in position 1 of the base string. For graphic strings, the start position is measured in double bytes.

The start location and the length of the substring to be extracted must be positive integers. The start location must not be greater than the length of the base string, and the length must not be greater than the length of the base string from the start location. If either or both of these conditions is not satisfied, the operation will not be performed, and if you specified an error indicator in positions 73 and 74, it will be set on. If you did not specify an error indicator, the exception/error handling routine receives control.

The result field must be character or graphic and can contain one of the following: a field name, array element, data structure, or table name. The result is left-justified. The result field's length should be at least as large as the length specified in factor 1. If the substring is longer than the field specified in the result field, the substring will be truncated from the right.

Note: You cannot use figurative constants in the factor 1, factor 2, or result fields. Overlapping is allowed for factor 1 and the result field or factor 2 and the result field. If factor 1 is shorter than the length of the result field, a P specified in the operation extender position indicates that the result field should be padded on the right with blanks after the substring occurs.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The SUBST operation extracts the substring from factor 2 starting
C* at position 3 for a length of 2.  The value 'CD' is placed in the
C* result field TARGET.  Indicator 90 is not set on because no error
C* occurred.
C
C                   Z-ADD     3             T                 2 0
C                   MOVEL     'ABCDEF'      String           10
C     2             SUBST     String:T      Target                 90
C*
C* In this SUBST operation, the length is greater than the length
C* of the string minus the start position plus 1.  As a result,
C* indicator 90 is set on and the result field is not changed.
C
C                   MOVE      'ABCDEF'      String            6
C                   Z-ADD     4             T                 1 0
C     5             SUBST     String:T      Result                 90
C
C* In this SUBST operation, 3 characters are substringed starting
C* at the fifth position of the base string.  Because P is not
C* specified, only the first 3 characters of TARGET are
C* changed.  TARGET contains '123XXXXX'.
C
C                   Z-ADD     3             Length            2 0
C                   Z-ADD     5             T                 2 0
C                   MOVE      'TEST123'     String            8
C                   MOVE      *ALL'X'       Target
C     Length        SUBST     String:T      Target            8
```

Figure 177 (Part 1 of 2). SUBST Operation

```
C*
C* This example is the same as the previous one except P
C* specified, and the result is padded with blanks.
C* TARGET equals '123ƀƀƀƀƀ'.
C
C                   Z-ADD     3             Length            2 0
C                   Z-ADD5                  T                 2 0
C                   MOVE      'TEST123'     String            8
C                   MOVE      *ALL'X'       Target
C     Length        SUBST(P)  String:T      Target            8
C
C
C*
C* In the following example, CITY contains the string
C* 'Toronto, Ontario'.  The SCAN operation is used to locate the
C* separating blank, position 9 in this illustration.  SUBST
C* without factor 1 places the string starting at position 10 and
C* continuing for the length of the string in field TCNTRE.
C* TCNTRE contains 'Ontario'.
C     ' '           SCAN      City          C
C                   ADD       1             C
C                   SUBST     City:C        TCntre
C*
C* Before the operations STRING='ƀƀƀJohnƀƀƀƀƀƀ'.
C* RESULT is a 10 character field which contains 'ABCDEFGHIJ'.
C* The CHECK operation locates the first nonblank character
C* and sets on indicator 10 if such a character exists.  If *IN10
C* is on, the SUBST operation substrings STRING starting from the
C* first non-blank to the end of STRING.  Padding is used to ensure
C* that nothing is left from the previous contents of the result
C* field.  If STRING contains the value '   HELLO ' then RESULT
C* will contain the value 'HELLO     ' after the SUBST(P) operation.
C* After the operations RESULT='Johnƀƀƀƀƀƀ'.
C
C     ' '           CHECK     STRING        ST                       10
C   10              SUBST(P)  STRING:ST     RESULT
```

Figure 177 (Part 2 of 2). SUBST Operation

TAG (Tag)

Code	Factor 1	Factor 2	Result Field	Indicators		
TAG	Label					

The declarative TAG operation names the label that identifies the destination of a "GOTO (Go To)" or "CABxx (Compare and Branch)" operation.

It can be specified anywhere within calculations, including within total calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, the LR indicator, or the L0 entry to group the statement within the appropriate section of the program. Conditioning indicator entries (positions 9 through 11) are not allowed.

Factor 1 must contain the name of the destination of a GOTO or CABxx operation. This name must be a unique symbolic name, which is specified in factor 2 of a GOTO operation or in the result field of a CABxx operation. The name can be used as a common point for multiple GOTO or CABxx operations.

Branching to the TAG from a different part of the RPG IV logic cycle may result in an endless loop. For example, if a detail calculation line specifies a GOTO operation to a total calculation TAG operation, an endless loop may occur.

See Figure 140 on page 373 for examples of the TAG operation.

TEST (Test Date/Time/Timestamp)

Code	Factor 1	Factor 2	Result Field	Indicators		
TEST (D/T/Z)	Date/Time format		Tested field	_	ER	_

The TEST operation code allows users to test the validity of date, time, or timestamp fields prior to using them.

Factor 1 must be blank if the result field contains a date, time or timestamp field or if a character or numeric field is being tested for a valid timestamp.

- If the result field contains fields declared as Date, Time or Timestamp:
  - Factor 1 must be blank
  - Operation code extender is not allowed
- If the result field contains fields declared as character or numeric then an operation code extender must be specified.
  - If the operation code extender is "D" (test Date),
    - Factor 1 is optional and may contain any of the valid Date formats (See “Date Data” on page 105).
    - If factor 1 is blank, the format specified on the control specification with the DATFMT keyword is assumed. If this keyword is not specified, *ISO is assumed.
  - If the operation code extender is "T" (test Time),
    - Factor 1 is optional and may contain any of the valid Time formats (See “Time Data” on page 107).
    - If factor 1 is blank, the format specified on the control specification with the TIMFMT keyword is assumed. If this keyword is not specified, *ISO is assumed.
  - If the operation code extender is "Z" (test Timestamp),
    - Factor 1 must be blank

Numeric fields are tested for valid digit portion of a Date, Time or Timestamp value. Character fields are tested for both valid digits and separators.

An indicator in positions 73-74 is required and will be set on if the content of the result field is not valid.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D
D Datefield       S               D   DATFMT(*JIS)
D Date1           S              6P 0 INZ(910921)
D Timefield       S              8    INZ('13:05 PM')
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*
C*  Indicator 20 is set since Date1 is not *DMY
C*
C     *DMY          TEST (D)                Date1                  20
C*
C*  No Factor 1 since result is a D data type field
C*  Indicator 21 will not be set on since the field
C*  contains a valid date
C
C                   TEST                    Datefield              21
C
C*  In the following test indicator 22 will be set on since the
C* Timefield does not contain a valid USA time
C
C     *USA          TEST (T)                Timefield              22
```

Figure 178. TEST (D/T/Z) Example

TESTB (Test Bit)

Code	Factor 1	Factor 2	Result Field	Indicators		
TESTB		Bit numbers	Character field	OF	ON	EQ

The TESTB operation compares the bits identified in factor 2 with the corresponding bits in the field named as the result field. The result field must be a one-position character field. Resulting indicators in positions 71 through 76 reflect the status of the result field bits. Factor 2 is always a source of bits for the result field.

Factor 2 can contain:

- *Bit numbers 0-7:* From 1 to 8 bits can be tested per operation. The bits to be tested are identified by the numbers 0 through 7. (0 is the leftmost bit.) The bit numbers must be enclosed in apostrophes, and the entry must begin in position 33. For example, to test bits 0, 2, and 5, enter '025' in factor 2.
- *Field name:* You can specify the name of a one-position character field, table name, or array element in factor 2. The bits that are on in the field, table name, or array element are compared with the corresponding bits in the result field; bits that are off are not considered. The field specified in the result field can be an array element if each element of the array is a one-position character field.
- *Hexadecimal literal or named constant:* You can specify a 1-byte hexadecimal literal or hexadecimal named constant. Bits that are on in factor 2 are compared with the corresponding bits in the result field; bits that are off are not considered.

Figure 179 on page 475 illustrates uses of the TESTB operation.

Indicators assigned in positions 71 through 76 reflect the status of the result field bits. At least one indicator must be assigned, and as many as three can be assigned for one operation. For TESTB operations, the resulting indicators are set on as follows:

- *Positions 71 and 72:* An indicator in these positions is set on if the bit numbers specified in factor 2 or each bit that is on in the factor 2 field is off in the result field. That is, all of the specified bits are equal to off.
- *Positions 73 and 74:* An indicator in these positions is set on if the bit numbers specified in factor 2 or the bits that are on in the factor 2 field are of mixed status (some on, some off) in the result field. That is, at least one the specified bits is on.

  Note: If only one bit is to be tested, these positions must be blank. If a field name is specified in factor 2 and it has only one bit on, an indicator in positions 73 and 74 is not set on.
- *Positions 75 and 76:* An indicator in these positions is set on if the bit numbers specified in the factor 2 or each bit that is on in factor 2 field is on in the result field. That is, all of the specified bits are equal to on.

  Note: If the field in factor 2 has no bits on, then no indicators are set on.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The field bit settings are FieldF = 00000001, and FieldG = 11110001.
C*
C*  Indicator 16 is set on because bit 3 is off (0) in FieldF.
C*  Indicator 17 is set off.
C                   TESTB     '3'           FieldF               16  17
C*
C*  Indicator 16 is set on because both bits 3 and 6 are off (0) in
C*  FieldF.  Indicators 17 and 18 are set off.
C                   TESTB     '36'          FieldF               161718
C*
C*  Indicator 17 is set on because bit 3 is off (0) and bit 7 is on
C*  (1) in FLDF.  Indicators 16 and 18 are set off.
C                   TESTB     '37'          FieldF               161718
C*
C*  Indicator 17 is set on because bit 7 is on (1) in FLDF.
C*  Indicator 16 is set off.
C                   TESTB     '7'            FieldF              16  17
C*
C*  Indicator 17 is set on because bits 0,1,2, and 3 are off (0) and
C*  bit 7 is on (1).  Indicators 16 and 18 are set off.
C                   TESTB     FieldG         FieldF              161718
C*
C*  The hexadecimal literal X'88' (10001000) is used in factor 2.
C*  Indicator 17 is set on because at least one bit (bit 0) is on
C*  Indicators 16 and 18 are set off.
C                   TESTB     X'88'          FieldG              161718
```

Figure 179. TESTB Operation

TESTN (Test Numeric)

Code	Factor 1	Factor 2	Result Field	Indicators		
TESTN			Character field	NU	BN	BL

The TESTN operation tests a character result field for the presence of zoned decimal digits and blanks. The result field must be a character field. To be considered numeric, each character in the field, except the low-order character, must contain a hexadecimal F zone and a digit (0 through 9). The low-order character is numeric if it contains a hexadecimal C, hexadecimal D, or hexadecimal F zone, and a digit (0 through 9). Note that the alphabetic characters J through R, should they appear in the low-order position of a field, are treated as negative numbers by TESTN. As a result of the test, resulting indicators are set on as follows:

- *Positions 71 and 72:* Either the result field contains numeric characters, or it contains a 1-character field that consists of a letter from A to R.
- *Positions 73 and 74:* The result field contains both numeric characters and at least one leading blank. For example, the values ƀ123 or ƀƀ123 set this indicator on. However, the value ƀ1ƀ23 is not a valid numeric field because of the embedded blanks, so this value does not set this indicator on.

  Note: An indicator cannot be specified in positions 73 and 74 when a field of length one is tested because the character field must contain at least one numeric character and one leading blank.
- *Positions 75 and 76:* The result field contains all blanks.

The same indicator can be used for more than one condition. If any of the conditions exist, the indicator is set on.

The TESTN operation may be used to validate fields before they are used in operations where their use may cause undesirable results or exceptions (e.g. arithmetic operations).

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The field values are FieldA = 123, FieldB = 1X4, FieldC = 004,
C*  FieldD = ƀƀƀ, FieldE = ƀ1ƀ3, and FieldF = ƀ12.
C*
C*  Indicator 21 is set on because FieldA contains all numeric
C*  characters.
C                   TESTN                   FieldA               21
C*  Indicator 22 is set on because FieldA contains all numeric
C*  characters.  Indicators 23 and 24 remain off.
C                   TESTN                   FieldA               222324
C*  All indicators are off because FieldB does not contain valid
C*  numeric data.
C                   TESTN                   FieldB               252627
C*  Indicator 28 is set on because FieldC contains valid numeric data.
C*  Indicators 29 and 30 remain off.
C                   TESTN                   FieldC               282930
C*  Indicator 33 is set on because FieldD contains all blanks.
C*  Indicators 31 and 32 remain off.
C                   TESTN                   FieldD               313233
C*  Indicators 34, 35, and 36 remain off.  Indicator 35 remains off
C*  off because FieldE contains a blank after a digit.
C                   TESTN                   FieldE               343536
C*  Indicator 38 is set on because FieldF contains leading blanks and
C*  valid numeric characters.  Indicators 37 and 39 remain off.
C                   TESTN                   FieldF               373839
```

Figure 180. TESTN Operation

TESTZ (Test Zone)

Code	Factor 1	Factor 2	Result Field	Indicators		
TESTZ			Character field			

The TESTZ operation tests the zone of the leftmost character in the result field. The result field must be a character field. Resulting indicators are set on according to the results of the test. You must specify at least one resulting indicator positions 71 through 76. The characters &, A through I, and any character with the same zone as the character A set on the indicator in positions 71 and 72. The characters - (minus), J through R, and any character with the same zone as the character J set on the indicator in positions 73 and 74. Characters with any other zone set on the indicator in positions 75 and 76.

TIME (Time of Day)

Code	Factor 1	Factor 2	Result Field	Indicators		
TIME			Numeric field			

The TIME operation accesses the system time of day and, if specified, the system date at any time during program processing. The system time is based on the 24-hour clock.

The result field must specify the name of a 6-, 12- or 14-digit numeric field (no decimal positions) into which the time of day or the time of day and the system date are written.

To access the time of day only, specify the result field as a 6-digit numeric field. To access both the time of day and the system date, specify the result field as a 12- (2-digit year portion) or 14-digit (4-digit year portion) numeric field. The time of day is always placed in the first six positions of the result field in the following format:

hhmmss (hh=hours, mm=minutes, and ss=seconds)

If the system date is included, it is placed in positions 7 through 12 or 7 through 14 of the result field. The date format depends on the date format job attribute QDATFMT and can be mmddyy, ddmmyy, yymmdd, or Julian. The Julian format for 2-digit year portion contains the year in positions 7 and 8, the day (1 through 366, right-adjusted, with zeros in the unused high-order positions) in positions 9 through 11, and 0 in position 12. For 4-digit year portion, it contains the year in positions 7 through 10, the day (1 through 366, right-adjusted, with zeros in the unused high-order positions) in positions 11 through 13, and 0 in position 14.

The special fields UDATE and *DATE contain the job date. These values are not updated when midnight is passed, or when the job date is changed during the running of the program.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  When the TIME operation is processed (with a 6-digit numeric
C*  field), the current time (in the form hhmmss) is placed in the
C*  result field CLOCK.  The TIME operation is based on the 24-hour
C*  clock, for example, 132710.  (In the 12-hour time system, 132710
C*  is 1:27:10 p.m.)  CLOCK can then be specified in the output
C*  specifications.
C                   TIME                    Clock             6 0
C*  When the TIME operation is processed (with a 12-digit numeric
C*  field), the current time and day is placed in the result field
C*  TIMSTP.  The first 6 digits are the time, and the last 6 digits
C*  are the date; for example, 093315121579 is 9:33:15 a.m. on
C*  December 15, 1979.  TIMSTP can then be specified in the output
C*  specifications.
C                   TIME                    TimStp           12 0
C                   MOVEL     TimStp        Time              6 0
C                   MOVE      TimStp        SysDat            6 0
C*  This example duplicates the 12-digit example above but uses a
C*  14-digit field.  The first 6 digits are the time, and the last
C*  8 digits are the date;   for example, 13120001101992
C*  is 1:12:00 p.m. on January 10, 1992.
C*  TIMSTP can then be specified in the output specifications.
C                   TIME                    TimStp           14 0
C                   MOVEL     TimStp        Time              6 0
C                   MOVE      TimStp        SysDat            8 0
```

Figure 181. TIME Operation

UNLOCK (Unlock a Data Area or Release a Record)

Code	Factor 1	Factor 2	Result Field	Indicators		
UNLOCK		Data area or file name		_	ER	_

The UNLOCK operation is used to unlock data areas and release record locks.

Unlocking data areas

Factor 2 must contain the name of the data area to be unlocked, or the reserved word *DTAARA.

When *DTAARA is specified in factor 2, all data areas in the program that are locked are unlocked.

The data area must already be specified in the result field of an *DTAARA DEFINE statement or with the DTAARA keyword on the definition specification. Factor 2 must not refer to the local data area or the Program Initialization Parameters (PIP) data area. If the UNLOCK operation is specified to an already unlocked data area, an error does not occur.

You can specify a resulting indicator in positions 73 and 74 to be set if an error occurs on the operation. Positions 71,72,75 and 76 must be blank.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  TOTAMT, TOTGRS, and TOTNET are defined as data areas in the
C*  system.  The IN operation retrieves all the data areas defined in
C*  the program.  The program processes calculations, and
C*  then unlocks the data areas. The data areas can them be used
C*  by other programs.
C*
C     *LOCK         IN        *DTAARA
C                   :
C                   :
C                   UNLOCK    *DTAARA
C     *DTAARA       DEFINE                  TOTAMT            8 2
C     *DTAARA       DEFINE                  TOTGRS           10 2
C     *DTAARA       DEFINE                  TOTNET           10 2
```

Figure 182. Data area unlock operation

Releasing record locks

The UNLOCK operation also allows the most recently locked record to be unlocked for an update disk file.

Factor 2 must contain the name of the UPDATE disk file.

You can specify a resulting indicator in positions 73 and 74 to be set if an error occurs on the operation. Positions 71,72,75 and 76 must be blank.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
F*
FUPDATA    UF   E             DISK
F*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  Assume that the file UPDATA contains record format vendor.
C*  A record is read from UPDATA. Since the file is an update
C*  file, the record is locked. *IN50 is set somewhere else in
C*  the program to control whether an UPDATE should take place.
C*  otherwise the record is unlocked using the UNLOCK operation.
C*  Note that factor 2 of the UNLOCK operation is the file name,
C*  UPDATA, not the record format, VENDOR
C*
C                   READ      VENDOR                                 12
C                   :
C     *IN50         IFEQ      *ON
C                   UPDATE    VENDOR
C                   ELSE
C                   UNLOCK    UPDATA                               99
C                   ENDIF
```

Figure 183. Record unlock operation

UPDATE (Modify Existing Record)

Code	Factor 1	Factor 2	Result Field	Indicators		
UPDATE		File name	Data structure	_	ER	_

The UPDATE operation modifies the last locked record retrieved for processing from an update disk file or subfile. No other operation should be performed on the file between the input operation that retrieved the record and the UPDATE operation.

Factor 2 must contain the name of a file or record format to be updated. A record format name in factor 2 is required with an externally described file. The record format name must be the name of the last record read from the file; otherwise, an error occurs. A file name in factor 2 is required with a program described file.

The result field must contain a data structure name if factor 2 contains a file name. The updated record is written directly from the data structure to the file. The result field must be blank if factor 2 contains a record format name.

You can specify a resulting indicator in positions 73 and 74 to be set on if the UPDATE operation is not completed successfully.

Remember the following when using the UPDATE operation:

- When a record format name is specified in factor 2, the current values in the program for the fields in the record definition are used to modify the record.
- If some but not all fields in a record are to be updated, use the output specifications and not the UPDATE operation.
- Before UPDATE is issued to a file or record, a valid input operation with lock (READ, READC, READE, READP, READPE, CHAIN, or primary/secondary file) must be issued to the same file or record. If the read operation returns with an error condition or if it was read without locking, the record is not locked and UPDATE cannot be issued. The record must be read again with the default of a blank operation extender to specify a lock request.
- Consecutive UPDATE operations to the same file or record are not valid. Intervening successful read operations must be issued to position to and lock the record to be updated.
- Beware of using the UPDATE operation on primary or secondary files during total calculations. At this stage in the RPG IV cycle, the fields from the current record (the record that is about to be processed) have not yet been moved to the processing area. Therefore, the UPDATE operation updates the current record with the fields from the preceding record. Also, when the fields from the current record are moved to the processing area, they are the fields that were updated from the preceding record.
- For multiple device files, specify a subfile record format in factor 2. The operation is processed for the program device identified in the fieldname specified using the DEVID keyword in the file specification. If the program device is not specified, the device used in the last successful input operation is used. This device must be the same one you specified for the input operation that must precede the UPDATE operation. You must not process input or output operations to other devices in between the input and UPDATE operations. If you do, your UPDATE operation will fail.

- For a display file which has multiple subfile record formats, you must not process read-for-update operations to one subfile record in between the input and UPDATE operations to another subfile in the same display file. If you do, the UPDATE operation will fail.

WHEN (When True Then Select)

Code	Factor 1	Factor 2		
WHEN	Blank.	Expression		

The WHEN operation code is similar to the WHENxx operation code in that it controls the processing of lines in a SELECT operation. It differs in that the condition is specified by a logical expression in the extended-Factor 2 entry. The operations controlled by the WHEN operation are performed when the expression in the extended factor 2 field is true.

```
CL0N01Factor1+++++++OpcodeE+Extended-factor2++++++++++++++++++++++++++++++++
C*
C                   SELECT
C                   WHEN      *INKA
C                   :
C                   :
C                   :
C                   WHEN      NOT(*IN01) AND (DAY = 'FRIDAY')
C                   :
C                   :
C                   :
C                   WHEN      %SUBST(A:(X+4):3) = 'ABC'
C                   :
C                   :
C                   :
C                   OTHER
C                   :
C                   :
C                   :
C                   ENDSL
```

Figure 184. WHEN Operation

WHENxx (When True Then Select)

Code	Factor 1	Factor 2	Result Field	Indicators		
WHENxx	Comparand	Comparand				

The WHENxx operations of a select group determine where control passes after the “SELECT (Begin a Select Group)” operation is processed.

The WHENxx conditional operation is true if factor 1 and factor 2 have the relationship specified by xx If the condition is true, the operations following the WHENxx are processed until the next WHENxx, OTHER, ENDSL, or END operation.

When performing the WHENxx operation remember:

- After the operation group is processed, control passes to the statement following the ENDSL operation.
- You can code complex WHENxx conditions using ANDxx and ORxx. Calculations are processed when the condition specified by the combined WHENxx, ANDxx, and ORxx operations is true.
- The WHENxx group can be empty.
- Within total calculations, the control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry is for documentation purposes only. Conditioning indicator entries (positions 9 through 11) are not allowed.

Refer to “Compare Operations” on page 291 for valid values for xx.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The following example shows nested SELECT groups.  The employee
C* type can be one of 'C' for casual, 'T' for retired, 'R' for
C* regular, and 'S' for student.  Depending on the employee type
C* (EmpTyp), the number of days off per year (Days) will vary.
C*
C                   SELECT
C     EmpTyp        WHENEQ    'C'
C     EmpTyp        OREQ      'T'
C                   Z-ADD     0             Days
C     EmpTyp        WHENEQ    'R'
C*
C* When the employee type is 'R', the days off depend also on the
C* number of years of employment.  The base number of days is 14.
C* For less than 2 years, no extra days are added.  Between 2 and
C* 5 years, 5 extra days are added.  Between 6 and 10 years, 10
C* extra days are added, and over 10 years, 20 extra days are added.
C*
C                   Z-ADD     14            Days
C*
C* Nested select group.
C                   SELECT
C     Years         WHENLT    2
C     Years         WHENLE    5
C                   ADD       5             Days
C     Years         WHENLE    10
C                   ADD       10            Days
C                   OTHER
C                   ADD       20            Days
C                   ENDSL
C* End of nested select group.
C*
C     EmpTyp        WHENEQ    'S'
C                   Z-ADD     5             Days
C                   ENDSL
```

Figure 185 (Part 1 of 2). WHENxx Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C* Example of a SELECT group with complex WHENxx expressions.  Assume
C* that a record and an action code have been entered by a user.
C* Select one of the following:
C*   * When F3 has been pressed, do subroutine QUIT.
C*   * When action code(Acode) A (add) was entered and the record
C*     does not exist (*IN50=1), write the record.
C*   * When action code A is entered, the record exists, and the
C*     active record code for the record is D (deleted); update
C*     the record with active rec code=A.  When action code D is
C*     entered, the record exists, and the action code in the
C*     record (AcRec) code is A; mark the record as deleted.
C*   * When action code is C (change), the record exists, and the
C*     action code in the record (AcRec) code is A; update the record.
C*   * Otherwise, do error processing.
C*
C     RSCDE         CHAIN     FILE                                   50
C                   SELECT
C     *INKC         WHENEQ    *ON
C                   EXSR      QUIT
C     Acode         WHENEQ    'A'
C     *IN50         ANDEQ     *ON
C                   WRITE     REC
C     Acode         WHENEQ    'A'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'D'
C     Acode         OREQ      'D'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'A'
C                   MOVE      Acode         AcRec
C                   UPDATE    REC
C     Acode         WHENEQ    'C'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'A'
C                   UPDATE    REC
C                   OTHER
C                   EXSR      ERROR
C                   ENDSL
```

Figure 185 (Part 2 of 2). WHENxx Operation

WRITE (Create New Records)

Code	Factor 1	Factor 2	Result Field	Indicators		
WRITE		File name	Data structure	_	ER	_

The WRITE operation writes a new record to a file.

Factor 2 must contain the name of a file. A record format name is required in factor 2 with an externally described file. A file name in factor 2 is required with a program described file, and the result field must contain the name of a data structure. The record is written directly from the data structure to the file. The result field must be blank if factor 2 contains a record format name.

The result field must be a data structure name.

Positions 73 and 74 can contain an indicator to be set on if the WRITE operation is not completed successfully. The indicator in positions 73 and 74 will be set on if overflow is reached to an externally described print file and no overflow indicator has been specified on the File description specification. On a WRITE to a subfile (SFILE) record name, you can specify an indicator in positions 75 and 76; it is set on when the subfile is filled.

When using the WRITE operation remember:

- When factor 2 contains a record format name, the current values in the program for all the fields in the record definition are used to construct the record.
- When records that use relative record numbers are written to a file, you must update the field name specified on the RECNO File specification keyword (relative record number), so it contains the relative record number of the record to be written.
- When you use the WRITE operation to add records to a DISK file, you must specify an A in position 20 of the file description specifications. (See “Position 20 (File Addition)” on page 182.)
- Device dependent functions are limited. For example, if a "WRITE" is issued to a "PRINTER" device, the space after will be set to 1 if the keyword PRTCTL is not specified on the file specification (normally spacing or skipping information are specified in columns 41 through 51 of the output specifications). If the file is externally described, these functions are part of the external description.
- For a multiple device file, data is written to the program device named in the field name specified with the “DEVID(fieldname)” on page 190 keyword on the file description specifications. If the DEVID keyword is not specified, data is written to the program device for which the last successful input operation was processed.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C*  The WRITE operation writes the fields in the data structure
C*  DS1 to the file, FILE1.
C*
C                   WRITE     FILE1         DS1
```

Figure 186. WRITE Operation

XFOOT (Summing the Elements of an Array)

Code	Factor 1	Factor 2	Result Field	Indicators		
XFOOT (H)		Array name	Sum	+	–	Z

XFOOT adds the elements of an array together and places the sum into the field specified as the result field. Factor 2 contains the name of the array.

If half-adjust is specified, the rounding occurs after all elements are summed and before the results are moved into the result field. If the result field is an element of the array specified in factor 2, the value of the element before the XFOOT operation is used to calculate the total of the array.

For further rules for the XFOOT operation, see “Arithmetic Operations” on page 287.

See Figure 100 on page 289 for an example of the XFOOT operation.

XLATE (Translate)

Code	Factor 1	Factor 2	Result Field	Indicators		
XLATE (P)	From:To	String:start	Target String	_	ER	_

Characters in the source string (factor 2) are translated according to the From and To strings (both in factor 1) and put into a receiver field (result field). Source characters with a match in the From string are translated to corresponding characters in the To string. The From and the To strings, the source string and the target string must all be of the same type, either all character or all graphic. XLATE starts translating the source at the location specified in factor 2 and continues character by character, from left to right. If a character of the source string exists in the From string, the corresponding character in the To string is placed in the result field. Any characters in the source field before the starting position are placed unchanged in the result field.

Factor 1 must contain the From string, followed by a colon, followed by the To string. The From and To strings can contain one of the following: a field name, array element, named constant, data structure name, literal, or table name.

Factor 2 must contain either the source string or the source string followed by a colon and the start location. The source string portion of factor 2 must be character or graphic, and can contain one of the following: a field name, array element, named constant, data structure name, data structure subfield, literal, or table name. If the operation uses graphic data, the start position refers to 2-byte characters. The start location portion of factor 2 must be numeric with no decimal positions and can be a named constant, array element, field name, literal, or table name. If no start location is specified, a value of 1 is used.

The result field can be a character or graphic field, character or graphic array element, data structure or a character or graphic table. The length of the result field should be as large as the source string specified in factor 2. If the result field is larger than the source string, the result will be left adjusted. If the result field is shorter than the source string, the result field will contain the leftmost part of the translated source.

If a character in the From string is duplicated, the first occurrence (leftmost) is used.

Note: Figurative constants cannot be used in factor 1, factor 2, or result fields. No overlapping in a data structure is allowed for factor 1 and the result field, or factor 2 and the result field.

Any valid indicator can be specified in columns 7 to 11.

If factor 2 is shorter than the result field, a P specified in the operation extender position indicates that the result field should be padded on the right with blanks after the translation. If the result field is graphic and P is specified, then graphic blanks will be used.

Columns 71 and 72 must be blank. An indicator in positions 73-74 turns on if an error occurs on the operation. Columns 75-76 must be blank.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C*
C* The following translates the blank in NUMBER to '-'.  The result
C* in RESULT will be '999-9999'.
C*
C                   MOVE      '999 9999'    Number            8
C     ' ':'-'       XLATE     Number        Result            8
```

Figure 187. XLATE Operation

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D*
D* In the following example, all values in STRING are translated to
D* uppercase.  As a result, RESULT='RPG DEP'.
D*
D Up              C                     'ABCDEFGHIJKLMNOPQRS-
D                                       'TUVWXYZ'
D Lo              C                     'abcdefghijklmnopqrs-
D                                       'tuvwxyz'
C*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C
C                   MOVE      'rpg dep'     String            7
C     Lo:Up         XLATE     String        Result              90
C*
C* In the following example all values in the string are translated
C* to lowercase.  As a result, RESULT='rpg dep'.
C*
C                   MOVE      'RPG DEP'     String            7
C     Up:Lo         XLATE     String        Result              90
```

Figure 188. XLATE Operation With Named Constants

Z-ADD (Zero and Add)

Code	Factor 1	Factor 2	Result Field	Indicators		
Z-ADD (H)		Addend	Sum	+	–	Z

Factor 2 is added to a field of zeros. The sum is placed in the result field. Factor 1 is not used. Factor 2 must be numeric and can contain one of: an array, array element, field, figurative constant, literal, named constant, subfield, or table name.

The result field must be numeric, and can contain one of: an array, array element, subfield, or table name.

Half-adjust can be specified.

For the rules for the Z-ADD operation, see “Arithmetic Operations” on page 287.

See Figure 100 on page 289 for an example of the Z-ADD operation.

Z-SUB (Zero and Subtract)

Code	Factor 1	Factor 2	Result Field	Indicators		
Z-SUB (H)		Subtrahend	Difference	+	–	Z

Factor 2 is subtracted from a field of zeros. The difference, which is the negative of factor 2, is placed in the result field. You can use the operation to change the sign of a field. Factor 1 is not used. Factor 2 must be numeric and can contain one of the following: an array, array element, field, figurative constant, literal, named constant, subfield, or table name.

The result field must be numeric, and can contain one of the following: an array, array element, subfield, or table name.

Half-adjust can be specified.

For the rules for the Z-SUB operation, see "Arithmetic Operations" on page 287.

See Figure 100 on page 289 for an example of the Z-SUB operation.

Appendixes

Appendix A. RPG IV Restrictions

Function	Restriction
Array/table input record length for compile time	Maximum length is 100.
Character or graphic field length	Maximum length is 32767 bytes
Control fields (position 63 and 64 of input specifications) length	Maximum length is 256.
Name data structure length	Maximum of 32767.
Unnamed data structure length	Maximum of 9999999.
Data structure occurrences (number of)	Maximum of 32767 per data structure.
Edit Word	Maximum length of 24 for literals or 115 for named constants.
Elements in an array/table (DIM keyword on the definition specifications)	Maximum of 32767 per array/table.
Levels of nesting in structured groups	Maximum of 100.
Look-ahead	Can be specified only once for a file. Can be specified only for primary and secondary files.
Named Constant	Maximum length of 1024 bytes for a character, hexadecimal, or graphic named constant and 30 digits with 30 decimal positions for a numeric named constant.
Overflow indicator	Only 1 unique overflow indicator can be specified per printer file.
Parameters to programs	Maximum of 255
Parameters to procedures	Maximum of 399
Primary file (P in position 18 of file description specifications)	Maximum of 1 per program.
Printer file (PRINTER in positions 36 through 42 of file description specifications)	Maximum of 8 per program.
Printing lines per page	Minimum of 2; maximum of 255.
Program status data structure	Only 1 allowed per program.
Record address file (R in position 18 of file description specifications)	Only 1 allowed per program.
Record length for program described file (positions 23 through 27 of file description specifications)	Maximum length is 99999. [1]
Structured groups (see levels of nesting)	
Tables (see arrays)	
[1]Any device record size restraints override this value.	

Appendix B. EBCDIC Collating Sequence

EBCDIC Collating Sequence

Table 30 (Page 1 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Represen-tation	Hex Represen-tation
65	ƀ	Space	64	40
. . .				
75	¢	Cent sign	74	4A
76	.	Period, decimal point	75	4B
77	<	Less than sign	76	4C
78	(	Left parenthesis	77	4D
79	+	Plus sign	78	4E
80	|	Vertical bar, Logical OR	79	4F
81	&	Ampersand	80	50
. . .				
91	!	Exclamation point	90	5A
92	$	Dollar sign	91	5B
93	*	Asterisk	92	5C
94	)	Right parenthesis	93	5D
95	;	Semicolon	94	5E
96	¬	Logical NOT	95	5F
97	-	Minus, hyphen	96	60
98	/	Slash	97	61
. . .				
107	¦	Split vertical bar	106	6A
108	,	Comma	107	6B
109	%	Percent sign	108	6C
110	_	Underscore	109	6D
111	>	Greater than sign	110	6E
112	?	Question mark	111	6F
. . .				

Table 30 (Page 2 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Represen-tation	Hex Represen-tation
122	`	Accent grave	121	79
123	:	Colon	122	7A
124	#	Number sign, pound sign	123	7B
125	@	At sign	124	7C
126	'	Apostrophe, prime sign	125	7D
127	=	Equal sign	126	7E
128	"	Quotation marks	127	7F
. . .				
130	a		129	81
131	b		130	82
132	c		131	83
133	d		132	84
134	e		133	85
135	f		134	86
136	g		135	87
137	h		136	88
138	i		137	89
. . .				
146	j		145	91
147	k		146	92
148	l		147	93
149	m		148	94
150	n		149	95
151	o		150	96
152	p		151	97
153	q		152	98
154	r		153	99
. . .				
162	˜	Tilde	161	A1
163	s		162	A2
164	t		163	A3
165	u		164	A4
166	v		165	A5

Table 30 (Page 3 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Represen-tation	Hex Represen-tation
167	w		166	A6
168	x		167	A7
169	y		168	A8
170	z		169	A9
. . .				
193	{	Left brace	192	C0
194	A		193	C1
195	B		194	C2
196	C		195	C3
197	D		196	C4
198	E		197	C5
199	F		198	C6
200	G		199	C7
201	H		200	C8
202	I		201	C9
. . .				
209	}	Right brace	208	D0
210	J		209	D1
211	K		210	D2
212	L		211	D3
213	M		212	D4
214	N		213	D5
215	O		214	D6
216	P		215	D7
217	Q		216	D8
218	R		217	D9
. . .				
225	\	Left slash	224	E0
. . .				
227	S		226	E2
228	T		227	E3

Table 30 (Page 4 of 4). EBCDIC Collating Sequence

Ordinal Number	Symbol	Meaning	Decimal Represen-tation	Hex Represen-tation
229	U		228	E4
230	V		229	E5
231	W		230	E6
232	X		231	E7
233	Y		232	E8
234	Z		233	E9
. . .				
241	0		240	F0
242	1		241	F1
243	2		242	F2
244	3		243	F3
245	4		244	F4
246	5		245	F5
247	6		246	F6
248	7		247	F7
249	8		248	F8
250	9		249	F9

Bibliography

For additional information about topics related to RPG IV programming on the AS/400 system, refer to the following IBM AS/400 publications:

- *CL Programming*, SC41-3721, provides a wide-ranging discussion of AS/400 programming topics including a general discussion on objects and libraries, CL programming, controlling flow and communicating between programs, working with objects in CL programs, and creating CL programs. Other topics include predefined and impromptu messages and message handling, defining and creating user-defined commands and menus, application testing, including debug mode, breakpoints, traces, and display functions.
- *CL Reference*, SC41-3722, provides a description of the AS/400 control language (CL) and its OS/400 commands. (Non-OS/400 commands are described in the respective licensed program publications.) Also provides an overview of *all* the CL commands for the AS/400 system, and it describes the syntax rules needed to code them.
- *Communications Management*, SC41-3406, provides information about work management in a communications environment, communications status, tracing and diagnosing communications problems, error handling and recovery, performance, and specific line speed and subsystem storage information.
- *Data Management*, SC41-3710, provides information about using files in application programs. Includes information on the following topics:
  - Fundamental structure and concepts of data management support on the system
  - Overrides and file redirection (temporarily making changes to files when an application program is run)
  - Copying files by using system commands to copy data from one place to another
  - Tailoring a system using double-byte data
- *DB2/400 Database Programming*, SC41-3701, provides a detailed discussion of the AS/400 database organization, including information on how to create, describe, and update database files on the system. Also describes how to define files to the system using OS/400 data description specifications (DDS) keywords.
- *DDS Reference*, SC41-3712, provides detailed descriptions for coding the data description specifications (DDS) for file that can be described externally. These files are physical, logical, display, print, and intersystem communication function (ICF) files.
- *Distributed Data Management*, SC41-3307, provides information about remote file processing. It describes how to define a remote file to OS/400 distributed data management (DDM), how to create a DDM file, what file utilities are supported through DDM, and the requirements of OS/400 DDM as related to other systems.
- *Experience RPG IV Multimedia Tutorial* is an interactive self-study program explaining the differences between RPG III and RPG IV and how to work within the new ILE environment. An accompanying workbook provides additional exercises and doubles as a reference upon completion of the tutorial. ILE RPG/400 code examples are shipped with the tutorial and run directly on the AS/400. Dial 1-800-IBM-CALL to order the tutorial.
- *GDDM Programming*, SC41-3717, provides information about using OS/400 graphical data display manager (GDDM) to write graphics application programs. Includes many example programs and information to help users understand how the product fits into data processing systems.
- *GDDM Reference*, SC41-3718, provides information about using OS/400 graphical data display manager (GDDM) to write graphics application programs. This manual provides detailed descriptions of all graphics routines available in GDDM. Also provides information about high-level language interfaces to GDDM.
- *ICF Programming*, SC41-3442, provides information needed to write application programs that use AS/400 communications and the OS/400 intersystem communications function (OS/400-ICF). Also contains information on data description specifications (DDS) keywords, system-supplied formats, return codes, file transfer support, and program examples.
- *IDDU Use*, SC41-3704, describes how to use the AS/400 interactive data definition utility (IDDU) to describe data dictionaries, files, and records to the system. Includes:
  - An introduction to computer file and data definition concepts
  - An introduction to the use of IDDU to describe the data used in queries and documents
  - Representative tasks related to creating, maintaining, and using data dictionaries, files, record formats, and fields
  - Advanced information about using IDDU to work with files created on other systems and information about error recovery and problem prevention.

- *ILE Concepts*, SC41-3606, explains concepts and terminology pertaining to the Integrated Language Environment (ILE) architecture of the OS/400 licensed program. Topics covered include creating modules, binding, running programs, debugging programs, and handling exceptions.
- *ILE RPG/400 Programmer's Guide*, SC09-1525,

  provides information about the ILE RPG/400 programming language, which is an implementation of the RPG IV language in the Integrated Language Environment (ILE) on the AS/400 system. It includes information on creating and running programs, with considerations for procedure calls and interlanguage programming. The guide also covers debugging and exception handling and explains how to use AS/400 files and devices in RPG programs. Appendixes include information on migration to RPG IV and sample compiler listings. It is intended for people with a basic understanding of data processing concepts and of the RPG language.
- *ILE RPG Reference Summary*, SX09-1261,

  provides information about the RPG III and RPG IV programming language. This manual contains tables and lists for all specifications and operations in both languages. A key is provided to map RPG III specifications and operations to RPG IV specifications and operations.
- *System API Reference*, SC41-3801, provides information for the experienced programmer on how to use the application programming interfaces (APIs) to such OS/400 functions as:

  - Dynamic Screen Manager
  - Files (database, spooled, hierarchical)
  - Message handling
  - National language support
  - Network management
  - Objects
  - Problem management
  - Registration facility
  - Security
  - Software products
  - Source debug
  - UNIX-type
  - User-defined communications
  - User interface
  - Work management

  Includes original program model (OPM), Integrated Language Environment (ILE), and UNIX-type APIs.
- *System Operation*, SC41-3203, provides information about handling messages, working with jobs and printer output, devices communications, working with support functions, cleaning up your system, and so on.

Index

Communicating Your Comments to IBM

AS/400
ILE RPG/400
Reference
Version 3

Publication No. SC09-1526-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - United States and Canada: 416-448-6161
  - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
  - Internet: torrcf@vnet.ibm.com
  - IBMLink: toribm(torrcf)
  - IBM/PROFS: torolab4(torrcf)
  - IBMMAIL: ibmmail(caibmwt9)

Readers' Comments — We'd Like to Hear from You

AS/400
ILE RPG/400
Reference
Version 3

Publication No. SC09-1526-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SC09-1526-00

IBM®

Cut or F
Along Li

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Fold and Tape **Please do not staple** Fold and Tape

Cut or F
Along Li

IBM®

Program Number: 5763-RG1

Printed in Denmark by
Scanprint a·s, Viby J.

SC09-1526-00